华 清 远 见

FARSIGHT®

始 于 2 0 0 4

FS4412 Arduino 模块

实验手册

华清远见教育集团 ● 研发中心


170-9108-5953


2306275952/2668462267


http://dev.hqyj.com


support@farsight.com.cn

目 录

目 录 I
第 1 章 平台概述 ..... - 1 -
第 2 章 基于 Linux 系统的 Arduino 设备驱动开发实验 ..... - 2 -
2.1 直流电机驱动实验 ..... - 4 -
2.1.1 【实验目的】 ..... - 4 -
2.1.2 【实验环境】 ..... - 4 -
2.1.3 【实验内容】 ..... - 4 -
2.1.4 【实验原理】 ..... - 4 -
2.1.5 【源码分析】 ..... - 5 -
2.1.6 【实验步骤】 ..... - 11 -
2.1.7 【实验结果】 ..... - 17 -
2.2 步进电机驱动实验 ..... - 17 -
2.2.1 【实验目的】 ..... - 17 -
2.2.2 【实验环境】 ..... - 17 -
2.2.3 【实验内容】 ..... - 18 -
2.2.4 【实验原理】 ..... - 18 -
2.2.5 【源码分析】 ..... - 20 -
2.2.6 【实验步骤】 ..... - 24 -
2.2.7 【实验结果】 ..... - 25 -
2.3 舵机驱动实验 ..... - 26 -

华清远见 dev.hqyj.com

2.3.1 【实验目的】 - 26 -
2.3.2 【实验环境】 - 26 -
2.3.3 【实验内容】 - 27 -
2.3.4 【实验原理】 - 27 -
2.3.5 【源码分析】 - 28 -
2.3.6 【实验步骤】 - 32 -
2.3.7 【实验结果】 - 33 -
2.4 继电器驱动实验 - 33 -
2.4.1 【实验目的】 - 33 -
2.4.2 【实验环境】 - 34 -
2.4.3 【实验内容】 - 34 -
2.4.4 【实验原理】 - 34 -
2.4.5 【源码分析】 - 36 -
2.4.6 【实验步骤】 - 38 -
2.4.7 【实验结果】 - 39 -
2.5 蜂鸣器驱动实验 - 41 -
2.5.1 【实验目的】 - 41 -
2.5.2 【实验环境】 - 41 -
2.5.3 【实验内容】 - 41 -
2.5.4 【实验原理】 - 42 -
2.5.5 【源码分析】 - 42 -
2.5.6 【实验步骤】 - 45 -

华清远见 dev.hqv1.com

2.5.7 【实验结果】 - 46 -
2.6 酒精传感器驱动实验 - 47 -
2.6.1 【实验目的】 - 47 -
2.6.2 【实验环境】 - 48 -
2.6.3 【实验内容】 - 48 -
2.6.4 【实验原理】 - 48 -
2.6.5 【源码分析】 - 49 -
2.6.6 【实验步骤】 - 53 -
2.6.7 【实验结果】 - 53 -
2.7 光敏传感器驱动实验 - 55 -
2.7.1 【实验目的】 - 55 -
2.7.2 【实验环境】 - 55 -
2.7.3 【实验内容】 - 55 -
2.7.4 【实验原理】 - 56 -
2.7.5 【源码分析】 - 57 -
2.7.6 【实验步骤】 - 60 -
2.7.7 【实验结果】 - 61 -
2.8 火焰传感器驱动实验 - 62 -
2.8.1 【实验目的】 - 62 -
2.8.2 【实验环境】 - 62 -
2.8.3 【实验内容】 - 63 -
2.8.4 【实验原理】 - 63 -

华清远见 dev.hqyj.com

2.8.5 【源码分析】 - 64 -
2.8.6 【实验步骤】 - 67 -
2.8.7 【实验结果】 - 68 -
2.9 气体传感器驱动实验 - 69 -
2.9.1 【实验目的】 - 69 -
2.9.2 【实验环境】 - 69 -
2.9.3 【实验内容】 - 69 -
2.9.4 【实验原理】 - 70 -
2.9.5 【源码分析】 - 71 -
2.9.6 【实验步骤】 - 75 -
2.9.7 【实验结果】 - 75 -
2.10 按键扫描数码管显示驱动实验 - 77 -
2.10.1 【实验目的】 - 77 -
2.10.2 【实验环境】 - 77 -
2.10.3 【实验内容】 - 77 -
2.10.4 【实验原理】 - 77 -
2.10.5 【源码分析】 - 81 -
2.10.6 【实验步骤】 - 92 -
2.10.7 【实验结果】 - 93 -
2.11 温度传感器驱动实验 - 94 -
2.11.1 【实验目的】 - 94 -
2.11.2 【实验环境】 - 95 -

华清远见 dev.hqyj.com

2.11.3 【实验内容】 - 95 -
2.11.4 【实验原理】 - 95 -
2.11.5 【实验源码】 - 97 -
2.11.6 【实验步骤】 - 99 -
2.11.7 【实验结果】 - 100 -
第 3 章 基于 Android 系统的 Arduino 设备开发例程 - 101 -
3.1 实验应用平台介绍 - 101 -
3.2 程序的介绍 - 101 -
3.2.1 【展示目的】 - 101 -
3.2.2 【源码分析】 - 101 -
3.2.3 【界面展示】 - 110 -
3.3 酒精传感器实验 - 112 -
3.3.1 【实验目的】 - 112 -
3.3.2 【实验环境】 - 112 -
3.3.3 【实验内容】 - 112 -
3.3.4 【实验原理】 - 112 -
3.3.5 【实验步骤】 - 113 -
3.3.6 【实验结果】 - 119 -
3.3.7 【源码分析】 - 119 -
3.4 光电闸实验 - 123 -
3.4.1 【实验目的】 - 123 -
3.4.2 【实验环境】 - 124 -

华清远见 dev.hqv1.com

3.4.3 【实验内容】 - 124 -
3.4.4 【实验原理】 - 124 -
3.4.5 【实验步骤】 - 124 -
3.4.6 【实验结果】 - 130 -
3.4.7 【源码分析】 - 131 -
3.5 蜂鸣器实验 - 134 -
3.5.1 【实验目的】 - 134 -
3.5.2 【实验环境】 - 135 -
3.5.3 【实验内容】 - 135 -
3.5.4 【实验原理】 - 135 -
3.5.5 【实验步骤】 - 135 -
3.5.6 【实验结果】 - 141 -
3.5.7 【源码分析】 - 141 -
3.6 温度传感器实验 - 146 -
3.6.1 【实验目的】 - 146 -
3.6.2 【实验环境】 - 146 -
3.6.3 【实验内容】 - 146 -
3.6.4 【实验原理】 - 146 -
3.6.5 【实验步骤】 - 147 -
3.6.6 【实验结果】 - 153 -
3.6.7 【源码分析】 - 154 -
3.7 直流电机实验 - 158 -

华清远见 dev.hqv1.com

3.7.1 【实验目的】 - 158 -
3.7.2 【实验环境】 - 158 -
3.7.3 【实验内容】 - 158 -
3.7.4 【实验原理】 - 159 -
3.7.5 【实验步骤】 - 159 -
3.7.6 【实验结果】 - 165 -
3.7.7 【源码分析】 - 165 -
3.8 可燃气传感器实验 - 170 -
3.8.1 【实验目的】 - 170 -
3.8.2 【实验环境】 - 170 -
3.8.3 【实验内容】 - 170 -
3.8.4 【实验原理】 - 170 -
3.8.5 【实验步骤】 - 171 -
3.8.6 【实验结果】 - 177 -
3.8.7 【源码分析】 - 178 -
3.9 光照传感器实验 - 182 -
3.9.1 【实验目的】 - 182 -
3.9.2 【实验环境】 - 182 -
3.9.3 【实验内容】 - 182 -
3.9.4 【实验原理】 - 182 -
3.9.5 【实验步骤】 - 183 -
3.9.6 【实验结果】 - 189 -

华清远见 dev.hqyj.com

3.9.7 【源码分析】 - 190 -
3.10 矩阵键盘实验 - 193 -
3.10.1 【实验目的】 - 193 -
3.10.2 【实验环境】 - 194 -
3.10.3 【实验内容】 - 194 -
3.10.4 【实验原理】 - 194 -
3.10.5 【实验步骤】 - 194 -
3.10.6 【实验结果】 - 200 -
3.10.7 【源码分析】 - 201 -
3.11 继电器实验 - 207 -
3.11.1 【实验目的】 - 207 -
3.11.2 【实验环境】 - 207 -
3.11.3 【实验内容】 - 207 -
3.11.4 【实验原理】 - 207 -
3.11.5 【实验步骤】 - 208 -
3.11.6 【实验结果】 - 214 -
3.11.7 【源码分析】 - 215 -
3.12 RFID 实验 - 217 -
3.12.1 【实验目的】 - 217 -
3.12.2 【实验环境】 - 217 -
3.12.3 【实验内容】 - 218 -
3.12.4 【实验原理】 - 218 -

华清远见 dev.hqyj.com

3.12.5 【实验步骤】 - 218 -
3.12.6 【实验结果】 - 224 -
3.12.7 【源码分析】 - 225 -
3.13 舵机实验 - 228 -
3.13.1 【实验目的】 - 228 -
3.13.2 【实验环境】 - 229 -
3.13.3 【实验内容】 - 229 -
3.13.4 【实验原理】 - 229 -
3.13.5 【实验步骤】 - 230 -
3.13.6 【实验结果】 - 235 -
3.13.7 【源码分析】 - 235 -
3.14 步进电机实验 - 240 -
3.14.1 【实验目的】 - 240 -
3.14.2 【实验环境】 - 240 -
3.14.3 【实验内容】 - 240 -
3.14.4 【实验原理】 - 240 -
3.14.5 【实验步骤】 - 241 -
3.14.6 【实验结果】 - 247 -
3.14.7 【源码分析】 - 248 -
3.15 火焰传感器实验 - 252 -
3.15.1 【实验目的】 - 252 -
3.15.2 【实验环境】 - 252 -

华清远见 dev.hqyj.com

3.15.3 【实验内容】 - 252 -
3.15.4 【实验原理】 - 252 -
3.15.5 【实验步骤】 - 253 -
3.15.6 【实验结果】 - 259 -
3.15.7 【源码分析】 - 260 -
3.16 数码管实验 - 263 -
3.16.1 【实验目的】 - 263 -
3.16.2 【实验环境】 - 264 -
3.16.3 【实验内容】 - 264 -
3.16.4 【实验原理】 - 264 -
3.16.5 【实验步骤】 - 265 -
3.16.6 【实验结果】 - 270 -
3.16.7 【源码分析】 - 271 -

华清远见 dev.hqv1.com

第 1 章　平台概述

华清远见 FS4412 综合实训开发平台（下文简称 FS4412_Arduino）是由华清远见研发中心专为培训教学和项目研发定制的高性能，多兼容性的 ARM Cortex-A9 开发平台，该平台结合包括 ARM CortexM4、M3、M0 等核心 MCU，以及 430、51 等常用的单片机核心；除了 MCU 外，该平台还兼容标准 Arduino 接口的所有模块；此外支持 FSEC20 模块（4G），多频段 RFID 模块。FS4412 采用三星 ARM Exynos 4412 四核处理器，运行主频可高达 1.5GHz，其处理速度和节能能力比起双核大幅提高。对比上一代的双核处理器，四核能提供翻倍的处理能力以及减半的功耗，这也为精细的显示效果、1080P 拍摄及播放、以及各方面的超高流畅度运行奠定了基础。FS4412 提供了丰富的板载资源以及扩展接口，搭配华清远见研发中心自主研发的 FS-JTAG ARM 仿真器，无论是用于教学、自学还是项目研发，都是非常完善的开发平台。

华清远见 dev.hqyj.com

第 2 章　基于 Linux 系统的 Arduino 设备驱动开发实验

注意：以下所有的设备驱动实验，都是基于 Linux3.0 内核，u-boot 版本为 2010.03，文件系统通过 nfs 网络方式挂载。镜像存放的位置为：【华清远见-嵌入式 ARM 实验箱资料-I\华清远见-嵌入式 ARM 实验箱资料\烧写镜像\Linux3.0 烧写镜像】

在本实验平台上的所有 Arduino 设备都属于字符设备，所谓字符设备是指那些必须以串行顺序依次进行访问的设备，不能随机读取设备内存中的某一数据，如触摸屏、磁带驱动器、鼠标、LED 等。在 Linux 系统中，字符设备以特别的文件方式在文件目录树中占据位置并拥有相应的结点。结点中的文件类型指明该文件是字符设备文件。可以使用与普通文件相同的文件操作命令对字符设备文件进行操作，例如打开、关闭、读、写等。

在进行实验之前，应该先掌握下面几个知识点，这样更有利于理解后续的实验原理：

设备号

主设备号用来标识与设备文件相连的驱动程序，次设备号被驱动程序用来辨别操作的是哪个设备。简单的说主设备号用来反映设备类型，次设备号用来区分同类型的哪一个设备。内核中通过 dev_t 来描述设备号，其实质是 unsigned int 32 位的整数，其中高 12 位为主设备号，低 20 位为次设备号。那么 Linux 怎么给设备分配设备号呢？可以通过静态申请和动态分配两种方式（针对主设备号）。

静态申请：int　register_chrdev_region(dev_t from ,unsigned count ,const char *name)

动态分配：Int alloc_chrdev_region(dev_t *dev,　unsigned baseminor,　unsigned count,　const char *name)

无论使用哪种方式创建的设备号，在不用的时候都应该将其注销掉，这样方便以后再使用。

创建设备文件

使用 mknod 命令手动创建，具体的命令格式如下：

mknod filename type major minor

filename 为设备文件名，type　设备文件类型，major　主设备号，minor　次设备号

重要的结构体

Struct file 代表一个打开的文件，系统中每一个打开的文件都有一个对应的 struct file，它由内核在文件打开时创建，在文件关闭后释放。一个文件被打开一次对应一个 struct file，这个文件同时被打开 10 次，则对应 10 个 struct file。

Struct inode 用来记录文件在物理上的信息。因此，他和代表打开文件的 file 结构体是不同的。一个文件可以对应多个 file 结构，但只有一个 inode 结构。

Struct file_operation 一个函数指针的集合，定义能在设备上进行的操作。结构体中成员指向驱动中的函数，这些函数实现一个特别的操作，对于不支持的操作保留为 NULL。

设备注册

在 Linux3.0 内核中，字符设备使用 struct cdev 来描述。字符设备的注册可以分为三个步骤：分配 cdev；初始化 cdev；注册（添加）cdev；cdev 的分配可以使用 cdev_alloc 函数来完成，cdev 的初始化可以使用 cdev_init 函数来完成，cdev 的注册使用 cdev_add 函数来完成。

设备操作

对设备的操作主要有打开、关闭、读取、写入等操作。具体调用的 API 如下：

打开设备文件：int (*open)(struct inode*, struct file*)

从设备中读取数据：ssize_t(*read)(struct file*,char_user*,size_t,loff_t*)

向设备写入数据：ssize_t(*write)(struct file*,char_user*,size_t,loff_t*)

控制设备：int (*ioctl)(struct inode*, struct file*,unsigned int,unsigned long)

关闭设备：void (*release)(struct inode*,struct file*)

华清远见 dev.hqyj.com

2.1 直流电机驱动实验

2.1.1 【实验目的】

（1） 掌握直流电机的工作原理；

（2） 掌握 Linux 下直流电机驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.1.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

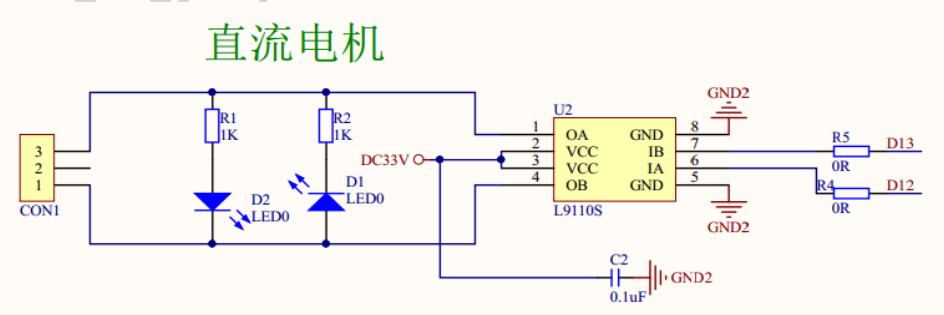
（5） NFS 网络文件系统；

2.1.3 【实验内容】

基于 Linux3.0 内核实现直流电机的控制，控制直流电机的转向等状态。

2.1.4 【实验原理】

该实验需要使用到开发板上的直流电机，相关硬件电路如下：



直流电机驱动电路

从原理图的网络标号查找，或者直接查看【单片机试验箱接口对照表.pdf】可以看到直流电机与 FS4412 相连的 GPIO 分别为：GPC1_1、GPM4_6。

L9110 描述：

L9110 是为控制和驱动电机设计的两通道推挽式功率放大专用集成电路器件，将分立电路集成在单片 IC 之中，使外围器件成本降低，整机可靠性提高。该芯片有两个 TTL/CMOS 兼容电平的输入，具有良好的抗干扰性；两个输出端能直接驱动电机的正反向运动，它具有较大的电流驱动能力，每通道能通过 800mA 的持续电流，峰值电流能力可达 1.5A；(直流小电机一般需要 350mA) 在驱动继电器、直流电机、步进电机或开关功率管的使用上安全可靠。L9110 被广泛应用于玩具汽车电机驱动、脉冲电磁阀门驱动，步进电机驱动和开关功率管等电路上。

2.1.5 【源码分析】

```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <asm/io.h>
#include <asm/uaccess.h>

/* 定义模块版本，及作者 */
MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("Farsight <dev.hqyj.com>");

/* 定义命令类型 */
#define DC_DYNAMO_MAJIC 'D'

/* 定义操作设备的命令，打开，关闭 */
#define DC_DYNAMO_ON _IOW(DC_DYNAMO_MAJIC, 0, int)
#define DC_DYNAMO_OFF _IOW(DC_DYNAMO_MAJIC, 1, int)

/* 定义主次设备号，主设备号 800，次设备号 3 */
#define DC_DYNAMO_MAJOR 800
```

```c
#define DC_DYNAMO_MINOR 3

/*
 *IB<-->D12<-->GPM4_6
 *IA<-->D13<-->GPC1_1
 * */
/* 定义 GPIO 控制寄存器和数据寄存器物理地址 */
/*pin 1 register GPC1_1*/
#define GPC1CON 0x11400080
#define GPC1DAT 0x11400084
/*pin 2 register GPM4_6*/
#define GPM4CON 0x110002E0
#define GPM4DAT 0x110002E4

/* 定义 GPIO 相关寄存器指针，方便后面地址映射，和具体操作 */
static unsigned int *gpc1con;
static unsigned int *gpc1dat;

static unsigned int *gpm4con;
static unsigned int *gpm4dat;

/* 定义字符设备结构体 cdev */
struct cdev cdev;

/* 启动直流电机功能函数实现 */
void fs4412_DC_Dynamo_on(int nr)
{
    switch (nr)
    {
    case 1:
        /* 向 GPIO 数据寄存器写入值，将对应 GPIO 设置为高电平 */
        writel(readl(gpc1dat) | 0x1 << 1, gpc1dat);
        break;
    case 2:
        writel(readl(gpm4dat) | 0x1 << 6, gpm4dat);
        break;
    }
}
/* 关闭直流电机功能函数实现 */
void fs4412_DC_Dynamo_off(int nr)
{
    switch (nr)
    {
```

```
66      case 1:
67          /* 向 GPIO 数据寄存器写入值，将对应 GPIO 设置为低电平 */
68          writel(readl(gpc1dat) & ~(0x1 << 1), gpc1dat);
69          break;
70      case 2:
71          writel(readl(gpm4dat) & ~(0x1 << 6), gpm4dat);
72          break;
73      }
74  }
75
76  /* 打开直流电机这个设备对应的设备文件函数实现 */
77  static int fs4412_DC_Dynamo_open(struct inode *inode, struct file *file)
78  {
79      return 0;
80  }
81
82  /* 在设备使用结束之后为了防止其占用资源，需要将设备释放掉 */
83  static int fs4412_DC_Dynamo_release(struct inode *inode, struct file *file)
84  {
85      fs4412_DC_Dynamo_off(1);
86      fs4412_DC_Dynamo_off(2);
87      return 0;
88  }
89
90  /* 控制直流电机函数实现，当用户层调用 ioctl 函数时，该函数将被调用  */
91  static long fs4412_DC_Dynamo_unlocked_ioctl(struct file *file,
92          unsigned int cmd, unsigned long arg)
93  {
94      int nr;
95      int n;
96
97      if (arg < 1 || arg > 4)
98          return -EINVAL;
99
100     switch(cmd)
101     {
102         /* 如果检测到用户层的命令是打开电机，那调用打开电机的函数  */
103     case DC_DYNAMO_ON:
104         fs4412_DC_Dynamo_on(arg);
105         break;
106         /* 如果检测到用户层的命令是关闭电机，那调用关闭电机的函数  */
107     case DC_DYNAMO_OFF:
108         fs4412_DC_Dynamo_off(arg);
```

```
109         break;
110
111     }
112     return 0;
113
114 }
115
116 /* 直流电机 GPIO 初始化函数，主要配置 GPIO 为输出模式，以及设置引脚的出事电平   */
117 void fs4412_DC_Dynamo_io_init(void)
118 {
119     /*init GPC1_1 GPC1CON1 -> output GPC1DAT1 -> 0*/
120     writel((readl(gpc1con) & ~(0xf << 4) | (0x1 << 4)), gpc1con);
121     writel((readl(gpc1dat) & ~(0x1 << 1)), gpc1dat);
122
123     /*init GPM4_6 GPM4CON6 -> output GPM4DAT6 -> 0*/
124     writel((readl(gpm4con) & ~(0xf << 24) | (0x1 << 24)), gpm4con);
125     writel((readl(gpm4dat) & ~(0x1 << 6)), gpm4dat);
126
127     printk("DC_DYNAMO REGISTER INIT OKAY\n");
128 }
129
130 /* 直流电机 GPIO 相关寄存器地址映射，因为在驱动中是无法直接使用物理地址的   */
131 int fs4412_DC_Dynamo_ioremap(void)
132 {
133     int ret;
134
135     gpc1con = ioremap(GPC1CON, 4);
136     if(gpc1con == NULL)
137     {
138         printk("ioremap gpc1con\n");
139         ret = -ENOMEM;
140         return ret;
141     }
142     gpc1dat = ioremap(GPC1DAT, 4);
143     if(gpc1dat == NULL)
144     {
145         printk("ioremap gpc1dat\n");
146         ret = -ENOMEM;
147         return ret;
148     }
149     gpm4con = ioremap(GPM4CON, 4);
150     if(gpm4con == NULL)
151     {
```

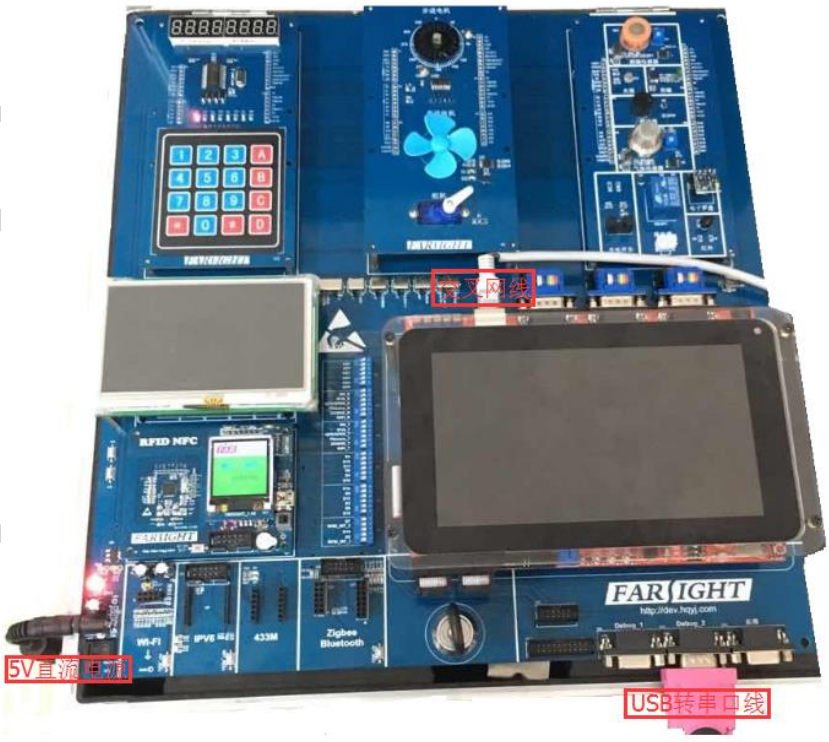
```
152         printk("ioremap gpm4con\n");
153         ret = -ENOMEM;
154         return ret;
155     }
156     gpm4dat = ioremap(GPM4DAT, 4);
157     if(gpm4dat == NULL)
158     {
159         printk("ioremap gpm4dat\n");
160         ret = -ENOMEM;
161         return ret;
162     }
163 
164     return 0;
165 }
166 
167 /* 在驱动使用结束之后，将映射的地址释放掉  */
168 void fs4412_DC_Dynamo_iounmap(void)
169 {
170     iounmap(gpc1con);
171     iounmap(gpc1dat);
172     iounmap(gpm4con);
173     iounmap(gpm4dat);
174 }
175 
176 /* 操作设备文件结构体定义，这里包含三个操作方式，打开、关闭、控制  */
177 struct file_operations fs4412_DC_Dynamo_fops =
178 {
179     .owner = THIS_MODULE,
180     .open  = fs4412_DC_Dynamo_open,
181     .release = fs4412_DC_Dynamo_release,
182     .unlocked_ioctl = fs4412_DC_Dynamo_unlocked_ioctl,
183 
184 };
185 
186 /* 驱动模块初始化函数  */
187 static int fs4412_DC_Dynamo_init(void)
188 {
189     /* 将主设备号和次设备号转换成 dev_t 类型  */
190     dev_t devno = MKDEV(DC_DYNAMO_MAJOR, DC_DYNAMO_MINOR);
191     int ret;
192     /* 为一个字符驱动获取一个或多个设备编号来使用  */
193     ret = register_chrdev_region(devno, 1, "DYNAMO");
194     if(ret < 0)
```

```
195     {
196         printk("register_chrdev_region failed\n");
197         return ret;
198     }
199     /* cdev 结构体初始化  */
200     cdev_init(&cdev, &fs4412_DC_Dynamo_fops);
201     cdev.owner = THIS_MODULE;
202     /* 将 cdev 添加到系统中去  */
203     ret = cdev_add(&cdev, devno, 1);
204     if (ret < 0)
205     {
206         printk("cdev_add\n");
207         goto err1;
208     }
209     /* 调用 GPIO 映射函数  */
210     ret = fs4412_DC_Dynamo_ioremap();
211     if (ret < 0)
212     {
213         goto err2;
214     }
215     /*init con & dat register*/
216     fs4412_DC_Dynamo_io_init();
217     return 0;
218 
219 err2:
220     /* 从系统中移除 cdev 结构体  */
221     cdev_del(&cdev);
222 err1:
223     /* 释放之前注册的设备号  */
224     unregister_chrdev_region(devno, 1);
225     return ret;
226 }
227 
228 /* 卸载驱动模块时调用该函数  */
229 static void fs4412_DC_Dynamo_exit(void)
230 {
231     dev_t devno = MKDEV(DC_DYNAMO_MAJOR, DC_DYNAMO_MINOR);
232 
233     /* 卸载驱动模块时，将之前映射的地址释放掉  */
234     fs4412_DC_Dynamo_iounmap();
235 
236     /* 卸载驱动模块时，删除系统中之前添加的 cdev  */
237     cdev_del(&cdev);
```

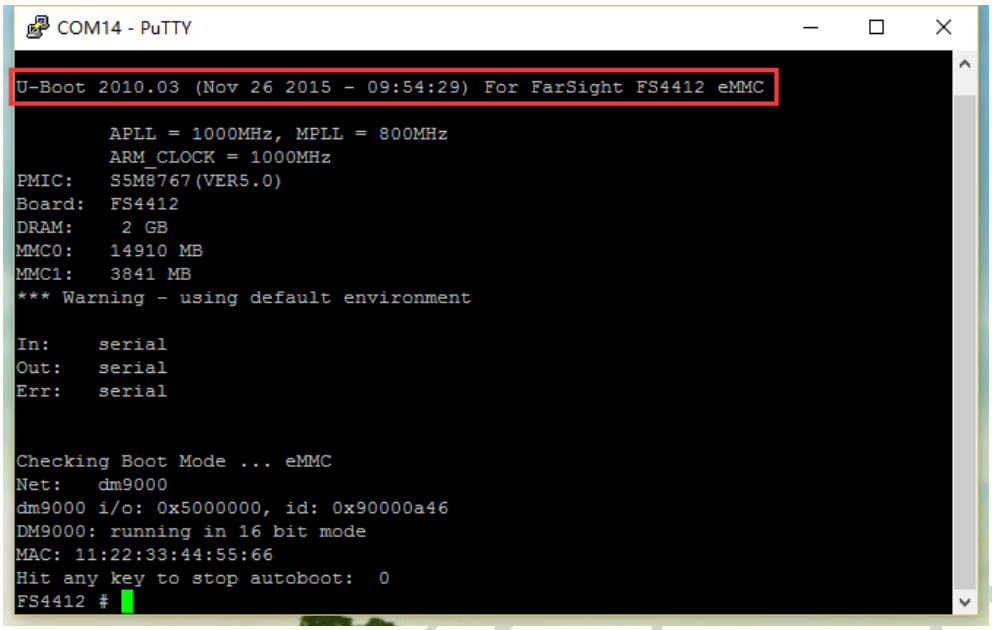
```
238
239     /* 卸载驱动模块时，释放之前注册的设备号  */
240     unregister_chrdev_region(devno, 1);
241     printk("STEPPER DRIVER EXIT\n");
242 }
243
244 /* 驱动程序在 insmod 时，会调用该函数  */
245 module_init(fs4412_DC_Dynamo_init);
246 /* 驱动程序在 rmmod 时，会调用该函数 */
247 module_exit(fs4412_DC_Dynamo_exit);
248
```

2.1.6 【实验步骤】

（1）将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、USB 转串口线、网线。如下图：



（2）打开串口助手 Putty，找到对应的 com 口，设置波特率为 115200，给开发板上电，检查 u-boot 版本是否为 2010.03，如下：



如果 u-boot 的版本不是 2010.03 ,请参考【华清远见嵌入式 ARM 实验箱-I-FS4412\使用手册\FS4412 使用手册.pdf】第五章 烧写镜像，将 EMMC 中的 u-boot 跟换为 u-boot-2010.03 版本。

（3）设置开发板启动环境变量，具体命令如下：

设置开发板 IP 地址

```
setenv ipaddr 192.168.1.192
save
```

设置虚拟机 IP 地址

```
setenv serverip 192.168.1.133
save
```

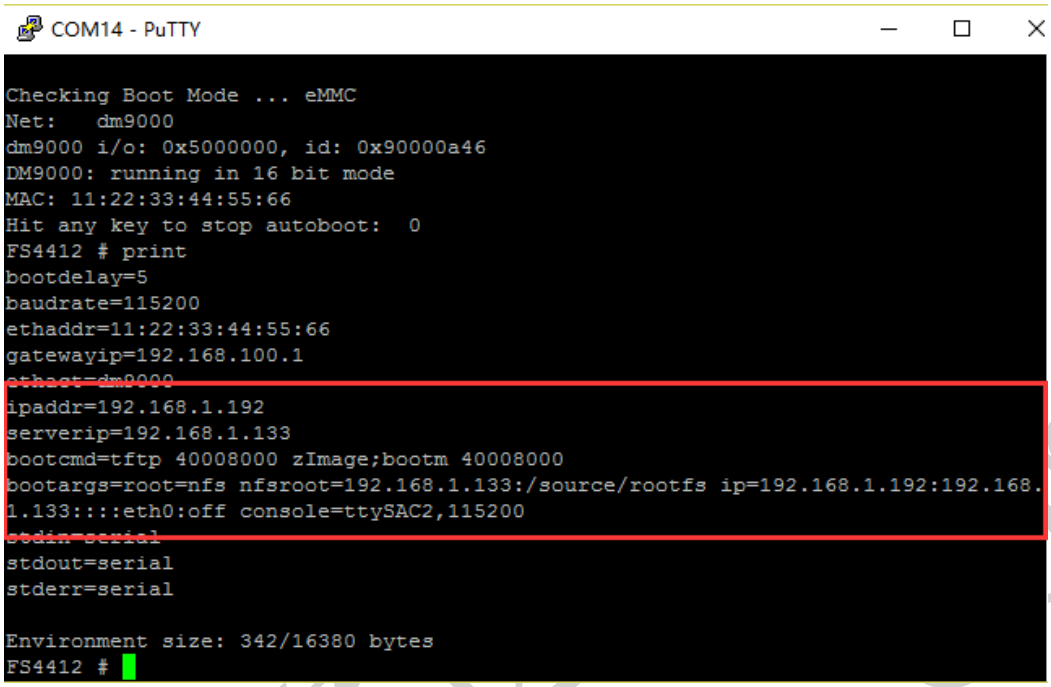
设置开发板启动方式为 tftp 下载内核镜像，下载完成后启动

```
setenv bootcmd tftp 40008000 zImage\; bootm 40008000
save
```

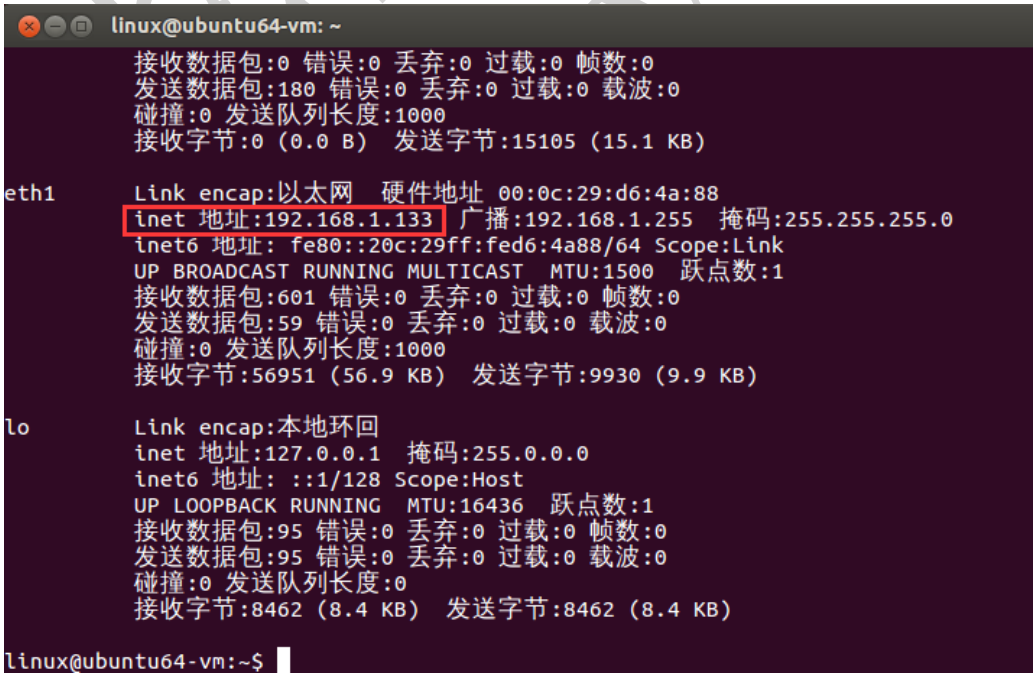
设置挂载网络文件系统，网络文件系统挂载的位置在虚拟机的/source/rootfs 下

```
setenv bootargs root=nfs nfsroot=192.168.1.133:/source/rootfs ip=192.168.1.192:192.168.1.133::::eth0:off console=ttySAC2,115200
save
```

设置完成后，使用 print 命令查看刚才设置的环境变量是否正确：



（4）确保虚拟机有线网卡的 IP 地址为 192.168.1.133（只要在同一网段，可自行修改），在虚拟机的命令行中输入 ifconfig 命令即可查看。



如果 ip 地址不对，可以使用下面的命令动态修改：

```
sudo ifconfig eth1 192.168.1.133
```

（5） 将【华清远见嵌入式 ARM 实验箱-I-FS4412\烧写镜像\Linux3.0 烧写镜像】下的内核镜像 zImage 拷贝到虚拟机的/tftpboot 目录下，将文件系统 rootfs 文件夹拷贝到/source 目录下。

```
linux@ubuntu64-vm:/tftpboot$ ls
exynos4412-fs4412.dtb  fs210  fs4412  fs4412_new  test  uImage  zImage
linux@ubuntu64-vm:/tftpboot$ cd /source/
linux@ubuntu64-vm:/source$ ls
rootfs
linux@ubuntu64-vm:/source$
```

（6）在 Putty 中输入 boot 命令，开发板继续启动，首先会通过 TFTP 的方式下载内核，内核存放的位置在主机的/tftpboot 路径下，内核镜像下载完成后，会启动内核，内核启动完成后紧接着会去挂载网络文件系统。最终可以看到整个系统成功启动了，在 Putty 中输入 ls 命令，会有相应的结果输出。

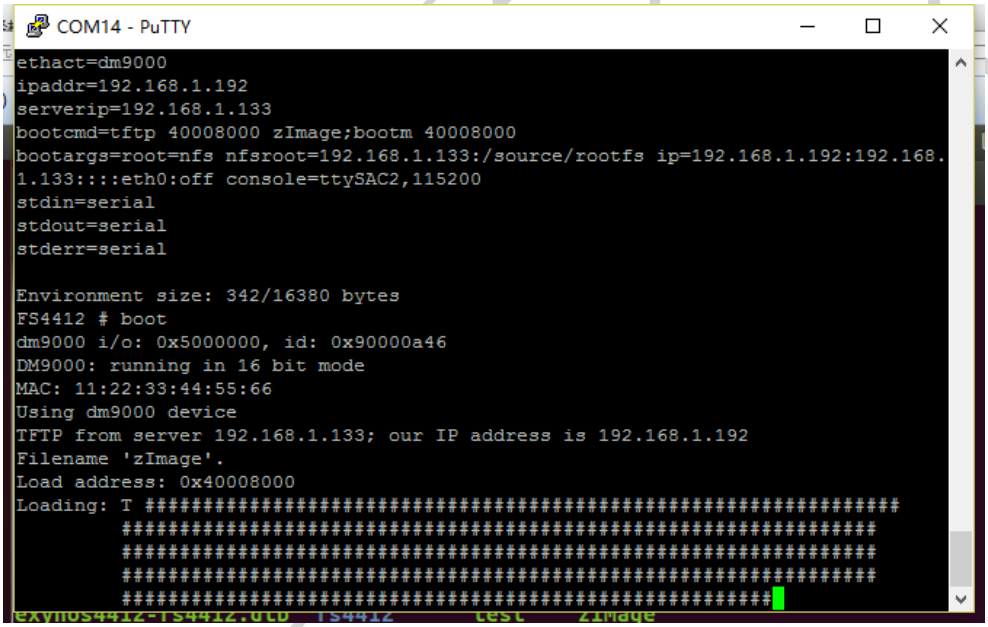
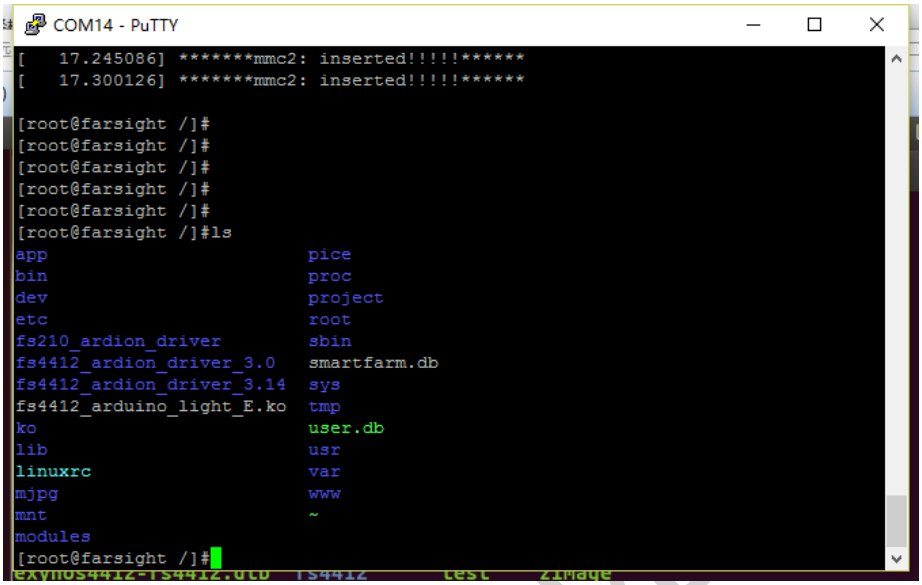


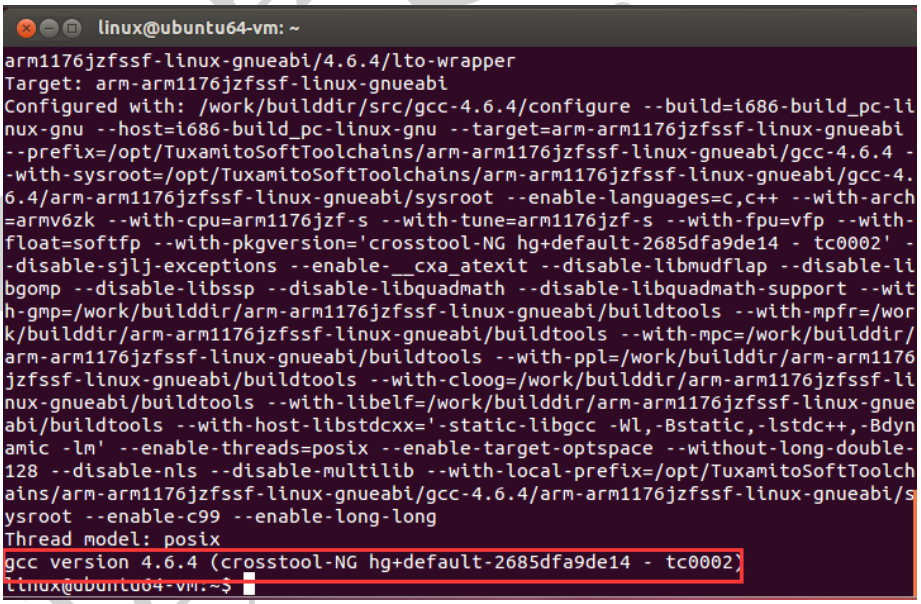
图 内核镜像下载图



这样 fs4412 上 Linux3.0 系统成功运行起来了。

（7）检查虚拟机环境中的交叉编译工具链的版本是否为 4.6.4，具体操作是使用下面命令查看：

```
arm-none-linux-gnueabi-gcc –v
```



如果工具链的版本不对，可以在/etc/bash.bashrc 文件中添加或修改。具体的操作是：

```
sudo vi /etc/bash.bashrc
```

在上面打开的文件的最后添加：

```
export PATH=$PATH:/usr/local/toolchain-4.6.4/bin
```

```
62          fi
63      }
64 fi
65 export PATH=$PATH:/usr/local/toolchain/toolchain-4.6.4/bin/
66 #export PATH=$PATH:/usr/local/toolchain/toolchain-4.4.6/bin/
/etc/bash.bashrc
```

修改完成后保存退出，并执行下面指令：

```
Source /etc/bash.bashrc
```

（8）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\DC_dynamo】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd DC_dynamo
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
  sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c -o DC_dynamo
```

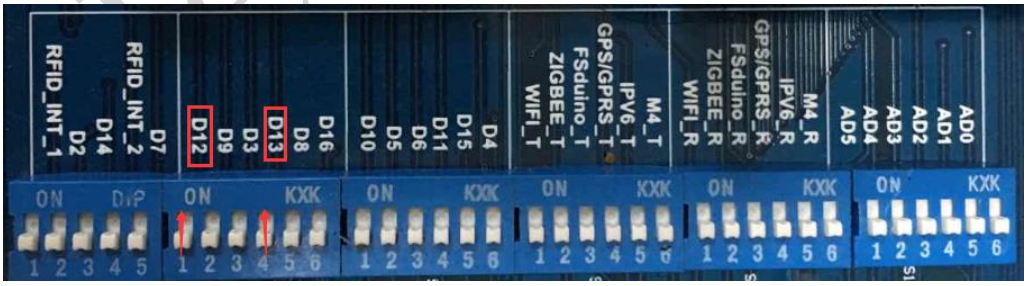
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
  sudo cp DC_dynamo /source/rootfs
```

（9）加载驱动，创建设备文件，执行应用程序

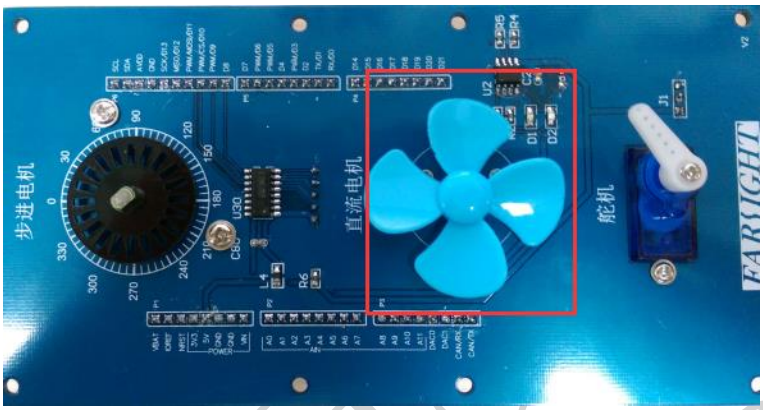
```
insmod    fs4412_dynamo.ko
mknod /dev/dynamo c 800 3
./DC_dynamo
```

（10）将拨码开关中的 D12、D13 拨至 ON，其他拨码全为 OFF，如下图



2.1.7 【实验结果】

如果以上步骤正确完成，可以看到直流电机每隔大概一秒中变换一次转向，如果想要停止该程序，在 Putty 中使用 Ctrl + c 键即可。



2.2 步进电机驱动实验

2.2.1 【实验目的】

（1） 掌握步进电机的工作原理；

（2） 掌握 Linux 下步进电机驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.2.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

华清远见 dev.hqyj.com

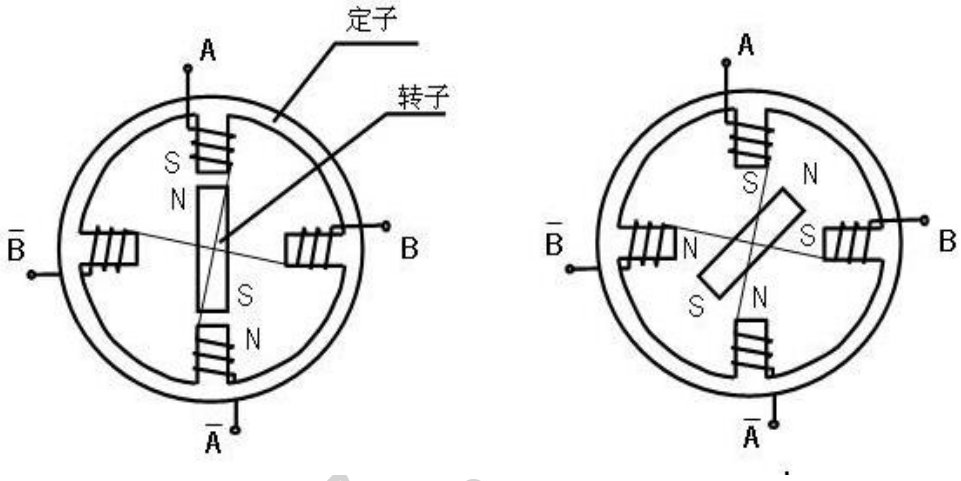
2.2.3 【实验内容】

基于 Linux3.0 内核实现驱动控制步进电机工作。

2.2.4 【实验原理】

步进电动机是一种将电脉冲信号转换成角位移或线位移的机电元件。步进电动机的输入量是脉冲序列，输出量则为相应的增量位移或步进运动。正常运动情况下，它每转一周具有固定的步数；做连续步进运动时 ,其旋转转速与输入脉冲的频率保持严格的对应关系(所以可以控制每个脉冲的宽度去调节速度每个脉冲转速)，由于步进电动机能直接接受数字量的控制，所以特别适宜采用微机进行控制。

步进电机接线：



8 拍的方式八个状态：

1、A 高电平

2、A 与 B 高电平

3、B 高电平

4、B 与 A-高电平

5、A-给高电平

6、A-与 B-给高电平

7、B-给高电平

8、B-与 A 给高电平

按以上八个状态轮流供电，控制一下脉宽（用精确延迟就可以控制）就可以了。

4 拍的方式

1、A 高电平

2、B 高电平

3、A-高电平

4、B-高电平

如果要实现反转：只要把时序倒过来就可以实现。

我们所用步进电机型号为 28byj-48，参数如下

主要技术参数

N=64

电机型号	电压 V	相数	相电阻Ω ±10%	步距角度	减速比	起动转矩 100P.P.S g.cm	起动频率 P.P.S	定位转矩 g.cm	摩擦转矩 g.cm	噪声 dB	绝缘介电强度
28BYJ48	5	4	300	5.625/64	1:64	≥300	≥550	≥300	—	≤35	600VAC 1S

主要技术参数：

1、相数：产生不同对极 N、S 磁场的激磁线圈对数。常用 m 表示。

2、拍数：完成一个磁场周期性变化所需脉冲数或导电状态用 n 表示，或指电机转过一个齿距角所需脉冲数，以四相电机为例，有四相四拍运行方式即 AB-BC-CD-DA-AB，四相八拍运行方式即 A-AB-B-BC-C-CD-D-DA-A。

3、步距角：对应一个脉冲信号，电机转子转过的角位移用 θ 表示。θ=360 度/（转子齿数*运行拍数），以常规二、四相，转子齿为 50 齿电机为例。四拍运行时步距角为 θ=360 度/（50*4）=1.8 度（俗称整步），八拍运行时步距角为 θ=360 度/（50*8）=0.9 度（俗称半步）。

我们通过控制磁场方向的变化来改变转动方向，这时磁场转动一圈并不代表电机对应要转一圈，这里有一个参数为减速比，通过内部的转子齿来减速，所以磁场转动一圈，电机转动 360/64 度。

步进电机硬件接线图：

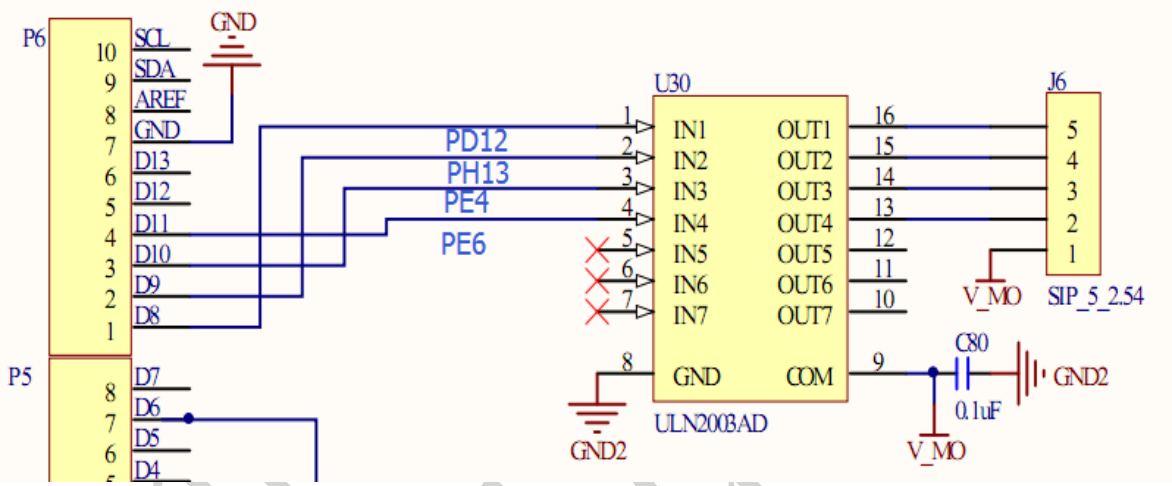


图 原理图

2.2.5 【源码分析】

有些类似的程序这里就不再赘述了，可以参考【直流电机驱动实验】中的【源码分析】，这里只描述重点的代码。

```
1
2  /* 将步进电机相关 GPIO 设置为高电平 */
3  void fs4412_stepper_on(int nr)
4  {
5      switch (nr)
6      {
7      case 1:
8          writel(readl(gpl1dat) | 0x1 << 0, gpl1dat);
```

```
9           break;
10      case 2:
11          writel(readl(gpc1dat) | 0x1 << 4, gpc1dat);
12          break;
13      case 3:
14          writel(readl(gpc1dat) | 0x1 << 1, gpc1dat);
15          break;
16      case 4:
17          writel(readl(gpm4dat) | 0x1 << 6, gpm4dat);
18          break;
19      }
20  }
21
22  /* 将步进电机相关 GPIO 设置为低电平 */
23  void fs4412_stepper_off(int nr)
24  {
25      switch (nr)
26      {
27      case 1:
28          writel(readl(gpl1dat) & ~(0x1 << 0), gpl1dat);
29          break;
30      case 2:
31          writel(readl(gpc1dat) & ~(0x1 << 4), gpc1dat);
32          break;
33      case 3:
34          writel(readl(gpc1dat) & ~(0x1 << 1), gpc1dat);
35          break;
36      case 4:
37          writel(readl(gpm4dat) & ~(0x1 << 6), gpm4dat);
38          break;
39      }
40  }
41
42  /* 打开步进电机这个设备对应的设备文件 */
43  static int fs4412_stepper_open(struct inode *inode, struct file *file)
44  {
45      return 0;
46  }
47
48  /* 在设备使用结束之后为了防止其占用资源，需要将设备释放掉 */
49  static int fs4412_stepper_release(struct inode *inode, struct file *file)
50  {
51      fs4412_stepper_off(1);
```

```
52      fs4412_stepper_off(2);
53      fs4412_stepper_off(3);
54      fs4412_stepper_off(4);
55      return 0;
56  }
57
58  /* 控制步进电机函数实现，当用户层调用 ioctl 函数时，该函数将被调用 */
59  static long fs4412_stepper_unlocked_ioctl
60  (struct file *file, unsigned int cmd, unsigned long arg)
61  {
62      int nr;
63      int n;
64
65      if (arg < 1 || arg > 4)
66          return -EINVAL;
67
68      switch(cmd)
69      {
70          /* 如果检测到用户层的命令是打开电机，就将对应的 GPIO 设置为高电平 */
71      case STEPPER_ON:
72          printk("fs4412_stepper_on\n");
73          fs4412_stepper_on(arg);
74          break;
75          /* 如果检测到用户层的命令是关闭电机，就将对应的 GPIO 设置为低电平*/
76      case STEPPER_OFF:
77          printk("fs4412_stepper_off\n");
78          fs4412_stepper_off(arg);
79          break;
80
81      }
82      return 0;
83
84  }
85
86  /* 步进电机 GPIO 初始化函数，主要配置 GPIO 为输出模式，以及配置引脚的初始电平 */
87  void fs4412_stepper_io_init(void)
88  {
89      /*init GPL1_0 GPL1CON0 -> output  GPL1DAT0 -> 0*/
90      writel((readl(gpl1con) & ~(0xf << 0) | (0x1 << 0)), gpl1con);
91      writel((readl(gpl1dat) & ~(0x1 << 0)), gpl1dat);
92      /*init GPC1_4 GPC1CON4 -> output GPC1DAT4 -> 0*/
93      writel((readl(gpc1con) & ~(0xf << 16) | (0x1 << 16)), gpc1con);
94      writel((readl(gpc1dat) & ~(0x1 << 4)), gpc1dat);
```

```
95
96      /*init GPC1_1 GPC1CON1 -> output GPC1DAT1 -> 0*/
97      writel((readl(gpc1con) & ~(0xf << 4) | (0x1 << 4)), gpc1con);
98      writel((readl(gpc1dat) & ~(0x1 << 1)), gpc1dat);
99
100     /*init GPM4_6 GPM4CON6 -> output GPM4DAT6 -> 0*/
101     writel((readl(gpm4con) & ~(0xf << 24) | (0x1 << 24)), gpm4con);
102     writel((readl(gpm4dat) & ~(0x1 << 6)), gpm4dat);
103
104     printk("STEPPER REGISTER INIT OKAY\n");
105 }
106
107 /* 操作设备结构体定义，这里包含三个操作方式，打开、关闭、控制 */
108 struct file_operations fs4412_stepper_fops =
109 {
110     .owner = THIS_MODULE,
111     .open  = fs4412_stepper_open,
112     .release = fs4412_stepper_release,
113     .unlocked_ioctl = fs4412_stepper_unlocked_ioctl,
114
115 };
116
117 /* 驱动模块初始化函数，加载驱动时该函数会被调用到 */
118 static int fs4412_stepper_init(void)
119 {
120     /* 将主次设备号转换成 dev_t 类型 */
121     dev_t devno = MKDEV(STEPPER_MAJOR, STEPPER_MINOR);
122     int ret;
123     /* 为一个字符驱动获取一个或多个设备编号来使用 */
124     ret = register_chrdev_region(devno, 1, "STEPPER");
125     if(ret < 0)
126     {
127         printk("register_chrdev_region failed\n");
128         return ret;
129     }
130     /* cdev 结构体初始化 */
131     cdev_init(&cdev, &fs4412_stepper_fops);
132     cdev.owner = THIS_MODULE;
133     /* 将 cdev 添加到系统中去 */
134     ret = cdev_add(&cdev, devno, 1);
135     if (ret < 0)
136     {
137         printk("cdev_add\n");
```

```
138         goto err1;
139     }
140     /* 调用 GPIO 地址映射函数，进行地址映射 */
141     ret = fs4412_stepper_ioremap();
142     if (ret < 0)
143     {
144         goto err2;
145     }
146     /* 调用 GPIO 初始化函数，配置 GPIO 相关属性 */
147     fs4412_stepper_io_init();
148     return 0;
149
150 err2:
151     /* 从系统中移除 cdev 结构体 */
152     cdev_del(&cdev);
153 err1:
154     /* 释放之前注册的设备号 */
155     unregister_chrdev_region(devno, 1);
156     return ret;
157 }
158 /* 卸载驱动模块时调用该函数 */
159 static void fs4412_stepper_exit(void)
160 {
161
162     dev_t devno = MKDEV(STEPPER_MAJOR, STEPPER_MINOR);
163     /* 卸载驱动模块时，将之前映射的地址释放掉 */
164     fs4412_stepper_iounmap();
165     /* 卸载驱动模块时，删除系统中之前添加的 cdev */
166     cdev_del(&cdev);
167     /* 卸载驱动模块时，释放之前注册的设备号 */
168     unregister_chrdev_region(devno, 1);
169     printk("STEPPER DRIVER EXIT\n");
    }
```

2.2.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱-II \Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\stepper】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd stepper
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c –o stepper
```

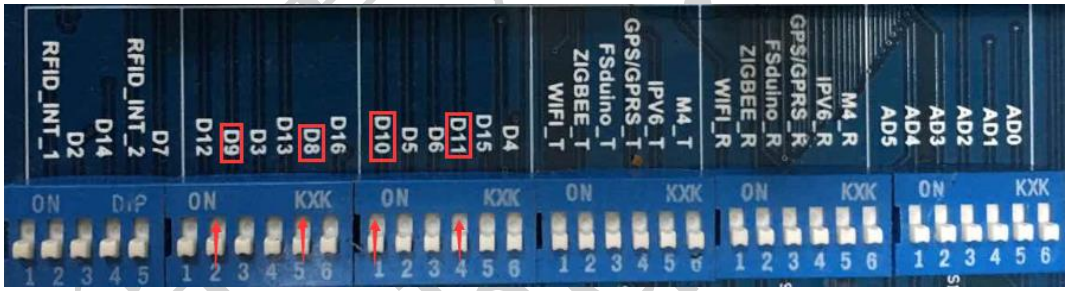
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp stepper /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

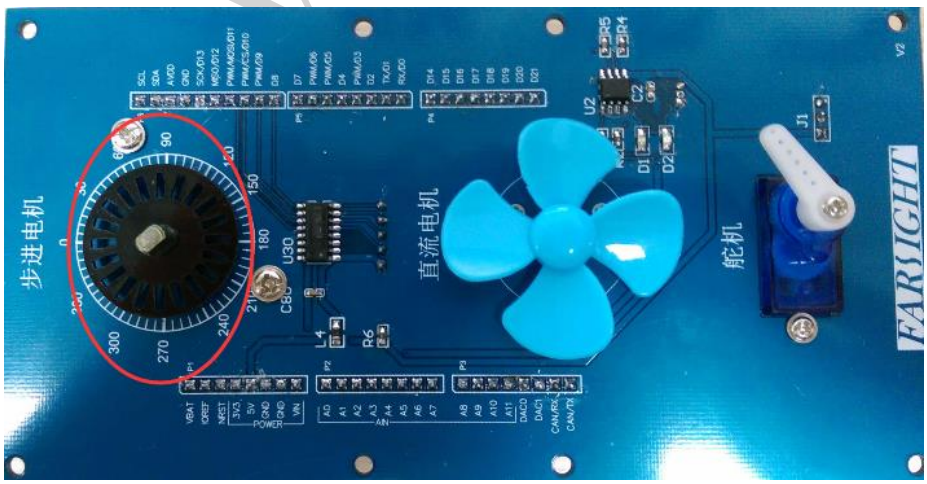
```
insmod    fs4412-stepper.ko
mknod /dev/stepper c 800 2
./stepper
```

（3）将拨码开关中的 D8、D9、D10、D11 拨至 ON，其他拨码全为 OFF，如下图

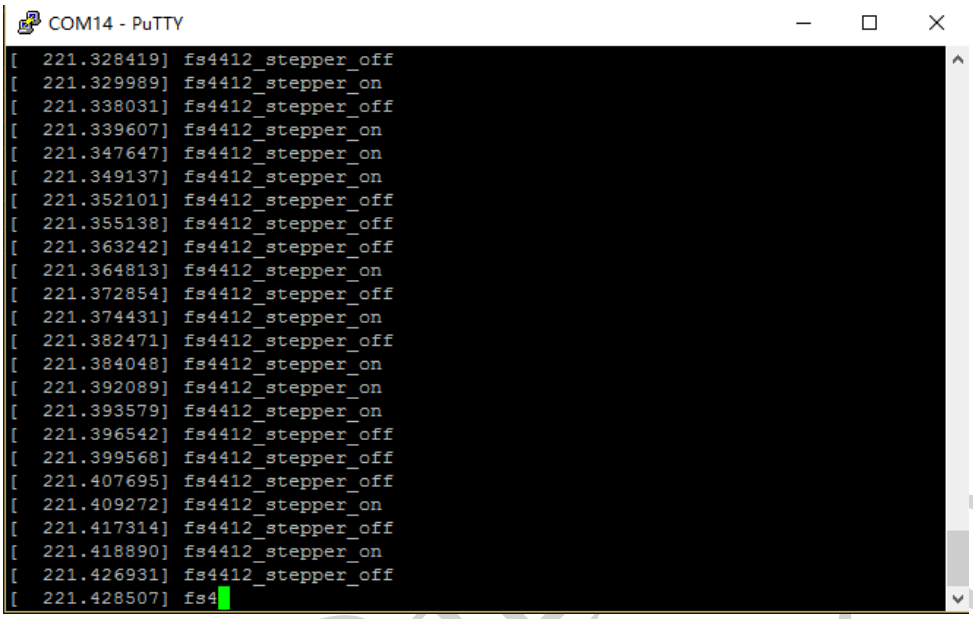


2.2.7 【实验结果】

如果正确按照上面的步骤操作完成，会看到步进电机开始转动：



在 Putty 中会有如下显示：



2.3 舵机驱动实验

2.3.1 【实验目的】

（1） 掌握舵机的工作原理；

（2） 掌握 Linux 下舵机驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.3.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

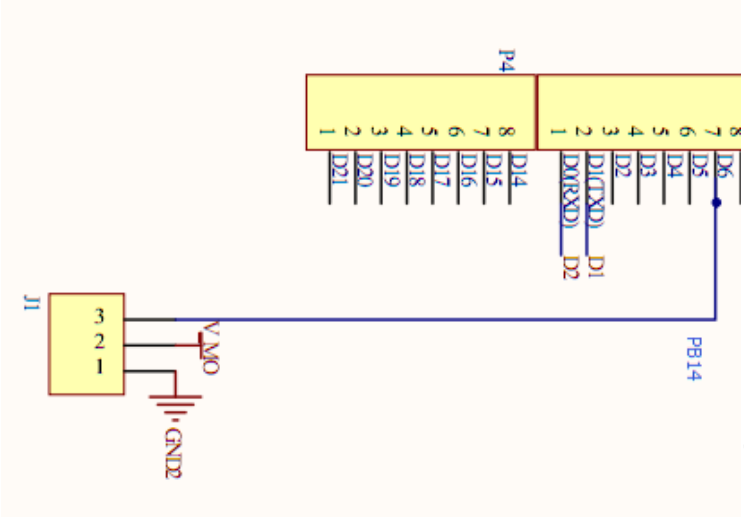
华清远见 dev.hqyj.com

2.3.3 【实验内容】

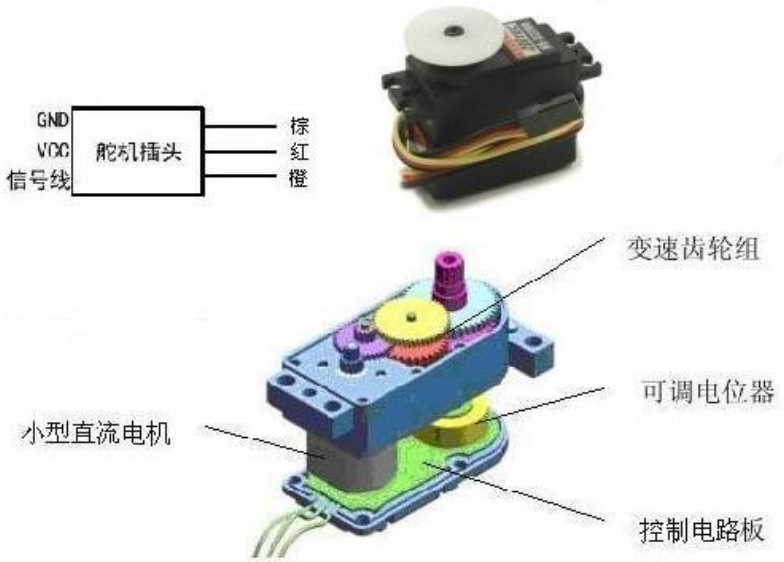
基于 Linux3.0 内核实现驱动舵机工作。

2.3.4 【实验原理】

该实验需要使用到开发板上的舵机，相关硬件电路如下：



舵机连接电路



舵机原理图

控制信号由接收机的通道进入信号调制芯片，获得直流偏置电压。它内部有一个基准电路，产生周期

为 20ms，宽度为 1.5ms 的基准信号，将获得的直流偏置电压与电位器的电压比较，获得电压差输出。最后，电压差的正负输出到电机驱动芯片决定电机的正反转。当电机转速一定时，通过级联减速齿轮带动电位器旋转，使得电压差为 0，电机停止转动。

舵机的控制一般需要一个 20ms 左右的时基脉冲，该脉冲的高电平部分一般为 0.5ms-2.5ms 范围内的角度控制脉冲部分，总间隔为 2ms。以 180 度角度伺服为例，那么对应的控制关系是这样的：

0.5ms----------0 度；

1.0ms----------45 度；

1.5ms----------90 度；

2.0ms----------135 度；

2.5ms----------180 度；

可以看到舵机的控制主要通过 PWM 来完成，但是在当前的实验平台上面，FS4412 与舵机连接的 GPIO 没有 PWM 功能，所以我们采用的是 IO 模拟的方式来实现的。具体实验过程可以参考驱动源码。

2.3.5 【源码分析】

有些类似的程序这里就不再赘述了，可以参考【直流电机驱动实验】中的【源码分析】，这里只描述重点的代码。

```
1   /* 设置舵机相关 GPIO 为高电平 */
2  void fs4412_servo_on(int nr)
3  {
4      switch (nr)
5      {
6      case 1:
7          writel(readl(gpl1dat) | 0x1 << 0, gpl1dat);
8          printk("gpl1dat on\n");
9          break;
10     }
11 }
12
```

```
13 /* 设置舵机相关 GPIO 为低电平 */
14 void fs4412_servo_off(int nr)
15 {
16     switch (nr)
17     {
18     case 1:
19         writel(readl(gpl1dat) & ~(0x1 << 0), gpl1dat);
20         printk("gpl1data off\n");
21         break;
22     }
23 }
24
25 /* 打开舵机这个设备对应的设备文件 */
26 static int fs4412_servo_open(struct inode *inode, struct file *file)
27 {
28     return 0;
29 }
30 /* 在设备使用结束之后为了防止其占用资源，需要将设备释放掉 */
31 static int fs4412_servo_release(struct inode *inode, struct file *file)
32 {
33     fs4412_servo_off(1);
34     return 0;
35 }
36
37 /* 控制舵机函数实现，当用户层调用 ioctl 时，该函数将被调用 */
38 static long fs4412_servo_unlocked_ioctl
39 (struct file *file, unsigned int cmd, unsigned long arg)
40 {
41     int nr;
42     int n;
43
44     if (arg < 1 || arg > 4)
45         return -EINVAL;
46
47     switch(cmd)
48     {
49         /* 如果检测到用户层的命令 SERVO_ON,就将对应 GPIO 设置为高电平 */
50     case SERVO_ON:
51         fs4412_servo_on(arg);
52         break;
53         /* 如果检测到用户层的命令 SERVO_OFF,就将对应 GPIO 设置为低电平 */
54     case SERVO_OFF:
55         fs4412_servo_off(arg);
```

```
56          break;
57      }
58      return 0;
59
60  }
61
62  /* 舵机 GPIO 初始化函数，主要配置 GPIO 为输出模式，以及配置引脚的初始电平 */
63  void fs4412_servo_io_init(void)
64  {
65      /*init GPL1_0 GPL1CON0 -> output  GPL1DAT0 -> 0*/
66      writel((readl(gpl1con) & ~(0xf << 0) | (0x1 << 0)), gpl1con);
67      writel((readl(gpl1dat) & ~(0x1 << 0)), gpl1dat);
68      printk("SERVO REGISTER INIT OKAY\n");
69  }
70
71  /* 舵机 GPIO 相关寄存器地址映射 */
72  int fs4412_servo_ioremap(void)
73  {
74      int ret;
75
76      gpl1con = ioremap(GPL1CON, 4);
77      if(gpl1con == NULL)
78      {
79          printk("ioremap gpl1con\n");
80          ret = -ENOMEM;
81          return ret;
82      }
83      gpl1dat = ioremap(GPL1DAT, 4);
84      if(gpl1dat == NULL)
85      {
86          printk("ioremap gpl1dat\n");
87          ret = -ENOMEM;
88          return ret;
89      }
90      return 0;
91  }
92
93  /* 驱动卸载时需要将之前映射的地址释放掉 */
94  void fs4412_servo_iounmap(void)
95  {
96      iounmap(gpl1con);
97      iounmap(gpl1dat);
98  }
```

```
99
100 /* 操作设备结构体定义，这里包含三个操作方式，打开、关闭、控制 */
101 struct file_operations fs4412_servo_fops =
102 {
103     .owner = THIS_MODULE,
104     .open  = fs4412_servo_open,
105     .release = fs4412_servo_release,
106     .unlocked_ioctl = fs4412_servo_unlocked_ioctl,
107
108 };
109
110 /* 驱动模块初始化函数，加载驱动时该函数会被调用 */
111 static int fs4412_servo_init(void)
112 {
113     /* 将主次设备号转换成 dev_t 类型 */
114     dev_t devno = MKDEV(SERVO_MAJOR, SERVO_MINOR);
115     int ret;
116     /* 为一个字符驱动获取一个或多个设备编号来使用 */
117     ret = register_chrdev_region(devno, 1, "SERVO");
118     if(ret < 0)
119     {
120         printk("register_chrdev_region failed\n");
121         return ret;
122     }
123     /* cdev 结构体初始化 */
124     cdev_init(&cdev, &fs4412_servo_fops);
125     cdev.owner = THIS_MODULE;
126     /* 将 cdev 添加到系统中去 */
127     ret = cdev_add(&cdev, devno, 1);
128     if (ret < 0)
129     {
130         printk("cdev_add\n");
131         goto err1;
132     }
133     /* 舵机 GPIO 相关寄存器地址映射 */
134     ret = fs4412_servo_ioremap();
135     if (ret < 0)
136     {
137         goto err2;
138     }
139     /*init con & dat register*/
140     fs4412_servo_io_init();
141     return 0;
```

```
142
143 err2:
144     cdev_del(&cdev);
145 err1:
146     unregister_chrdev_region(devno, 1);
147     return ret;
148 }
```

2.3.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱资料-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\servo】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd servo
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c –o servo
```

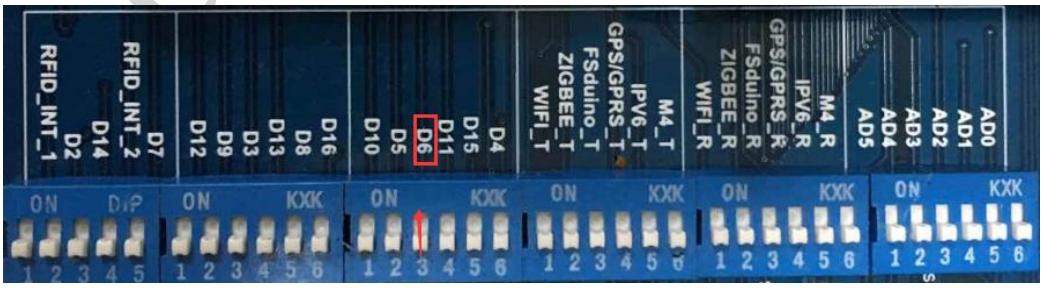
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp servo /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

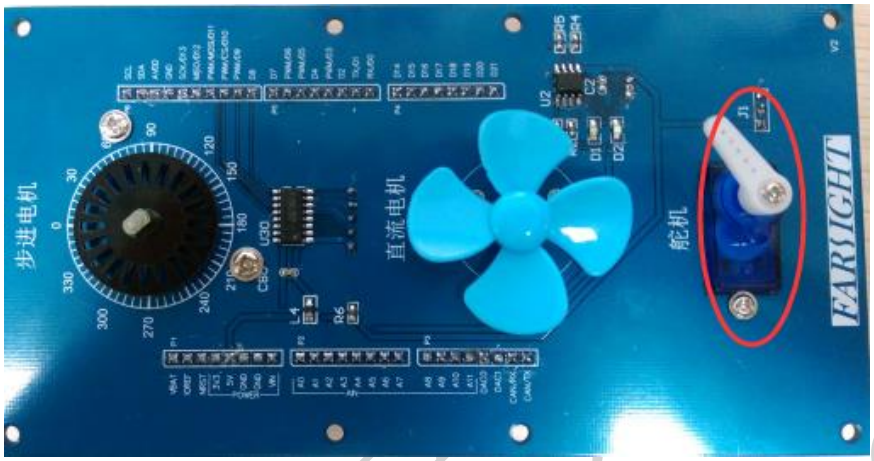
```
insmod    fs4412_servo.ko
mknod /dev/servo c 501 0
./servo
```

（3）将拨码开关中的 D6 拨至 ON，其他拨码全为 OFF，如下图

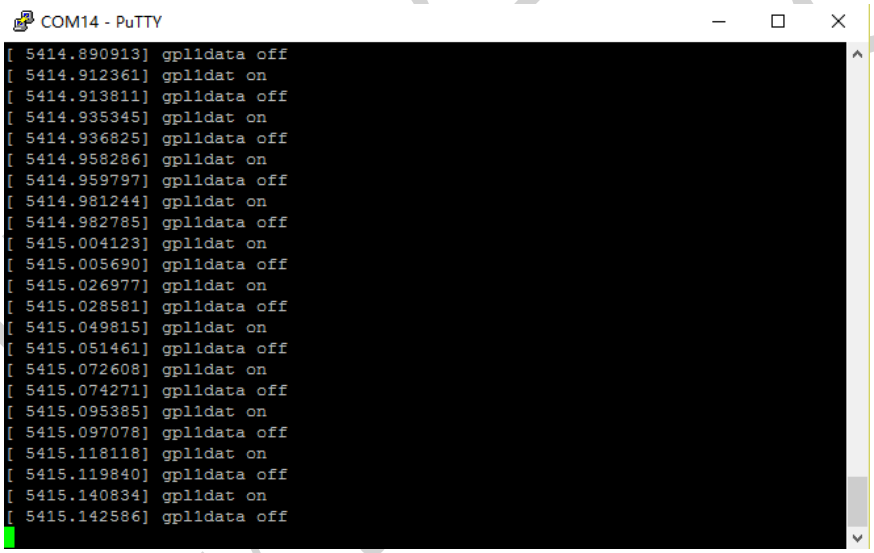


2.3.7 【实验结果】

如果按照上面的步骤操作正确的话，可以看到舵机开始摆动。



从 Putty 中可以看到如下显示：



2.4 继电器驱动实验

2.4.1 【实验目的】

（1） 掌握继电器的工作原理；

（2） 掌握 Linux 下继电器驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.4.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

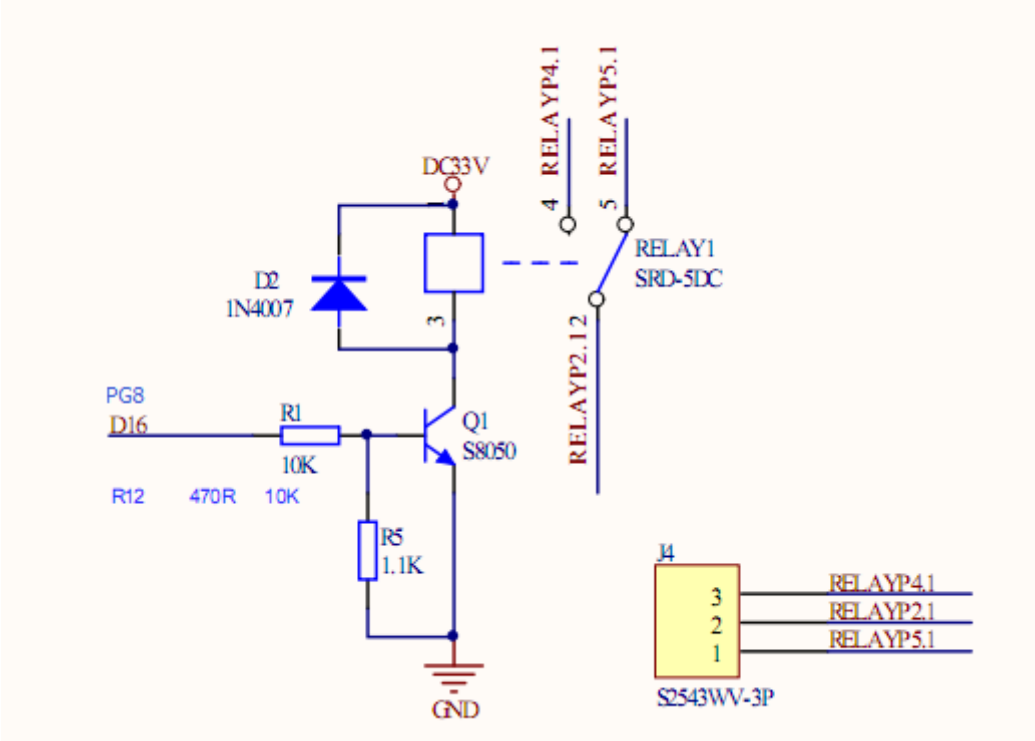
（5） NFS 网络文件系统；

2.4.3 【实验内容】

基于 Linux3.0 内核实现驱动控制继电器通断。

2.4.4 【实验原理】

该实验需要使用到开发板上的直流电机，相关硬件电路如下：



继电器连接电路图

由上图可以看出单片机控制的 IO 口在 PG8 上。继电器原理说明如下：单片机是一个弱电器件,一般情况下它们大都工作在 5V 甚至更低.驱动电流在 mA 级以下而要把它用于一些大功率场合,比如控制电动机,显然是不行的.所以,就要有一个环节来衔接,这个环节就是所谓的"功率驱动".继电器驱动就是一个典型的简单的功率驱动环节.在这里,继电器驱动含有两个意思:一是对继电器进行驱动,因为继电器本身对于单片机来说就是一个功率器件;还有就是继电器去驱动其他负载,比如继电器可以驱动中间继电器,可以直接驱动接触器,所以,继电器驱动就是单片机与其他大功率负载接口.这个很重要,因为,一直让我们的电气工程师(我指的是那些没有学习过相应的电子技术的)感到迷惑不解的是:一个小小的芯片,怎么会有如此强大的威力来控制像电动机这样强大的东西?

怎么样理解这个电路图? 要理解这个电路,其实也比较容易.那么请您按照我的思路来,应该没有问题: 首先的,里面三极管是电子电路里很重要的一个元件.怎么样理解三极管呢? 简单的来说三极管有两个作用一个是放大作用,一个是开关作用.(严格来讲开关作用是放大作用的极限情况不过没关系,把两者分开,更便于理解它的工作原理).在这里,我们只了解它跟本电路有关的开关作用. 首先把三极管想成一个水龙头. 上面的 Vcc 就是水池,继电器是一个水轮机,下面的 GND 是比水池低的任何一点.刚才说过,三极管就是水龙头,它的把手就是那个带有电阻的引脚. 现在,单片机的某一个需要控制这个继电器电路的输出引脚就是一只"手",当单片机的这个引脚输出低电平的时候,就像"手"在打开三极管"水龙头",水就从上往下流,继电器"水轮机"就开始转起来了.反之,如果是输出高电平,"手"就开始关"水龙头",继电器"水轮机"因为没有水流下来,就会停止. 这就是三极管的开关作用. 简单的理解和记忆就是:三极管是一个开关器件,其实你真的可以将它看成是一个开关,只不过它不是用手来控制,而是用电压(电流)来控制的,因此,三极管有些时候也被称做电子开关(与机械开关相区别). 图上还有一个东西,是保护二极管,如果不需要深入理解的话,你大可不必追就为什么有它存在,但是一定得记住,只要是用三极管驱动继电器的场合,一般都有它的存在.需要特别注意的是它的接法:并联在继电器两端 阴极一定是接 Vcc 。

2.4.5 【源码分析】

有些类似的程序这里就不再赘述了，可以参考【直流电机驱动实验】中的【源码分析】，这里只描述重点的代码。

```
    /* 设置 GPIO 脚为高电平 */
void fs4412_gpio_on(int nr)
{
    switch(nr)
    {
    case 1:
        writel(readl(gpc1dat) | 1 << 1, gpc1dat);
        break;
    case 2:
        writel(readl(gpc1dat) | 1 << 4, gpc1dat);
        break;
    }
}

/* 设置 GPIO 脚为低电平 */
void fs4412_gpio_off(int nr)
{
    switch(nr)
    {
    case 1:
        writel(readl(gpc1dat) & ~(1 << 1), gpc1dat);
        break;
    case 2:
        writel(readl(gpc1dat) & ~(1 << 4), gpc1dat);
        break;
    }
}

/* 控制设备函数实现，当用户层调用 ioctl 函数时，该函数会被调用 */
static long fs4412_gpio_unlocked_ioctl
(struct file *file, unsigned int cmd, unsigned long arg)
{
    int nr;
    /* 从用户空间获取需要控制的 GPIO 管脚 */
    if(copy_from_user((void *)&nr, (void *)arg, sizeof(nr)))
        return -EFAULT;

```

```
38      if (nr < 1 || nr > 4)
39          return -EINVAL;
40
41      switch (cmd)
42      {
43          /* 当检测到用户空间传来的命令为 GPIO_ON 时，将对应的 GPIO 设置为高电平 */
44      case GPIO_ON:
45          fs4412_gpio_on(nr);
46          break;
47          /* 当检测到用户空间传来的命令为 GPIO_OFF 时，将对应的 GPIO 设置为低电平 */
48      case GPIO_OFF:
49          fs4412_gpio_off(nr);
50          break;
51      default:
52          printk("Invalid argument");
53          return -EINVAL;
54      }
55
56      return 0;
57  }
58  /* GPIO 管脚初始化函数 */
59  void fs4412_gpio_io_init(void)
60  {
61
62      writel((readl(gpc1con) & ~(0xf << 16)) | (0x1 << 16), gpc1con);
63      writel(readl(gpc1dat) & ~(0x1 << 4), gpc1dat);
64
65      writel((readl(gpc1con) & ~(0xf << 4)) | (0x1 << 4), gpc1con);
66      writel(readl(gpc1dat) & ~(0x1 << 1), gpc1dat);
67  }
68
69  /* 字符设备文件操作结构体，定义对文件的操作方法 */
70  struct file_operations fs4412_gpio_fops =
71  {
72      .owner = THIS_MODULE,
73      .open = fs4412_gpio_open,
74      .release = fs4412_gpio_release,
75      .unlocked_ioctl = fs4412_gpio_unlocked_ioctl,
76  };
77
78  /* 驱动模块初始化函数 */
79  static int fs4412_gpio_init(void)
80  {
```

```
81      /* 将主次设备号转换成 dev_t 形式 */
82      dev_t devno = MKDEV(GPIO_MA, GPIO_MI);
83      int ret;
84      /* 为字符设备驱动获取一个或几个设备号 */
85      ret = register_chrdev_region(devno, GPIO_NUM, "newgpio");
86      if (ret < 0)
87      {
88          printk("register_chrdev_region\n");
89          return ret;
90      }
91      /* cdev 结构体初始化 */
92      cdev_init(&cdev, &fs4412_gpio_fops);
93      cdev.owner = THIS_MODULE;
94      /* 将 cdev 添加到系统中去 */
95      ret = cdev_add(&cdev, devno, GPIO_NUM);
96      if (ret < 0)
97      {
98          printk("cdev_add\n");
99          goto err1;
100     }
101     /* GPIO 相关寄存器物理地址映射 */
102     ret = fs4412_gpio_ioremap();
103     if (ret < 0)
104         goto err2;
105     /* GPIO 管脚初始化，设置为输出模式 */
106     fs4412_gpio_io_init();
107
108     printk("gpio init\n");
109
110     return 0;
111 err2:
112     cdev_del(&cdev);
113 err1:
114     unregister_chrdev_region(devno, GPIO_NUM);
115     return ret;
116 }
```

2.4.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱资料-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\fs4412_gpio】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd fs4412_gpio
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c –o gpio
```

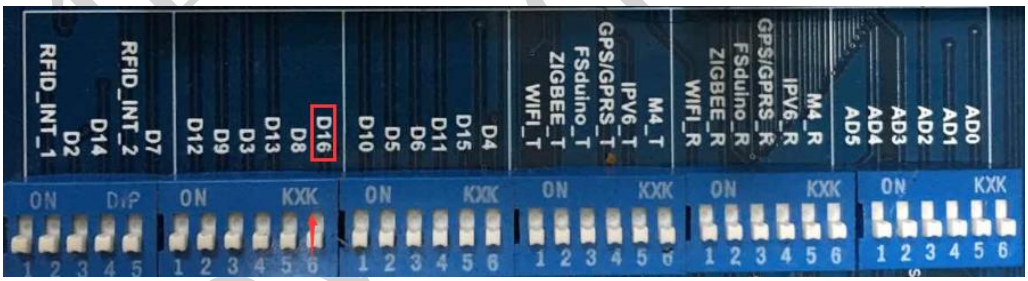
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp gpio /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

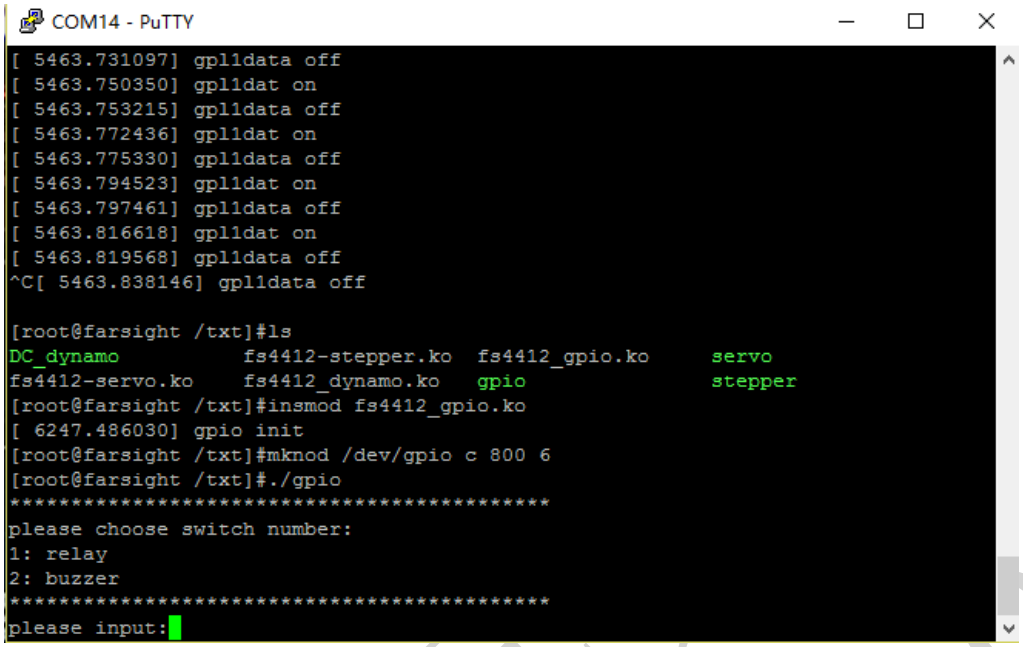
```
insmod    fs4412_gpio.ko
mknod /dev/gpio c 800 6
./gpio
```

（3）将拨码开关中的 D16 拨至 ON，其他拨码全为 OFF，如下图

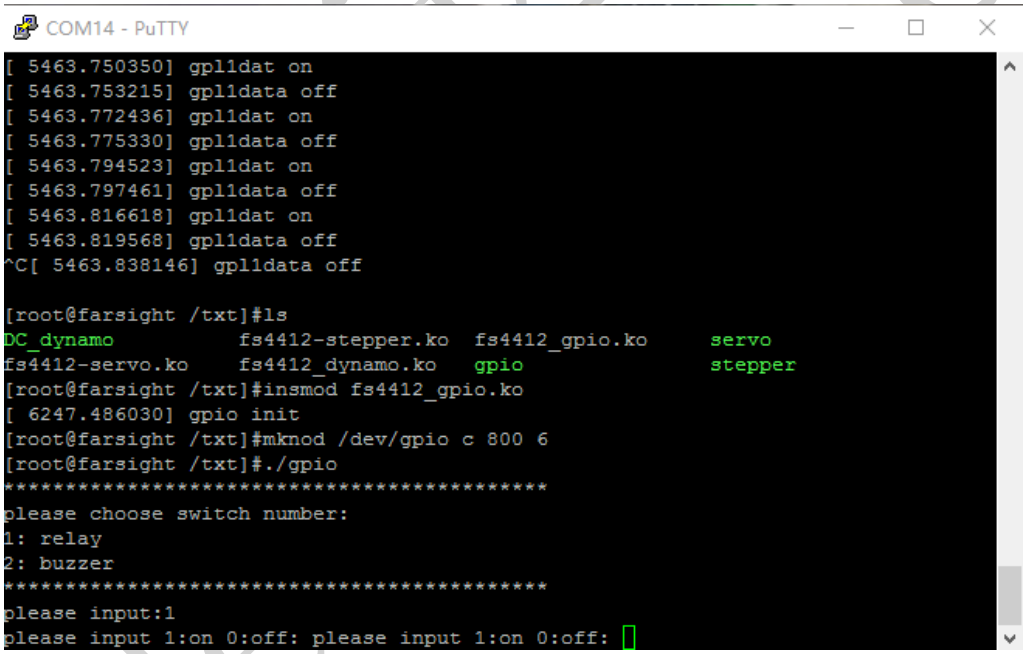


2.4.7 【实验结果】

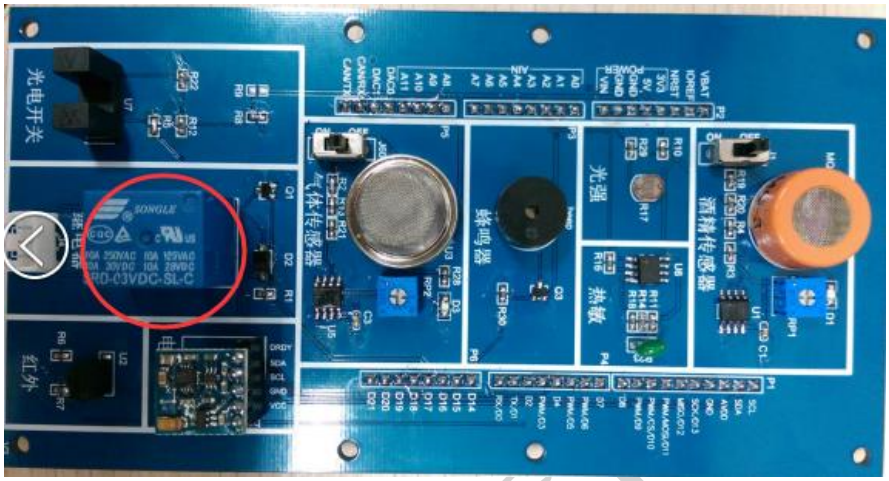
如果按照上面的步骤正确操作，在 Putty 上会出现如下界面：



这里测试的是继电器，所以输入 1，可以看到如下提示：



输入 1 是打开，输入 0 是闭合，在这个过程中可以听到继电器发出的声音。



2.5 蜂鸣器驱动实验

2.5.1 【实验目的】

（1） 掌握蜂鸣器的工作原理；

（2） 掌握 Linux 下蜂鸣器驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.5.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

2.5.3 【实验内容】

基于 Linux3.0 内核实现驱动控制蜂鸣器工作。

华清远见 dev.hqyj.com

2.5.4 【实验原理】

蜂鸣器类型：有源蜂鸣器和无源蜂鸣器。

有源蜂鸣器内置振荡电路，直接加电源就可以正常发声，通常频率固定。

无源蜂鸣器则需要通过外部的正弦或方波信号驱动，直接加电源只能发出很轻微的振动声。

蜂鸣器原理图：

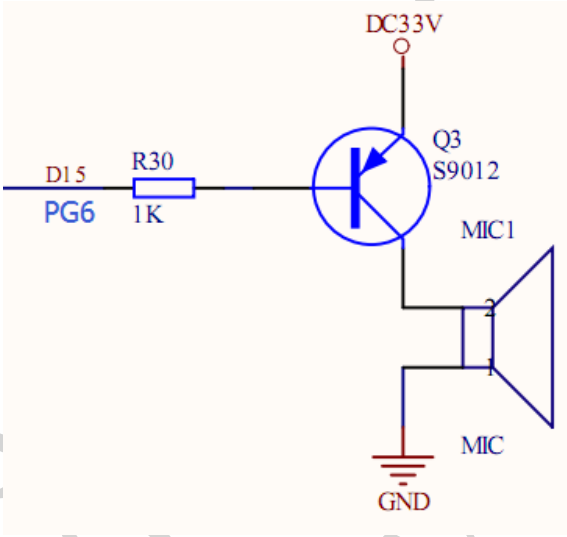


图 蜂鸣器硬件原理图

本平台使用的蜂鸣器是无源蜂鸣器，所以需要使用 PWM 去控制，但是由于扩展 IO 有限，这里跟蜂鸣器相连接的 GPIO 没有 PWM 功能，所以只能模拟，具体的模拟方式请参考驱动源码。

2.5.5 【源码分析】

有些类似的程序这里就不再赘述了，可以参考【直流电机驱动实验】中的【源码分析】，这里只描述重点的代码。

```
1    /* 设置 GPIO 脚为高电平 */
2 void fs4412_gpio_on(int nr)
3 {
4     switch(nr)
5     {
6     case 1:
7         writel(readl(gpc1dat) | 1 << 1, gpc1dat);
```

```
        break;
    case 2:
        writel(readl(gpc1dat) | 1 << 4, gpc1dat);
        break;
    }
}

/* 设置 GPIO 脚为低电平 */
void fs4412_gpio_off(int nr)
{
    switch(nr)
    {
    case 1:
        writel(readl(gpc1dat) & ~(1 << 1), gpc1dat);
        break;
    case 2:
        writel(readl(gpc1dat) & ~(1 << 4), gpc1dat);
        break;
    }
}

/* 控制设备函数实现，当用户层调用 ioctl 函数时，该函数会被调用 */
static long fs4412_gpio_unlocked_ioctl
(struct file *file, unsigned int cmd, unsigned long arg)
{
    int nr;
    /* 从用户空间获取需要控制的 GPIO 管脚 */
    if(copy_from_user((void *)&nr, (void *)arg, sizeof(nr)))
        return -EFAULT;

    if (nr < 1 || nr > 4)
        return -EINVAL;

    switch (cmd)
    {
        /* 当检测到用户空间传来的命令为 GPIO_ON 时，将对应的 GPIO 设置为高电平 */
    case GPIO_ON:
        fs4412_gpio_on(nr);
        break;
        /* 当检测到用户空间传来的命令为 GPIO_OFF 时，将对应的 GPIO 设置为低电平 */
    case GPIO_OFF:
        fs4412_gpio_off(nr);
        break;
```

```
    default:
        printk("Invalid argument");
        return -EINVAL;
    }

    return 0;
}
/* GPIO 管脚初始化函数 */
void fs4412_gpio_io_init(void)
{

    writel((readl(gpc1con) & ~(0xf << 16)) | (0x1 << 16), gpc1con);
    writel(readl(gpc1dat) & ~(0x1 << 4), gpc1dat);

    writel((readl(gpc1con) & ~(0xf << 4)) | (0x1 << 4), gpc1con);
    writel(readl(gpc1dat) & ~(0x1 << 1), gpc1dat);
}

/* 字符设备文件操作结构体，定义对文件的操作方法 */
struct file_operations fs4412_gpio_fops =
{
    .owner = THIS_MODULE,
    .open = fs4412_gpio_open,
    .release = fs4412_gpio_release,
    .unlocked_ioctl = fs4412_gpio_unlocked_ioctl,
};

/* 驱动模块初始化函数 */
static int fs4412_gpio_init(void)
{
    /* 将主次设备号转换成 dev_t 形式 */
    dev_t devno = MKDEV(GPIO_MA, GPIO_MI);
    int ret;
    /* 为字符设备驱动获取一个或几个设备号 */
    ret = register_chrdev_region(devno, GPIO_NUM, "newgpio");
    if (ret < 0)
    {
        printk("register_chrdev_region\n");
        return ret;
    }
    /* cdev 结构体初始化 */
    cdev_init(&cdev, &fs4412_gpio_fops);
    cdev.owner = THIS_MODULE;
```

```
94      /* 将 cdev 添加到系统中去 */
95      ret = cdev_add(&cdev, devno, GPIO_NUM);
96      if (ret < 0)
97      {
98          printk("cdev_add\n");
99          goto err1;
100     }
101     /* GPIO 相关寄存器物理地址映射 */
102     ret = fs4412_gpio_ioremap();
103     if (ret < 0)
104         goto err2;
105     /* GPIO 管脚初始化，设置为输出模式 */
106     fs4412_gpio_io_init();
107
108     printk("gpio init\n");
109
110     return 0;
111 err2:
112     cdev_del(&cdev);
113 err1:
114     unregister_chrdev_region(devno, GPIO_NUM);
115     return ret;
116 }
```

2.5.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱资料-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\fs4412_gpio】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd fs4412_gpio
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c –o gpio
```

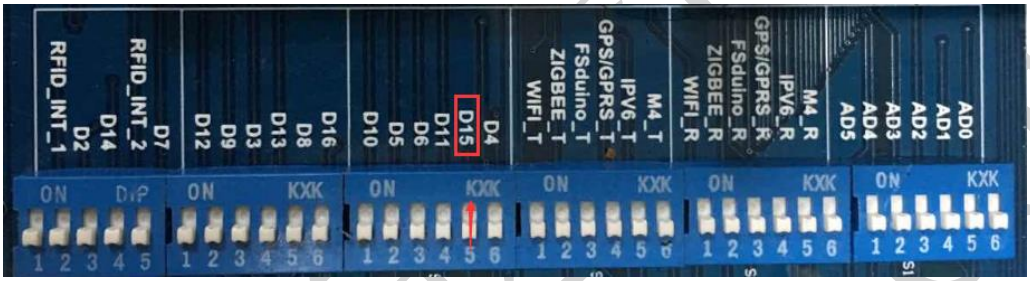
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp gpio /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

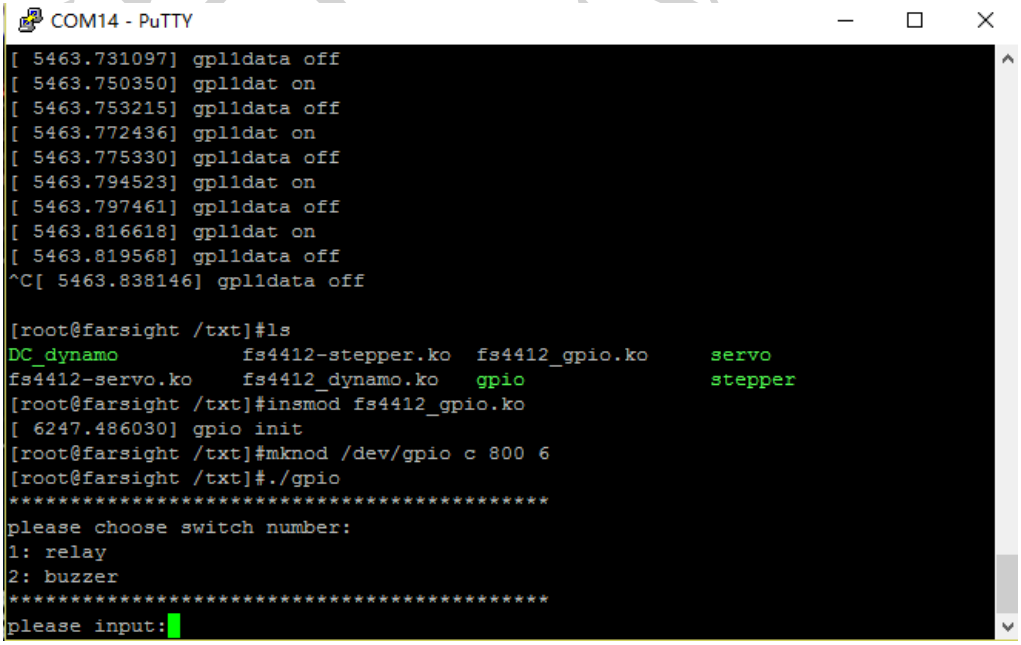
```
insmod  fs4412_gpio.ko
mknod /dev/gpio c 800 6
./gpio
```

（3）将拨码开关中的 D15 拨至 ON，其他拨码全为 OFF，如下图

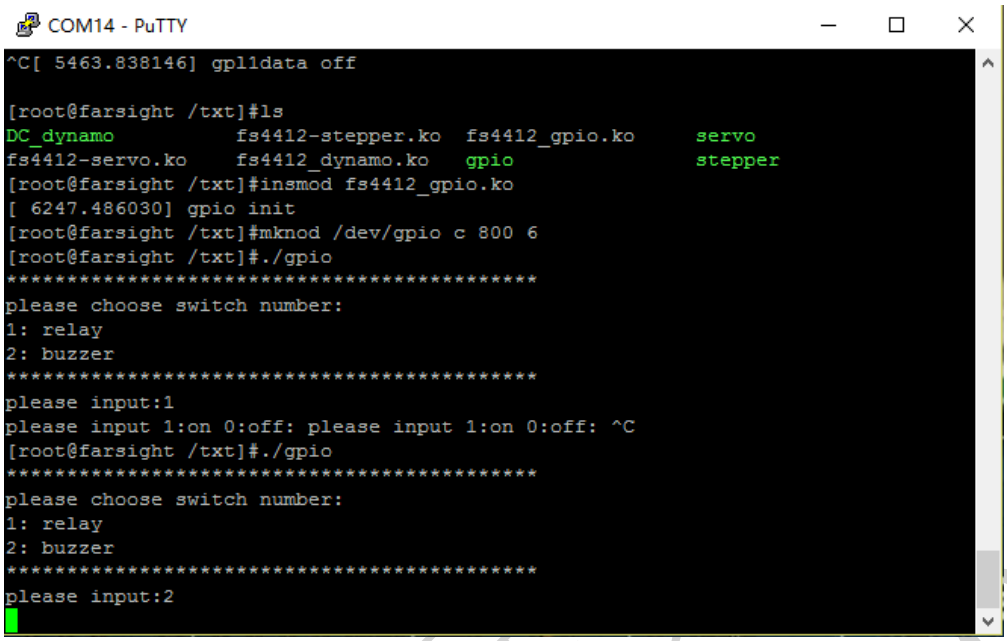


2.5.7 【实验结果】

如果按照上面的步骤正确操作，在 Putty 上会出现如下界面：



这里测试的是蜂鸣器，所以输入 2，可以看到如下界面，同时听到蜂鸣器发出声音：



如果想要关闭蜂鸣器，只需用组合键 Ctrl + C 键。

2.6 酒精传感器驱动实验

2.6.1 【实验目的】

（1） 掌握酒精传感器的工作原理；

（2） 掌握 Linux 下 ADC 驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.6.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

2.6.3 【实验内容】

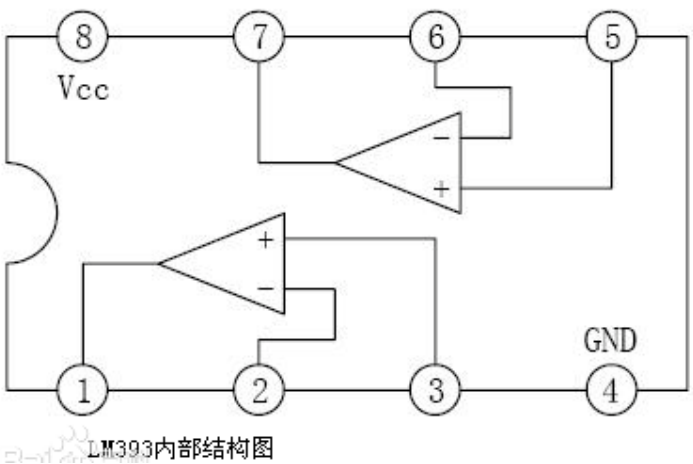
基于 Linux3.0 内核实现驱动酒精传感器采集当前环境中的酒精浓度值。

2.6.4 【实验原理】

LM393 是双电压比较器集成电路。

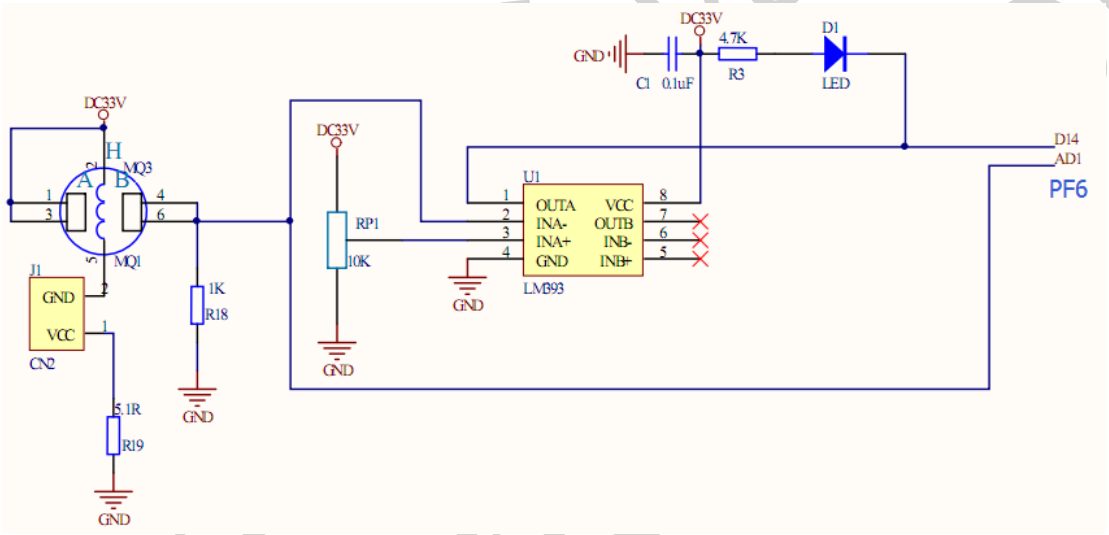
输出负载电阻能衔接在可允许电源电压范围内的任何电源电压上，不受 Vcc 端电压值的限制.此输出能作为一个简单的对地 SPS 开路(当不用负载电阻没被运用)，输出部分的陷电流被可能得到的驱动和器件的β值所限制.当达到极限电流(16mA)时，输出晶体管将退出而且输出电压将很快上升。

LM393 内部采用双列直插 8 脚塑料封装（DIP8）和微形的双列 8 脚塑料封装（SOP8）



LM393内部结构图

LM393 是高增益,宽频带器件，像大多数比较器一样，如果输出端到输入端有寄生电容而产生耦合，则很容易产生振荡。这种现象仅仅出现在当比较器改变状态时，输出电压过渡的间隙，电源加旁路滤波并不能解决这个问题，标准 PC 板的设计对减小输入—输出寄生电容耦合是有助的。减小输入电阻至小于 10K 将减小反馈信号 ,而且增加甚至很小的正反馈量(滞回 1.0~10mV)能导致快速转换,使得不可能产生由于寄生电容引起的振荡，除非利用滞后，否则直接插入 IC(集成电路板 integrated circuit，缩写：IC) 并在引脚上加上电阻将引起输入—输出在很短的转换周期内振荡，如果输入信号是脉冲波形，并且上升和下降时间相当快，则滞回将不需要。



酒精传感器接口

2.6.5 【源码分析】

有些类似的程序这里就不再赘述了，可以参考【直流电机驱动实验】中的【源码分析】，这里只描述重点的代码。

```
1   /* 实现读取 adc 采样值函数 */
    static ssize_t fs4412_adc_read
2   (struct file *file, char *buf, size_t count, loff_t *loff)
3   {
4       int data = 0;
5
```

```
    if (count != 4)
        return -EINVAL;
    /* 设置 adc 通道，及 adc 控制器相关属性 */
    writel(ch, adc_base + FS4412_ADCMUX);
    writel(1 << 0 | 1 << 14 | 0xff << 6 | 0x1 << 16,
           adc_base + FS4412_ADCCON);

    udelay(100);
    /* 读取 adc 数据寄存器中的值 */
    data = readl(adc_base + FS4412_ADCDAT) & 0xfff;
    /* 将读取到的值传送给用户层 */
    if (copy_to_user(buf, &data, sizeof(data)))
        return -EFAULT;

    flags = 0;

    return count;
}
/* 实现对 adc 设备控制函数 */
static long fs4412_adc_ioctl
(struct file *file, unsigned int cmd, unsigned long arg)
{
    switch(cmd)
    {
    case SET_CHANNEL:
        ch = arg;
        writel(arg, adc_base + FS4412_ADCMUX);
        break;
    default:
        printk("invalid command\n");
        break;
    }

    return 0;
}
/* 设备文件操作结构体定义 */
static struct file_operations fs4412_dt_adc_fops =
{
    .owner = THIS_MODULE,
    .open = fs4412_adc_open,
    .release = fs4412_adc_release,
    .read = fs4412_adc_read,
    .unlocked_ioctl = fs4412_adc_ioctl,
```

```
};
/* 设备与设备驱动匹配成功后该函数将会被调用 */
static int fs4412_dt_probe(struct platform_device *pdev)
{
    int ret;
    /* 将主次设备号转换成 dev_t 类型 */
    dev_t devno = MKDEV(adc_major, adc_minor);

    printk("match OK\n");
    /* 初始化等待队列 */
    init_waitqueue_head(&readq);
    /* 获取平台资源 */
    mem_res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    irq_res = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
    if (mem_res == NULL || irq_res == NULL)
    {
        printk("No resource !\n");
        return -ENODEV;
    }
    printk("mem = %x: irq = %d\n", mem_res->start, irq_res->start);

    /* adc 相关寄存器地址映射 */
    adc_base = ioremap(mem_res->start, mem_res->end - mem_res->start);
    if (adc_base == NULL)
    {
        printk("failed to ioremap address reg\n");
        return -EINVAL;
    };

    printk("major = %d, minor = %d, devno = %x\n", adc_major, adc_minor, devno);
    /* 为 adc 设备申请一个或多个设备号 */
    ret = register_chrdev_region(devno, 1, "fs4412-adc");
    if (ret < 0)
    {
        printk("failed register char device region\n");
        goto err2;
    }
    /* cdev 结构体初始化 */
    cdev_init(&cdev, &fs4412_dt_adc_fops);
    cdev.owner = THIS_MODULE;
    /* 将 cdev 结构体加入到系统中 */
    ret = cdev_add(&cdev, devno, 1);
    if (ret < 0)
```

```
92      {
93          printk("failed add device\n");
94          goto err3;
95      }
96      /* 获取时钟信息，开启对应外设时钟 */
97      clk = clk_get(NULL, "adc");
98      clk_enable(clk);
99      return 0;
100 err3:
101     unregister_chrdev_region(devno, 1);
102 err2:
103     free_irq(irq_res->start, NULL);
104 err1:
105     iounmap(adc_base);
106     return ret;
107 }
108
109 /* 定义 adc 设备信息 */
110 #ifdef CONFIG_OF
111 static const struct of_device_id fs4412_dt_of_matches[] =
112 {
113     { .compatible = "fs4412,adc"},
114 };
115 /* 将 fs4412_dt_of_matches 结构输出到用户空间，这样模块加载系统
116  * 在加载模块时就知道了什么模块对应什么硬件设备*/
117 MODULE_DEVICE_TABLE(of, fs4412_dt_of_matches);
118 #endif
119 /* platform 平台驱动结构体定义 */
120 struct platform_driver fs4412_dt_driver =
121 {
122     .driver = {
123         .name = "fs4412-adc",
124         .owner = THIS_MODULE,
125 #ifdef CONFIG_OF
126         .of_match_table = of_match_ptr(fs4412_dt_of_matches),
127 #endif
128     },
129     .probe = fs4412_dt_probe,
130     .remove = fs4412_dt_remove,
131 };
132
```

2.6.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱资料-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\fs4412_adc】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd fs4412_adc
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c –o adc
```

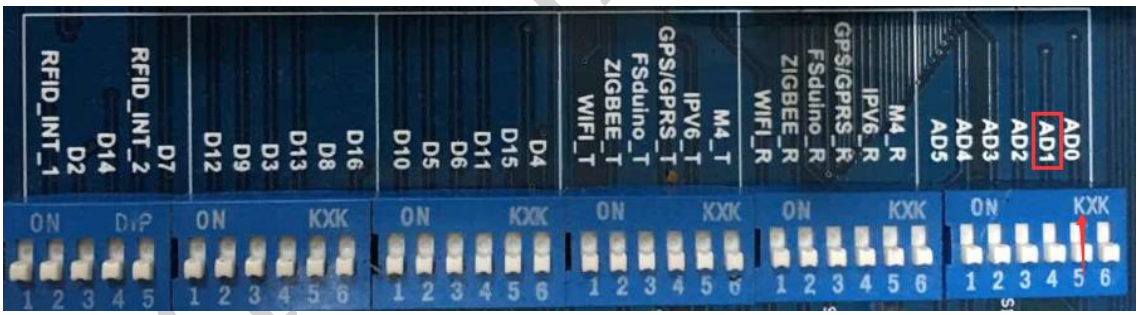
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp adc /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

```
insmod  fs4412_adc.ko
mknod /dev/adc c 800 5
```

（3）将拨码开关中的 AD1 拨至 ON，其他拨码全为 OFF，如下图



2.6.7 【实验结果】

执行./adc

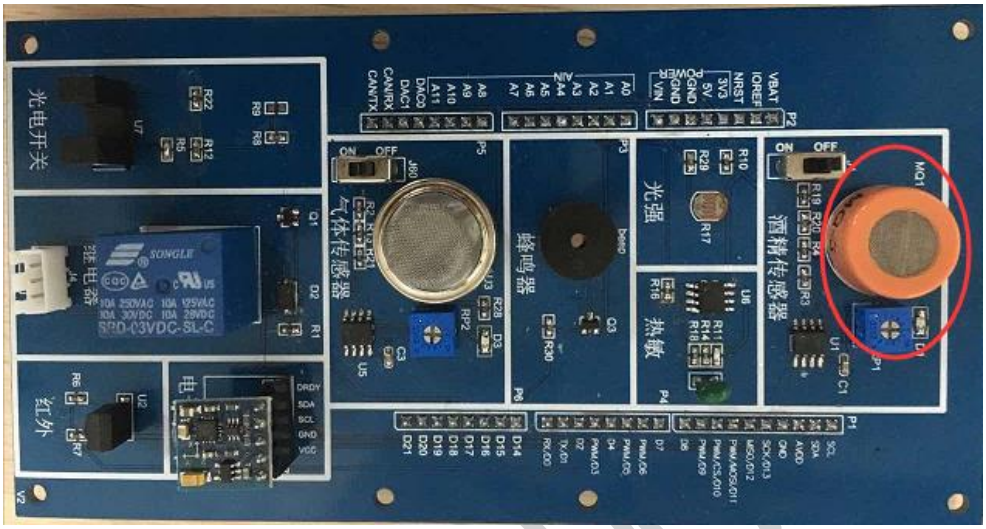
出现如下界面：

```
[root@farsight /txt]#./adc
*****************[  186.048772] fs4412_adc_open:38
**************
please change Sensor type:
A: alcohol sensor
L: light sensor
S: sensitive sensor
G: gas sensor
*******************************
input type:
```

输入大写字母“A”，可以看到不停的显示当前采集到的模拟电压值，用打火机的气体测试，当气体浓度越大，值越大：

```
[root@farsight /txt]#./adc
*****************[ 1627.553175] fs4412_adc_open:38
**************
please change Sensor type:
A: alcohol sensor
L: light sensor
S: sensitive sensor
G: gas sensor
*******************************
input type:A
Vol: 0.00V
Vol: 0.00V
Vol: 0.00V
Vol: 0.00V
Vol: 0.02V
Vol: 0.02V
```

酒精传感器在模块中的位置如下图所示：



2.7 光敏传感器驱动实验

2.7.1 【实验目的】

（1） 掌握光敏传感器的工作原理；

（2） 掌握 Linux 下 ADC 驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.7.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

2.7.3 【实验内容】

基于 Linux3.0 内核实现驱动光敏传感器采集当前环境中的光照强度值。

华清远见 dev.hqyj.com

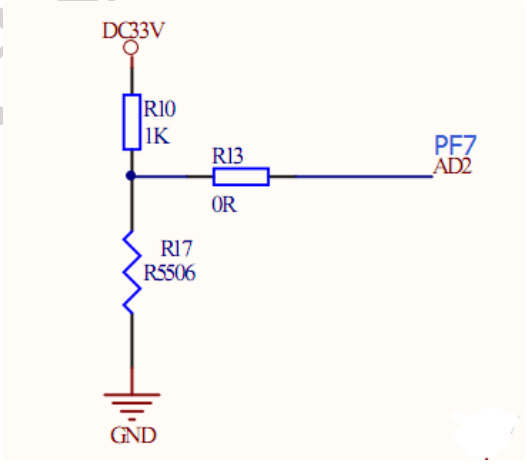
2.7.4 【实验原理】

光敏传感器是最常见的传感器之一，它的种类繁多，主要有：光电管、光电倍增管、光敏电阻、光敏三极管、太阳能电池、红外线传感器、紫外线传感器、光纤式光电传感器、色彩传感器、CCD 和 CMOS 图像传感器等。国内主要厂商有 OTRON 品牌等。光传感器是目前产量最多、应用最广的传感器之一，它在自动控制和非电量电测技术中占有非常重要的地位。最简单的光敏传感器是光敏电阻，当光子冲击接合处就会产生电流。

光传感器是利用光敏元件将光信号转换为电信号的传感器，它的敏感波长在可见光波长附近，包括红外线波长和紫外线波长。光传感器不只局限于对光的探测，它还可以作为探测元件组成其他传感器，对许多非电量进行检测，只要将这些非电量转换为光信号的变化即可。

光敏传感器主要应用于太阳能草坪灯、光控小夜灯、照相机、监控器、光控玩具、声光控开关、摄像头、防盗钱包、光控音乐盒、生日音乐蜡烛、音乐杯、人体感应灯、人体感应开关等电子产品光自动控制领域。

光敏传感器中最简单的电子器件是光敏电阻，它能感应光线的明暗变化，输出微弱的电信号，通过简单电子线路放大处理，可以控制 LED 灯具的自动开关。因此在自动控制、家用电器中得到广泛的应用，对于远程的照明灯具，例如：在电视机中作亮度自动调节，照相机种作自动曝光；另外，在路灯、航标等自动控制电路、卷带自停装置及防盗报警装置中等



华清远见 dev.hqyj.com

光敏传感器接口

2.7.5 【源码分析】

有些类似的程序这里就不再赘述了，可以参考【直流电机驱动实验】中的【源码分析】，这里只描述重点的代码。

```
    /* 实现读取 adc 采样值函数 */
static ssize_t fs4412_adc_read
(struct file *file, char *buf, size_t count, loff_t *loff)
{
    int data = 0;

    if (count != 4)
        return -EINVAL;
    /* 设置 adc 通道，及 adc 控制器相关属性 */
    writel(ch, adc_base + FS4412_ADCMUX);
    writel(1 << 0 | 1 << 14 | 0xff << 6 | 0x1 << 16,
           adc_base + FS4412_ADCCON);

    udelay(100);
    /* 读取 adc 数据寄存器中的值 */
    data = readl(adc_base + FS4412_ADCDAT) & 0xfff;
    /* 将读取到的值传送给用户层 */
    if (copy_to_user(buf, &data, sizeof(data)))
        return -EFAULT;

    flags = 0;

    return count;
}
/* 实现对 adc 设备控制函数 */
static long fs4412_adc_ioctl
(struct file *file, unsigned int cmd, unsigned long arg)
{
    switch(cmd)
    {
    case SET_CHANNEL:
        ch = arg;
        writel(arg, adc_base + FS4412_ADCMUX);
        break;
```

```
35      default:
36          printk("invalid command\n");
37          break;
38      }
39
40      return 0;
41  }
42  /* 设备文件操作结构体定义 */
43  static struct file_operations fs4412_dt_adc_fops =
44  {
45      .owner = THIS_MODULE,
46      .open = fs4412_adc_open,
47      .release = fs4412_adc_release,
48      .read = fs4412_adc_read,
49      .unlocked_ioctl = fs4412_adc_ioctl,
50  };
51  /* 设备与设备驱动匹配成功后该函数将会被调用 */
52  static int fs4412_dt_probe(struct platform_device *pdev)
53  {
54      int ret;
55      /* 将主次设备号转换成 dev_t 类型 */
56      dev_t devno = MKDEV(adc_major, adc_minor);
57
58      printk("match OK\n");
59      /* 初始化等待队列 */
60      init_waitqueue_head(&readq);
61      /* 获取平台资源 */
62      mem_res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
63      irq_res = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
64      if (mem_res == NULL || irq_res == NULL)
65      {
66          printk("No resource !\n");
67          return -ENODEV;
68      }
69      printk("mem = %x: irq = %d\n", mem_res->start, irq_res->start);
70
71      /* adc 相关寄存器地址映射 */
72      adc_base = ioremap(mem_res->start, mem_res->end - mem_res->start);
73      if (adc_base == NULL)
74      {
75          printk("failed to ioremap address reg\n");
76          return -EINVAL;
77      };
```

```
78
79      printk("major = %d, minor = %d, devno = %x\n", adc_major, adc_minor, devno);
80      /* 为 adc 设备申请一个或多个设备号 */
81      ret = register_chrdev_region(devno, 1, "fs4412-adc");
82      if (ret < 0)
83      {
84          printk("failed register char device region\n");
85          goto err2;
86      }
87      /* cdev 结构体初始化 */
88      cdev_init(&cdev, &fs4412_dt_adc_fops);
89      cdev.owner = THIS_MODULE;
90      /* 将 cdev 结构体加入到系统中 */
91      ret = cdev_add(&cdev, devno, 1);
92      if (ret < 0)
93      {
94          printk("failed add device\n");
95          goto err3;
96      }
97      /* 获取时钟信息，开启对应外设时钟 */
98      clk = clk_get(NULL, "adc");
99      clk_enable(clk);
100     return 0;
101 err3:
102     unregister_chrdev_region(devno, 1);
103 err2:
104     free_irq(irq_res->start, NULL);
105 err1:
106     iounmap(adc_base);
107     return ret;
108 }
109
110 /* 定义 adc 设备信息 */
111 #ifdef CONFIG_OF
112 static const struct of_device_id fs4412_dt_of_matches[] =
113 {
114     { .compatible = "fs4412,adc"},
115 };
116 /* 将 fs4412_dt_of_matches 结构输出到用户空间，这样模块加载系统
117  * 在加载模块时就知道了什么模块对应什么硬件设备*/
118 MODULE_DEVICE_TABLE(of, fs4412_dt_of_matches);
119 #endif
120 /* platform 平台驱动结构体定义 */
```

```
121 struct platform_driver fs4412_dt_driver =
122 {
123     .driver = {
124         .name = "fs4412-adc",
125         .owner = THIS_MODULE,
126 #ifdef CONFIG_OF
127         .of_match_table = of_match_ptr(fs4412_dt_of_matches),
128 #endif
129     },
130     .probe = fs4412_dt_probe,
131     .remove = fs4412_dt_remove,
132 };
```

2.7.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱资料-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\fs4412_adc】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd fs4412_adc
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c –o adc
```

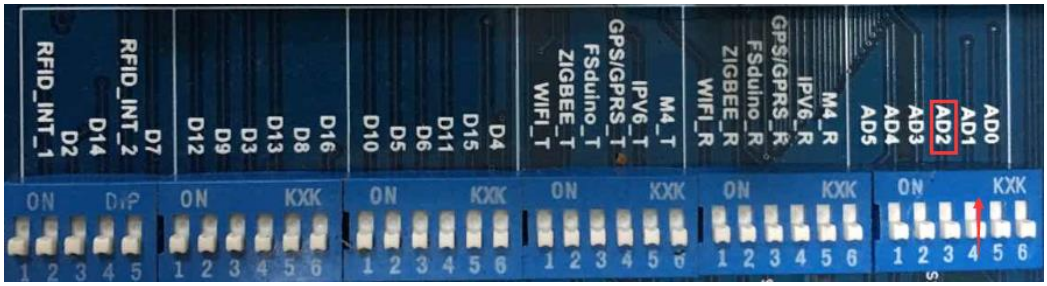
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp adc /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

```
insmod  fs4412_adc.ko
mknod /dev/adc c 800 5
```

（3）将拨码开关中的 AD2 拨至 ON，其他拨码全为 OFF，如下图



2.7.7 【实验结果】

执行./adc

出现如下界面：

```
[root@farsight /txt]#./adc
******************[  186.048772] fs4412_adc_open:38
**************
please change Sensor type:
A: alcohol sensor
L: light sensor
S: sensitive sensor
G: gas sensor
********************************
input type:
```

输入大写字母“L”，可以看到不停的显示当前采集到的模拟电压值，当用手电筒照射光敏传感器时值会变小：

```
[root@farsight /txt]#./adc
******************[  186.048772] fs4412_adc_open:38
**************
please change Sensor type:
A: alcohol sensor
L: light sensor
S: sensitive sensor
G: gas sensor
********************************
input type:L
Vol: 1.28V
Vol: 1.28V
Vol: 1.28V
Vol: 1.28V
Vol: 1.28V
Vol: 1.27V
Vol: 1.27V
```

关照传感器在模块上的位置如下图所示：

2.8 火焰传感器驱动实验

2.8.1 【实验目的】

（1） 掌握热敏传感器的工作原理；

（2） 掌握 Linux 下 ADC 驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.8.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

华清远见 dev.hqyj.com

2.8.3 【实验内容】

基于 Linux3.0 内核实现热敏传感器采集当前环境中的温度信息。

2.8.4 【实验原理】

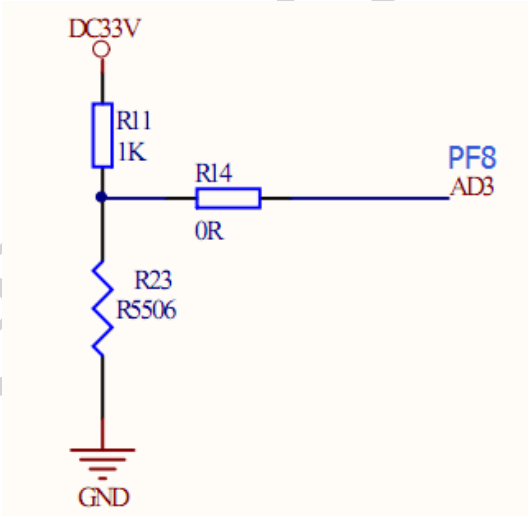
半导体热敏传感器是利用 PN 结的结电阻的温度特性制做的热敏传感器。

PN 结电阻在不同温度下有差别的.根据这个阻值的变化就可以测量环境温度的变化。

N 型半导体 在硅或锗等本征半导体材料中掺入微量的磷、锑、砷等五价元素，就变成了以电子导电为主的半导体，即 N 型半导体。

P 型半导体 在硅或锗等本征半导体材料中掺入微量的硼、铟、镓或铝等三价元素，就就成了以空穴导电为主的半导体，即 P 型半导体。

紧密相连的 P 型半导体和 N 型半导体之间会形成一个空间电荷区称 PN 结。PN 结具有单向导电性,二极管就是利用 PN 结的这个特性做成的。 PN 结的结电阻结电容等参数都是随温度变化的，可以利用这种变化制作温度传感器，即热敏传感器。



热敏传感器接口

2.8.5 【源码分析】

有些类似的程序这里就不再赘述了，可以参考【直流电机驱动实验】中的【源码分析】，这里只描述重点的代码。

```
    /* 实现读取 adc 采样值函数 */
static ssize_t fs4412_adc_read
(struct file *file, char *buf, size_t count, loff_t *loff)
{
    int data = 0;

    if (count != 4)
        return -EINVAL;
    /* 设置 adc 通道，及 adc 控制器相关属性 */
    writel(ch, adc_base + FS4412_ADCMUX);
    writel(1 << 0 | 1 << 14 | 0xff << 6 | 0x1 << 16,
           adc_base + FS4412_ADCCON);

    udelay(100);
    /* 读取 adc 数据寄存器中的值 */
    data = readl(adc_base + FS4412_ADCDAT) & 0xfff;
    /* 将读取到的值传送给用户层 */
    if (copy_to_user(buf, &data, sizeof(data)))
        return -EFAULT;

    flags = 0;

    return count;
}
/* 实现对 adc 设备控制函数 */
static long fs4412_adc_ioctl
(struct file *file, unsigned int cmd, unsigned long arg)
{
    switch(cmd)
    {
    case SET_CHANNEL:
        ch = arg;
        writel(arg, adc_base + FS4412_ADCMUX);
        break;
    default:
        printk("invalid command\n");
        break;
```

```
38      }
39
40      return 0;
41  }
42  /* 设备文件操作结构体定义 */
43  static struct file_operations fs4412_dt_adc_fops =
44  {
45      .owner = THIS_MODULE,
46      .open = fs4412_adc_open,
47      .release = fs4412_adc_release,
48      .read = fs4412_adc_read,
49      .unlocked_ioctl = fs4412_adc_ioctl,
50  };
51  /* 设备与设备驱动匹配成功后该函数将会被调用 */
52  static int fs4412_dt_probe(struct platform_device *pdev)
53  {
54      int ret;
55      /* 将主次设备号转换成 dev_t 类型 */
56      dev_t devno = MKDEV(adc_major, adc_minor);
57
58      printk("match OK\n");
59      /* 初始化等待队列 */
60      init_waitqueue_head(&readq);
61      /* 获取平台资源 */
62      mem_res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
63      irq_res = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
64      if (mem_res == NULL || irq_res == NULL)
65      {
66          printk("No resource !\n");
67          return -ENODEV;
68      }
69      printk("mem = %x: irq = %d\n", mem_res->start, irq_res->start);
70
71      /* adc 相关寄存器地址映射 */
72      adc_base = ioremap(mem_res->start, mem_res->end - mem_res->start);
73      if (adc_base == NULL)
74      {
75          printk("failed to ioremap address reg\n");
76          return -EINVAL;
77      };
78
79      printk("major = %d, minor = %d, devno = %x\n", adc_major, adc_minor, devno);
80      /* 为 adc 设备申请一个或多个设备号 */
```

```
81      ret = register_chrdev_region(devno, 1, "fs4412-adc");
82      if (ret < 0)
83      {
84          printk("failed register char device region\n");
85          goto err2;
86      }
87      /* cdev 结构体初始化 */
88      cdev_init(&cdev, &fs4412_dt_adc_fops);
89      cdev.owner = THIS_MODULE;
90      /* 将 cdev 结构体加入到系统中 */
91      ret = cdev_add(&cdev, devno, 1);
92      if (ret < 0)
93      {
94          printk("failed add device\n");
95          goto err3;
96      }
97      /* 获取时钟信息，开启对应外设时钟 */
98      clk = clk_get(NULL, "adc");
99      clk_enable(clk);
100     return 0;
101 err3:
102     unregister_chrdev_region(devno, 1);
103 err2:
104     free_irq(irq_res->start, NULL);
105 err1:
106     iounmap(adc_base);
107     return ret;
108 }
109
110 /* 定义 adc 设备信息 */
111 #ifdef CONFIG_OF
112 static const struct of_device_id fs4412_dt_of_matches[] =
113 {
114     { .compatible = "fs4412,adc"},
115 };
116 /* 将 fs4412_dt_of_matches 结构输出到用户空间，这样模块加载系统
117  * 在加载模块时就知道了什么模块对应什么硬件设备*/
118 MODULE_DEVICE_TABLE(of, fs4412_dt_of_matches);
119 #endif
120 /* platform 平台驱动结构体定义 */
121 struct platform_driver fs4412_dt_driver =
122 {
123     .driver = {
```

```
124         .name = "fs4412-adc",
125         .owner = THIS_MODULE,
126 #ifdef CONFIG_OF
127         .of_match_table = of_match_ptr(fs4412_dt_of_matches),
128 #endif
129     },
130     .probe = fs4412_dt_probe,
131     .remove = fs4412_dt_remove,
132 };
```

2.8.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱资料-II\外部模块部分\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\fs4412_adc】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd fs4412_adc
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c –o adc
```

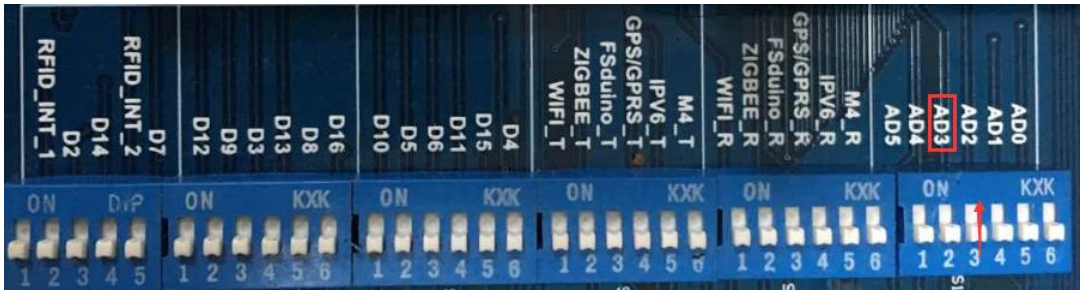
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp adc /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

```
insmod  fs4412_adc.ko
mknod /dev/adc c 800 5
```

（3）将拨码开关中的 AD3 拨至 ON，其他拨码全为 OFF，如下图



2.8.7 【实验结果】

执行./adc

出现如下界面：

```
[root@farsight /txt]#./adc
*****************[  186.048772] fs4412_adc_open:38
**************
please change Sensor type:
A: alcohol sensor
L: light sensor
S: sensitive sensor
G: gas sensor
*******************************
input type:
```

输入大写字母“S”，可以看到不停的显示当前采集到的模拟电压值，当用手指捂住热敏传感器时值会变小：

```
[root@farsight /txt]#./adc
*****************[  687.718654] fs4412_adc_open:38
**************
please change Sensor type:
A: alcohol sensor
L: light sensor
S: sensitive sensor
G: gas sensor
*******************************
input type:S
Vol: 1.27V
Vol: 1.27V
Vol: 1.28V
Vol: 1.28V
Vol: 1.28V
Vol: 1.28V
```

热敏传感器在模块上的位置如下图：

2.9 气体传感器驱动实验

2.9.1 【实验目的】

（1） 掌握气体传感器的工作原理；

（2） 掌握 Linux 下 ADC 驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.9.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

2.9.3 【实验内容】

基于 Linux3.0 内核实现有毒气体传感器通过 ADC 采集当前环境下有毒气体浓度。

华清远见 dev.hqyj.com

2.9.4 【实验原理】

气体传感器是一种将气体的成份、浓度等信息转换成可以被人员、仪器仪表、计算机等利用的信息的装置！气体传感器一般被归为化学传感器的一类，尽管这种归类不一定科学。“气体传感器”包括：半导体气体传感器、电化学气体传感器、催化燃烧式气体传感器、热导式气体传感器、红外线气体传感器、固体电解质气体传感器等。

气体传感器是化学传感器的一大门类。从工作原理、特性分析到测量技术，从所用材料到制造工艺，从检测对象到应用领域，都可以构成独立的分类标准，衍生出一个个纷繁庞杂的分类体系，尤其在分类标准的问题上目前还没有统一，要对其进行严格的系统分类难度颇大。接下来了解一下气体传感器的主要特性：

1、稳定性

稳定性是指传感器在整个工作时间内基本响应的稳定性，取决于零点漂移和区间漂移。零点漂移是指在没有目标气体时，整个工作时间内传感器输出响应的变化。区间漂移是指传感器连续置于目标气体中的输出响应变化，表现为传感器输出信号在工作时间内的降低。理想情况下，一个传感器在连续工作条件下，每年零点漂移小于 10%。

2、灵敏度

灵敏度是指传感器输出变化量与被测输入变化量之比，主要依赖于传感器结构所使用的技术。大多数气体传感器的设计原理都采用生物化学、电化学、物理和光学。首先要考虑的是选择一种敏感技术，它对目标气体的阀限制（TLV-thresh-oldlimitvalue）或最低爆炸限（LEL-lowerexplosivelimit）的百分比的检测要有足够的灵敏性。

3、选择性

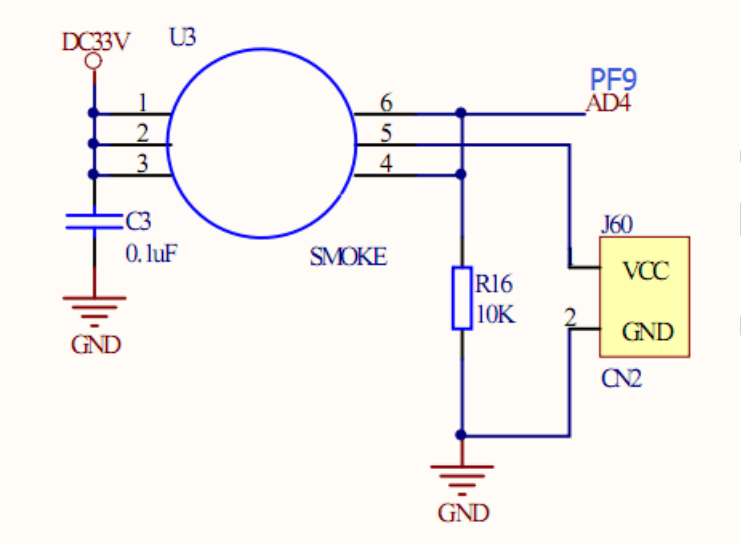
选择性也被称为交叉灵敏度。可以通过测量由某一种浓度的干扰气体所产生的传感器响应来确定。这个响应等价于一定浓度的目标气体所产生的传感器响应。这种特性在追踪多种气体的应用中是非常重要的，

因为交叉灵敏度会降低测量的重复性和可靠性，理想传感器应具有高灵敏度和高选择性。

4、抗腐蚀性

抗腐蚀性是指传感器暴露于高体积分数目标气体中的能力。在气体大量泄漏时，探头应能够承受期望气体体积分数 10~20 倍。在返回正常工作条件下，传感器漂移和零点校正值应尽可能小。

气体传感器的基本特征，即灵敏度、选择性以及稳定性等，主要通过材料的选择来确定。选择适当的材料和开发新材料，使气体传感器的敏感特性达到最优。



气体传感器接口

2.9.5 【源码分析】

有些类似的程序这里就不再赘述了，可以参考【直流电机驱动实验】中的【源码分析】，这里只描述重点的代码。

```
1    /* 实现读取 adc 采样值函数 */
2 static ssize_t fs4412_adc_read
3 (struct file *file, char *buf, size_t count, loff_t *loff)
4 {
5     int data = 0;
6
7     if (count != 4)
```

```
        return -EINVAL;
    /* 设置 adc 通道，及 adc 控制器相关属性 */
    writel(ch, adc_base + FS4412_ADCMUX);
    writel(1 << 0 | 1 << 14 | 0xff << 6 | 0x1 << 16,
           adc_base + FS4412_ADCCON);

    udelay(100);
    /* 读取 adc 数据寄存器中的值 */
    data = readl(adc_base + FS4412_ADCDAT) & 0xfff;
    /* 将读取到的值传送给用户层 */
    if (copy_to_user(buf, &data, sizeof(data)))
        return -EFAULT;

    flags = 0;

    return count;
}
/* 实现对 adc 设备控制函数 */
static long fs4412_adc_ioctl
(struct file *file, unsigned int cmd, unsigned long arg)
{
    switch(cmd)
    {
    case SET_CHANNEL:
        ch = arg;
        writel(arg, adc_base + FS4412_ADCMUX);
        break;
    default:
        printk("invalid command\n");
        break;
    }

    return 0;
}
/* 设备文件操作结构体定义 */
static struct file_operations fs4412_dt_adc_fops =
{
    .owner = THIS_MODULE,
    .open = fs4412_adc_open,
    .release = fs4412_adc_release,
    .read = fs4412_adc_read,
    .unlocked_ioctl = fs4412_adc_ioctl,
};
```

```
51 /* 设备与设备驱动匹配成功后该函数将会被调用 */
52 static int fs4412_dt_probe(struct platform_device *pdev)
53 {
54     int ret;
55     /* 将主次设备号转换成 dev_t 类型 */
56     dev_t devno = MKDEV(adc_major, adc_minor);
57 
58     printk("match OK\n");
59     /* 初始化等待队列 */
60     init_waitqueue_head(&readq);
61     /* 获取平台资源 */
62     mem_res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
63     irq_res = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
64     if (mem_res == NULL || irq_res == NULL)
65     {
66         printk("No resource !\n");
67         return -ENODEV;
68     }
69     printk("mem = %x: irq = %d\n", mem_res->start, irq_res->start);
70 
71     /* adc 相关寄存器地址映射 */
72     adc_base = ioremap(mem_res->start, mem_res->end - mem_res->start);
73     if (adc_base == NULL)
74     {
75         printk("failed to ioremap address reg\n");
76         return -EINVAL;
77     };
78 
79     printk("major = %d, minor = %d, devno = %x\n", adc_major, adc_minor, devno);
80     /* 为 adc 设备申请一个或多个设备号 */
81     ret = register_chrdev_region(devno, 1, "fs4412-adc");
82     if (ret < 0)
83     {
84         printk("failed register char device region\n");
85         goto err2;
86     }
87     /* cdev 结构体初始化 */
88     cdev_init(&cdev, &fs4412_dt_adc_fops);
89     cdev.owner = THIS_MODULE;
90     /* 将 cdev 结构体加入到系统中 */
91     ret = cdev_add(&cdev, devno, 1);
92     if (ret < 0)
93     {
```

```
94          printk("failed add device\n");
95          goto err3;
96      }
97      /* 获取时钟信息，开启对应外设时钟 */
98      clk = clk_get(NULL, "adc");
99      clk_enable(clk);
100     return 0;
101 err3:
102     unregister_chrdev_region(devno, 1);
103 err2:
104     free_irq(irq_res->start, NULL);
105 err1:
106     iounmap(adc_base);
107     return ret;
108 }
109
110 /* 定义 adc 设备信息 */
111 #ifdef CONFIG_OF
112 static const struct of_device_id fs4412_dt_of_matches[] =
113 {
114     { .compatible = "fs4412,adc"},
115 };
116 /* 将 fs4412_dt_of_matches 结构输出到用户空间，这样模块加载系统
117  * 在加载模块时就知道了什么模块对应什么硬件设备*/
118 MODULE_DEVICE_TABLE(of, fs4412_dt_of_matches);
119 #endif
120 /* platform 平台驱动结构体定义 */
121 struct platform_driver fs4412_dt_driver =
122 {
123     .driver = {
124         .name = "fs4412-adc",
125         .owner = THIS_MODULE,
126 #ifdef CONFIG_OF
127         .of_match_table = of_match_ptr(fs4412_dt_of_matches),
128 #endif
129     },
130     .probe = fs4412_dt_probe,
131     .remove = fs4412_dt_remove,
132 };
```

2.9.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将【华清远见-嵌入式 ARM 实验箱资料-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\fs4412_adc】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd fs4412_adc
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc test.c –o adc
```

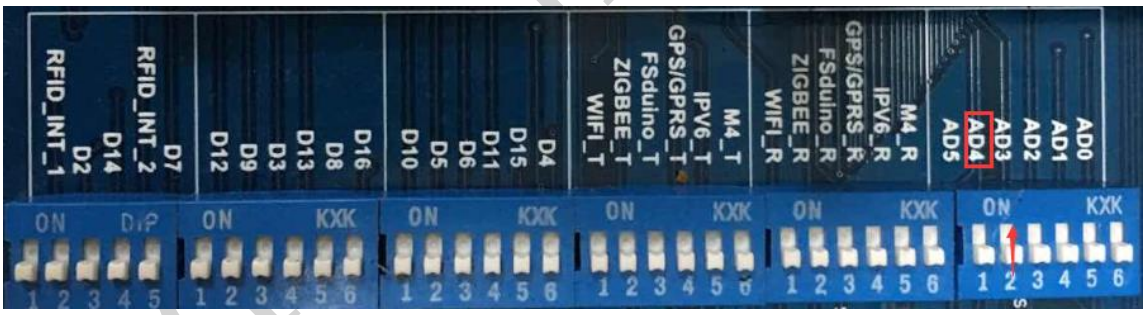
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp adc /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

```
insmod  fs4412_adc.ko
mknod /dev/adc c 800 5
```

（3）将拨码开关中的 AD4 拨至 ON，其他拨码全为 OFF，如下图



2.9.7 【实验结果】

执行./adc

出现如下界面：

```
[root@farsight /txt]#./adc
******************[  186.048772] fs4412_adc_open:38
**************
please change Sensor type:
A: alcohol sensor
L: light sensor
S: sensitive sensor
G: gas sensor
********************************
input type:
```

输入大写字母“G”，可以看到不停的显示当前采集到的模拟电压值，用打火机的气体模拟检测，当检测到气体浓度越大，值越大：

```
[root@farsight /txt]#./adc
******************[ 1155.356165] fs4412_adc_open:38
**************
please change Sensor type:
A: alcohol sensor
L: light sensor
S: sensitive sensor
G: gas sensor
********************************
input type:G
Vol: 0.01V
Vol: 0.03V
Vol: 0.01V
Vol: 0.01V
Vol: 0.02V
```

气体传感器在模块中的位置如下图所示：

2.10 按键扫描数码管显示驱动实验

2.10.1 【实验目的】

（1） 掌握周立功矩阵键盘的工作原理；

（2） 掌握 Linux 下 GPIO 中断驱动开发流程；

（3） 理解字符设备驱动的工作原理；

2.10.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

2.10.3 【实验内容】

基于 Linux3.0 内核实现矩阵键盘和数码管显示的驱动控制。

2.10.4 【实验原理】

在智能仪表中，经常会用到键盘、数码管等外设。因此，一个稳定、占用系统资源少的人机对话通道设计非常重要。传统的键盘与数码管解决方案，由于键盘与数码管是分离的，因而电路连接比较复杂，不管是独立式键盘还是矩阵式键盘，都会浪费微控制器的端口资源，而且都需要人为进行去抖动处理，且抗干扰性差。而数码管部分，不管是静态显示方式还是动态显示方式，在不进行锁存器扩展的前提下。仍然要占用 8 根 I/O 端口线，这将严重浪费系统的端口资源。

ZLG7290B 是广州周立功单片机发展有限公司自行设计的数码管显示驱动及键盘扫描管理芯片。能够

华清远见 dev.hqyj.com

直接驱动 8 位共阴式数码管(或 64 只独立的 LED),同时还可以扫描管理多达 64 只按键。其中有 8 只按键还可以作为功能键使用,就像电脑键盘上的 Ctrl、Shift、Alt 键一样。另外 ZLG7290B 内部还设置有连击计数器,能够使某键按下后不松手而连续有效。采用 I2C 总线方式,与微控制器的接口仅需两根信号线。该芯片为工业级芯片,抗干扰能力强,在工业测控中已有大量应用。

ZLG7290B 可以扫描管理多达 64 个按键,K1～K56 为普通按键,F0～F7 为功能键。普通按键还有连击检测功能。ZLG7290B 内部有 8 个显示缓冲寄存器 DpRam0～DpRam7,(本实验采用的是直接写显示缓冲寄存器,它们地址分别是 10H~17H)它们直接决定数码管显示的内容。ZLG7290B 提供有两种显示控制方式,一种是直接向显存写入字型数据,另一种是通过向命令缓冲寄存器写入控制指令实现自动译码显示。访问这些寄存器需要通过 I2C 总线接口来实现。ZLG7290B 的 I2C 总线器件地址是 70H(写操作)和 71H(读操作)。访问内部寄存器要通过“子地址”来实现。

在这里介绍本实验用到的几个寄存器作用:

系统寄存器 SystemReg(地址:00H)

系统寄存器的第 0 位(LSB)称作 KeyAvi,标志着按键是否有效,0 - 没有按键被按下,1 - 有某个按键被按下。SystemReg 寄存器的其它位暂时没有定义。当按下某个键时,ZLG7290B 的 INT 引脚会产生一个低电平的中断请求信号。当读走键值后,中断信号就会自动撤销。而 KeyAvi 也同时予以反映。正常情况下,微控制器只需要判断 INT 引脚就可以了。通过不断查询 KeyAvi 位也能判断是否有键按下,这样就可以节省微控制器的一根 I/O 口线,但是代价是 I2C 总线处于频繁的活动状态,多消耗电流并且不利于抗干扰。(本节采用外部中断判断 INT 引脚,不读该寄存器)

键值寄存器 Key(地址:01H)

如果某个普通键(图 3.1 中的 K1～K56)被按下,则微控制器可以从键值寄存器 Key 中读取相应的键值 1～56。如果微控制器发现 ZLG7290B 的 INT 引脚产生了中断请求,而从 Key 中读到的键值是 0,则表示按下的可能是功能键。键值寄存器 Key 的值在被读走后自动变成 0。

I2C 总线的定义及特点：

I2C(Inter - Integrated Circuit)总线是一种由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。I2C 总线产生于在 80 年代，最初为音频和视频设备开发，如今主要在服务器管理中使用，其中包括单个组件状态的通信。例如管理员可对各个组件进行查询，以管理系统的配置或掌握组件的功能状态，如电源和系统风扇。可随时监控内存、硬盘、网络、系统温度等多个参数，增加了系统的安全性，方便了管理。

I2C 总线最主要的优点是其简单性和有效性。由于接口直接在组件之上，因此 I2C 总线占用的空间非常小，减少了电路板的空间和芯片管脚的数量，降低了互联成本。总线的长度可高达 25 英尺，并且能够以 10Kbps 的最大传输速率支持 40 个组件。 I2C 总线的另一个优点是，它支持多主控(multimastering)， 其中任何能够进行发送和接收的设备都可以成为主总线。一个主控能够控制信号的传输和时钟频率。当然，在任何时间点上只能有一个主控。它是由数据线 SDA 和时钟 SCL 构成的串行总线，可发送和接收数据。在 CPU 与被控 IC 之间、 IC 与 IC 之间进行双向传送，高速 I2C 总线一般可达 400kbps 以上。

I2C 总线的工作原理：

I2C 总线是由一根数据线(SDA)和一根时钟线(SCL)构成的串行总线，可发送和接收数据。在 CPU 与被控 IC 之间、IC 与 IC 之间进行双向传送，最高传送速率 400kbps。各种被控制模块均并联在这条总线上，每个模块都有唯一的地址标识。在信息的传输过程中，I2C 总线上并接的每一模块电路既可以是主控器，又可以是发送器，这取决于它所要完成的功能。CPU 发出的控制信号分为地址码和控制量两部分，地址码用来选址，即接通需要控制的电路，确定控制的种类；控制量决定该调整的类别(如对比度、亮度等)及需要调整的量。这样，各控制电路虽然挂在同一条总线上，却彼此独立，互不相关。

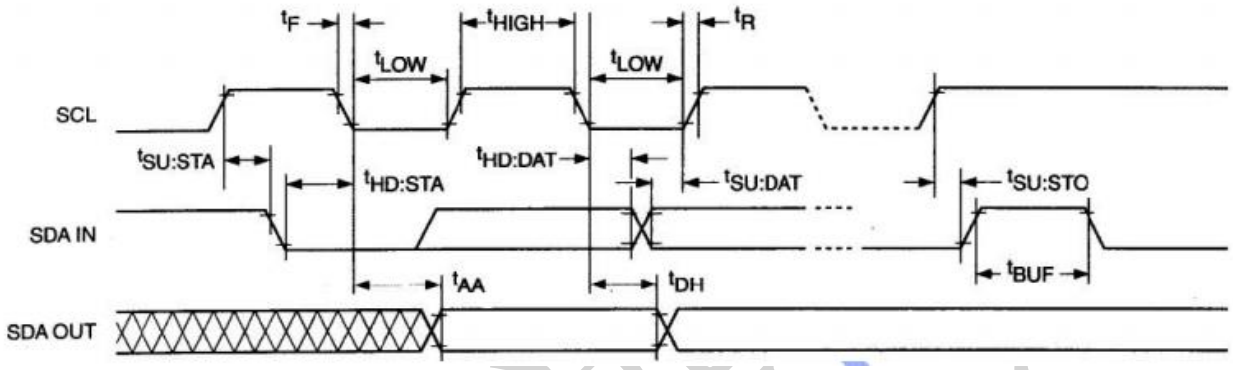
I2C 总线在传送数据过程中共有三种类型信号， 它们分别是：开始信号、结束信号和应答信号。

开始信号： SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据。

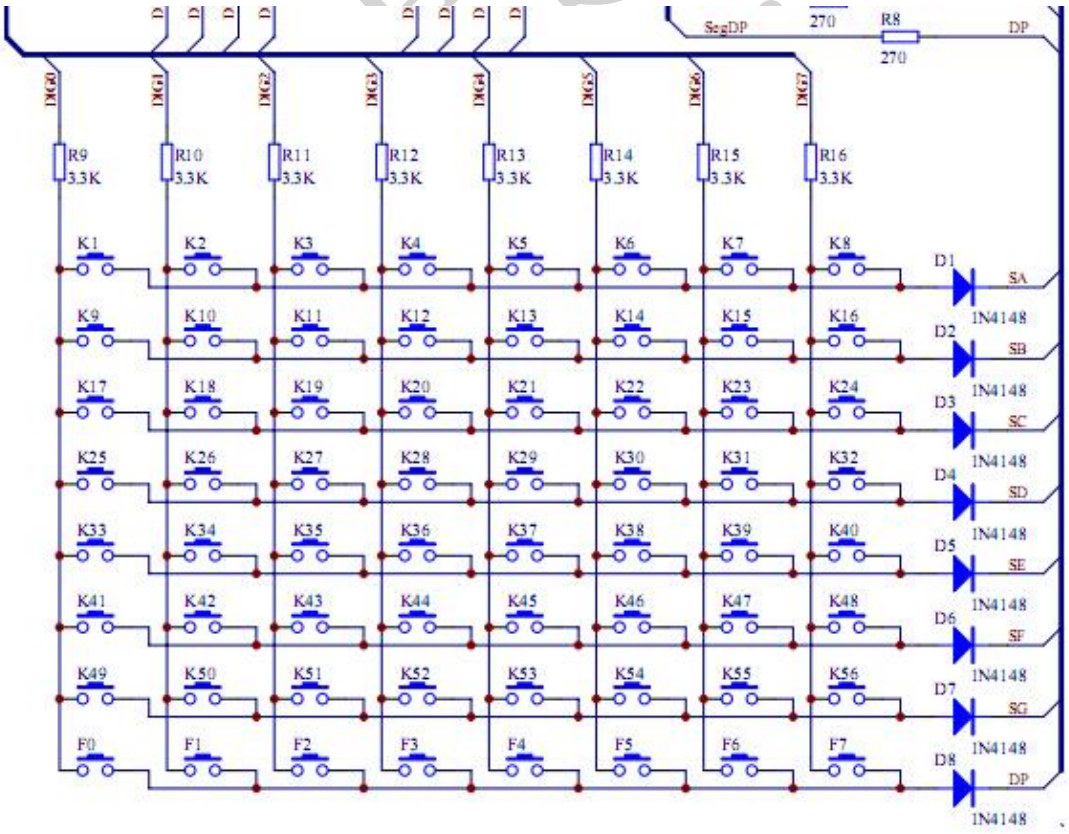
结束信号： SCL 为高电平时， SDA 由低电平向高电平跳变，结束传送数据。

应答信号： 接收数据的 IC 在接收到 8bit 数据后，向发送数据的 IC 发出特定的低电平脉冲，表示已收到数据。CPU 向受控单元发出一个信号后 等待受控单元发出一个应答信号，CPU 接收到应答信号后，根据实际情况作出是否继续传递信号的判断。若未收到应答信号，由判断为受控单元出现故障。

I2C 总线时序图如下：



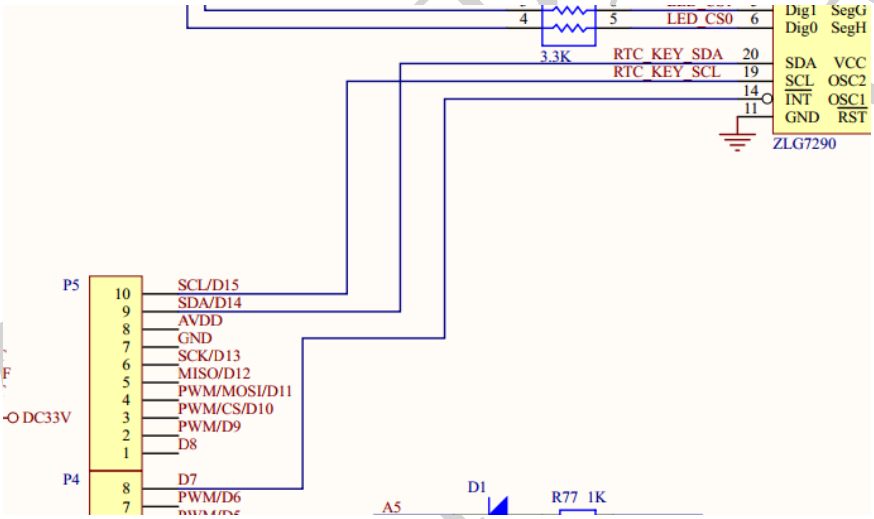
矩阵键盘扫描原理：



根据电路原理图，键盘扫描方法是：行线为输入线，列线为输出线。一开始单片机将列线全部输出高电平，此时读入行线数据，若行线全为低电平则没有键按下，当行线有出现高电平时调用延时程序以此来去除按

键抖动。延时完成后再判断是否有高电平，如果此时读入行线数据还是有高电平，则说明确实有键按下。最后一步确定键值。现在我们以第二行的 K10 键为例，若按下 K2 后我们应该怎么得到这个键值呢？当判断确实有键按下之后，列线轮流输出高电平，根据读入行线的数据可以确定键值。首先，芯片将第一列输出为高电平，其它列输出低电平，此时读取行线的数据全为低电平，说明没有在第一行有键按下；其次，单片机将第二列输出高电平，其它仍为低电平，此时再来读取行线数据，发现行线读到的数据有高电平。根据如上判断，就可以得出是 K10 键按下了。当然这些处理扫描是 ZLG7290 芯片完成的，我们只要读取键值就行了，在这里我们只是理解下键盘扫描的原理。详细原理图请参考 ZLG7290 手册。

ZLG7290 硬件接线如下：



2.10.5 【源码分析】

由于本此实验中用到的周立功的芯片 zlg7290，是通过 I2C 接口跟 CPU 之间建立通信的，当然矩阵键盘还有一根中断线，所以在分析源码之前，有必要先解释一下 Linux 内核中的两个子系统，也就是 I2C 子系统和 input 子系统。理解了这两个子系统的架构，再后面分析源码就很容易理解。

I2C 子系统

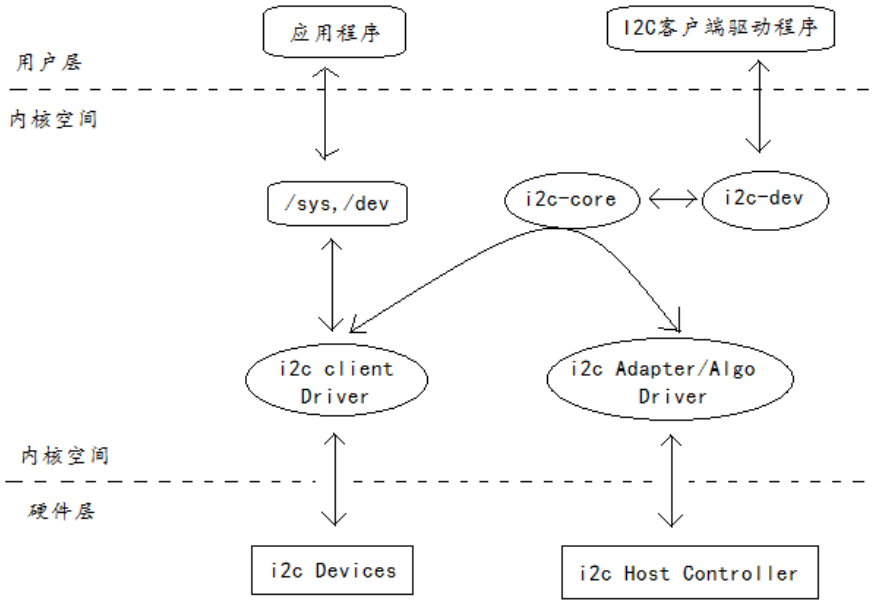
I2C 子系统的三大组成部分：

I2C 核心（ i2c-core ）：i2c 核心提供了 i2c 总线驱动和设备驱动的注册、注销方法，i2c 通信方法上层的、

与具体适配器无关的代码以及探测设备、检测设备地址的上层代码等。

I2C 总线驱动（I2Cadapter/Algo driver）：i2c 总线驱动是对 i2c 硬件体系结构中适配器端的实现，适配器可由 CPU 控制，甚至集成在 CPU 内部（大多数微控制器都这么做）。适配器就是我们经常所说的控制器。经由 i2c 总线驱动的代码，我们可以控制 i2c 适配器以主控方式产生开始位，停止位，读写周期，以及以从设备方式被读写，产生 ACK 等。I2c 总线驱动由 i2c_adapter 和 i2c_algorithm 来描述。

I2C 客户驱动程序（I2Cclient driver）：I2C 客户驱动是对 I2C 硬件体系结构中设备端的实现，设备一般挂接在由 CPU 控制的 I2C 适配器上，通过 I2C 适配器与 CPU 交换数据。I2C 客户驱动程序由 I2C_driver 来描述。



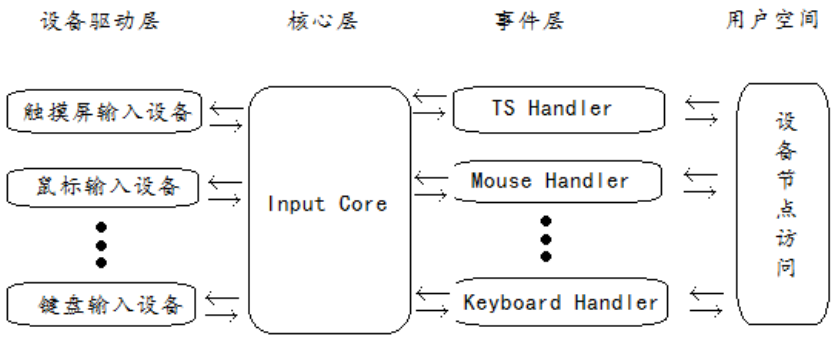
I2c 子系统结构图

Input 子系统

输入设备(如按键,键盘，触摸屏，鼠标等)是典型的字符设备，其一般的工作机制是低层在按键，触摸等 动作发生时产生一个中断(或驱动通过 timer 定时查询)，然后 cpu 通过 SPI，I2C 或者外部存储器总线读取键值，坐标等数据，放一个缓冲区，字符设 备驱动管理该缓冲区，而驱动的 read()接口让用户可以读取键值，坐标等数据。

在 Linux 中，输入子系统是由输入子系统设备驱动层、输入子系统核心层(Input Core)和输入子系统事件处理层(Event Handler)组成。其中设备驱动层提供对硬件各寄存器的读写访问和将底层硬件对用户输入访问的响应转换为标准的输入事件，再通过核心层提交给事件处理层；而核心层对下提供了设备驱动层的编程接口，对上又提供了事件处理层的编程接口；而事件处理层就为我们用户空间的应用程序提供了统一访问设备的接口和驱动层提交来的事件处理。所以这使得我们输入设备的驱动部分不在用关心对设备文件的操作，而是要关心对各硬件寄存器的操作和提交的输入事件。

Input 子系统结构图如下：



Input 子系统结构图

在 Linux 中，Input 设备用 input_dev 结构体描述，定义在 input.h 中。设备的驱动只需要按照如下步骤就可实现了。

（1）在驱动模块加载函数中设置 Input 设备支持 input 子系统的哪些事件；

（2）将 Input 设备注册到 input 子系统中；

（3）在 Input 设备发生输入操作时（如：键盘被按下/抬起），提交所发生的事件及对应的键值/坐标等状态。

大致了解了 i2c 子系统和 Input 子系统之后，就可以分析源码了。

```
1 /* 为了方便起见，将需要操作的设备封装一个结构体 */
2 struct zlg7290
3 {
```

```
    /* 由于该设备通过 i2c 总线通信，所以需要将它定义为一个 i2c 从设备 */
    struct i2c_client *client;
    /* 定义工作队列 */
    struct delayed_work work;
    /* 定义 input device */
    struct input_dev *key;
    /* 定义字符设备结构体 */
    struct cdev cdev;
};

struct zlg7290  *zlg7290;

/* 定义 linux 内核标准键值 */
unsigned int key_value[65] =
{
    0,
    KEY_D, KEY_NUMERIC_POUND, KEY_0, KEY_NUMERIC_STAR,  5,  6,  7,  8,
    KEY_C, KEY_9, KEY_8, KEY_7, 13, 14, 15, 16,
    KEY_B, KEY_6, KEY_5, KEY_4, 21, 22, 23, 24,
    KEY_A, KEY_3, KEY_2, KEY_1, 29, 30, 31, 32,
    33, 34, 35, 36, 37, 38, 39, 40,
    41, 42, 43, 44, 45, 46, 47, 48,
    49, 50, 51, 52, 53, 54, 55, 56,
    57, 58, 59, 60, 61, 62, 63, 64,
};

/* 通过 i2c 方式向 zlg7290 从机写入数据 */
static int zlg7290_hw_write
(struct  zlg7290 *ctr_zlg7290,  int len, size_t *retlen, char *buf)
{
    struct i2c_client *client = ctr_zlg7290->client;
    int ret;
    /* 定义传输数据包 */
    struct i2c_msg msg[] =
    {
        { client->addr, 0, len, buf},   /*the buf contains register address*/
    };
    /* 通过 i2c 传输前面定义的数据包 msg */
    ret = i2c_transfer(client->adapter, msg, 1);
    if (ret < 0)
    {
        printk("ret=%d,addr=%x\n", ret, client->addr);
        dev_err(&client->dev, "i2c read error/n");
```

```
47          return -EIO;
48      }
49      *retlen = len;
50      return 0;
51  }
52
53  /* 通过 I2C 方式从 zlg7290 从机中读取数据 */
54  static int zlg7290_hw_read
55  (struct zlg7290 *ctr_zlg7290 , int len, size_t *retlen, char *buf)
56  {
57      struct i2c_client *client = ctr_zlg7290->client;
58      int ret;
59      /* 定义 i2c 发送的数据包 i2c_msg */
60      struct i2c_msg msg[] =
61      {
62          /* 第一个元素是从机地址，第二个元素是 flag，
63           * 第三个元素是字节长度，第四个元素是存放读写数据的 buf*/
64          { client->addr, 0, len, buf},
65          { client->addr, I2C_M_RD, len, buf },
66      };
67      /* 驱动通过该函数将 msg 数据包传送出去，读数据或者写数据 */
68      ret = i2c_transfer(client->adapter, msg, 2);
69      if (ret < 0)
70      {
71          printk("ret=%d,addr=%x\n", ret, client->addr);
72          dev_err(&client->dev, "i2c read error/n");
73          return -EIO;
74      }
75      *retlen = len;
76      return 0;
77  }
78
79  /* 工作队列处理函数 */
80  static void zlg7290_work(struct work_struct *work)
81  {
82      /* 通过 zlg7290 结构体中的一个成员获取整个结构体的指针 */
83      struct zlg7290 *ctr_zlg7290 = container_of(work, struct zlg7290, work.work);
84      unsigned char val = 0;
85      size_t len;
86      unsigned char status = 0;
87      /* 从 zlg7290 这个从设备中读取数据，数据存放在 status 地址 */
88      zlg7290_hw_read(ctr_zlg7290, 1, &len, &status);
89      if(status & 0x1)
```

```
90      {
91          val = 1;
92          zlg7290_hw_read(ctr_zlg7290, 1, &len, &val);
93          if (val == 0)
94          {
95              val = 3;
96              zlg7290_hw_read(ctr_zlg7290, 1, &len, &val);
97              if (val == 0 || val == 0xFF)
98                  goto out;
99          }
100         if (val > 56)
101         {
102             switch (val)
103             {
104             case 0xFE:
105                 val = 57;
106                 break;
107             case 0xFD:
108                 val = 58;
109                 break;
110             case 0xFB:
111                 val = 59;
112                 break;
113             case 0xF7:
114                 val = 60;
115                 break;
116             case 0xEF:
117                 val = 61;
118                 break;
119             case 0xDF:
120                 val = 62;
121                 break;
122             case 0xBF:
123                 val = 63;
124                 break;
125             case 0x7F:
126                 val = 64;
127                 break;
128             }
129         }
130         /* 如果有按键按下，通过 input 上报键值,第二个参数是键值，第三个参数是 event 值 */
131         input_report_key(ctr_zlg7290->key, key_value[val], 1);
132         input_report_key(ctr_zlg7290->key, key_value[val], 0);
```

```
133         input_sync(ctr_zlg7290->key);
134     }
135 out:
136     /* 延时工作队列每隔 1s 中读取一次从设备，看是否有按键按下 */
137     schedule_delayed_work(&zlg7290->work, HZ / 5);
138 }
139 
140 /* 设备控制函数实现，当用户层调用 ioctl 函数时，该函数会被调用 */
141 static long zlg7290_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
142 {
143     unsigned char buf[8] = {0};
144     ssize_t len = 0;
145     unsigned char val[2] = {0};
146     unsigned char reg[8] = {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17};
147     int i = 0;
148     switch (cmd)
149     {
150         /* 当检测到用户层传递命令 SET_VAL 时，从用户层获取需要往数码管显示的数据 */
151     case SET_VAL:
152         if (copy_from_user(buf, (void *)arg, 8))
153             return -EFAULT;
154         for(i = 0; i < 8; i++)
155         {
156             val[0] = reg[i];
157             /* 将需要显示的数据，匹配成数码管对应的显示值 */
158             switch(buf[i])
159             {
160             case '0':
161                 val[1] = 0xfc;
162                 break;
163             case '1':
164                 val[1] = 0x0c;
165                 break;
166             case '2':
167                 val[1] = 0xda;
168                 break;
169             case '3':
170                 val[1] = 0xf2;
171                 break;
172             case '4':
173                 val[1] = 0x66;
174                 break;
175             case '5':
```

```
176                 val[1] = 0xb6;
177                 break;
178             case '6':
179                 val[1] = 0xbe;
180                 break;
181             case '7':
182                 val[1] = 0xe0;
183                 break;
184             case '8':
185                 val[1] = 0xfe;
186                 break;
187             case '9':
188                 val[1] = 0xf6;
189                 break;
190             case 'a':
191             case 'A':
192                 val[1] = 0xee;
193                 break;
194             case 'b':
195             case 'B':
196                 val[1] = 0x3e;
197                 break;
198             case 'c':
199             case 'C':
200                 val[1] = 0x9c;
201                 break;
202             case 'd':
203             case 'D':
204                 val[1] = 0x7a;
205                 break;
206             case 'e':
207             case 'E':
208                 val[1] = 0x9e;
209                 break;
210             case 'f':
211             case 'F':
212                 val[1] = 0x8e;
213                 break;
214             case ' ':
215                 val[1] = 0x00;
216                 break;
217             default:
218                 val[1] = 0x00;
```

```
219                 break;
220             }
221             msleep(10);
222             /* 将匹配好的数据写入 zlg7290，这样就可以在数码管上显示出来了 */
223             zlg7290_hw_write(zlg7290, 2, &len, val);
224         }
225         break;
226     }
227     return 0;
228 }
229 
230 /* 定义操作字符设备文件结构体，主要操作有打开，关闭，控制 */
231 static struct file_operations zlg7290_led_fops =
232 {
233     .owner = THIS_MODULE,
234     .open = zlg7290_open,
235     .release = zlg7290_release,
236     .unlocked_ioctl = zlg7290_ioctl,
237 };
238 
239 /* 字符设备初始化函数 */
240 static int register_led(void)
241 {
242     int ret;
243     /* 将主次设备号转换成 dev_t 类型 */
244     dev_t devno = MKDEV(led_major, led_minor);
245     /* 为该设备申请一个或几个设备号，这里的第三个参数"led"，
246      * 只是给该设备起名，无实际意义 */
247     ret = register_chrdev_region(devno, 1, "led");
248     if (ret < 0)
249     {
250         printk("Failed: register_chrdev_region\n");
251         return -1;
252     }
253     /* 字符设备结构体初始化 */
254     cdev_init(&zlg7290->cdev, &zlg7290_led_fops);
255     zlg7290->cdev.owner = THIS_MODULE;
256     /* 将 cdev 添加到系统中 */
257     ret = cdev_add(&zlg7290->cdev, devno, 1);
258     if (ret < 0)
259     {
260         unregister_chrdev_region(devno, 1);
261         printk("Failed: cdev_add\n");
```

```
262         return -1;
263     }
264     return 0;
265 }
266 
267 /* 卸载驱动时将之前申请的设备号注销掉 */
268 static int unregister_led(void)
269 {
270     dev_t devno = MKDEV(led_major, led_minor);
271     cdev_del(&zlg7290->cdev);
272     unregister_chrdev_region(devno, 1);
273     return 0;
274 }
275 
276 /* 设备与驱动匹配成功后会调用该函数 */
277 static int zlg7290_probe(struct i2c_client *client, const struct i2c_device_id *id)
278 {
279     struct input_dev *key_dev;
280     int ret = 0;
281     int i = 0;
282     int err = -EINVAL;
283     /* 为 zlg7290 设备分配地址空间 */
284     if (!(zlg7290 = kzalloc(sizeof(struct zlg7290), GFP_KERNEL)))
285         return -ENOMEM;
286     printk("irq = %d\n", client->irq);
287     /* 为 input 设备分配地址空间 */
288     zlg7290->key = input_allocate_device();
289     if (zlg7290->key == NULL)
290     {
291         kfree(zlg7290);
292         return -ENOMEM;
293     }
294     /* 将 zlg7290 设备的相关属性配置到 input 设备中 */
295     key_dev = zlg7290->key;
296     key_dev->name = "zlg7290";
297     key_dev->id.bustype = BUS_HOST;
298     key_dev->id.vendor = 0x0001;
299     key_dev->id.product =  0x0001;
300     key_dev->id.version = 0x0001;
301     key_dev->evbit[0] = BIT_MASK(EV_KEY);
302     for(i = 1; i <= 64; i++)
303     {
304         key_dev->keybit[BIT_WORD(key_value[i])] |= BIT_MASK(key_value[i]);
```

```
305     }
306     /* 注册 input 设备 */
307     ret = input_register_device(key_dev);
308     if (ret < 0)
309     {
310         printk("Failed to register input device\n");
311         goto err1;
312     }
313     printk("zlg7290 probe\n");
314     /* 检测 i2c 控制器是否支持我们的需求，如果返回 1 则支持，0 则不支持 */
315     if (!i2c_check_functionality(client->adapter, I2C_FUNC_I2C))
316     {
317         err = -ENODEV;
318         return err;
319     }
320
321     zlg7290->client = client;
322     /* 将自定义的设备结构 dev 赋给设备驱动 client 的私有指针 */
323     i2c_set_clientdata(client, zlg7290);
324     /* 初始化延时工作队列 */
325     INIT_DELAYED_WORK(&zlg7290->work, zlg7290_work);
326     /* 经过一段时间（HZ/5）的延时再去执行工作，调用下面的函数 */
327     schedule_delayed_work(&zlg7290->work, HZ / 5);
328     /* 调用字符设备初始化函数 */
329     ret = register_led();
330     if (ret < 0)
331         goto err1;
332
333     return 0;
334
335 err1:
336     input_free_device(key_dev);
337     kfree(zlg7290);
338     return ret;
339 }
340 /* 定义 i2c 设备信息 */
341 static const struct i2c_device_id zlg7290_id[] =
342 {
343     {ZLG7290_NAME, 0 },
344     { }
345 };
346 /* 将 zlg7290 结构输出到用户空间，这样模块加载系统在加载模块时，
347  * 就知道了什么模块对应什么硬件设备*/
```

```
348 MODULE_DEVICE_TABLE(i2c, zlg7290_id);
349
350 /* 定义 i2c 设备驱动结构体 */
351 static struct i2c_driver  zlg7290_driver =
352 {
353     .probe      = zlg7290_probe,
354     .remove     = zlg7290_remove,
355     .id_table   = zlg7290_id,
356     .driver = {
357         .name   = ZLG7290_NAME,
358         .owner  = THIS_MODULE,
359     },
360 };
```

2.10.6 【实验步骤】

（1）驱动文件拷贝编译，将光盘【华清远见-嵌入式 ARM 实验箱资料-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\zlg7290】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd zlg7290
make
```

将编译生成的.ko 文件拷贝到/source/rootfs 目录下

```
sudo cp *.ko /source/rootfs
```

编译应用程序

```
arm-none-linux-gnueabi-gcc led_test.c –o 7led
arm-none-linux-gnueabi-gcc keyboard_test.c –o keyboard_test
```

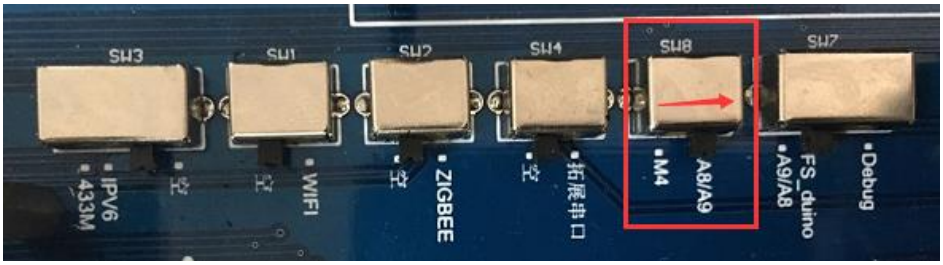
将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp 7led keyboard_test /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

```
insmod    zlg7290.ko
mknod /dev/zlg7290 c 800 1
```

（3）该实验只需要注意 SW8 拨码拨至 A8/A9，因为在硬件设计时，将 SW8 作为 MCU 和 CUP 的 i2c 主机切换开关，如下图



2.10.7 【实验结果】

执行./7led

可以看到在 Arduino 模块数码管上显示如下信息：

在 Putty 中也能看到如下打印信息：

```
00 08 49
00 08 50
00 08 51
00 08 53
00 08 54
00 08 55
00 08 56
00 08 58
00 08 59
00 09 00
00 09 01
```

在 Putty 中使用组合键 ctrl + c 停止该应用程序的执行。

执行./keyboard_test

当按下按键时，在 Putty 中也能看到打印出对应的键值（这里的键值是 Linux 内核中的标准键值，所以跟模块上面的标记不一致，不过这个可以自行匹配跟标记一致的键值）。

```
[root@farsight /txt]#./keyboard_test
user:key_val = 2 1
user:key_val = 2 0
user:key_val = 3 1
user:key_val = 3 0
user:key_val = 4 1
user:key_val = 4 0
user:key_val = 5 1
user:key_val = 5 0
user:key_val = 6 1
user:key_val = 6 0
user:key_val = 7 1
user:key_val = 7 0
user:key_val = 8 1
user:key_val = 8 0
user:key_val = 9 1
user:key_val = 9 0
```

2.11 温度传感器驱动实验

2.11.1 【实验目的】

掌握温度传感器 LM75 的使用方法

掌握 I2C 设备驱动编写思路

2.11.2 【实验环境】

（1） Vmware Workstations 虚拟机；

（2） Cortex-A9 FS4412 开发平台；

（3） Linux3.0 内核环境；

（4） u-boot-2010.03 版本；

（5） NFS 网络文件系统；

2.11.3 【实验内容】

通过 LM75 温度传感器采集当前环境中的温度值。

2.11.4 【实验原理】

LM75A 是一个使用了内置带隙温度传感器和∑-△模数转换技术的温度-数字转换器。它也是一个温度检测器，可提供一个过热检测输出。 LM75A 包含许多数据寄存器：配置寄存器（ Conf），用来存储器件的某些配置，如器件的工作模式、 OS 工作模式、 OS 极性和 OS 故障队列等（在功能描述一节中有详细描述）；温度寄存器（ Temp），用来存储读取的数字温度；设定点寄存器（ Tos & Thyst），用来存储可编程的过热关断和滞后限制，器件通过 2 线的串行 I_2C 总线接口与控制器通信。 LM75A 还包含一个开漏输出（ OS），当温度超过编程限制的值时该输出有效。 LM75A 有 3 个可选的逻辑地址管脚，使得同一总线上可同时连接 8 个器件而不发生地址冲突。

LM75A 可配置成不同的工作条件。 它可设置成在正常工作模式下周期性地对环境温度进行监控或进入关断模式来将器件功耗降至最低。OS 输出有 2 种可选的工作模式：OS 比较器模式和 OS 中断模式。OS 输出可选择高电平或低电平有效。故障队列和设定点限制可编程，为了激活 OS 输出，故障队列定义了许多连续的故障。

温度寄存器通常存放着一个 11 位的二进制数的补码，用来实现 0.125℃的精度。这个高精度在需要精确地测量温度偏移或超出限制范围的应用中非常有用。

正常工作模式下 ,当器件上电时，OS 工作在比较器模式 ,温度阈值为 80℃ ,滞后 75℃ ,这时，LM75A 就可用作一个具有以上预定义温度设定点的独立的温度控制器。

特性：

- 器件可以完全取代工业标准的 LM75，并提供了良好的温度精度（ 0.125℃），单个器件的电源可超出 2.8V～5.5V 的范围。
- 小型 8 脚封装： SO8 和 TSSOP8。
- 具有 I2C 总线接口，同一总线上可连接多达 8 个器件。
- 电源电压范围： 2.8V～5.5V。
- 温度范围：－55℃～ +125℃。
- 提供 0.125℃的精度的 11 位的 ADC。
- 温度精度：－25℃～＋100℃时为±2℃；－55℃～＋125℃时为±3℃
- 可编程温度阈值和滞后设定点。
- 为了降低功耗，关断模式下消耗的电流仅为 3.5uA
- 上电时器件可用作一个独立的温度控制器

下面是 LM75 的硬件原理图：

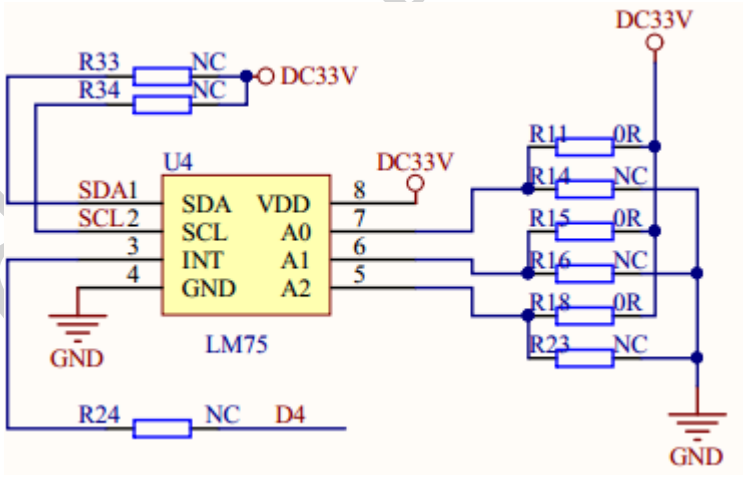


图 LM75 硬件原理图

2.11.5 【实验源码】

C++ Code

```
   #include <stdio.h>
#include <linux/types.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <errno.h>
#define I2C_RETRIES 0x0701
#define I2C_TIMEOUT 0x0702
#define I2C_RDWR 0x0707
/*********定义 struct i2c_rdwr_ioctl_data 和 struct
 * i2c_msg，要和内核一致*******/

struct i2c_msg
{
    unsigned short addr;
    unsigned short flags;
#define I2C_M_TEN 0x0010
#define I2C_M_RD 0x0001
    unsigned short len;
    unsigned char *buf;
};
struct i2c_rdwr_ioctl_data
{
    struct i2c_msg *msgs;
    int nmsgs;
    /* nmsgs 这个数量决定了有多少开始信号，对于“单开始时序”，取 1*/
};
/***********主程序***********/
int main()
{
    int fd, ret;
    short temp_val = 0;
    struct i2c_rdwr_ioctl_data lm75_data;
    fd = open("/dev/i2c-1", O_RDWR);
    if(fd < 0)
    {
        perror("open error");
    }
```

```
40      lm75_data.nmsgs = 2;
41      /*
42       *          *因为操作时序中，最多是用到 2 个开始信号（字节读操作中），所以此将
43       *                  *lm75_data.nmsgs 配置为 2
44       **/
45      lm75_data.msgs = (struct i2c_msg *)malloc(lm75_data.nmsgs * sizeof(struct i2c_msg));
46      if(!lm75_data.msgs)
47      {
48          perror("malloc error");
49          exit(1);
50      }
51      ioctl(fd, I2C_TIMEOUT, 1); /*超时时间*/
52      ioctl(fd, I2C_RETRIES, 2); /*重复次数*/
53      sleep(1);
54      /******read
55       * data
56       * from
57       * lm75*******/
58      while(1)
59      {
60          lm75_data.nmsgs = 2;
61          (lm75_data.msgs[0]).len = 1; //lm75 目标数据的地址
62          (lm75_data.msgs[0]).addr = 0x4f; // lm75 设备地址
63          (lm75_data.msgs[0]).flags = 0; //write
64          (lm75_data.msgs[0]).buf = (unsigned char *)malloc(2);
65          (lm75_data.msgs[0]).buf[0] = 0x0; //lm75 数据地址
66          (lm75_data.msgs[1]).len = 2; //读出的数据
67          (lm75_data.msgs[1]).addr = 0x4f; // lm75 设备地址
68          (lm75_data.msgs[1]).flags = I2C_M_RD; //read
69          (lm75_data.msgs[1]).buf = (unsigned char *)malloc(2); //存放返回值的地址。
70          (lm75_data.msgs[1]).buf[0] = 0; //初始化读缓冲
71          (lm75_data.msgs[1]).buf[1] = 0; //初始化读缓冲
72          ret = ioctl(fd, I2C_RDWR, (unsigned long)&lm75_data);
73          if(ret < 0)
74          {
75              perror("ioctl error2");
76          }
77
78          temp_val = (lm75_data.msgs[1]).buf[0] << 8 | (lm75_data.msgs[1]).buf[1];
79
80          if(temp_val >> 15)
81              temp_val = (~(temp_val - 0x80) >> 7);
82          else
```

```
            temp_val = temp_val >> 7;

        printf("temp =  %01f\n", (float)temp_val / 2);

        sleep(1);
    }
    /***打印读出的值，没错的话，就应该是前面写的 0x58 了***/
    close(fd);
    return 0;
}
```

2.11.6 【实验步骤】

前期的辅助步骤请参考 2.1.6 章节

（1）驱动文件拷贝编译，将光盘【华清远见-嵌入式 ARM 实验箱资料-II\Arduino\程序源码\Linux3.0 内核驱动源码\V1.0_3.0\complete\temprature】中的文件全部拷贝到虚拟机的 workdir 工作目录中。进入文件夹

```
cd temprature
```

编译应用程序

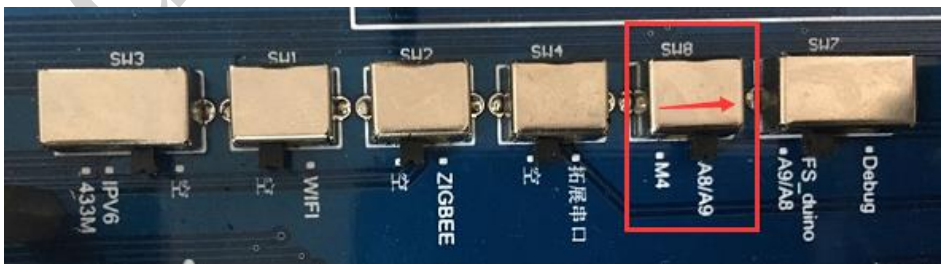
```
arm-none-linux-gnueabi-gcc temp_app_mode.c –o temp
```

将生成的可执行程序，拷贝到/source/rootfs 目录下

```
sudo cp temp /source/rootfs
```

（2）加载驱动，创建设备文件，执行应用程序

（3）该实验只需要注意 SW8 拨码拨至 A8/A9，因为在硬件设计时，将 SW8 作为 MCU 和 CUP 的 i2c 主机切换开关，如下图



（11）./temp

2.11.7 【实验结果】

```
[root@farsight fs4412_arduino_driver]# ./temp
temp =  30.500000
temp =  30.500000
temp =  30.500000
temp =  31.000000
temp =  31.500000
temp =  32.000000
temp =  32.500000
temp =  32.000000
^C
```

温度传感器在模块上的位置如下：

dev.farsight.com

第 3 章　基于 Android 系统的 Arduino 设备开发例程

注意：以下所有的实验，都是基于 Android5.0 系统，u-boot 版本为 2010.03。镜像存放的位置为：光盘【华清远见-嵌入式 ARM 实验箱资料-I-FS4412\烧写镜像\Android 烧写镜像】、

3.1　实验应用平台介绍

Android 相关实验是使用 Android studio 工具进行相关开发的，Android studio 是谷歌公司基于 intelliJ IDEA 开发的一套集成开发环境，用于 android 开发和调试。

NDK（Native Development Kit）,使我们在 Android 中进行 JNI 编程的工具。

Android studio 的环境搭建及使用例程，详见《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

3.2　程序的介绍

3.2.1　【展示目的】

（1）　掌握定义需要的适配器

（2）　掌握 ListView 的使用方法

（3）　掌握 Viewpager 和 Fragment 的使用方法

（4）　掌握异步处理机制 Handler 的用法

3.2.2　【源码分析】

实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4】。用户可以将其解压到电脑任意位置，这里将程序解压到 E:\Administrator\AndroidStudio。

（1）　主界面程序

在主界面是对 ListView、Viewpager 里面进行初始化和适配，在 Viewpager 里面添加 Fragment，在点击 ListVeiw 的时候显示对应的 Fragment；

源码位置：com.hqyj.dev.procedurem4.activities.MainActivity。java

华清远见 dev.hqyj.com

NormalText Code

```
public class MainActivity extends FragmentActivity implements
AdapterView.OnItemClickListener, ViewPager.OnPageChangeListener {

    private ViewPager viewPager;
    private List<Fragment> fragmentList;
    private ListView listView;
    private List<Module> modulesNameList;
    private View oldView = null;
    int oldPositon = -1;
    private DrawListViewItem drawListViewItem;
    private View view;
    private int position;
    private final String TAG = "Main";
    private TextView txvShowModuleName;
    private DrawSwitcher drawSwitcher;

    /**
     * 异步处理机制，处理线程传过来的数据，实现了在子线程中更新 UI 界面
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler() {
        /**
         * 在 handleMessage 里面处理线程传过来的数据
         * @param msg msg 里面包含了线程传过来的各种类型的数据
         */
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);

            switch (msg.what) {
                /**
                 *点击 ListVeiw 的时候进行图片切换，灰色的变为亮色的
                 */
                case 1:
                    if (position != oldPositon) {
                        if (oldView != null) {
                            drawListViewItem = (DrawListViewItem) oldView.
                                    findViewById(R.id.item_list);
                            drawListViewItem.setIsChoose(false);
                            drawListViewItem.invalidate();
                        }
                        drawListViewItem = (DrawListViewItem) view.
```

```
                                    findViewById(R.id.item_list);
                drawListViewItem.setIsChoose(true);
                drawListViewItem.invalidate();
                oldPositon = position;
                oldView = view;
            }
            break;
        /**
         * 通过点击 ListView,设置 ViewPager 当前的界面
         */
        case 2:
            if (view != oldView) {
                drawListViewItem = (DrawListViewItem) view.
                            findViewById(R.id.item_list);
                drawListViewItem.setIsChoose(true);
                drawListViewItem.invalidate();

                if (oldView != null) {
                    drawListViewItem = (DrawListViewItem) oldView.
                                    findViewById(R.id.item_list);
                    drawListViewItem.setIsChoose(false);
                    drawListViewItem.invalidate();
                }
            }
            oldView = view;
            oldPositon = position;
            viewPager.setCurrentItem(position);
            break;
        /**
         * 设置界面的名字，显示提示拨码开关的图片
         */
        case 3:
            int get = msg.getData().getInt(TAG);

            Module module = modulesNameList.get(get);

            txvShowModuleName.setText(module.getName());
            Log.d(TAG, module.getName());
            drawSwitcher.setBitmap(module.getSwitcher());
            drawSwitcher.invalidate();
            break;
        default:
            break;
```

```
86              }
87          }
88      };
89      @Override
90      protected void onCreate(@Nullable Bundle savedInstanceState) {
91          super.onCreate(savedInstanceState);
92          requestWindowFeature(Window.FEATURE_NO_TITLE);//设置全屏和无标题栏
93          getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
94                               WindowManager.LayoutParams.FLAG_FULLSCREEN);
95          setContentView(R.layout.main_activity);
96          ModulesInfo.application = getApplication();
97          initView();//初始化界面
98      }
99      @Override
100     protected void onDestroy() {
101         super.onDestroy();
102     }
103     /**
104      * 初始化 ListView、Viewpager 控件通过适配器，并设置监听
105      */
106     private void initView() {
107
108         txvShowModuleName = (TextView) findViewById(R.id.txv_show_module_name);
109         viewPager = (ViewPager) findViewById(R.id.view_pager);
110         listView = (ListView) findViewById(R.id.list_view);
111         drawSwitcher = (DrawSwitcher) findViewById(R.id.draw_switcher);
112         fragmentList = new ArrayList<>();
113         modulesNameList = new ArrayList<>();
114         setFragmentList();//创建节点引用，向 ListView、Viewpager 增加内容
115         SwitchAdapter switchAdapter = new SwitchAdapter
116             (getSupportFragmentManager(), fragmentList);
117         viewPager.setAdapter(switchAdapter);
118         ListViewAdapter listViewAdapter = new ListViewAdapter
119                                         (this, modulesNameList);
120         listView.setAdapter(listViewAdapter);
121         listView.setOnItemClickListener(this);
122         viewPager.addOnPageChangeListener(this);
123     }
124
125     /**
126      * 创建节点引用，向 ListView、Viewpager 增加内容
127      */
128     private void setFragmentList() {
```

```
129
130         AlcoholFragment alcoholFragment = new AlcoholFragment();
131         BrakeFragment brakeFragment = new BrakeFragment();
132         BuzzerFragment buzzerFragment = new BuzzerFragment();
133         DCMotorFragment dcMotorFragment = new DCMotorFragment();
134         GasFragment gasFragment = new GasFragment();
135         LightFragment lightFragment = new LightFragment();
136         MatrixFragment matrixFragment = new MatrixFragment();
137         RelayFragment relayFragment = new RelayFragment();
138         RFIDFragment rfidFragment = new RFIDFragment();
139         ServoFragment servoFragment = new ServoFragment();
140         SteeperFragment steeperFragment = new SteeperFragment();
141         ThermistorFragment thermistorFragment = new ThermistorFragment();
142         TubeFragment tubeFragment = new TubeFragment();
143         TempFragment tempFragment = new TempFragment();
144
145         fragmentList.add(alcoholFragment);
146         fragmentList.add(brakeFragment);
147         fragmentList.add(buzzerFragment);
148         fragmentList.add(dcMotorFragment);
149         fragmentList.add(gasFragment);
150         fragmentList.add(lightFragment);
151         fragmentList.add(matrixFragment);
152         fragmentList.add(relayFragment);
153         fragmentList.add(rfidFragment);
154         fragmentList.add(servoFragment);
155         fragmentList.add(steeperFragment);
156         fragmentList.add(thermistorFragment);
157         fragmentList.add(tubeFragment);
158         fragmentList.add(tempFragment);
159
160         modulesNameList.add(fragmentList.indexOf(alcoholFragment),
161                 Alcohol.getAlcohol());
162         modulesNameList.add(fragmentList.indexOf(brakeFragment),
163                 Brake.getBrake());
164         modulesNameList.add(fragmentList.indexOf(buzzerFragment),
165                 Buzzer.getBuzzer());
166         modulesNameList.add(fragmentList.indexOf(dcMotorFragment),
167                 DCMotor.getDcMotor());
168         modulesNameList.add(fragmentList.indexOf(gasFragment),
169                 Gas.getGas());
170         modulesNameList.add(fragmentList.indexOf(lightFragment),
171                 Light.getLight());
```

```
172         modulesNameList.add(fragmentList.indexOf(matrixFragment),
173                 Matrix.getMatrix());
174         modulesNameList.add(fragmentList.indexOf(relayFragment),
175                 Relay.getRelay());
176         modulesNameList.add(fragmentList.indexOf(rfidFragment),
177                 Rfid.getRfid());
178         modulesNameList.add(fragmentList.indexOf(servoFragment),
179                 Servo.getServo());
180         modulesNameList.add(fragmentList.indexOf(steeperFragment),
181                 Steeper.getSteeper());
182         modulesNameList.add(fragmentList.indexOf(thermistorFragment),
183                 Thermistor.getThermistor());
184         modulesNameList.add(fragmentList.indexOf(tubeFragment),
185                 Tube.getTube());
186         modulesNameList.add(fragmentList.indexOf(tempFragment),
187                 Temp.getTemp());
188     }
189
190     /**
191      * ListView 的点击监听
192      * @param parent 适配器的对象
193      * @param view 点击的画面
194      * @param position 点击按钮在 ListView 中的位置
195      * @param id 点击按钮在 ListView 中的 id
196      */
197     @Override
198     public void onItemClick(AdapterView<?> parent, View view,
199                                         int position, long id) {
200
201         Log.d("TEST", position + ":" + id + ":");
202         this.position = position;
203         this.view = view;
204         handler.sendEmptyMessage(2);
205     }
206
207     @Override
208     public void onPageScrolled(int position, float positionOffset,
209                                       int positionOffsetPixels) {}
210     /**
211      * 点击 ListView 按钮的作用
212      * @param position 点击按钮在 ListView 中的位置
213      */
214     @Override
```

```
    public void onPageSelected(int position) {

        this.position = position;
        int child = listView.getChildCount();
        int hChild = child - child / 2;
        Message msgPosition = new Message();
        Bundle bundle = new Bundle();
        bundle.putInt(TAG, position);
        msgPosition.setData(bundle);
        msgPosition.what = 3;
        //对 handler 发送信息
        handler.sendMessage(msgPosition);
        //设置点击的按钮，让它显示在屏幕中间
        if (position < hChild - 1) {
            listView.setSelection(0);
            view = listView.getChildAt(position);
        } else if (position > modulesNameList.size() - hChild) {
            listView.setSelection(modulesNameList.size() - child);
            view = listView.getChildAt(position - child);
        } else {
            listView.setSelection(position - hChild + 1);
            //从前往后，向上
            if (oldPositon < position) {
                if (position == hChild - 1) {
                    view = listView.getChildAt(hChild - 1);
                } else {
                    view = listView.getChildAt(hChild);
                }
            } else {//从后往前，向下
                if (position == modulesNameList.size() - hChild) {
                    view = listView.getChildAt(hChild -1);
                } else {
                    view = listView.getChildAt(hChild - 2);
                }
            }
        }
        handler.sendEmptyMessage(1);
    }
    @Override
    public void onPageScrollStateChanged(int state) {}
}
```

（2） 自己定义的适配器

自定义适配器是控件在需要更好的适配的时候自己定义的，这样使得控件使用起来更加好看，更加多样化。

源码位置：com.hqyj.dev.procedurem4.activities.adapter ListViewAdapter.

NormalText Code

```
public class ListViewAdapter extends BaseAdapter {

private Context mContext;
private LayoutInflater mLayoutInflater;
private List<Module> modulesList = new ArrayList<>();

/**
 * @param context 当前文本
 * @param modules 模块引用的列表
 */
public ListViewAdapter(Context context, List<Module> modules) {
    modulesList = modules;
    mContext = context;
    mLayoutInflater = LayoutInflater.from(mContext);
}

/**
 * @return 列表的大小
 */
@Override
public int getCount() {
    return modulesList.size();
}
public class OC{
    public DrawListViewItem drawListViewItem;
}

/**
 * @param position 当前点击的位置
 * @return 获得当前点击的位置
 */
@Override
public Object getItem(int position) {
    return (modulesList == null) ? null : modulesList.get(position);
}

@Override
public long getItemId(int position) {
```

```
39          return position;
40      }
41
42      /**
43       * @param position 当前列表的位置
44       * @param convertView 想要转换的视图
45       * @param parent 视图租
46       * @return 返回转换的视图
47       */
48      @Override
49      public View getView(int position, View convertView, ViewGroup parent) {
50
51          OC oc = null;
52          if (convertView == null){
53              oc = new OC();
54              convertView = mLayoutInflater.inflate(R.layout.listview_item, null);
55              oc.drawListViewItem = (DrawListViewItem) convertView.
56                                      findViewById(R.id.item_list);
57              Module module = modulesList.get(position);
58              oc.drawListViewItem.setName(module.getName());
59              oc.drawListViewItem.setIsChoose(false);
60              convertView.setTag(oc);
61          }else {
62              oc = (OC) convertView.getTag();
63          }
64          Module module = modulesList.get(position);
65          oc.drawListViewItem.setName(module.getName());
66          oc.drawListViewItem.setIsChoose(false);
67          oc.drawListViewItem.invalidate();
68          return convertView;
69      }
}
```

3.2.3 【界面展示】

图 数码管操作界面

图 步进电机操作界面

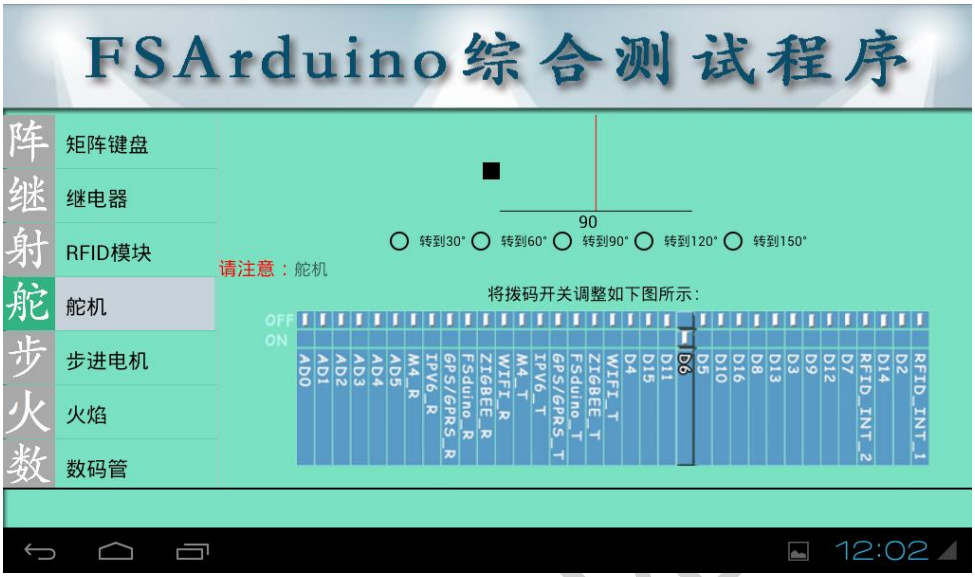


图 舵机操作界面

图 RFID 操作界面

图 继电器操作界面

3.3 酒精传感器实验

3.3.1 【实验目的】

（1） 掌握酒精传感器的工作原理；

（2） 掌握对传感器数据进行接受和处理；

（3） 掌握在 Android 系统下的传感器开发流程；

3.3.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

3.3.3 【实验内容】

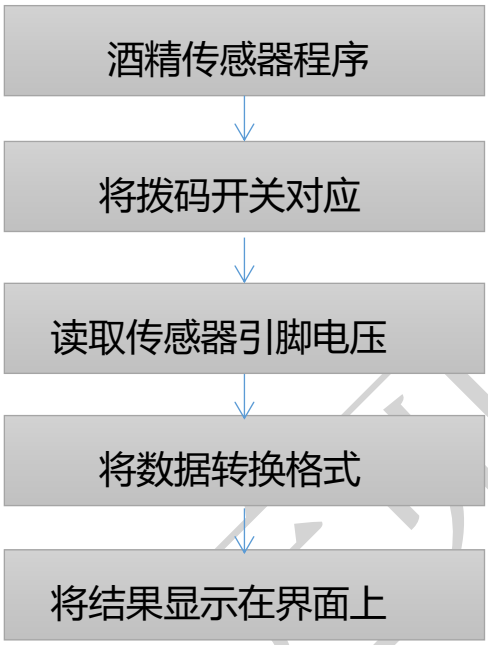
基于 Android5.0 系统实现读取酒精传感器两个引脚的电压值。

3.3.4 【实验原理】

硬件原理参见第二章的酒精传感器驱动实验；

在应用程序里通过调用接口读取酒精传感器两引脚的电压，并将读取到的数据进行运算和格式转换，

然后将结果显示出来。

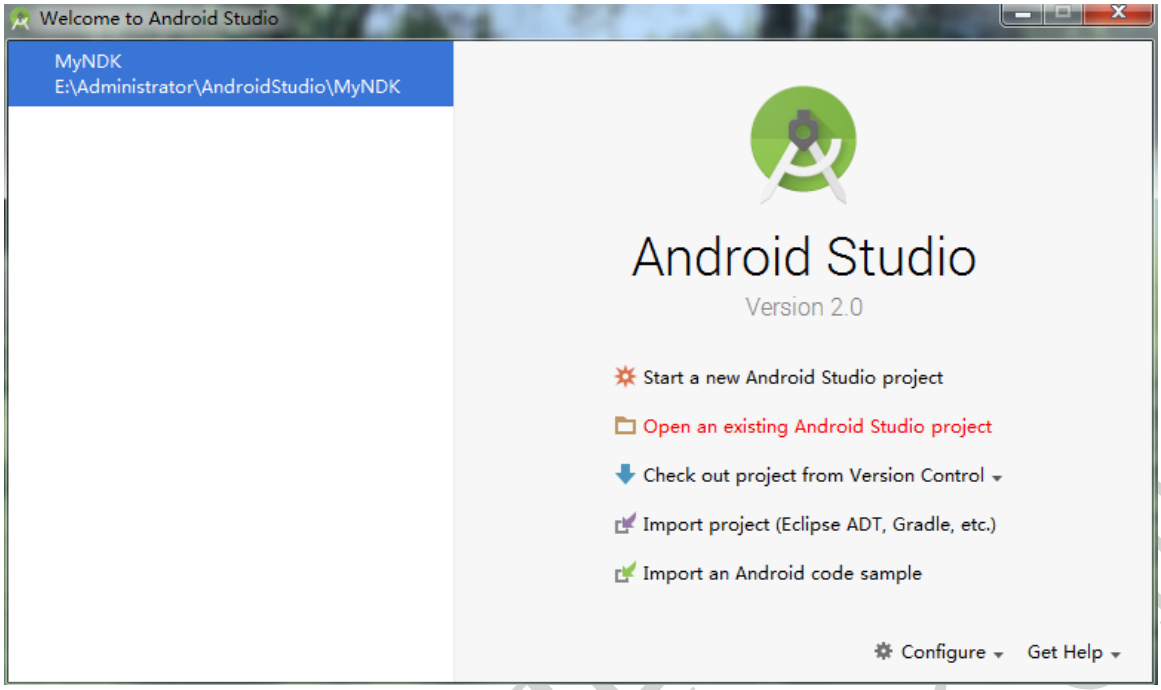


3.3.5 【实验步骤】

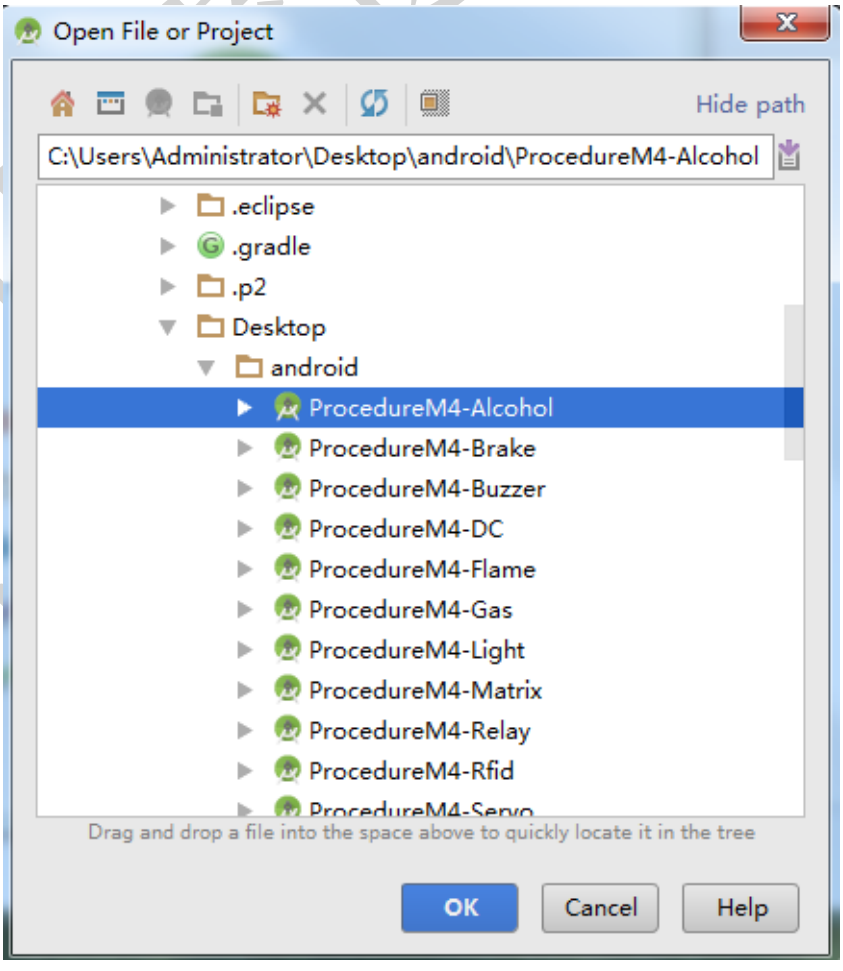
酒精传感器实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Alcohol】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹里。

打开 Android studio，导入实验程序：

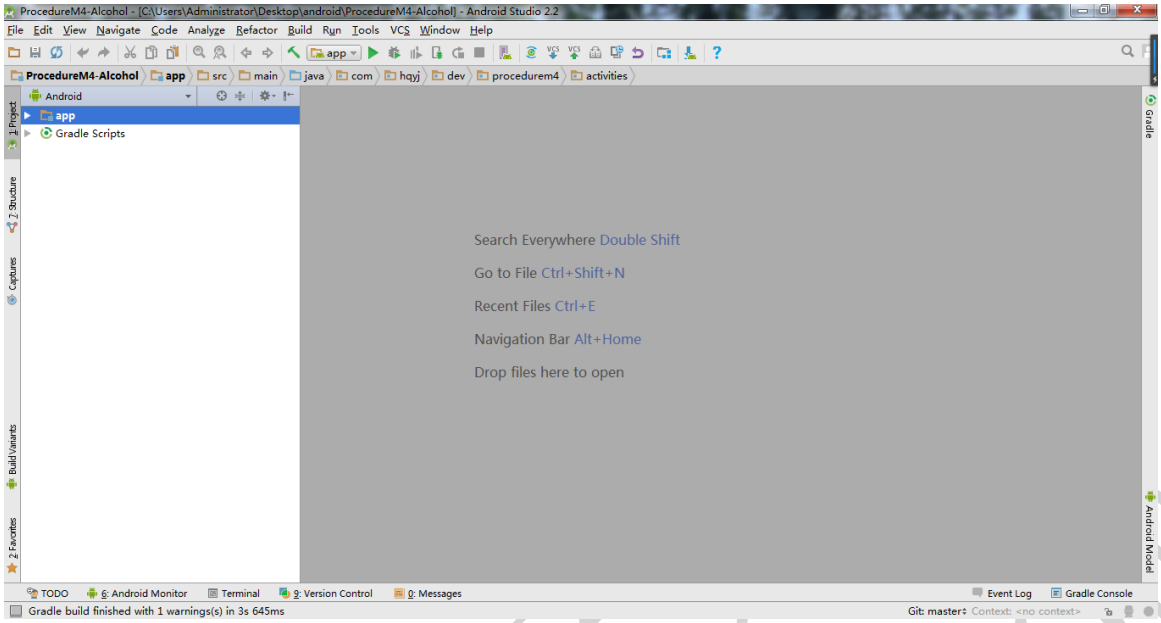
华清远见 dev.hqyj.com



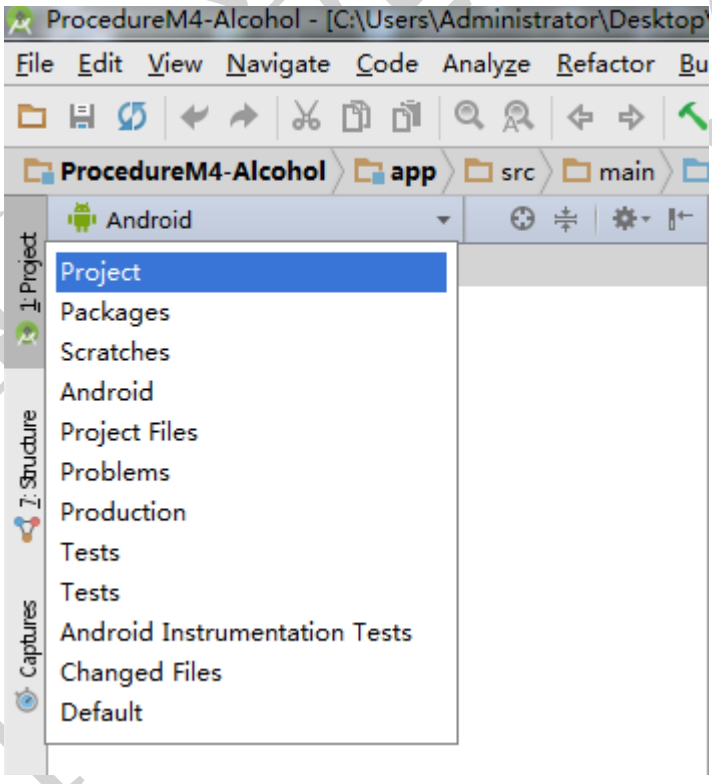
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



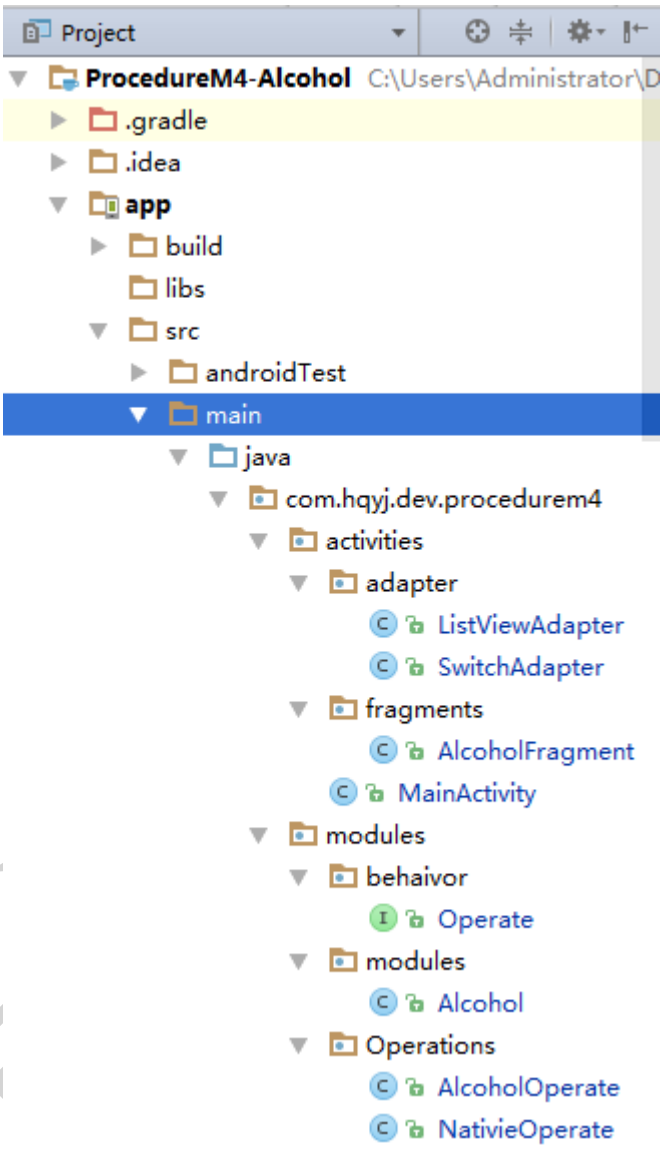
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：

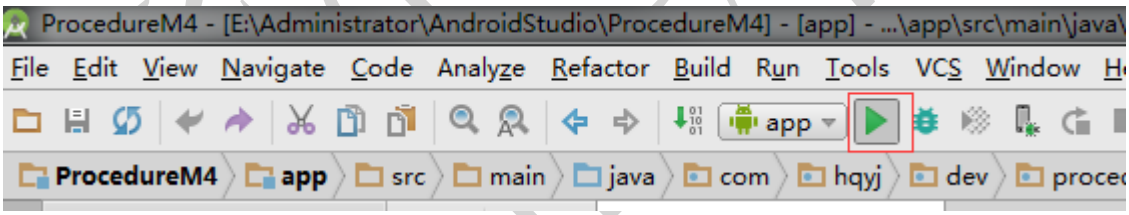


程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

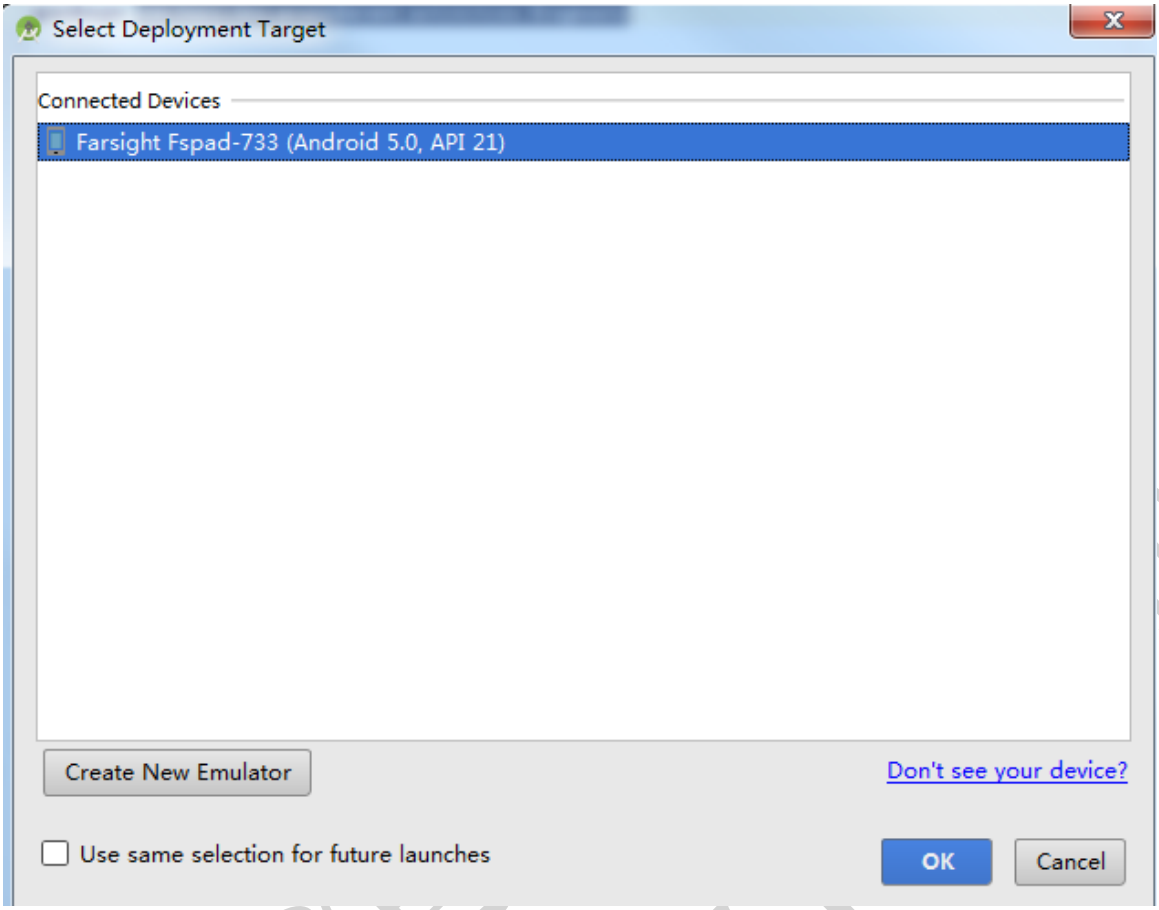
代码写好之后，将代码编译并运行在 4412 板子上。

然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

点击进入程序进行测试。

3.3.6 【实验结果】

将拨码开关中的 AD1 拨至 ON，其他拨码全为 OFF，如图中提示。

当周围环境酒精浓度越大，值越大，结果如上图所示。

酒精传感器在模块中的位置如下图所示：

3.3.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-0jiujing.app.src.main.jni.Android.mk

NormalText Code

```
  LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate//库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 adc 的数据

include $(BUILD_SHARED_LIBRARY)
```

读取 adc 数据的.c 文件

源码位置：ProcedureM4-0jiujing.app.src.main.jni.operate_adc.c

NormalText Code

```
/**
*实现读取 adc 的采样值
*/
int read_adc(int which){
    int fd;
    int value;
    //以可读可写的形式打开"/dev/adc"这个路径下的 adc，创建流通道
    fd = open("/dev/adc", O_RDWR);
    if (fd < 0){
        LOGI("Open ADC error");
        return fd;
    }
    switch(which){
        case ADC_ALCOHOL://设置酒精传感器传输数据通道
            ioctl(fd, SET_CHANNEL, ALCOHOL_CHANNEL);
            break;
        case ADC_LIGHT:
            ioctl(fd, SET_CHANNEL, LIGHT_CHANNEL);
            break;
        case ADC_SENSITIVE:
            ioctl(fd, SET_CHANNEL, SENSITIVE_CHANNEL);
            break;
```

```
        case ADC_GAS:
            ioctl(fd, SET_CHANNEL, GAS_CHANNEL);
            break;
        default:
            LOGI("ERROR ADC");
            break;
    }
    //通过通道读取传感器的数据
    read(fd, &value, sizeof(value));
    LOGI("value : %d\n", value );
    //关闭流通道
    close(fd);
    return value;
}
```

（2） Android 端对数据的接受和处理

在具体的酒精的 Fragment 中，开启线程读取传感器的数据，然后用 handler 异步处理，对 UI 界面进行动态显示的改变。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.AlcoholFragment.java

NormalText Code

```
  public class AlcoholFragment extends Fragment {
1
2     private View mView;
3     private TextView textView;
4     private boolean threadOn = false;
5     private Alcohol alcohol;
6     private AlcoholReadThread alcoholReadThread = null;
7     private final String ALOCHOL = "ALOCHOL";
8     /**
9      *装载名字叫"operate"的库文件
10     */
11    static{
12        System.loadLibrary("operate");
13    }
14    /**
15     * 异步处理机制，实现子线程更新 UI 界面的功能
16     */
17    @SuppressLint("HandlerLeak")
18    private Handler handler = new Handler(){
19        @Override
20        public void handleMessage(Message msg) {
21            super.handleMessage(msg);
```

```
            switch (msg.what){
                //将处理过的数据在界面上显示出来
                case 1:
                    String value = msg.getData().getString(ALOCHOL);
                    textView.setTextSize(50);
                    textView.setText(String.format("%sV", value));
                    break;
                default:
                    break;
            }
        }
    };
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        alcohol = Alcohol.getAlcohol();
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable
    ViewGroup container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.alcohol_fragment,
                                        container, false);
        initShow();//初始化控件
        //开启读取传感器数据的线程
        alcoholReadThread = new AlcoholReadThread();
        threadOn = true;
        alcoholReadThread.start();
        return mView;
    }
    private void initShow() {
        textView = (TextView) mView.findViewById(R.id.alcohol);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();

    }
    @Override
    public void onDestroyView() {
        super.onDestroyView();
        //关闭线程
        threadOn = false;
```

```
        alcoholReadThread.interrupt();
    }

    /**
     * 读取传感器数据的线程
     */
    private class AlcoholReadThread extends Thread{
        @Override
        public void run() {
            super.run();
            while (threadOn){
                Bundle b = new Bundle();
                Message msg = new Message();
                //将数据显示成"#.##"这种格式
                DecimalFormat df = new DecimalFormat("#.##");
                //读取酒精传感器的数据
                int value = alcohol.operate.read()[0];
                String result = df.format((double)value*7.2/4096);
                b.putString(ALOCHOL, result);
                msg.what = 1;
                msg.setData(b);
                //将数据交个 handler 处理
                handler.sendMessage(msg);
                try {
                    sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

3.4 光电闸实验

3.4.1 【实验目的】

（1） 掌握光电闸的工作原理；

（2） 掌握对传感器数据进行接受和处理；

（3） 掌握在 Android 系统下的传感器开发流程；

3.4.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

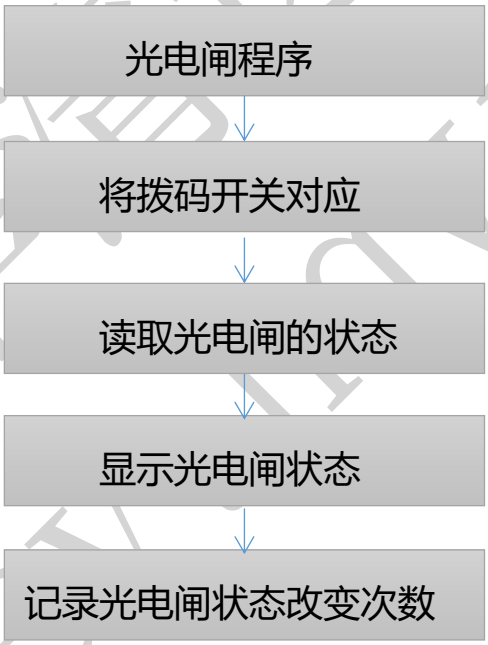
（3） u-boot-2010.03 版本；

3.4.3 【实验内容】

基于 Android5.0 系统通过光电闸对物体穿过的感应，显示当前状态并进行计数。

3.4.4 【实验原理】

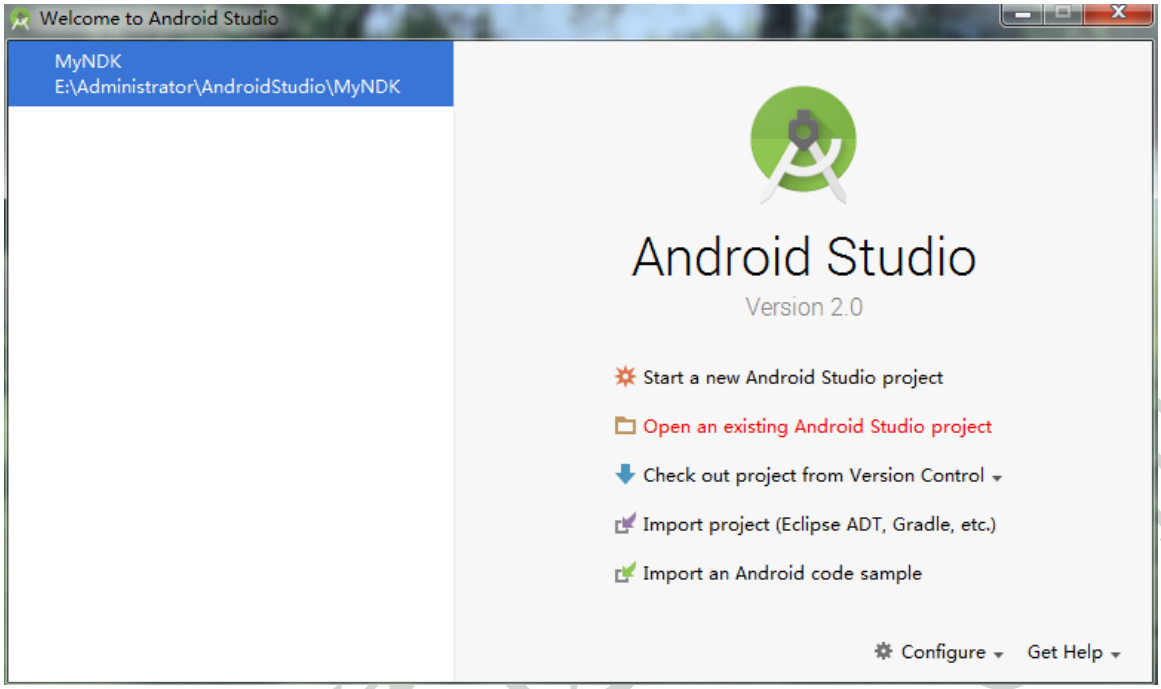
在应用程序里通过调用接口读取光电闸的状态，并将关电闸的状态在屏幕上显示出来，记录物体通过光电闸的次数。



3.4.5 【实验步骤】

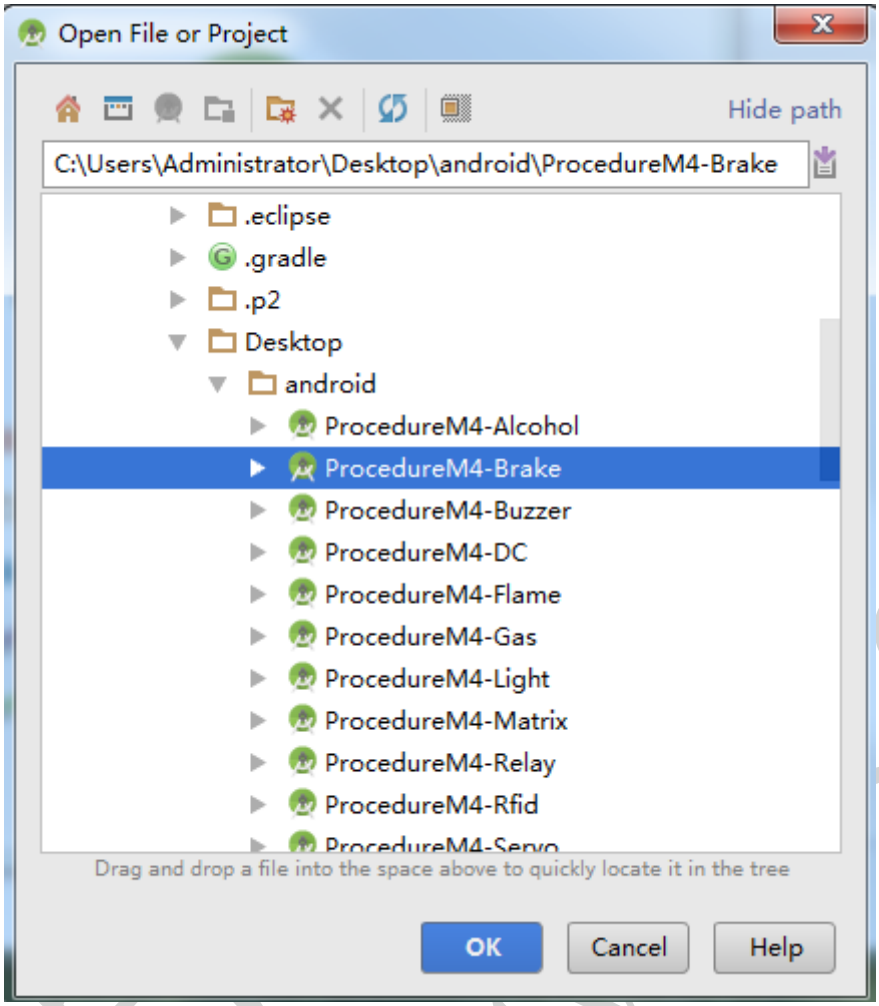
光电闸实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Brake】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面 android 文件夹下

打开 Android studio，导入实验程序：

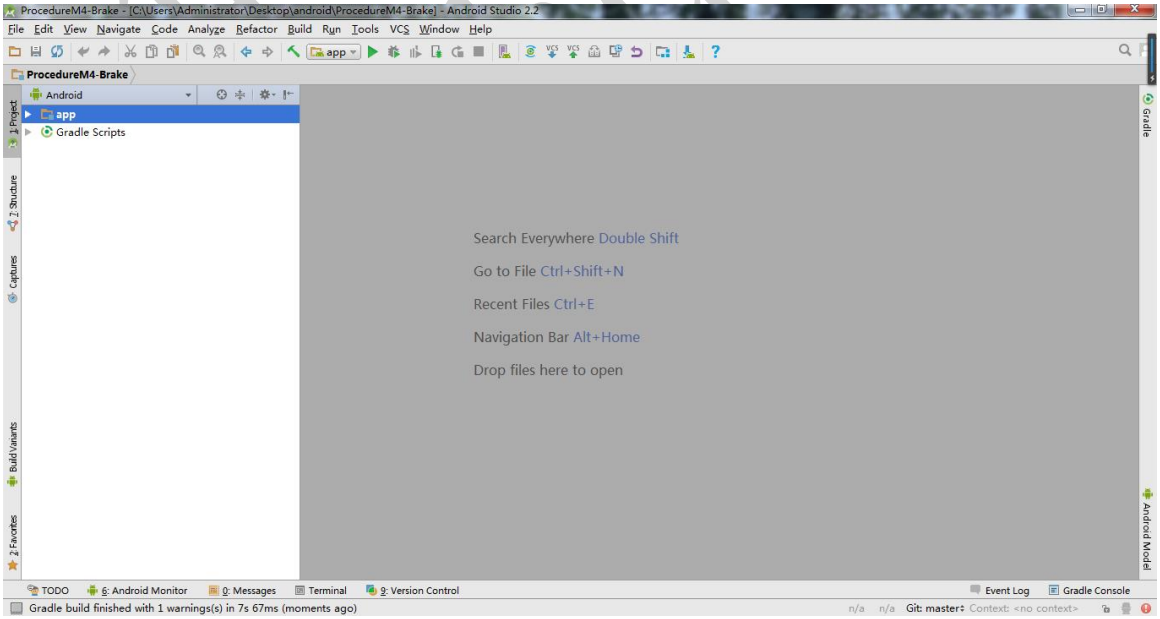


选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：

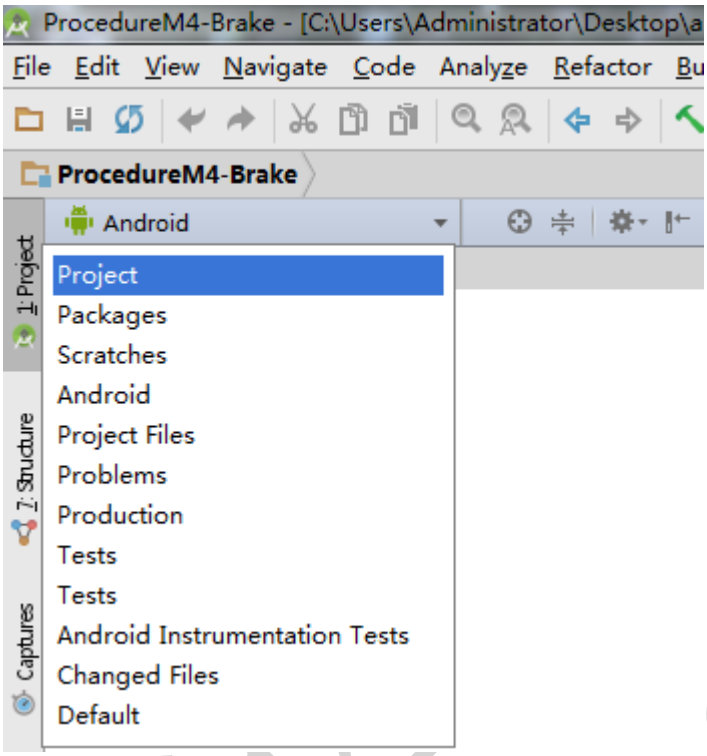
华清远见 dev.hqyj.com



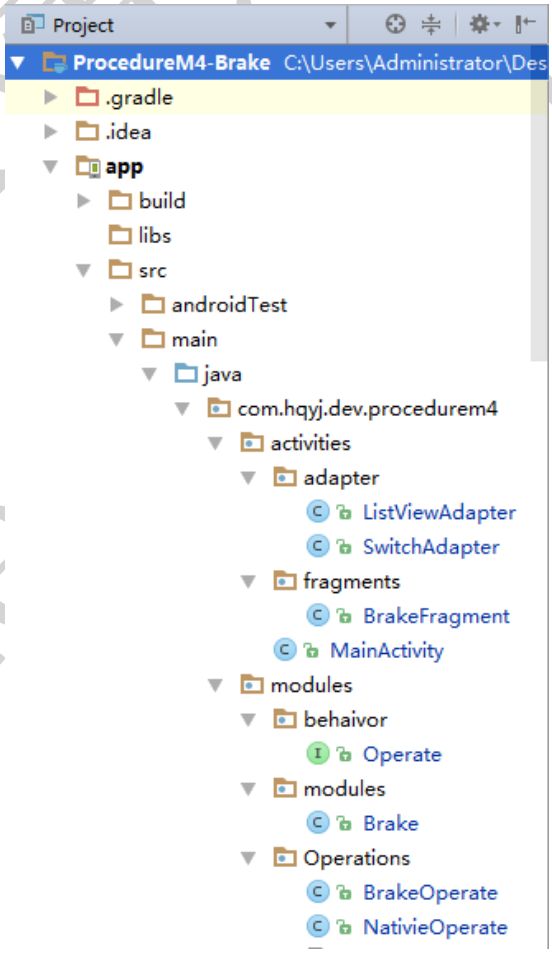
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：

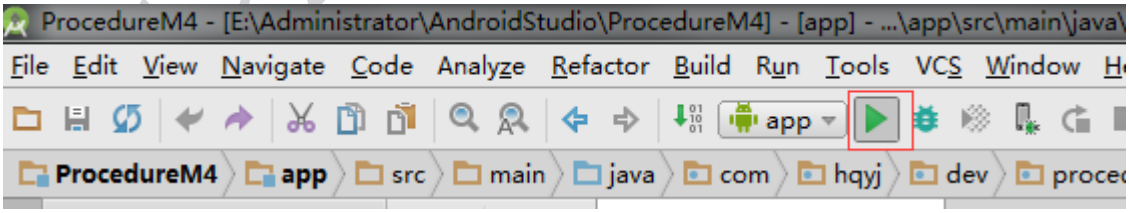


程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

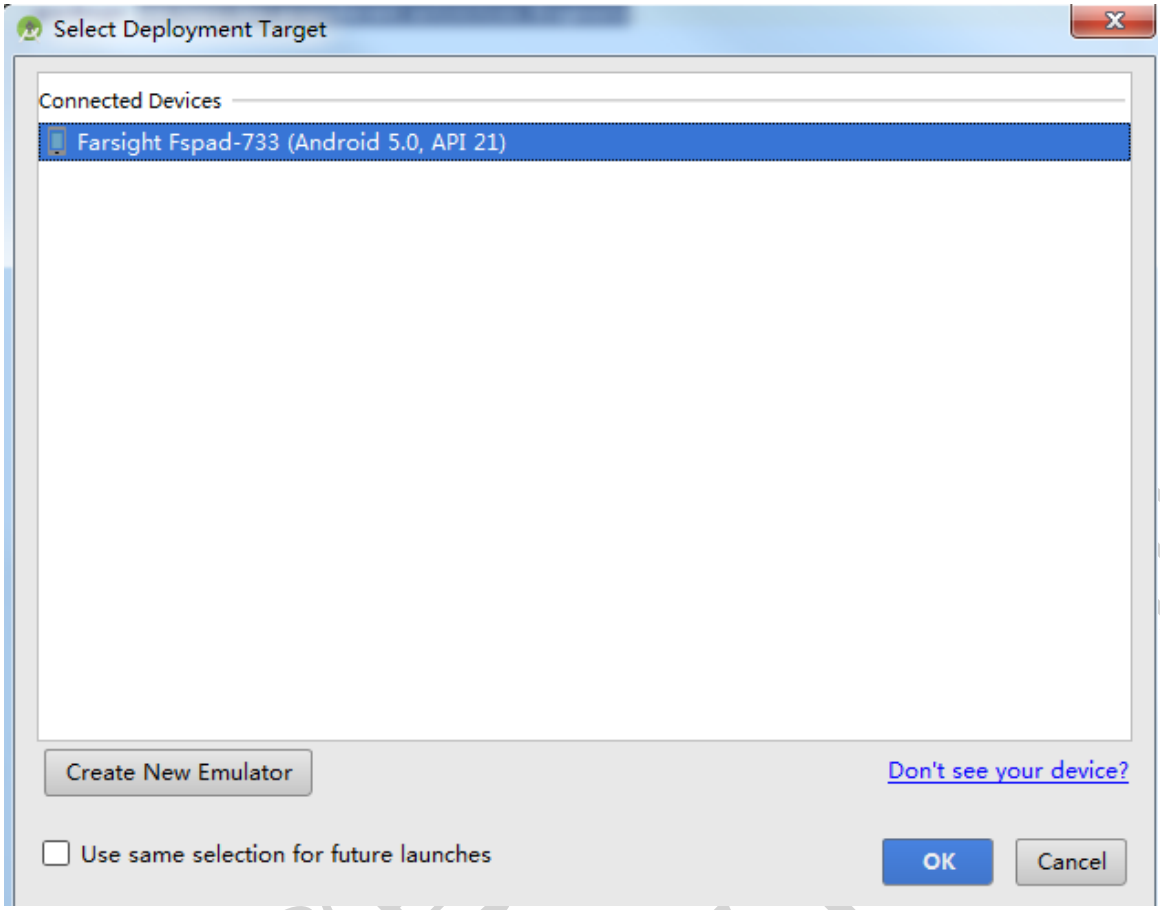
代码写好之后，将代码编译并运行在 4412 板子上。

然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

点击进入程序进行测试。

3.4.6 【实验结果】

将拨码开关中的 D14 拨至 ON，其他拨码全为 OFF，如图中提示。

可以用不透明的东西挡住进行测试，结果如上图所示。

光电闸在模块中的位置如下图所示：

3.4.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-1guandian.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 break 的数据

include $(BUILD_SHARED_LIBRARY)
```

读取 brake 状态的.c 文件

源码位置：ProcedureM4-1guandian.app.src.main.jni.operate_brake.c

NormalText Code

```
/**
*实现读取 break 的采样值
*/
int read_brake_state(){
    int fd;
    int state;
    int data;
    //以可读可写的形式打开 BRAKE_FILE，创建流通道
    fd = open(BRAKE_FILE, O_RDWR);
    if(fd < 0){
        LOGI("Open Error : %s", "Brake");
        return -1;
    }
    //读取 break 的状态
    state = read(fd, &data, 1);
    //关闭流通道
    close(fd);
```

```
    return state;
}
```

（2） Android 端对数据的接受和处理

在具体的光电闸的 Fragment 中，开启线程读取传感器的数据，然后用 handler 异步处理，并在 UI 界面上显示出光电闸的状态。

源码位置：com.hqyj.dev.procedurem4.activities.fragments. BrakeFragment.java

NormalText Code

```
public class BrakeFragment extends Fragment {
    private View mView;
    private TextView textView;
    private TextView textViewCount;
    private boolean threadOn = false;
    private BrakeReadThread brakeReadThread;
    private Brake brake;
    private int count = 0;
    private String TAG = "BRAKE";
    private int oldState = -1;
    @SuppressLint("HandlerLeak")
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
     */
    private Handler handler = new Handler() {
        @SuppressLint("DefaultLocale")
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what) {
                //将光电闸的状态在界面上显示出来
                case 1:
                    int state = msg.getData().getInt(TAG);
                    textView.setTextSize(30);
                    textView.setText(String.format("光电闸：%s",
     (state == 0x00) ? "开" : (state == 0x10) ? "关" : "未知"));
                    textViewCount.setText(String.format
                        ("光电闸开关次数：%d", count));
                    break;
            }
        }
    };
    /**
```

```
34      *装载名字叫"operate"的库文件
35      */
36     static {
37         System.loadLibrary("operate");
38     }
39     @Override
40     public void onCreate(@Nullable Bundle savedInstanceState) {
41         super.onCreate(savedInstanceState);
42         brake = Brake.getBrake();
43     }
44     @Nullable
45     @Override
46     public View onCreateView(LayoutInflater inflater, @Nullable
47     ViewGroup container, @Nullable Bundle savedInstanceState) {
48         mView = inflater.inflate(R.layout.brake_fragment,
49                                         container, false);
50         initShow();//初始化控件
51         threadOn = true;
52         //开启读取传感器数据的线程
53         brakeReadThread = new BrakeReadThread();
54         brakeReadThread.start();
55         return mView;
56     }
57     private void initShow() {
58         textView = (TextView) mView.findViewById(R.id.txv_brake);
59         textViewCount = (TextView) mView.findViewById
60                                 (R.id.txv_count_brake);
61     }
62     @Override
63     public void onDestroyView() {
64         super.onDestroyView();
65         threadOn = false;
66         brakeReadThread.interrupt();
67     }
68     @Override
69     public void onDestroy() {
70         super.onDestroy();
71         //关闭线程
72         threadOn = false;
73         brakeReadThread.interrupt();
74         count = 0;
75         oldState = -1;
76     }
```

```
77      /**
78       * 读取传感器数据的线程
79       */
80      private class BrakeReadThread extends Thread {
81          @Override
82          public void run() {
83              super.run();
84              while (threadOn) {
85                  int value = brake.operate.read()[0];
86                  if (value != oldState) {
87                      Bundle bundle = new Bundle();
88                      bundle.putInt(TAG, value);
89                      Message msg = new Message();
90                      msg.what = 1;
91                      msg.setData(bundle);
92                      //将数据交给 handler 处理
93                      handler.sendMessage(msg);
94                      oldState = value;
95                      if (value == 0)
96                          count ++;
97                      try {
98                          sleep(1000);
99                      } catch (InterruptedException e) {
100                         e.printStackTrace();
101                     }
102                 }
103             }
104         }
105     }
106 }
```

3.5 蜂鸣器实验

3.5.1 【实验目的】

（1） 掌握蜂鸣器的工作原理；

（2） 掌握对控制节点数据进行操控；

（3） 掌握在 Android 系统下的控制节点开发流程；

3.5.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

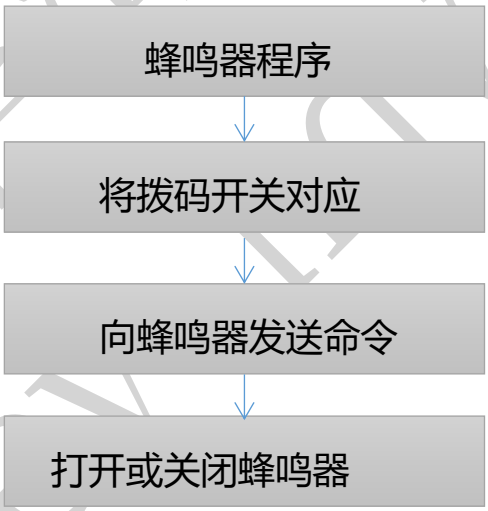
3.5.3 【实验内容】

基于 Android5.0 系统实现控制蜂鸣器工作。

3.5.4 【实验原理】

硬件原理参见第二章的蜂鸣器驱动实验；

在应用程序里设置按钮，用来控制蜂鸣器的打开或关闭，点击按钮时通过调用接口向蜂鸣器发送打开或关闭的命令，控制蜂鸣器的打开或关闭。

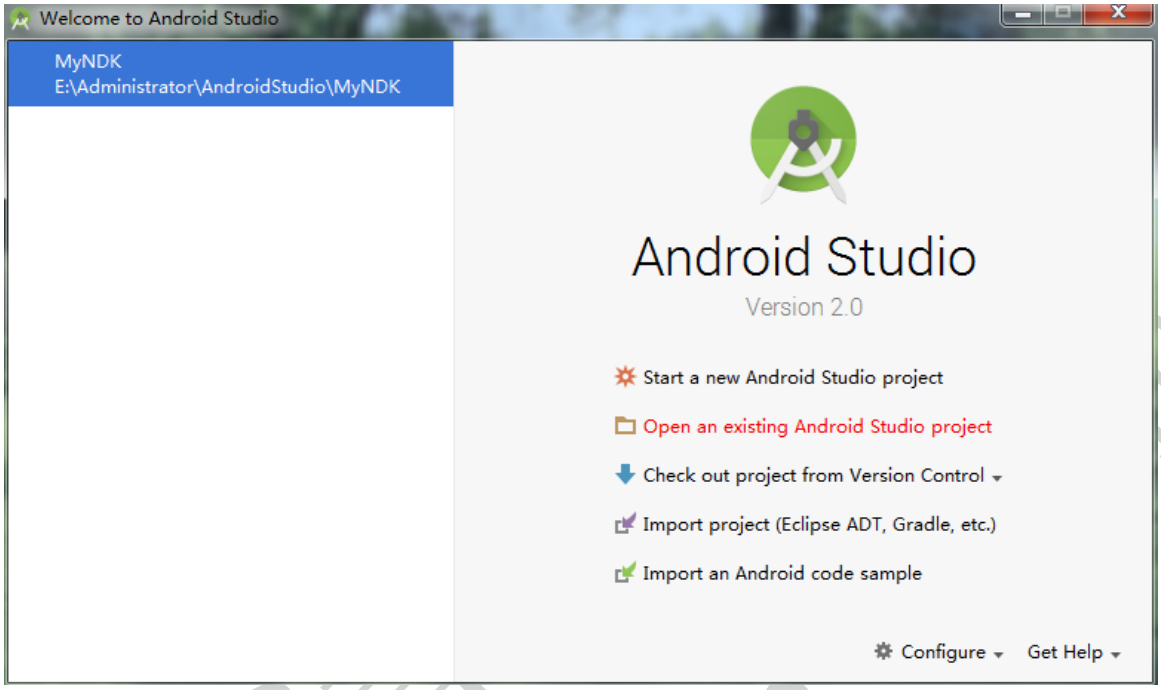


3.5.5 【实验步骤】

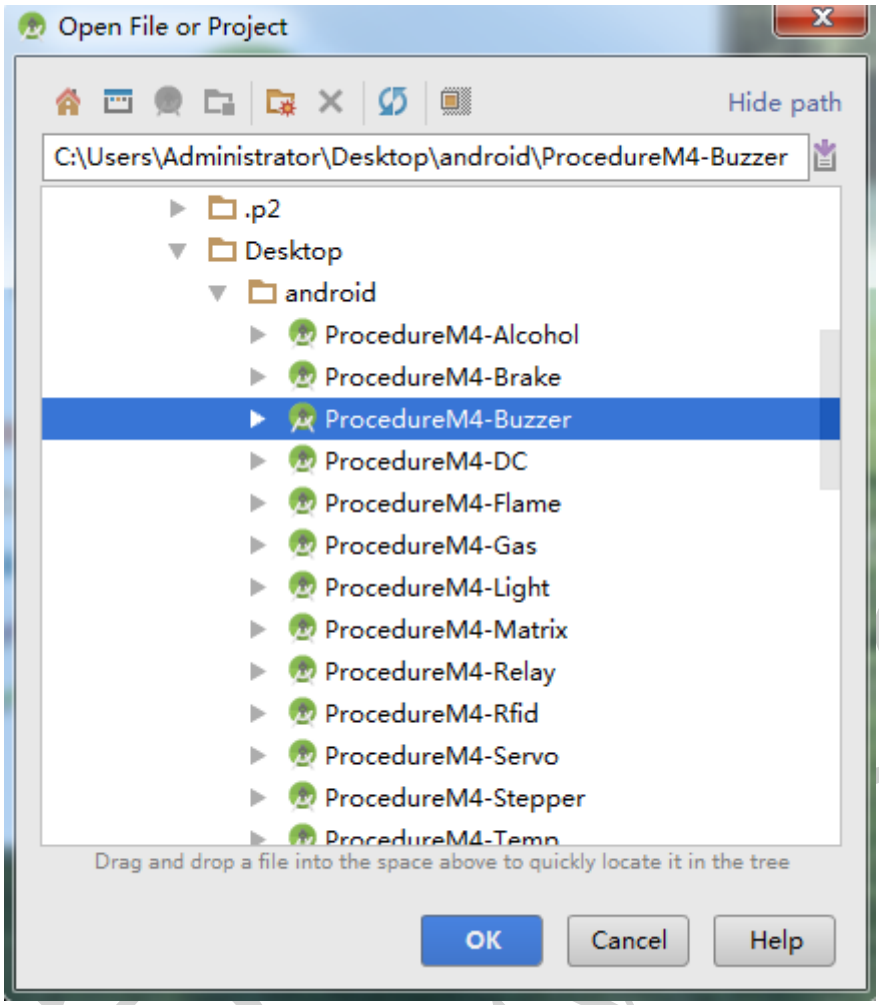
蜂鸣器实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码

\ProcedureM4-Buzzer。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

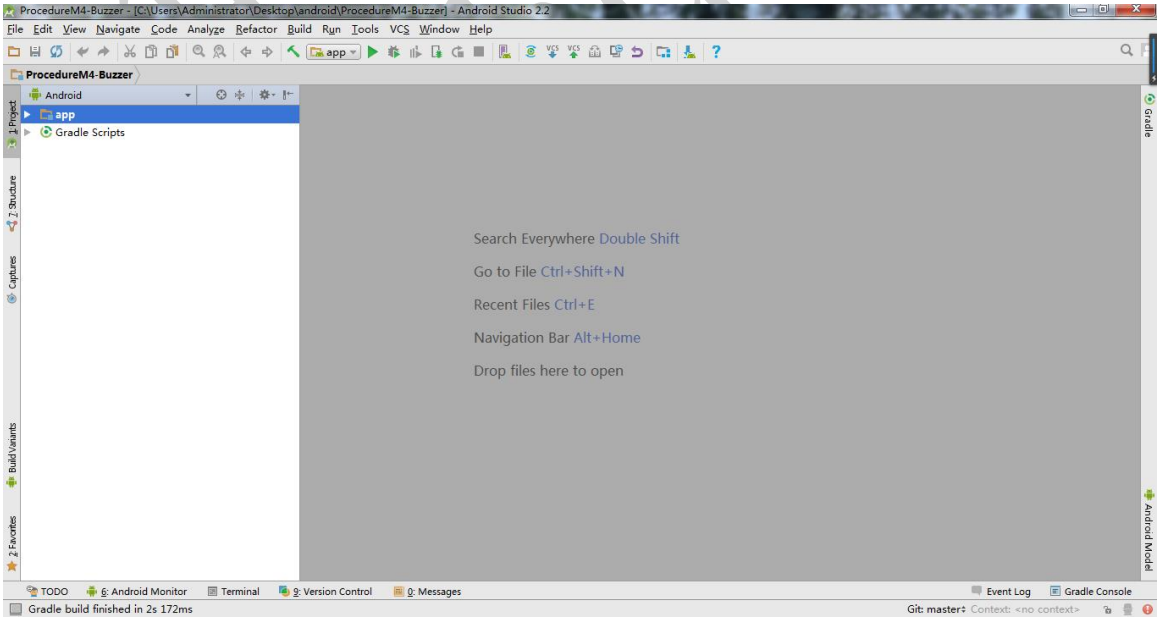
打开 Android studio，导入实验程序：



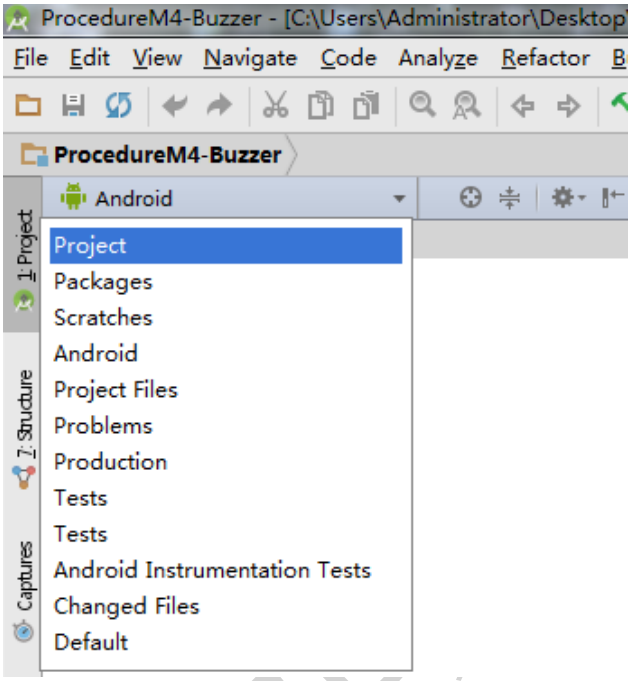
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



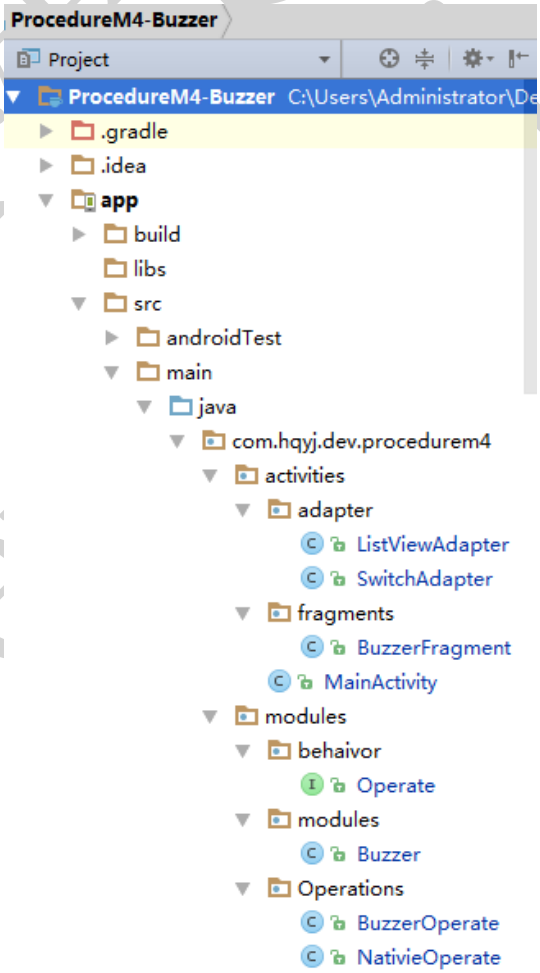
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：

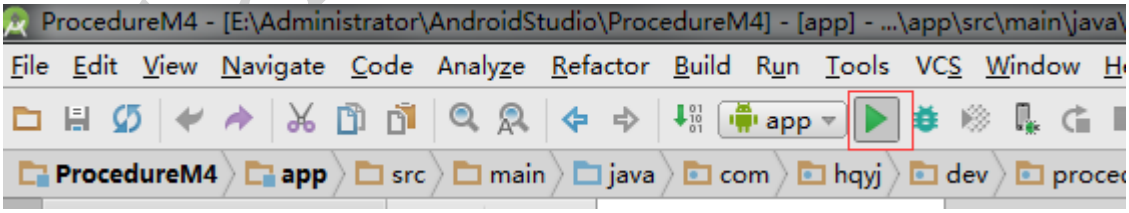


程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

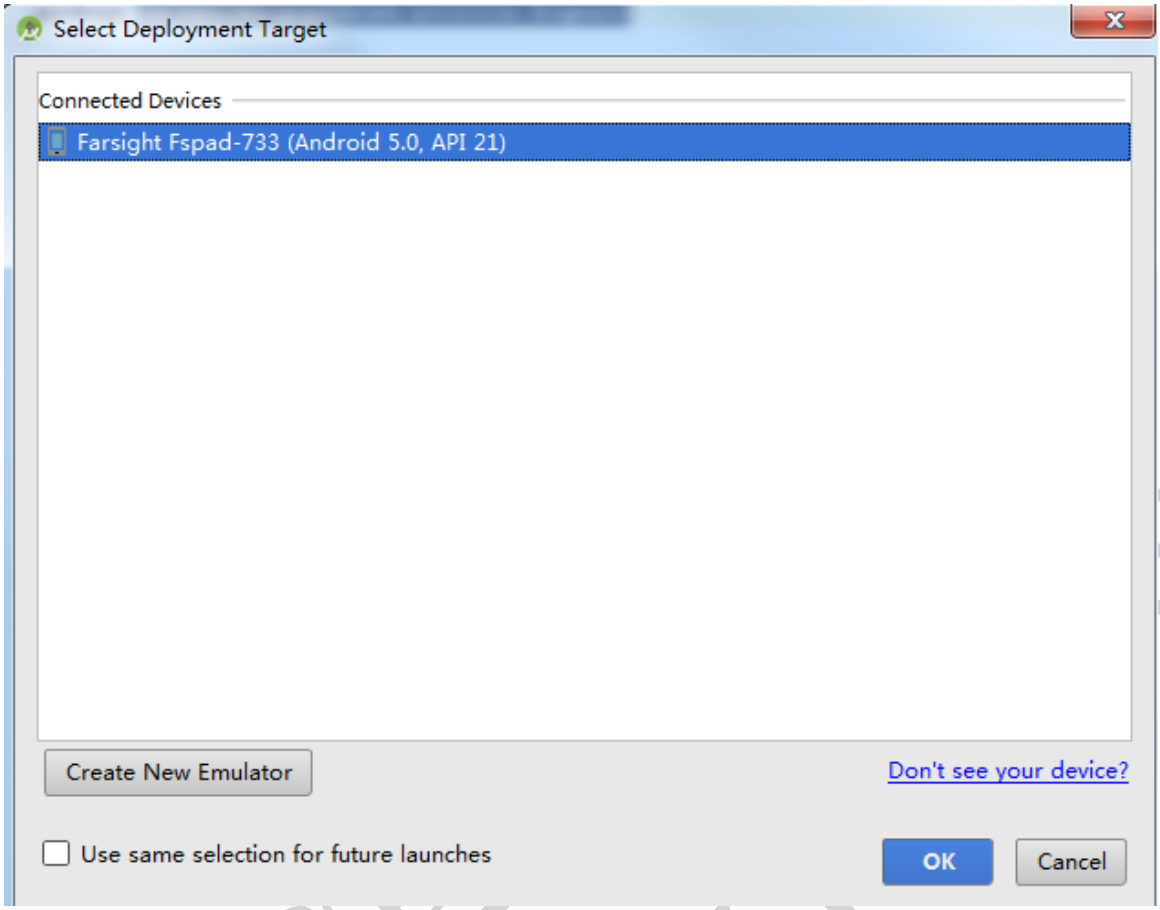
代码写好之后，将代码编译并运行在 4412 板子上。

然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

点击进入程序进行测试。

3.5.6 【实验结果】

将拨码开关中的 D15 拨至 ON，其他拨码全为 OFF，如图中提示。

电机上面的“打开蜂鸣器”和“关闭蜂鸣器”分别打开和关闭蜂鸣器，打开的时有响声。

蜂鸣器在模块中的位置如下图所示：

3.5.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-2fengming.app.src.main.jni.Android.mk

NormalText Code

```
  LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 buzzer 的数据

include $(BUILD_SHARED_LIBRARY)
```

发送 buzzer 的控制命令的.c 文件，控制蜂鸣器的响和不响。

源码位置：ProcedureM4-2fengming.app.src.main.jni.operate_buzzer.c

NormalText Code

```
/**
*打开和关闭蜂鸣器的函数
*/
void write_buzzer(int number){

    if(fd_gpio == -1){
        //创建通道
        fd_gpio = open(BUZZER_FILE, O_RDWR);
        if(fd_gpio < 0){
            LOGI("OPEN ERROR:%s", BUZZER_FILE);
            usleep(1000);
            return;
        }
    }
    usleep(number);
    //打开蜂鸣器
    ioctl(fd_gpio, BUZZER_ON, &buzzer_type);
    usleep(number);
    //关闭蜂鸣器
    ioctl(fd_gpio, BUZZER_OFF, &buzzer_type);
    LOGI("Sleep: %d", number);
```

```
}
/**
*关闭通道
*/
void stop_buzzer(){
    if (fd_gpio = -1){
        usleep(10000);
    }else{
        usleep(10000);
        ioctl(fd_gpio, BUZZER_OFF, &buzzer_type);
        if (fd_gpio != -1){
            close(fd_gpio);
            fd_gpio = -1;
        }
    }
}
```

（2） Android 端对数据的接受和处理

在具体的蜂鸣器的 Fragment 中，设置两个按钮，用来开启和关闭线程，开启和关闭线程对应的是打开和关闭蜂鸣器。

源码位置：com.hqyj.dev.procedurem4.activities.fragments. BuzzerFragment.java

NormalText Code

```
public class BuzzerFragment extends Fragment
            implements View.OnClickListener {

    private View mView;
    private WriteBuzzerThread writeBuzzerThread;
    private boolean threadOn = false;
    private Buzzer buzzer;
    private String TAG = "Buzzer";
    private int num = 80;
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what) {
```

```
                case 1:

                    break;
            }
        }
    };
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        buzzer = Buzzer.getBuzzer();
    }
    /**
     *装载名字叫"operate"的库文件
     */
    static {
        System.loadLibrary("operate");
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable
    ViewGroup container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.buzzer_fragment,
                                        container, false);
        //初始化两个按钮并监听
        mView.findViewById(R.id.btn_open).setOnClickListener(this);
        mView.findViewById(R.id.btn_close).setOnClickListener(this);

        return mView;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        //关闭线程
        threadOn = false;
        writeBuzzerThread = null;
    }
    @Override
    public void onDestroyView() {
        super.onDestroyView();
        //关闭线程
        threadOn = false;
        writeBuzzerThread = null;
    }
```

```
62      /**
63       * 两个按钮的监听事件
64       * @param v 点击的按钮视图
65       */
66      @Override
67      public void onClick(View v) {
68          switch (v.getId()) {
69              case R.id.btn_open://打开蜂鸣器
70                  if (writeBuzzerThread == null) {
71                      threadOn = true;
72                      num = 80;
73                      writeBuzzerThread = new WriteBuzzerThread();
74                      writeBuzzerThread.start();
75                  }
76                  break;
77              case R.id.btn_close://关闭蜂鸣器
78                  if (writeBuzzerThread != null) {
79                      num = 0;
80                      threadOn = false;
81                      writeBuzzerThread.interrupt();
82                      writeBuzzerThread = null;
83                  }
84                  buzzer.operate.write(0);
85                  break;
86              default:
87                  break;
88          }
89      }
90      /**
91      *开启蜂鸣器线程
92      */
93      private class WriteBuzzerThread extends Thread {
94          @Override
95          public void run() {
96              super.run();
97              while (threadOn) {
98                  //发送命令，打开蜂鸣器
99                  buzzer.operate.write(num);
100             }
        }
    }
}
```

3.6 温度传感器实验

3.6.1 【实验目的】

（1） 掌握温度传感器的工作原理；

（2） 掌握对传感器数据进行接受和处理；

（3） 掌握在 Android 系统下的传感器开发流程；

3.6.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

3.6.3 【实验内容】

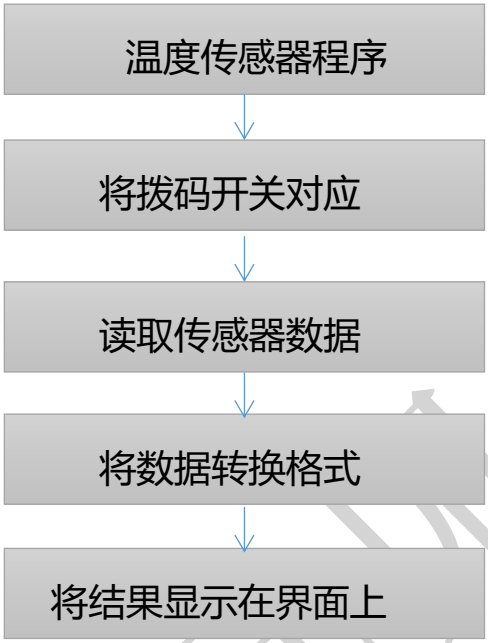
基于 Android5.0 系统实现测量周围环境温度的功能。

3.6.4 【实验原理】

硬件原理参见第二章的温度传感器驱动实验；

在应用程序里通过调用接口读取温度传感器采集到的数据，并将读取到的数据进行运算和格式转换，然后将结果显示在屏幕上，显示内容是当前的环境的温度。

华清远见 dev.hqyj.com

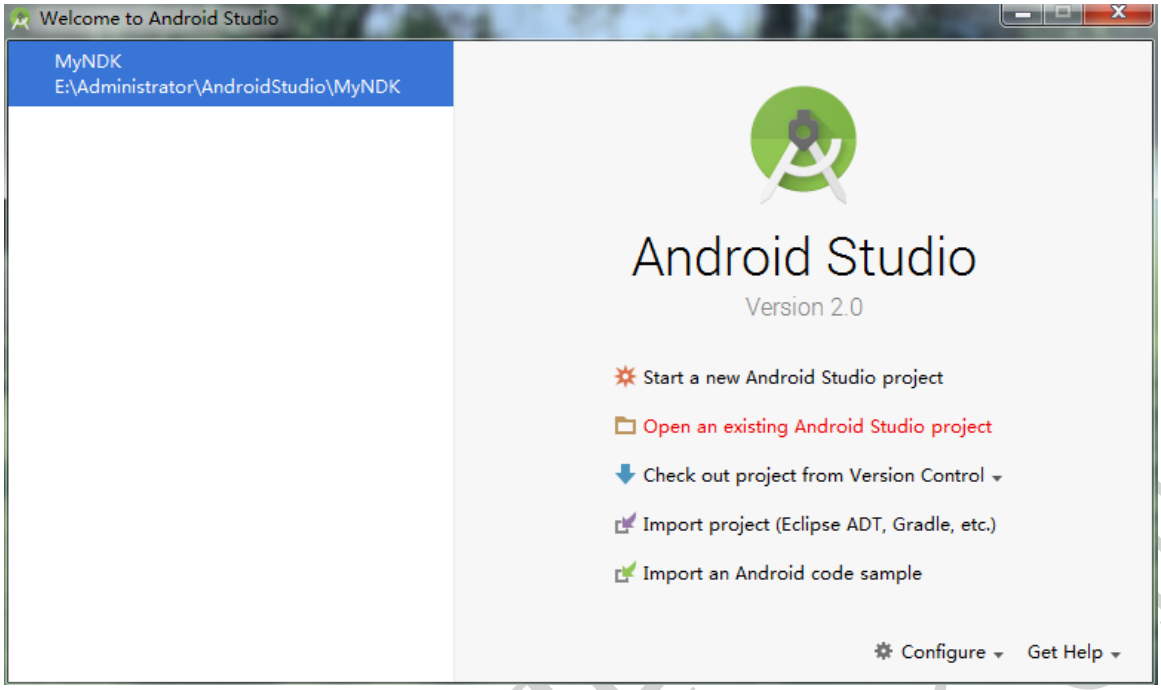


3.6.5 【实验步骤】

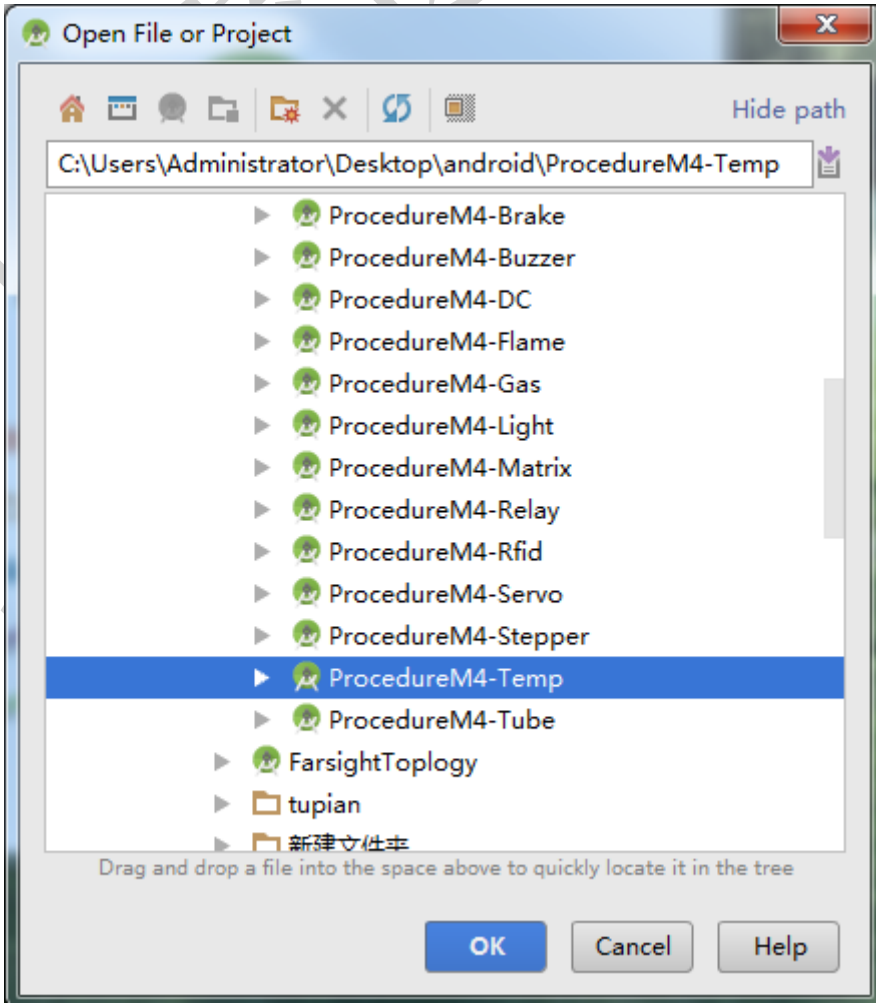
温度实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Temp】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

打开 Android studio，导入实验程序：

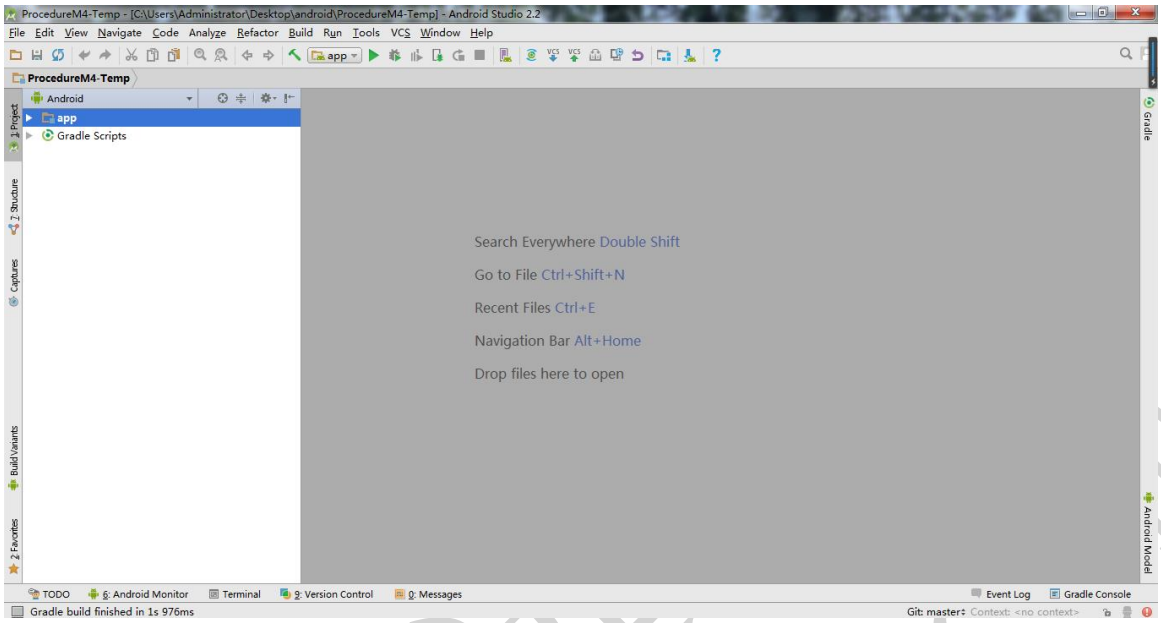
华清远见 dev.hqyj.com



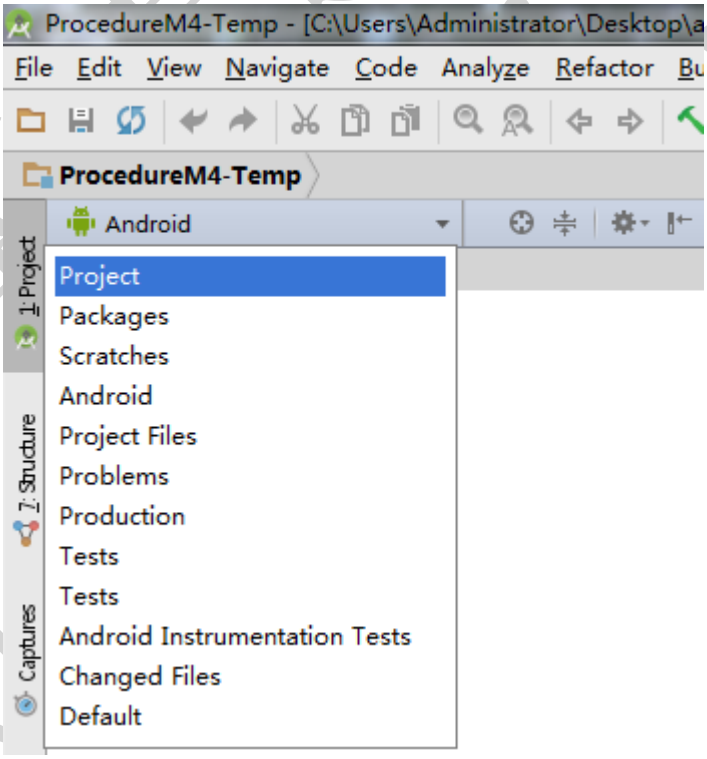
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



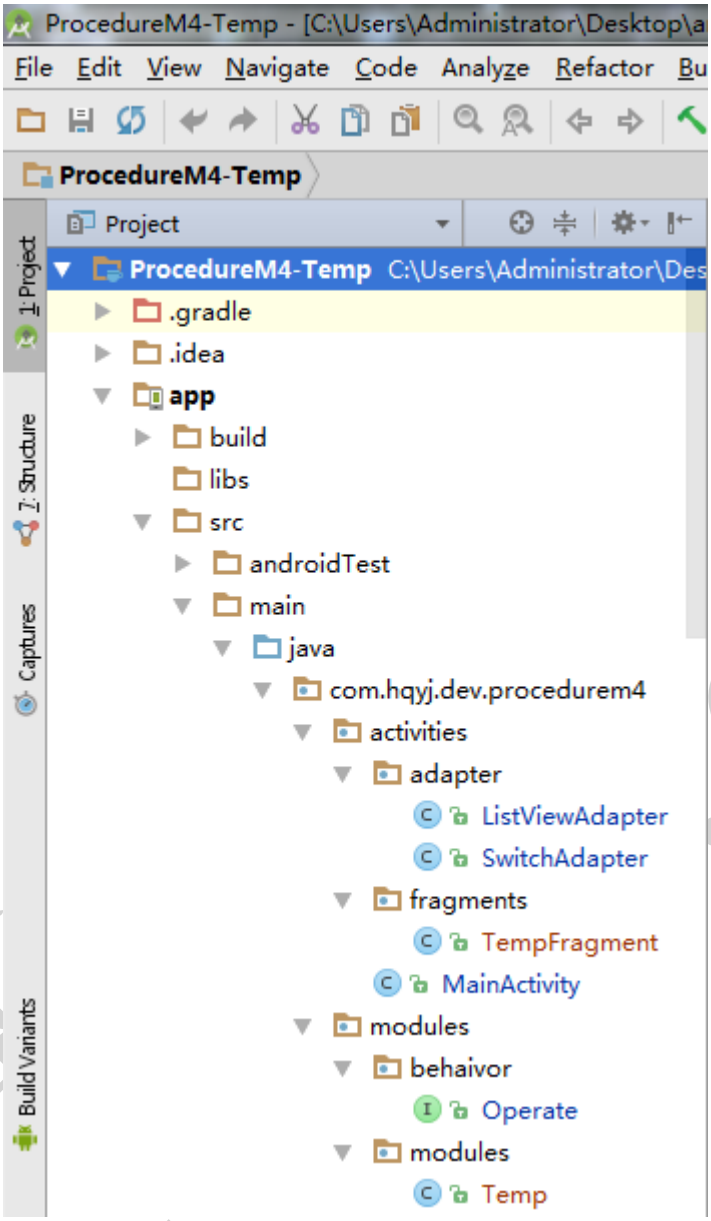
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：

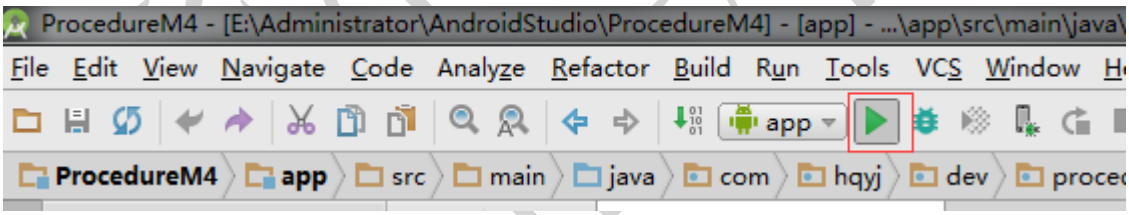


程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

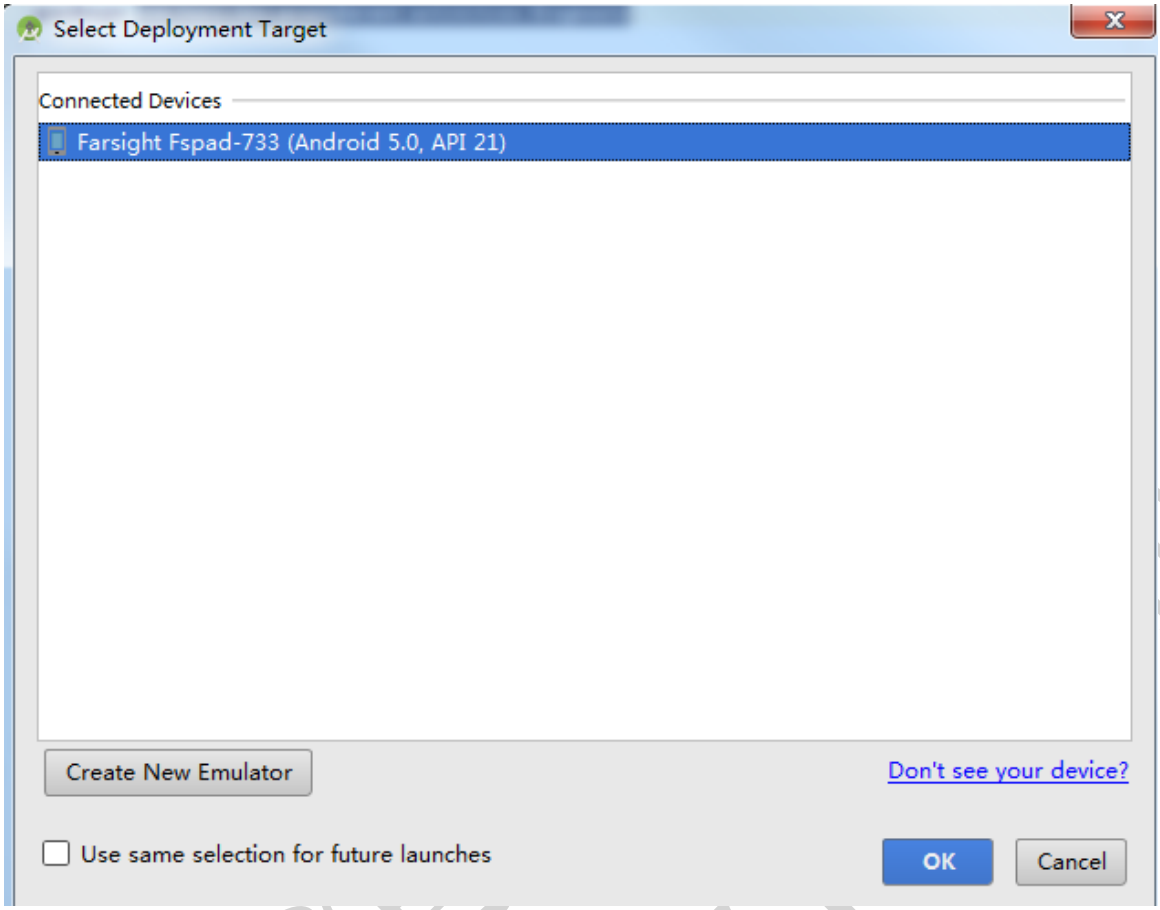
代码写好之后，将代码编译并运行在 4412 板子上。

然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

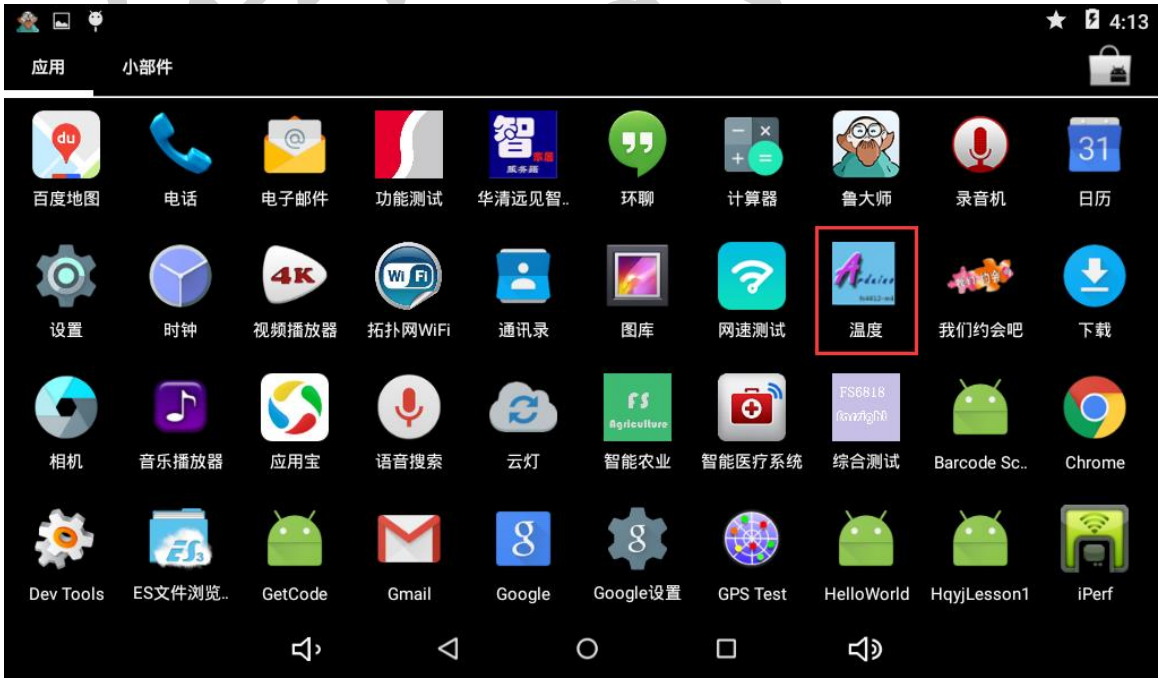
连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

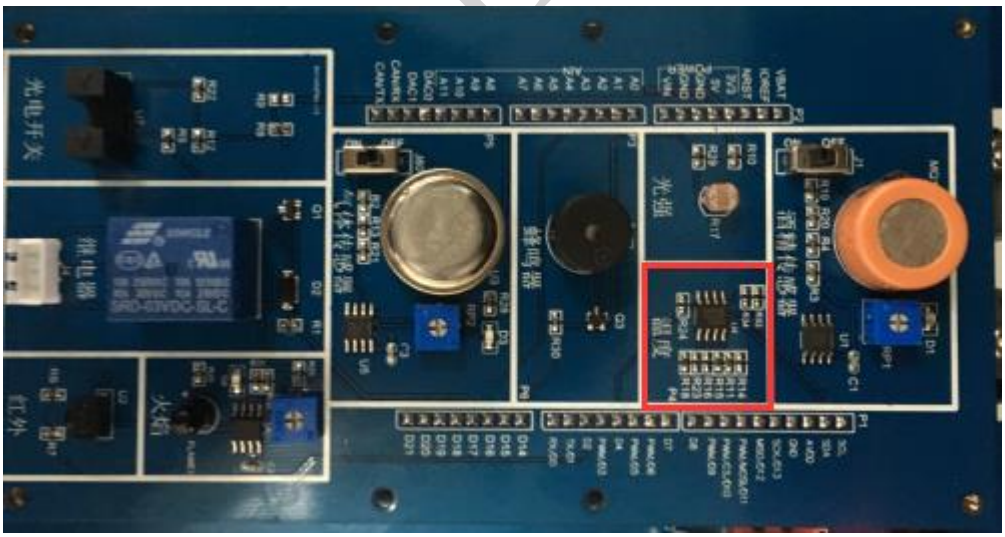


点击进入程序进行测试。

3.6.6 【实验结果】

该实验只需要注意 SW8 拨码拨至 A8/A9，因为在硬件设计时，将 SW8 作为 MCU 和 CUP 的 i2c 主机切换开关，如图中提示。

温度传感器在模块中的位置如下图所示：



3.6.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-3dianzi.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_temp.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 compass 的数据

include $(BUILD_SHARED_LIBRARY)
```

读取 compass 数据的.c 文件

源码位置：ProcedureM4-3dianzi.app.src.main.jni.operate_temp.

NormalText Code

```
#include "operate.h"
#define I2C_RETRIES 0x0701
#define I2C_TIMEOUT 0x0702
#define I2C_RDWR 0x0707

struct i2c_msg
{
    unsigned short addr;
    unsigned short flags;
#define I2C_M_TEN 0x0010
#define I2C_M_RD 0x0001
    unsigned short len;
    unsigned char *buf;
};
struct i2c_rdwr_ioctl_data
{
    struct i2c_msg *msgs;
```

```
18      int nmsgs;
19  };
20
21  int read_temp(){
22      int fd,ret;
23      short temp_val = 0;
24      struct i2c_rdwr_ioctl_data lm75_data;
25      fd=open("/dev/i2c-1",O_RDWR);
26      if(fd<0)
27      {
28          LOGI("open temp failed");
29          return -1;
30      }
31      lm75_data.nmsgs=2;
32      lm75_data.msgs=(struct i2c_msg*)malloc
33          (lm75_data.nmsgs*sizeof(struct i2c_msg));
34      if(!lm75_data.msgs)
35      {
36          LOGI("lm75_data.msgs = false");
37          return -1;
38      }
39      ioctl(fd,I2C_TIMEOUT,1);/*超时时间*/
40      ioctl(fd,I2C_RETRIES,2);/*重复次数*/
41      sleep(1);
42      lm75_data.nmsgs=2;
43      (lm75_data.msgs[0]).len=1; //lm75 目标数据的地址
44      (lm75_data.msgs[0]).addr=0x4f; // lm75 设备地址
45      (lm75_data.msgs[0]).flags=0;//write
46      (lm75_data.msgs[0]).buf=(unsigned char*)malloc(2);
47      (lm75_data.msgs[0]).buf[0]=0x0;//lm75 数据地址
48      (lm75_data.msgs[1]).len=2;//读出的数据
49      (lm75_data.msgs[1]).addr=0x4f;// lm75 设备地址
50      (lm75_data.msgs[1]).flags=I2C_M_RD;//read
51      //存放返回值的地址。
52      (lm75_data.msgs[1]).buf=(unsigned char*)malloc(2);
53      (lm75_data.msgs[1]).buf[0]=0;//初始化读缓冲
54      (lm75_data.msgs[1]).buf[1]=0;//初始化读缓冲
55      ret=ioctl(fd,I2C_RDWR,(unsigned long)&lm75_data);
56      if(ret<0)
57      {
58          LOGI("ret open failed");
59          return -1;
60      }
```

```
    temp_val = (lm75_data.msgs[1]).buf[0] << 8 |
                    (lm75_data.msgs[1]).buf[1];

    if(temp_val >> 15)
        temp_val = (~(temp_val - 0x80) >> 7);
    else
        temp_val = temp_val >> 7;
    LOGI("short %d\n",temp_val);
    close(fd);
    return (int)temp_val;
}
```

（2） Android 端对数据的接受和处理

在具体的温度的 Fragment 中，开启线程读取传感器的数据，然后用 handler 异步处理，对 UI 界面进行动态显示的改变。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.tempFragment.java

NormalText Code

```
public class TempFragment extends Fragment{
    private String TAG = TempFragment.class.getSimpleName();
    private View view;
    private Temp temp;
    private boolean isRun = false;
    private TempThread tempThread;
    private TextView textView;

    static {
        System.loadLibrary("operate");
    }

    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what){
                case 1:
                    String value = msg.getData().getString(TAG);
                    textView.setTextSize(25);
                    textView.setText(String.format("%s", value));
                    break;
```

```
                default:
                    break;
            }
        }
    };
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        temp = Temp.getTemp();
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable
      ViewGroup container, @Nullable Bundle savedInstanceState) {
        view = inflater.inflate(R.layout.temp_fragment, container, false);
        initShow();

        tempThread = new TempThread();
        isRun = true;
        tempThread.start();

        return view;
    }
    private void initShow() {
        textView = (TextView) view.findViewById(R.id.temp);
    }
    @Override
    public void onDestroyView() {
        super.onDestroyView();
        isRun = false;
        tempThread.interrupt();
    }
    private class TempThread extends Thread{
        @Override
        public void run() {
            super.run();
            while (isRun){

                Bundle b = new Bundle();
                Message msg = new Message();

                double value = temp.operate.read()[0];
                Log.d(TAG,"value = " + value);
```

```
                @SuppressLint("DefaultLocale")
                String result = String.format("温度：%.1f ℃", value/2);

                b.putString(TAG, result);
                msg.what = 1;
                msg.setData(b);
                handler.sendMessage(msg);
                try {
                    sleep(1500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

3.7 直流电机实验

3.7.1 【实验目的】

（1） 掌握直流电机的工作原理；

（2） 掌握对控制节点数据进行操控；

（3） 掌握在 Android 系统下的控制节点开发流程；

3.7.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

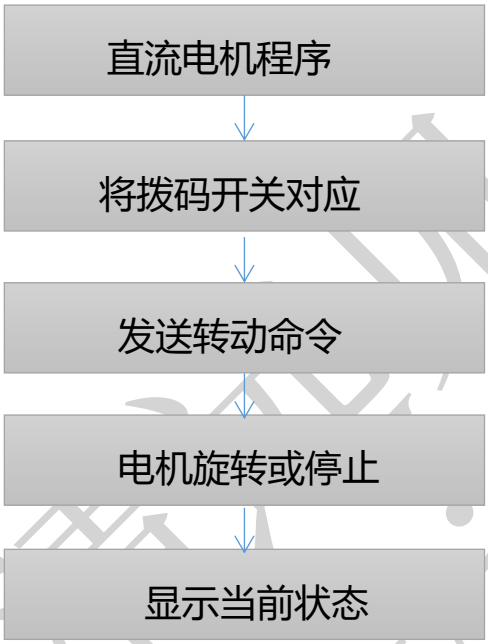
3.7.3 【实验内容】

基于 Android5.0 系统实现对直流电机的控制，控制直流电机的转向等状态。

3.7.4 【实验原理】

硬件原理参见第二章的直流电机驱动实验；

在应用程序里通过调用接口发送命令给直流电机，让其向左、向右或停止，将当前直流电机的状态显示在屏幕上。

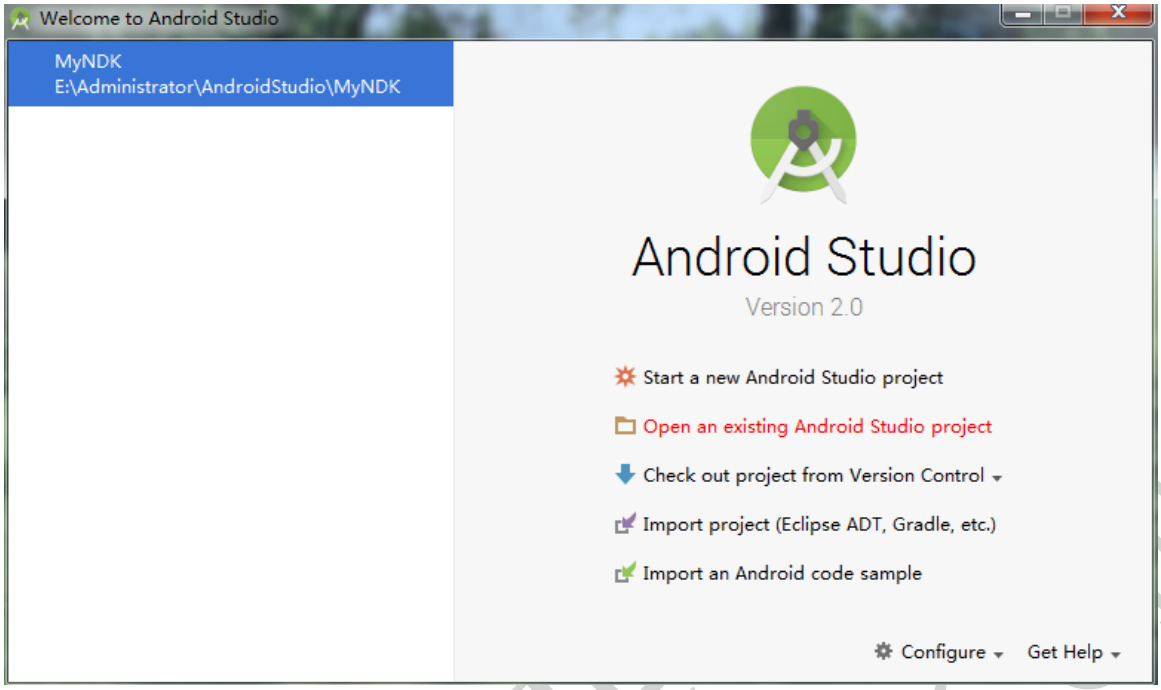


3.7.5 【实验步骤】

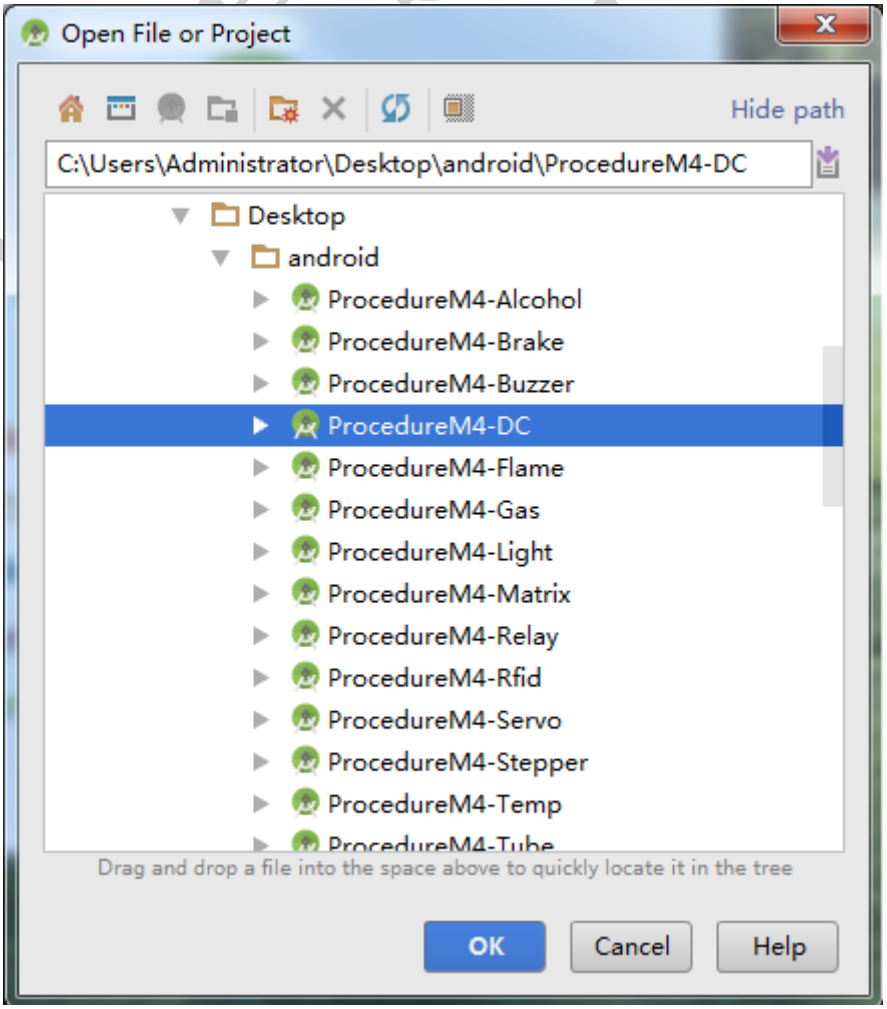
直流电机实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-DC】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下。

打开 Android studio，导入实验程序：

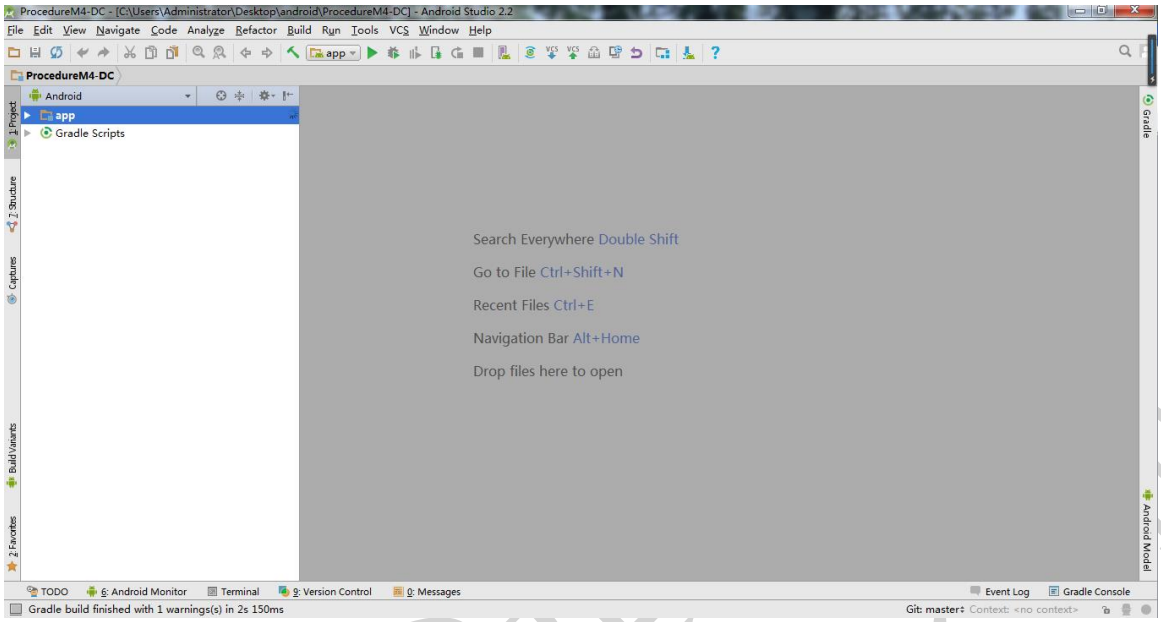
华清远见 dev.hqyj.com



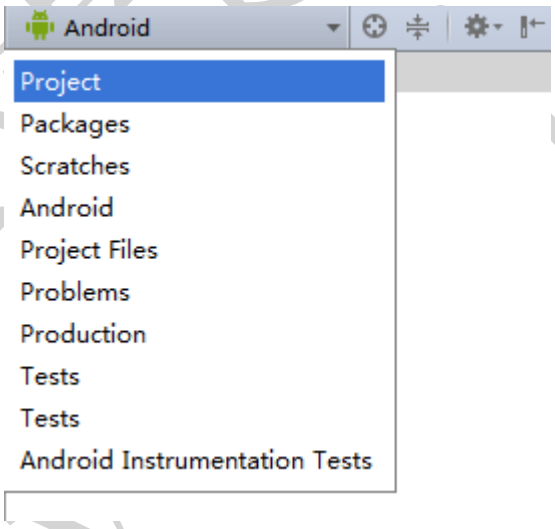
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



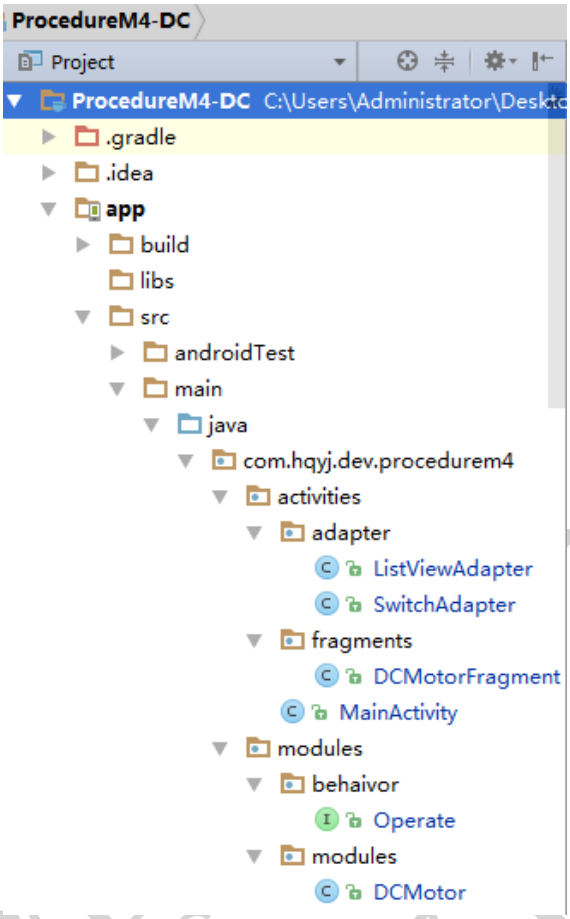
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



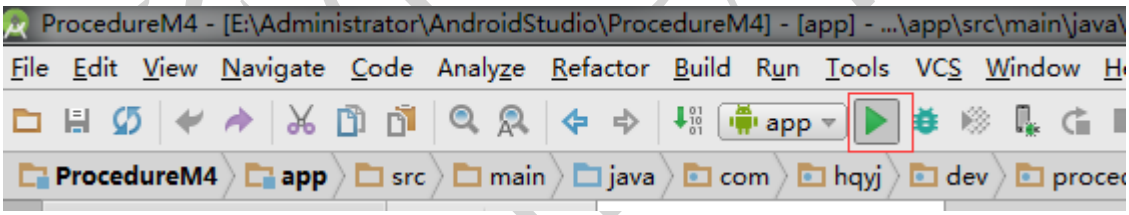
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

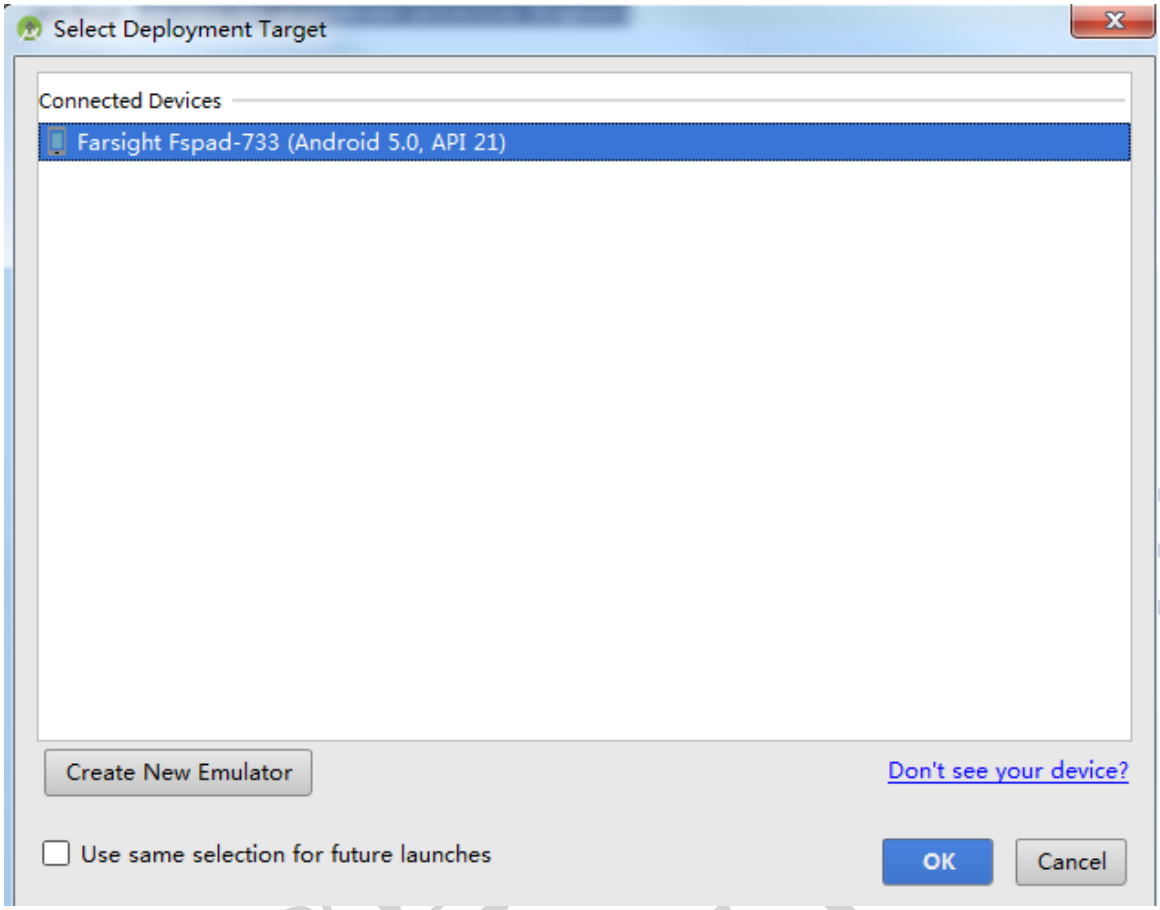
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

华清远见 dev.hqyj.com

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

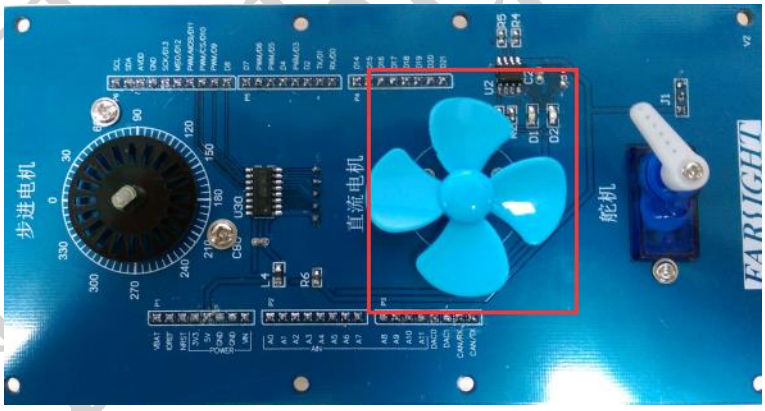
点击进入程序进行测试。

3.7.6 【实验结果】

将拨码开关中的 D12、D13 拨至 ON，其他拨码全为 OFF，如图中提示。

这里有三个按钮，左边的按钮是让步进电机反转，中间是停止按钮，右边是正转按钮。

直流电机在模块中的位置如下图所示：



3.7.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-4zhiliu.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 dc 的数据

include $(BUILD_SHARED_LIBRARY)
```

发送 dc 的控制命令的.c 文件，控制向左、向右、停止转动。

源码位置：ProcedureM4-4zhiliu.app.src.main.jni.operate_dc.c

NormalText Code

```
/**
*停止直流电机的旋转状态
*/
void stop_motor(){
    if(fd == -1){
        fd = open(DC_MOTOR_FILE, O_RDWR);
        if(fd < 0){
            LOGI("OPEN ERROR:  %s", DC_MOTOR_FILE);
            return;
        }
    }
    //左右两边的转动都停止
    ioctl(fd, DC_DYNAMO_OFF, 1);
    ioctl(fd, DC_DYNAMO_OFF, 2);
}
/**
*直流电机向右转
*/
void start_right_motor(){
    if(fd == -1){
        fd = open(DC_MOTOR_FILE, O_RDWR);
        if(fd < 0){
            LOGI("OPEN ERROR:  %s", DC_MOTOR_FILE);
```

```
            return;
        }
    }
    //开启右转，关闭左转
    ioctl(fd, DC_DYNAMO_ON, 1);
    ioctl(fd, DC_DYNAMO_OFF, 2);
    LOGI("#################send left");
}
/**
*直流电机向左转
*/
void start_left_motor(){
    if(fd == -1){
        fd = open(DC_MOTOR_FILE, O_RDWR);
        if(fd < 0){
            LOGI("OPEN ERROR:  %s", DC_MOTOR_FILE);
            return;
        }
    }
    //关闭又转，开启左转
    ioctl(fd, DC_DYNAMO_OFF, 1);
    ioctl(fd, DC_DYNAMO_ON, 2);
    LOGI("#################send right");
}
/**
*关闭流
*/
void close_motor(){
    if(fd != -1){
        close(fd);
        fd = -1;
    }
}
```

（2） Android 端对数据的接受和处理

在具体的直流电机的 Fragment 中，点击命令按钮，然后在 handler 异步处理，向直流电机发送转动的命令，并在 UI 界面进行相应显示出来。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.DCMotorFragment.java

NormalText Code

```
  public class DCMotorFragment extends Fragment {
1
2    private View mView;
```

```
    private DCMotor dcMotor;
    private final String TAG = "DC_MOTOR";
    private DrawDCButton dcButton;
    private TextView textView;
    /**
     *装载名字叫"operate"的库文件
     */
    static {
        System.loadLibrary("operate");
    }
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what){
                case 1:
                    int which = msg.getData().getInt(TAG);
                    Log.d(TAG,""+which);
                    //向直流电机发送命令
                    dcMotor.operate.write(which);

                    switch (which){
                        case 0://提示关闭
                            textView.setText(String.format("%s 按钮被按下","关闭"));
                            break;
                        case 1://提示正转
                            textView.setText(String.format("%s 按钮被按下","正转"));
                            break;
                        case 2://提示反转
                            textView.setText(String.format("%s 按钮被按下","反转"));
                            break;
                        default:
                            break;
                    }
                    break;
            }
        }
    };
    @Override
```

```
46    public void onCreate(@Nullable Bundle savedInstanceState) {
47        super.onCreate(savedInstanceState);
48        dcMotor = DCMotor.getDcMotor();
49    }
50    @Nullable
51    @Override
52    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
53                                                @Nullable Bundle savedInstanceState) {
54        mView = inflater.inflate(R.layout.dc_motor_fragment, container, false);
55        initShow();//初始化按钮和文本
56        dcMotor.operate.init();
57        //设置按钮的状态，监听点击事件
58        dcButton.setState(0);
59        dcButton.invalidate();
60        dcButton.setOnButtonClicked(new DrawDCButton.OnButtonClicked() {
61            //点击按钮，发送消息给 handler，进行异步处理
62            @Override
63            public void onclicked(int which) {
64                Bundle b = new Bundle();
65                b.putInt(TAG, which);
66                Message msg = new Message();
67                msg.what = 1;
68                msg.setData(b);
69                handler.sendMessage(msg);
70            }
71        });
72        return mView;
73    }
74    private void initShow() {
75        dcButton = (DrawDCButton) mView.findViewById(R.id.dc);
76        textView = (TextView) mView.findViewById(R.id.txv_dc);
77    }
78    @Override
79    public void onDestroy() {
80        super.onDestroy();
81        dcButton.setState(3);
82        dcButton.invalidate();
83        dcMotor.operate.write(3);
84    }
85    @Override
86    public void onDestroyView() {
87        super.onDestroyView();
88        dcMotor.operate.write(3);
```

```
89      }
90  }
```

3.8 可燃气传感器实验

3.8.1 【实验目的】

（1） 掌握可燃气传感器的工作原理；

（2） 掌握对传感器数据进行接受和处理；

（3） 掌握在 Android 系统下的传感器开发流程；

3.8.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

3.8.3 【实验内容】

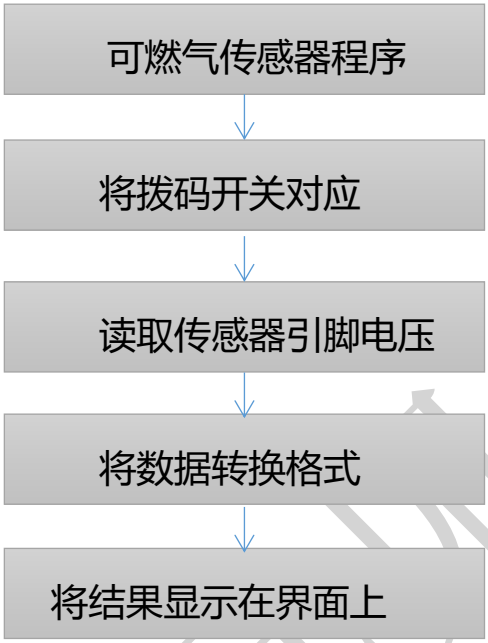
基于 Android5.0 系统实现读取可燃气传感器两个引脚的电压值。

3.8.4 【实验原理】

硬件原理参见第二章的气体传感器驱动实验；

在应用程序里通过调用接口读取可燃气传感器两个引脚的电压，并将读取到的数据进行运算和格式转换，然后将结果显示在屏幕上，显示内容是可燃气传感器两引脚的电压值。

华清远见 dev.hqyj.com

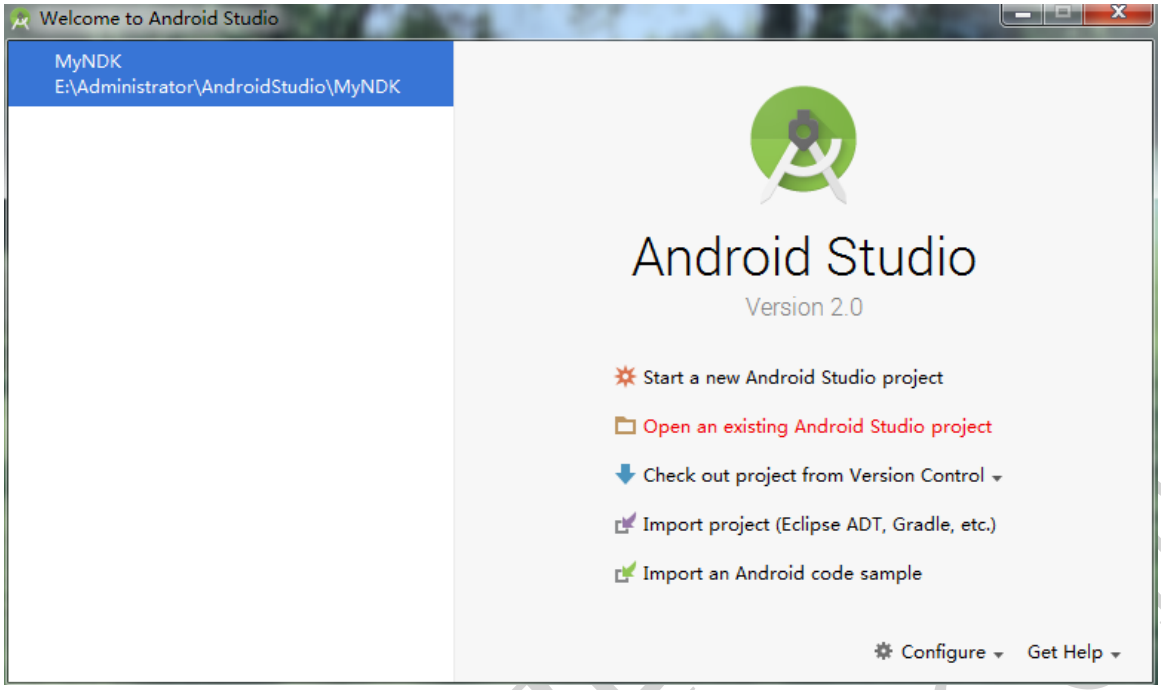


3.8.5 【实验步骤】

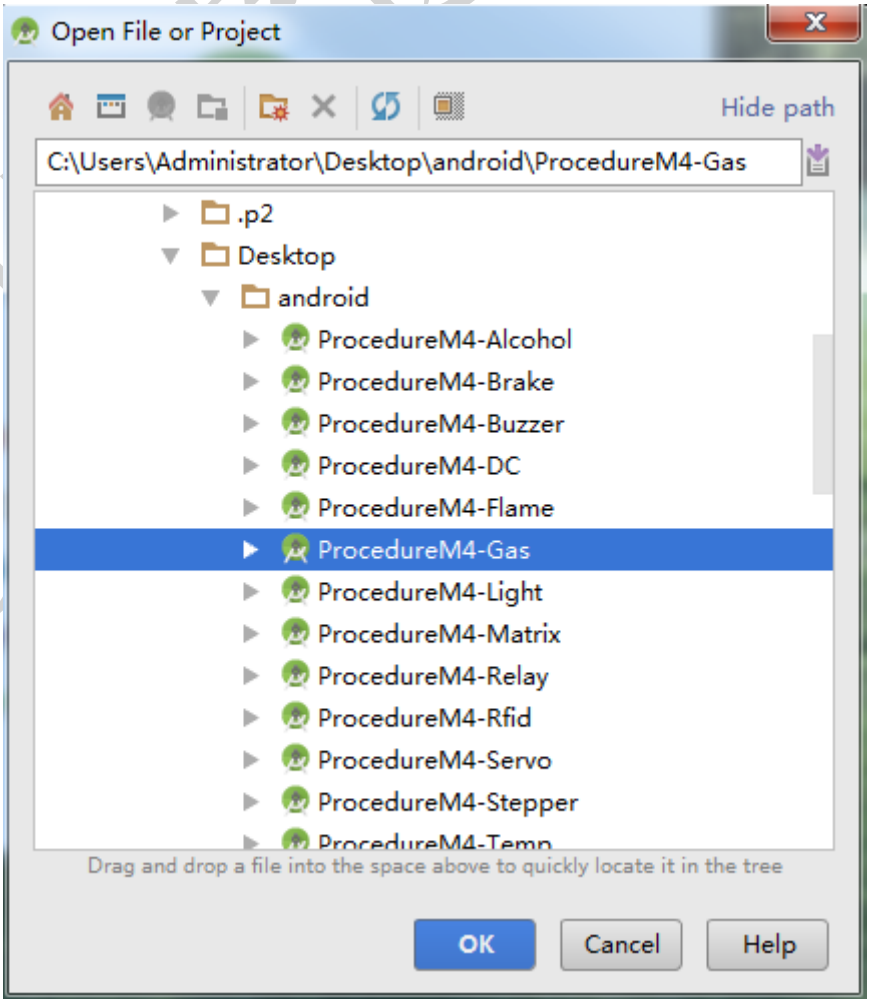
可燃气传感器实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Gas】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

打开 Android studio，导入实验程序：

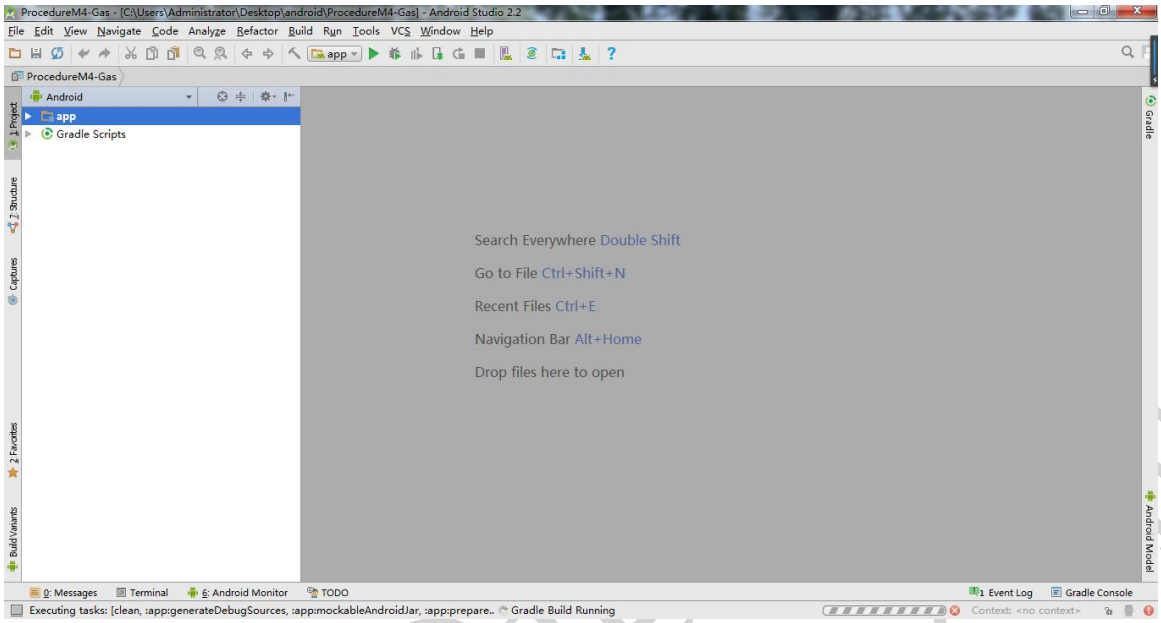
华清远见 dev.hqyj.com



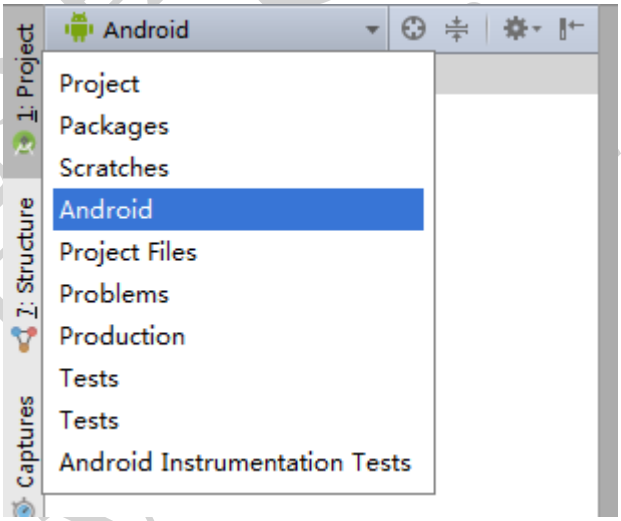
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



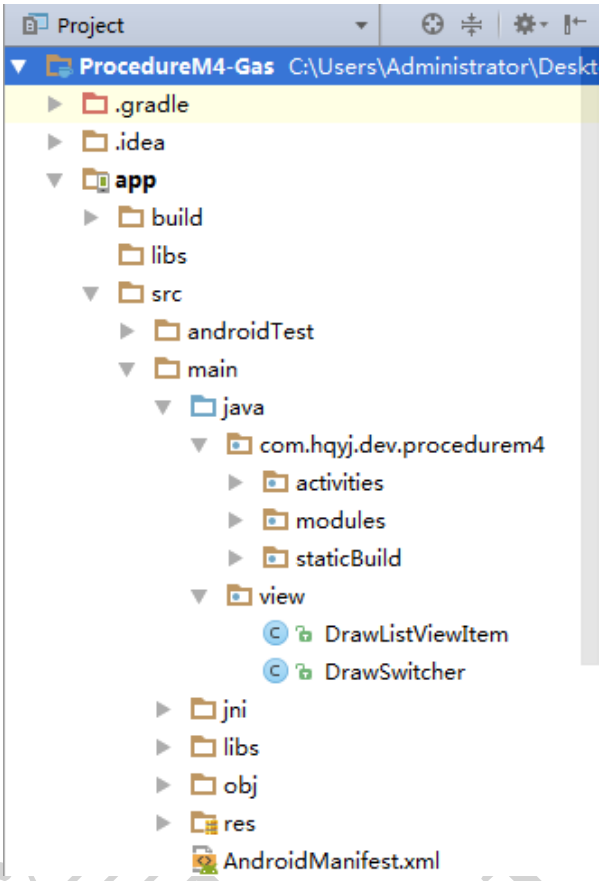
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



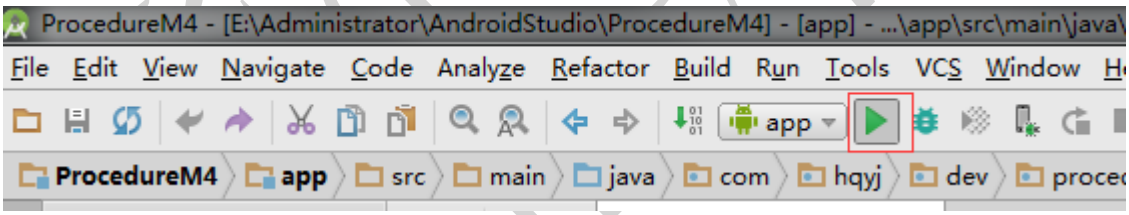
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

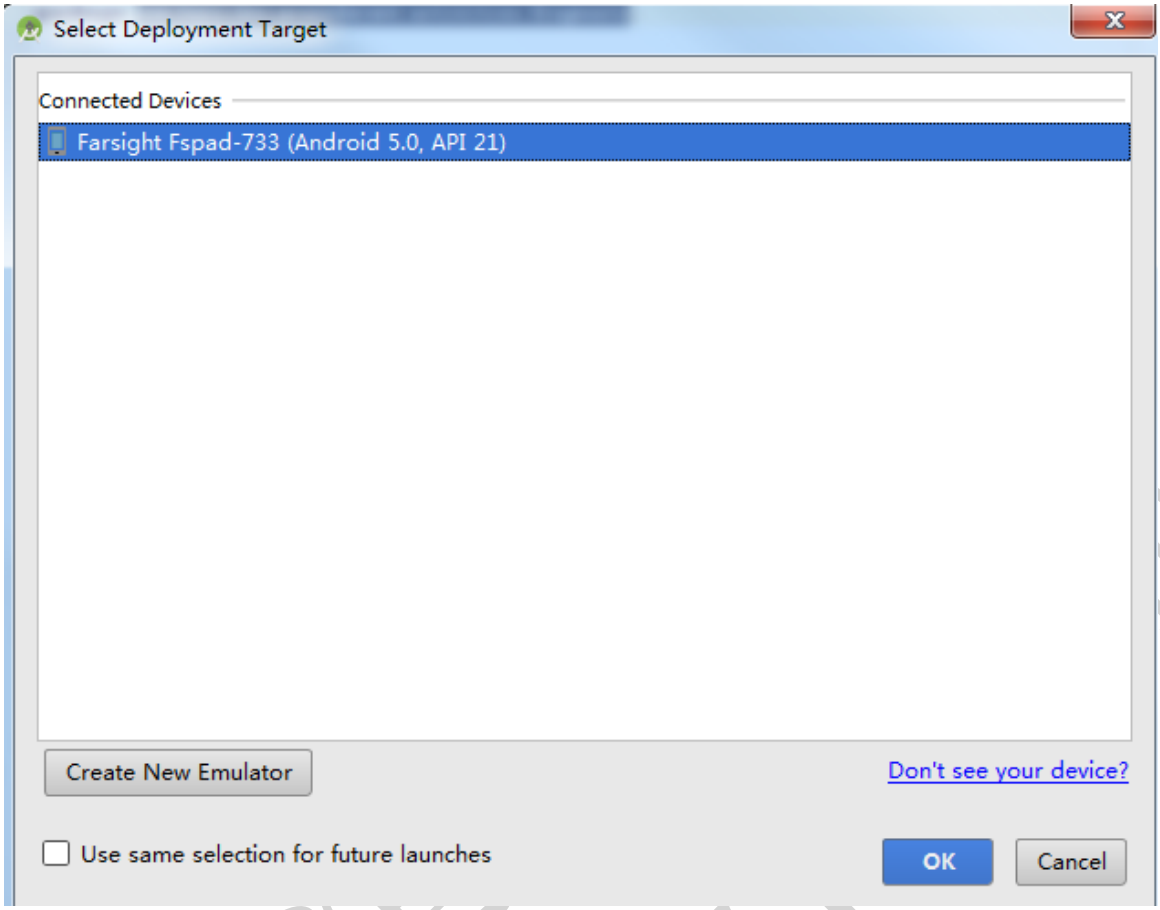
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

华清远见 dev.hqyj.com

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

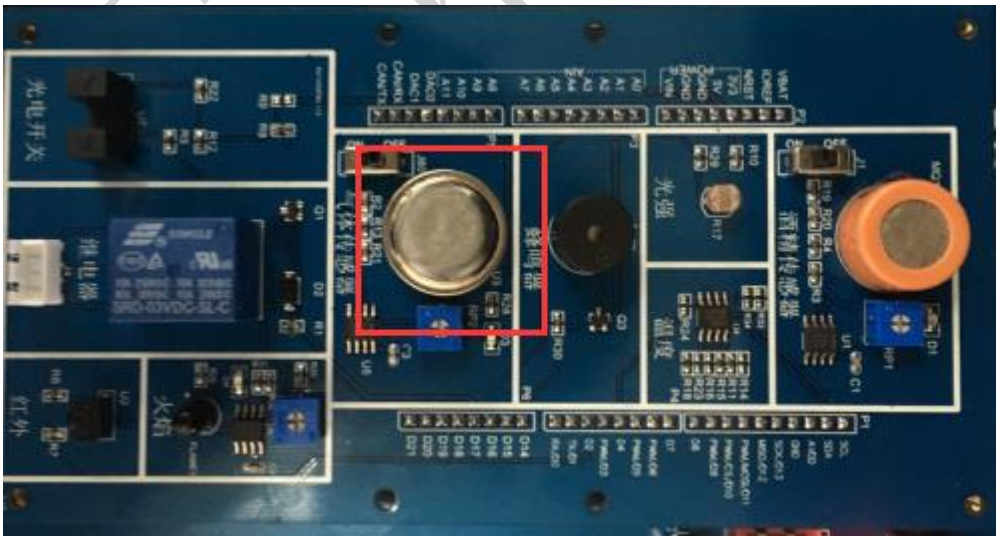
点击进入程序进行测试。

3.8.6 【实验结果】

将拨码开关中的 AD4 拨至 ON，其他拨码全为 OFF，如图中提示。

用打火机的气体测试，当气体浓度越大，值越大，结果如上图所示。

可燃气传感器在模块中的位置如下图所示：



3.8.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-5keran.app.src.main.jni.Android.mk

NormalText Code

```
  LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 adc 的数据

include $(BUILD_SHARED_LIBRARY)
```

读取 adc 数据的.c 文件

源码位置：ProcedureM4-5keran.app.src.main.jni.operate_adc.c

NormalText Code

```
/**
*读取 adc 的数据
*/
int read_adc(int which){
    int fd;
    int value;
    //以可读可写的形式打开"/dev/adc"这个路径下的 adc，创建流通道
    fd = open("/dev/adc", O_RDWR);
    if (fd < 0){
        LOGI("Open ADC error");
        return fd;
    }
    switch(which){
        case ADC_ALCOHOL:
            ioctl(fd, SET_CHANNEL, ALCOHOL_CHANNEL);
            break;
        case ADC_LIGHT:
```

```
            ioctl(fd, SET_CHANNEL, LIGHT_CHANNEL);
            break;
        case ADC_SENSITIVE:
            ioctl(fd, SET_CHANNEL, SENSITIVE_CHANNEL);
            break;
        case ADC_GAS://设置可燃气传感器传输数据通道
            ioctl(fd, SET_CHANNEL, GAS_CHANNEL);
            break;
        default:
            LOGI("ERROR ADC");
            break;
    }
    //读取可燃气体浓度数据
    read(fd, &value, sizeof(value));
    LOGI("value : %d\n", value );
    //关闭流
    close(fd);
    return value;
}
```

（2） Android 端对数据的接受和处理

在具体的可燃气的 Fragment 中，开启线程读取传感器的数据，然后用 handler 异步处理，对 UI 界面进行动态显示的改变。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.GasFragment.java

NormalText Code

```
public class GasFragment extends Fragment {

    private View mView;
    private GasReadThread gasReadThread;
    private boolean threadOn = false;
    private TextView textView;
    private Gas gas;
    private String TAG = "GAS";
    /**
     *装载名字叫"operate"的库文件
     */
    static{
        System.loadLibrary("operate");
    }
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
```

```
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what){
                case 1://显示可燃气体的浓度
                    String value = msg.getData().getString(TAG);
                    textView.setTextSize(50);
                    textView.setText(String.format("%sV", value));
                    break;
                default:
                    break;
            }
        }
    };
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        gas = Gas.getGas();
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
                        container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.gas_fragment, container, false);
        initShow();//初始化控件
        //开启读取数据线程
        threadOn = true;
        gasReadThread = new GasReadThread();
        gasReadThread.start();
        return mView;
    }
    /**
     *
     */
    private void initShow() {
        //...
        textView = (TextView) mView.findViewById(R.id.txv_gas);
    }
    @Override
    public void onDestroy() {
```

```
60          super.onDestroy();
61      }
62      @Override
63      public void onDestroyView() {
64          super.onDestroyView();
65          //关闭线程
66          threadOn = false;
67          gasReadThread.interrupt();
68      }
69      /**
70       * 读取可燃气体浓度的线程
71       */
72      private class GasReadThread extends Thread{
73          @Override
74          public void run() {
75              super.run();
76              while (threadOn) {
77                  Bundle b = new Bundle();
78                  Message msg = new Message();
79                  //将数据显示成"#.##"这种格式
80                  DecimalFormat df = new DecimalFormat("#.##");
81                  //读取数据
82                  int value = gas.operate.read()[0];
83                  String result = df.format((double) value * 7.2 / 4096);
84                  b.putString(TAG, result);
85                  Log.d(TAG, result);
86                  msg.what = 1;
87                  msg.setData(b);
88                  //将数据交给 handler 处理
89                  handler.sendMessage(msg);
90                  try {
91                      sleep(2000);
92                  } catch (InterruptedException e) {
93                      e.printStackTrace();
94                  }
95              }
96          }
97      }
98  }
```

3.9 光照传感器实验

3.9.1 【实验目的】

（1） 掌握光照传感器的工作原理；

（2） 掌握对传感器数据进行接受和处理；

（3） 掌握在 Android 系统下的传感器开发流程；

3.9.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

3.9.3 【实验内容】

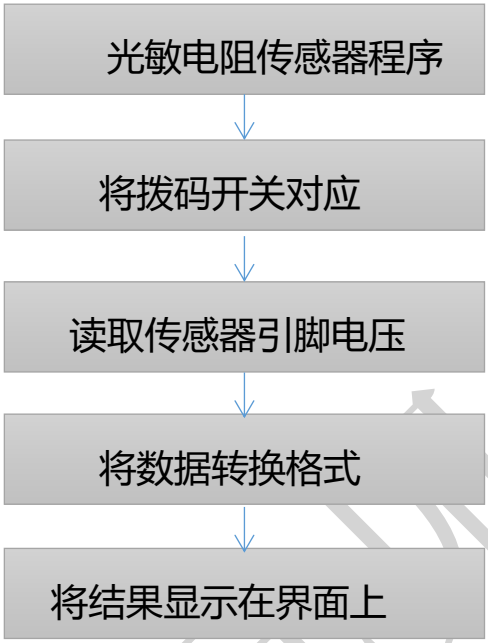
基于 Android5.0 系统实现读取光敏电阻两个引脚的电压值。

3.9.4 【实验原理】

硬件原理参见第二章的光敏传感器驱动实验；

在应用程序里通过调用接口读取光敏电阻两端电压，并将读取到的数据进行运算和格式转换，然后将结果显示在屏幕上，显示内容是光敏电阻两引脚的电压。

华清远见 dev.hqyj.com

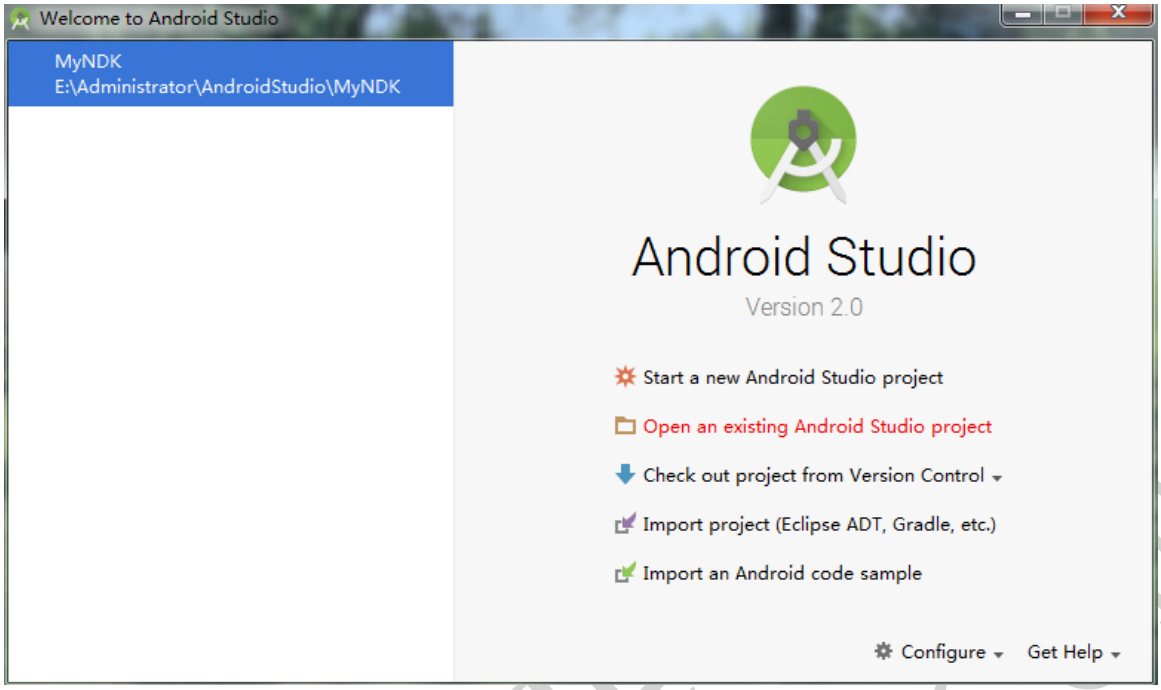


3.9.5 【实验步骤】

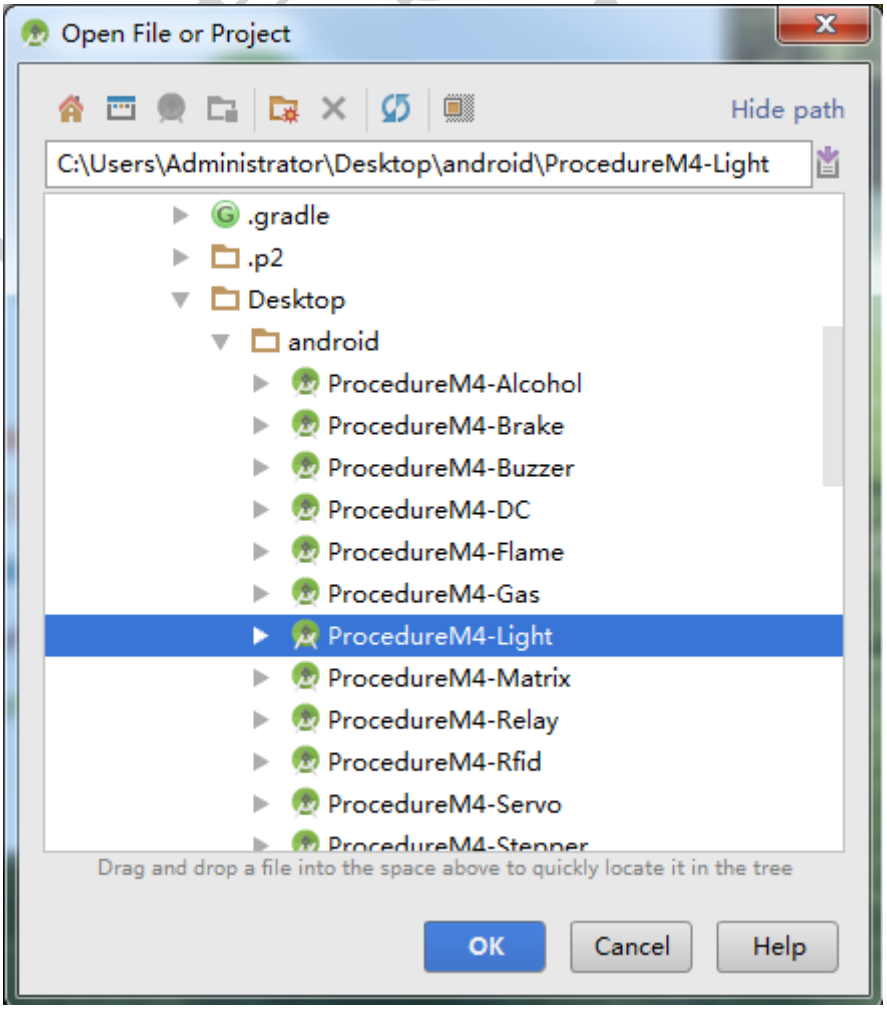
光照传感器实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Light】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

打开 Android studio，导入实验程序：

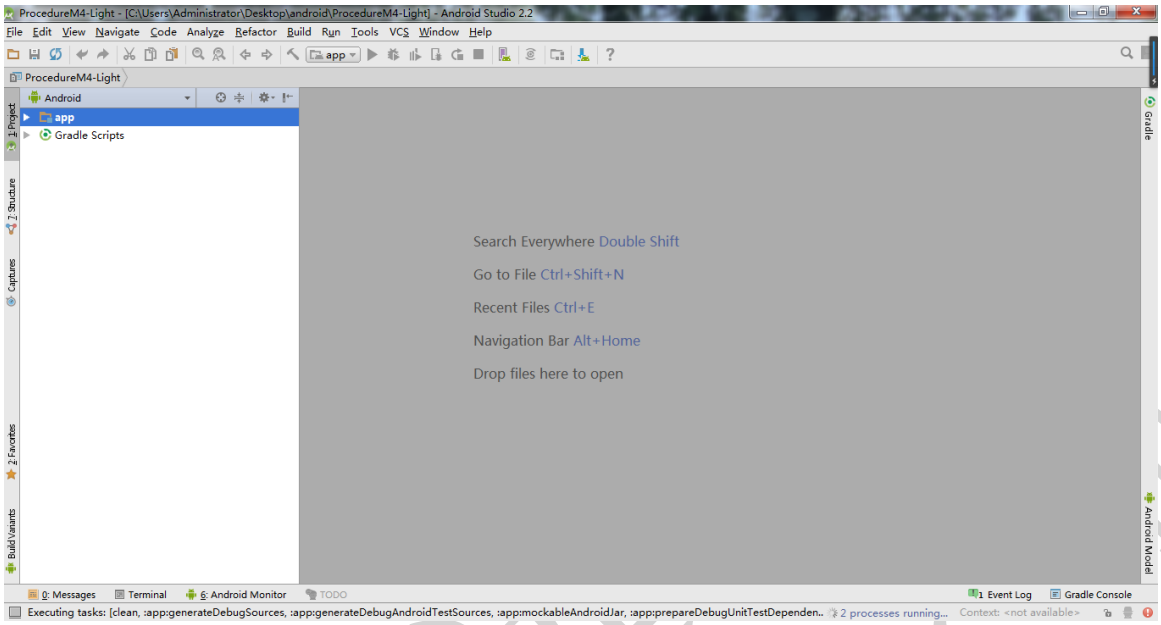
华清远见 dev.hqyj.com



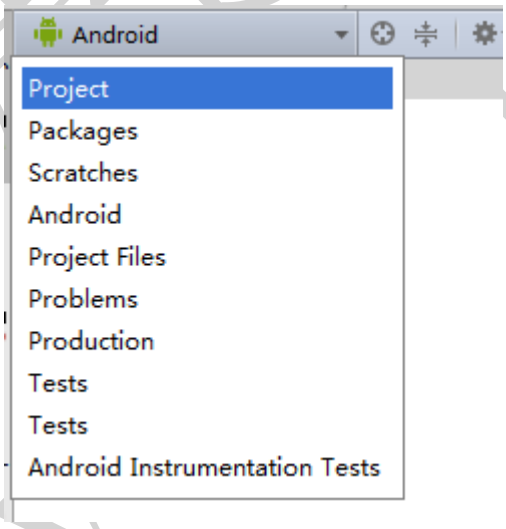
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



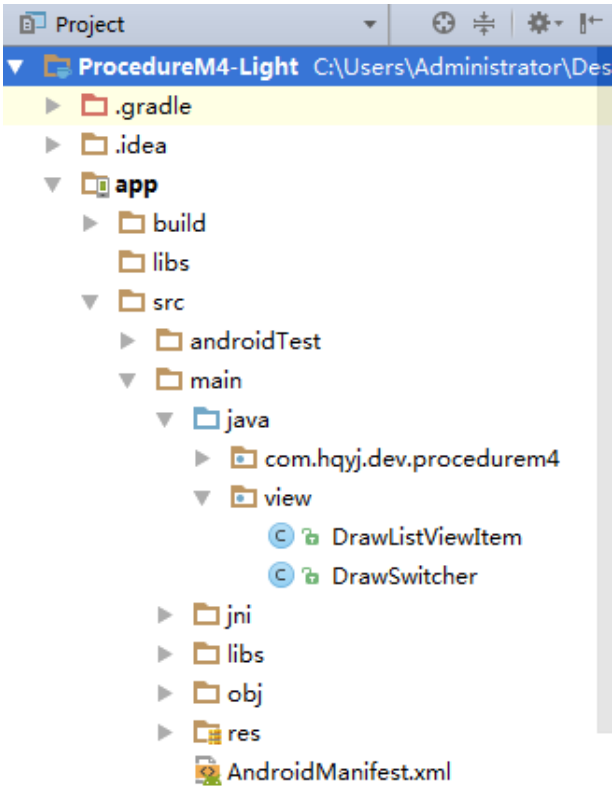
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



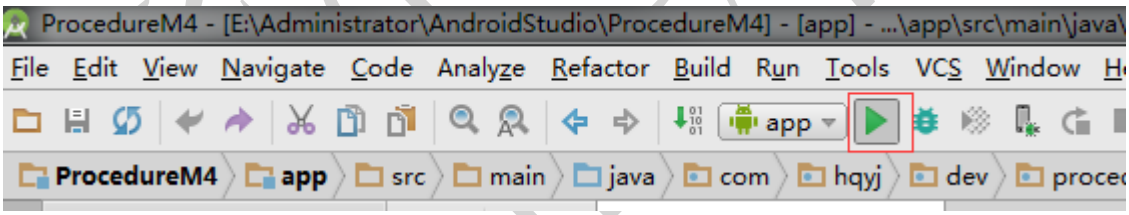
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

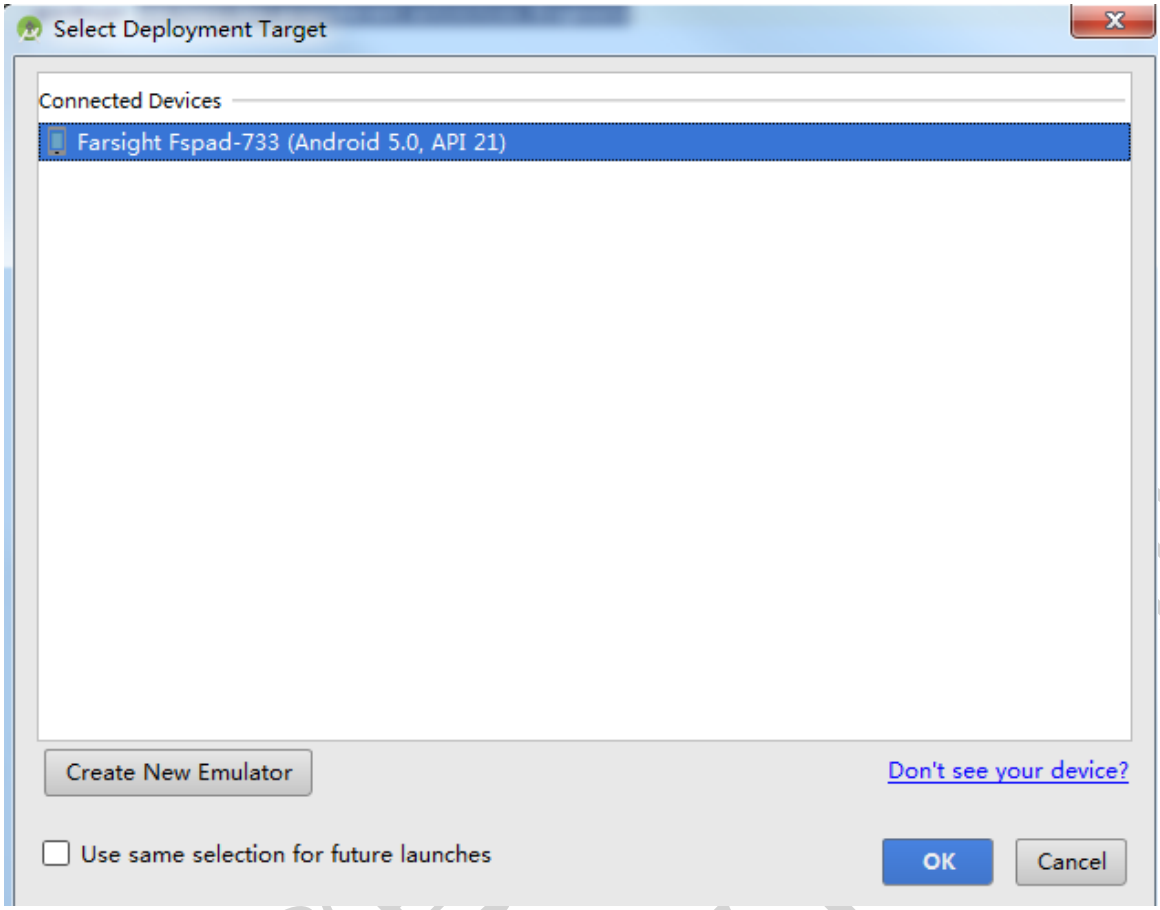
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

华清远见 dev.hqyj.com

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

点击进入程序进行测试。

3.9.6 【实验结果】

将拨码开关中的 AD2 拨至 ON，其他拨码全为 OFF，如图中提示。

周围关照强度越高，显示数字越大，如上图所示。

光照传感器在模块中的位置如下图所示：

3.9.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-6guangzhao.app.src.main.jni.Android.mk

NormalText Code

```
  LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 adc 的数据

include $(BUILD_SHARED_LIBRARY)
```

读取 adc 数据的.c 文件

源码位置：ProcedureM4-6guangzhao.app.src.main.jni.operate_adc.c

NormalText Code

```
/**
*读取 adc 数据
*/
int read_adc(int which){
    int fd;
    int value;
    //以可读可写的形式打开"/dev/adc"这个路径下的 adc，创建流通道
    fd = open("/dev/adc", O_RDWR);
    if (fd < 0){
        LOGI("Open ADC error");
        return fd;
    }
    switch(which){
        case ADC_ALCOHOL:
            ioctl(fd, SET_CHANNEL, ALCOHOL_CHANNEL);
            break;
        case ADC_LIGHT://设置光照传感器传输数据通道
```

```
            ioctl(fd, SET_CHANNEL, LIGHT_CHANNEL);
            break;
        case ADC_SENSITIVE:
            ioctl(fd, SET_CHANNEL, SENSITIVE_CHANNEL);
            break;
        case ADC_GAS:
            ioctl(fd, SET_CHANNEL, GAS_CHANNEL);
            break;
        default:
            LOGI("ERROR ADC");
            break;
    }
    //读取光照强度数据
    read(fd, &value, sizeof(value));
    LOGI("value : %d\n", value );
    //关闭流
    close(fd);
    return value;
}
```

（2） Android 端对数据的接受和处理

在具体的光照的 Fragment 中，开启线程读取传感器的数据，然后用 handler 异步处理，对 UI 界面进行动态显示的改变。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.LightFragment.java

NormalText Code

```
 public class LightFragment extends Fragment {

    private View mView;
    private boolean threadOn = false;
    private LightReadThread lightReadThread;
    private TextView txvLight;
    private Light light;
    private final String TAG = "LIGHT";
    /**
     *装载名字叫"operate"的库文件
     */
    static{
        System.loadLibrary("operate");
    }
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
```

```
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what){
                case 1://将处理过的数据在界面上显示出来
                    String value = msg.getData().getString(TAG);
                    txvLight.setTextSize(50);
                    txvLight.setText(String.format("%sV", value));
                    break;
                default:
                    break;
            }
        }
    };
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        light = Light.getLight();
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
                        container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.light_fragment, container, false);
        initShow();//初始化控件
        //开启读取光照强度的线程
        lightReadThread = new LightReadThread();
        threadOn = true;
        lightReadThread.start();
        return mView;
    }
    private void initShow() {
        txvLight = (TextView) mView.findViewById(R.id.txv_light);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
    }
    @Override
    public void onDestroyView() {
```

```
60          super.onDestroyView();
61          threadOn = false;//关闭线程
62          lightReadThread.interrupt();
63      }
64      /**
65       * 读取光照强度线程
66       */
67      private class LightReadThread extends Thread {
68          @Override
69          public void run() {
70              super.run();
71              while (threadOn){
72                  Bundle b = new Bundle();
73                  Message msg = new Message();
74                  //将数据显示成"#.##"这种格式
75                  DecimalFormat df = new DecimalFormat("#.##");
76                  //读取光照强度
77                  int value = light.operate.read()[0];
78                  String result = df.format((double)value*7.2/4096);
79                  b.putString(TAG, result);
80                  msg.what = 1;
81                  msg.setData(b);
82                  //将数据交给 handler 处理
83                  handler.sendMessage(msg);
84                  try {
85                      sleep(2000);
86                  } catch (InterruptedException e) {
87                      e.printStackTrace();
88                  }
89              }
90          }
91      }
92  }
```

3.10 矩阵键盘实验

3.10.1 【实验目的】

（1） 掌握矩阵键盘的工作原理；

（2） 掌握对控制节点数据进行操控；

（3） 掌握在 Android 系统下的控制节点开发流程；

3.10.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

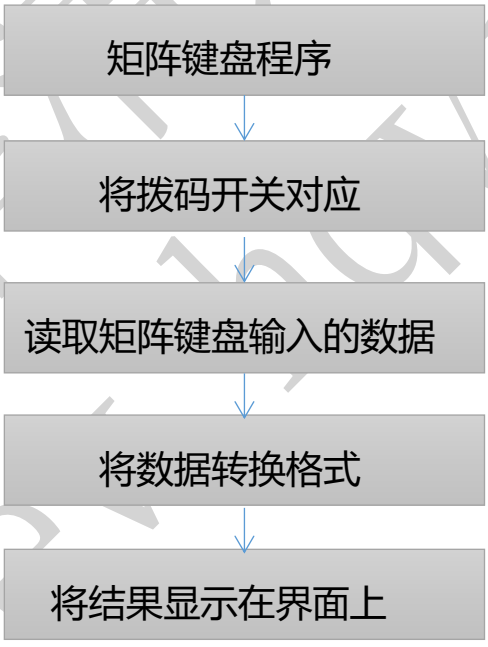
3.10.3 【实验内容】

基于 Android5.0 系统实现在矩阵键盘上输入，在 Android 界面上显示。

3.10.4 【实验原理】

硬件原理参见第二章的按键扫描数码管显示驱动实验；

在应用程序里通过调用接口读取矩阵键盘输入的数据，并将读取到的数据进行运算和格式转换，然后将结果显示在屏幕上。

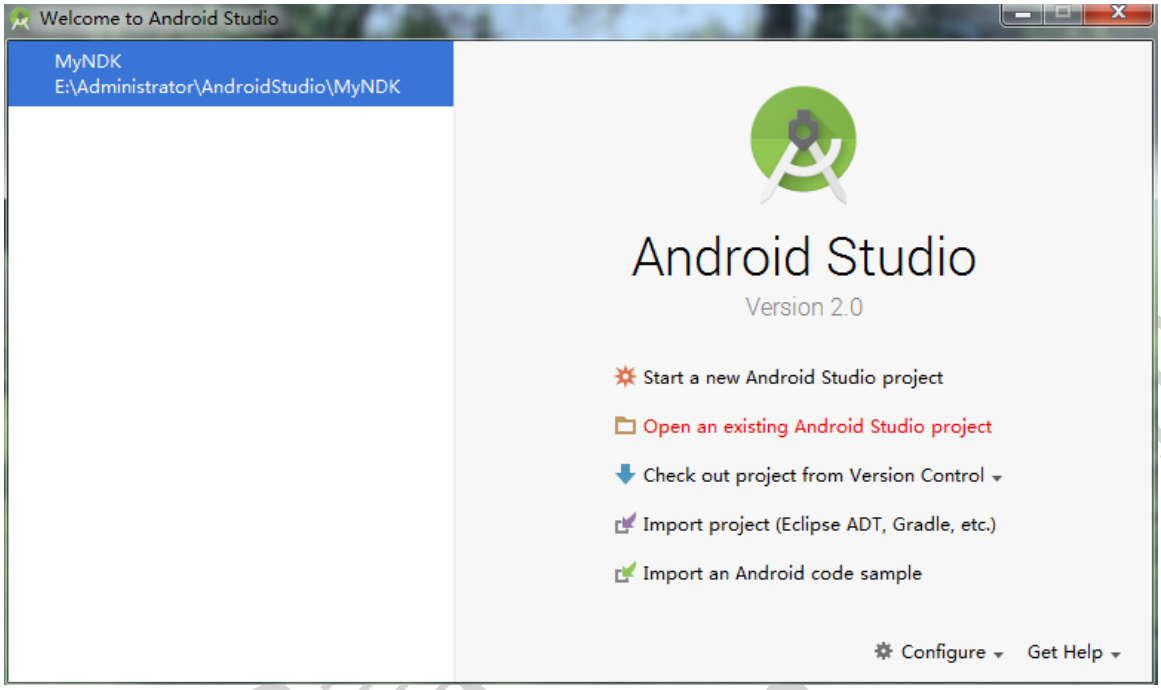


3.10.5 【实验步骤】

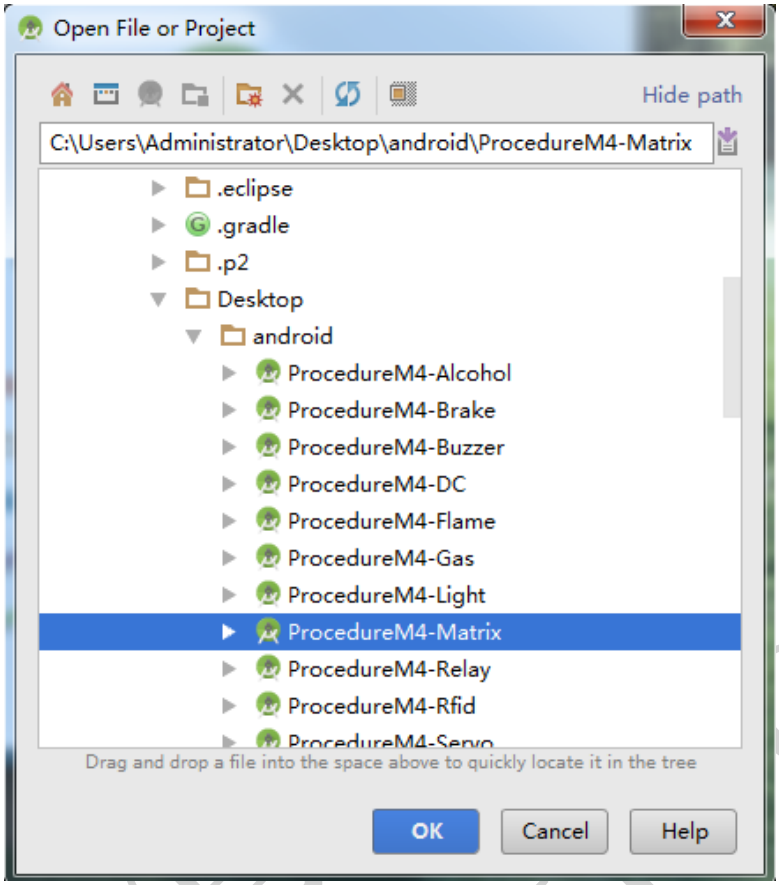
矩阵键盘实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码

\ProcedureM4-Matrix】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

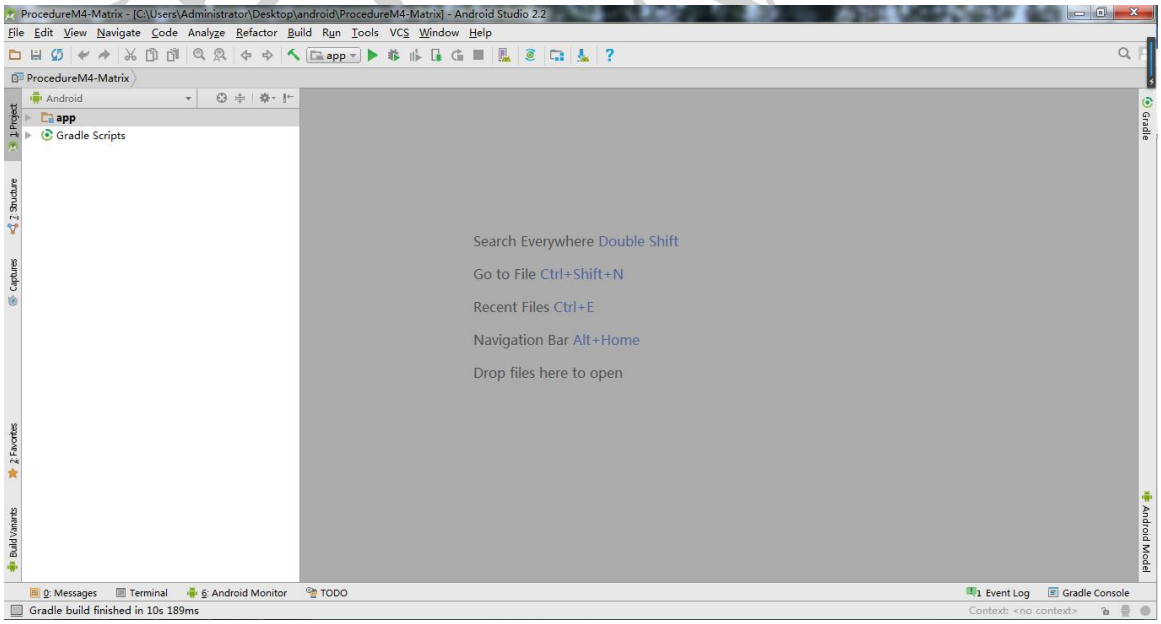
打开 Android studio，导入实验程序：



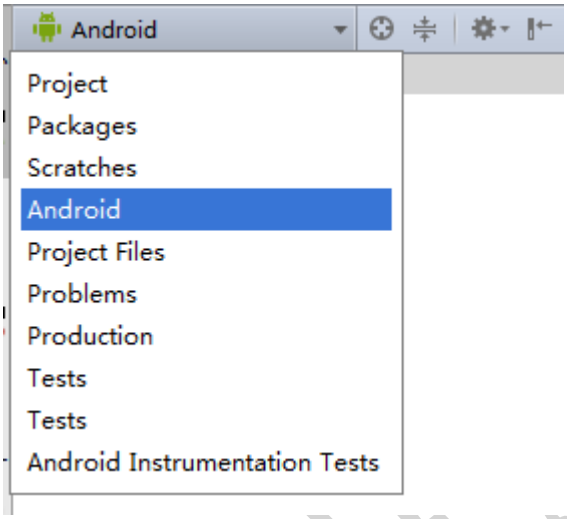
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



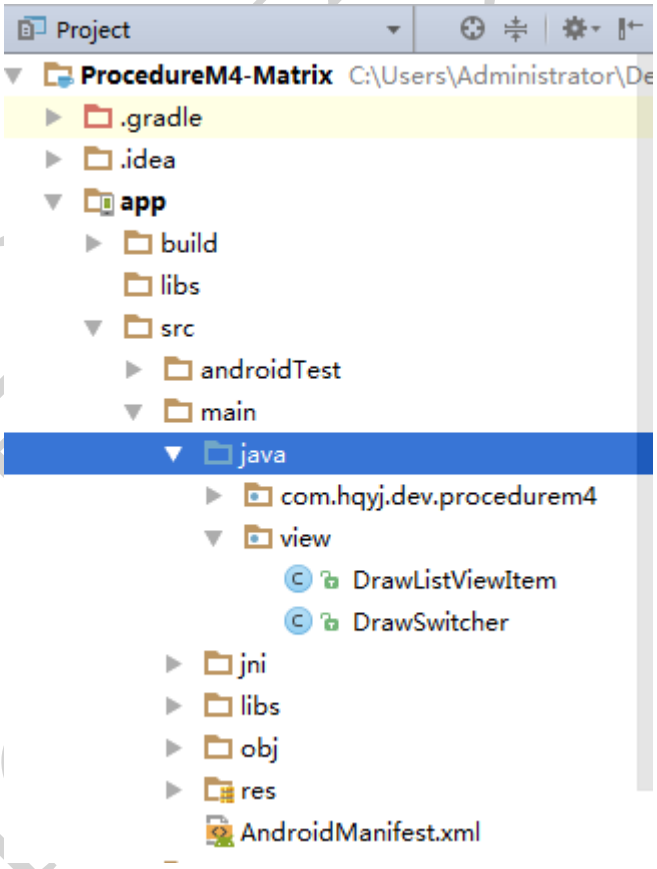
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



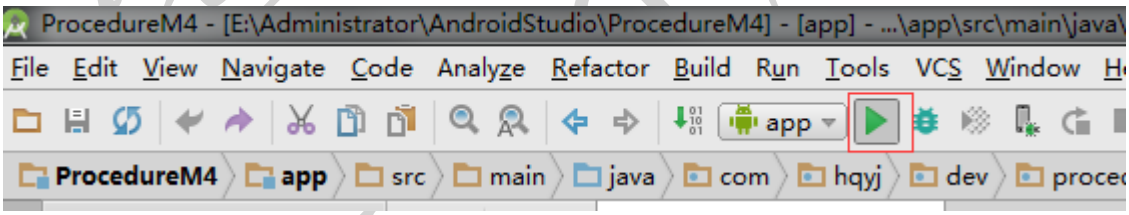
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

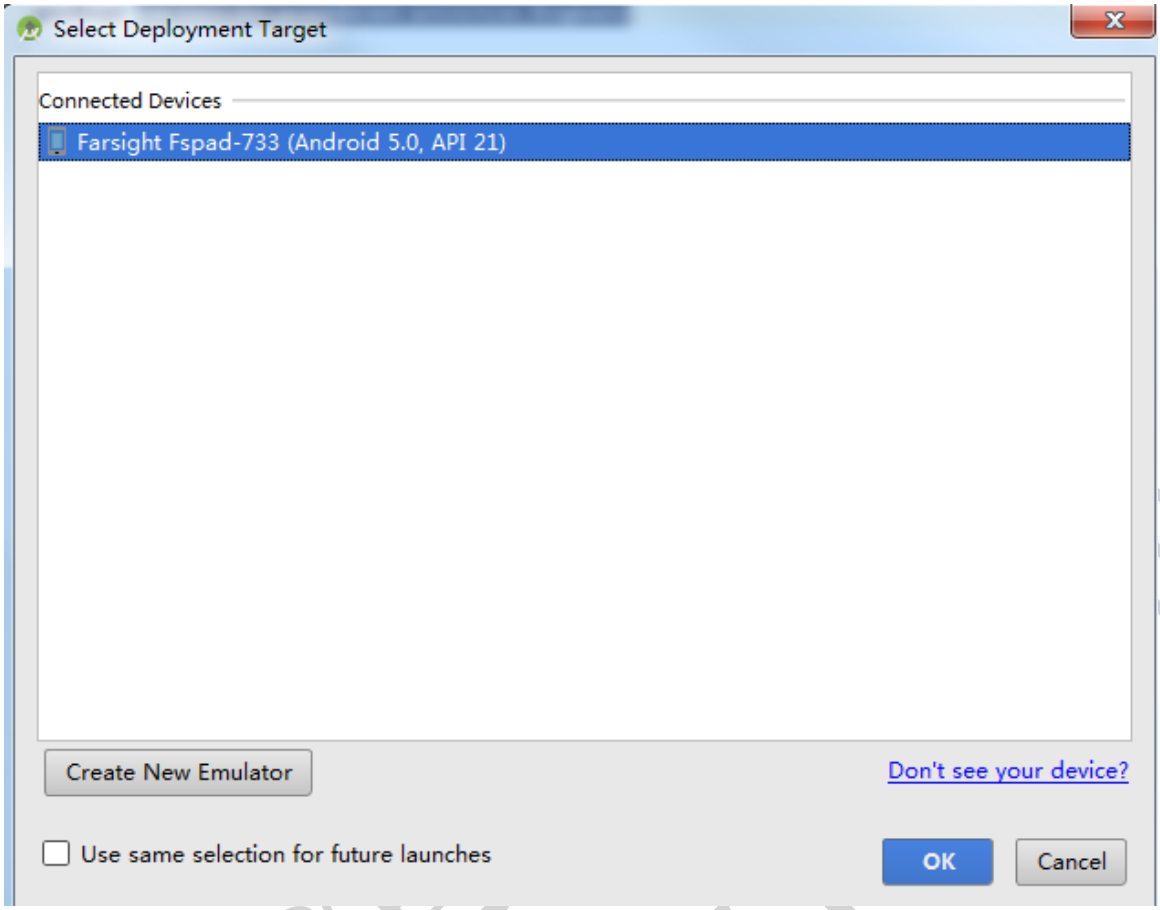
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。

如下图：

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

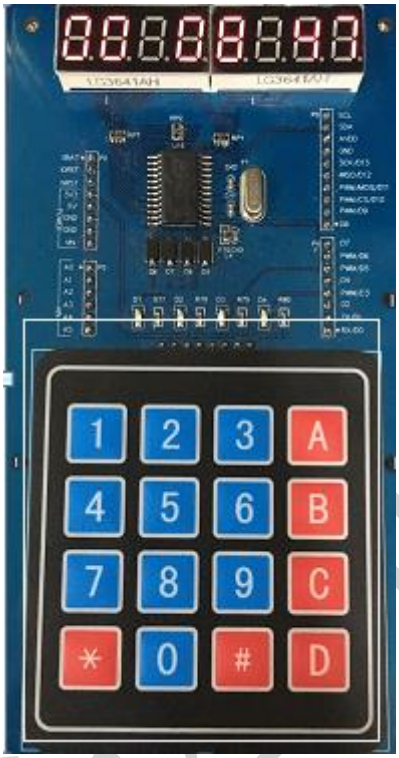
点击进入程序进行测试。

3.10.6 【实验结果】

将拨码开关中的 D7 拨至 ON，其他拨码全为 OFF，如图中提示。

在键盘上输入的数字会显示在屏幕上面，如上图所示“等待输入。。。”

矩阵键盘在模块中的位置如下图所示：



3.10.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-7juzhen.app.src.main.jni.Android.mk

NormalText Code

```
   LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 zlg 的数据

include $(BUILD_SHARED_LIBRARY)
```

读取 zlg 数据的.c 文件

源码位置：ProcedureM4-7juzhen.app.src.main.jni.operate_zlg.c

NormalText Code

```
/**
*读取矩阵键盘键入的值
*/
void read_zlg(int value[2]){

    find_zlg();//
    //以只读的形式打开 file_path 这个路径下的 zlg，创建流通道
    int fd = open(file_path, O_RDONLY);
    if(fd < 0){
        value = NULL;
    } else{
        //读取数据
        read(fd, &ev, sizeof(ev));
        switch(ev.type){
            case EV_KEY:
                value[0] = ev.code;
                value[1] = ev.value;
                LOGI("value[0]: %d", value[0]);
                LOGI("value[1]: %d", value[1]);
            break;
            default:
                value = NULL;
            break;
        }
    }
    close(fd);//关闭流
}
/**
*找到矩阵输入的数据
*/
int find_zlg(){
    int fd;
    int i = 0;
    for(i = 0; i < 10; i++){
        char path[128];
        char buf[128];
        bzero(path, sizeof(path));
        sprintf(path, "/sys/class/input/input%d/name", i);

        if((fd = open(path, O_RDONLY)) < 0){
            continue;
```

```
        }
        int ret = 0;
        bzero(buf, sizeof(buf));
        ret = read(fd, buf, sizeof(buf));
        LOGI("buf = %s", buf);
        bzero(file_path, sizeof(file_path));
        if(strncmp(ZLG_NAME, buf, 7) == 0){
            sprintf(file_path, "/dev/input/event%d", i);
            LOGI("file_path ======= %s", file_path);
            close(fd);
            return 0;
        }else{
            close(fd);
            continue;
        }
        if(i == 9){
            close(fd);
            return -1;
        }
    }
    close(fd);
    return 0;
}
```

（2） Android 端对数据的接受和处理

在具体的矩阵键盘的 Fragment 中，开启线程读取传感器的数据，得到数据之后再转换为对应矩阵键盘上的数字，然后用 handler 异步处理，对 UI 界面进行动态显示的改变。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.MatrixFragment.java

NormalText Code

```
public class MatrixFragment extends Fragment {

    private View mView;
    private boolean threadOn = false;
    private MatrixReadThread matrixReadThread;
    private TextView textView;
    private String string;
    private String TAG = "Matrix";
    private Matrix matrix;
    /**
     *装载名字叫"operate"的库文件
     */
```

```
    static {
        System.loadLibrary("operate");
    }
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what) {
                case 1://将接收到的数据在界面上显示出来
                    String value = msg.getData().getString(TAG);
                    textView.setTextSize(50);
                    textView.setText(String.format("%s", value));
                    break;
                default:
                    break;
            }
        }
    };
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        matrix = Matrix.getMatrix();
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
                        container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.matrix_fragment, container, false);
        initShow();//初始化控件
        threadOn = true;
        matrixReadThread = new MatrixReadThread();
        matrixReadThread.start();//开启线程
        return mView;
    }
    private void initShow() {
        textView = (TextView) mView.findViewById(R.id.txv_matrix);
    }
    @Override
    public void onDestroy() {
```

```
56          super.onDestroy();
57      }
58      @Override
59      public void onDestroyView() {
60          super.onDestroyView();
61          //关闭线程
62          threadOn = false;
63          matrixReadThread.interrupt();
64          matrixReadThread = null;
65          string = null;
66      }
67      /**
68       * 读取矩阵键盘输入的数据
69       */
70      private class MatrixReadThread extends Thread {
71          @Override
72          public void run() {
73              super.run();
74              while (threadOn) {
75                  Bundle b = new Bundle();
76                  Message msg = new Message();
77                  Log.d(TAG, "1");
78                  //读取矩阵键盘输入的数据
79                  int[] value = matrix.operate.read();
80                  Log.d(TAG, "2");
81                  if (value != null && value[0]!=0 && value[1]!=0) {
82                      String valueKey = getKey(value[0]);
83                      Log.d(TAG, valueKey);
84                      if (!valueKey.equalsIgnoreCase( "null")) {
85                          if (string == null) {
86                              string = getKey(value[0]);
87                          } else {//前后按的按钮字符串会叠加起来
88                              string = string + getKey(value[0]);
89                          }
90                          msg.what = 1;
91                          b.putString(TAG, string);
92                          Log.d(TAG, string);
93                          msg.setData(b);
94                          //将数据交给 handler 处理
95                          handler.sendMessage(msg);
96                      }
97                      //如果数据超过 15 字节，就重新设置为 null
98                      if (string!=null &&string.length() >= 15)
```

```
                        string = null;
                }
            }
        }
    }
    /**
     * 将数字转化为相对应的字符串
     * @param key 传入的数字
     * @return 数字所对应矩阵键盘上的字符串
     */
    @SuppressLint("DefaultLocale")
    private String getKey(int key){
        String value;
        switch (key){
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
            case 10:
                value = String.format("%d", key-1);
                break;
            case 11:
                value = String.format("%d", 0);
                break;
            case 30:
                value = "A";
                break;
            case 48:
                value = "B";
                break;
            case 46:
                value = "C";
                break;
            case 32:
                value = "D";
                break;
            case 522:
                value = "*";
                break;
```

```
            case 523:
                value = "#";
                break;
            default:
                value = "null";
                break;
        }
        return value;
    }
}

```

3.11 继电器实验

3.11.1 【实验目的】

（1） 掌握继电器的工作原理；

（2） 掌握对控制节点数据进行操控；

（3） 掌握在 Android 系统下的控制节点开发流程；

3.11.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

3.11.3 【实验内容】

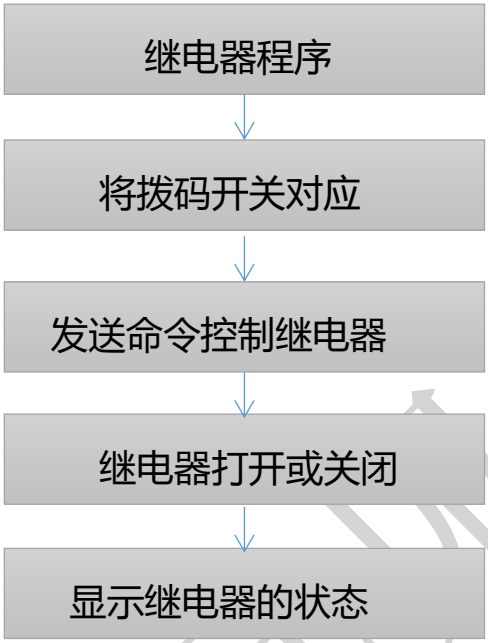
基于 Android5.0 系统实现控制继电器通断。

3.11.4 【实验原理】

硬件原理参见第二章的继电器驱动实验；

在应用程序里通过调用接口发送打开或关闭继电器的命令，然后将继电器状态显示在屏幕上。

华清远见 dev.hqyj.com

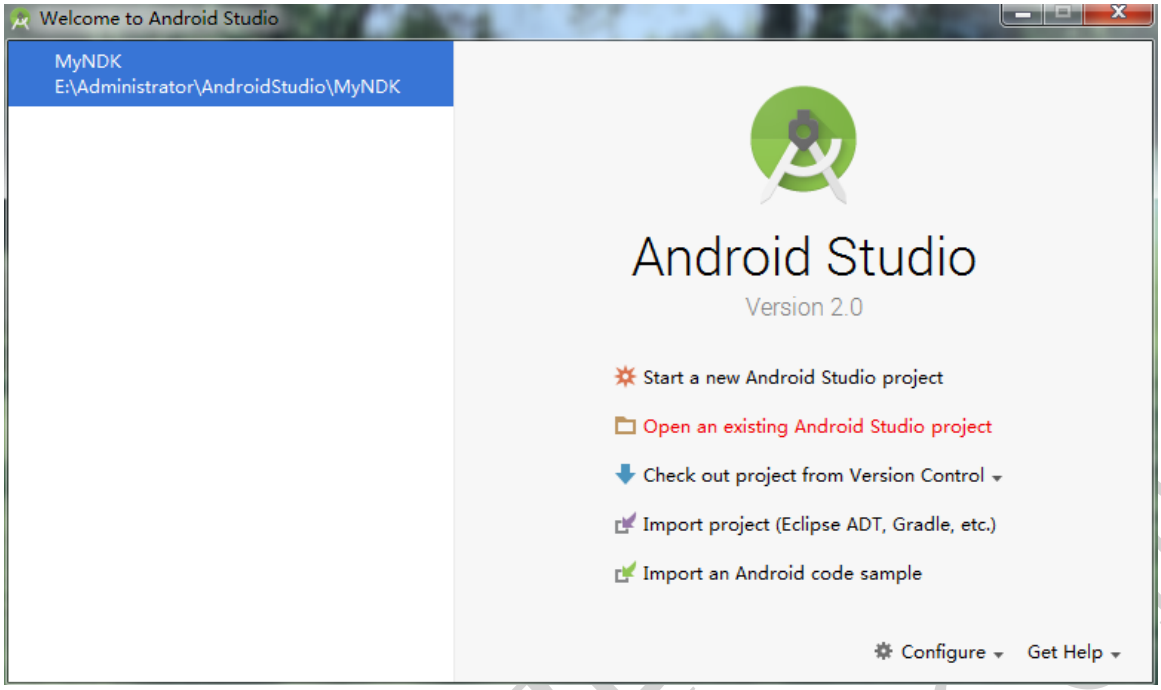


3.11.5 【实验步骤】

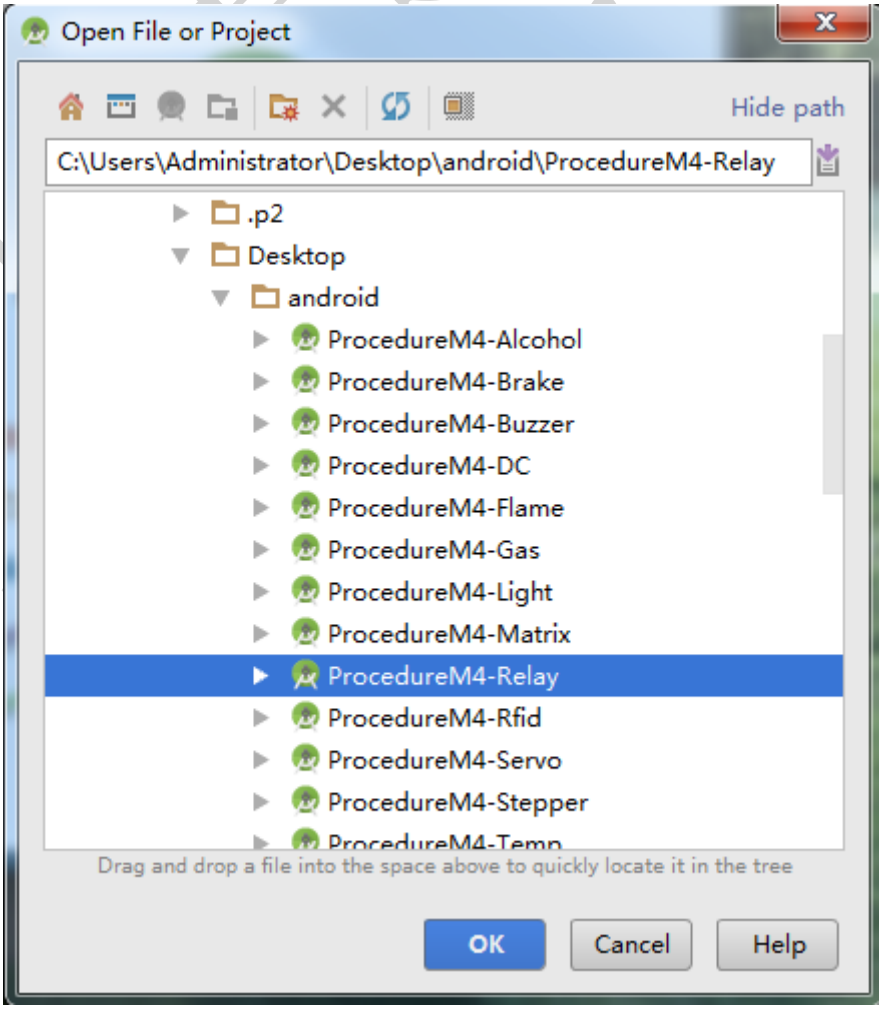
继电器实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Relay】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

打开 Android studio，导入实验程序：

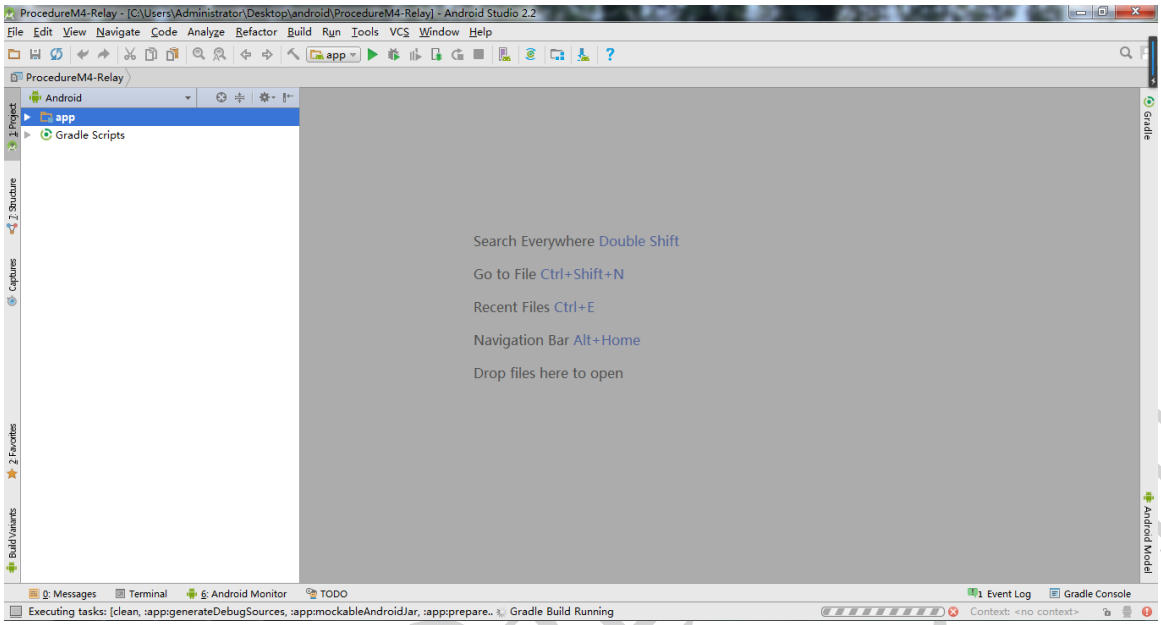
华清远见 dev.hqyj.com



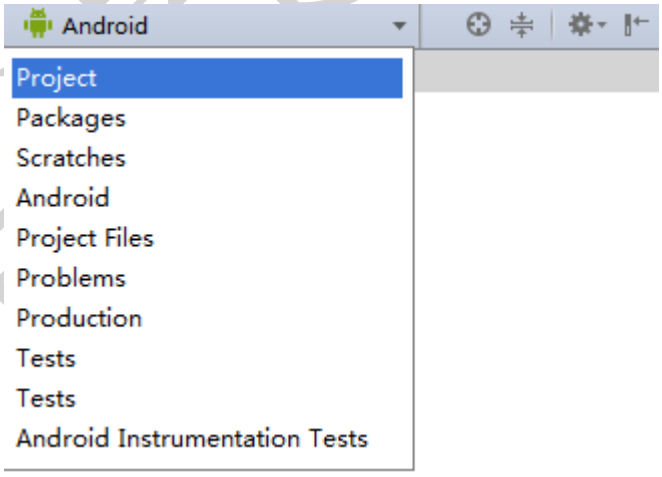
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



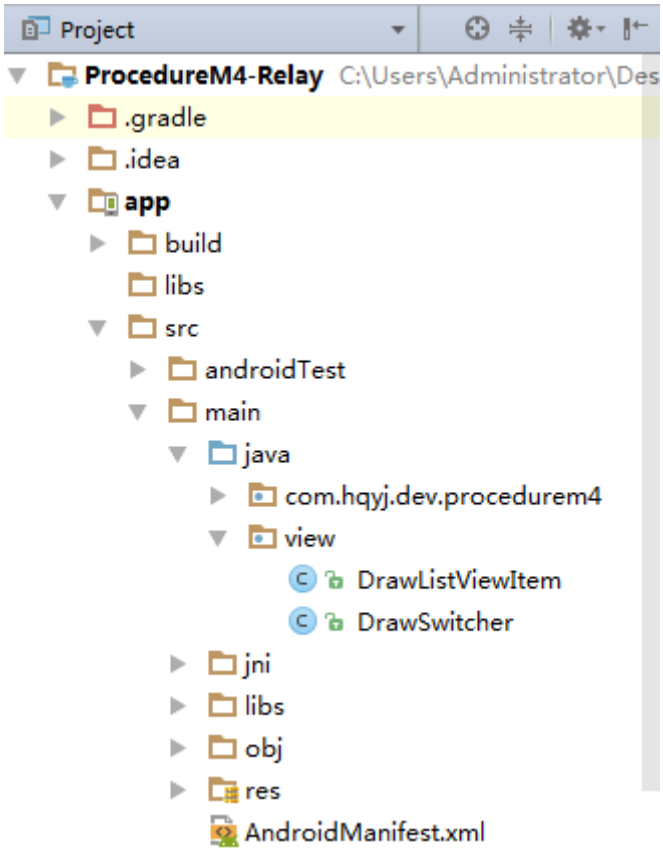
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



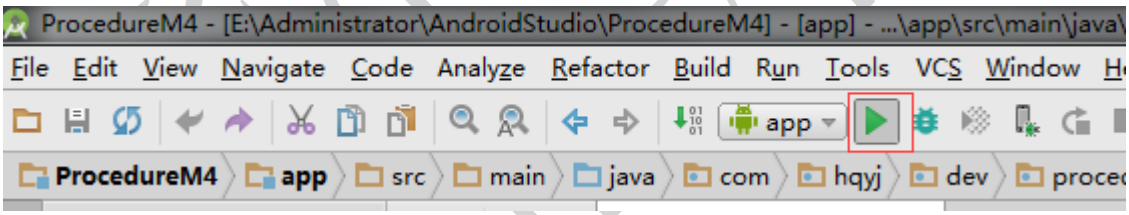
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

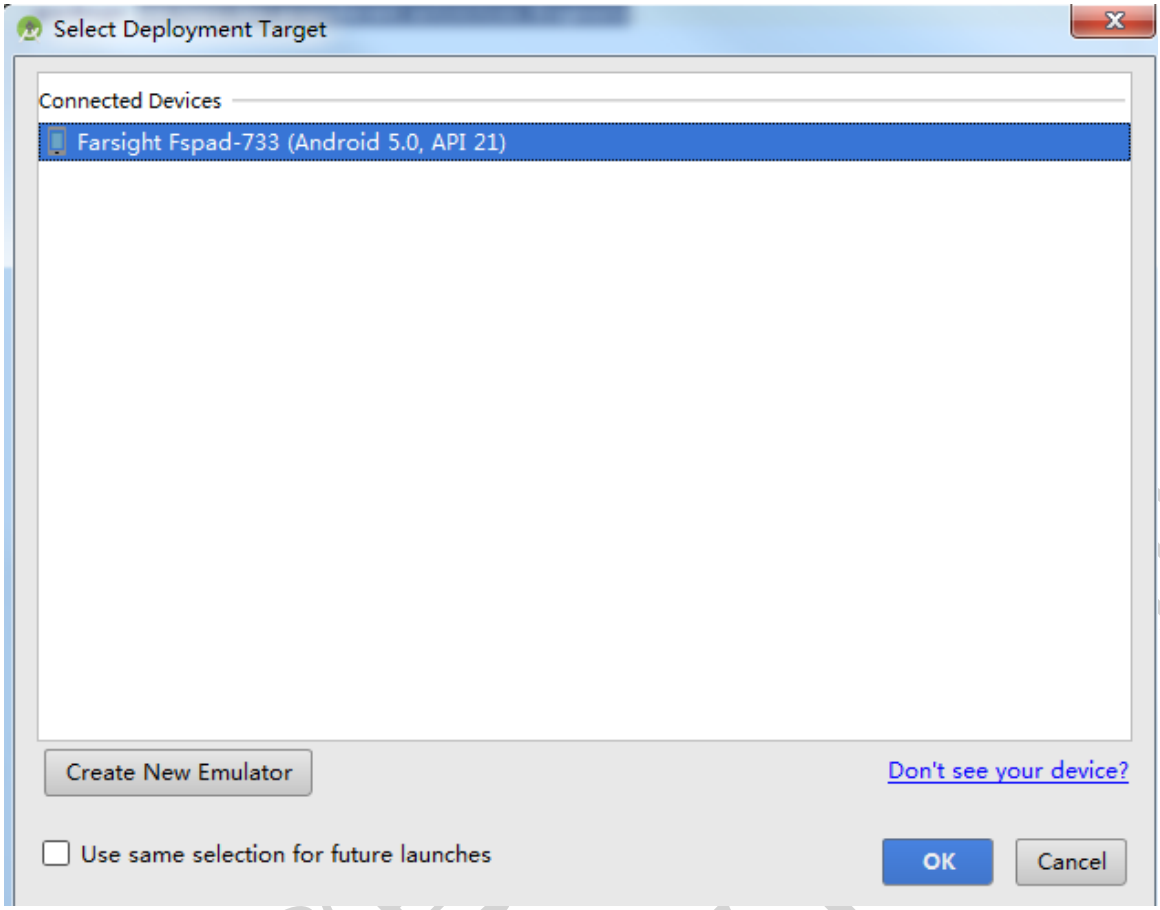
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

华清远见 dev.hqyj.com

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

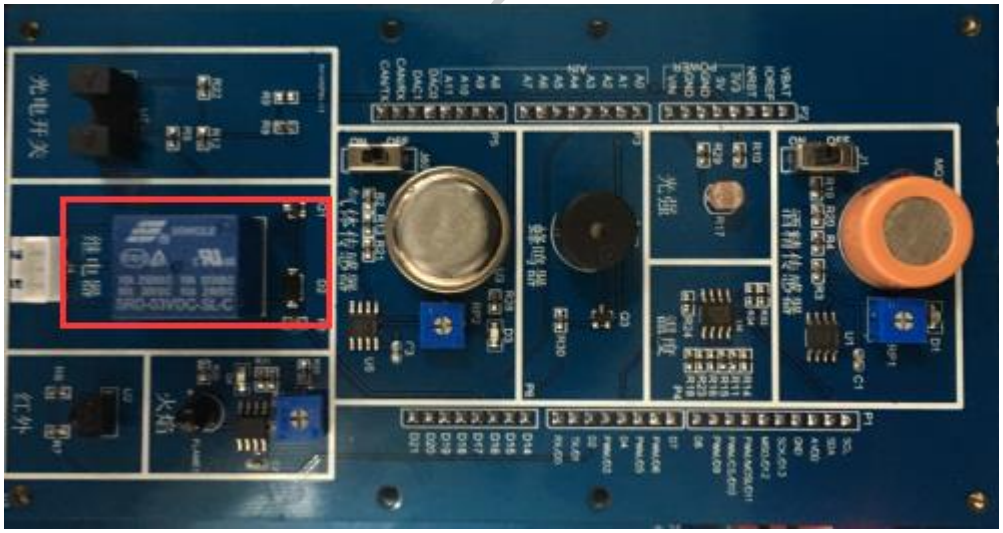
点击进入程序进行测试。

3.11.6 【实验结果】

将拨码开关中的 D16 拨至 ON，其他拨码全为 OFF，如图中提示。

如图所示两个按钮“继电器开”和“继电器关”分别用来对继电器的“开关”进行操作。继电器打开的时候回响一声。

继电器在模块中的位置如下图所示：



3.11.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-8jidianqi.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 relay 的数据

include $(BUILD_SHARED_LIBRARY)
```

发送对 relay 的控制命令.c 文件

源码位置：ProcedureM4-8jidianqi.app.src.main.jni.operate_relay.c

NormalText Code

```
/**
*发送对继电器的操作的命令
*/
void write_relay(int num){
    if(fd_relay == -1){
        fd_relay = open(RELAY_PATH, O_RDWR);
        if(fd_relay < 0){
            LOGI("ERROR OPEN : %s", RELAY_PATH);
            return;
        }
    }

    switch(num){
        case 0://打开继电器
            ioctl(fd_relay, RELAY_ON, 1);
        break;
        case 1://关闭继电器
```

```
            ioctl(fd_relay, RELAY_OFF, 1);
        break;
        default:
            close_relay();//关闭流
        break;
    }
}
void close_relay(){
    if(fd_relay != -1){
        close(fd_relay);
    }
}
```

（2） Android 端对数据的接受和处理

在具体的继电器的 Fragment 中，设置按钮并进行监听，点击按钮实现对继电器的开和关。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.RelayFragment.java

NormalText Code

```
public class RelayFragment extends Fragment implements View.OnClickListener{

    private View mView;
    private Relay relay;
    /**
     *装载名字叫"operate"的库文件
     */
    static {
        System.loadLibrary("operate");
    }
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        relay = Relay.getRelay();
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
                             container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.relay_fragment, container, false);
        //初始化两个按钮，并对按钮设置监听
        mView.findViewById(R.id.btn_relay_open).setOnClickListener(this);
        mView.findViewById(R.id.btn_relay_close).setOnClickListener(this);
        return mView;
```

```
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        relay.operate.write(3);
    }
    @Override
    public void onClick(View v) {
        switch (v.getId()){
            case R.id.btn_relay_open://打开继电器
                relay.operate.write(1);
                break;
            case R.id.btn_relay_close://关闭继电器
                relay.operate.write(0);
                break;
        }
    }
    @Override
    public void onDestroyView() {
        super.onDestroyView();
        relay.operate.write(3);
    }
}
```

3.12 RFID 实验

3.12.1 【实验目的】

（1） 掌握 RFID 的工作原理；

（2） 掌握射频的数据进行接受和处理；

（3） 掌握在 Android 系统下的射频开发流程；

3.12.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

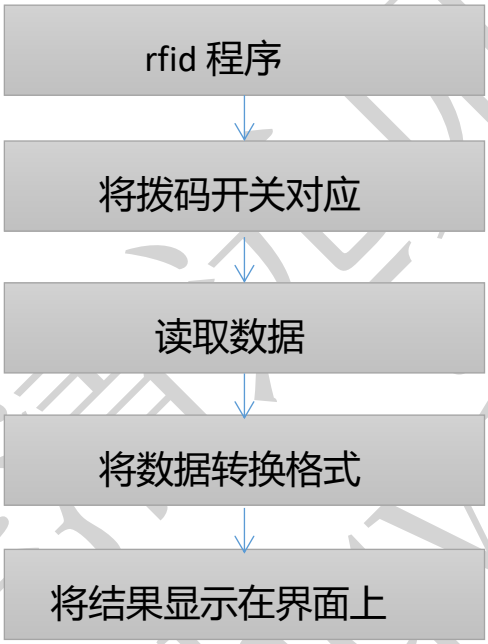
（3） u-boot-2010.03 版本；

3.12.3 【实验内容】

基于 Android5.0 系统实现在 RFID 上进行对卡的扫描，并将扫描到的数据经过处理显示出来。

3.12.4 【实验原理】

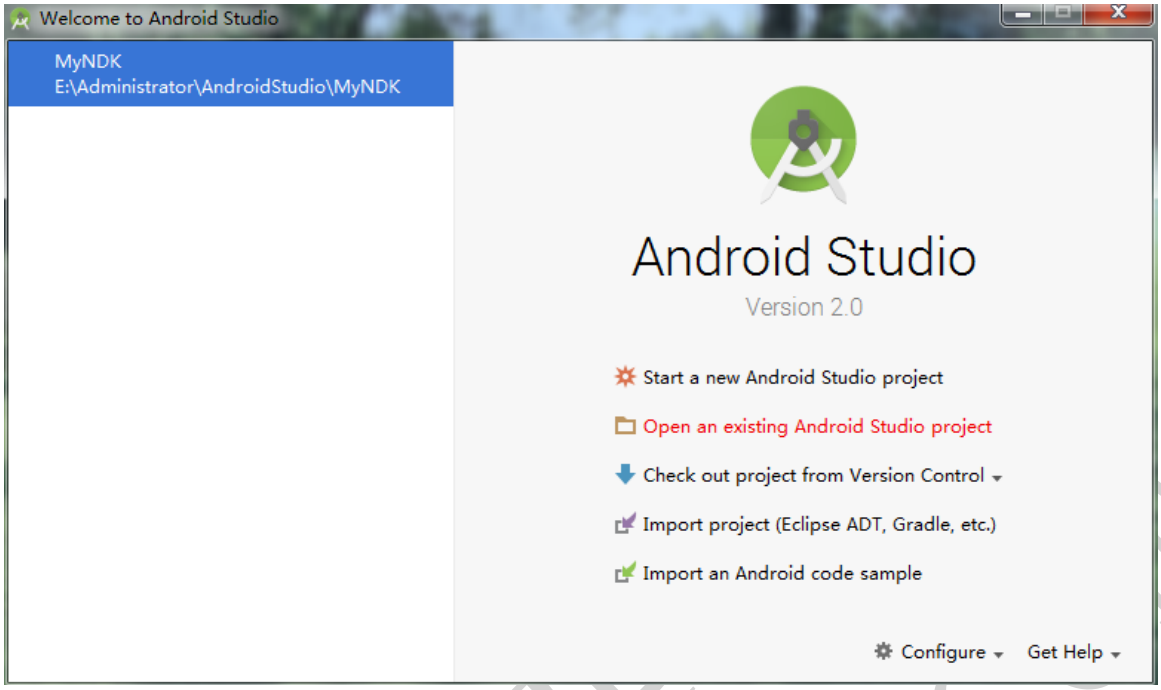
在应用程序里通过调用接口读取 rfid 采集到的数据，并将读取到的数据进行运算和格式转换，然后将结果显示在屏幕上。



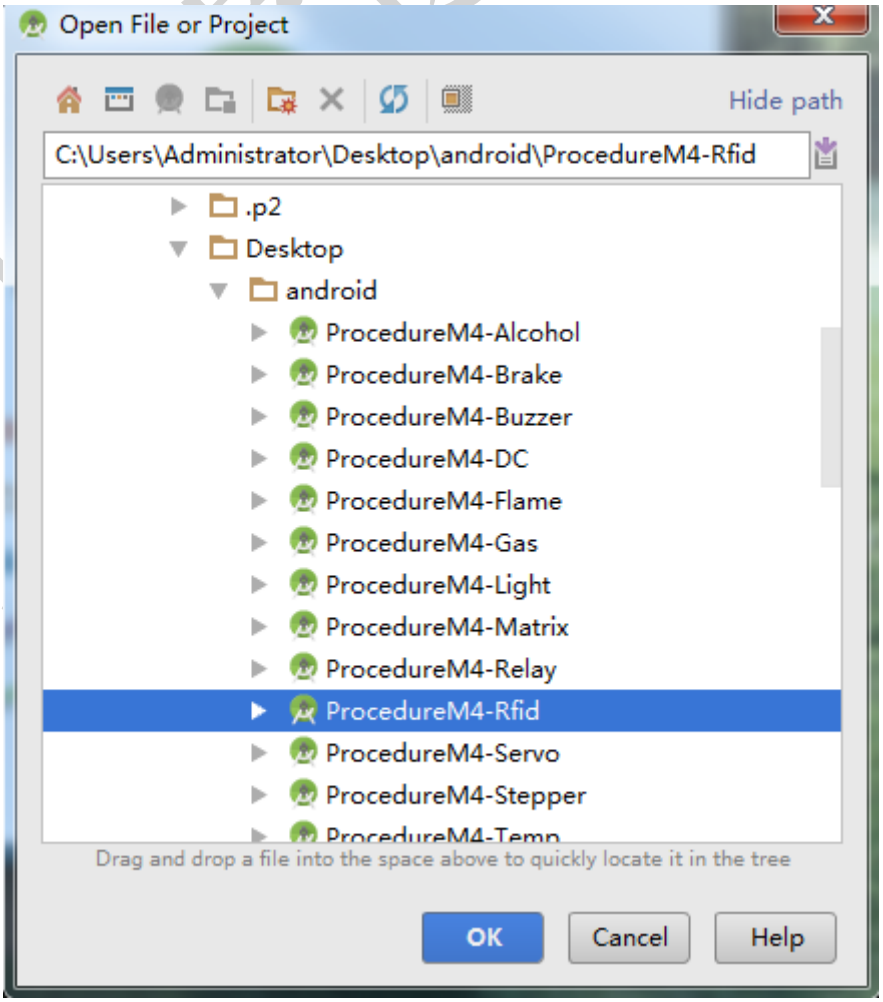
3.12.5 【实验步骤】

RFID 实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Rfid】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

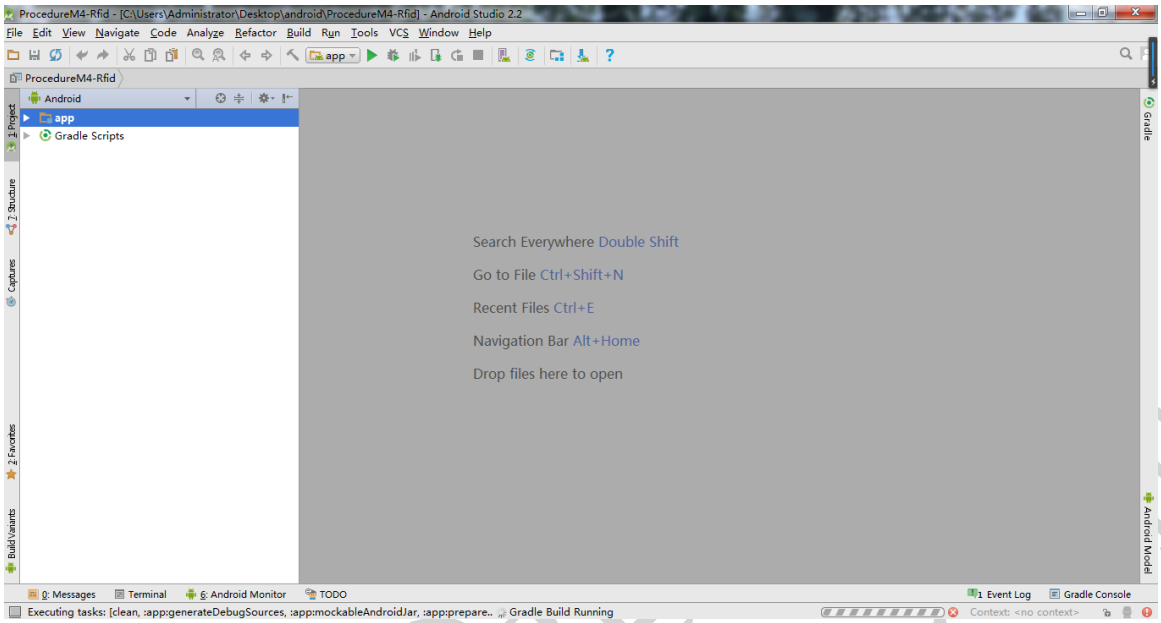
打开 Android studio，导入实验程序：



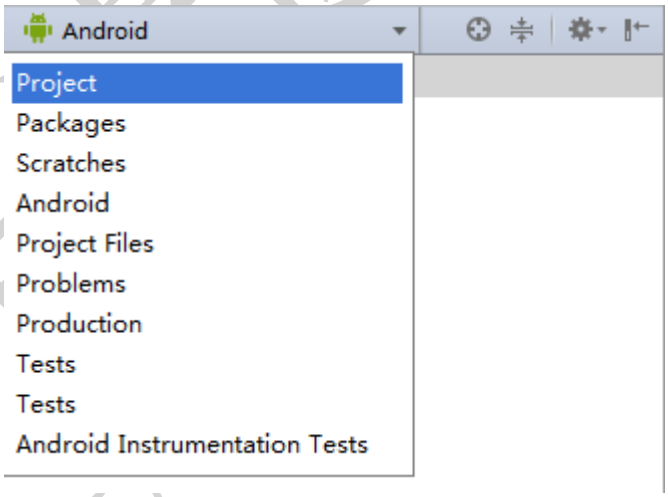
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



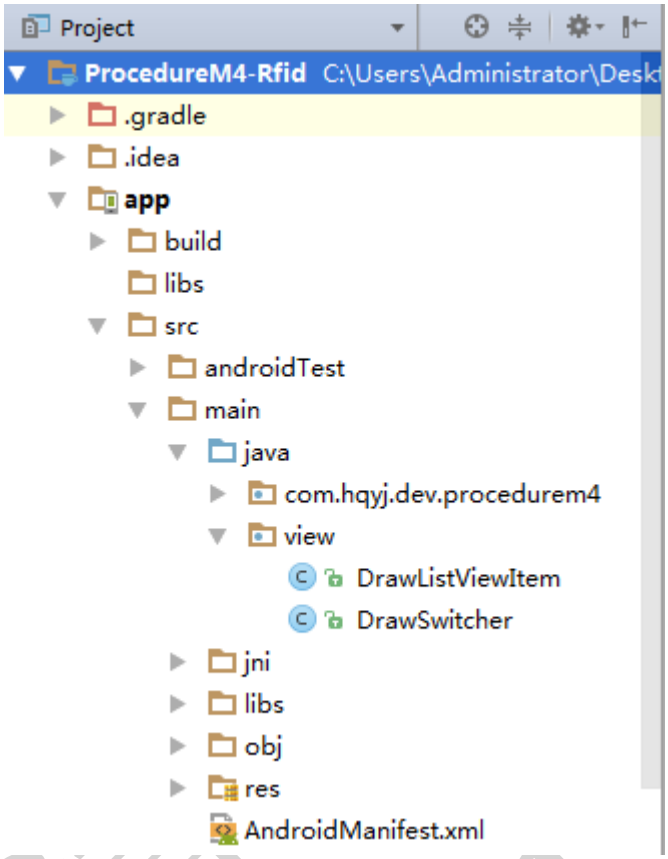
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



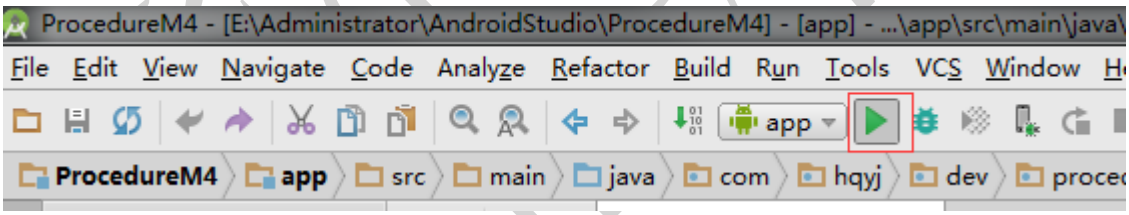
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

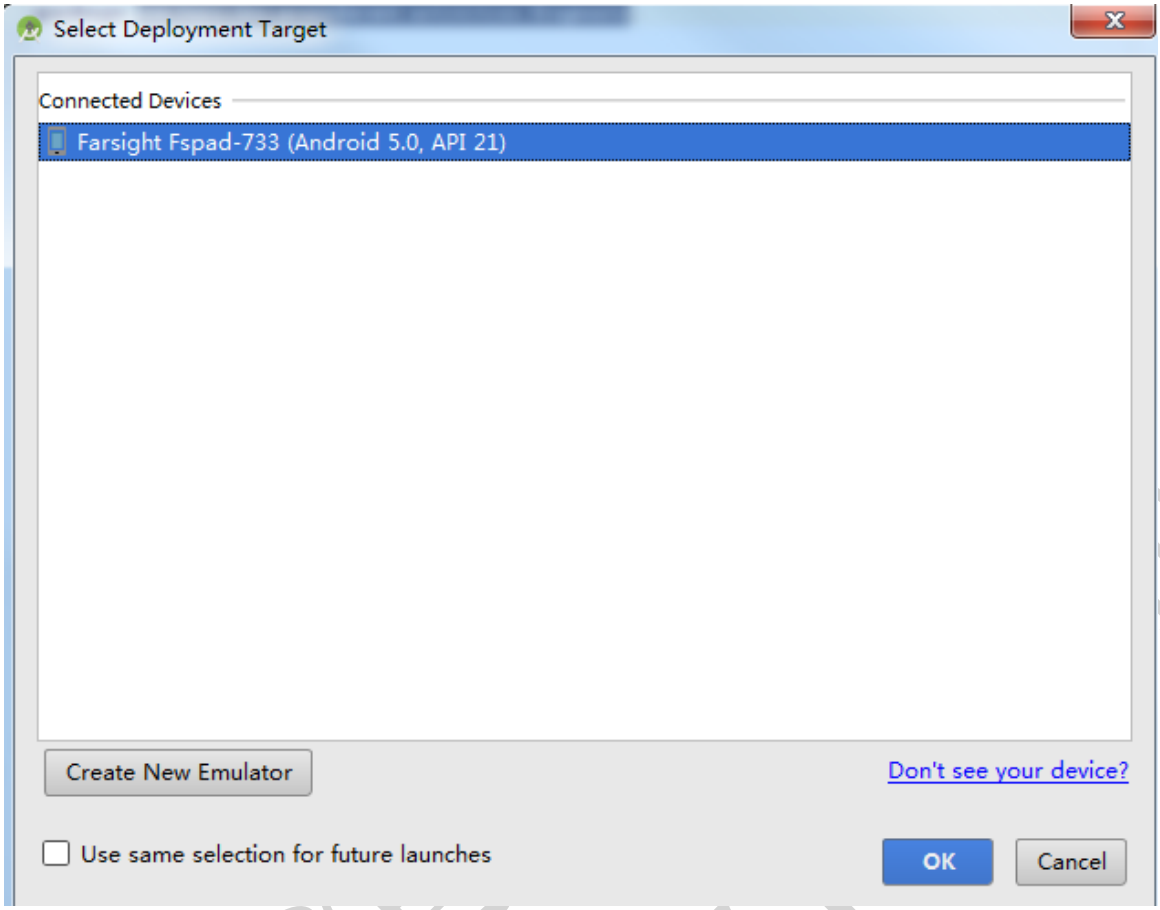
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

华清远见 dev.hqyj.com

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

点击进入程序进行测试。

3.12.6 【实验结果】

该实验只需要注意 SW8 拨码拨至 A8/A9，因为在硬件设计时，将 SW8 作为 MCU 和 CUP 的 i2c 主机切换开关，如图中提示。

RFID 模块读到数据之后会显示在上图所示的框中。上面显示的是“00 00 00 00”

RFID 在模块中的位置如下图所示：

3.12.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-9rfid.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 rfid 的数据

include $(BUILD_SHARED_LIBRARY)
```

读取 rfid 数据的.c 文件

源码位置：ProcedureM4-9rfid.app.src.main.jni.operate_rfid.c

NormalText Code

```
/**
*读取 Rfid 的数据
*/
int read_rfid(){
    uint8_t card_data[32] = {0};
    int number_return = 0;
    //创建流
    int fd = open(RFID_FILE, O_RDWR);
    if (fd < 0){
        LOGI("ERROR OPEN : %s", RFID_FILE);
        return;
    }
    int nbyte = 0;
    int i =0;
    //读取数据
    nbyte = read(fd, card_data, 4);
    if(nbyte != 4){
```

```
        LOGI("READ ERROR: %s", RFID_FILE);
    }
    //间数据进行偏移，得到最后先要显示的数据
    for(i = 0; i < 4; i++){
        number_return = (number_return << 8) | card_data[i];
    }
    return number_return;
}
```

（2） Android 端对数据的接受和处理

在具体的 RFID 的 Fragment 中，开启线程读取 Rfid 数据，然后用 handler 异步处理，对 UI 界面进行动态显示的改变。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.RFIDFragment.java

NormalText Code

```
public class RFIDFragment extends Fragment {

    private View mView;
    private TextView textView;
    private Rfid rfid;
    private String TAG = "RFID";
    private RfidReadThread rfidReadThread;
    private boolean threadOn = false;
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what){
                case 1://将得到的数据显示在界面上
                    String datas = changeIntoIntArray
                            (msg.getData().getInt(TAG));
                    textView.setTextSize(30);
                    textView.setText(datas);
                    break;
                default:
                    break;
            }
        }
```

```
    };
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        rfid = Rfid.getRfid();
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
                        container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.rfid_fragment, container, false);
        initShow();//初始化控件
        //开启线程
        threadOn = true;
        rfidReadThread = new RfidReadThread();
        rfidReadThread.start();
        return mView;
    }
    private void initShow() {
        textView = (TextView) mView.findViewById(R.id.txv_rfid);
    }
    @Override
    public void onDestroyView() {
        super.onDestroyView();
        //关闭线程
        threadOn = false;
        rfidReadThread = null;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
    }
    /**
     * 读取 Rfid 传输的数据
     */
    private class RfidReadThread extends Thread{
        @Override
        public void run() {
            super.run();
            while (threadOn){
                try {
                    sleep(1000);
                } catch (InterruptedException e) {
```

```
                    e.printStackTrace();
                }
                //读取数据
                int numbers = rfid.operate.read()[0];
                if (numbers == 0){
                    continue;
                }
                Bundle bundle = new Bundle();
                bundle.putInt(TAG, numbers);
                Message msg = new Message();
                msg.what = 1;
                msg.setData(bundle);
                //将数据交给 handler 处理
                handler.sendMessage(msg);
            }
        }
    }
    /**
     * 将 int 值转化为字符串
     * @param number 将要转化的 int 值
     * @return 字符串
     */
    @SuppressLint("DefaultLocale")
    private String changeIntoIntArray(int number){
        int[] intArray = new int[4];
        String string;
        for (int i = 0; i < 4; i++){
            intArray[i] = (number >> (3-i)*8) & 0x0f;
        }
        string = String.format("Card ID: %02d %02d %02d %02d",
                intArray[1],intArray[1],intArray[2],intArray[3]);
        return string;
    }
}
```

3.13 舵机实验

3.13.1 【实验目的】

（1） 掌握舵机的工作原理；

（2） 掌握对控制节点数据进行操控；

（3） 掌握在 Android 系统下的控制节点开发流程；

3.13.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

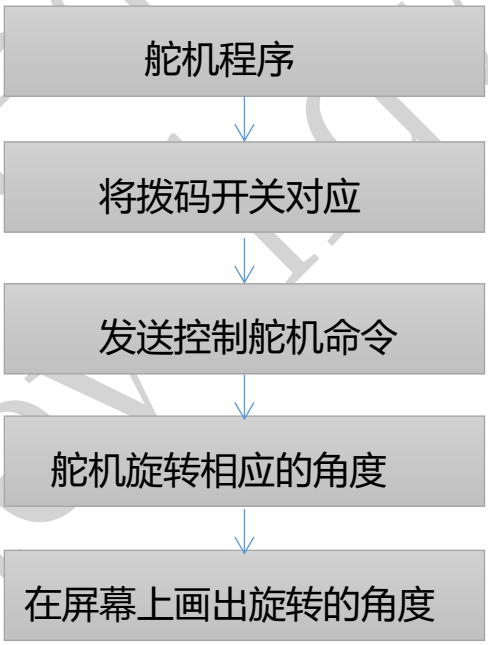
3.13.3 【实验内容】

基于 Android5.0 系统实现对舵机的控制，控制舵机的转动的角度。

3.13.4 【实验原理】

硬件原理参见第二章的舵机驱动实验。

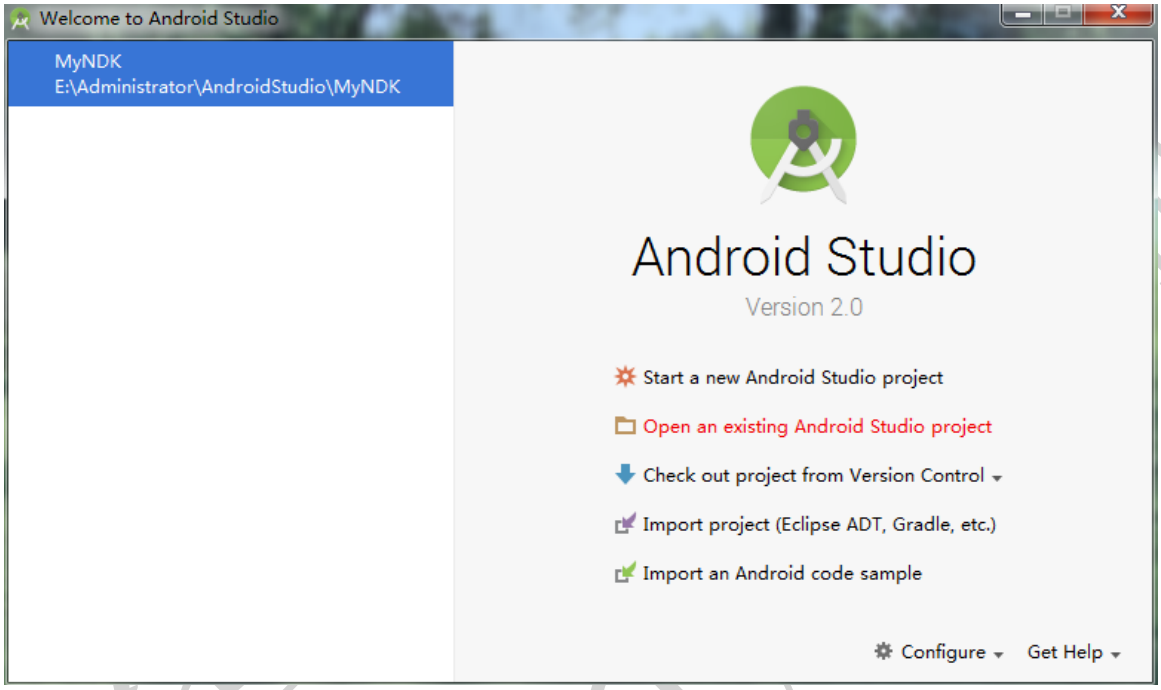
在应用程序里通过调用接口发送控制舵机的命令，使舵机旋转相应的角度，并在屏幕上画出相应的角度。



3.13.5 【实验步骤】

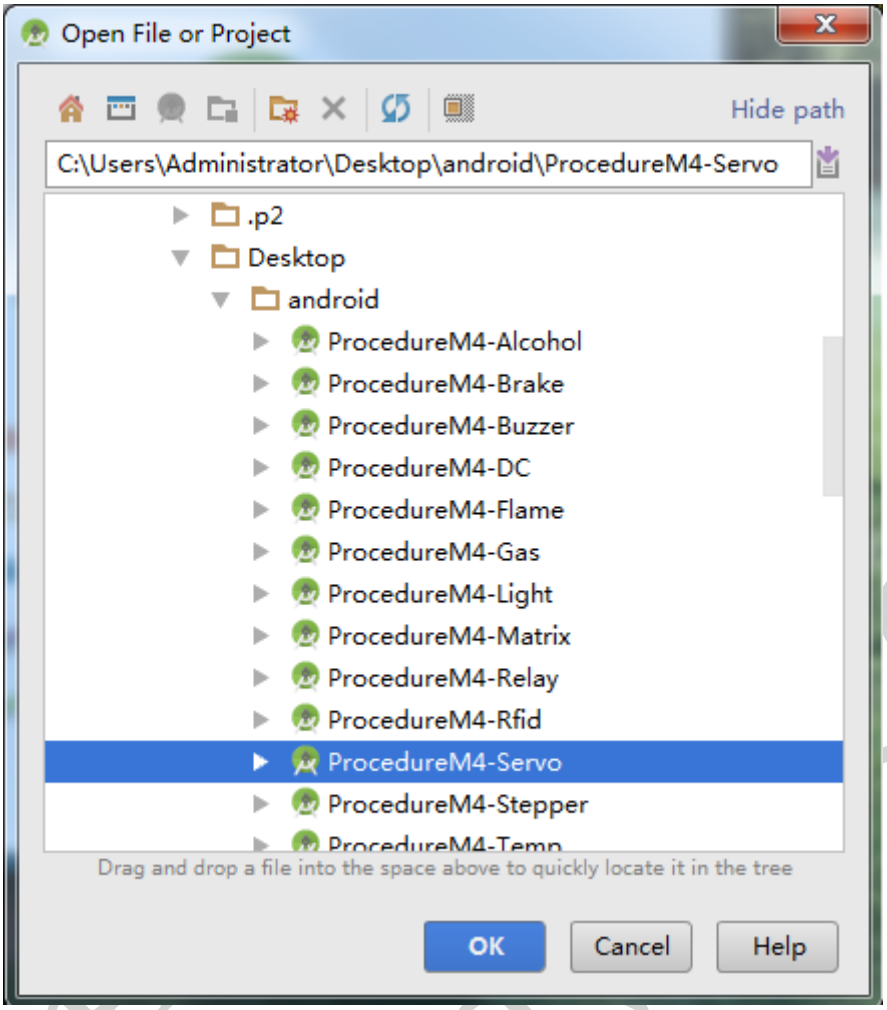
舵机实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Servo】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

打开 Android studio，导入实验程序：

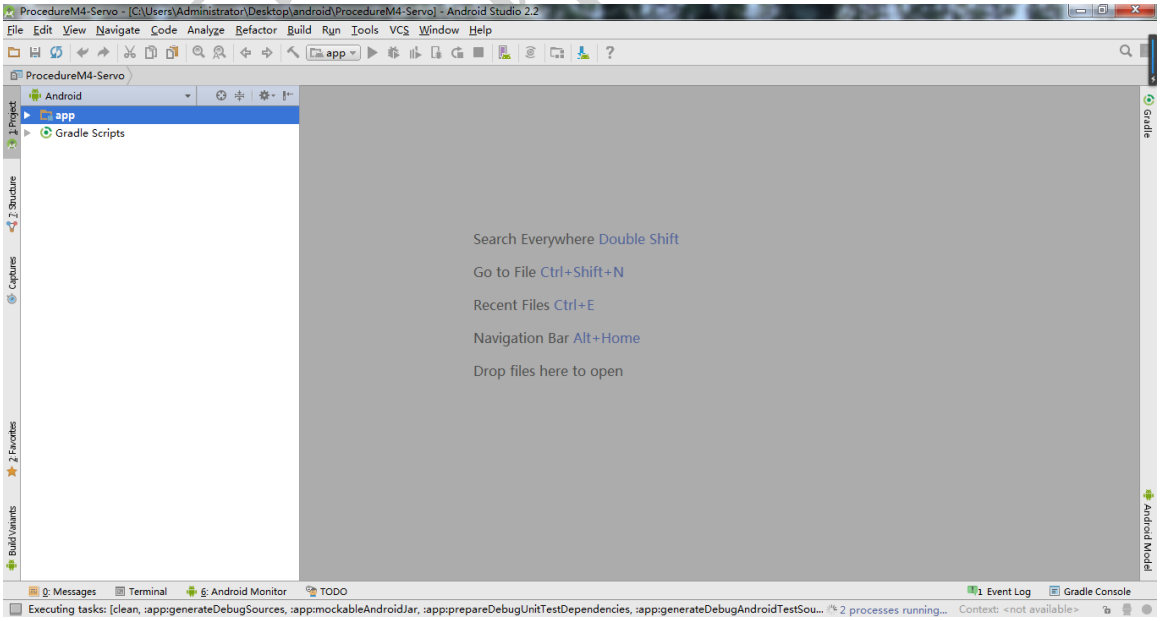


选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：

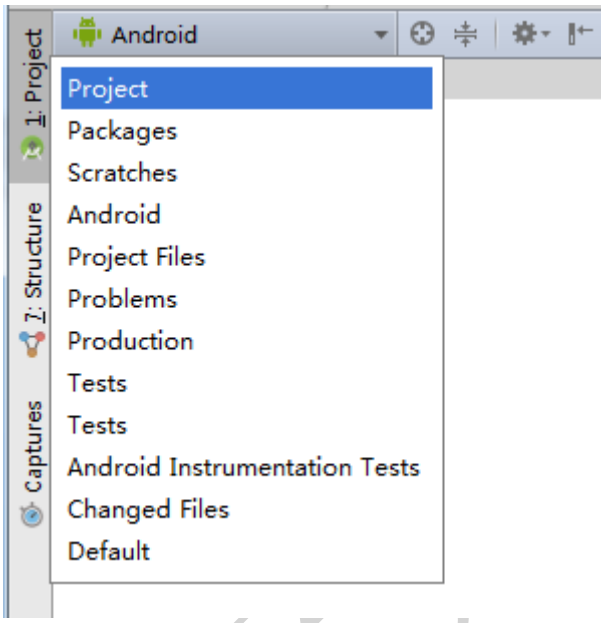
dev.hq



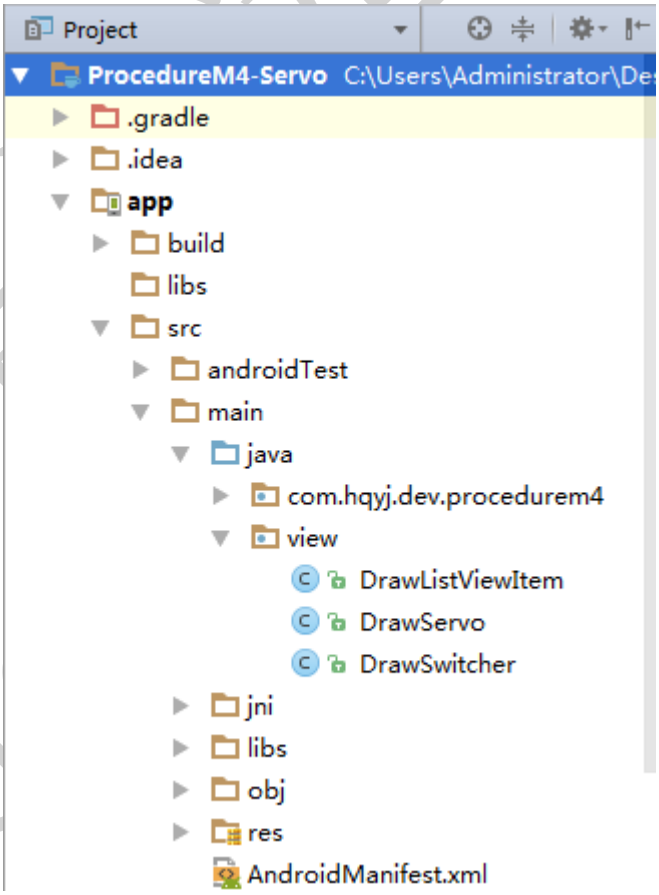
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



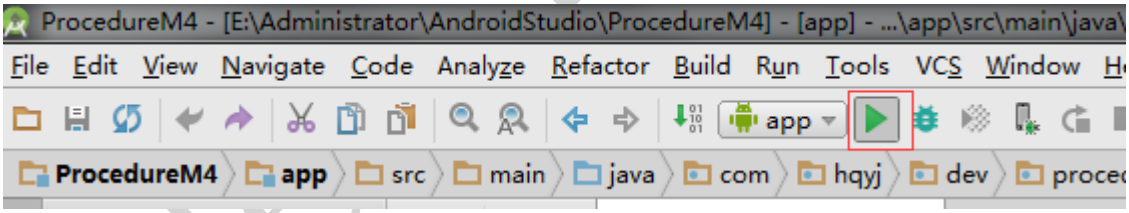
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

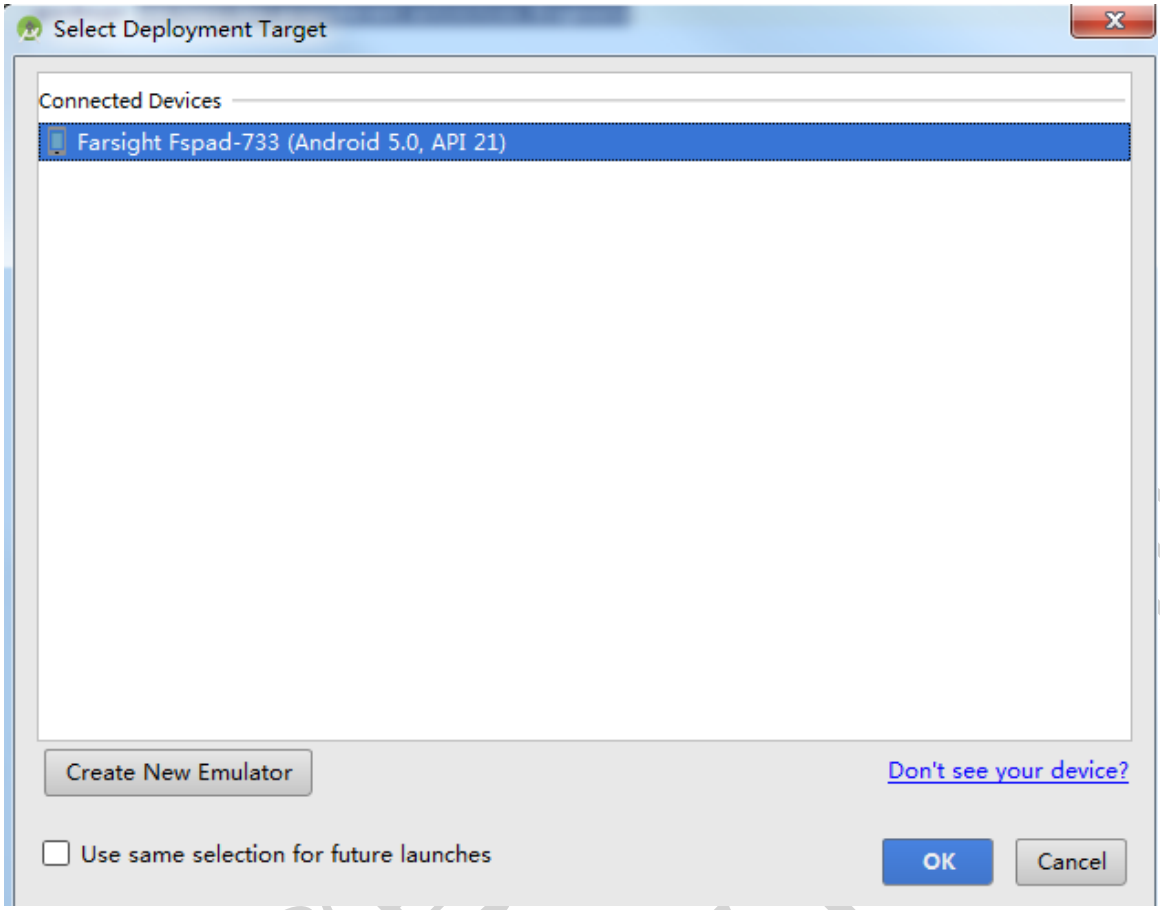
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。

如下图：

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

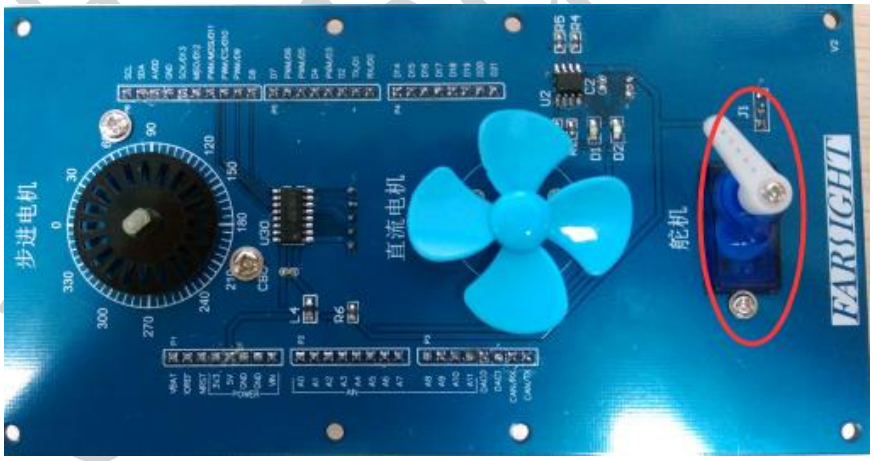
点击进入程序进行测试。

3.13.6 【实验结果】

将拨码开关中的 D6 拨至 ON，其他拨码全为 OFF，如图中提示。

如图所示上面有旋转多少度的按钮，点击之后可以看到舵机旋转的位置。

舵机在模块中的位置如下图所示：



3.13.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-10duoji.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 servo 的数据

include $(BUILD_SHARED_LIBRARY)
```

发送对 servo 的控制命令的.c 文件

源码位置：ProcedureM4-10duoji.app.src.main.jni.operate_servo.c

NormalText Code

```
int fd_servo = -1;
/**
*发送让舵机转换的命令
*/
void trun_angle(int operate){
    if (fd_servo == -1){
        //以可读可写的形式打开流，创建流通道
        fd_servo = open(SERVO_FILE, O_RDWR);
        if(fd_servo < 0){
            LOGI("OPEN ERROR:%s", SERVO_FILE);
            return;
        }
    }
    int number = operate * 100 /9;
    usleep(500);
    //打开舵机
    ioctl(fd_servo, SERVO_ON, 1);
    usleep(number + 500);
    //关闭舵机
    ioctl(fd_servo, SERVO_OFF, 1);
    usleep(19000 - number);
}
```

```
/**
*关闭流的函数
*/
void close_servo(){
    if(fd_servo > 0){
        ioctl(fd_servo, SERVO_OFF, 1);
        close(fd_servo);
        fd_servo = -1;
    }
}
```

（2） Android 端对数据的接受和处理

在具体的舵机的 Fragment 中，通过点击按钮来改变值，在开启的线程中发送旋转具体角度的命令，对舵机旋转角度进行控制。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.ServoFragment.java

NormalText Code

```
public class ServoFragment extends Fragment implements
                    RadioGroup.OnCheckedChangeListener{

        private View mView;
        private DrawServo drawServo;
        private Servo servo;
        private String TAG = "SERVO";
        private boolean threadOn = false;
        private ServoWriteThread servoWriteThread;
        private int degree = 0;
        /**
         * 异步处理机制，实现子线程更新 UI 界面的功能
         */
        @SuppressLint("HandlerLeak")
        private Handler handler = new Handler(){
            @Override
            public void handleMessage(Message msg) {
                super.handleMessage(msg);
                switch (msg.what){
                    case 1://给 degree 赋值
                        Log.d(TAG, msg.getData().getInt(TAG) +"角度");
                        degree = msg.getData().getInt(TAG);
                        break;
                    default:
                        break;
```

```
            }
        }
    };
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        servo = Servo.getServo();

    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
                             container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.servo_fragment, container, false);
        initShow();//初始化自定义控件并设置角度为 90 度
        //自定义控件的监听事件，是发送数据给 handler，进行处理
        drawServo.setOnDegreeChanged(new DrawServo.OnDegreeChanged() {
            @Override
            public void onDegreeChanged(int degree) {
                Bundle bundle = new Bundle();
                bundle.putInt(TAG, degree);
                Message msg = new Message();
                msg.what = 1;
                msg.setData(bundle);
                handler.sendMessage(msg);
            }
        });
        //初始化控件，设置监听
        ((RadioGroup) mView.findViewById(R.id.radio_group)).
            setOnCheckedChangeListener(ServoFragment.this);
        //开启线程
        servoWriteThread = new ServoWriteThread();
        threadOn = true;
        servoWriteThread.start();
        return mView;
    }
    private void initShow() {
        drawServo = (DrawServo) mView.findViewById(R.id.draw_servo);
        drawServo.setDegree(90);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
```

```
69          }
70          //按钮的改变的事件
71          @Override
72          public void onCheckedChanged(RadioGroup group, int checkedId) {
73              int degree = 0;
74              switch (group.getId()){
75                  case R.id.radio_group:
76                      switch (checkedId){
77                          case R.id.radio_1://设置舵机旋转角度为 30 度
78                              degree = 30;
79                              break;
80                          case R.id.radio_2://设置舵机旋转角度为 60 度
81                              degree = 60;
82                              break;
83                          case R.id.radio_3://设置舵机旋转角度为 90 度
84                              degree = 90;
85                              break;
86                          case R.id.radio_4://设置舵机旋转角度为 120 度
87                              degree = 120;
88                              break;
89                          case R.id.radio_5://设置舵机旋转角度为 150 度
90                              degree = 150;
91                              break;
92
93                          default:
94                              break;
95                      }
96                      break;
97              }
98              //在舵机转动角度的同时，图片也在转动
99              drawServo.setDegree(degree);
100             drawServo.invalidate();
101         }
102     /**
103      * 对舵机发送转动具体角度的命令
104      */
105     private class ServoWriteThread extends Thread{
106         @Override
107         public void run() {
108             super.run();
109             while(threadOn){
110                 servo.operate.write(degree);
111             }
```

```
        }
    }
    @Override
    public void onDestroyView() {
        super.onDestroyView();
        //关闭线程
        threadOn = false;
        servo.operate.write(-1);
        servoWriteThread.interrupt();
    }
}
```

3.14 步进电机实验

3.14.1 【实验目的】

（1） 掌握步进电机的工作原理；

（2） 掌握对控制节点数据进行操控；

（3） 掌握在 Android 系统下的控制节点开发流程；

3.14.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

3.14.3 【实验内容】

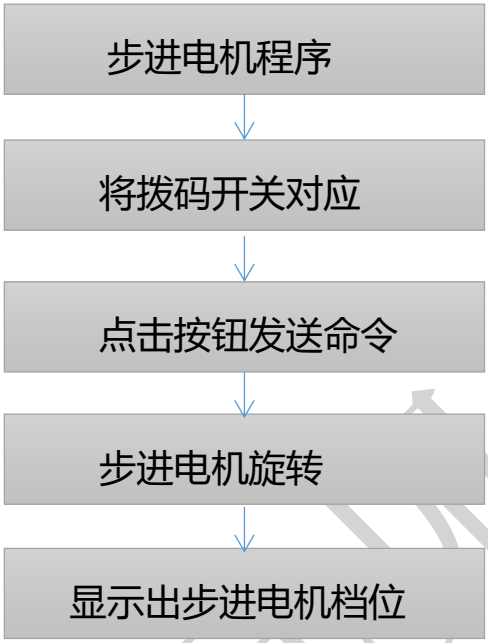
基于 Android5.0 系统实现对步进电机的控制，控制步进电机转动的快慢。

3.14.4 【实验原理】

硬件原理参见第二章的步进电机驱动实验；

在应用程序里通过调用接口发送控制步进电机的命令，使步进电机开始旋转，在程序里面可以调动步进电机的档位。

华清远见 dev.hqyj.com

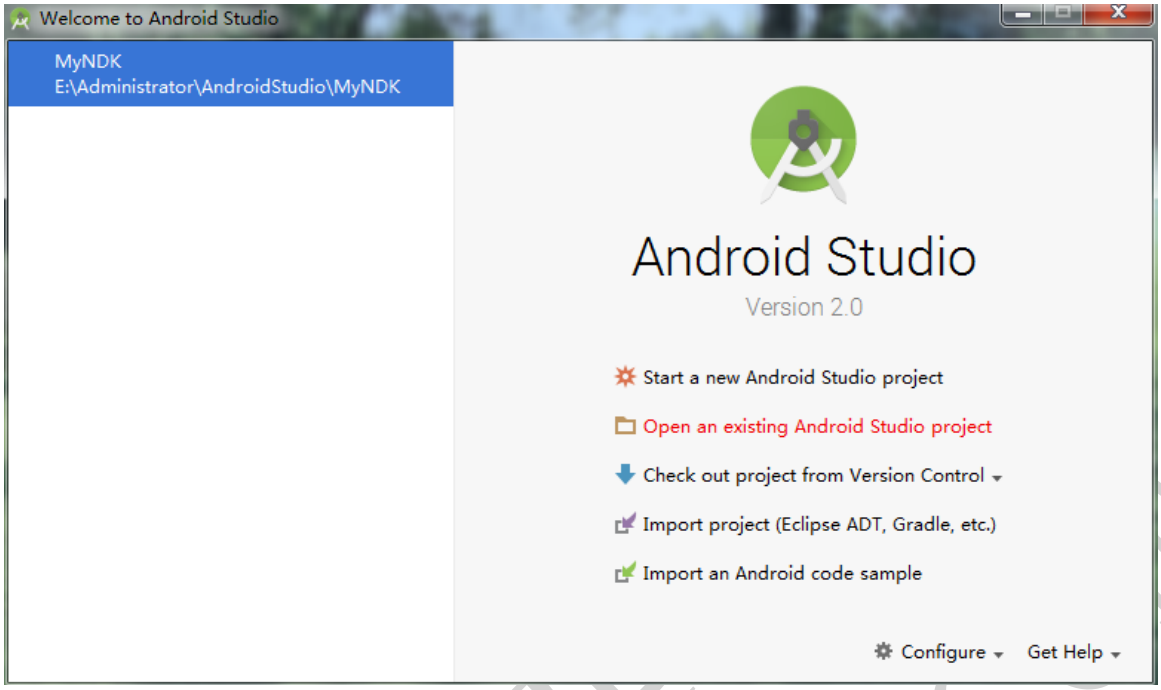


3.14.5 【实验步骤】

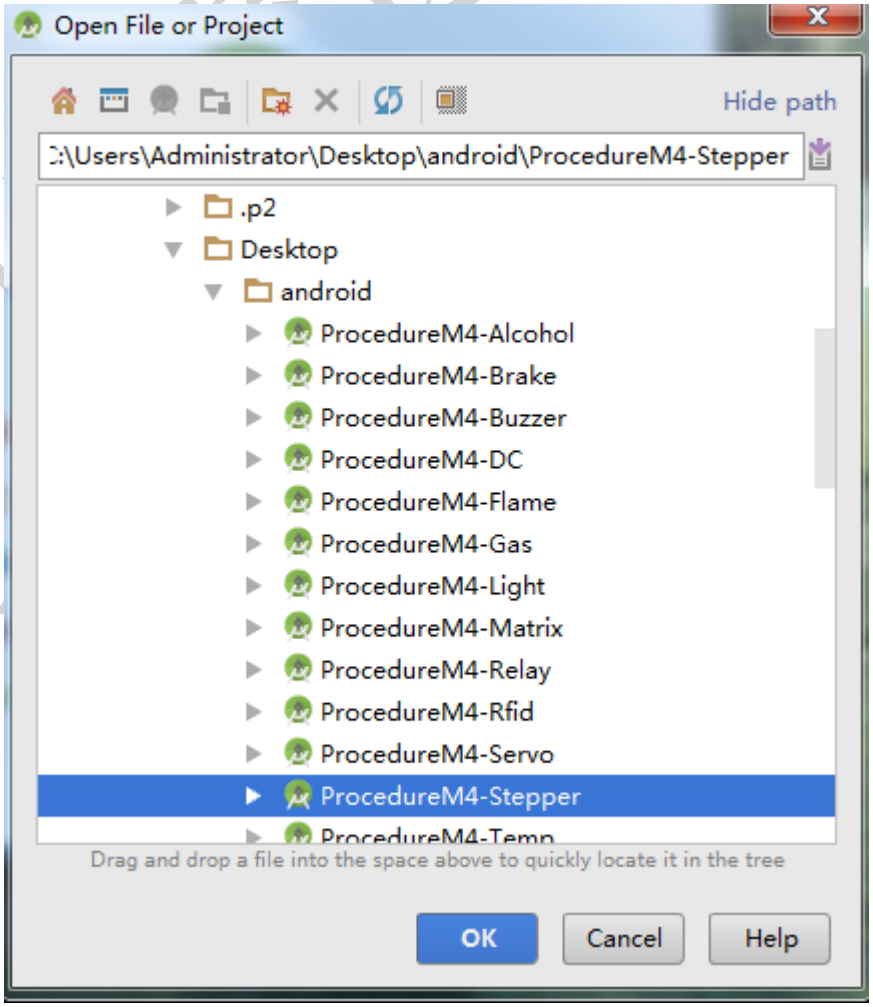
步进电机实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Stepper】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

打开 Android studio，导入实验程序：

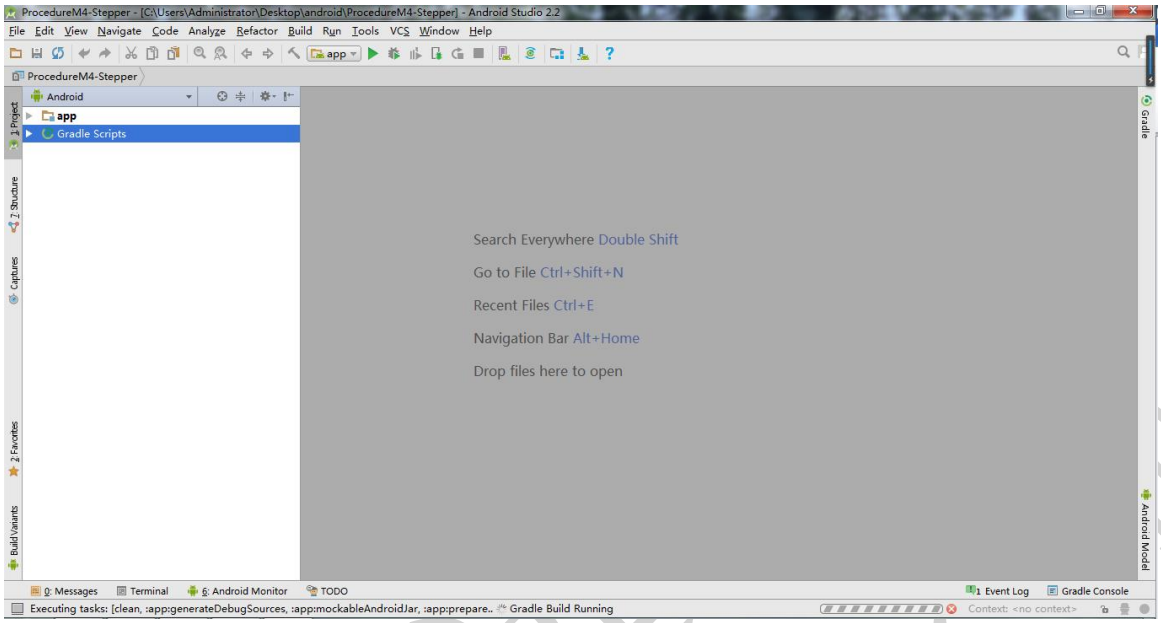
华清远见 dev.hqyj.com



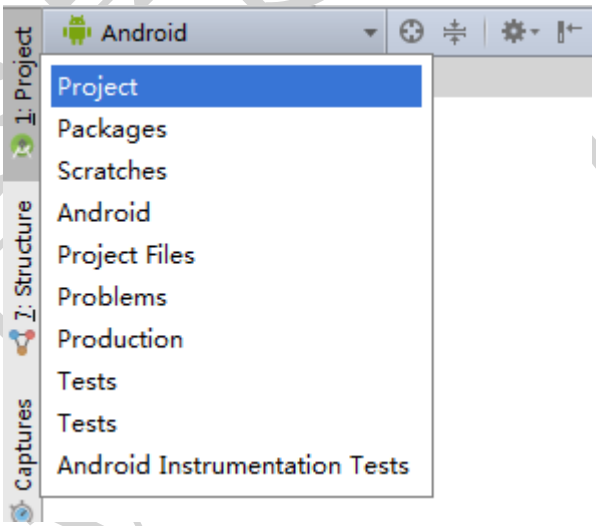
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



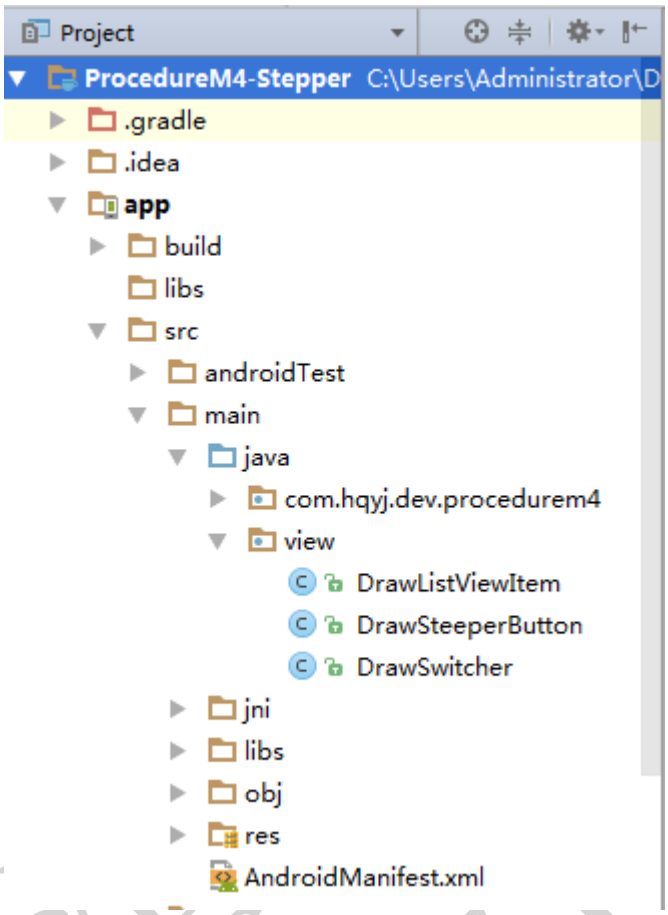
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



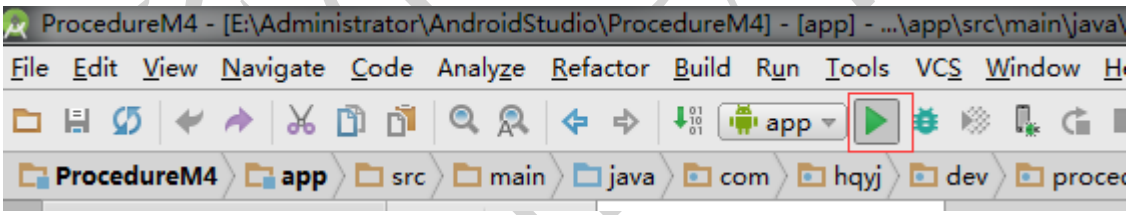
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

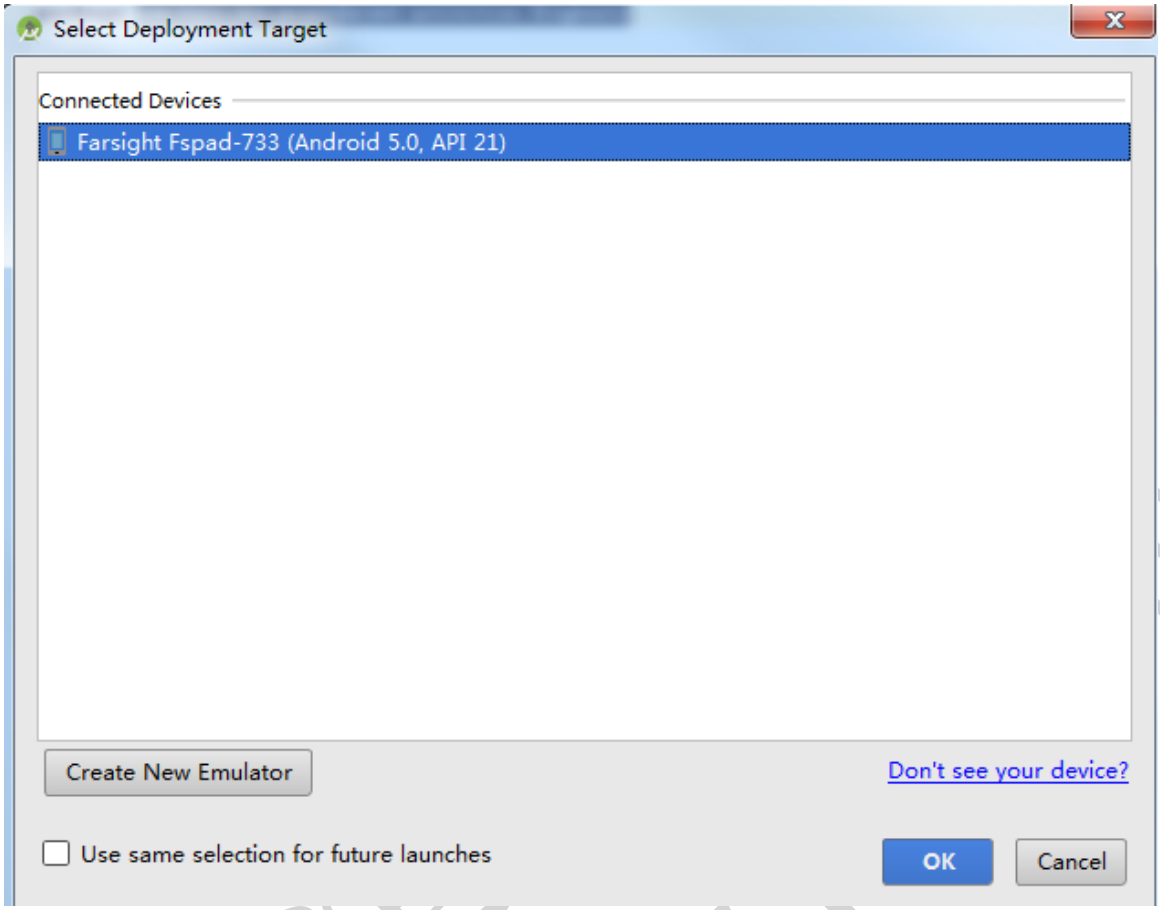
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

华清远见 dev.hqyj.com

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

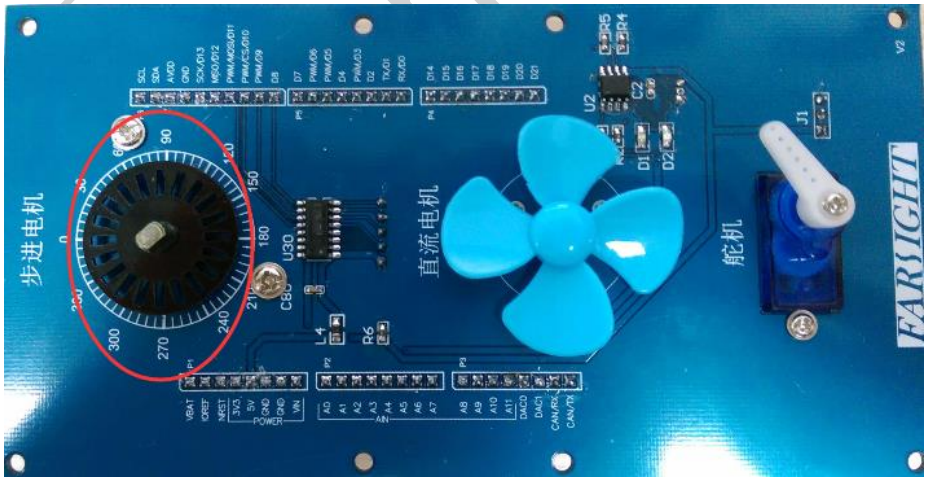
点击进入程序进行测试。

3.14.6 【实验结果】

将拨码开关中的 D8、D9、D10、D11 拨至 ON，其他拨码全为 OFF，如图中提示。

步进电机转分为 9 档，依次加快，可以看到步进电机在旋转。

步进电机在模块中的位置如下图所示：



3.14.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-11bujin.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 steeper 的数据

include $(BUILD_SHARED_LIBRARY)
```

发送对 steeper 控制的命令的.c 文件

源码位置：ProcedureM4-11bujin.app.src.main.jni.operate_steeper.c

NormalText Code

```
/**
*发送命令，打开步进电机
*/
void start_steeper(int speed){
    int delay = 0;
    if(fd_steeper == -1){
        //以可读可写的形式打开 STEEPER_FILE，创建流通道
        fd_steeper = open(STEEPER_FILE, O_RDWR);
        if(fd_steeper < 0){
            LOGI("ERROR OPEN:%s", "STEEPER");
            return;
        }
    }
    delay = (10-speed) * 1000;
    LOGI("%d", delay);
    //发送命令，对步进电机的各个开关的控制,形成循环转动
    ioctl(fd_steeper, STEEPER_ON, 1);
```

```
    ioctl(fd_steeper, STEEPER_ON, 2);
    ioctl(fd_steeper, STEEPER_OFF, 3);
    ioctl(fd_steeper, STEEPER_OFF, 4);
    usleep(delay);
    ioctl(fd_steeper, STEEPER_ON, 3);
    ioctl(fd_steeper, STEEPER_OFF, 1);
    usleep(delay);
    ioctl(fd_steeper, STEEPER_ON, 4);
    ioctl(fd_steeper, STEEPER_OFF, 2);
    usleep(delay);
    ioctl(fd_steeper, STEEPER_ON, 1);
    ioctl(fd_steeper, STEEPER_OFF, 3);
    usleep(delay);
}
/**
*关闭步进电机的各个引脚，关闭流
*/
void close_steeper(){
    if(fd_steeper != -1){
        ioctl(fd_steeper, STEEPER_OFF, 1);
        ioctl(fd_steeper, STEEPER_OFF, 2);
        ioctl(fd_steeper, STEEPER_OFF, 3);
        ioctl(fd_steeper, STEEPER_OFF, 4);
        close(fd_steeper);
    }
}
```

（2） Android 端对数据的接受和处理

在具体的步进电机的 Fragment 中,通过点击按钮来控制步进电机的旋转和停止还有旋转的速度，开启线程发送对步进电机控制的命令，在 handler 异步处理中，对步进电机状态和速度进行显示。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.SteeperFragment.java

NormalText Code

```
public class SteeperFragment extends Fragment {

    private View mView;
    private Steeper steeper;
    private DrawSteeperButton drawSteeperButton;
    private TextView textView;
    private String TAG = "STEEPER";
    private boolean threadOn = false;
    private int steeperSpeed = 0;
    private SteeperWriteThread steeperWriteThread = null;
```

```
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler(){
        @SuppressLint("DefaultLocale")
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what){
                case 1://显示当前步进电器的档位
                    int speed = msg.getData().getInt(TAG);
                    textView.setText(String.format("速度：%d 档", speed));
                    steeperSpeed = speed;
                    break;
            }
        }
    };
    /**
     *装载名字叫"operate"的库文件
     */
    static {
        System.loadLibrary("operate");
    }
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        steeper = Steeper.getSteeper();
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
                        container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.steeper_fragment, container, false);
        initShow();//初始化控件
        //开启线程，用来启动步进电机
        threadOn = true;
        steeperWriteThread = new SteeperWriteThread();
        steeperWriteThread.start();
        //按钮的监听事件：把当前的档位告知 handler，进行处理
        drawSteeperButton.setOnSpeedChanage(new DrawSteeperButton.
                                                OnSpeedChanage() {
```

```
54            @Override
55            public void speed(int speed) {
56                Bundle b = new Bundle();
57                b.putInt(TAG, speed);
58                Message msg = new Message();
59                msg.what = 1;
60                msg.setData(b);
61                handler.sendMessage(msg);
62            }
63        });
64        drawSteeperButton.setState(0);
65        return mView;
66    }
67    private void initShow() {
68        drawSteeperButton = (DrawSteeperButton) mView.findViewById(R.id.draw_steeper);
69        textView = (TextView) mView.findViewById(R.id.txv_steeper);
70    }
71    @Override
72    public void onDestroy() {
73        super.onDestroy();
74    }
75    @Override
76    public void onDestroyView() {
77        super.onDestroyView();
78        //关闭线程
79        threadOn = false;
80        steeperWriteThread.interrupt();
81    }
82    /**
83     * 此线程是用来发送启动步进电机的命令的
84     */
85    private class SteeperWriteThread extends Thread{
86        @Override
87        public void run() {
88            super.run();
89            while (threadOn){
90                steeper.operate.write(steeperSpeed);
91                if (steeperSpeed == 0){
92                    try {
93                        sleep(1000);
94                    } catch (InterruptedException e) {
95                        e.printStackTrace();
96                    }
```

```
                }
            }
        }
    }
}
```

3.15 火焰传感器实验

3.15.1 【实验目的】

（1） 掌握火焰传感器的工作原理；

（2） 掌握对传感器数据进行接受和处理；

（3） 掌握在 Android 系统下的传感器开发流程；

3.15.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

3.15.3 【实验内容】

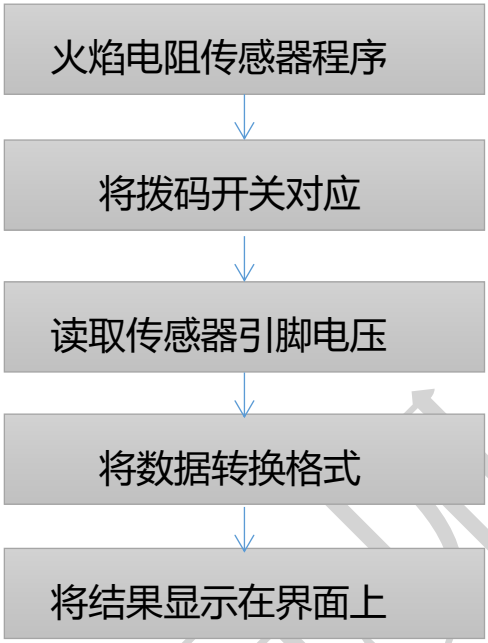
基于 Android5.0 系统实现读取热敏电阻两个引脚的电压值。

3.15.4 【实验原理】

硬件原理参见第二章的火焰传感器驱动实验；

在应用程序里通过调用接口读取火焰传电阻两端电压，并将读取到的数据进行运算和格式转换，然后将结果显示在屏幕上，显示内容是当前火焰传感器电阻两端电压。

华清远见 dev.hqyj.com

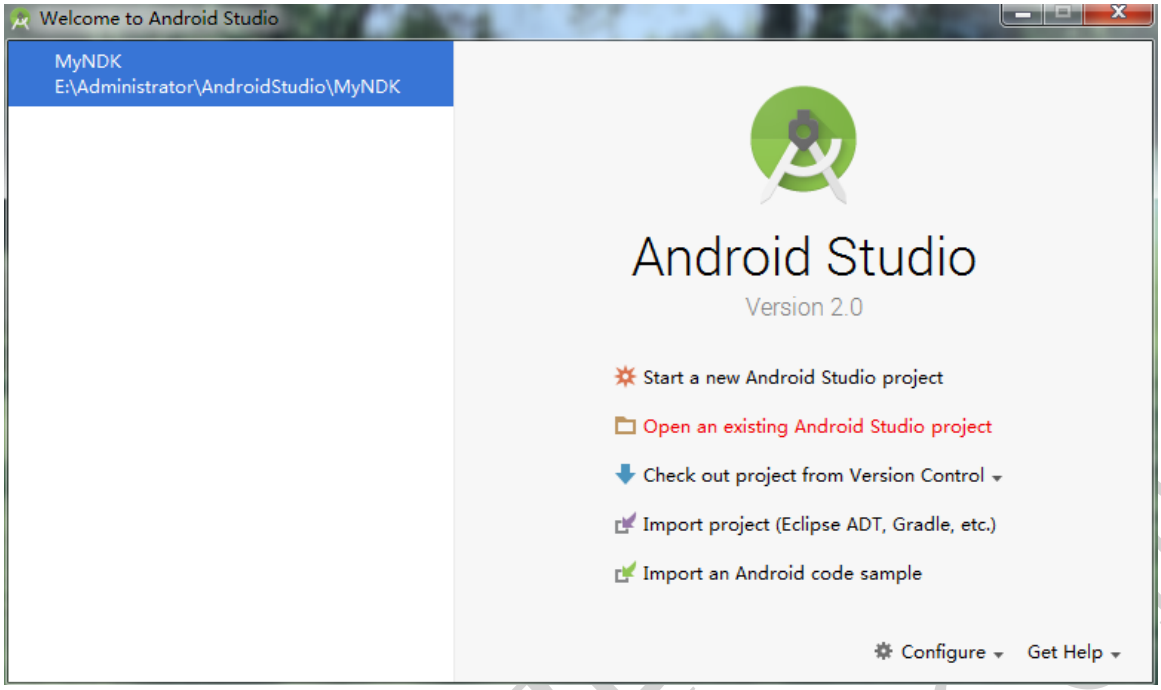


3.15.5 【实验步骤】

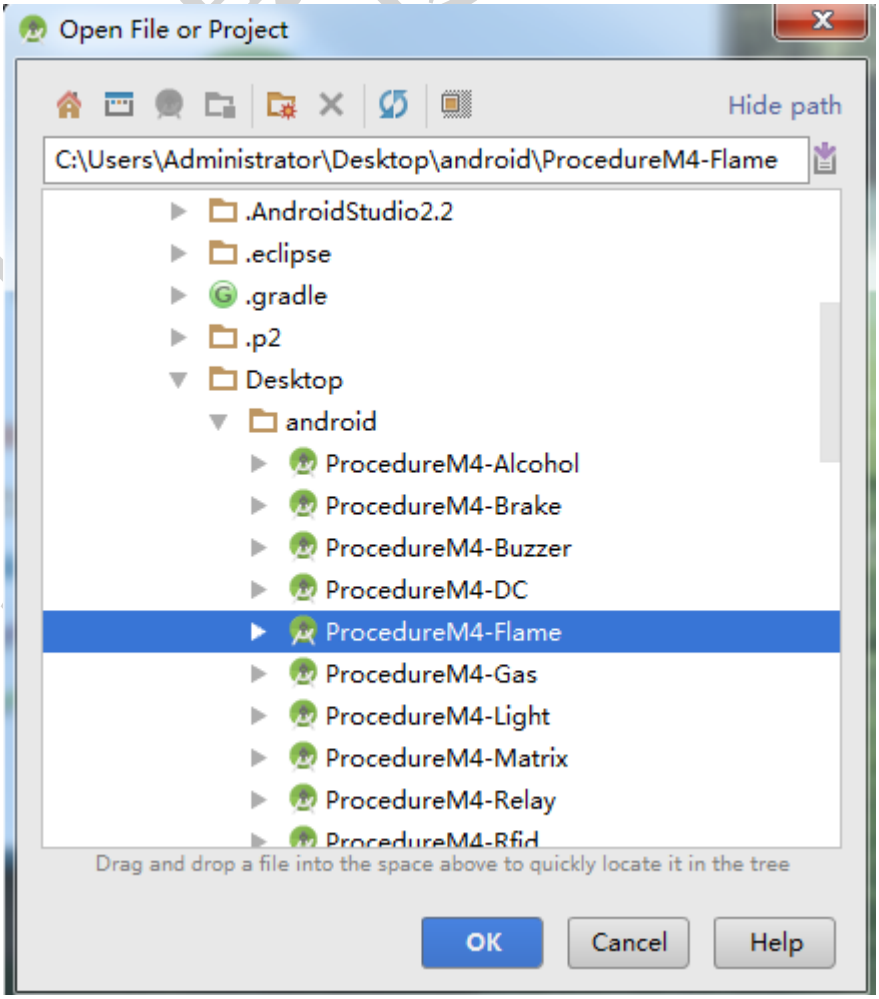
火焰电阻实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Flame】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

打开 Android studio，导入实验程序：

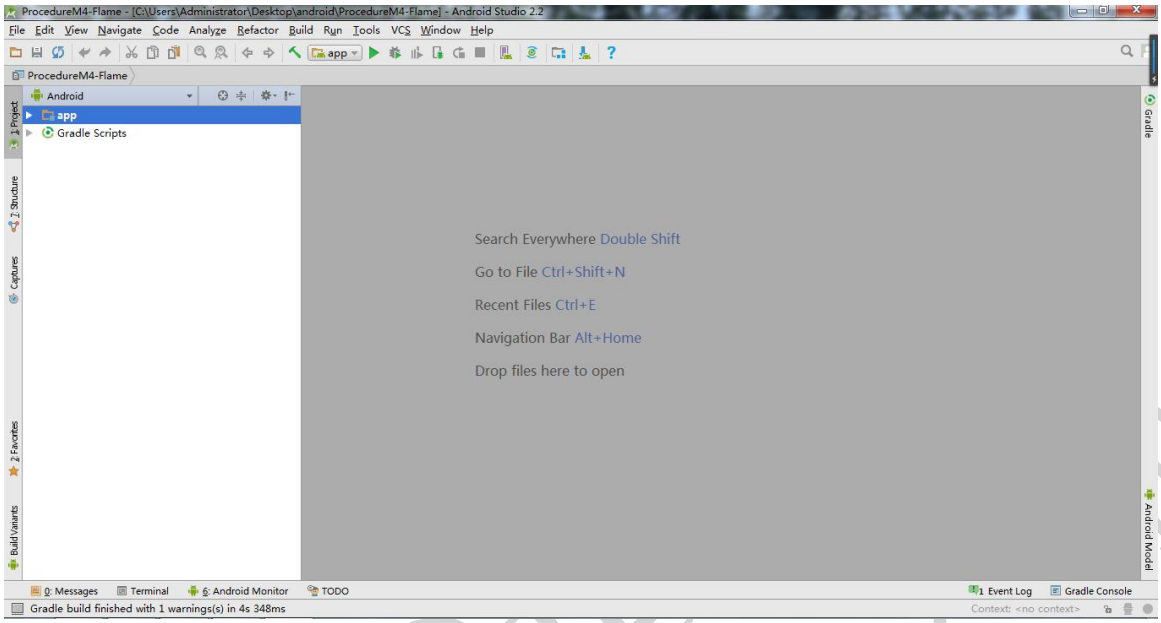
华清远见 dev.hqyj.com



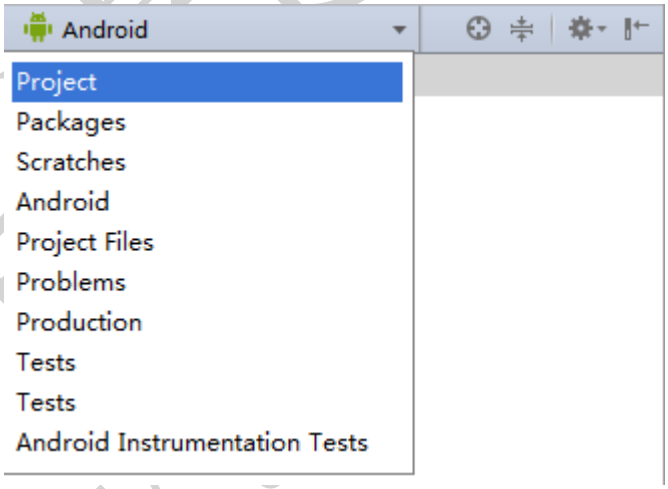
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



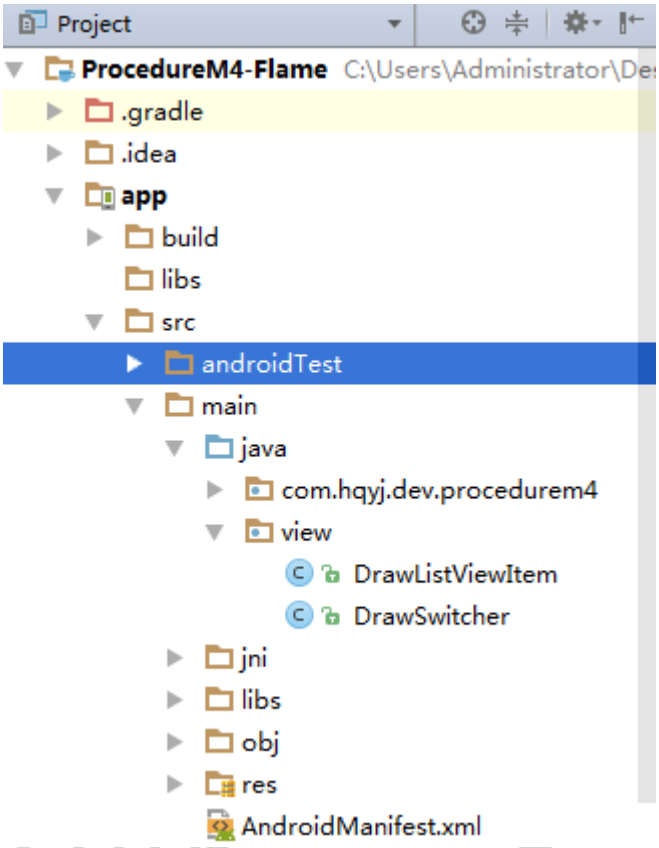
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：



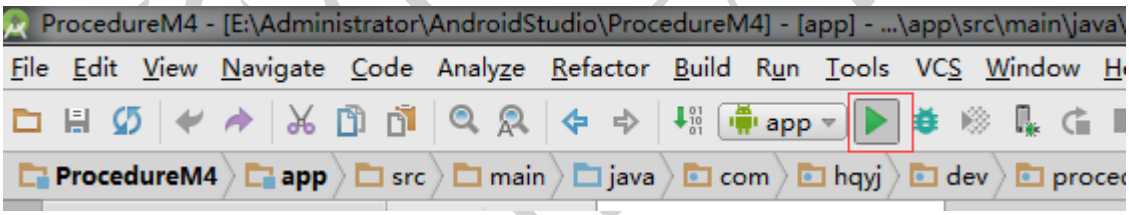
程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

代码写好之后，将代码编译并运行在 4412 板子上。

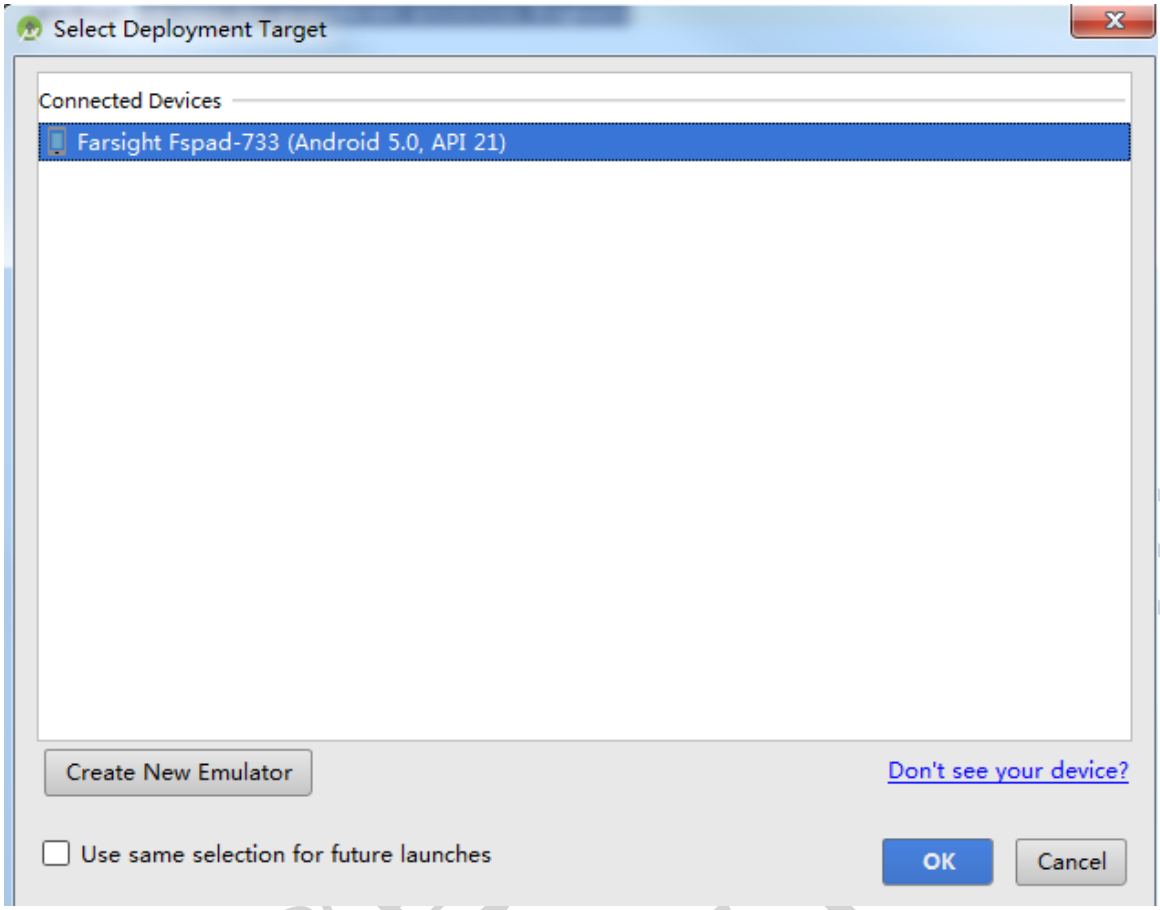
然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

华清远见 dev.hqyj.com

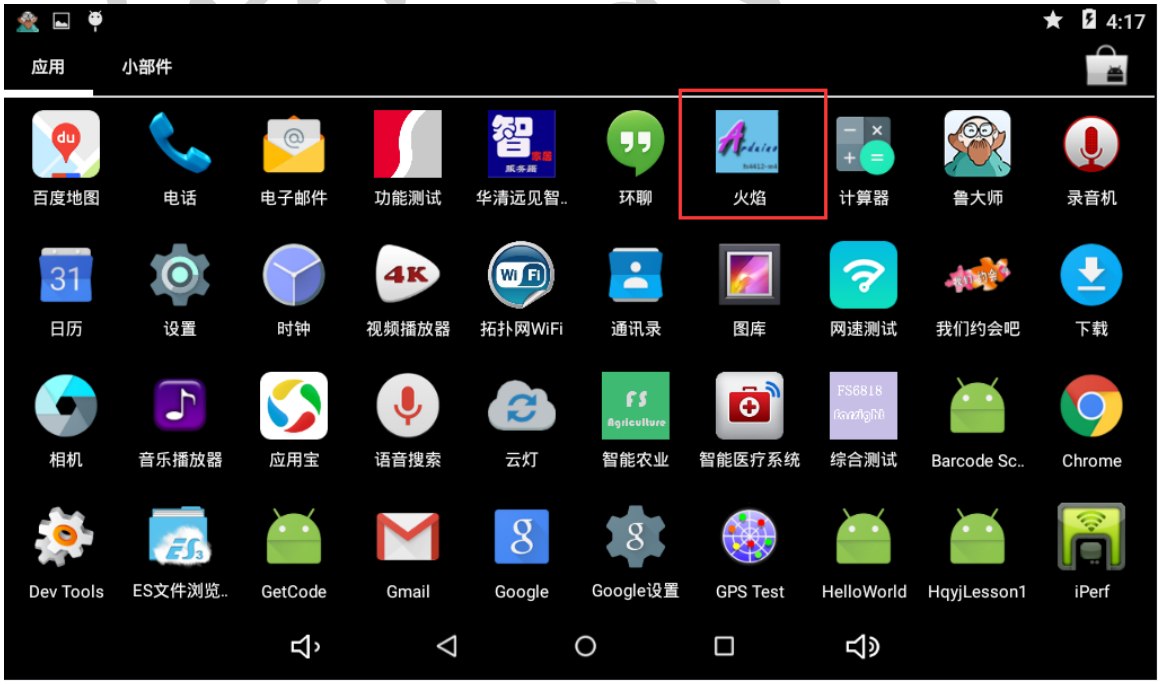
连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：



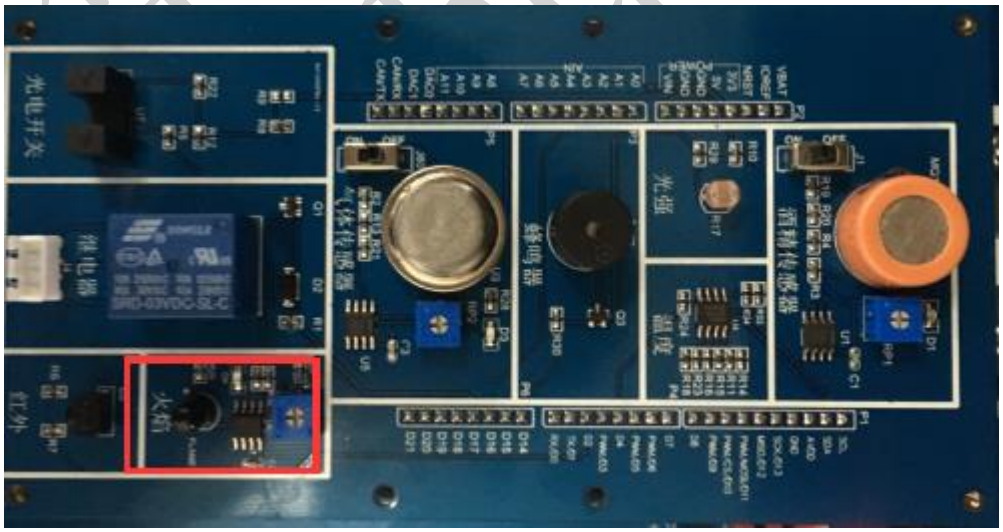
点击进入程序进行测试。

3.15.6 【实验结果】

将拨码开关中的 AD3 拨至 ON，其他拨码全为 OFF，如图中提示。

可以用打火机点着后靠近，可以看到上面数字会变化，离开的时候又变回来。

火焰传感器在模块中的位置如下图所示：



3.15.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-12remin.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 adc 的数据

include $(BUILD_SHARED_LIBRARY)
```

读取 adc 数据的.c 文件

源码位置：ProcedureM4-12remin.app.src.main.jni.operate_adc.c

NormalText Code

```
/**
*读取 adc 数据
*/
int read_adc(int which){
    int fd;
    int value;
    //以可读可写的形式打开"/dev/adc"这个路径下的 adc，创建流通道
    fd = open("/dev/adc", O_RDWR);
    if (fd < 0){
        LOGI("Open ADC error");
        return fd;
    }
    switch(which){
        case ADC_ALCOHOL:
            ioctl(fd, SET_CHANNEL, ALCOHOL_CHANNEL);
            break;
        case ADC_LIGHT:
```

```
            ioctl(fd, SET_CHANNEL, LIGHT_CHANNEL);
            break;
        case ADC_SENSITIVE://设置热敏电阻传感器传输数据通道
            ioctl(fd, SET_CHANNEL, SENSITIVE_CHANNEL);
            break;
        case ADC_GAS:
            ioctl(fd, SET_CHANNEL, GAS_CHANNEL);
            break;
        default:
            LOGI("ERROR ADC");
            break;
    }
    //读取周围温度数据
    read(fd, &value, sizeof(value));
    LOGI("value : %d\n", value );
    //关闭流
    close(fd);
    return value;
}
```

（2） Android 端对数据的接受和处理

在具体的热敏电阻的 Fragment 中，开启线程读取传感器的数据，然后用 handler 异步处理，对 UI 界面进行动态显示的改变。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.ThermistorFragment.java

NormalText Code

```
public class ThermistorFragment extends Fragment {

    private View mView;
    private Thermistor thermistor;
    private TextView textView;
    private ThermistorReadThread thermistorReadThread;
    private boolean threadOn = false;
    private String TAG = "THERMISTOR";
    /**
     *装载名字叫"operate"的库文件
     */
    static {
        System.loadLibrary("operate");
    }
    /**
```

```
16      * 异步处理机制，实现子线程更新 UI 界面的功能
17      */
18     @SuppressLint("HandlerLeak")
19     private Handler handler = new Handler(){
20         @Override
21         public void handleMessage(Message msg) {
22             super.handleMessage(msg);
23             switch (msg.what){
24                 case 1://将周围温度显示在 IU 界面上
25                     String value = msg.getData().getString(TAG);
26                     textView.setTextSize(50);
27                     textView.setText(String.format("%sV", value));
28                     break;
29                 default:
30                     break;
31             }
32         }
33     };
34     @Override
35     public void onCreate(@Nullable Bundle savedInstanceState) {
36         super.onCreate(savedInstanceState);
37         thermistor = Thermistor.getThermistor();
38     }
39     @Nullable
40     @Override
41     public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
42                         container, @Nullable Bundle savedInstanceState) {
43         mView = inflater.inflate(R.layout.thermistor_fragment, container, false);
44         initShow();//初始化控件
45         threadOn = true;
46         thermistorReadThread = new ThermistorReadThread();
47         thermistorReadThread.start();//开启线程
48         return mView;
49     }
50     private void initShow() {
51         textView = (TextView) mView.findViewById(R.id.txv_thermistor);
52     }
53     @Override
54     public void onDestroy() {
55         super.onDestroy();
56     }
57     @Override
58     public void onDestroyView() {
```

```
        super.onDestroyView();
        //关闭线程
        threadOn = false;
        thermistorReadThread.interrupt();
        thermistorReadThread = null;
    }
    /**
     *每隔两秒读取一次热敏电阻传输的数据
     */
    private class ThermistorReadThread extends Thread{
        @Override
        public void run() {
            super.run();
            while (threadOn){
                Bundle b = new Bundle();
                Message msg = new Message();
                //将数据显示成"#.##"这种格式
                DecimalFormat df = new DecimalFormat("#.##");
                //读取热敏电阻传输的数据
                int value = thermistor.operate.read()[0];
                String result = df.format((double)value*7.2/4096);
                b.putString(TAG, result);
                Log.d(TAG, result);
                msg.what = 1;
                msg.setData(b);
                //将转化好多数据格式交给 handller 处理
                handler.sendMessage(msg);
                try {
                    sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

3.16 数码管实验

3.16.1 【实验目的】

（1） 掌握数码管显示的工作原理；

（2） 掌握数码管对数据的接受和处理；

（3） 掌握在 Android 系统下对数码管传输数据开发流程；

3.16.2 【实验环境】

（1） Android Sutdio 开发平台；

（2） Android5.0 系统；

（3） u-boot-2010.03 版本；

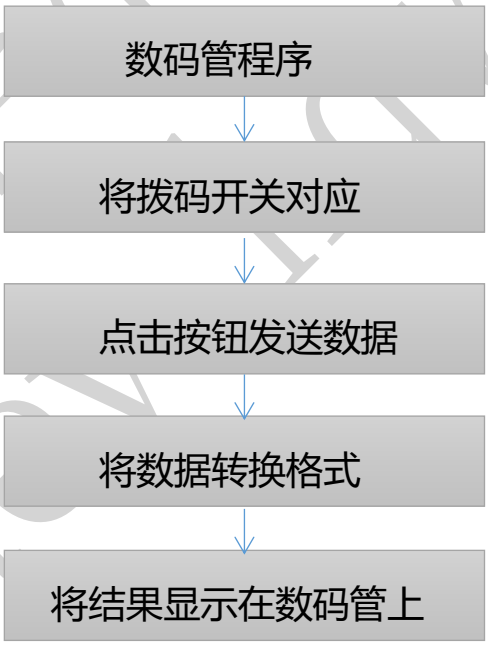
3.16.3 【实验内容】

基于 Android5.0 系统实现数码管显示数据的功能。

3.16.4 【实验原理】

硬件原理参加第二章的按键扫描数码管显示驱动实验；

在应用程序里点击按钮，通过调用接口向数码管发送数据，并显示在数码管上。

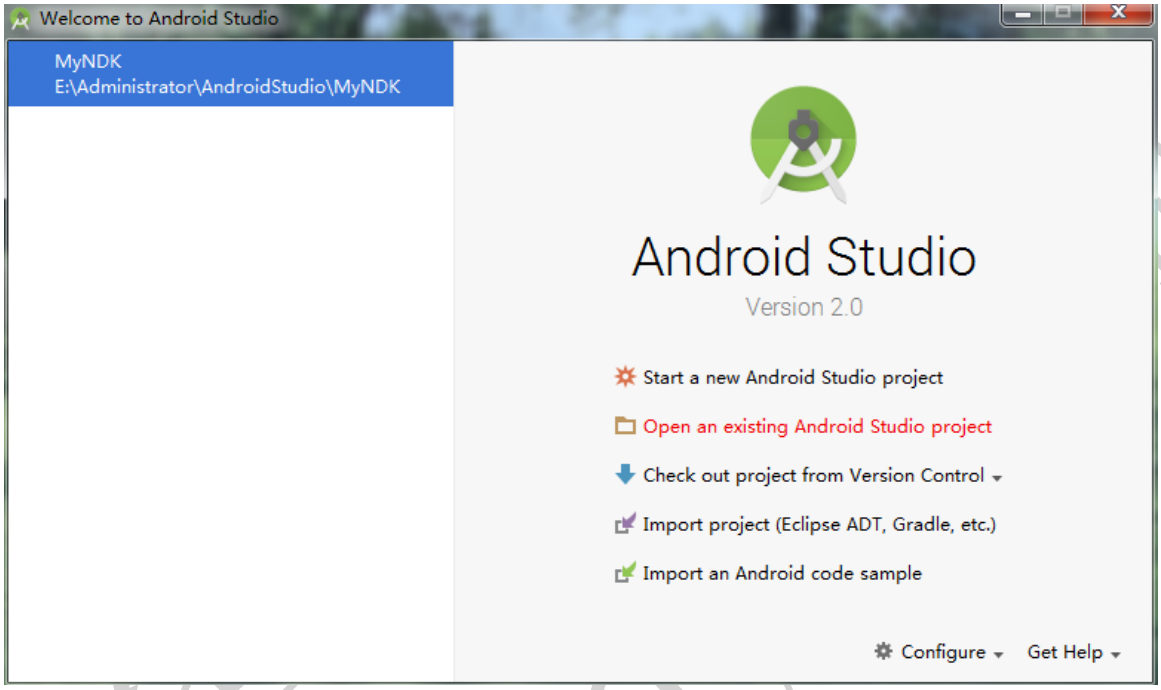


华清远见 www.farsight.com.cn

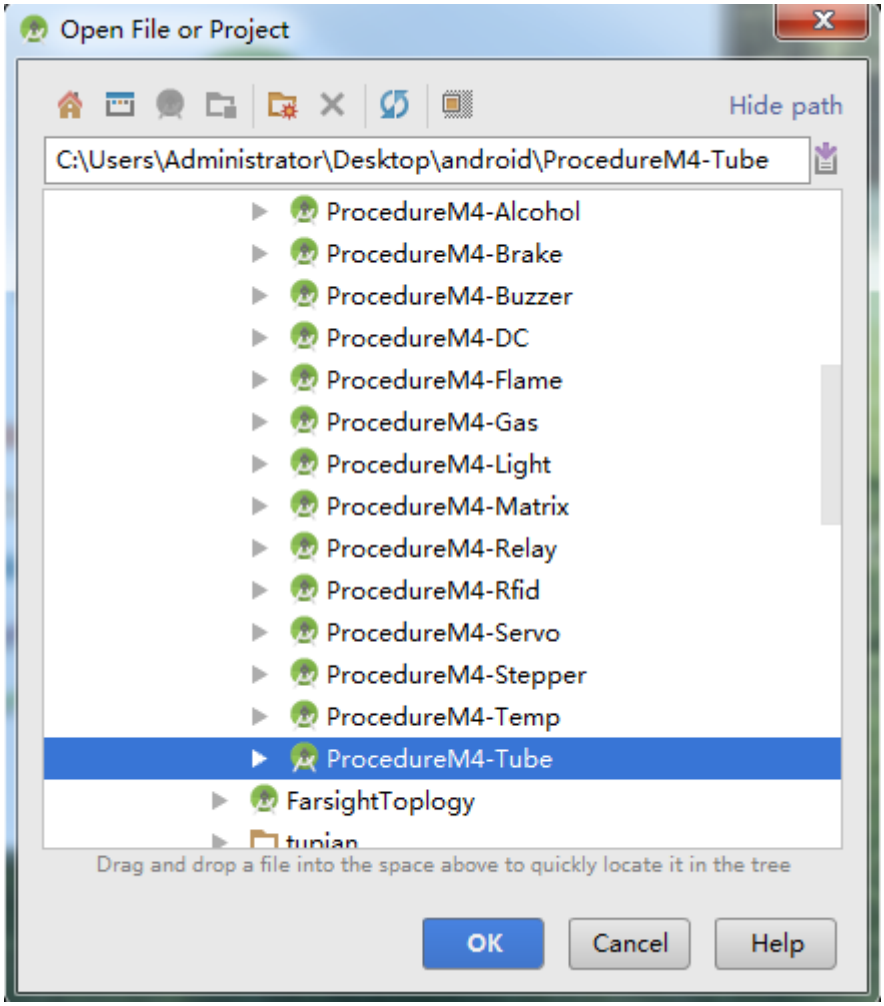
3.16.5 【实验步骤】

数码管实验源码在光盘【华清远见-嵌入式 ARM 实验箱资料-II\程序源码\Android 实验源码\ProcedureM4-Tube】。用户可以将其解压到电脑任意位置，这里将程序解压到桌面的 android 文件夹下

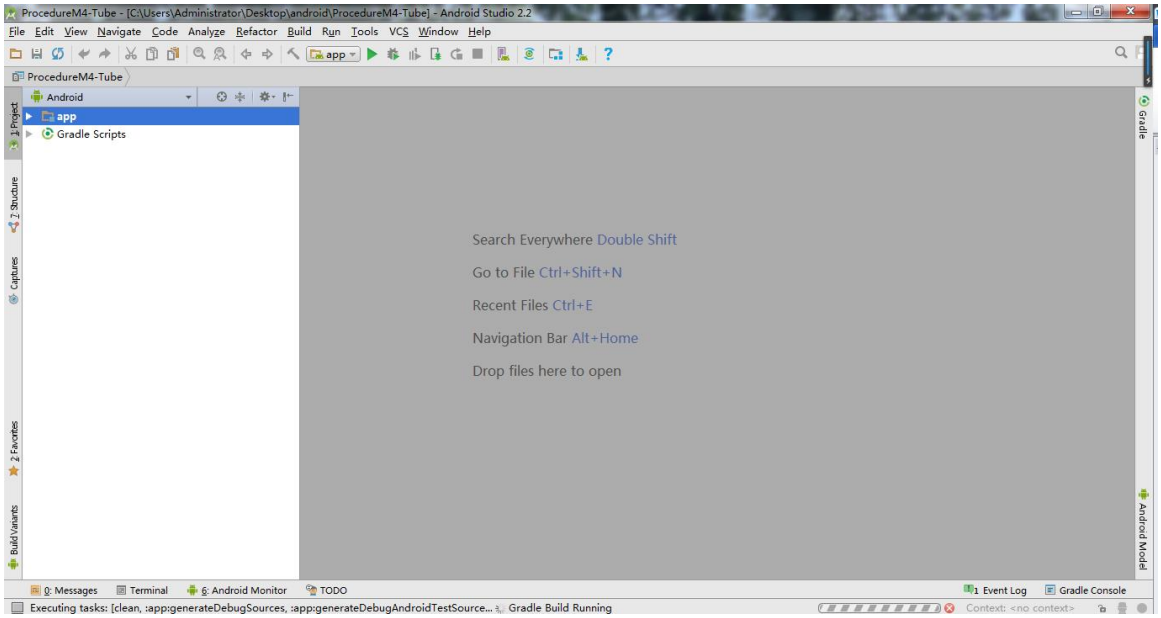
打开 Android studio，导入实验程序：



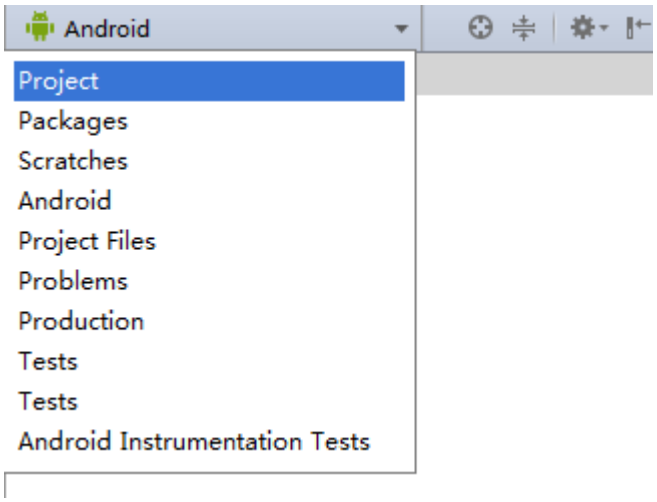
选择第二项“open an existing Adnroid Studio project”，打开刚才解压的程序：



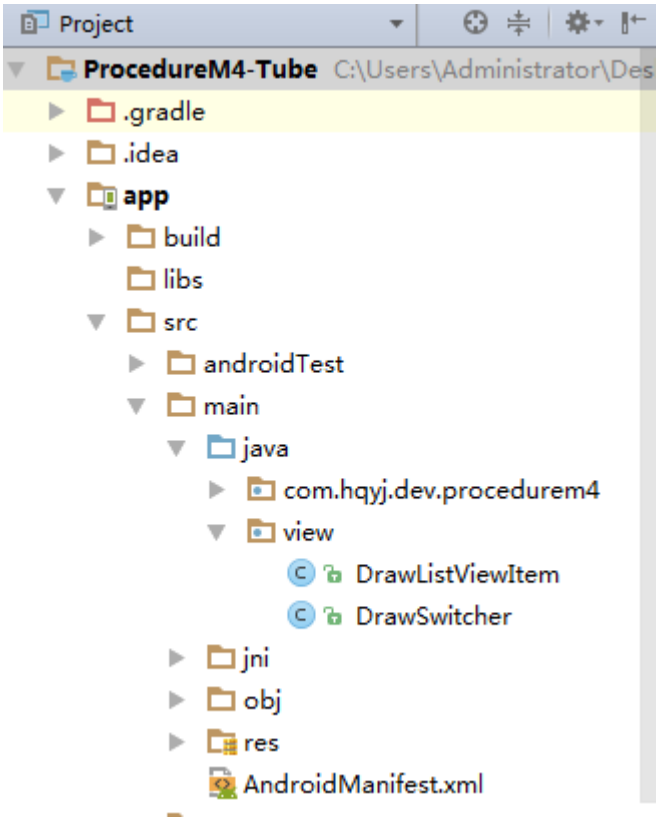
选中之后点击“ok”



点击 Android，选择 project：



程序部分位置如下图所示：

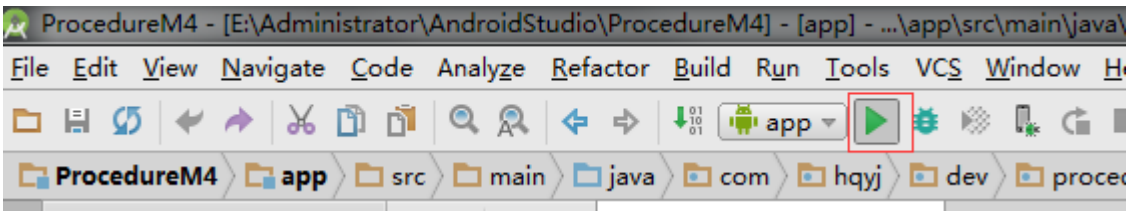


程序中涉及到 NDK 的部分可以参考《FS-AS 资料》里的“AS（android studio）使用手册”，其中对 NDK 的安装和使用有详细介绍。

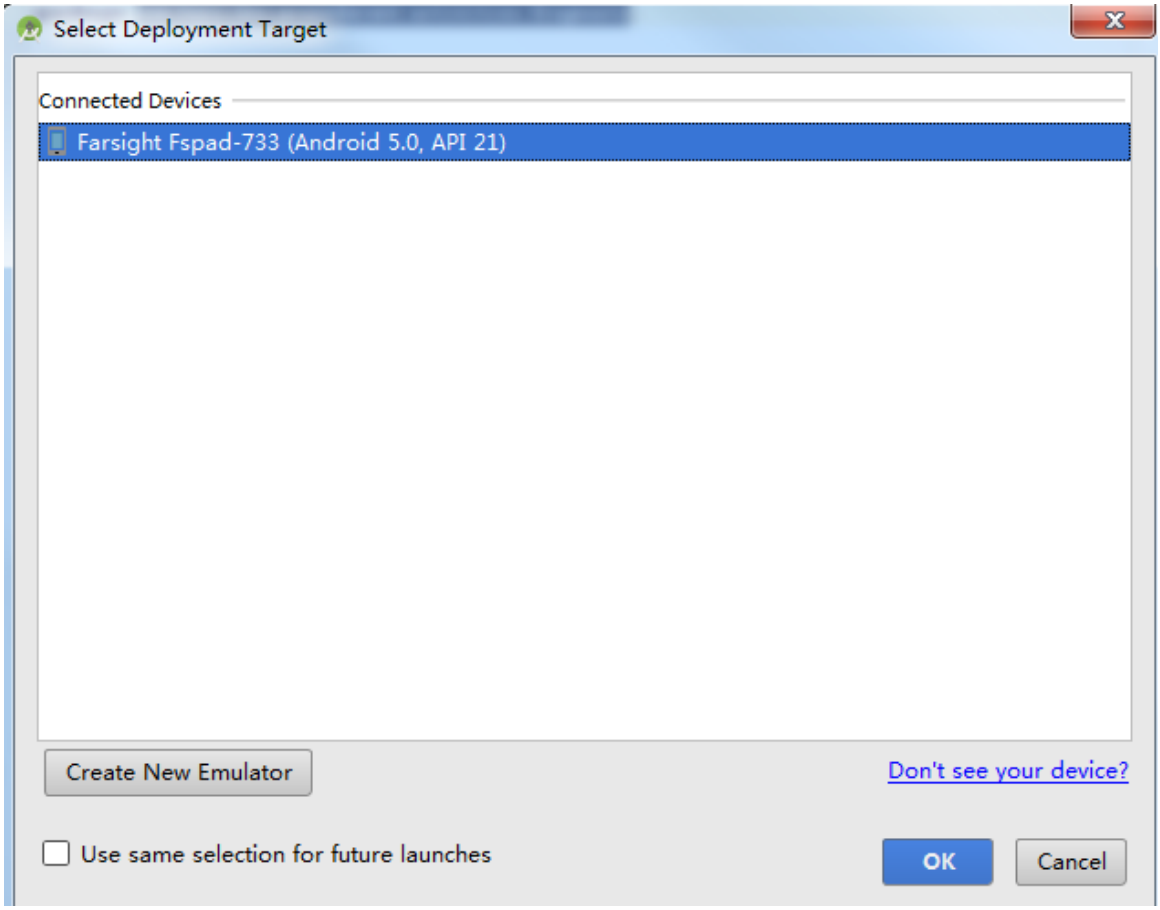
代码写好之后，将代码编译并运行在 4412 板子上。

然后将 FS4412 启动拨码拨至 EMMC 启动，即 0110，给开发板连接 5V 直流电源、micro 线连接电脑。如下图：

连接并启动板子之后，点击 Android Studio 上面的运行按钮，将程序在板子上运行起来：



点击“ok”按钮，开始运行：



安装成功，之后在 4412 板子上可以看到此应用程序：

点击进入程序进行测试。

3.16.6 【实验结果】

该实验只需要注意 SW8 拨码拨至 A8/A9，因为在硬件设计时，将 SW8 作为 MCU 和 CUP 的 i2c 主机切换开关，如图中提示。

点击键盘，在数码管里面会显示出点击的数字，并且会依次向后排列。

数码管在模块中的位置如下图所示：

3.16.7 【源码分析】

将刚才解压的源码文件进行分析

（1） Android.mk 文件

在配置文件里面配置对应的文件名称和读取数据类型

源码位置：ProcedureM4-13shuma.app.src.main.jni.Android.mk

NormalText Code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS        := -lm -llog
LOCAL_MODULE        := operate //库文件的名称
LOCAL_SRC_FILES     := operate.c operate_adc.c operate_brake.c operate_zlg.c
LOCAL_SRC_FILES     += operate_compass.c operate_dc.c operate_steeper.c
LOCAL_SRC_FILES     += operate_buzzer.c operate_relay.c operate_tube.c
LOCAL_SRC_FILES     += operate_rfid.c operate_servo.c//读取 tube 的数据

include $(BUILD_SHARED_LIBRARY)
```

发送对 tube 控制的命令的.c 文件

源码位置：ProcedureM4-13shuma.app.src.main.jni.operate_tube.c

NormalText Code

```
/**
*设置数码管将要显示的数值，
*/
void set_tube(int operate){
    char buf[8];
    //定义一个大小为 8 的 buf，因为数码管最多显示 8 个数字,并打开一个流。
    int fd = open(TUBE_FILE, O_RDWR);
    if(fd < 0){
        LOGI("ERROR OPEN: %s", TUBE_FILE);
        return;
    }
    int i;
    //设置 buf 里面都是 0
    bzero(buf, sizeof(buf));
    //将想要设置的数字保存到 buf 中
    for( i = 0; i < 8; i++){
        char get = (char)(((operate >> (i * 4)) & 0x0f));
        if (get ==0x0f){
            get =' ';
        } else {
            get = get + '0';
        }
        buf[7 - i] = get;
    }
    //发送设置数码管的命令。
    ioctl(fd, SET_VAL, buf);
    //关闭流
    close(fd);
}
```

（2） Android 端对数据的接受和处理

在具体的数码管的 Fragment 中，设置了 11 个按钮，分别是“初始化”、“系统时间”和数字 1—9。在点击按钮的时候是向数码管发送命令，让数码管把对应的按钮上的数字显示出来。

源码位置：com.hqyj.dev.procedurem4.activities.fragments.TubeFragment.java

NormalText Code

```
 public class TubeFragment extends Fragment implements View.OnClickListener {

  private View mView;
  private String TAG = "TUBE";
```

```
    private Tube tube;
    /**
     * 异步处理机制，实现子线程更新 UI 界面的功能
     */
    @SuppressLint("HandlerLeak")
    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what) {
                case 1://发送数码管显示数字的命令
                    tube.operate.write(msg.getData().getInt(TAG));
                    break;
                default:
                    break;
            }
        }
    };
    /**
     *装载名字叫"operate"的库文件
     */
    static {
        System.loadLibrary("operate");
    }
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        tube = Tube.getTube();
    }
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
                        container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.tube_fragment, container, false);
        //初始化 11 个按钮，并设置监听
        mView.findViewById(R.id.btn_tube_submit).setOnClickListener(this);
        mView.findViewById(R.id.btn_tube_system_time).setOnClickListener(this);
        mView.findViewById(R.id.btn_0).setOnClickListener(this);
        mView.findViewById(R.id.btn_1).setOnClickListener(this);
        mView.findViewById(R.id.btn_2).setOnClickListener(this);
        mView.findViewById(R.id.btn_3).setOnClickListener(this);
        mView.findViewById(R.id.btn_4).setOnClickListener(this);
        mView.findViewById(R.id.btn_5).setOnClickListener(this);
```

```
48          mView.findViewById(R.id.btn_6).setOnClickListener(this);
49          mView.findViewById(R.id.btn_7).setOnClickListener(this);
50          mView.findViewById(R.id.btn_8).setOnClickListener(this);
51          mView.findViewById(R.id.btn_9).setOnClickListener(this);
52          return mView;
53      }
54      @Override
55      public void onDestroyView() {
56          super.onDestroyView();
57      }
58      @Override
59      public void onDestroy() {
60          super.onDestroy();
61      }
62      /**
63       * 按钮的点击事件
64       * @param v 按钮所在的视图
65       */
66      @Override
67      @SuppressLint("DefaultLocale")
68      public void onClick(View v) {
69          switch (v.getId()) {
70              case R.id.btn_tube_submit://设置数码管显示如下字符串
71                  String valeu = "12345678";
72                  setCMD(valeu);
73                  break;
74              case R.id.btn_tube_system_time://设置数码管显示当前系统时间
75                  Calendar c = Calendar.getInstance();
76                  int hour = c.get(Calendar.HOUR_OF_DAY);
77                  int minute = c.get(Calendar.MINUTE);
78                  int second = c.get(Calendar.SECOND);
79                  String valueT =
80                          String .format("%02d %02d %02d", hour, minute, second);
81                  setCMD(valueT);
82                  break;
83              case R.id.btn_0://设置数码管显示字符串“0”
84                  setStringShow(0);
85                  break;
86              case R.id.btn_1://设置数码管显示字符串“1”
87                  setStringShow(1);
88                  break;
89              case R.id.btn_2://设置数码管显示字符串“2”
90                  setStringShow(2);
```

```
                break;
            case R.id.btn_3://设置数码管显示字符串“3”
                setStringShow(3);
                break;
            case R.id.btn_4://设置数码管显示字符串“4”
                setStringShow(4);
                break;
            case R.id.btn_5://设置数码管显示字符串“5”
                setStringShow(5);
                break;
            case R.id.btn_6://设置数码管显示字符串“6”
                setStringShow(6);
                break;
            case R.id.btn_7://设置数码管显示字符串“7”
                setStringShow(7);
                break;
            case R.id.btn_8://设置数码管显示字符串“8”
                setStringShow(8);
                break;
            case R.id.btn_9://设置数码管显示字符串“9”
                setStringShow(9);
                break;
        }
    }
    /**
     *将字符串转化为需要的 int 值，并将转化后的结果交个 handler 进行处理
     * @param value
     */
    void setCMD(String value){
        int operate = 0;
        //将字符串转化为数字
        for (int i = 0; i < value.length(); i++) {
            if (value.charAt(i) == ' ') {
                operate = operate << 4 | (0x0f);
            } else {
                operate = operate << 4 | ((value.charAt(i) - '0') & 0x0f);
            }
        }
        Bundle bundle = new Bundle();
        bundle.putInt(TAG, operate);
        Message msg = new Message();
        msg.setData(bundle);
        msg.what = 1;
```

```
134         //将转化后的结果交个 handler 进行处理
135         handler.sendMessage(msg);
136     }
137     StringBuilder stringBuilder = new StringBuilder();
138     /**
139      * 将单个输入的数字经过叠加但是不能超过 8 个，然后显示在数码管上
140      * @param x 将要显示的数字
141      */
142     void setStringShow(int x){
143         stringBuilder.append(x);
144         if (stringBuilder.length() > 8){
145             stringBuilder.deleteCharAt(0);
146         }
147         setCMD(stringBuilder.toString());
148     }
149 }
```