成绩：

# 湖南科技学院

## 电子与信息工程学院

《嵌入式体系结构与编程》

# 项 目 设 计 报 告 书

题　　目 智能家居系统的设计与开发

姓　　名 _____ 学号 _____

同 组 成 员 _____

专业、年级 2013级　计算机科学与技术专业

指 导 教 师 _____

2015 年 12 月

自我总结：

评语：

老师签名：

# 目录

目录 ..... 1
第 1 章 引言 ..... 3
第 2 章 工具软件与技术基础 ..... 4
2.1 智能家居系统的开发环境和工具 ..... 4
2.2 智能家居系统的技术路线 ..... 4
2.3 设备连接 ..... 5
第 3 章 系统总体设计 ..... 6
3.1 智能家居系统需求分析 ..... 6
3.2 开发框架 ..... 6
3.3 具体实现设计 ..... 7
3.3.1 如何点亮 LED 灯 ..... 7
3.3.2 Linux 内核编译及配置 ..... 8
3.3.3 内核模块编程 ..... 9
第 4 章 界面设计 ..... 11
4.1 智能家居系统界面设计 ..... 11
第 5 章 系统实现 ..... 12
5.1 Android 客户端 ..... 12
5.2 Ubuntu 服务端 ..... 16
5.3 Ubuntu 客户端 ..... 18
5.4 Linux 内核驱动 ..... 20
5.5 LED 裸奔代码 ..... 21
第 6 章 结束语 ..... 23

# 第 1 章 引言

当今社会，随着生活质量的日益改善和生活节奏的不断加快，人们的工作、生活日益信息化。信息化社会改变了人们的生活方式与工作习惯，使得家居系统的智能化成为一种消费需求，智能家居系统越来越被重视。因此，将家庭中各种通信设备、家用电器和家庭安保装置通过家居控制系统进行整合，并进行远程控制和管理，已经成为近年来一个热门研究课题。

迄今为止，智能家居还没有一个普遍认同的统一的定义。通常智能家居系统利用先进的计算机和网络通讯技术将与家居生活有关的各种各样的子系统，通过特定的网络有机地结合在一起，通过科学管理，让家居生活更加舒适、有效、安全和节能。

以住宅为平台，兼备建筑、网络通信、信息家电、设备自动化，集系统、结构、服务、管理、控制为一体的高效、舒适、安全、便利、环保的居住环境。智能家居包含智能照明、电动窗帘、背景音乐、安防报警、楼宇对讲、远程控制等方面。

传统的家居智能控制系统一般采用有线方式来组建，如同轴电缆、USB、CAN 总线等。但有线网络具有布线麻烦，可扩展性差等固有的缺点，限制了有线网络技术在智能家居系统中的发展。因此，基于物联网，将无线网络技术应用于家庭网络已成为大势所趋。这不仅仅因为无线网络具有更大的灵活性、流动性，省去了布线的麻烦，更重要的是它符合家居控制网络的通讯特点。无线家居网络将人们生活与工作的广袤空间浓缩于人类的双手可以掌控的距离。

此次我们围绕基于网络的智能家居系统开发，简单的实现了使用网络对嵌入式设备的控制。

# 第 2 章 工具软件与技术基础

## 2.1 智能家居系统的开发环境和工具

智能家居系统的开发工具主要是： Ubuntu系统、 putty、 Eclipse。

Ubuntu（乌班图） 是一个以桌面应用为主的基于 Linux 内核开发的操作系统, 其名称来自非洲南部祖鲁语或豪萨语的“ ubuntu ”一词，意思是“ 人性 ”、“ 我的存在是因为大家的存在 ”，是非洲传统的一种价值观，类似华人社会的“ 仁爱 ”思想。 Ubuntu基于 Debian发行版和 GNOME桌面环境，而从 11.04版起， Ubuntu发行版放弃了 Gnome桌面环境，改为 Unity，与 Debian的不同在于它每 6个月会发布一个新版本。 Ubuntu的目标在于为一般用户提供一个最新的、 同时又相当稳定的主要由自由软件构建而成的操作系统。 Ubuntu具有庞大的社区力量， 用户可以方便地从社区获得帮助。 2013年1月3日，Ubuntu正式发布面向智能手机的移动操作系统。ubuntu基于 linux 的免费开源桌面 PC操作系统，十分契合英特尔的超极本定位，支持 x86、64位和 ppc架构。

putty是一个 Telnet、 SSH、 rlogin、纯 TCP以及串行接口连接软件。较早的版本仅支持 Windows平台，在最近的版本中开始支持各类 Unix 平台，并打算移植至 Mac OS X上。除了官方版本外，有许多第三方的团体或个人将 putty移植到其他平台上，像是以 Symbian为基础的移动电话。 putty为一开放源代码软件，主要由 Simon Tatham维护，使用 MIT licence 授权。随着 Linux 在服务器端应用的普及, Linux 系统管理越来越依赖于远程。在各种远程登录工具中， Putty是出色的工具之一。 putty是一个免费的、 Windows 32平台下的 telnet、rlogin 和 ssh客户端，但是功能丝毫不逊色于商业的 telnet类工具。

Eclipse 是一个开放源代码的、基于 Java的可扩展开发平台。就其本身而言，它只是一个框架和一组服务， 用于通过插件组件构建开发环境。 幸运的是，Eclipse 附带了一个标准的插件集，包括 Java开发工具（ Java Development Kit, JDK）。使用 Eclipse添加 Android SDK 和 ADT 后可进行 Android 项目的编写。

## 2.2 智能家居系统的技术路线

智能家居系统采用 C-S模式，客户端基于 Android 开发，服务端基于 C开发，服务端与客户端采用 Socket网络编程来进行通讯。服务端主要对开发板上各传感

器的数据进行处理、 响应客户端的各种请求， 服务端主要对从服务器获取的信息进行处理展示， 同时给服务器发送控制信息， 进行从而实现对开发板上某些功能部件的的控制。主要用到的技术是： Java Socket编程、 C Socket编程、 Ubuntu下Vim编辑器的使用、 Linux网络编程、相关驱动的开发、 putty串口连接开发板、TFTP（简单文件传输协议）进行文件下载。

## 2.3 设备连接

开发板的串口与计算机串口使用串口线进行连接， 将开发板使用网线接入局域网，再将当前计算机接入当前开发板的所在的局域网， 接通开发板电源， 同时将开发板、计算机中 Windows系统和 Ubuntu系统下三者的 IP设置在同一网段，设置开发板、Windows和Ubuntu的网关为同一网关， 此时开发板的服务端是 Ubuntu。

# 第 3 章 系统总体设计

## 3.1 智能家居系统需求分析

本系统设计的目的是向用户提供一个通过网络控制智能家居设备的系统。 通过Socket编程进行客户端和服务端的通讯，实现对智能家居设备的控制。

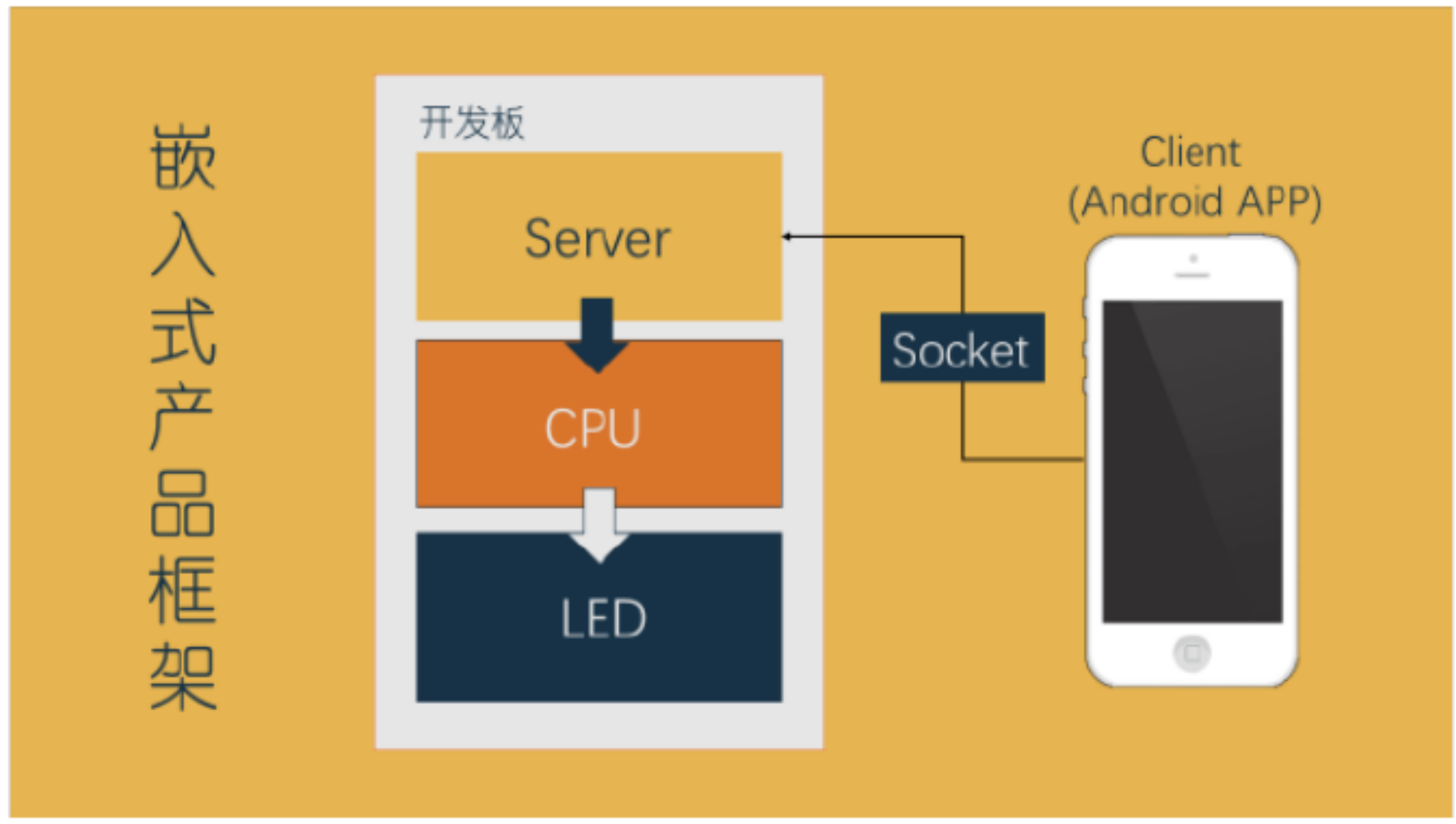
随着电子技术和网络技术的飞速发展， 人们的生活方式日益发生着变化， 人们对于生活有了更高层次的追求。 大家不再因拥有一个可以居住的空间而感到满足，更希望自己的家庭生活是便利的、舒适的、安全的，并且还兼具人性化、智能化，从而产生了智能家居系统 [1]。

与普通的家居相比，智能家居不仅具有一般的居住功能， 也能提供舒适安全、高品位且宜人的家庭生活空间， 还能提供全方位的信息交换功能， 帮助家庭与外部保持信息交流畅通， 优化人们的生活方式， 帮助人们有效安排时间， 增强家居生活的安全感，节约能源。

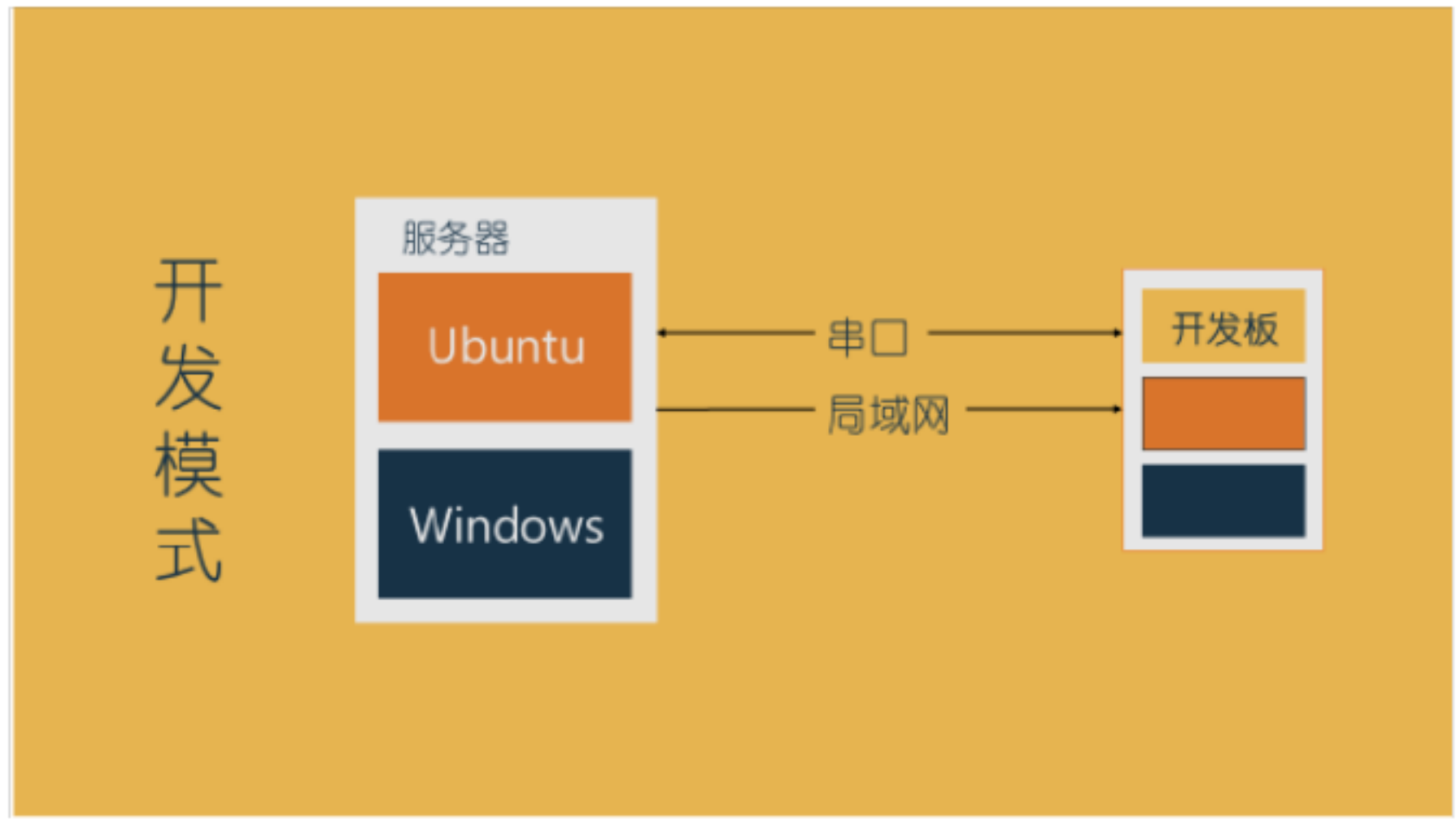
因此，智能家居必然成为今后日常生活的一种趋势。 而对智能家居系统的需求也将随着人们对生活品质的追求而发生着变化， 因此，智能家居系统也拥有着良好的发展前景。

## 3.2 开发框架

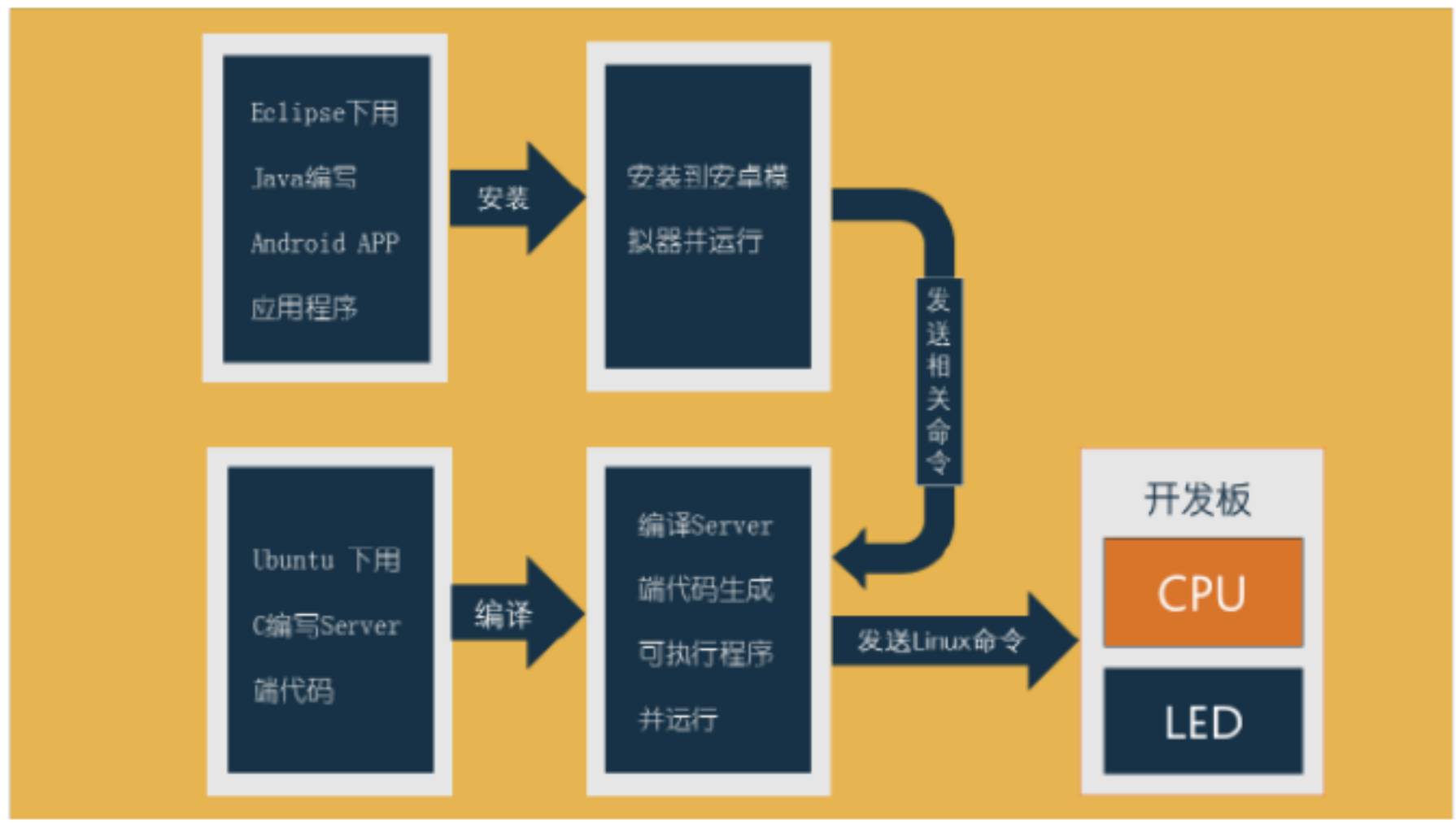
基本的嵌入式产品框架：



开发模式：




实现过程：




## 3.3 具体实现设计

### 3.3.1 如何点亮 LED 灯

1）原理

1.查看S5PC100开发手册，找到 4个LED灯的对应引脚 GPG3_0/3；

2.要点亮 LED灯，必须在 LED两端有正向压降，即 GPG3_0(LED4),GPG3_1(LED1),GPG3_2 (LED2),GPG3_3(LED3)要置为高电平；

3.控制寄存器： GPG3CON设置为输出引脚（ GPG3CON |= 0x1111 << 0）；

4.数据寄存器： GPG3DAT设置为高电平（ GPG3DAT |= 0xf << 0）；

2）运行裸奔代码

1.使用C语言编译点亮 LED 灯的逻辑代码；

2.设置好开发板上的参数

--serverip set serverip +服务器 ip地址

--ipaddr set ipaddr +开发板 ip

--gatewayip set gatewayip +网关

测试开发板与服务器的连通性

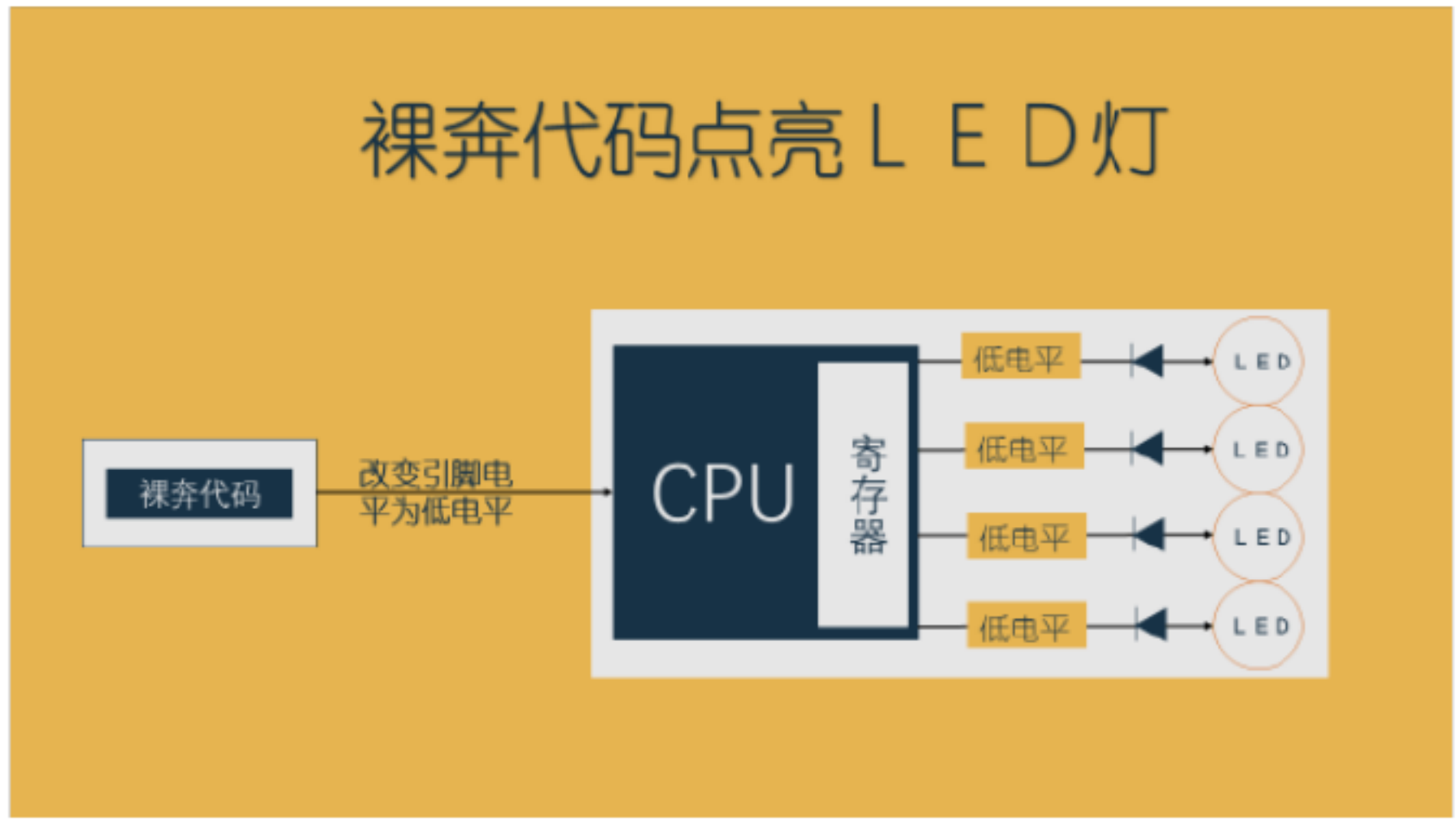
ping + 服务器 ip地址

3.设置启动方式

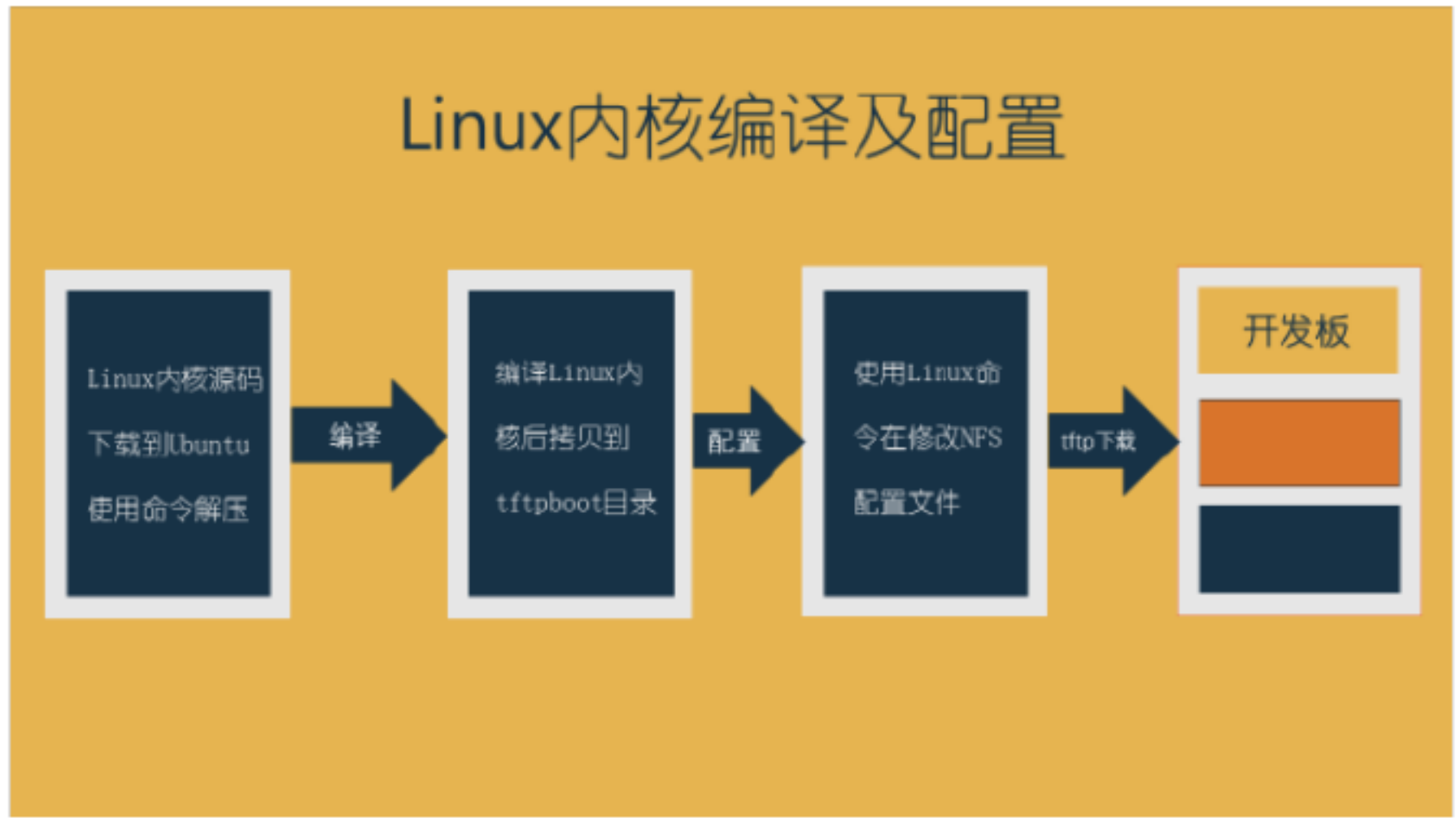
set bootcmd t zImage\; bootm 0x20008000

（这里的 0x20008000是寄存器的地址， 我们需要把在 Ubuntu下编译好的文件下载到这个寄 存器）

go 0x20008000 回车 （LED灯亮）




## 3.3.2 Linux 内核编译及配置




### 3.3.3 内核模块编程

1）先进入 Linux 内核所在目录 ,并调用 top Makefile 生成 test.o, 运行 MODPOST 生成临时文件 test.mod.c,然后根据 test.mod.c 生成 test.mod.o(产生了 Linux 所采用的可执行 /可链接的 ELF 文件格式 ) 最后,将 test.o 和 test.mod.o 链接生成 test.ko

2）模块的安装

1. make modules_install

命令把所有的 .ko 文件拷出来放到文件系统的

/lib/modules/2.6.30.4-liy/kernel/ 下的相应目录 (自动创建目录 )

2. 指定安装系统的路径放到我们开发板的根文件系统

make modules_install INSTALL_MOD_PATH=/nfs/rootfs

3）注册字符设备驱动

linux/fs.h

static inline int register_chrdev(unsigned int major, const char *name,

const struct *fops)

1、 major 主设备号

a) 内核主设备号文档 .

Documentation/device.txt

240-254 char LOCAL/EXPERIMENTAL USE

b) cat /proc/devices

c) 建立设备文件：mknod /dev/test c 241 0

crw-r--r-- 1 0 0 241, 0 Nov 17 2013 /dev/test

2、name 名字

name:设备驱动名，一看到 char*这种类型，很大可能用 cat /proc/devices 可以看到 .

3、 fops 驱动接口集合

fops:指针类型，一般情况下如果形参是指针，那么很大可能它会在调用函数中被初始化

4）编译驱动

a.把程序拷贝到虚拟机

b.编译 make

make之后生成一个可执行文件 led_driver.ko

c. arm-none-linux-gnueabi-gcc -o app app.c

程序执行：

a.插入和拔出驱动模块

# insmod led_driver.ko

# mknod /dev/led c 249 0

# ./app on led灯亮

# ./app off led灯灭

# 第 4 章 界面设计

## 4.1 智能家居系统界面设计


此页面为系统首页， 已注册过的用户输入自己的用户名和密码登录智能家居系统。登录后即可将以设置的命令自动发送至服务器， 服务器接受命令后执行相关代码，即可完成对开发板的控制。

# 第 5 章 系统实现

本系统设计的目的是向用户提供一个使用 Android APP客户端并通过网络控制智能家居设备的系统。原理是基于 Socke编程进行客户端和服务端的通讯，实现对智能家居设备的控制。

## 5.1 Android 客户端

Android客户端用 Eclipse进行编程，使用 Java语言，负责用户登录和命令发送，具体实现代码如下：（已省略源码中的导包代码）

```
public class MainActivity extends Activity {
    private EditText editText_name;
    private EditText editText_pwd;
    private Button button_login; //
    private Socket clientSocket;
    //  用来处理子线程发送过来的消息的
    private Handler mHandler = new Handler() {
        //处理消息
        public void handleMessage(android.os.Message msg) {
            switch (msg.what) {
            case 0x11:
                //获取消息中的内容
                String buffer =(String) msg.obj;
                //展示在 UI 界面上
                Toast.makeText(MainActivity.this, buffer,
Toast.LENGTH_SHORT).show();
                break;
            default:
                break;
            }
        };
    };
```

```
  @Override
//当我们运行应用程序的时候，就会自动的调用这个方法
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
          //  指定这个 Activity 所使用的布局
        setContentView(R.layout.activity_main);
          // 1. 获取布局中的控件，根据控件  id来获取
    editText_name = (EditText) findViewById(R.id.editText1);
    editText_pwd = (EditText) findViewById(R.id.editText2);
    button_login = (Button) findViewById(R.id.button1);
          // 2. 为按钮设置监听器，用来监听用户的点击事件
    button_login.setOnClickListener(new OnClickListener() {
            @Override
        //  当用户点击按钮的时候，会自动的调用这个方法
        public void onClick(View v) {
                // 1. 获取用户名，密码
                String name = editText_name.getText().toString();
                String pwd = editText_pwd.getText().toString();
                    // 2. 与服务端建立连接  ,就需要创建子线程  ,并启动子线程
                new Thread(new ConnectionThread(name,pwd)).start();
               }
        });
}
  //创建子线程
public class ConnectionThread implements Runnable {
    private String name;
    private String pwd;
    //构造方法
    public ConnectionThread(String name,String pwd) {
        this.name =name;
        this.pwd =pwd;
```

```
}
@Override
//  线程的执行体
public void run() {
        try {
                // 1.  与服务器建立连接
                System.out.println("---與服務器建立連接  -->>");
                clientSocket =new Socket();
                clientSocket.connect(new InetSocketAddress("172.20.135.6",
7775));
                // 2.  获取网络的输入，输出流
                OutputStream outputStream =clientSocket.getOutputStream();
                InputStream inputStream =clientSocket.getInputStream();
                //将这个输入流转换为缓冲流
                BufferedReader bufferedReader =new BufferedReader(new
InputStreamReader(inputStream));
                // 3.  接受服务端的信息
                readMsgServer(bufferedReader);
                // 4.  发送用戶名和密碼
                writeMsgServer(outputStream,name,pwd);
                // 5.  接收服务端发送过来的验证信息
                readMsgServer(bufferedReader);
                //writeMsgServer(outputStream,"on");
                outputStream.write("on".getBytes(),0,"on".length());
                outputStream.flush();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
}
```

```
//  接收服务端的信息
public void readMsgServer(BufferedReader bufferedReader) {
        String line="";
        String buffer="";
        try {
                //获取服务端的数据
            while((line=bufferedReader.readLine())!=null){
                if(line.equals("start")){
                    continue;
                }else if(line.equals("end")){
                    break;
                }else{
                    //真正的数据
                    buffer =line+buffer;
                    //System.out.println("---->>"+buffer.toString());
                }
                //  发送消息给 UI 主线程，并展示在  UI 界面上
                Message message =Message.obtain();
                message.what =0x11;                         //消息的类型
                message.obj=buffer.toString();      //消息的内容
                //发送消息
                mHandler.sendMessage(message);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

//  发送消息給服务端
public void writeMsgServer(OutputStream outputStream,String name,String
```

```
pwd) {
                try {
                    //发送用戶名
                    outputStream.write(name.getBytes(), 0, name.length());
                    outputStream.flush();
                    //发送密码
                    outputStream.write(pwd.getBytes(),0,pwd.length());
                    outputStream.flush();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
```

## 5.2 Ubuntu 服务端

Ubuntu服务端用与接受客户端发送的命令， 并执行相应代码， 完成客户端对开发板的控制，使用 C语言进行编写，具体实现代码如下： （已省略源码中头文件代码）

```
    void *do_client(void *arg);
    int main(void)
    {     int ret;
        int servfd;
          /*1.建立socket套接字*/
        servfd = socket(AF_INET,SOCK_STREAM,0);
        if(-1 == servfd)
    {     perror("socket");
        return -1;
    }
      /* 填充服务器地址和端口   */
    struct sockaddr_in servaddr;
    memset(&servaddr,0,sizeof(servaddr));
```

```
servaddr.sin_family = AF_INET;   //ipv4
servaddr.sin_port = htons(PORT);  //端口号
servaddr.sin_addr.s_addr = inet_addr(SERVIP);服务器地址
socklen_t slen = sizeof(servaddr);
/*2.bind  绑定服务器的 ip地址，端口号 */
ret = bind(servfd,(struct sockaddr *)&servaddr,slen);
if(-1 == ret)
{      perror("bind");
       return -1;
}
/*3.listen  主动监听客户端  ,客户端连接最多为  100*/
ret = listen(servfd,100);
if(-1 == ret)
{      perror("listen");
       return -1;
}
  int newfd;
pthread_t tid;
while(1)
{      /*4.accept 接受客户端，并且为新连接的客户端
          分配一个新的文件描述符去操作它     */
       newfd = accept(servfd,NULL,NULL);
       if(newfd < 0)
       {continue;}
       printf("==>%d connect success...\n",newfd);
       /* 为每个客户创建一个线程去维护它     */
       ret = pthread_create(&tid,NULL,do_client,&newfd);
       if(0 != ret)
       {       perror("pthread_create");
              return -1;
       }
```

```
            pthread_detach(tid);
          }
      close(servfd);
        return    0;
}
void *do_client(void *arg)
{      int newfd = *(int *)arg;
      char buf[BUFSZ] = {0};
      int ret;
      char *str = "have a try and you will go far";
        while(1)
      {      /*5. 通信*/
            memset(buf,0,sizeof(buf));
            ret = read(newfd,buf,sizeof(buf)-1);
            if(ret > 0)
            {      printf("readdata: %s\n",buf);
                  /* 发送数据给客户端  */
                  write(newfd,str,strlen(str));
              }
          }
          pthread_exit(NULL);
      }
```

## 5.3 Ubuntu 客户端

Ubuntu客户端用与项目初期，Android 客户端尚未开发时，测试服务端接所编写的基于 C语言的 Ubuntu客户端，具体实现代码如下：（已省略源码中头文件代码）

```
      int main(void)
      {
            int clifd;
            int ret;
            char buf[BUFSZ];
```

```
  /*1.建立socket套接字*/
clifd = socket(AF_INET,SOCK_STREAM,0);
if(-1 == clifd)
{       perror("socket");
        return -1;
}
  /* 填充服务器地址和端口   */
struct sockaddr_in servaddr;
memset(&servaddr,0,sizeof(servaddr));
servaddr.sin_family = AF_INET;   //ipv4
servaddr.sin_port = htons(PORT);  //端口号
servaddr.sin_addr.s_addr = inet_addr(SERVIP);服务器地址
socklen_t slen = sizeof(servaddr);
/*2.conenct服务器 */
ret = connect(clifd,(struct sockaddr *)&servaddr,slen);
if(-1 == ret)
{       perror("connect");
        return -1;
}
struct pollfd pfd[2];
pfd[0].fd = 0;
pfd[0].events = POLLIN; // 不阻塞的读
  pfd[1].fd = clifd;
pfd[1].events = POLLIN; // 不阻塞的读
  /*3.通信*/
while(1)
{       ret = poll(pfd,2,-1);
        if(ret > 0)
        {// 至少有一个文件描述符返回
                if(pfd[0].revents == POLLIN)
                {       /* 从终端读取数据  */
```

```
        memset(buf,0,sizeof(buf));
        ret = read(0,buf,sizeof(buf)-1);
        if(ret > 0)
        {write(clifd,buf,ret);}
}
```

```
        if(pfd[1].revents == POLLIN)
        {      /* 服务器接收数据  */
            memset(buf,0,sizeof(buf));
            ret = read(clifd,buf,sizeof(buf)-1);
            if(ret > 0)
            {printf("serversay : %s\n",buf);}
        }
}
else
{      /* 错误或者超时  */
                break;
            }
        }
        close(clifd);
          return 0;
    }
```

## 5.4 Linux  内核驱动

```
    int main(int argc,char **argv)
    {     int fd;
    if(argc < 2)
    {      printf("Usage: on/off\n");
           return -1;
    }
    fd = open("/dev/led",O_RDWR);
    if(fd < 0)
```

```
{     perror("open");
      return -1;
}
if(!strncasecmp("on",argv[1],2))
      ioctl(fd,LED_ON);
else if(!strncasecmp("off",argv[1],3))
          ioctl(fd,LED_OFF);
      return 0;
}
```

## 5.5 LED 裸奔代码

LED裸奔代码用于控制 CPU引脚电压，从而间接控制 LED的点亮与熄灭，具体实现代码如下：（已省略源码中的头文件）

```
#define GPG3CON (*(volatile unsigned long *)0xE03001C0)
#define GPG3DAT (*(volatile unsigned long *)0xE03001C4)
int main(int argc,char **argv)
{     led_init();
      led_flow_water();
      return 0;
}
void led_init(void)
{     /* 设置寄存器，把 [15-0]位清零 */
      GPG3CON &= ~(0xffff << 0);
       /* 把led1,led2 GPJ2CON引脚设置为输出 */
      GPG3CON |= (0x1111 << 0);
}
/* 点亮所有 led灯*/
void led_on(void)
{     /* 把相应的数据寄存器 GPJ2DAT位设置为高电平 */
      GPG3DAT |= 0xf<< 0;
       return 0;
}
```

```
/* 关闭所有 led灯*/
void led_off(void)
{      /* 把相应的数据寄存器  GPJ2DAT位设置为低电平  */
      GPG3DAT &= ~(0xf<< 0);
      return 0;
}
/* 实现流水灯  */
void led_flow_water(void)
{      /* 把相应的数据寄存器  GPJ2DAT位设置为高电平  */
      unsigned int i = 0;
      while(1)
      {      GPG3DAT |= (1 << i);
            i++;
               if(4 == i)
            {i = 0;}
            mydelay(500000);
                  led_off();
      }
}
/* 自己实现延时处理  */
void mydelay(unsigned int n)
{      int i;
       for(i = 0; i < n; i++);
}
```

# 第 6 章 结束语

嵌入式产品开发这个行业虽然由来已久， 我们生活中有很多产品也是嵌入式产品，但是可能由于这个行业的准入门槛比较高， 身边从事这方面工作的比较少，以至于对于嵌入式产品开发这一个概念却不是很了解， 不知道什么叫嵌入式产品开发，不知道如何来开发。 通过这一次的实际项目设计之后， 对这个行业有了一定的理解，在开发过程中受益匪浅、体会颇深。

嵌入式开发对个人的软硬件均有一定的的要求， 而硬件部分恰恰是一个比较枯燥无味的东西，平时涉及的也比较少， 此次实验虽然对硬件知识的要求比较低，但是又要通过软件来控制， 所以在学习过程中就更加的困难。 对于控制硬件的软件部分虽然简单，但是由于涉及到对硬件内存的操作，使得难度加大。在 C-S开发过程中，服务端跟客户端是基于 Socket网络编程进行通讯，因为在平时编程过程中疏于使用， 所以在软件开发的过程中也遇到了不少的困难。 正是因为这些原因使得，使得我们在开发智能家居系统的时候配到了不少的麻烦。 一开始的一无所知，在指导老师的耐心讲解以及伴随着自己在实际操作中的好奇， 让我对嵌入式产品开发有了一定的了解。

在本次实验中， 虽然遇到了不少的困难， 但是更多的是学到了知识； 学会了如何进行嵌入式产品的开发； 学到了如何去查找问题， 思考问题产生的原因， 解决问题；团队之间如何合作使得效率的最大化。 也为以后的工作积累了相关的经验。