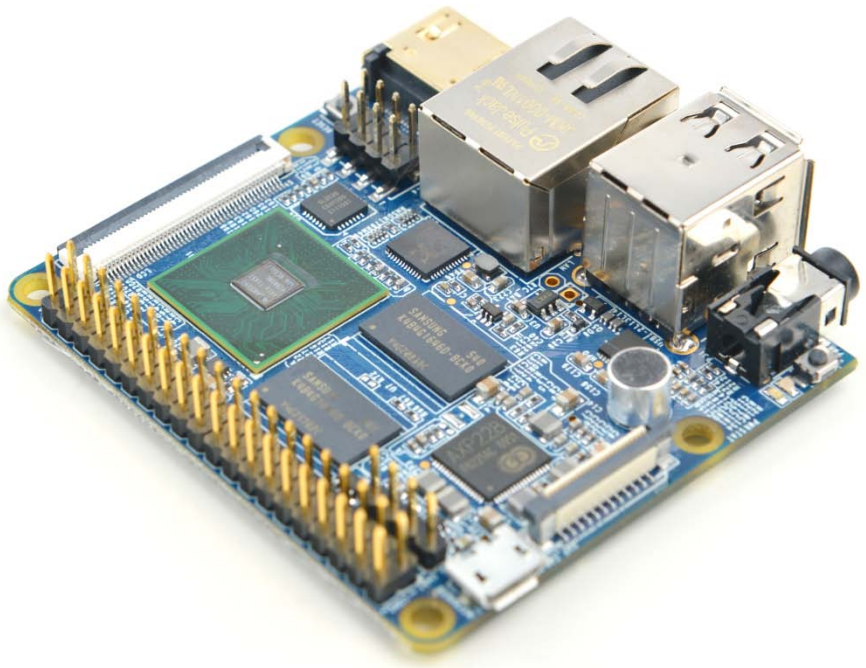
NanoPi M2 创客秘籍–03

版本日期：　2016-06-13



内容简介-

本创客秘籍主要收录“电子发烧友官网”的精华帖子编制而成，专门为创客、嵌入式爱好者而备。文中以 NanoPi M2 作为核心创客平台，讲解如何建立开发环境，如何使用 NanoPi M2 作为真机调试程序 ，如何在程序中访问硬件设备，包括 I2C, GPIO, SPI, 串口等硬件接口，如何编写 Python 程序，如何编译系统等教程。友善之臂特意介绍了很多极具挑战性和实用性的案例，你可以从这里领略到智能家居、物联网的风采，做到真正为你的项目开发助一臂之力。该创客秘籍同时适用于友善推出的 4418 系列的所有开发板。

本创客秘笈由广州友善之臂计算机科技有限公司 (以下简称“友善之臂”)创建和维护，友善之臂目前并不对其中的内容提供任何解释和解答服务，用户可以在相应论坛中交流和你所遇到的问题和疑问。本手册以首页日期为版本标志，并且本创客秘籍日后将会不断跟新添加新内容，请留意友善官方的最新动态。

本创客秘笈内容主要来自 “电子发烧友官网“，并且每篇文章都注明了作者网名和原文链接。我们欢迎各位玩家复制传播本手册，但不得擅自摘抄部分或全部内容用作商业用途，违者必究，友善之臂保留本手册的解释和修改权。

什么是 NanoPi M2?

NanoPi M2（以下简称 M2）是友善之臂团队面向创客、嵌入式爱好者，电子艺术家、发烧友等群体推出的一款完全开源的掌上创客神器，它的大小只有树莓派的大约 2/3，可运 行 Ubuntu Core、Debian、Android5.1、Ubuntu Mate 等操作系统。

相比树莓派 2B，NanoPi M2 采用了三星 A9 高性能处理器 S5P4418，集成千兆以太网，并采用了 AXP228 电源管理芯片，可支持软件开关机和睡眠唤醒等功能，M2 采用 microUSB 供电。 尽管体积很小，M2 依然保持了相当丰富的接口，诸如 HDMI，千兆网口，3.5mm 的音频输出，4 个 USB 口，板载麦克风等，并且兼容树莓派 GPIO 口，另外还支持 I2C 电容触摸 LCD 显示屏，摄像头接口，独立的调试串口等。

电子发烧友论坛 NanoPi2 论坛链接： http://bbs.elecfans.com/zhuti_831_1.html

目录

内容简介-.....2
技巧一：在 NanoPi 2 上开发 Qt 应用.....4
技巧二：Nanopi2 上配置 PostgreSQL 数据库.....9
技巧三：S5PV4418 串口通讯原理及裸机程序测试（一）.....15
技巧四：S5PV4418 串口通讯原理及裸机程序测试（二）.....22
技巧五：我的"潘多拉魔盒" 5——简易遥控器开发.....32
技巧六：Nanopi M2 之基于 QT 图形界面编程操作 MySQL 数据库.....50
技巧七：NanoPi M2 debian 中 NS-2 的安装和使用.....60
技巧八：NanoPi M2 + CC2530 的菜鸟开发.....62
技巧九：NanoPI M2 GPIO 篇——基于 Python3.....67
技巧十：智能车多功能遥控器开发项目结题报告.....73
技巧十一： 搭建 WordPress 或者 Emlog 个人网站（基于 APAche2+PHP5+mysql）.....93
技巧十二：Modbus-RTU 主站开发.....97
技巧十三：Modbus-TCP 主站开发（Modbus 协议二）.....107
技巧十四：基于 gsoap 的 Webservice 开发.....113
技巧十五：基于 gsaop 的 Webservice 远程控制 LED 灯.....122
技巧十六：OpenCV3.1 搭建.....139
技巧十七：OpenCV3.1 例程编译测试.....147
技巧十八：POLL 机制.....150
技巧十九：ad 转换，I2c.....152

技巧一：在 NanoPi 2 上开发 Qt 应用

开发板：NanoPi 2, Matrix - Compact Kit
系统：Ubuntu Core with Qt-Embedded

1. 环境配置

(1)制作带 Ubuntu Core with Qt-Embedded 的 SD 卡

到 http://wiki.friendlyarm.com/nanopi2/download/ 下载相关固件，将固件解压缩，用 dd 命令或者 win32image 工具制作 SD 卡即可。

PC 上安装 Qt-Embedded

下载名为 target-ate-4.8.6-to-hostpc.tgz 的压缩包，复制到根目录，进行解压：

```
cd /
tar xvzf  ~/target-ate-4.8.6-to-hostpc.tgz
```

解压完成后，执行 qmake 验证一下安装：

```
/usr/local/Trolltech/QtEmbedded-4.8.6-arm/bin/qmake -v
```

应得到如下信息：

```
QMake version 2.01a
Using Qt version 4.8.6 in /usr/local/Trolltech/QtEmbedded-4.8.6-arm/lib
```

PC 上安装交叉编译器(arm-linux-gcc 4.9.3)

首先下载并解压编译器:

```
git clone https://github.com/friendlyarm/prebuilts.git
sudo mkdir -p /opt/FriendlyARM/toolchain
sudo tar xf prebuilts/gcc-x64/arm-cortexa9-linux-gnueabihf-4.9.3.tar.xz -C /opt/FriendlyARM/toolchain/
```

然后将编译器的路径加入到 PATH 中，用 vi 编辑 vi ~/.bashrc，在末尾加入以下内容：

```
export PATH=/opt/FriendlyARM/toolchain/4.9.3/bin:$PATH
export GCC_COLORS=auto
```

执行一下~/.bashrc 脚本让设置立即在当前 shell 窗口中生效，注意"."后面有个空格：

```
. ~/.bashrc
```

这个编译器是 64 位的，不能在 32 位的 Linux 系统上运行，安装完成后，快速的验证是否安装成功：

```
arm-linux-gcc -v
```

当出现以下信息时，说明安装成功：

```
Thread model: posix
gcc version 4.9.3 (ctng-1.21.0-229g-FA)
```

2. 运行示例 Qt 程序 Hello world

在 PC 上(linux 系统)建立个 helloqt 目录并新建一个 main.cpp 的源文件:

```
cd ~
mkdir helloqt
vi main.cpp
```

main.cpp 的代码如下:

```
#include <QApplication>
#include <QPushButton>
int main(int argc, char **argv)
{
        QApplication app (argc, argv);
        QPushButton button ("Hello world !");
        button.show();
        return app.exec();
}
```

退出 vi 后，执行以下命令编译:

```
cd ~/helloqt
/usr/local/Trolltech/QtEmbedded-4.8.6-arm/bin/qmake -project
/usr/local/Trolltech/QtEmbedded-4.8.6-arm/bin/qmake
make
```

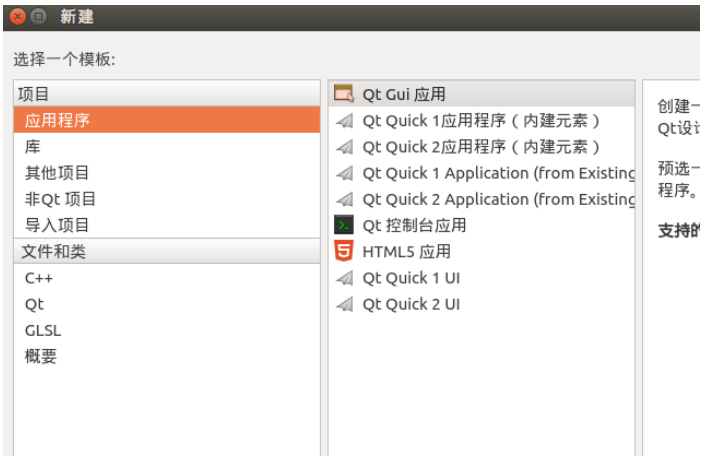
编译成功后，会得到了 helloqt 的二进制文件，将它上传到开发板后，在串口终端用以下命令运行:

```
. setqt4env
helloqt -qws&
```

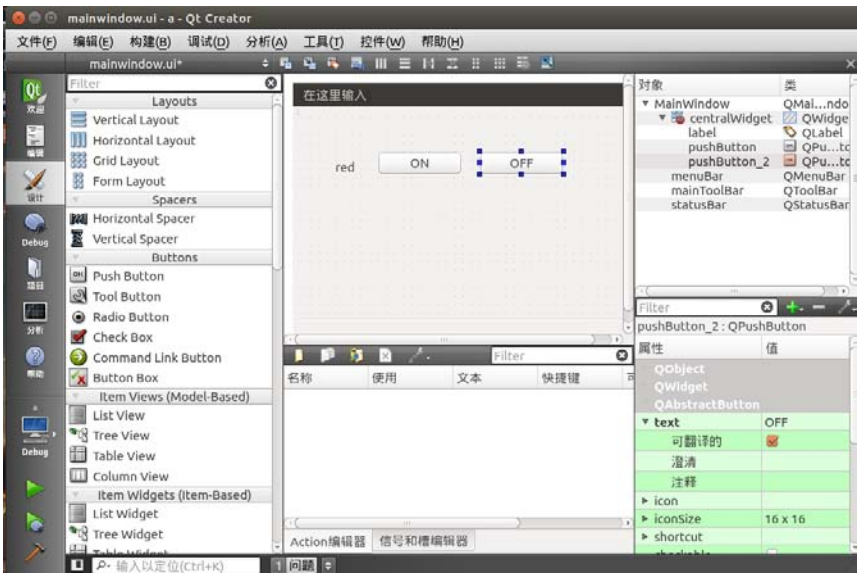
3. 编写并运行控制 LED 的 Qt 程序

（1）用 Qt Creator 编写程序

在 PC 上安装 Qt Creator，新建一个 Qt Gui 应用

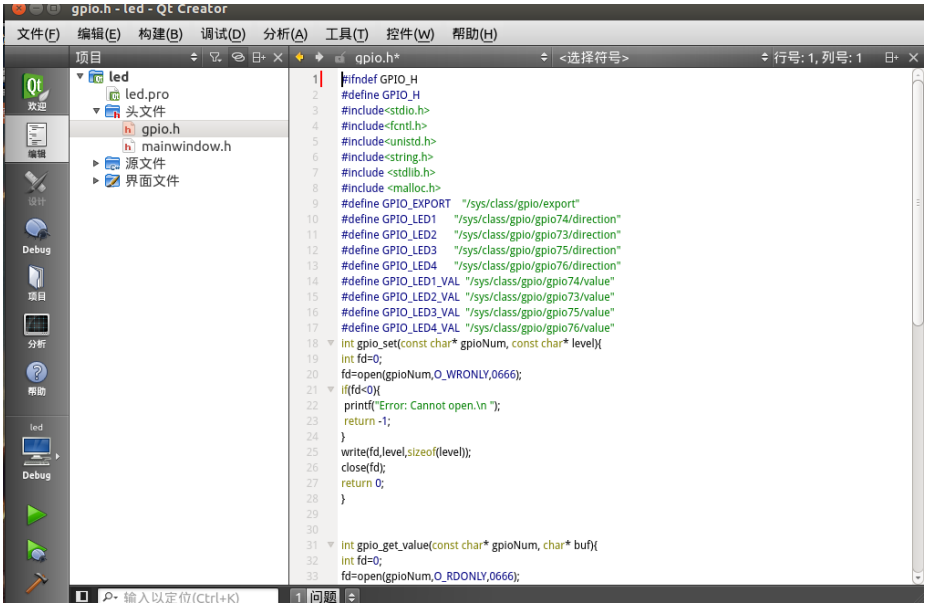


（2）切换到设计界面，向窗口上添加 Label 和 Push Button 控件



最后效果如下：

（3）添加 gpio.h 头文件，并编写代码



主要包括宏定义以及 set, get 和 init 的三个函数：

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<string.h>
#include <stdlib.h>
#include <malloc.h>
#define GPIO_EXPORT    "/sys/class/gpio/export"
#define GPIO_LED1      "/sys/class/gpio/gpio74/direction"
#define GPIO_LED2      "/sys/class/gpio/gpio73/direction"
#define GPIO_LED3      "/sys/class/gpio/gpio75/direction"
#define GPIO_LED4      "/sys/class/gpio/gpio76/direction"
#define GPIO_LED1_VAL  "/sys/class/gpio/gpio74/value"
#define GPIO_LED2_VAL  "/sys/class/gpio/gpio73/value"
#define GPIO_LED3_VAL  "/sys/class/gpio/gpio75/value"
#define GPIO_LED4_VAL  "/sys/class/gpio/gpio76/value"

int gpio_set(const char* gpioNum, const char* level){
int fd=0;
fd=open(gpioNum,O_WRONLY,0666);
if(fd<0){
 printf("Error: Cannot open.\n ");
 return -1;
}
write(fd,level,sizeof(level));
close(fd);
return 0;
}
```

```
int gpio_get_value(const char* gpioNum, char* buf){
int fd=0;
fd=open(gpioNum,O_RDONLY,0666);
if(fd<0){
 printf("Error: Cannot open.\n ");
 return -1;
}
read(fd,buf,sizeof("1"));
close(fd);
printf("%s: %s\n",gpioNum,buf);
return 0;
}

int  init(){
int fdl=0;
fdl=open(GPIO_EXPORT,O_WRONLY,0666);
if(fdl<0){
 printf("Error: Cannot open export.\n");
 return -1;
}
write(fdl,"74",sizeof("74"));
write(fdl,"73",sizeof("73"));
write(fdl,"75",sizeof("75"));
write(fdl,"76",sizeof("76"));
close(fdl);
gpio_set(GPIO_LED1,"out");
gpio_set(GPIO_LED2,"out");
gpio_set(GPIO_LED3,"out");
gpio_set(GPIO_LED4,"out");
}
```

（4）编写初始界面事件和按钮事件

在 mainwindow.cpp 里的 MainWindow 的构造函数中添加 init()函数，并未 Quit 按钮添加退出事件：

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    init();
    QObject::connect( ui->pushButton_2, SIGNAL(clicked()),this,SLOT(close() ));
}
```

在界面设计中，右键选择转到槽，编写按钮事件：

```
void MainWindow::on_pushButton_1_clicked()
{
 gpio_set(GPIO_LED1_VAL,"1");
}

void MainWindow::on_pushButton_3_clicked()
{
    gpio_set(GPIO_LED1_VAL,"0");
}

void MainWindow::on_pushButton_4_clicked()
{
    gpio_set(GPIO_LED2_VAL,"1");
}

void MainWindow::on_pushButton_5_clicked()
{
    gpio_set(GPIO_LED2_VAL,"0");
}
```

（5）构建项目后，得到 led.pro, gpio.h, mainwindow.h, mainwindow.cpp, main.cpp, mainwindow.ui 在命令行 cd 到项目所在目录，执行以下命令：

```
/usr/local/Trolltech/QtEmbedded-4.8.6-arm/bin/qmake
make
```

得到二进制文件后，传入 NanoPi 2 运行。

技巧二：Nanopi2 上配置 PostgreSQL 数据库

来自：jinyi7016　http://bbs.elecfans.com/jishu_545024_1_1.html

一、

安装世界上最先进的开源数据库 PostgreSQL 9.4 和 phpPgAdmin

```
# apt-get install postgresql postgresql-contrib
```

访问 PostgreSQL 命令窗口

默认的数据库名称和数据库用户名称都是 “postgres”。切换到 postgres 用户进行 postgresql

sudo -u postgres psql postgres

```
root@FriendlyARM:/# sudo -u postgres psql postgres
psql (9.4.4)
Type "help" for help.

postgres=#
```

二、

设置 “postgres” 用户密码为 123456

postgres=# \password postgres Enter new password: Enter itagain:

```
postgres=# \password postgres
Enter new password:
Enter it again:
```

安装 PostgreSQLAdminpack 扩展

postgres=# CREATE EXTENSION adminpack;

退出，回到终端

postgres=#\q

新建用户，用户名为 nanopi2 密码设为 fa

```
root@FriendlyARM:/# sudo -u postgres createuser -D -A -P nanopi2
Enter password for new role:
Enter it again:
root@FriendlyARM:/#
```

新建数据库，名为 fadb

```
root@FriendlyARM:/# sudo -u postgres createdb -O nanopi2 fadb
root@FriendlyARM:/#
```

要删除数据库与用户时

首先切换到 postgres 用户。

删除一个用户

drop user <user-name>

删除一个数据库

drop database <database-name>

三、

配置 PostgreSQL-MD5 验证

vim /etc/postgresql/9.4/main/pg_hba.conf

```
84 # Database administrative login by Unix domain socket
85 local   all             postgres                                peer
86
87 # TYPE  DATABASE        USER            ADDRESS                 METHOD
88
89 # "local" is for Unix domain socket connections only
90 local   all             all                                     md5
91 # IPv4 local connections:
92 host    all             all             127.0.0.1/32            md5
93 host    all             all             192.168.2.139/24
94 # IPv6 local connections:
95 host    all             all             ::1/128                 md5
96 # Allow replication connections from localhost, by a user with the
97 # replication privilege.
98 #local   replication     postgres                                peer
99 #host    replication     postgres        127.0.0.1/32            md5
```

192.168.2.139 是本机的地址。

重启 postgresql 服务

```
root@FriendlyARM:/# /etc/init.d/postgresql restart
Restarting postgresql (via systemctl): postgresql.service.
root@FriendlyARM:/#
```

四、

配置 PostgreSQL TCP/IP 配置

vim /etc/postgresql/9.4/main/postgresql.conf 将 #listen_addresses = 'localhost' 改为 listen_addresses = '*'

监听所有用户

重启 postgresql 服务

```
root@FriendlyARM:/# systemctl restart postgresql
root@FriendlyARM:/#
```

五、

用 phpPgAdmin 管理 PostgreSQL

安装# apt-get install phppgadmin

时间会比较长，安装完成后

```
Creating config file /etc/php5/mods-available/pgsql.ini with new version
php5_invoke: Enable module pgsql for apache2 SAPI
php5_invoke: Enable module pgsql for cli SAPI

Creating config file /etc/php5/mods-available/pdo_pgsql.ini with new versi
php5_invoke: Enable module pdo_pgsql for apache2 SAPI
php5_invoke: Enable module pdo_pgsql for cli SAPI
Setting up php5-readline (5.6.7+dfsg-1) ...

Creating config file /etc/php5/mods-available/readline.ini with new versio
php5_invoke: Enable module readline for apache2 SAPI
php5_invoke: Enable module readline for cli SAPI
Setting up postgresql-doc-9.4 (9.4.4-0+deb8u1) ...
Setting up postgresql-doc (9.4+165) ...
Setting up phppgadmin (5.1-1.1) ...
apache2_invoke: Enable configuration phppgadmin
Processing triggers for libc-bin (2.19-18+deb8u1) ...
Processing triggers for systemd (215-17+deb8u2) ...
Processing triggers for libapache2-mod-php5 (5.6.7+dfsg-1) ...
root@FriendlyARM:/#
```

vim /etc/apache2/conf-available/phppgadmin.conf

找到 Requirelocal 的一行在这行前面添加# 注释掉它

添加下面的一行：

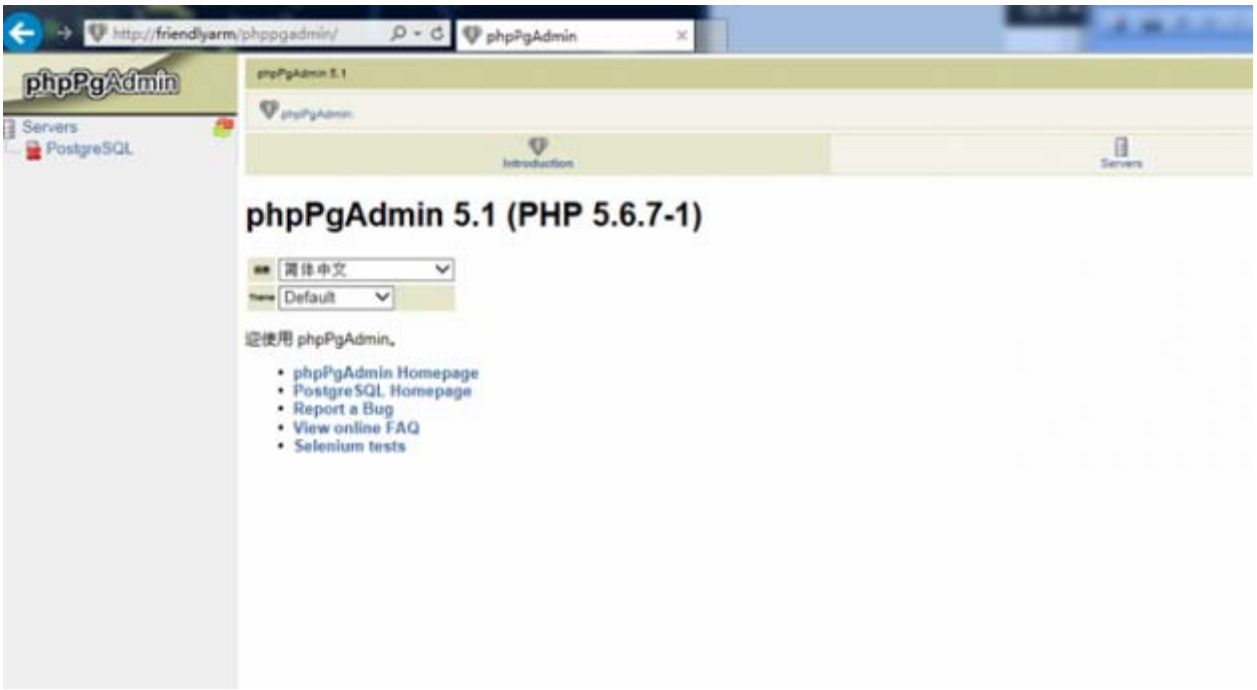
allow fromall

保存并退出文件。

然后重启 apache 服务。

systemctl restart apache2

在本地系统的 web 浏览器用 http://localhost/phppgadmin 访问 phppgadmin



六、
配置 phpPgAdmin

vim /etc/phppgadmin/config.inc.php 更改列表如下：

conf['servers'][0]['host'] = ''; ↵	conf['servers'][0]['host'] = 'FriendlyARM'; ↵
conf['extra_login_security'] = true; ↵	conf['extra_login_security'] = false; ↵
conf['owned_only'] = false; ↵	conf['owned_only'] = true; ↵

保存并关闭文件。重启 postgresql 服务和 Apache 服务。
service postgresql restart# service apache2 restart
现在打开你的浏览器输入地址 http://FriendlyARM/phppgadmin
用你之前创建的用户登录。我之前已经创建了一个名为 “nanppi2” 的用户，密码是 “fa”

http://friendlyarm/phppgadmin/
phpPgAdmin
phpPgAdmin
Servers
PostgreSQL
phpPgAdmin 5.1
phpPgAdmin:
Login to PostgreSQL
用名
nanopi2
密码
登录

技巧三：S5PV4418 串口通讯原理及裸机程序测试（一）

来自：chengtao http://bbs.elecfans.com/jishu_545705_1_1.html

测试好了裸机程序可以运行之后，前进的步伐不能停止，我们要继续努力。
接下来该干什么呢？
当然是让我们内置的外设控制器动起来啊！
这样就会催生出无穷无尽的应用，想想就就觉得美好的生活在前方等待着我们。
此贴针对刚入门的新人，高手请无视。

一、 两种通讯方式的比较

众所周知，带有处理能力的芯片，比如带 cpu，soc 等设备和其他带有同样的芯片的设备进行交互时，使用的两种通讯方式，一种是并行通讯，另外一种是串行通讯。
关于这两种通讯方式的比较，请看下面示意图：

串行通讯
一条信息的各位数据被逐位按顺序传送的通讯方式称为串行通讯。
串行通讯的特点是：
数据位传送，传按位顺序进行，最少只需一根传输线即可完成，成本低但送速度慢。串行通讯的距离可以从几米到几千米。
根据信息的传送方向，串行通讯可以进一步分为单工、半双工和全双工三种。
见下图

信息只能单向传送为单工；
信息能双向传送但不能同时双向传送称为半双工；
信息能够同时双向传送则称为全双工。
串行通讯又分为异步通讯和同步通讯两种方式。
在单片机中，主要使用异步通讯方式。串行通讯中，两个设备之间通过一对信号线进行通讯，其中一根为信号线，另外一根为信号地线，信号电流通过信号线到达目标设备，再经过信号地线返回，构成一个信号回路。
有人要问了为何不让信号电流从电源地线返回，干嘛多出个地线呢？
这是因为公共地线上存在各种杂乱的电流，会把我们需要传输的有用信号淹没，很难辨识，从而造成通讯误差。因此所有的信号线都使用信号地线而不是电源地线，以避免干扰。
这一对信号线每次只传送 1bit（比特）的信号，比如 1Byte（字节）的信号需要 8 次才能发完。传输的信号可以是数据、指令或者控制信号，这取决于采用的是何种通讯协议以及传输状态。串行信号本身也可以带有时钟信息，并且可以通过算法校正时钟。因此不需要额外的时钟信号进行控制。
并行通讯
并行通讯中，基本原理与串行通讯没有区别。
只不过使用了成倍的信号线路，从而一次可以传送更多 bit 的信号。

并行通讯通常可以一次传送 8bit、16bit、32bit 甚至更高的位数，相应地就需要 8 根、16 根、32 根信号线，同时需要加入更多的信号地线。比如传统的 PATA 线路有 40 根线，其中有 16 根信号线和 7 根信号地线，其他为各种控制线，一次可以传送 2Byte 的数据。并行通讯中，数据信号中无法携带时钟信息，为了保证各对信号线上的信号时序一致，并行设备需要严格同步时钟信号，或者采用额外的时钟信号线。

通过串行通讯与并行通讯的对比，可以看出：

串行通讯很简单，但是相对速度低；

并行通讯比较复杂，但是相对速度高。

更重要的是，串行线路仅使用一对信号线，线路成本低并且抗干扰能力强，因此可以用在长距离通讯上；而并行线路使用多对信号线（还不包括额外的控制线路），线路成本高并且抗干扰能力差，因此对通讯距离有非常严格的限制。

至于超高速通讯和长距离通讯方面，由于线路成本比接口成本要重要得多，因此一直都是串行通讯的应用领域。

除了比并行通讯速度慢以外，串行通讯还有有一些缺点：在计算机内部，从 cpu 到通讯接口间的数据大部分是并行方式传送的，当采用串行方式与外界通讯时必须经过串/并转换处理。从而增加了串行通讯的时间开销，不过瑕不掩瑜，目前的集成逻辑处理电路技术已经解决了串/并转换速度慢的问题。由于现在集成逻辑处理电路的成本也比 IO 连接单元更便宜，因此串行通讯再次显示出它的优势。如果说集成电路技术一度帮助并行通讯流行起来，那么现在的高度集成水平则帮助串行通讯重返主流应用领域。

所以说串口通讯目前仍然是单片机时代的 cpu 和外界外围设备打交道的重要手段，也是单片机扩展功能必不可少的手段，说他是单片机的神经系统也不为过。所以现在它仍然是单片机攻城狮必须掌握的基本技能之一。

到了 4 核 arm 时代，串口通讯的重要性非但没有减少，反而越来越重要了。

为什么这么说呢？

个人感觉还是出于对嵌入式系统低成本和通讯方便性的需求就决定了串口通讯一定会长时间存在于我们的开发工作中。

好了，啰嗦了半天，开始步入正题，下面详细介绍一下串口通讯。

二、 串口通讯的两种方式：

串口通讯共有两种方式：同步通讯和异步通讯。

同步通信

同步通信是一种连续串行传送数据的通信方式，一次通信只传送一帧信息。这里的信息帧与异步通信中的字符帧不同，通常含有若干个数据字符。

它们均由同步字符、数据字符和校验字符（CRC）组成。其中同步字符位于帧开头，用于确认数据字符的开始。数据字符在同步字符之后，个数没有限制，由所需传输的数据块长度来决定；校验字符有 1 到 2 个，用于接收端对接收到的字符序列进行正确性的校验。同步通信的缺点是要求发送时钟和接收时钟保持严格的同步。

异步通信

异步通信中，在异步通信中有两个比较重要的指标：字符帧格式和波特率。数据通常以字符或者字节为单位组成字符帧传送。字符帧由发送端逐帧发送，通过传输线被接收设备逐帧接收。发送端和接收端可以由各自的时钟来控制数据的发送和接收，这两个时钟源彼此独立，互不同步。

接收端检测到传输线上发送过来的低电平逻辑"0"（即字符帧起始位）时，确定发送端已开始发送数据，每当接收端收到字符帧中的停止位时，就知道一帧字符已经发送完毕。

串行异步传输时的数据格式：

(1) 起始位：起始位必须是持续一个比特时间的逻辑“0”电平，标志传送一个字符的开始。

(2) 数据位：数据位为 5-8 位，它紧跟在起始位之后，是被传送字符的有效数据位。传送时先传送字符的低位，后传送字符的高位。数据位究竟是几位，可由硬件或软件来设定。

(3) 奇偶位：奇偶校验位仅占一位，用于进行奇校验或偶校验，也可以不设奇偶位。

(4) 停止位：停止位为 1 位、1.5 位或 2 位，可有软件设定。它一定是逻辑“1”电平，标志着传送一个字符的结束。

(5) 空闲位：空闲位表示线路处于空闲状态，此时线路上为逻辑“1”电平。空闲位可以没有，此时异步传送的效率为最高。

三、串口通讯的接口和协议

1、串行通讯的物理连接如下示意图：

2、RS-232-C 接口

最被人们熟悉的串行通信技术标准是 EIA－232、EIA-422 和 EIA－485，也就是以前所称的 RS-232、RS-422 和 RS-485。由于 EIA 提出的建议标准都是以“RS”作为前缀，所以在工业通信领域，仍然习惯将上述标准以 RS 作前缀称谓。RS-232-C 接口（又称 EIA RS-232-C）是目前最常用的一种串行通讯接口。它是在 1970 年由美国电子工业协会（EIA）联合贝尔系统、 调制解调器厂家及计算机终端生产厂家共同制定的用于串行通讯的标准。它的全名是“数据终端设备（DTE）和数据通讯设备（DCE）之间 串行二进制数据交换接口技术标准”该标准规定采用一个 25 个脚的 DB25 连接器，对连接器的每个引脚的信号内容加以规定，还对各种信 号的电平加以规定。

（1）接口的信号内容

实际上 RS-232-C 的 25 条引线中有许多是很少使用的，在计算机与终端通讯中一般只使用 3-9 条引线。RS-232-C 最常用的 9 条引线的信号内容见附表 1 所示。

（2）接口的电气特性

在 RS-232-C 中任何一条信号线的电压均为负逻辑关系。

即： 逻辑“1”，-5— -15V；

逻辑“0” +5— +15V 。噪声容限为 2V。

即 要求接收器能识别低至+3V 的信号作为逻辑“0”，高到-3V 的信号 作为逻辑“1”

(3) 接口的物理结构

RS-232-C 接口连接器一般使用型号为 DB-25 的 25 芯插头座,通常插头在 DCE 端,插座在 DTE 端. 一些设备与 PC 机连接的 RS-232-C 接口,因为不使用对方的传送控制信号,只需三条接口线,即“发送数据”、“接收数据”和“信号地”。所以采用 DB-9 的 9 芯插头座，传输线采用屏蔽双绞线。

(4) rs232<->ttl 的电平转换

一个值得注意的问题是，由于计算机使用的电平是-15v~+15v，而我们的单片机上使用的 ttl 电平是-5 V~+5v，所以两者进行通讯时必须进行电平转换，这就是我们在单片机上经常能看到一个 232 转换芯片的原因。

四、三星 s5p4418 的串口通讯功能：

这个 4 核心的 soc，三星赋予了它 5 个原生的串口。怎见得呢？

打开圣经，俺不信教，哪来的圣经啊！

哎！是 datasheet 啊！

翻到第 1 章。

先看 Block 图

6 个 channel？好牛 x 啊！
等等，好像有问题。
再往下看：

1.4.17 UART& ISO7816 Sim Card Interface

- 5-ch UART controller
- Programmable use of UART or IrDA SIR input/output.
- Separate 32 × 8 transmit and 3 2 ×12 receive First-In, First-Out (FIFO) memory buffers to reduce CPU interrupts.
- Programmable FIFO disabling for 1-byte depth.
- Programmable baud rate generator. This enables division of the reference clock by (1 × 16) to (65535 × 16) and generates an internal × 16 clock. The divisor can be a fractional number enabling you to use any clock with a frequency >3.6864 MHzas the reference clock.
- Standard asynchronous communication bits (start, stop and parity). These are added prior to transmission and removed on reception.
- Independent masking of transmit FIFO, receive FIFO, receive timeout, modem status, and error condition interrupts.
- Support for Direct Memory Access (DMA).
- False start bit detection.
- Line break generation and detection.
- Support of the modem control functions CTS, DCD, DSR, RTS, DTR, and RI.
- Programmable hardware flow control.
- Fully-programmable serial interface characteristics:
  - data can be 5, 6, 7, or 8 bits
  - even, odd, stick, or no-parity bit generation and detection
  - 1 or 2 stop bit generation
  - baud rate generation, dc up to UARTCLK/16
- IrDA SIR ENDEC block providing:
  - programmable use of IrDA SIR or UART input/output
  - support of IrDA SIR ENDEC functions for data rates up to 115200 bps half-duplex
  - support of normal 3/16 and low-power (1.41-2.23 μs) bit durations
  - programmable division of the UARTCLK reference clock to generate the appropriate bit duration for low-power IrDA mode.
- Identification registers that uniquely identify the UART. These can be used by an operating system to automatically configure itself.

SAMSUNG Confidential
only for internal use 2015.01.13

有图为证：

我去，明明是 5 个 uart channel 吗？这土财主干活也太粗心了吧！就知道骗俺们钱，手册都不好写写。继续验证一下，看看寄存器那里怎么说？

24.5.1.1 UARTDR

- Base Address: C00A_1000 (UART0)
- Base Address: C00A_0000 (UART1)
- Base Address: C00A_2000 (UART2)
- Base Address: C00A_3000 (UART3)
- Base Address: C006_D000 (UART4)
- Address = Base Address + 0x00, 0x00, 0x00, 0x00, 0x00, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:12]	–	Reserved	–
OE	[11]	R	Overrun error. This bit is set to 1 if data is received and the receive FIFO is already full. This is cleared to 0 once there is an empty space in the FIFO and a new character can be written to it.	1'b0
BE	[10]	R	Break error. This bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits). In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state), and the next valid start bit is received.	1'b0
PE	[9]	R	Parity error. When set to 1, it indicates that the parity of the received data character does not match the parity that the EPS and SPS bits in the Line Control Register, UARTLCR_H select. In FIFO mode, this error is associated with the character at the top of the FIFO.	1'b0
FE	[8]	R	Framing error. When set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1). In FIFO mode, this error is associated with the character at the top of the FIFO.	1'b0
DATA	[7:0]	RW	Receive (read) data character. Transmit (write) data character.	8'b0

SAMSUNG Confidential
cn / Industrial Solution

这下清清楚楚了，从 0-4，共五个串口通道！
好了，不纠结了。
再看看友善的攻城狮为我们引出了几个串口通道吧！

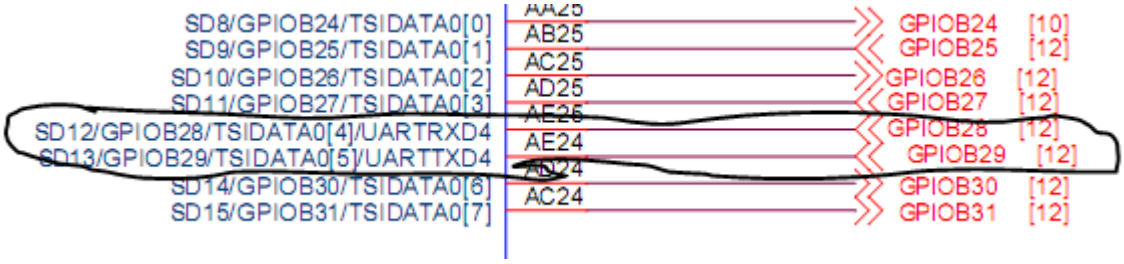
- GPIO1管脚定义

Pin#	Name	Pin#	Name
1	VDD_SYS_3.3V	2	VDD_5V
3	I2C0_SDA	4	VDD_5V
5	I2C0_SCL	6	DGND
7	GPIOB28	8	UART3_TXD
9	DGND	10	UART3_RXD
11	GPIOB29	12	GPIOB26
13	GPIOB30	14	DGND
15	GPIOB31	16	PWM2
17	VDD_SYS_3.3V	18	GPIOB27
19	SPI0_MOSI	20	DGND
21	SPI0_MISO	22	PWM0
23	SPI0_CLK	24	SPI0_CS
25	DGND	26	PWM1
27	I2C1_SDA	28	I2C1_SCL
29	GPIOC8	30	DGND
31	SPI2_CLK	32	GPIOC28
33	SPI2_CS	34	DGND
35	SPI2_MOSI	36	GPIOC7
37	SPI2_MISO	38	ALIVEGPIO2
39	DGND	40	ALIVEGPIO3

- Debug Port CON1 (UART0)

Pin#	Name
1	DGND
2	VDD_5V
3	TXD0
4	RXD0

明显能看出来的是 2 个，还有一个是隐式提供的，根据 pi2 原理图，能看到 Uart4 也被通过 GPIOB28 和 GPIOB29 引出了，

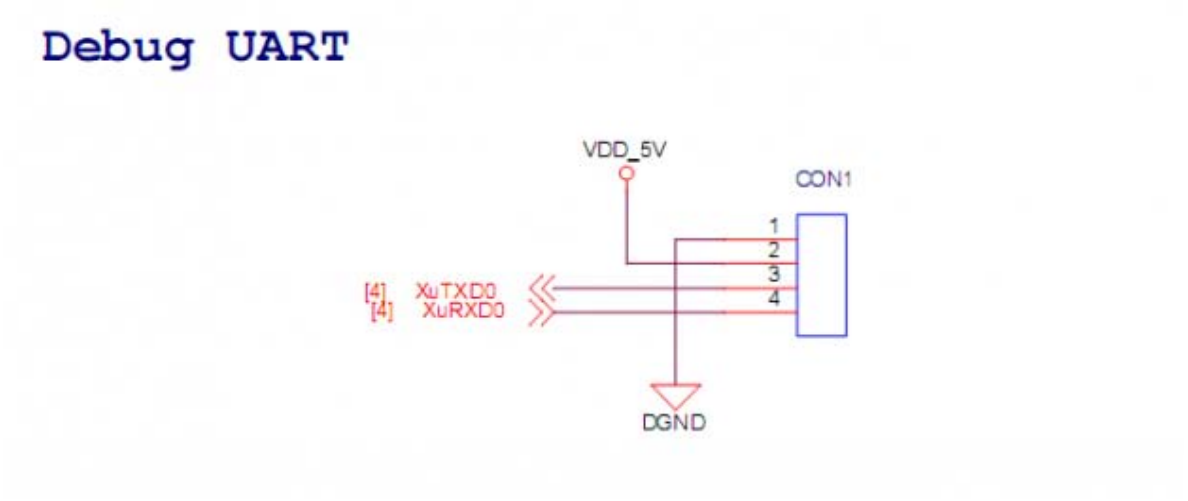


OK, 至此，我们知道一共引出了 3 个串口。

技巧四：S5PV4418 串口通讯原理及裸机程序测试（二）

来自：chengtao　http://bbs.elecfans.com/jishu_545707_1_1.html

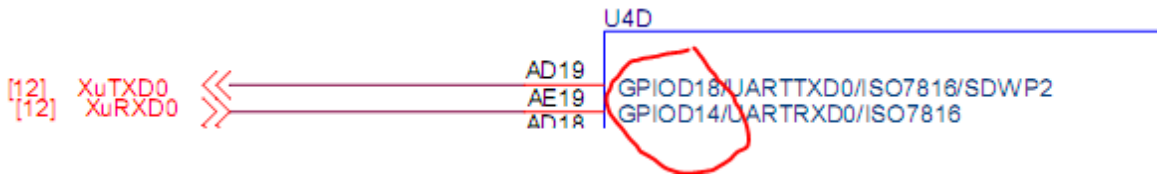
五、裸机串口程序测试
根据上面的分析，我们这次拿 uart0，来做测试，这个串口就是友善的攻城狮拿来做 debug 口的，提供了串口的小附件，正好方便了我们做实验。
Uart0 对应的是哪些引脚呢？
继续来看图



15.PNG (18.43 KB, 下载次数: 0)

下载附件　保存到相册

2016-1-30 11:18 上传



16.PNG (17.9 KB, 下载次数: 0)

下载附件　保存到相册

2016-1-30 11:18 上传

由此可知，uart0 分别对应 s5pv4418 的 GPIO14 和 GPIO18，先记住这两个引脚，一会编程时，还要用到。
根据以前的文章，我们还是先建立几个裸机测试程序，把它链接接到 2ndboot 后面 ，由 2ndboot 来加载它，并把控制权交给它来执行。
根据 s5pv4418 的 datasheet，在使用 uart 进行串口通讯前，我们需要做些准备工作：
众所周知，由于 ARM 处理器的好多引脚都是复用的，所以我们在使用它内部集成的控制器的时候，需要对这些控制器做初始化。当然 usart 也不例外。
这些初始化工作，主要就是配置引脚功能、时钟源及时钟源的频率、根据串口通讯频率，对时钟源进行分频，取出串口通讯需要的频率，设计串口通讯的数据帧格式。
下面程序参考了哀其不幸大侠的程序，在此表示感谢，没有前辈们的无私奉献，不知我们要走多少弯路才能打到这里。
主要的过程就是配置相关寄存器，
1.、设置引脚，UART0 的 RX、TX 对应引脚是 gpio14,gpio18，分别设置成输入模式、输出模式，且都设置成 UART 通信功能。
2、 选择时钟脉冲源并使能时钟，由 2ndboot 初始化过程的打印信息可以知道各个 Pll 时钟脉冲源的频率分别被初始化为
PLL0: 550000000 PLL1: 1200000000 PLL2: 800000000 PLL3:612000000 ， 这里选择 PLL0 即频率 550MHz，
3、 对 PLL0 分频和设置 UART 波特率，采用 55 分频因数，得到 10MHz，我们的 UART 实验采用 115200 的波特率，
根据公式：波特率分频因数=UART 时钟/(16*波特率)　　得到分频因数为 5.42534722。
分频因数由整数和分数部分组成（即 BRDI 和 BRDF），从结果可知 BRDI=5，BRDF 计算公式为
m=integer(BRDF*2n+0.5),n=32,
　　得到 m 为 27，写入寄存器中时 m 要减 1。
4、设置 UART0：奇偶校验禁止，1 位停止位，8 位数据，使能 FIFO、UART0 接收和发送功能。
主要的配置就是这些，当然其他的 cpu 初始化等等都是要做好的，说实话，初始化工作还是比较繁琐的。本来想自己从头写一个完整的串口通讯程序的，后来偶然看到了 x4418 团队的串口例程，就打消了这个念头。思量一下，暂时还达不到他们的水平，写出来有误导读者的嫌疑，不如就拿他们的做例子吧！。在这里向 x4418 的攻城狮致敬，正是因为前辈们的心辛勤劳动，才让我们的学习之路平坦无比。
X4418 的例程包括: main.c，main.h，
在主程序 main 中调用各个部分的初始化函数，进行整个系统的初始化工作
Main.c

```
#include <main.h>
extern int tester_serial_stdio(int argc, char * argv[]);
static void do_system_initial(void)
{
	malloc_init();
	s5p4418_clk_init();
	s5p4418_irq_init();
	s5p4418_gpiochip_init();
	s5p4418_pwm_init();
	s5p4418_serial_initial();
	s5p4418_tick_initial();
	s5p4418_tick_delay_initial();
```

```
        s5p4418_fb_initial();
        led_initial();
        beep_initial();
        key_initial();
}
int main(int argc, char * argv[])
{
        do_system_initial();
        tester_serial_stdio(argc, argv);
        return 0;
}
```

我们着重看看串口的初始化过程：

```
s5p4418_serial_initial();
```

它位于 s5p4418-serial.c 中，程序如下：

```
#include <s5p4418-serial.h>
static void s5p4418_serial_init(int ch)
{
        switch(ch)
        {
        case 0:
                clk_enable("GATE-UART0");
                gpio_set_cfg(S5P4418_GPIOD(14),0x1);
                gpio_set_cfg(S5P4418_GPIOD(18), 0x1);
                gpio_set_direction(S5P4418_GPIOD(14), GPIO_DIRECTION_INPUT);
                gpio_set_direction(S5P4418_GPIOD(18), GPIO_DIRECTION_OUTPUT);
                break;
        case 1:
                clk_enable("GATE-UART1");
                gpio_set_cfg(S5P4418_GPIOD(15), 0x1);
                gpio_set_cfg(S5P4418_GPIOD(19), 0x1);
                gpio_set_direction(S5P4418_GPIOD(15), GPIO_DIRECTION_INPUT);
                gpio_set_direction(S5P4418_GPIOD(19), GPIO_DIRECTION_OUTPUT);
                break;
        case 2:
                clk_enable("GATE-UART2");
                gpio_set_cfg(S5P4418_GPIOD(16), 0x1);
                gpio_set_cfg(S5P4418_GPIOD(20), 0x1);
                gpio_set_direction(S5P4418_GPIOD(16), GPIO_DIRECTION_INPUT);
                gpio_set_direction(S5P4418_GPIOD(20), GPIO_DIRECTION_OUTPUT);
                break;
        case 3:
                clk_enable("GATE-UART3");
```

```
                gpio_set_cfg(S5P4418_GPIOD(17), 0x1);
                gpio_set_cfg(S5P4418_GPIOD(21), 0x1);
                gpio_set_direction(S5P4418_GPIOD(17), GPIO_DIRECTION_INPUT);
                gpio_set_direction(S5P4418_GPIOD(21), GPIO_DIRECTION_OUTPUT);
                break;
        default:
                return;
        }
        s5p4418_serial_setup(ch, B115200, DATA_BITS_8, PARITY_NONE, STOP_BITS_1);
}
void s5p4418_serial_initial(void)
{
        s5p4418_serial_init(0);
        s5p4418_serial_init(1);
        s5p4418_serial_init(2);
        s5p4418_serial_init(3);
}
bool_t s5p4418_serial_setup(int ch, enum baud_rate_t baud, enum data_bits_t data, enum
parity_bits_t parity, enum stop_bits_t stop)
{
        u32_t base =S5P4418_UART0_BASE;
        u32_t ibaud, divider,fraction;
        u32_t temp, remainder;
        u8_t data_bit_reg,parity_reg, stop_bit_reg;
        u64_t uclk;
        switch(ch)
        {
        case 0:
                base = S5P4418_UART0_BASE;
                break;
        case 1:
                base = S5P4418_UART1_BASE;
                break;
        case 2:
                base = S5P4418_UART2_BASE;
                break;
        case 3:
                base = S5P4418_UART3_BASE;
                break;
        default:
                break;
        }
```

```
switch(baud)
{
case B50:
        ibaud = 50;
        break;
case B75:
        ibaud = 75;
        break;
case B110:
        ibaud = 110;
        break;
case B134:
        ibaud = 134;
        break;
case B200:
        ibaud = 200;
        break;
case B300:
        ibaud = 300;
        break;
case B600:
        ibaud = 600;
        break;
case B1200:
        ibaud = 1200;
        break;
case B1800:
        ibaud = 1800;
        break;
case B2400:
        ibaud = 2400;
        break;
case B4800:
        ibaud = 4800;
        break;
case B9600:
        ibaud = 9600;
        break;
case B19200:
        ibaud = 19200;
        break;
case B38400:
```

```
        ibaud = 38400;
        break;
case B57600:
        ibaud = 57600;
        break;
case B76800:
        ibaud = 76800;
        break;
case B115200:
        ibaud = 115200;
        break;
case B230400:
        ibaud = 230400;
        break;
case B380400:
        ibaud = 380400;
        break;
case B460800:
        ibaud = 460800;
        break;
case B921600:
        ibaud = 921600;
        break;
default:
        return FALSE;
}
switch(data)
{
case DATA_BITS_5:
        data_bit_reg = 0x0;
        break;
case DATA_BITS_6:
        data_bit_reg = 0x1;
        break;
case DATA_BITS_7:
        data_bit_reg = 0x2;
        break;
case DATA_BITS_8:
        data_bit_reg = 0x3;
        break;
default:
        return FALSE;
```

```
}
switch(parity)
{
case PARITY_NONE:
        parity_reg = 0x0;
        break;
case PARITY_EVEN:
        parity_reg = 0x2;
        break;
case PARITY_ODD:
        parity_reg = 0x1;
        break;
default:
        return FALSE;
}
switch(stop)
{
case STOP_BITS_1:
        stop_bit_reg = 0;           break;
case STOP_BITS_1_5:
        return -1;
case STOP_BITS_2:
        stop_bit_reg = 1;           break;
default:
        return -1;
}
switch(ch)
{
case 0:
        uclk = clk_get_rate("GATE-UART0");
        break;
case 1:
        uclk = clk_get_rate("GATE-UART1");
        break;
case 2:
        uclk = clk_get_rate("GATE-UART2");
        break;
case 3:
        uclk = clk_get_rate("GATE-UART3");
        break;
default:
        return FALSE;
```

```
        }
        /*
           * IBRD = UART_CLK / (16 * BAUD_RATE)
           * FBRD = ROUND((64 * MOD(UART_CLK, (16 * BAUD_RATE))) / (16 *BAUD_RATE))
           */
        temp = 16 * ibaud;
        divider = (u32_t)(uclk / temp);
        remainder = (u32_t)(uclk % temp);
        temp = (8 * remainder) / ibaud;
        fraction = (temp >> 1) + (temp & 1);
        write32(phys_to_virt(base + UART_IBRD), divider);
        write32(phys_to_virt(base + UART_FBRD), fraction);
        write32(phys_to_virt(base + UART_LCRH), (1 << 4) |(data_bit_reg<<5 | stop_bit_reg<<3 |
parity_reg<<1));
        return TRUE;
}
ssize_t s5p4418_serial_read(int ch, u8_t * buf, size_t count)
{
        u32_t base =S5P4418_UART0_BASE;
        ssize_t i;
        switch(ch)
        {
        case 0:
                base = S5P4418_UART0_BASE;
                break;
        case 1:
                base = S5P4418_UART1_BASE;
                break;
        case 2:
                base = S5P4418_UART2_BASE;
                break;
        case 3:
                base = S5P4418_UART3_BASE;
                break;
        default:
                break;
        }
        for(i = 0; i < count; i++)
        {
                if( !(read8(phys_to_virt(base + UART_FR)) & UART_FR_RXFE) )
                        buf = read8(phys_to_virt(base + UART_DATA));
                else
```

```
                        break;
        }
        return i;
}
ssize_t s5p4418_serial_write(int ch, u8_t * buf, size_t count)
{
        u32_t base =S5P4418_UART0_BASE;
        ssize_t i;
        switch(ch)
        {
        case 0:
                base = S5P4418_UART0_BASE;
                break;
        case 1:
                base = S5P4418_UART1_BASE;
                break;
        case 2:
                base = S5P4418_UART2_BASE;
                break;
        case 3:
                base = S5P4418_UART3_BASE;
                break;
        default:
                break;
        }
        for(i = 0; i < count; i++)
        {
                while( (read8(phys_to_virt(base + UART_FR)) & UART_FR_TXFF) );
                write8(phys_to_virt(base + UART_DATA), buf);
        }
        return i;
}
ssize_t s5p4418_serial_write_string(int ch, const char * buf)
{
        return s5p4418_serial_write(ch,(u8_t *)buf, strlen(buf));
}
```

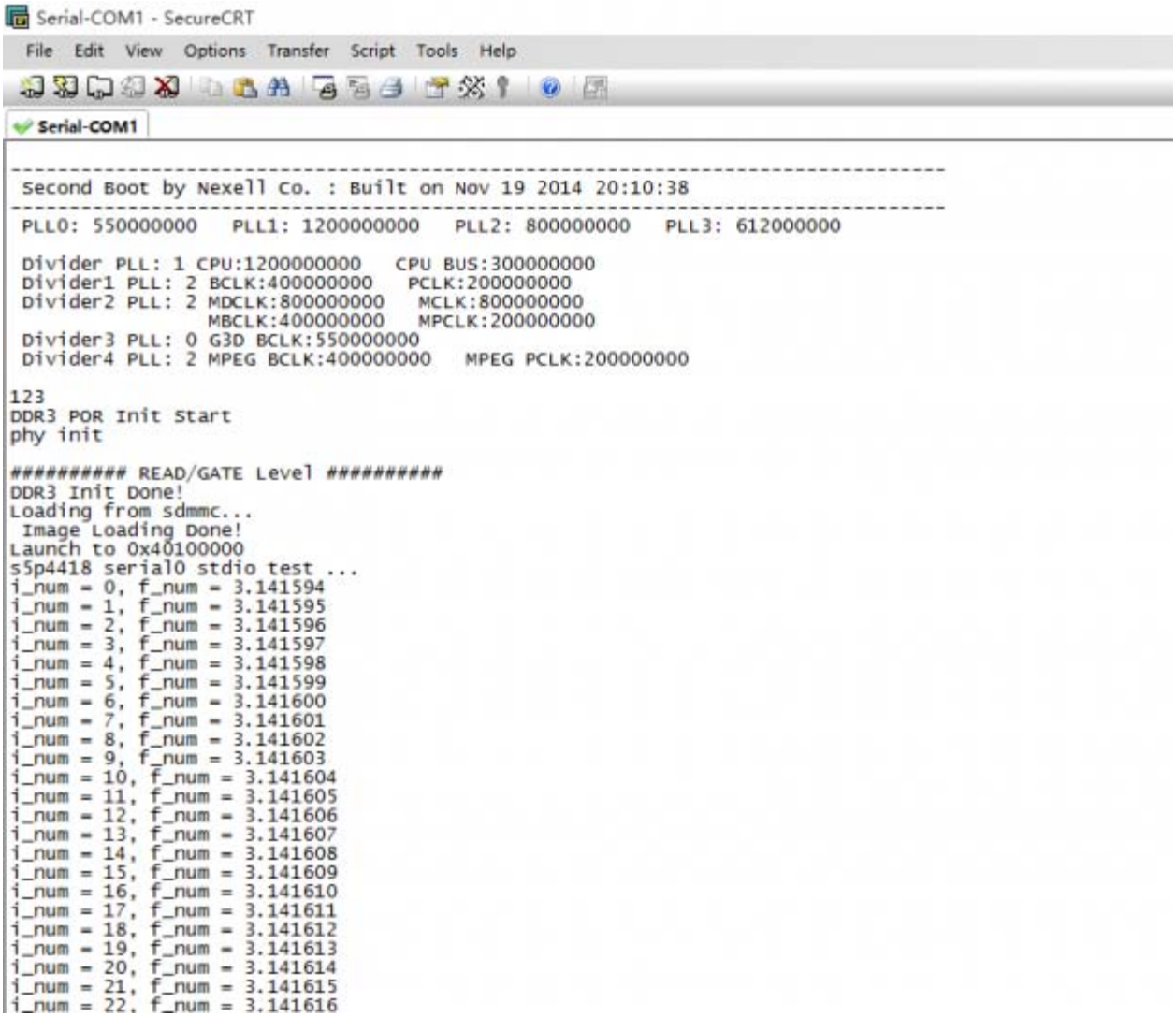
X4418 把 5 个串口都做了初始化，我们用的是第一个 uart0，注意看跟 uart0 有关的红颜色标记的部分。从串口寄存器配置，到串口波特率配置一应俱全。

在此给 x4418 的前辈点个赞，程序写的规规矩矩，很值得后来者学习。具体程序不一一解读了，自己看程序吧！

限于本文篇幅，又需要此测试程序的，请私信给我。

在 pc 端连接好交叉串口线，通讯参数：115200，8N1,无 。

下面是程序运行结果的串口显示信息：



17.PNG (57.59 KB, 下载次数: 0)

下载附件　保存到相册

2016-1-30 11:18 上传

至此，串口通讯篇就介绍完了。

技巧五：我的"潘多拉魔盒" 5——简易遥控器开发

来自网友：辉哥的歌 http://bbs.elecfans.com/jishu_702846_1_1.html

1. 引言

在第 4 讲“我的‘潘多拉魔盒’4 — samba 服务器搭建详细过程和 GPIO 开发初探”中采用了 Python 的 GPIO 库进行了流水灯的示例开发。但要完成我的项目“智能车多功能遥控器开发”，必须能够完成 GPIO 的输入输出操作，而流水灯只有输出操作，看了大多数烧友进行 NanoPi 的 GPIO 的开发示例也基本只有输出的例子，很少有涉及输入的开发的，而我要做遥控器的开发，遥控器上的按键是作为输入，必须得进行 GPIO 的输入开发。

开始使用 Python 进行输入开发时，由于现有 python 库对 GPIO 输入控制的底层驱动不健全，基本无法使用，而我的按键是最普通的按键和 NanoPi wiki 所提供示例的按键不一样。所以我决定放弃现有的不完整的 python 库，采用配置寄存器的方法在 Geany 环境中进行输入输出的开发。其原理主要是通过 mmap()函数将寄存器的地址通过映射到用户层，然后在用户层即可操作寄存器，控制方法参考三星 S5P4418 用户手册，这是一个需要耐心的过程。

2. 硬件接线说明

（1） 简易遥控器的硬件部分所需材料见表 1：

表 1 简易遥控器开发所需材料

序号	材料	个数	说明
1	长条面包板	1	作遥控手柄
2	普通按键	4	控制小车前后左右运动
3	LED 灯	4	显示控制结果
4	电阻	8	用于上拉或下拉
5	公母插线	10	用于连接 NanoPi GPIO 接口和面包板

（2） 按键及发光二极管接线如图 1 所示，控制的逻辑关系如表 2 所示。其中，按键主要连接 GPIO 的 C 组接口，LED 灯连接 GPIO 的 B 组接口，具体的接口号见图 1。

基本的控制关系是：

1） C11 的按键按下时 B30 灯亮代表着小车前进，松下按键时灯灭代表停止前进；

2） C13 的按键按下时 B27 灯亮代表着小车右转，松下按键时灯灭代表停止右转；

3） C7 的按键按下时 B29 灯亮代表着小车左转，松下按键时灯灭代表停止左转；

4） C8 的按键按下时 B31 灯亮代表着小车后退，松下按键时灯灭代表停止后退。

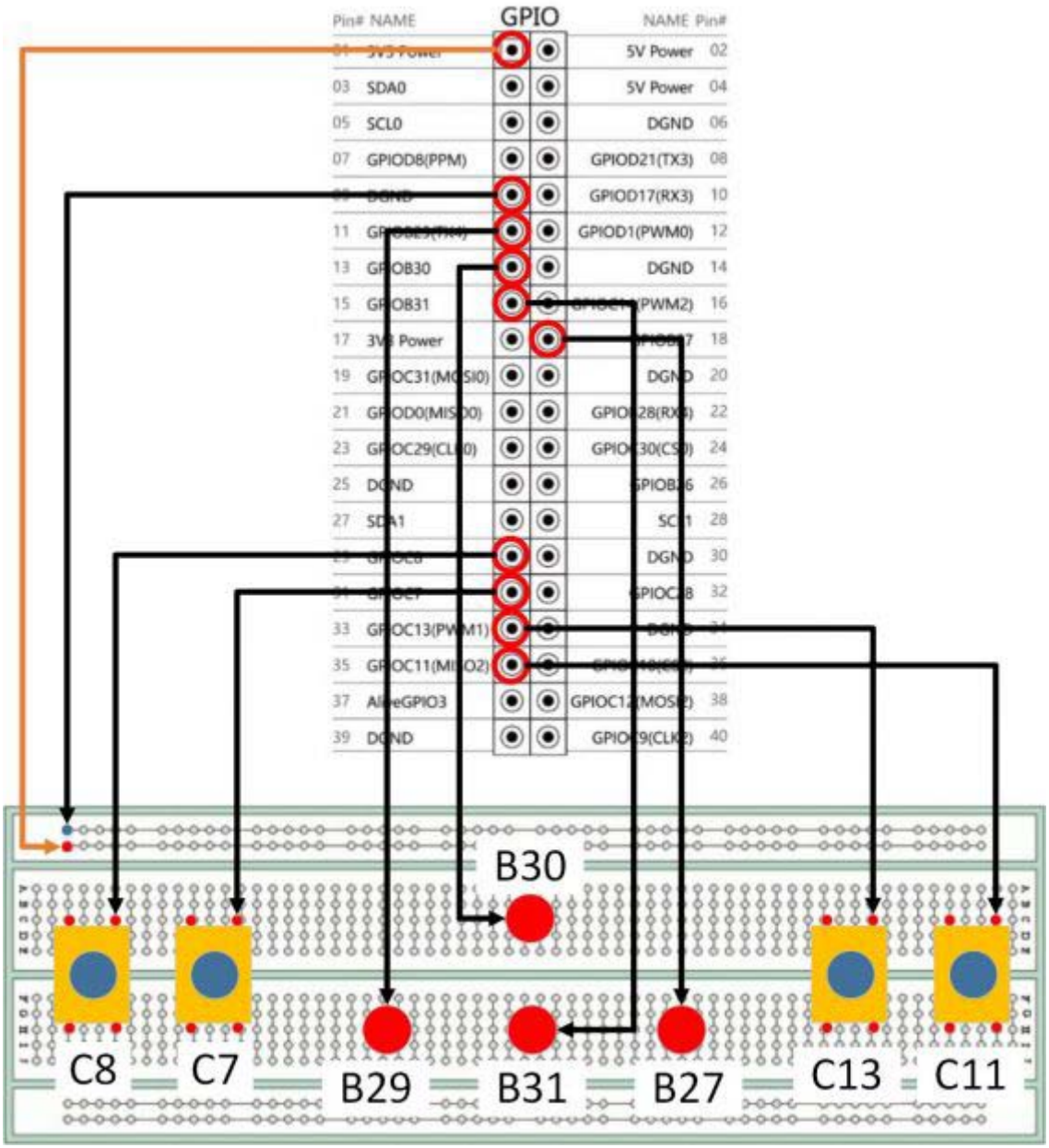


图 1 按键及发光二极管接线说明

表 2 按键控制的对应的 LED 灯代表小车的动作逻辑关系

按键	LED 灯	控制小车动作
GPIOC11	GPIOB30	前进
GPIOC13	GPIOB27	右转
GPIOC7	GPIOB29	左转
GPIOC8	GPIOB31	后退

（3）　　硬件布局和接线完成实物见图 2-3 所示。

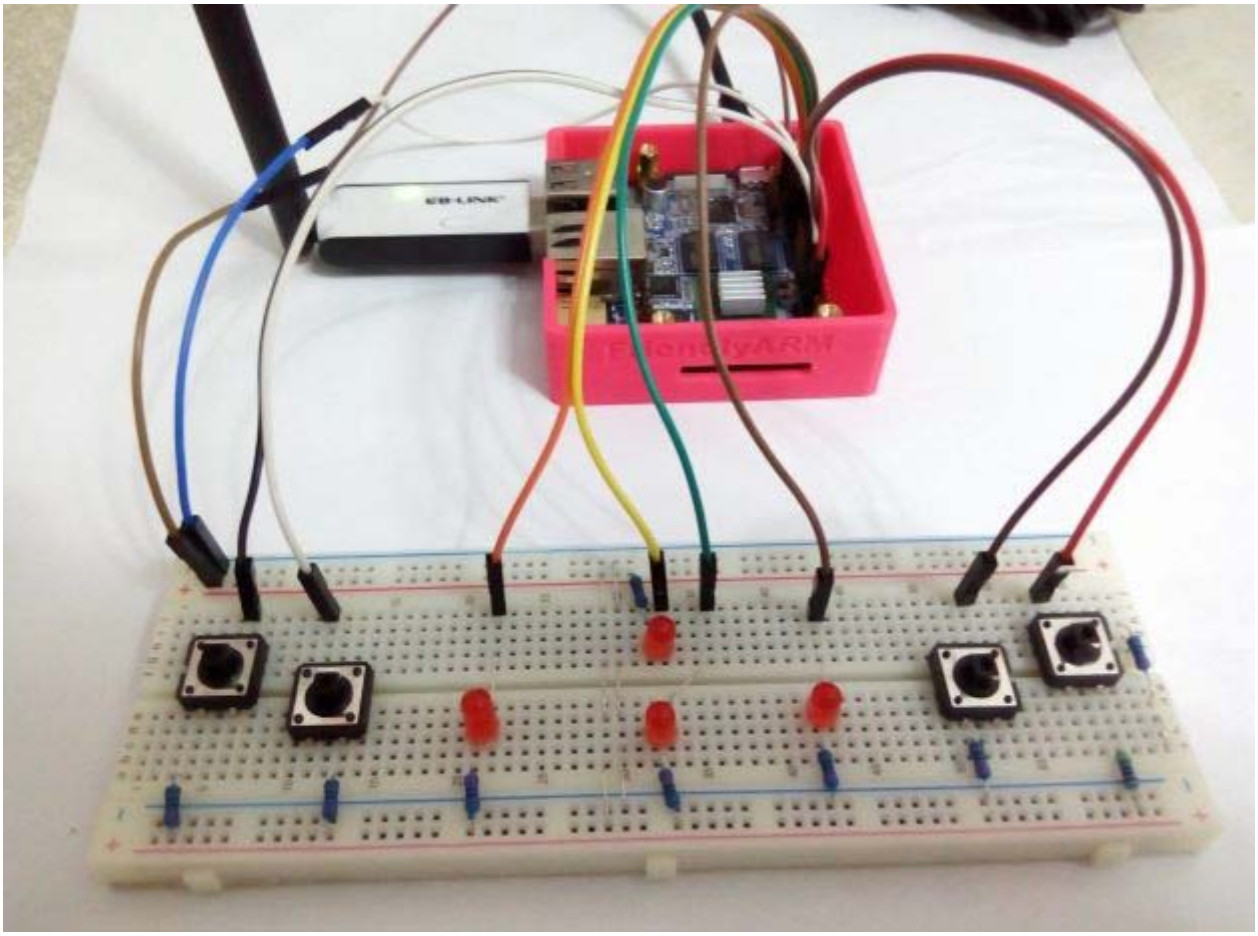


图 2 接线实物效果

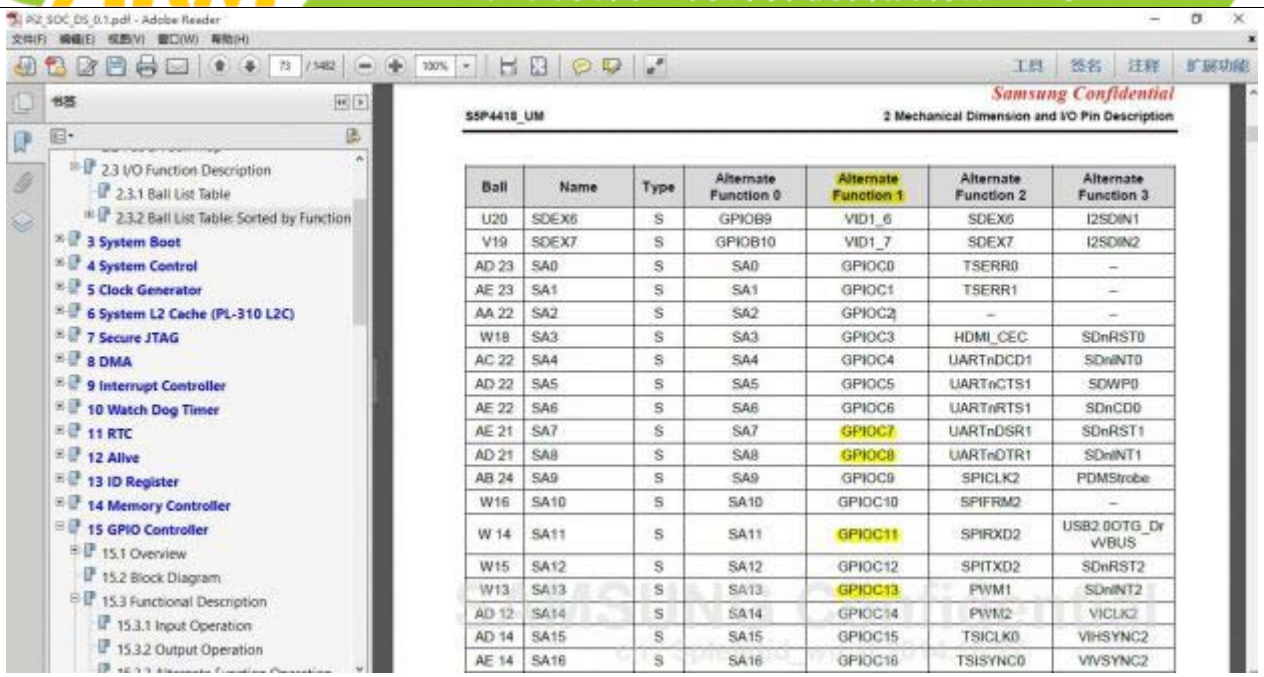
图 3 对应 GPIO 接口

3. 软件程序开发

本部分的开发主要参考三星 S5P4418 用户手册，很多同学可能刚看这个有点不知怎么去用这个手册，其实很简单，参考如下步骤就可以揭开它神秘的外纱。

1） 确定 GPIO 接口的复用功能函数

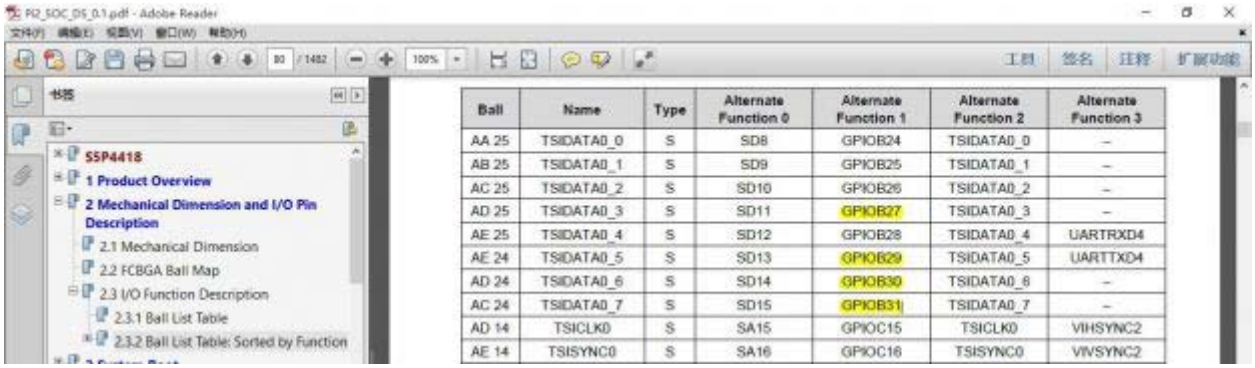
首先打开手册至 2.3.2Ball List Table: Sorted by Function 查看 GPIOC13, 11, 8, 7 复用功能函数均为 AlternateFunction 1. 意思是将 GPIOC13, 11, 8, 7 对应的引脚定义为 GPIO 的功能，而不是其他的功能（如 UART 功能，或 TX 功能，或 PWM 功能等）。



Ball	Name	Type	Alternate Function 0	Alternate Function 1	Alternate Function 2	Alternate Function 3
U20	SDEX6	S	GPIOB9	VID1_6	SDEX6	I2SDIN1
V19	SDEX7	S	GPIOB10	VID1_7	SDEX7	I2SDIN2
AD 23	SA0	S	SA0	GPIOC0	TSERR0	–
AE 23	SA1	S	SA1	GPIOC1	TSERR1	–
AA 22	SA2	S	SA2	GPIOC2	–	–
W18	SA3	S	SA3	GPIOC3	HDMI_CEC	SDnRST0
AC 22	SA4	S	SA4	GPIOC4	UARTnDCD1	SDnINT0
AD 22	SA5	S	SA5	GPIOC5	UARTnCTS1	SDWP0
AE 22	SA6	S	SA6	GPIOC6	UARTnRTS1	SDnCD0
AE 21	SA7	S	SA7	GPIOC7	UARTnDSR1	SDnRST1
AD 21	SA8	S	SA8	GPIOC8	UARTnDTR1	SDnINT1
AB 24	SA9	S	SA9	GPIOC9	SPICLK2	PDMStrobe
W16	SA10	S	SA10	GPIOC10	SPIFRM2	–
W 14	SA11	S	SA11	GPIOC11	SPIRXD2	USB2.0OTG_Dr VVBUS
W15	SA12	S	SA12	GPIOC12	SPITXD2	SDnRST2
W13	SA13	S	SA13	GPIOC13	PWM1	SDnINT2
AD 12	SA14	S	SA14	GPIOC14	PWM2	VICLK2
AD 14	SA15	S	SA15	GPIOC15	TSICLK0	VIHSYNC2
AE 14	SA16	S	SA16	GPIOC16	TSISYNC0	VIVSYNC2

图 4　　GPIOC13，11，8，7 的复用功能函数为 AlternateFunction 1

查看 GPIOB31，30，29，27 的复用功能函数也均为 AlternateFunction 1.如图 5 所示：



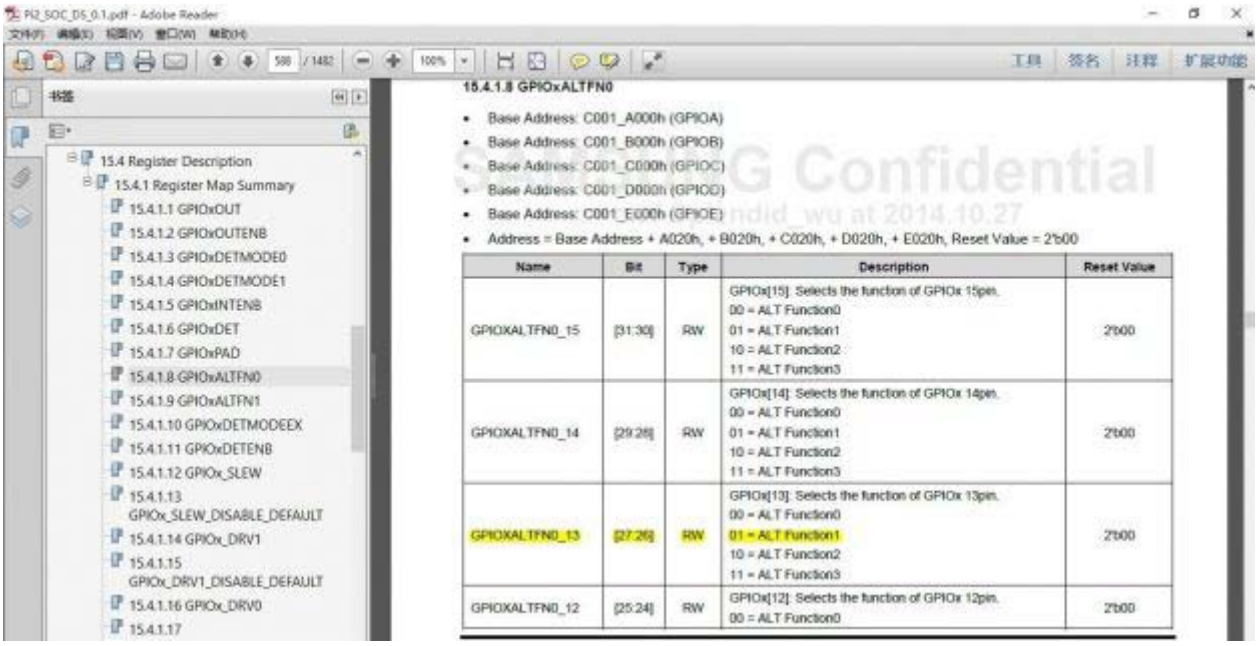
Ball	Name	Type	Alternate Function 0	Alternate Function 1	Alternate Function 2	Alternate Function 3
AA 25	TSIDATA0_0	S	SD8	GPIOB24	TSIDATA0_0	–
AB 25	TSIDATA0_1	S	SD9	GPIOB25	TSIDATA0_1	–
AC 25	TSIDATA0_2	S	SD10	GPIOB26	TSIDATA0_2	–
AD 25	TSIDATA0_3	S	SD11	GPIOB27	TSIDATA0_3	–
AE 25	TSIDATA0_4	S	SD12	GPIOB28	TSIDATA0_4	UARTRXD4
AE 24	TSIDATA0_5	S	SD13	GPIOB29	TSIDATA0_5	UARTTXD4
AD 24	TSIDATA0_6	S	SD14	GPIOB30	TSIDATA0_6	–
AC 24	TSIDATA0_7	S	SD15	GPIOB31	TSIDATA0_7	–
AD 14	TSICLK0	S	SA15	GPIOC15	TSICLK0	VIHSYNC2
AE 14	TSISYNC0	S	SA16	GPIOC16	TSISYNC0	VIVSYNC2

图 5　　GPIOB31，30，29，27 的复用功能函数也均为 AlternateFunction 1

2）　　确定 GPIO 引脚号与复用功能函数 ALTFN 寄存器的位号对应关系

打开手册至 **15.4.1.8GPIOxALTFN0** 可查得 GPIOx13 引脚号对应 GPIOXALTFN0 寄存器的第 27 和第 26 位（Bit[27:26]，以第 0 位开始），如过这两位设置为[0 1]则为 GPIOx13 引脚选择 ALT Function1 的功能，即 GPIO 功能，也就是将该引脚定义为 GPIO 引脚。

同理将 GPIOC11，8,7 对应 GPIOXALTFN0 寄存器的位置分别设置 01 即可，如图 6-7 所示；将和 GPIOB31，30，29，27 对应 GPIOXALTFN1 寄存器的位置分别设置 01 即可，如图 8 所示。

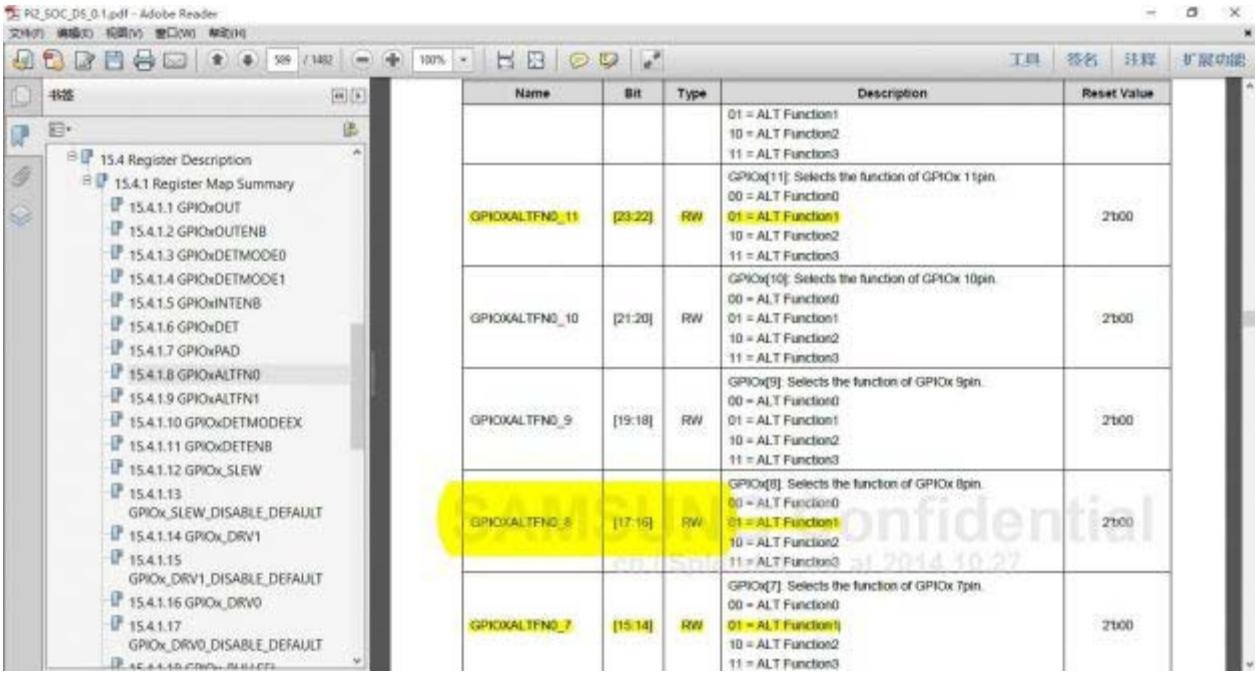


15.4.1.8 GPIOxALTFN0

- Base Address: C001_A000h (GPIOA)
- Base Address: C001_B000h (GPIOB)
- Base Address: C001_C000h (GPIOC)
- Base Address: C001_D000h (GPIOD)
- Base Address: C001_E000h (GPIOE)
- Address = Base Address + A020h, + B020h, + C020h, + D020h, + E020h, Reset Value = 2'b00

Name	Bit	Type	Description	Reset Value
GPIOXALTFN0_15	[31:30]	RW	GPIOx[15]: Selects the function of GPIOx 15pin. 00 = ALT Function0 01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	2'b00
GPIOXALTFN0_14	[29:28]	RW	GPIOx[14]: Selects the function of GPIOx 14pin. 00 = ALT Function0 01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	2'b00
GPIOXALTFN0_13	[27:26]	RW	GPIOx[13]: Selects the function of GPIOx 13pin. 00 = ALT Function0 01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	2'b00
GPIOXALTFN0_12	[25:24]	RW	GPIOx[12]: Selects the function of GPIOx 12pin. 00 = ALT Function0	2'b00

图 6　　将 GPIOXALTFN0 寄存器的第 27 和第 26 位分别设置为 0 和 1



Name	Bit	Type	Description	Reset Value
			01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	
GPIOXALTFN0_11	[23:22]	RW	GPIOx[11]: Selects the function of GPIOx 11pin. 00 = ALT Function0 01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	2'b00
GPIOXALTFN0_10	[21:20]	RW	GPIOx[10]: Selects the function of GPIOx 10pin. 00 = ALT Function0 01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	2'b00
GPIOXALTFN0_9	[19:18]	RW	GPIOx[9]: Selects the function of GPIOx 9pin. 00 = ALT Function0 01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	2'b00
GPIOXALTFN0_8	[17:16]	RW	GPIOx[8]: Selects the function of GPIOx 8pin. 00 = ALT Function0 01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	2'b00
GPIOXALTFN0_7	[15:14]	RW	GPIOx[7]: Selects the function of GPIOx 7pin. 00 = ALT Function0 01 = ALT Function1 10 = ALT Function2 11 = ALT Function3	2'b00

图 7 将 GPIOXALTFN0 寄存器的第[23:22]，[17:16]，[15:14]位分别设置为 0 和 1

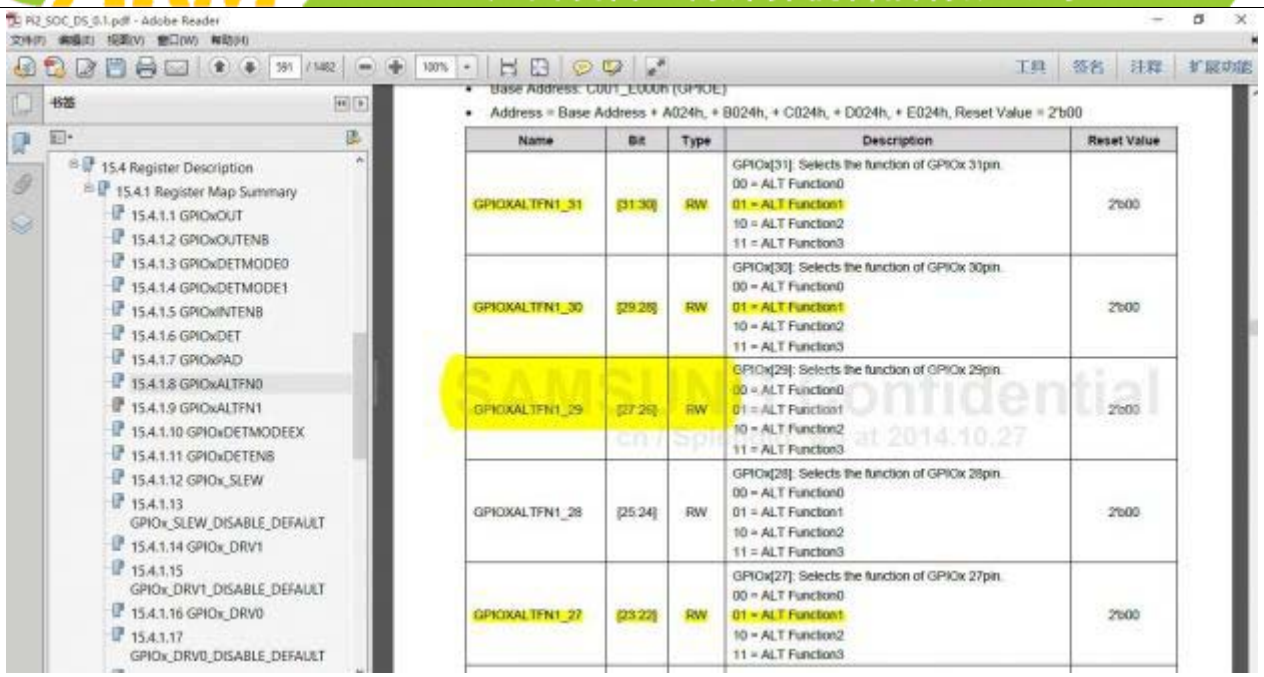


图 8　　GPIOB31，30，29，27 对应 GPIOXALTFN1 寄存器

3）　　设置输入输出使能寄存器 GPIOXOUTENB

打开手册至 **15.4.1.2GPIOxOUTENB** 如图 9 所示，可见 **GPIOxOUTENB** 寄存器有 32 位，每一位代表相应的 GPIOxPin 引脚号 Pin 的值，如第 31 位为 GPIOx31 引脚，若将其设置为 1 则该引脚定义为输出，若设置为 0 则定义为输入。

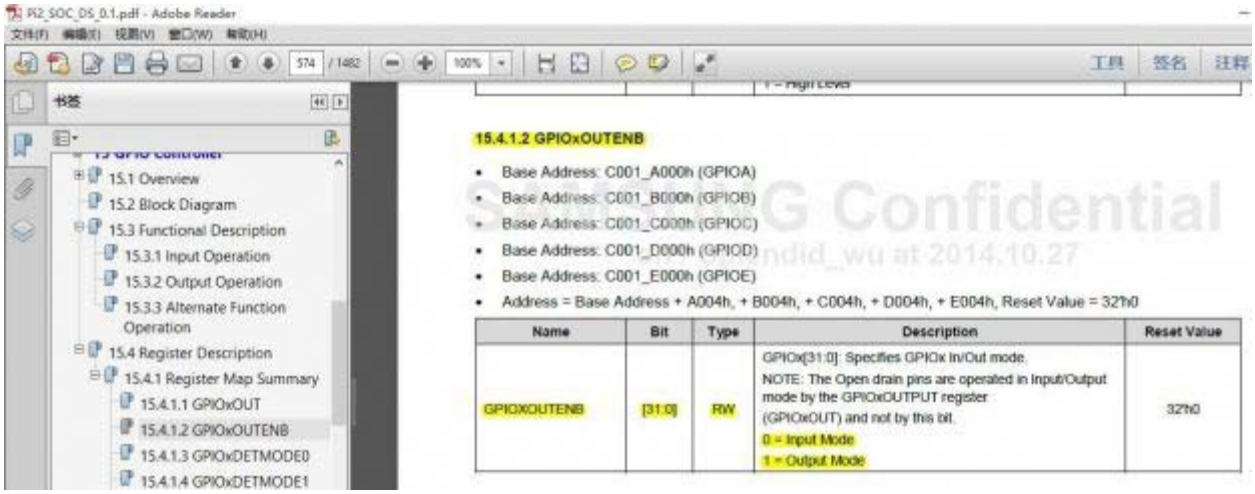
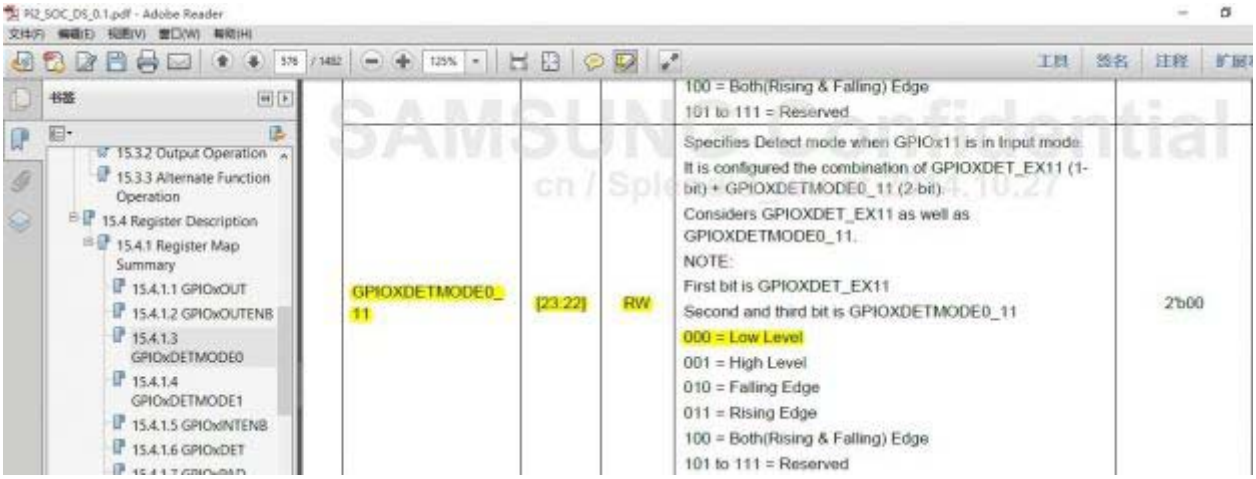


图 9 输出/入使能

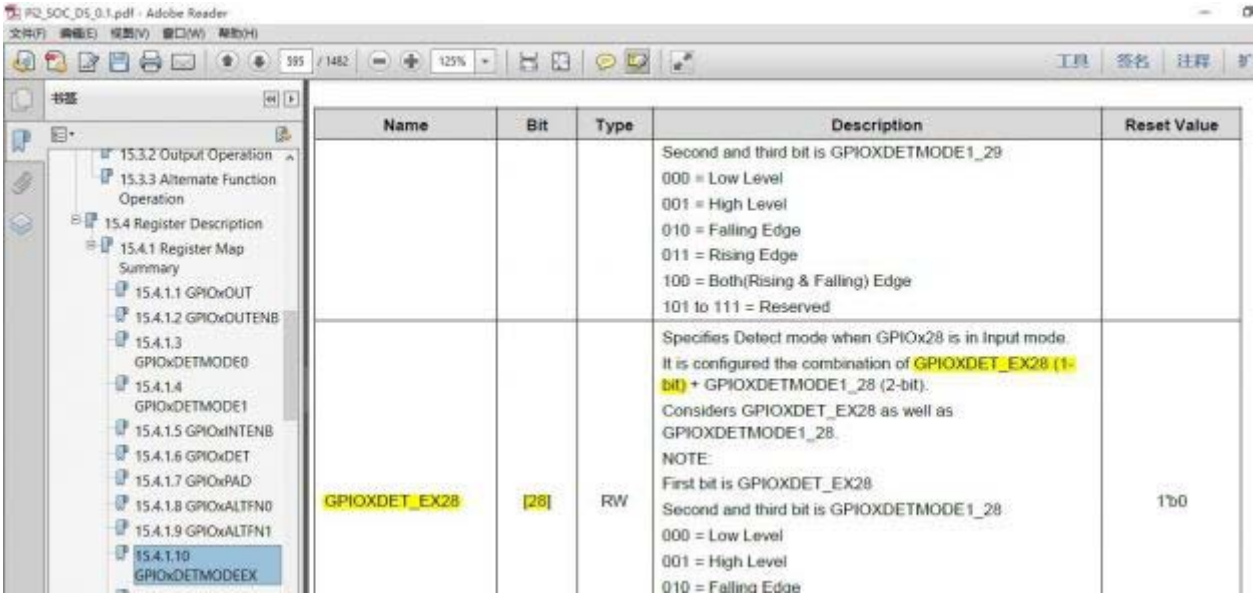
4）　　设置输入检测模式寄存器 **GPIOxDETMODE0 和 GPIOxDETMODEEX**

打开手册至 **15.4.1.3GPIOxDETMODE0** 将检所使用的 GPIO 的 C 组引脚 C13,11,8,7 测模式设置为高电平检测，也就是将对应的 GPIOxDETMODE0 寄存器的对应位设置为 01，而将 GPIOxDETMODEEX 的对应位全设置为 0. 如图 10-11 所示。



			100 = Both(Rising & Falling) Edge 101 to 111 = Reserved	
GPIOXDETMODE0_11	[23:22]	RW	Specifies Detect mode when GPIOx11 is in Input mode. It is configured the combination of GPIOXDET_EX11 (1-bit) + GPIOXDETMODE0_11 (2-bit). Considers GPIOXDET_EX11 as well as GPIOXDETMODE0_11. NOTE: First bit is GPIOXDET_EX11 Second and third bit is GPIOXDETMODE0_11 000 = Low Level 001 = High Level 010 = Falling Edge 011 = Rising Edge 100 = Both(Rising & Falling) Edge 101 to 111 = Reserved	2'b00

图 10 输入电平检测模式寄存器设置



Name	Bit	Type	Description	Reset Value
			Second and third bit is GPIOXDETMODE1_29 000 = Low Level 001 = High Level 010 = Falling Edge 011 = Rising Edge 100 = Both(Rising & Falling) Edge 101 to 111 = Reserved	
GPIOXDET_EX28	[28]	RW	Specifies Detect mode when GPIOx28 is in Input mode. It is configured the combination of GPIOXDET_EX28 (1-bit) + GPIOXDETMODE1_28 (2-bit). Considers GPIOXDET_EX28 as well as GPIOXDETMODE1_28. NOTE: First bit is GPIOXDET_EX28 Second and third bit is GPIOXDETMODE1_28 000 = Low Level 001 = High Level 010 = Falling Edge	1'b0

图 11 检测模式扩展寄存器设置

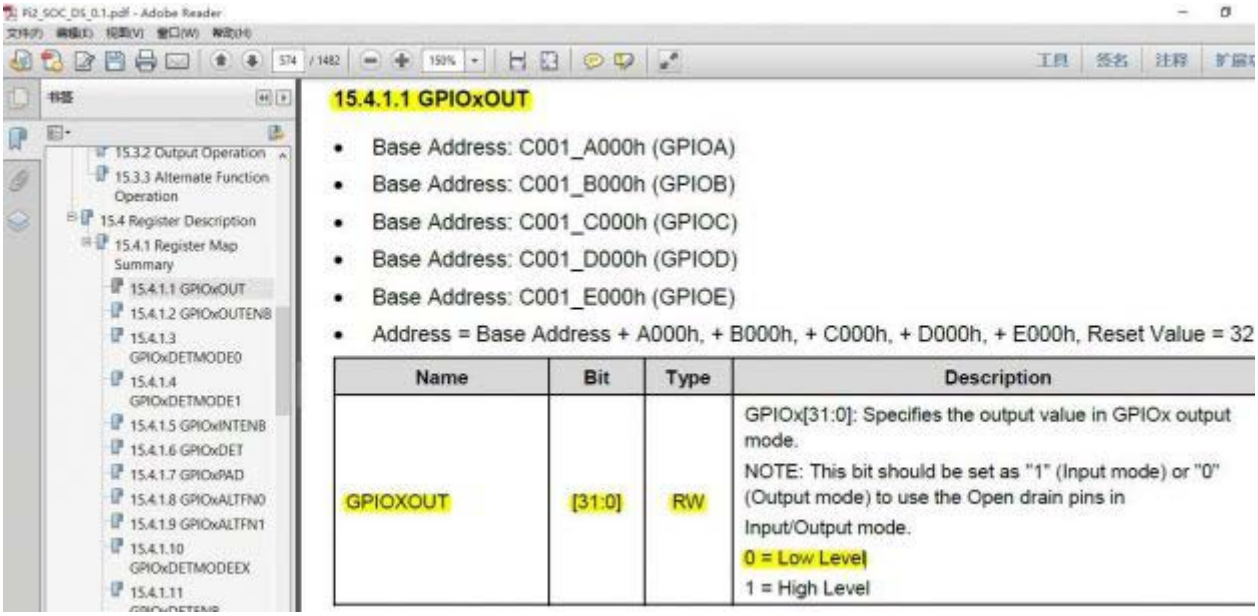
具体程序细节如下：

```
//C13=[27:26],C11=[23:22],C8=[17:16],C7=[15:14], 检测模式设置为 01
GPIOCDETMODE0 &=~((0x2<<26)|(0x2<<22)|(0x2<<16)|(0x2<<14));
// C13=bit[13],C11=[11],C8=[8],C7=[7] = 0 检测模式拓展为设置为 0
GPIOCDETMODEEX &=~((0x1<<13)|(0x1<<11)|(0x1<<8)|(0x1<<7));
```

5）　初始化输出寄存器

参见手册 **15.4.1.1 GPIOxOUT** 将 GPIO 的 B 组引脚 B31,30,29,27 初始输出为 0。如图 12 所示，具体程序细节如下：

```
GPIOBOUT &= ~((0x1<<31)|(0x1<<30)|(0x1<<29)|(0x1<<27));
```



15.4.1.1 GPIOxOUT

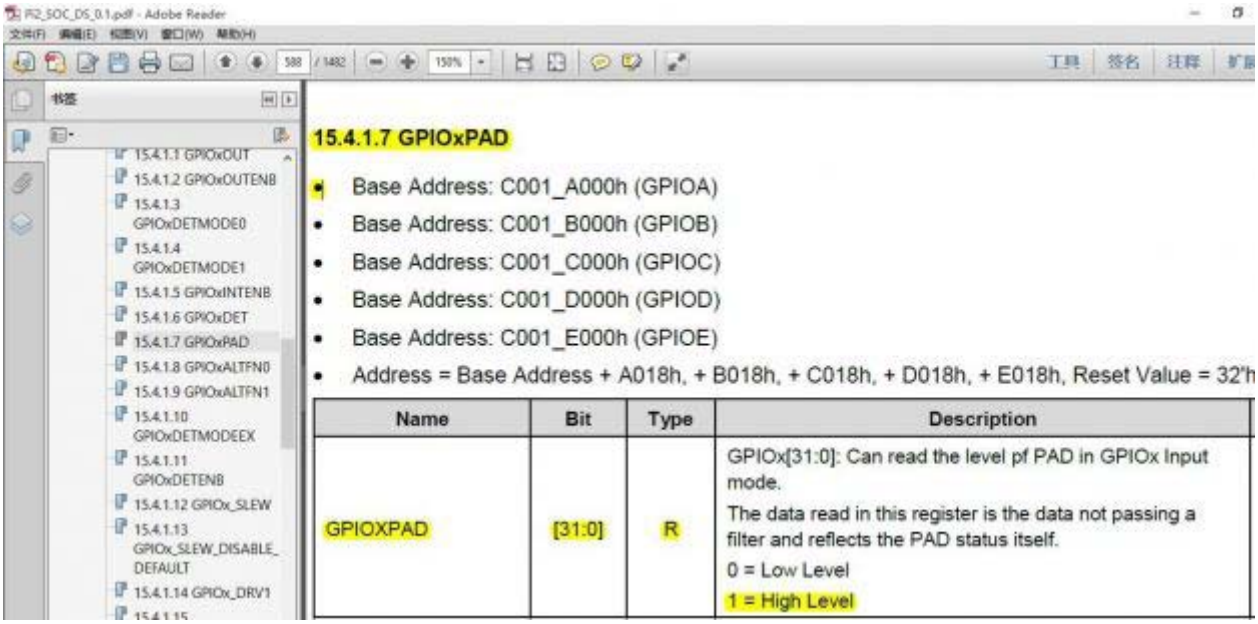
- Base Address: C001_A000h (GPIOA)
- Base Address: C001_B000h (GPIOB)
- Base Address: C001_C000h (GPIOC)
- Base Address: C001_D000h (GPIOD)
- Base Address: C001_E000h (GPIOE)
- Address = Base Address + A000h, + B000h, + C000h, + D000h, + E000h, Reset Value = 32

Name	Bit	Type	Description
GPIOXOUT	[31:0]	RW	GPIOx[31:0]: Specifies the output value in GPIOx output mode. NOTE: This bit should be set as "1" (Input mode) or "0" (Output mode) to use the Open drain pins in Input/Output mode. 0 = Low Level 1 = High Level

图 12 输出寄存器初始化设置

6）　主循环，检测到按键高电平就点亮对应 LED 灯

这里主要是检测只读状态寄存器 GPIOxPAD，参考手册 15.4.1.7 GPIOxPAD 的说明，如图 13 所示。读取 C13,11,8,7 引脚对应 GPIOxPAD 寄存器的位，若为 1 则说明输入的是高电平，若为 0 则说明是输入的是低电平。



15.4.1.7 GPIOxPAD

- Base Address: C001_A000h (GPIOA)
- Base Address: C001_B000h (GPIOB)
- Base Address: C001_C000h (GPIOC)
- Base Address: C001_D000h (GPIOD)
- Base Address: C001_E000h (GPIOE)
- Address = Base Address + A018h, + B018h, + C018h, + D018h, + E018h, Reset Value = 32'h

Name	Bit	Type	Description
GPIOXPAD	[31:0]	R	GPIOx[31:0]: Can read the level pf PAD in GPIOx Input mode. The data read in this register is the data not passing a filter and reflects the PAD status itself. 0 = Low Level 1 = High Level

图 13　**GPIOxPAD 状态寄存器读取说明**

具体程序细节如下：

```
if(GPIOCPAD & (0x1<<13))     // 判断 C13 引脚是否有高电平输入
{
                                        usleep(100000);
                                        GPIOBOUT&= ~(0x1<<27);      // 若有高平输入，则让 B27 的灯亮
}
```

```
else
{
                                        GPIOBOUT|= (0x1<<27);      //     若无高平输入，则 B27 的灯灭
}
```

其它按键对应 LED 灯的程序与之类似，最终的完整程序见图 14 和附件。

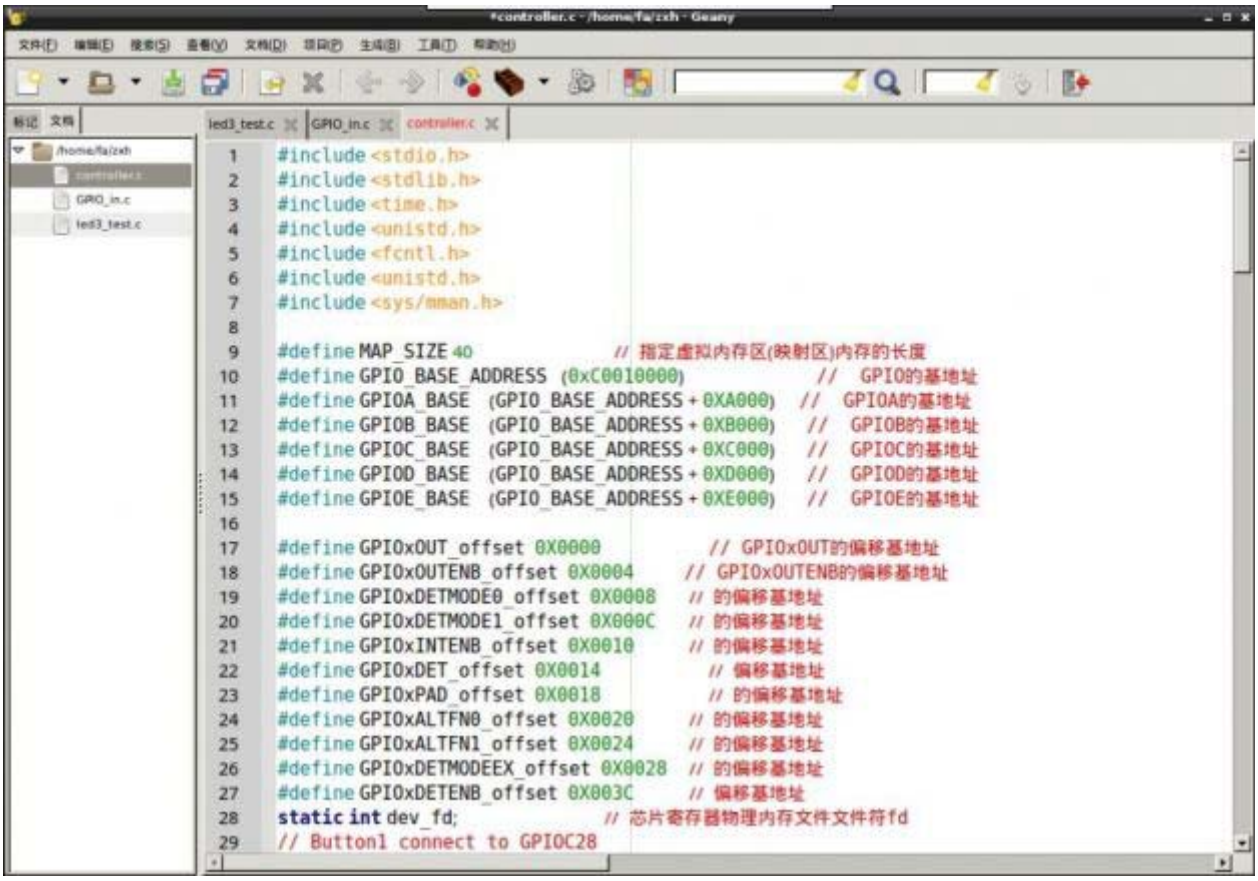


图 14 头文件和基地址定义

```
32  int main(int argc, char **argv)
33  {
34    int i = 0;
35    /*第一步：打开芯片寄存器物理内存文件*/
36    dev_fd = open("/dev/mem", O_RDWR | O_NDELAY); //打开芯片寄存器物理内存文件，可读写
37    if (dev_fd < 0)
38    {
39      printf("open(/dev/mem) failed.");
40      return 0;
41    }
42    /* 第二步：芯片寄存器物理地址映射为用户空间的虚拟地址*/
43    // 即将寄存器的那段内存映射到用户空间，返回映射首地址指针base
44    unsigned int base = (unsigned int)mmap(NULL, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, dev_fd, GPIOC_BASE );
45
46    #define GPIOCOUT *(unsigned int *)(base + GPIOxOUT_offset)
47    #define GPIOCOUTENB *(unsigned int *)(base + GPIOxOUTENB_offset) //0x04
48    #define GPIOCALTFN0 *(unsigned int *)(base + GPIOxALTFN0_offset) //0x20
49    #define GPIOCALTFN1 *(unsigned int *)(base + GPIOxALTFN1_offset) //0x20
50    #define GPIOCDETMODE0 *(unsigned int *)(base + GPIOxDETMODE0_offset) //0x20
51    #define GPIOCDETMODE1 *(unsigned int *)(base + GPIOxDETMODE1_offset) //0x20
52    #define GPIOCDETMODEEX *(unsigned int *)(base + GPIOxDETMODEEX_offset) //0x20
53    #define GPIOCINTENB *(unsigned int *)(base + GPIOxINTENB_offset) //0x20
54    #define GPIOCPAD *(unsigned int *)(base + GPIOxPAD_offset)
55    #define GPIOCINTENB *(unsigned int *)(base + GPIOxINTENB_offset)
56    #define GPIOCINTENB *(unsigned int *)(base + GPIOxINTENB_offset)
```

图 15 打开芯片 C 组 GPIO 寄存器物理内存文件并映射为虚拟地址

```
56  #define GPIOCINTENB *(unsigned int *)(base + GPIOxINTENB_offset)
57
58  unsigned int base_B = (unsigned int)mmap(NULL, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, dev_fd, GPIOB_BASE );
59  #define GPIOBOUT *(unsigned int *)(base_B + GPIOxOUT_offset)
60  #define GPIOBOUTENB *(unsigned int *)(base_B + GPIOxOUTENB_offset) //0x04
61  #define GPIOBALTFN0 *(unsigned int *)(base_B + GPIOxALTFN0_offset) //0x20
62  #define GPIOBALTFN1 *(unsigned int *)(base_B + GPIOxALTFN1_offset) //0x20
63  #define GPIOBDETMODE0 *(unsigned int *)(base_B + GPIOxDETMODE0_offset) //0x20
64  #define GPIOBDETMODE1 *(unsigned int *)(base_B + GPIOxDETMODE1_offset) //0x20
65  #define GPIOBDETMODEEX *(unsigned int *)(base_B + GPIOxDETMODEEX_offset) //0x20
66  #define GPIOBINTENB *(unsigned int *)(base_B + GPIOxINTENB_offset) //0x20
67  #define GPIOBPAD *(unsigned int *)(base_B + GPIOxPAD_offset)
68  #define GPIOBINTENB *(unsigned int *)(base_B + GPIOxINTENB_offset)
69  #define GPIOBINTENB *(unsigned int *)(base_B + GPIOxINTENB_offset)
70
71  /* 第三步：初始化GPIO*/
72  //设置端口复用
73  // 定义GPIOC11复用功能 ALTfunction1 = 01
74  // GPIOCALTFN0 &= ~(0x3<<22); // 将GPIOCALTFN0的第22位和第23位清零：xx11左移22位后取反xx00...0，对应GPIOx11 [22:23]
75  // GPIOCALTFN0 |= (0x1<<22);   // 定义GPIO11引脚为GPIO功能：xx01左移22位 : ALT Function1 = 01
76  GPIOCALTFN0 &= ~((0xf<<14)|(0x3<<22)|(0x3<<26)); //C13=[27:26],C11=[23:22],C8=[17:16],C7=[15:14]，都为 01
77  GPIOCALTFN0 |= ((0x1<<22)|(0x1<<26)|(0x5<<14));
78
79  GPIOBALTFN0 &= ~((0x3f<<26)|(0x3<<22)); //B31=[31:30],B30=[29:28],B29=[27:26],B27=[23:22]， 都为 01
80  GPIOBALTFN0 |= ((0x15<<26)|(0x1<<22));
```

图 16　　将 B 组 GPIO 寄存器物理内存文件映射为虚拟地址

```
80  GPIOBALTFN0 |= ((0x15<<26)|(0x1<<22));
81
82  //设置GPIOC模式
83  //使能GPIO的输入/出功能
84  GPIOCOUTENB &= ~((0x1<<13)|(0x1<<11)|(0x1<<8)|(0x1<<7)); //设置GPIOC13,11,8,7为输入模式，对应位置0，Bit=[31:0]
85  GPIOBOUTENB |= ((0x1<<31)|(0x1<<30)|(0x1<<29)|(0x1<<27)); //设置GPIOB31,30,29,27为输出模式，对应位置1：第31位置1代表GPIOx31 Bi
86
87  //预设GPIOC状态
88  GPIOBOUT &= ~((0x1<<31)|(0x1<<30)|(0x1<<29)|(0x1<<27)); //设置GPIOC31，30，29，27为低电平，LED灭
89  //GPIOBOUT &=  ~ (0x1<<31);
90  //初始化输入寄存器GPIOC13,11,8,7的检测模式为低电平检测
91  GPIOCDETMODE0 &= ~((0x2<<26)|(0x2<<22)|(0x2<<16)|(0x2<<14)); // C13=[27:26],C11=[23:22],C8=[17:16],C7=[15:14], 都为00
92  GPIOCDETMODEEX &= ~((0x1<<13)|(0x1<<11)|(0x1<<8)|(0x1<<7)); // C13=bit[13],C11=[11],C8=[8],C7=[7] = 0
93  GPIOCINTENB &= ~((0x1<<13)|(0x1<<11)|(0x1<<8)|(0x1<<7));    // 关闭中断始能
```

图 17 设置 GPIO 输入输出功能和电平检测模式

```
93  GPIOCINTENB &= ~((0x1<<13)|(0x1<<11)|(0x1<<8)|(0x1<<7));    // 关闭中断始能
94
95  //while(i<10000)
96  while(1)
97  {
98    //usleep(100000);
99    // *****************Right*************************//
100   if(GPIOCPAD & (0x1<<13)) // 判断PAD第13位是否是高电平
101   {
102     usleep(100000);
103     GPIOBOUT &= ~(0x1<<27); //高电平灯灭
104     //GPIOCOUT |= (0x1<<11); // 高电平灯亮
105   }
106   else
107   {
108     //usleep(100000);
109     GPIOBOUT |= (0x1<<27); //低电平灯亮
110     //GPIOCOUT &= ~(0x1<<11); //低电平灯灭
111   }
112
113   // *****************Forward************************//
114
```

图 18 主循环检测按键输入判断对应 LED 输出代表相应的小车动作

```
113     // *****************Forward*************************//
114
115     if(GPIOCPAD & (0x1<<11)) // 判断PAD第11位是否是高电平
116     {
117       usleep(100000);
118       GPIOBOUT &= ~(0x1<<30);  // 高电平灯灭
119       //GPIOCOUT |= (0x1<<30); // 高电平灯亮
120     }
121     else
122     {
123       //usleep(100000);
124       GPIOBOUT |=(0x1<<30);  // 低电平灯亮
125       //GPIOCOUT &= ~(0x1<<30); //低电平灯灭
126     }
127
128     // *****************Backward*************************//
129     if(GPIOCPAD & (0x1<<8)) // 判断PAD第28位是否是高电平
130     {
131       usleep(100000);
132       GPIOBOUT &= ~(0x1<<31);  // 高电平灯灭
133       //GPIOCOUT |= (0x1<<11); // 高电平灯亮
134     }
135     else
136     {
137       //usleep(100000);
138       GPIOBOUT |=(0x1<<31);  // 低电平灯亮
139       //GPIOCOUT &= ~(0x1<<11); //低电平灯灭
140     }
141
```

图 19 前进和后退的控制

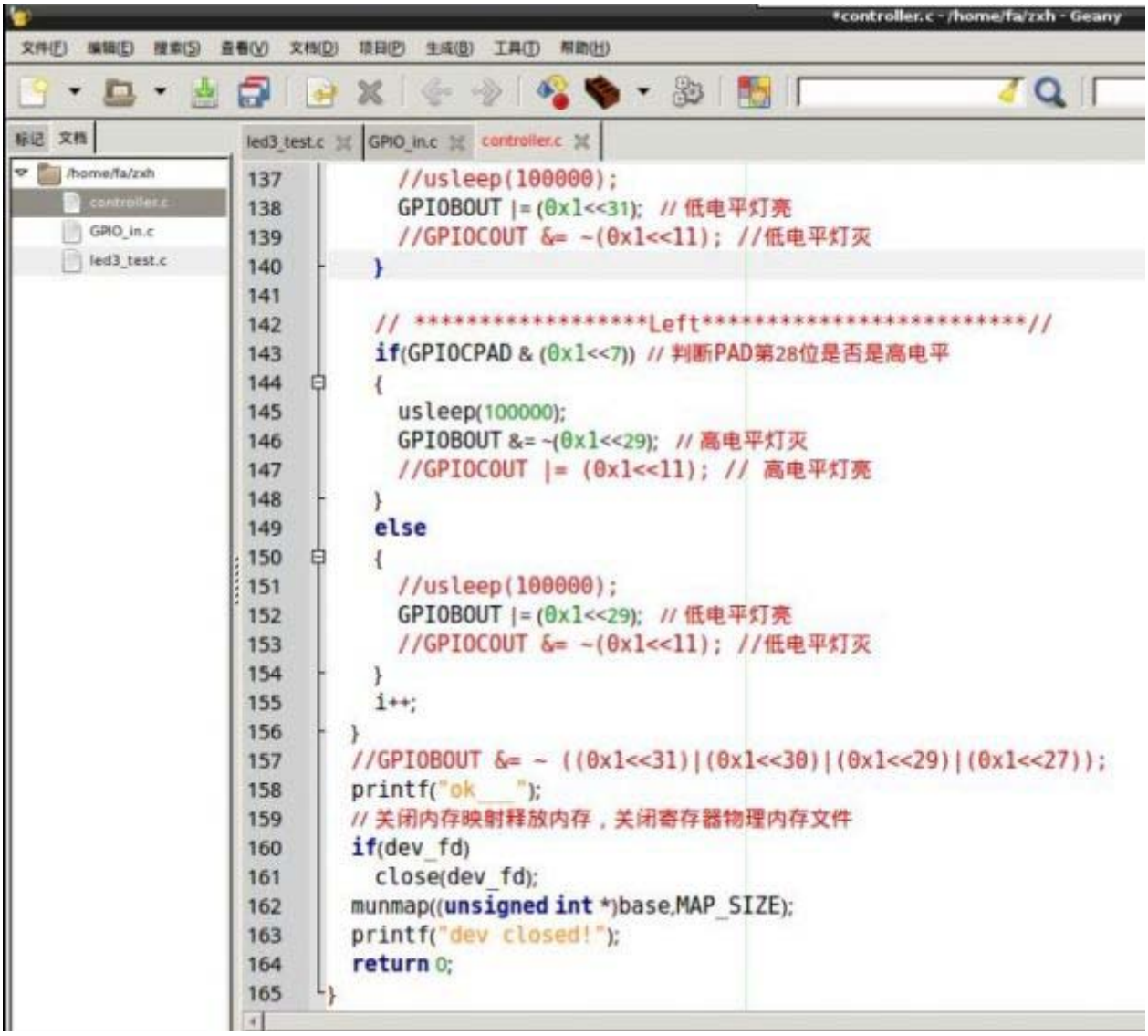


图 20 左转控制和运行结束和关闭寄存器文件

最后运程结果如图所示：

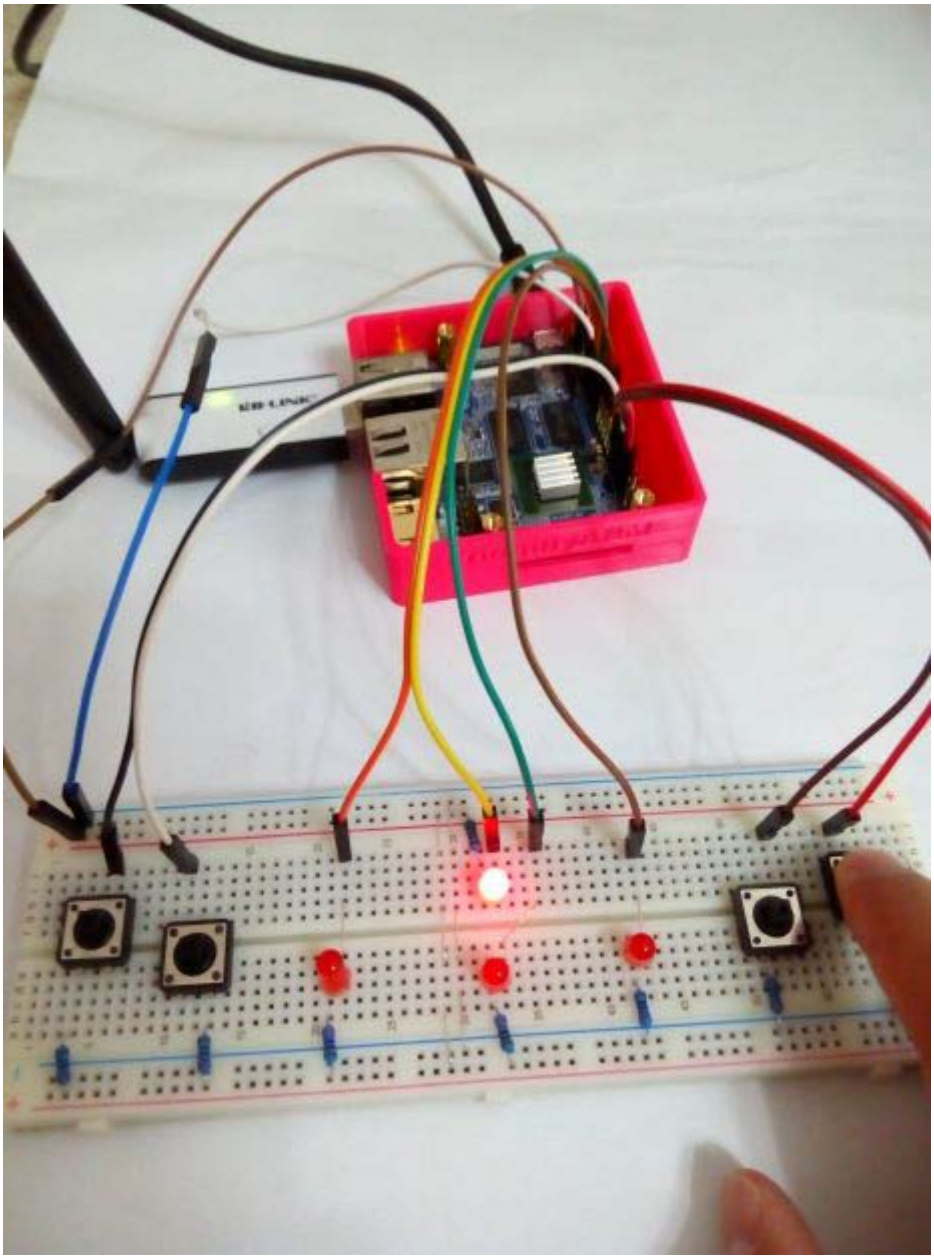


图 21 前进控制前灯亮

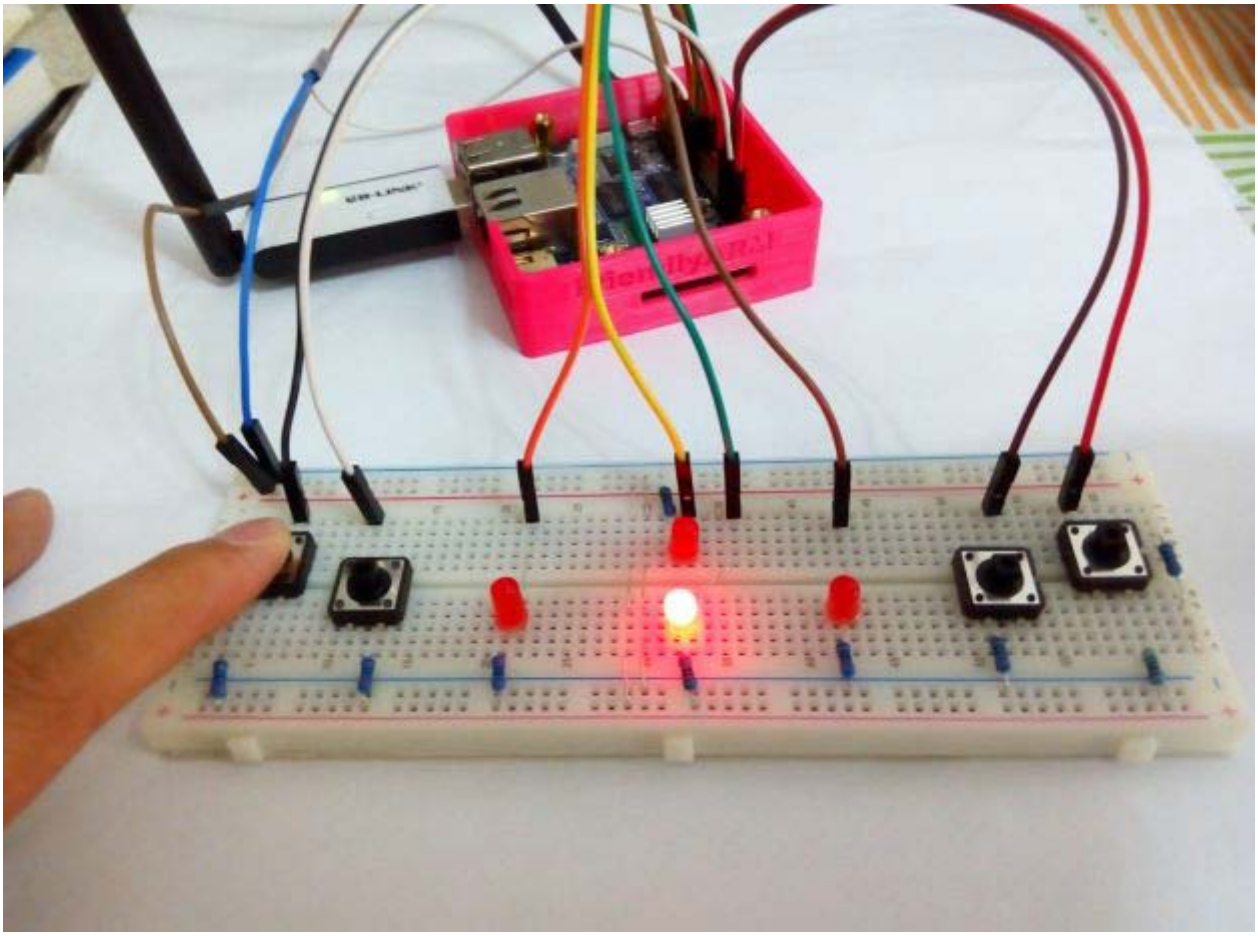


图 22 后退控制后灯亮

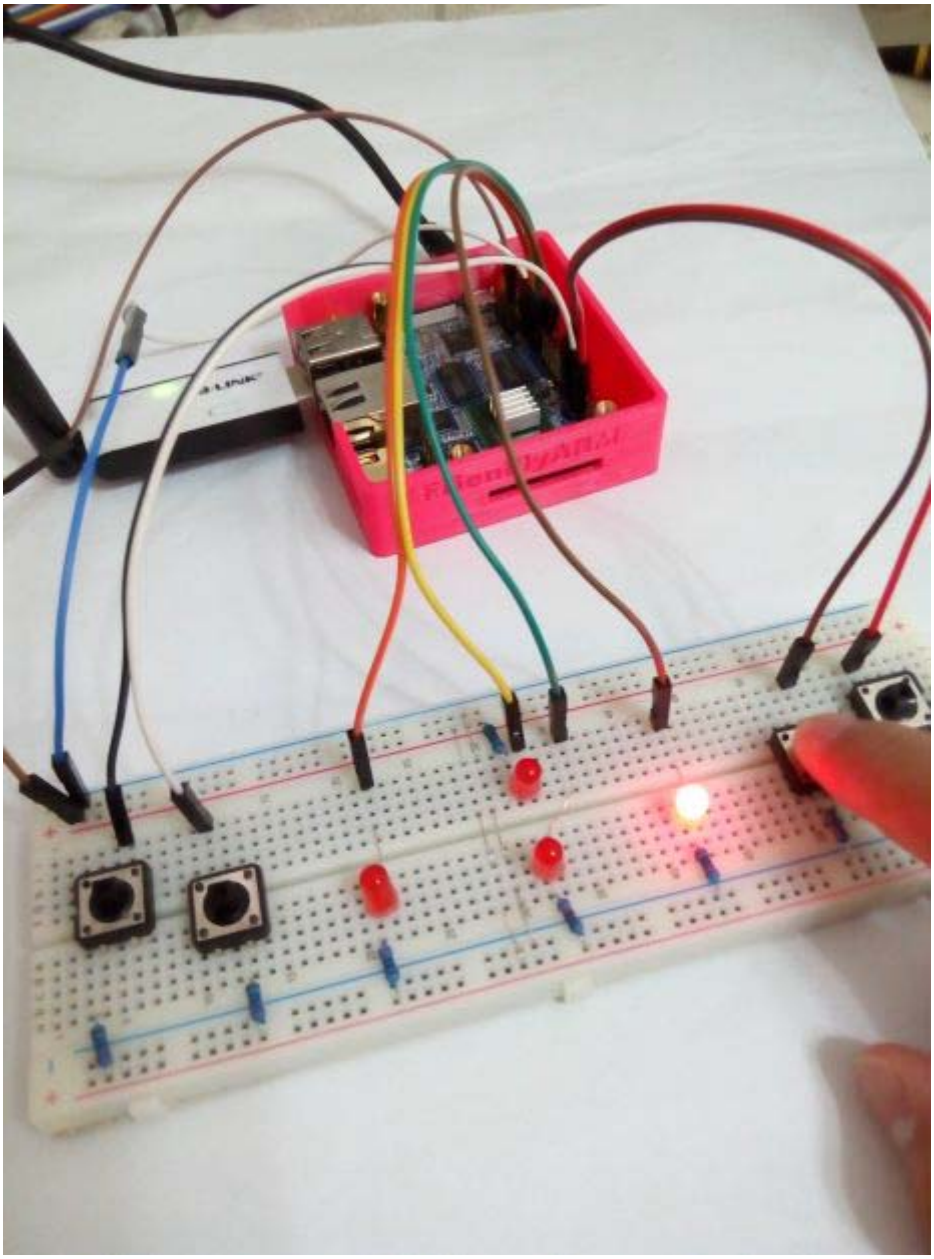


图 23 右转控制右灯亮

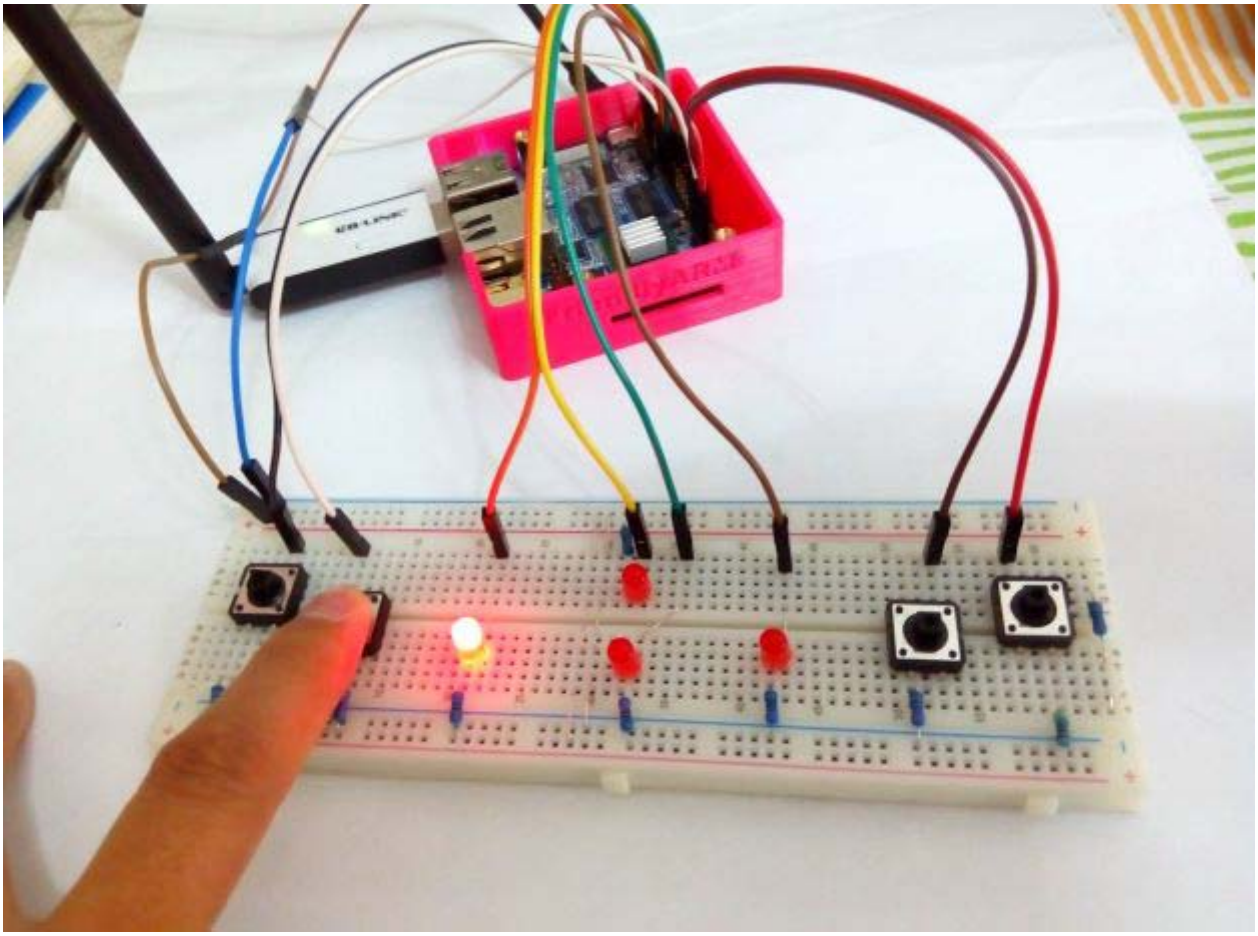


图 24 左转控制左灯亮

4. 总结

至此，本节内容结束，遥控器也初具模样，通过寄存器配置的方式实现了基本的输入输出控制。下一步将进行无线串口通信功能的开发，由于最近比较忙可能更新会稍晚一些，但一定会尽心尽力完成。中间可能会穿插一些小的玩法，比如通过网页登陆 NanoPi M2 远程操控，实现 NanoPi 真正意义的物联网价值。

技巧六：Nanopi M2 之基于 QT 图形界面编程操作 MySQL 数据库

来自：zxl_zxl http://bbs.elecfans.com/jishu_705217_1_1.html

在上一篇使用报告中，介绍了在 Nanopi M2 上安装 Mysql 数据库管理系统的服务器端和客户端软件，并介绍了如何安装 Mysql 数据库可视化管理软件，以及如何创建数据库、数据库表，通过 SQL 语句向 TUser 数据库表插入新数据。

在实际工作或项目开发过程中，我们几乎很少直接在服务器上通过数据库管理软件来对数据库表进行数据的“增加”、“删除”、“修改”和“查询”，通常都是经过一个网页或客户端图形界面软件，通过输入框、列表框等人机交互控件来完成。

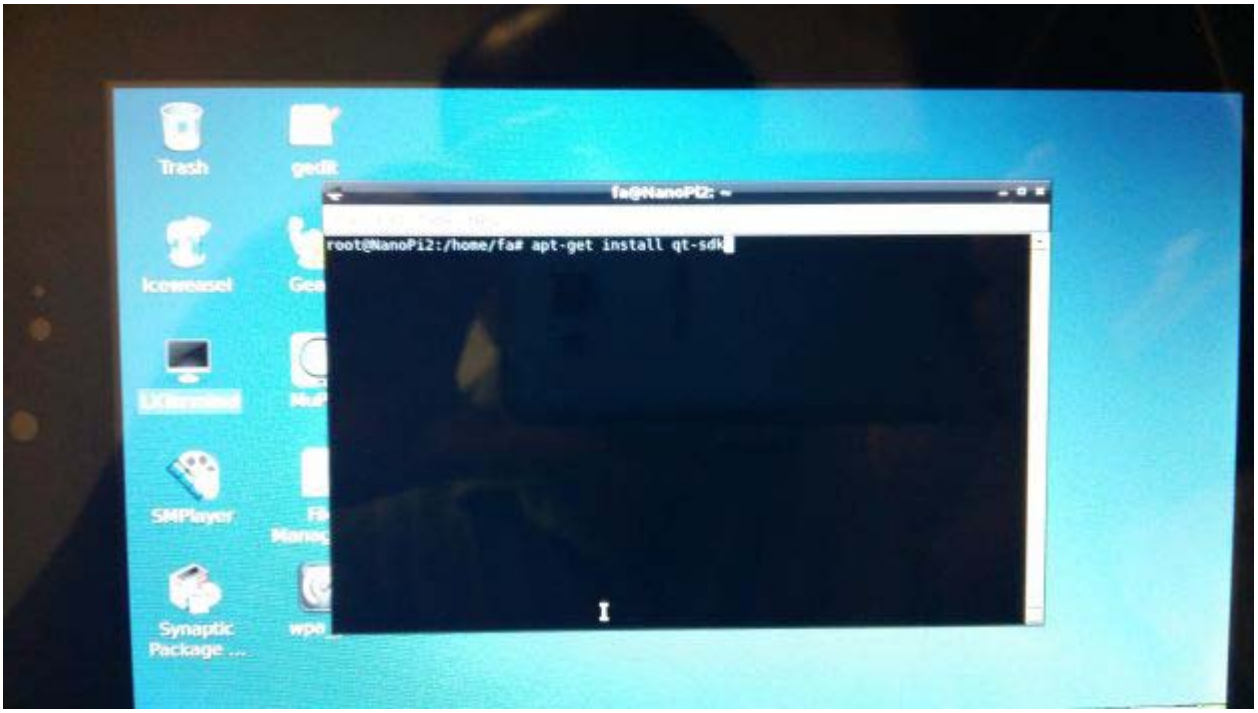
QT 作为一款开源的软件开发环境、丰富的图形控件，以及快速高效软件界面的设计，在基于桌面和嵌入式类型的 Linux 操作系统中得到了广泛应用。本篇报告将在上一篇报告的基础上，创建 QT 工程项目，开发基于 QT 的 MySQL 数据库客户端软件，实现一个简单的数据库表查询功能。

1.安装 QT

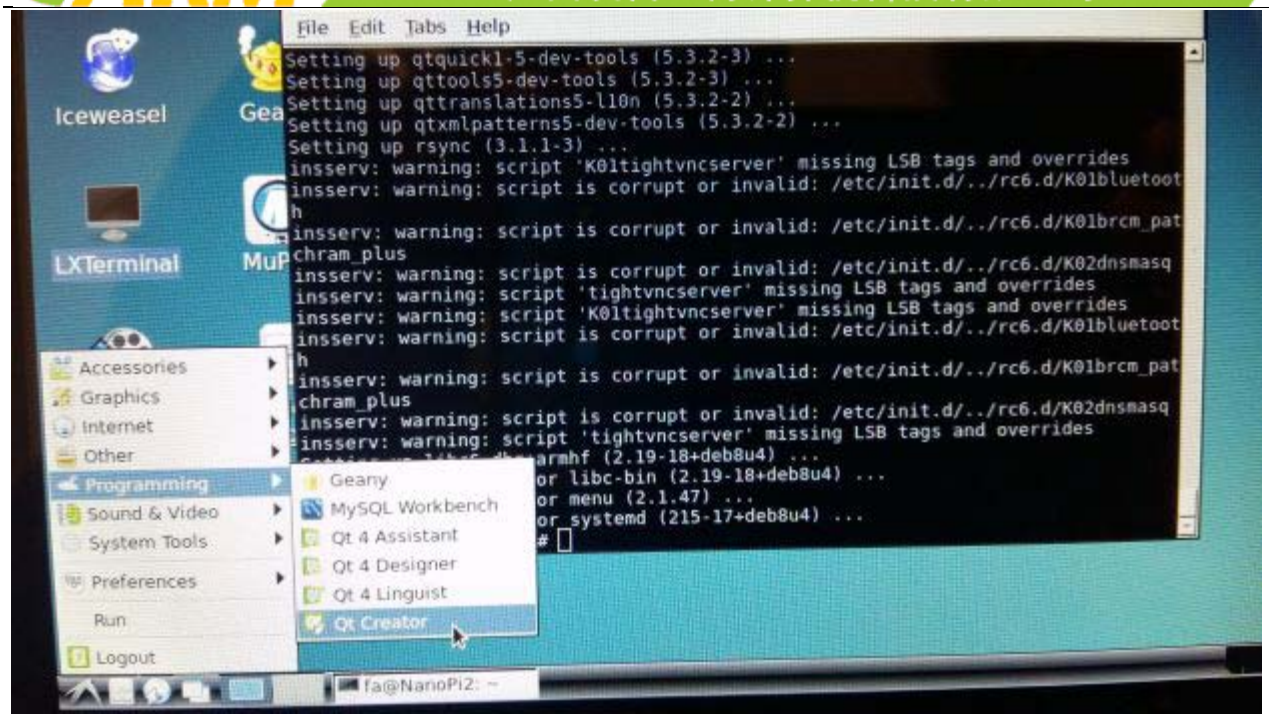
首先第一步，在 Nanopi M2 上安装 QT 开发环境，虽然有坛友介绍过一种 QT 的安装方法，本教程采用另外一种安装方法，一步实现 QT4.8，QT5,.3.2,以及 QTCreator 的安装。

输入命令：sudo apt-get install qt-sdk

如图所示：



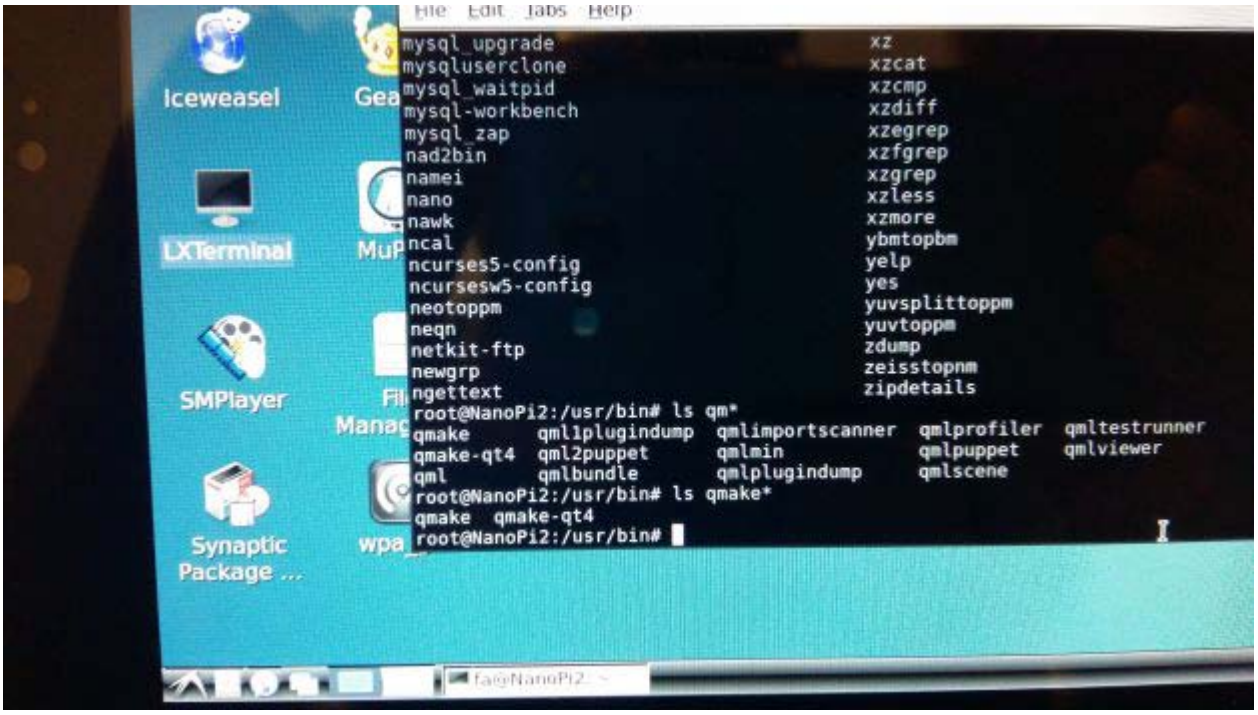
命令执行结束后，如系统所示，说明 QT 已在 Nanopi M2 上安装成功。



输入命令：ls /usr/bin/qmake*

输出：qmake qmake-qt4

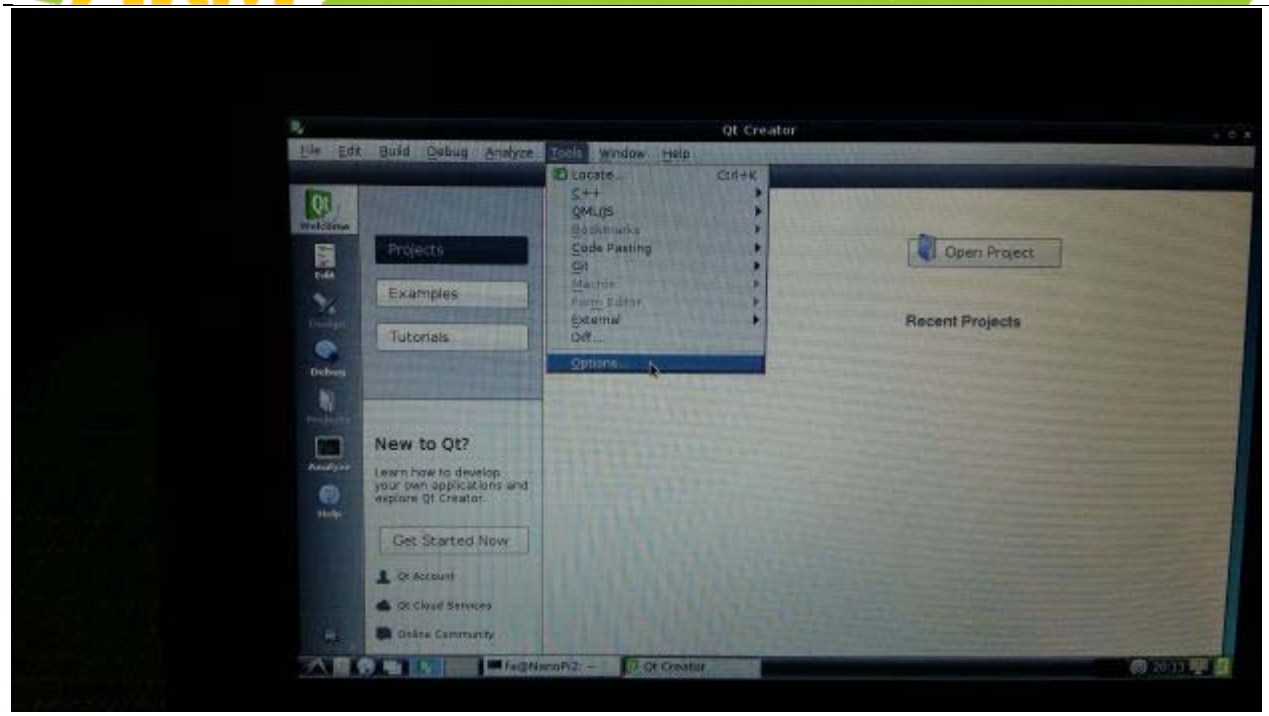
如图所示：



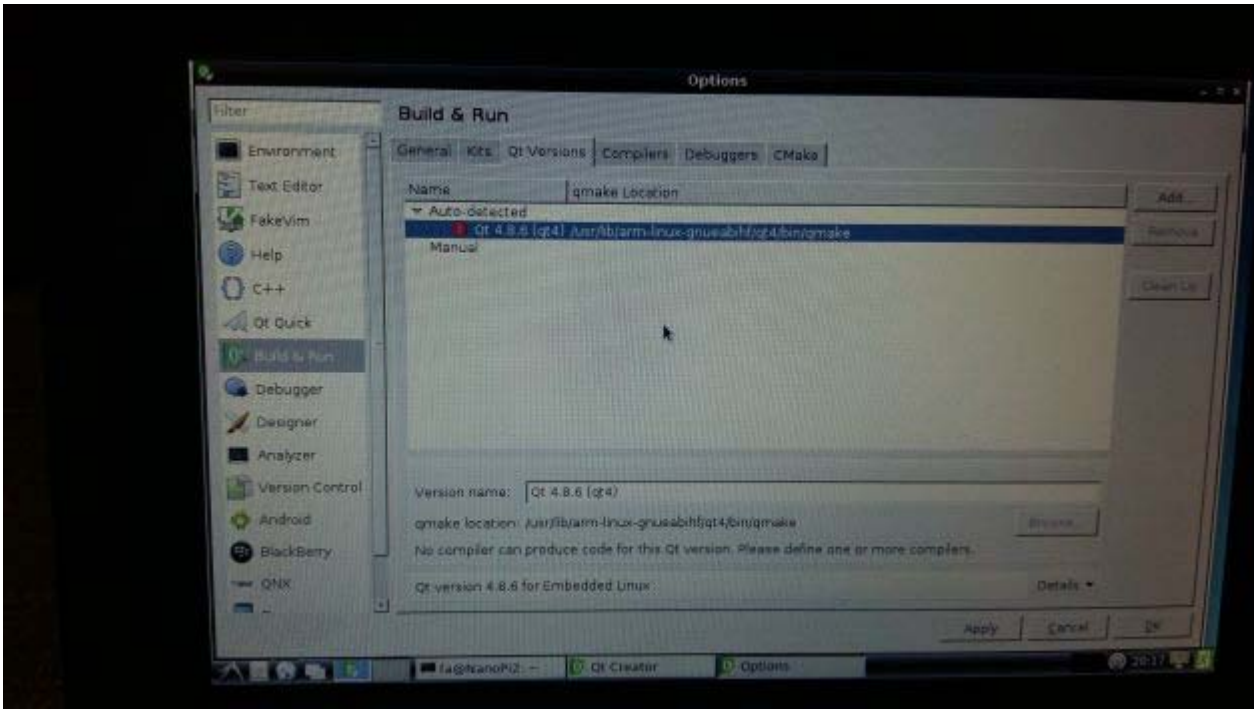
其中 qmake 为 QT5 版本，qmake-qt4 为 QT4.8.6 版本。

本篇报告主要介绍 QT4.8.6 的开发环境配置。

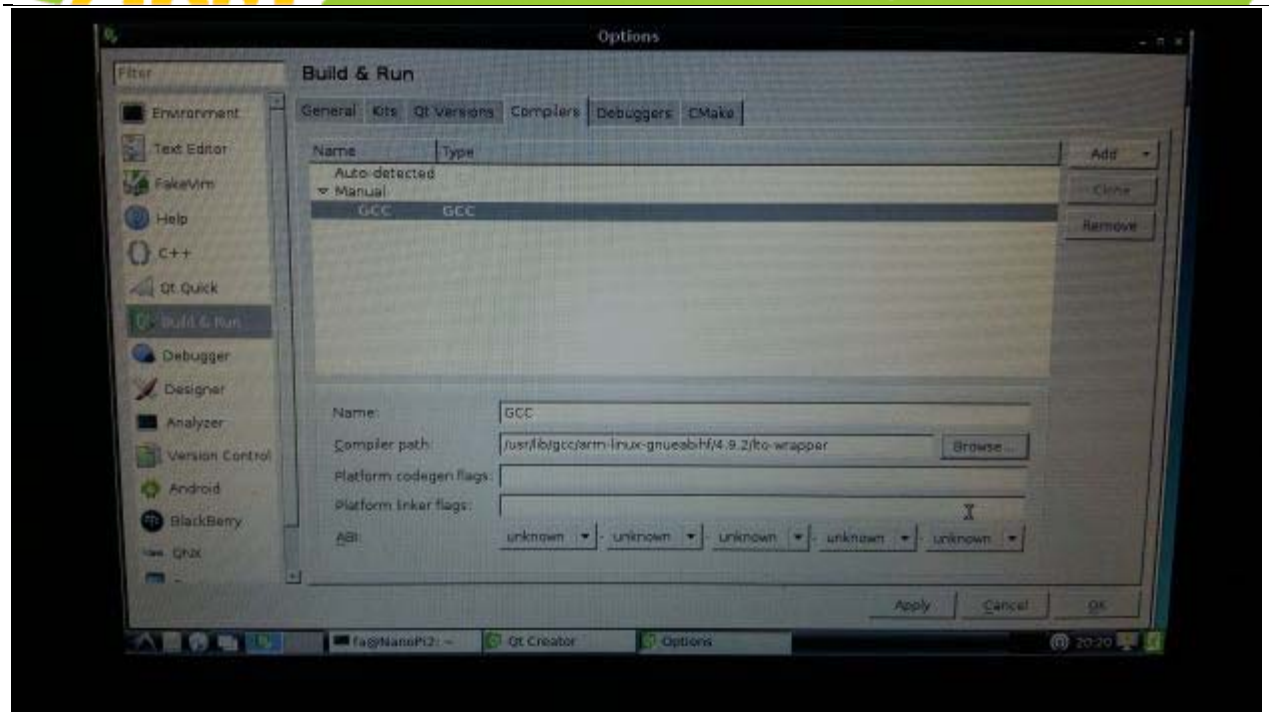
在开始——Programming——Qt Creator，启动 QT 项目开发界面，如图所示：



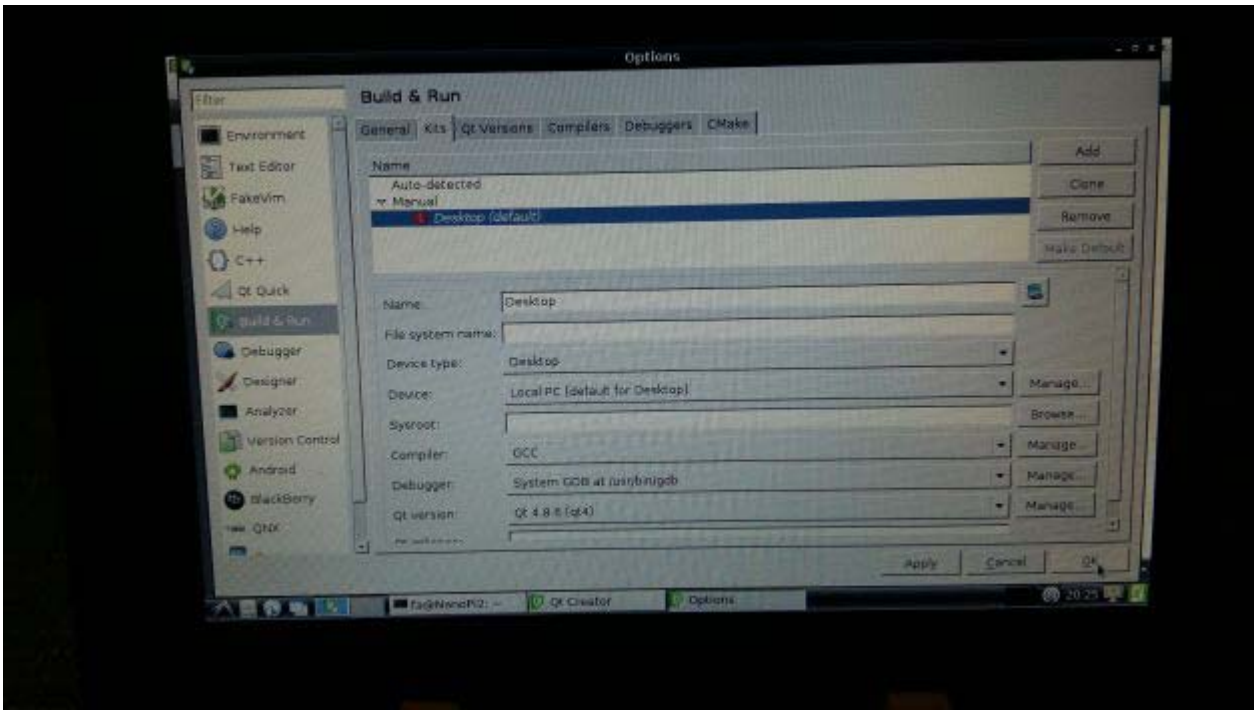
启动界面后，再点击菜单“Tools”，然后选择“Option”菜单项，弹出编译选项设置对话框，如下图所示：



在“Buile&Run”设置中，QT Version 已经自动识别了 QT4.8.6 的版本库，我们无需选择，点击 Compiler 便签，设置 gcc 编译器，如下图所示：



选择 kits 标签，设置如下：



然后点击“OK”按钮，退出编译设置对话框。

2. 安装 QT 访问 MySQL 数据库驱动程序

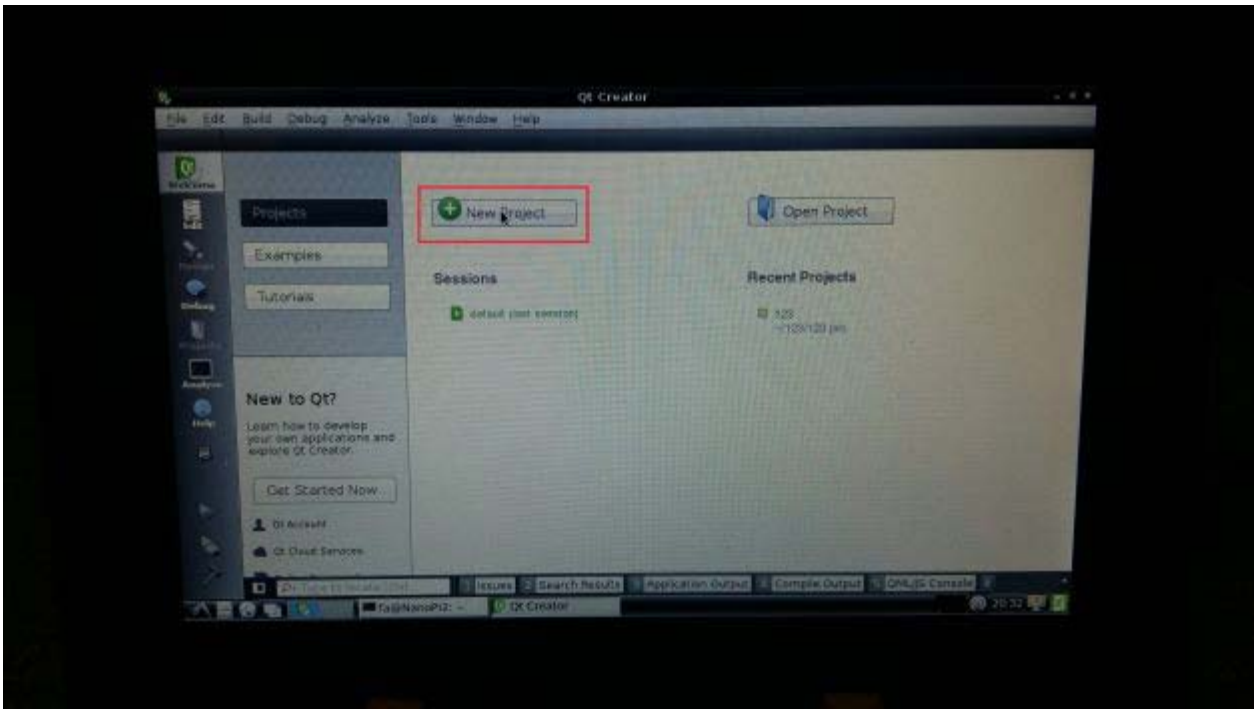
安装好 QT 后，还无法直接通过 QT 程序访问 MySQL 数据库，我们需要安装 QT 访问 MySQL 数据库的驱动程序。

对于 QT4，输入命令：sudo apt-get install　libqt4-sql-mysql

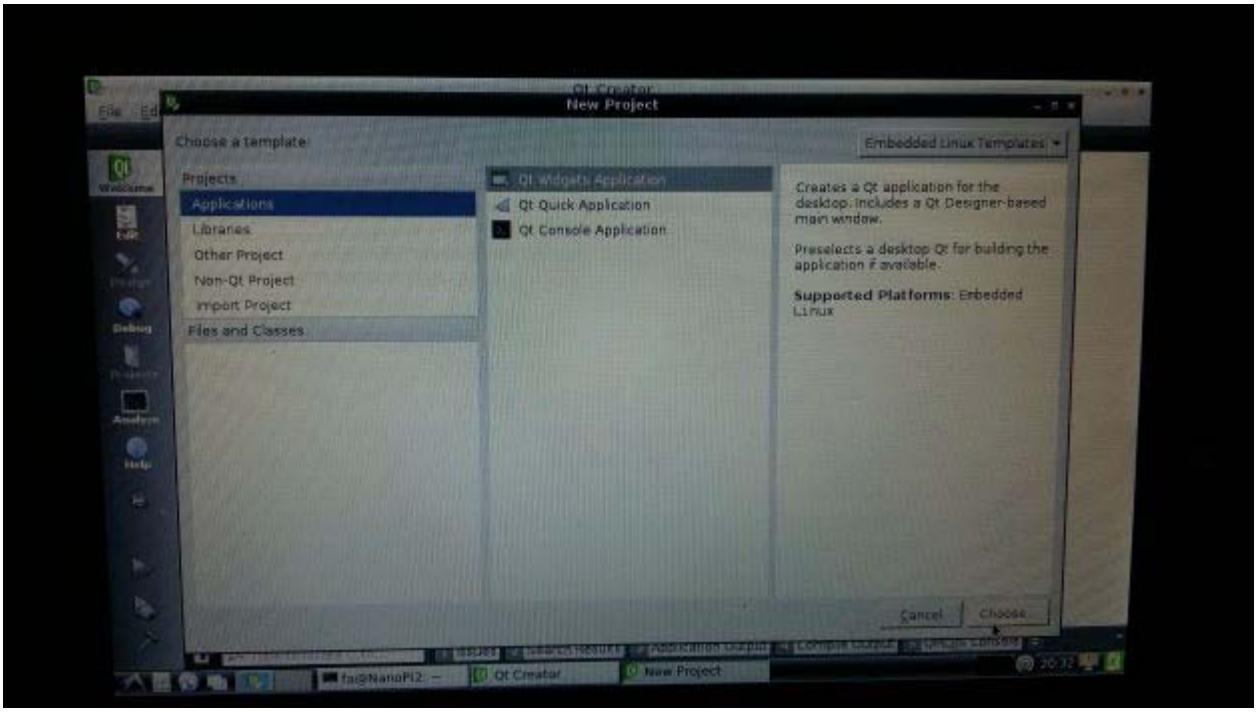
对于 QT5，输入命令：sudo apt-get install libqt5sql5-mysql

3. 创建 QT 工程

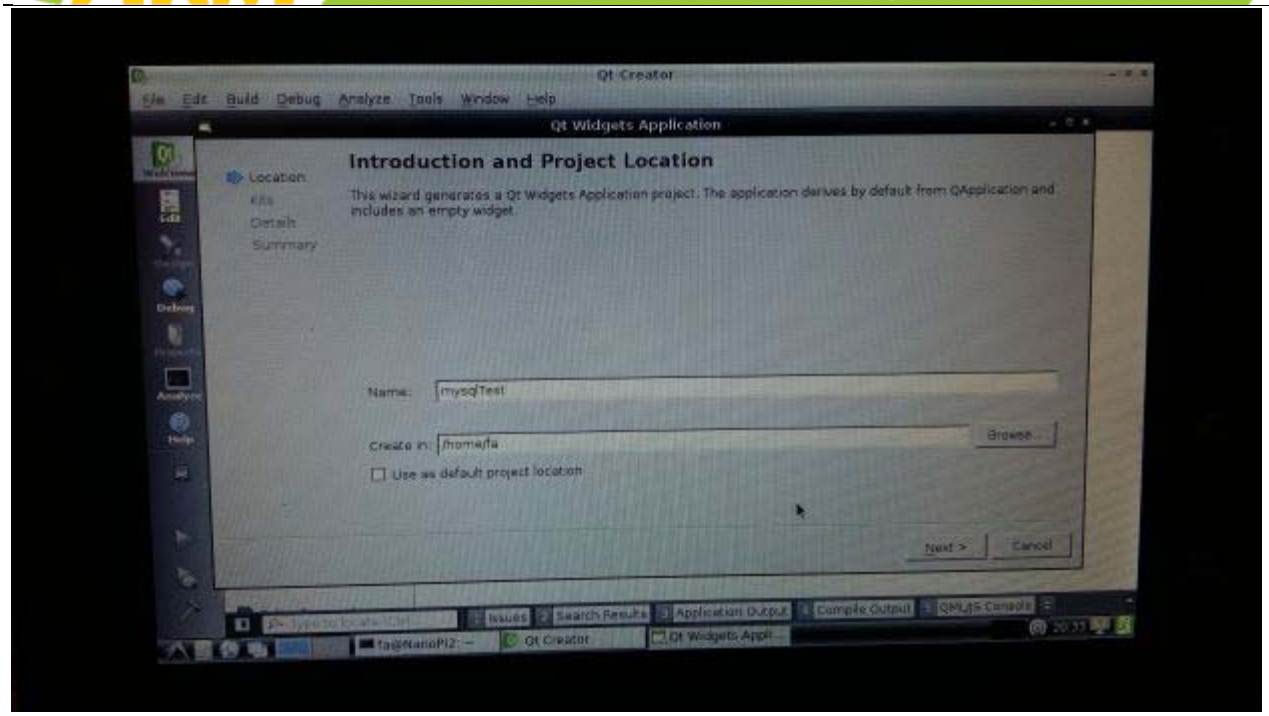
安装、配置好 QT 的开发环境之后，我们开始创建 QT 开发 MySQL 数据库客户端的工程项目，点击如下图所示的红色矩形框，进入启动创建项目命令：



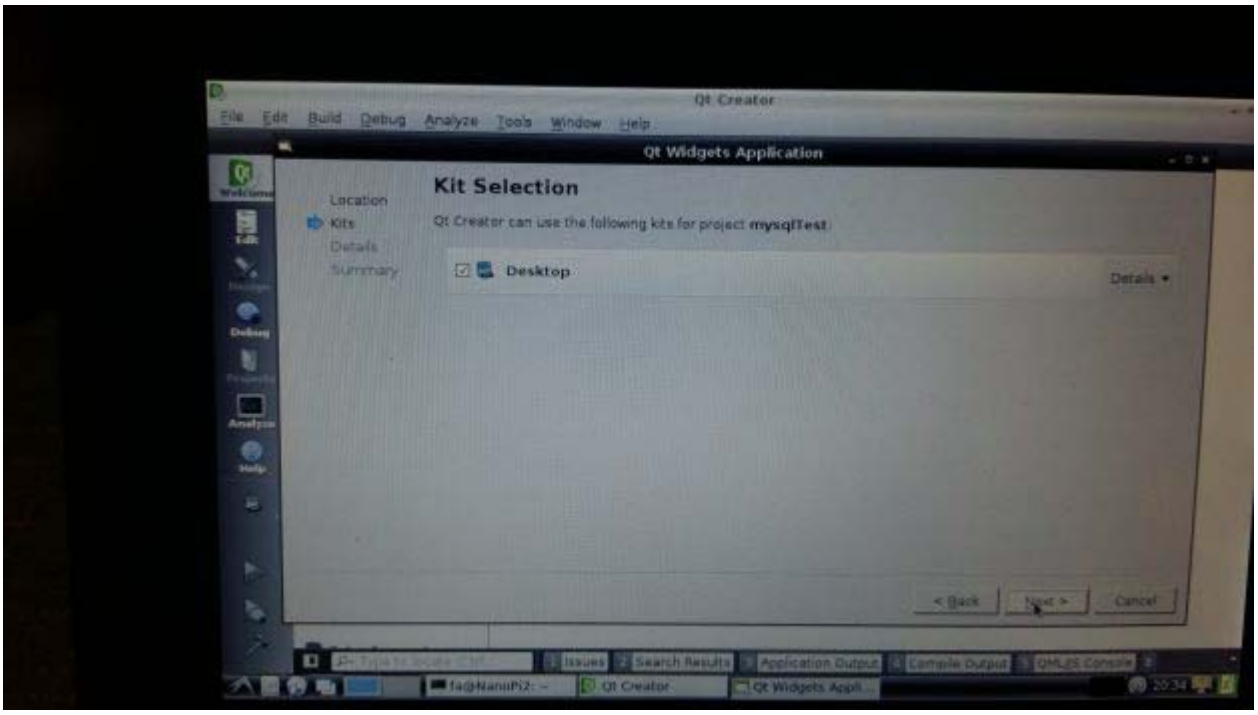
选择工程类型 QT Widgets application：



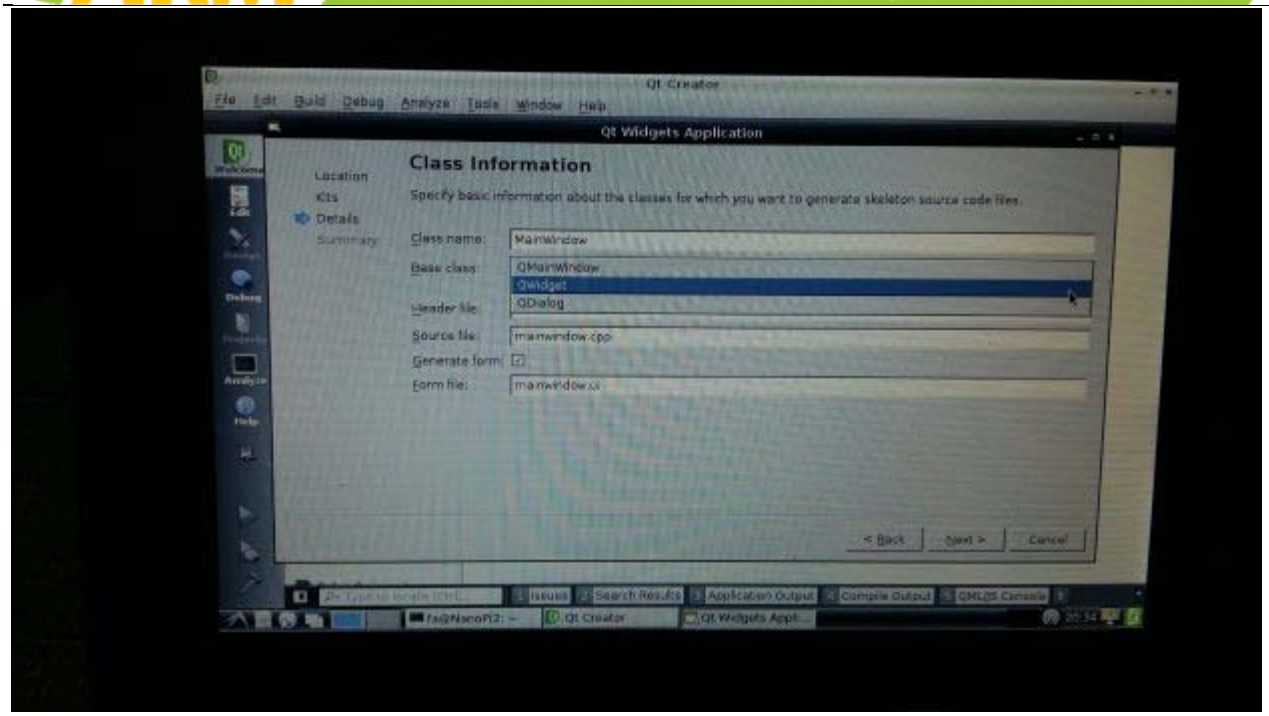
输入工程名称 mysqlTest：



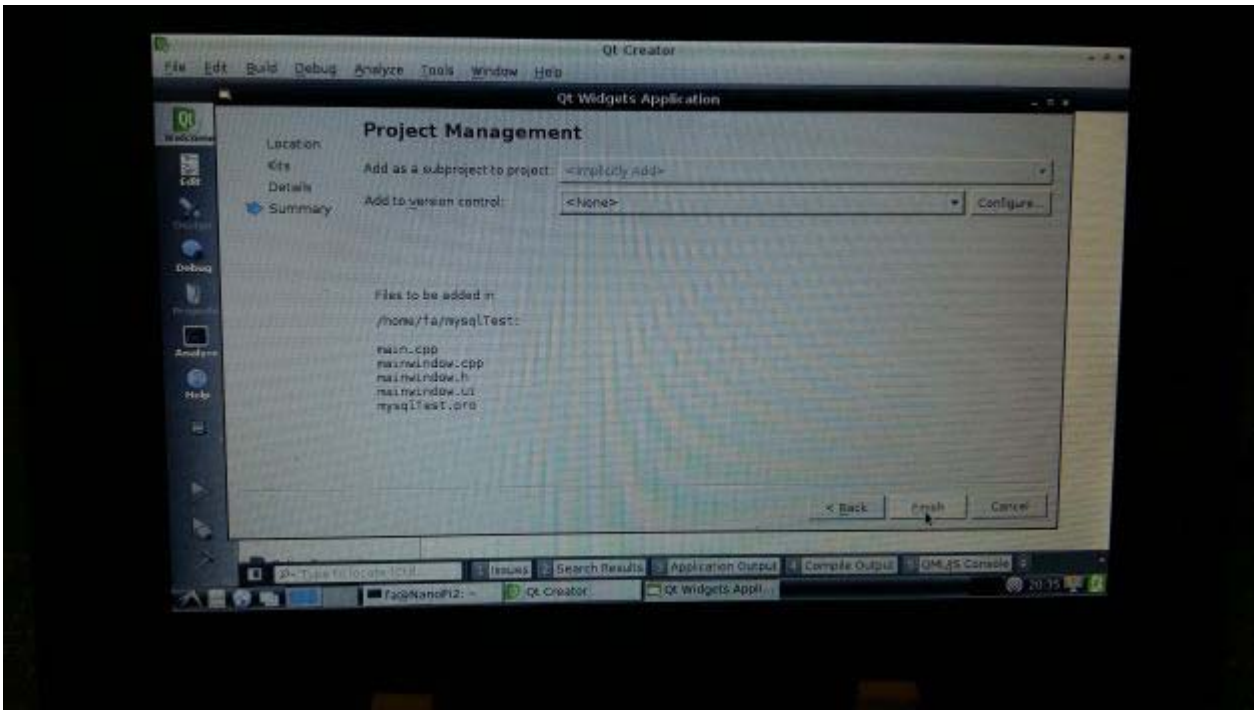
选择开发包类型：



选择基于对话框的开发类型 QWidget：



点击“Finish”按钮，完成工程创建：



3. 编写代码

创建好工程后，在如图的源代码中输入头文件，

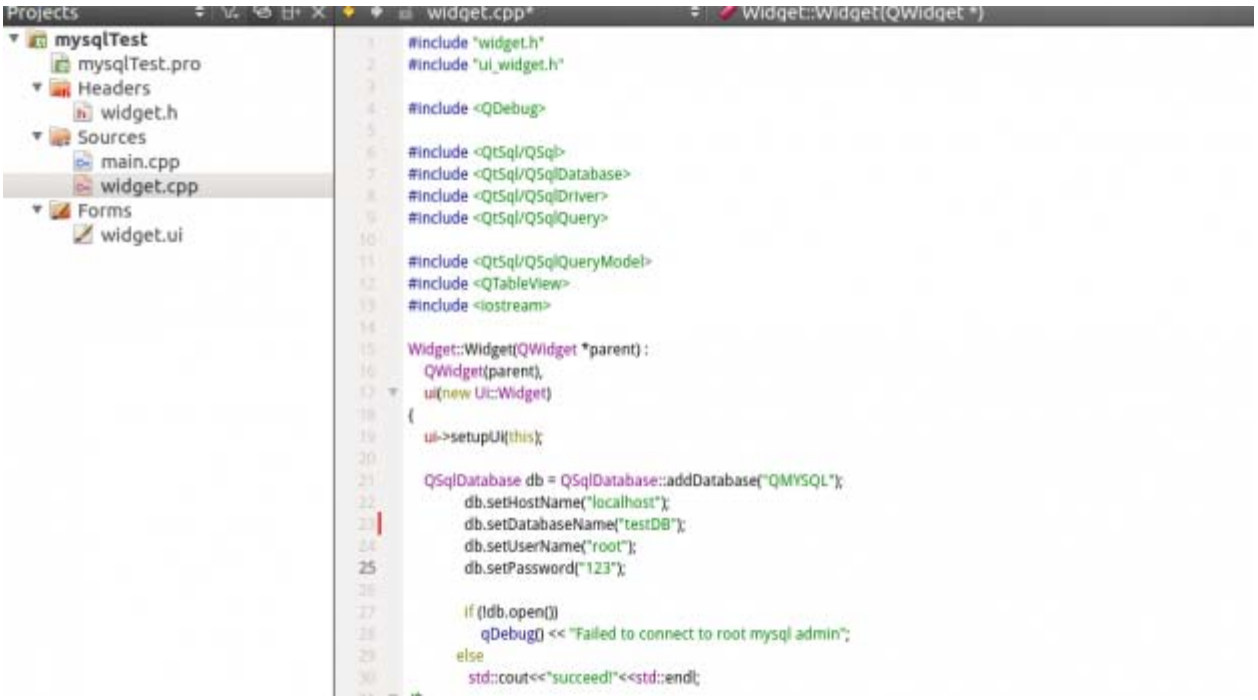
```
#include<QDebug>
#include<QtSql/QSql>
#include<QtSql/QSqlDatabase>
#include<QtSql/QSqlDriver>
#include<QtSql/QSqlQuery>
#include<QtSql/QSqlQueryModel>
#include<QTableView>
#include<iostream>
```

在 Widget 构造函数中，输入如下代码：

```
QSqlDatabase db =QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("localhost");
db.setDatabaseName("testDB");
db.setUserName("root");
db.setPassword("123");
if (!db.open())
     qDebug() << "Failed to connect toroot mysql admin";
else
     std::cout<<"succeed!"<<std::endl;
QSqlQueryModel*model = new QSqlQueryModel;
model->setQuery("select* from TUser");
model->setHeaderData(0,Qt::Horizontal, "id");
model->setHeaderData(1,Qt::Horizontal, "姓名");
model->setHeaderData(2, Qt::Horizontal,"年龄");
QTableView *view = new QTableView(this);
view->resize(400,200);
view->setWindowTitle("QSqlQueryModel");
view->setModel(model);
```

```
view->show();
db.close();
```

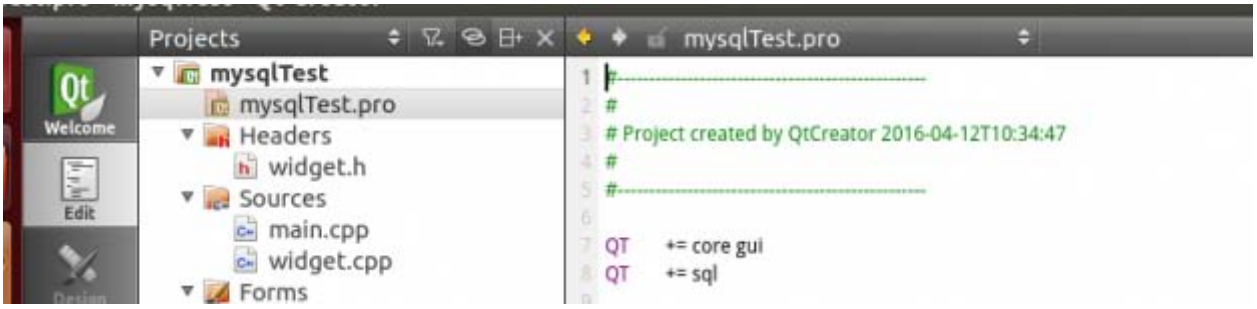
如图所示：



代码编辑结束后，还有一项重要的设置，在工程文件中输入：

```
QT                    +=sql
```

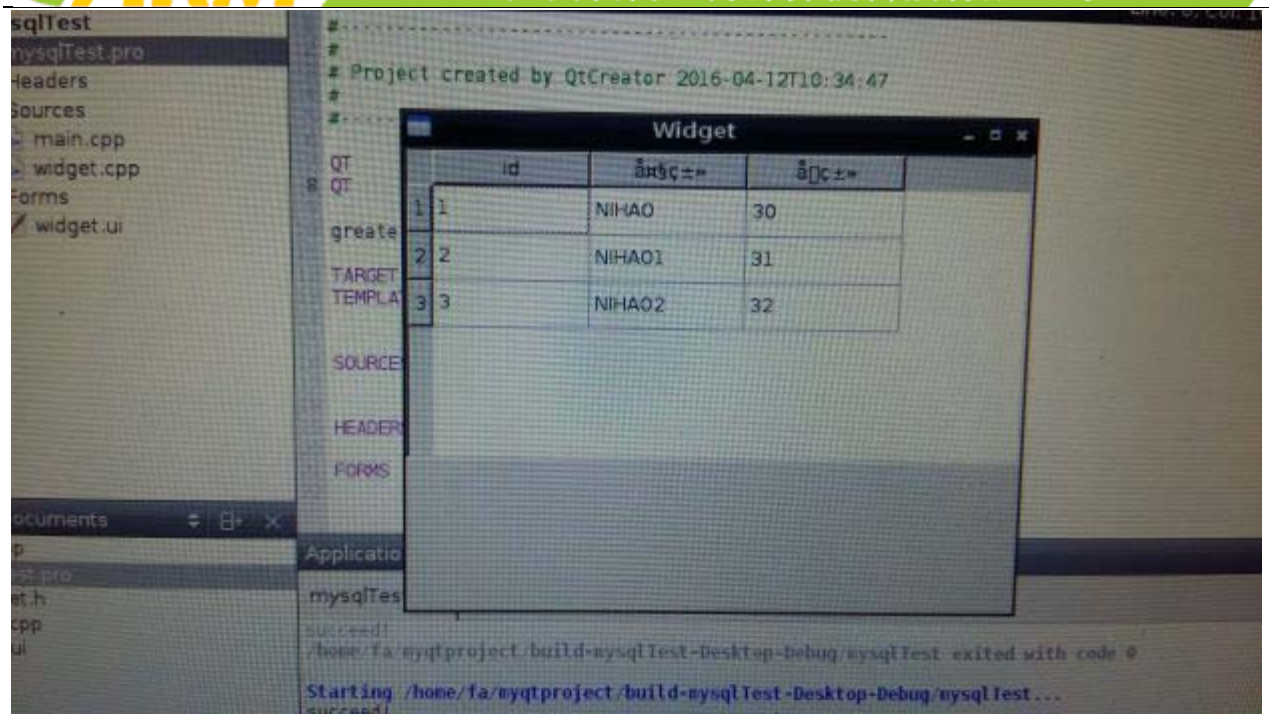
如图所示：



4. 编译执行

工程项目编辑结束后，点击 Run 图标执行编译和运行命令，如图所示：

软件执行结果，如下图所示：



列表控件中显示出了我们上一篇通过 INSER 语句插入的数据。

5.小结

这篇教程简单介绍了 QT 的安装、配置，以及 QT 访问 MySQL 数据库需要的驱动程序，然后介绍了需要的头文件和 C++代码。通过简单的一个小工程，即可实现对 MySQL 数据库可编程化操作，为后续的项目中的数据管理做了最基础工作，希望能够给大家带来小小的帮助，起到抛砖引玉的效果。

技巧七：NanoPi M2 debian 中 NS-2 的安装和使用

来自：liliang19910310 http://bbs.elecfans.com/jishu_789211_1_1.html

NS2 是指 Network Simulator version 2，NS(Network Simulator) 是一种针对网络技术的源代码公开的、免费的软件模拟平台，研究人员使用它可以很容易的进行网络技术的开发，而且发展到今天，它所包含的模块几乎涉及到了网络技术的所有方面。所以，NS 成了目前学术界广泛使用的一种网络模拟软件。此外，NS 也可作为一种辅助教学的工具，已被广泛应用在了网络技术的教学方面。因此，目前在学术界和教育界，有大量的人正在使用或试图使用 NS。

安装 NS-2 的过程如下：

1 首先先下载 NS-2 软件，下载软件安装包 ns-allinone-2.35.tar.gz；

2 在联网的情形下比较好，然后要做的是解压安装包 ns-allinone-2.35.tar.gz，在 root 下执行命令：

```
# tar xvfz ns-allinone-2.35.tar.gz
```

在解压完成之后执行第 3 步；

3 然后安装 NS-2 软件，即进入 ns-allinone-2.35 文件夹，进入文件夹命令为：

```
# cd ns-allinone-2.35
```

之后在执行安装命令：

```
# ./install
```

安装时间还挺长的，希望大家有耐心。

当然一定安装过程中会有问题的，一般会有如下几个问题，当然我也是从微博中收集到的：

出现问题：

1.make: *** [tk3d.o] 错误 1

tk8.5.10 make failed! Exiting ...

安装 libx11-dev 可以解决问题

sudo apt-get install libx11-dev

2.can't find X includes

otcl-1.14 configuration failed! Exiting ...

安装 libXt-dev 可以解决问题

sudo apt-get install libXt-dev

3.make: g++：命令未找到

make: *** [tcl2c++] 错误 127

tclcl-1.20 make failed! Exiting ...

先输入 g++看 g++有没有被安装，原来我的 g++还没安装好如果没由安装好则

sudo apt-get install g++

如果出现问题最好是联网，如果不联网安装的话，那会有点儿麻烦，因为依赖的文件太多。

关于 ns-2 软件安装有很多资料，其中大多数安装过程的最后都有环境变量的设置。

最后安装好设置环境变量　　在/home/user 下编辑 .bashrc

```
export NS_HOME=/home/chenkai/ns/ns-allinone-2.35
export
PATH=$NS_HOME/nam-1.15:$NS_HOME/tcl8.5.10/unix:$NS_HOME/tk8.5.10/unix:$NS_HOME/bin:$PATH
export
LD_LIBRARY_PATH=$NS_HOME/tcl8.5.10/unix:$NS_HOME/tk8.5.10/unix:$NS_HOME/otcl-1.14:$NS_HOME/
lib:$LD_LIBRARY_PATH
export TCL_LIBRARY=$NS_HOME/tcl8.5.10/library
```

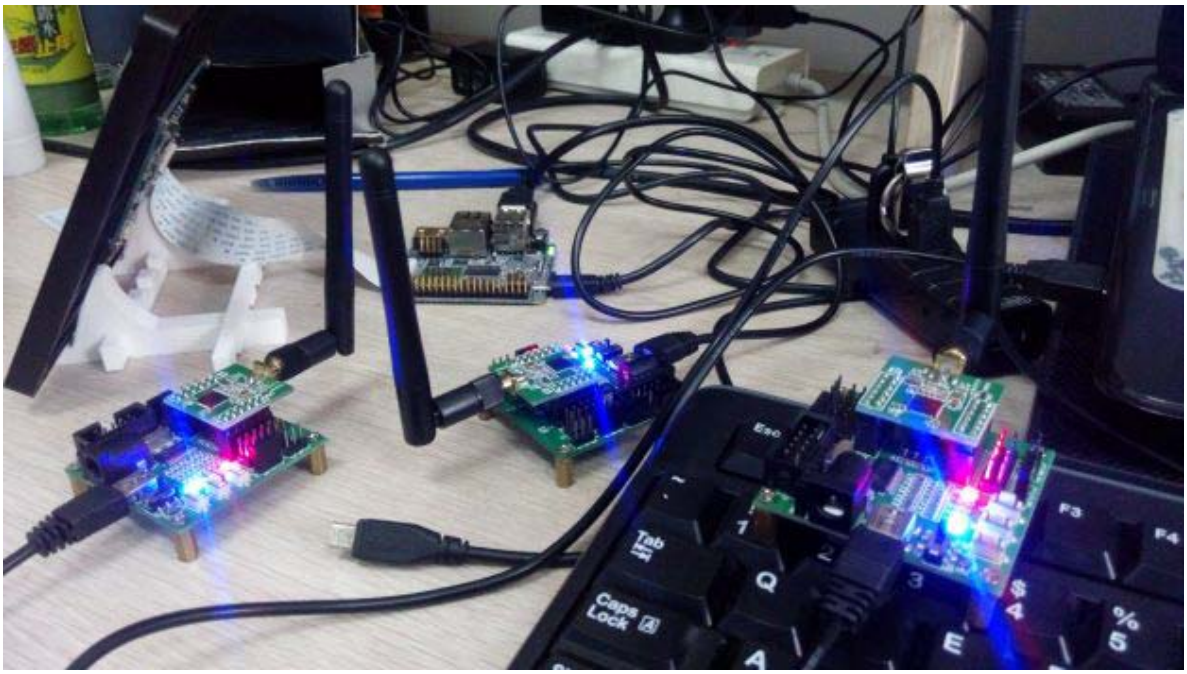
输入 source .bashrc　　使环境变量有效。
最后验证 ns。
在控制台输入 ns
%
显示安装成功。

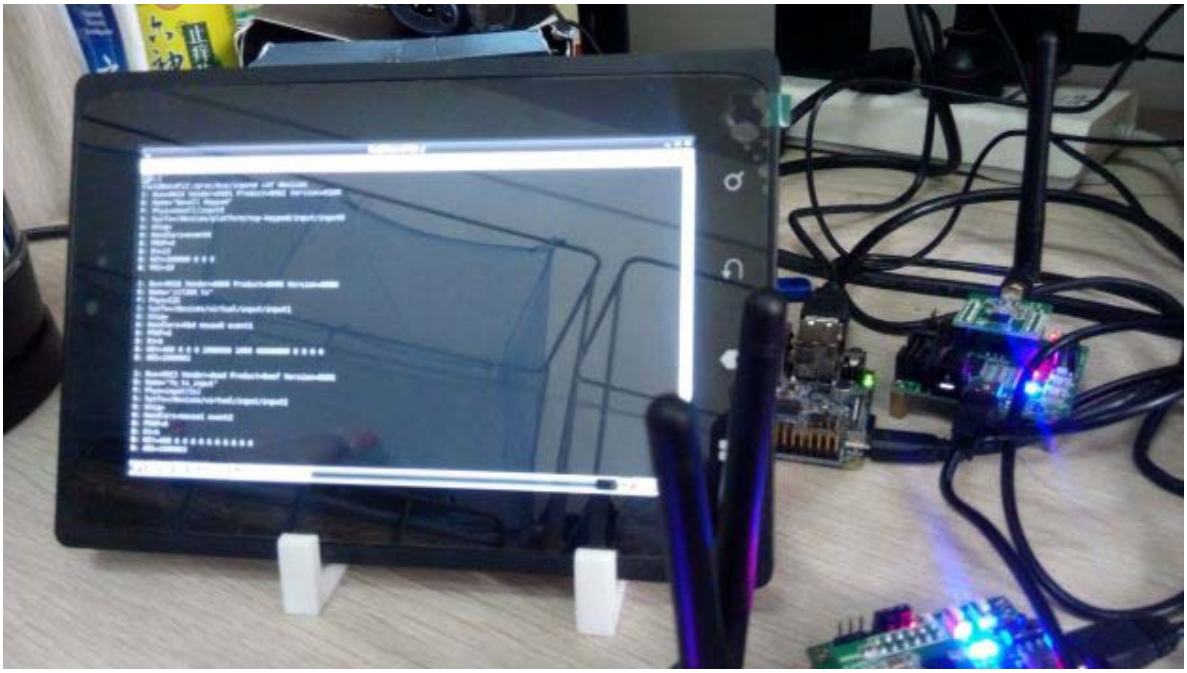
技巧八：NanoPi M2 + CC2530 的菜鸟开发

来自：liliang19910310 http://bbs.elecfans.com/jishu_866096_1_1.html

最近在做一个无线传感器网络方面的项目，采用的方案是 ZigBee 技术，基于硬件平台为 CC2530，最初并没有想加入终端处理机，正好想到可以使用 NanoPi M2 作为终端机，但是由于本身是菜鸟又是初次，所以方案很青涩，欢迎大家来多提方案（在这里先谢谢大家了）。由于在做原型机，所以只有 3 个 ZigBee 节点+NanoPi M2。

啥都不说了，先上图吧！



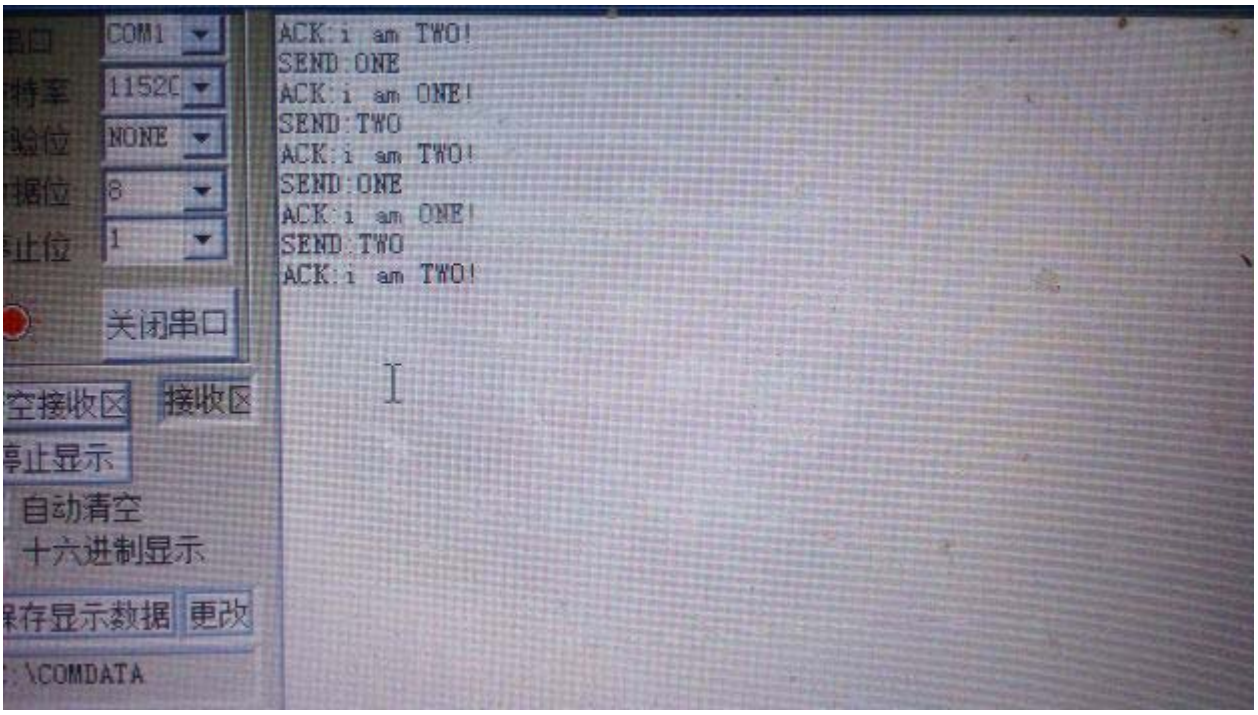


关于 ZigBee 协议栈的解读在这里就不多做赘述，当然 TI 的 ZigBee 协议栈 ZStack 的 NWK 层和 MAC 层都是不开源的，所以协议栈真正开发的地方为 HAL 层和 APL 层，为什么国内没有成熟的协议栈呢，值得思考（顺口一提而已，不必当真。）

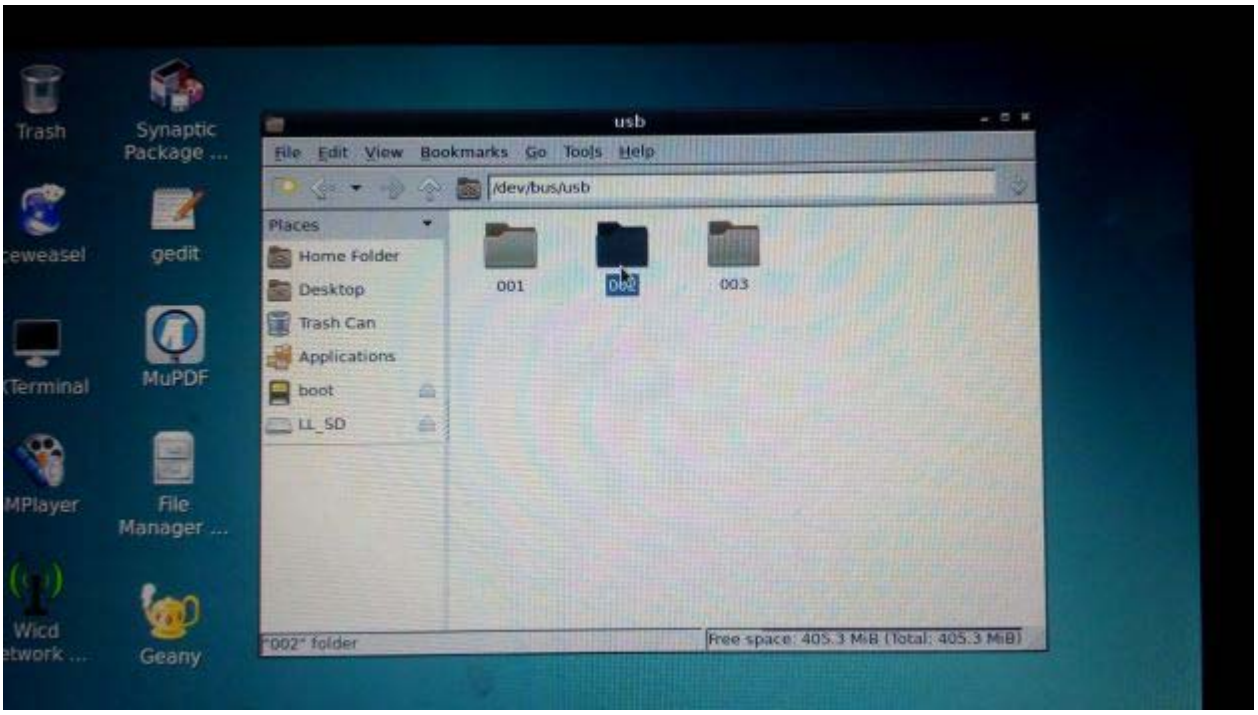
首先就是要找一个在 Linux 下的通信串口助手，关于这个，看过一个博客写的挺好的，大家可以参考一下：

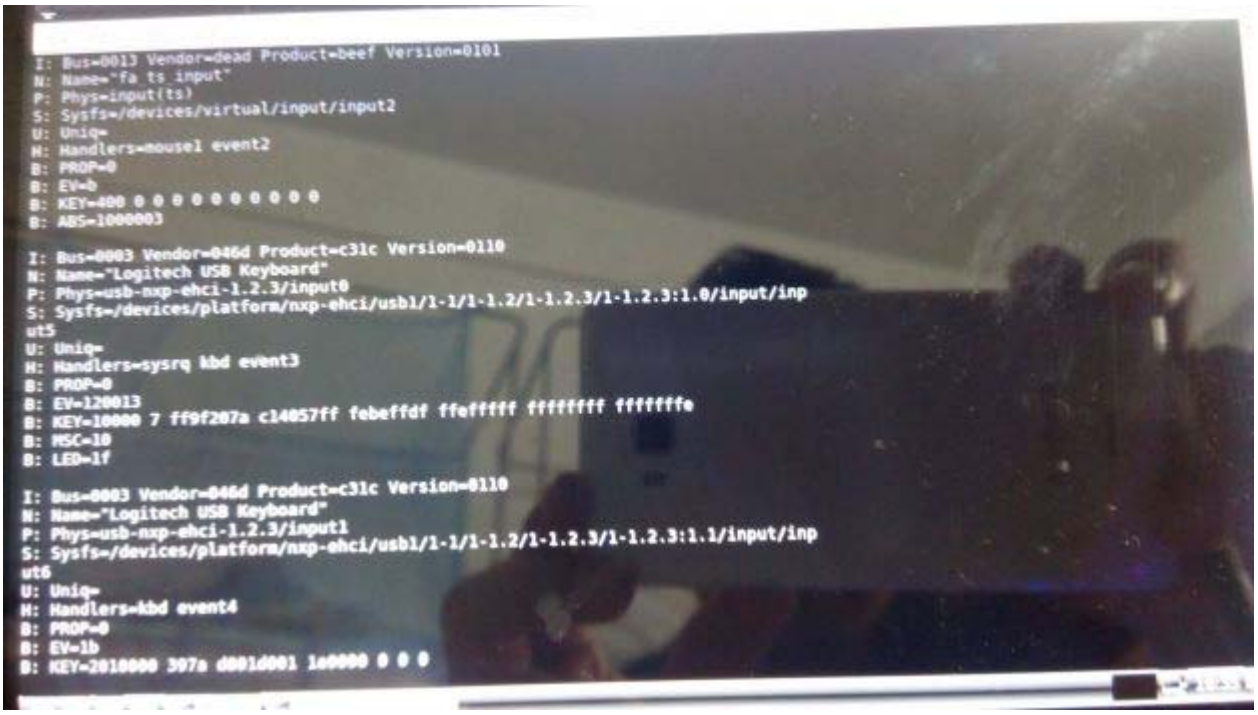
http://www.cnblogs.com/sky1991/archive/2012/11/04/2753481.html

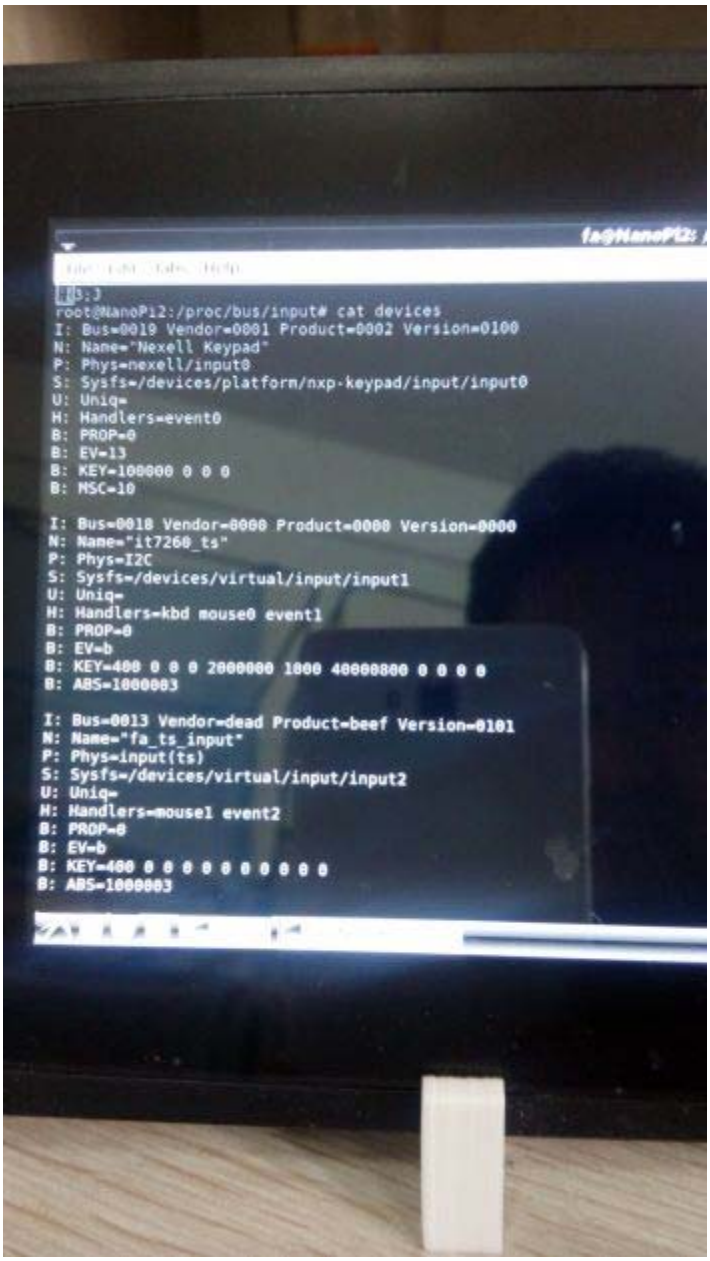
第二，安装好通信串口助手之后，就可以看到实验结果，三个 ZigBee 节点，一个协调器，两个终端节点。



三个节点都通过 USB 连接在 M2 模块上，下面是 USB 设备信息




I: Bus=0013 Vendor=dead Product=beef Version=0101
N: Name="fa ts input"
P: Phys=input(ts)
S: Sysfs=/devices/virtual/input/input2
U: Uniq=
H: Handlers=mouse1 event2
B: PROP=0
B: EV=b
B: KEY=400 0 0 0 0 0 0 0 0 0
B: ABS=1000003
I: Bus=0003 Vendor=046d Product=c31c Version=0110
N: Name="Logitech USB Keyboard"
P: Phys=usb-nxp-ehci-1.2.3/input0
S: Sysfs=/devices/platform/nxp-ehci/usb1/1-1/1-1.2/1-1.2.3/1-1.2.3:1.0/input/inp
ut5
U: Uniq=
H: Handlers=sysrq kbd event3
B: PROP=0
B: EV=120013
B: KEY=10000 7 ff9f207a c14057ff febeffdf ffefffff ffffffff fffffffe
B: MSC=10
B: LED=1f
I: Bus=0003 Vendor=046d Product=c31c Version=0110
N: Name="Logitech USB Keyboard"
P: Phys=usb-nxp-ehci-1.2.3/input1
S: Sysfs=/devices/platform/nxp-ehci/usb1/1-1/1-1.2/1-1.2.3/1-1.2.3:1.1/input/inp
ut6
U: Uniq=
H: Handlers=kbd event4
B: PROP=0
B: EV=1b
B: KEY=2010000 307a d001d001 1e0000 0 0 0



就是一次尝试，仅仅搭起了一个框架，也有许多改进和扩展的地方，所以希望大家能够多提建议和批评，能够使我得到提升，这也是我发帖的初衷（有点儿装了）。谢谢！

技巧九：NanoPI M2 GPIO 篇——基于 Python3

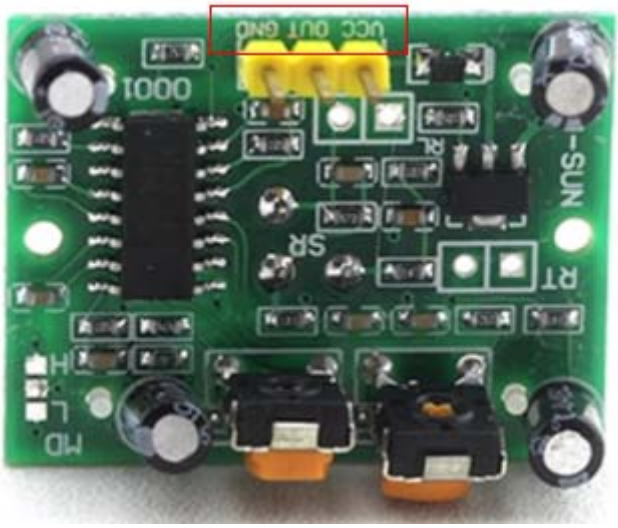
来自：逃出生天 http://bbs.elecfans.com/jishu_705328_1_1.html

NanoPI M2 的 GPIO 控制使用普通的配置寄存器的方法太麻烦了，需要查看寄存器表，写很多驱动函数！但是使用 python 的第三方库 gpio 控制 GPIO 却很简单，只需要通过简单的几行代码就可以控制 GPIO 的输入输出。下面以一个人体红外感应模块检测的问题介绍 gpio 库的使用。

一. 所需材料

1. 面包板 x1
2. 杜邦线 Xn
3. LED 灯 x1
4. 电阻 1kΩx1
5. 人体红外感应模块 x1

人体红外感应模块是一个人体红外线检测传感器，当检测到人体红外线时，输出端便会输出一个高电平，引脚定义如下所示：



二. 环境搭建

打开终端执行

```
1.  sudo apt-get update
2.  sudo apt-get install python3-pip
3.  sudo pip3 install gpio
```

复制代码

三.gpio 库介绍

项目地址：https://github.com/vitiral/gpio

gpio 库是一个第三方库，支持 Python2 和 Python3,使用标准的 linux sysfs 接口控制 GPIO 的库。也就是说，只要明确 GPIO 的存器就可以控制 gpio，其他工作库替你完成了细节。

支持的主要功能函数：

```
input(pin) or read(pin)  #读取引脚的电平状态
set(pin, value)          #设置引脚电平
mode(pin)                #获取引脚模式
setup(pin, mode)         #设置引脚模式，输入='in'，输出='out'
cleanup(pin)             #清除引脚电平
```

复制代码

更多的函数使用 help(gpio)查看,注意程序退出时要调用 cleanup(pin)函数清除引脚电平,否则下次使用时电平任然为上次使用时的状态。

要使用 gpio 库控制 NanoPI M2 的 GPIO，得知道寄存器对应的引脚编号，参考前面的 nanopi2 的帖子，知道了计算的方法为：**引脚编号 = 控制引脚的寄存器基数 + 控制引脚寄存器位数**；

比如下面将使用的 62 号[GPIOB30]和 63 号[GPIOB31]引脚

引脚编号 = 控制引脚的寄存器基数 + 控制引脚寄存器位数

62 = 32x1 + 30

63 = 32x1 + 31

依次类推，其中引脚图上的 A,B,C,D 分别与 0，1，2，3 对应，比如 GPIOC28 在 gpio 库中的编号就是

32x2+28=92

经过上面的计算我自己整理了如下所示的 GPIO 对应图【大家可收藏方便查看】：

NanoPi M2 Python gpio对应管脚图					
编号	Name	Pin#	Pin#	Name	编号
	SYS_3.3V	1	2	VDD_5V	
	I2C0_SDA	3	4	VDD_5V	
	I2C0_SCL	5	6	DGND	
104	GPIOD8/PPM	7	8	UART3_TXD/GPIOD21	117
	DGND	9	10	UART3_RXD/GPIOD17	113
61	UART4_TX/GPIOB29	11	12	GPIOD1/PWM0	97
62	GPIOB30	13	14	DGND	
63	GPIOB31	15	16	GPIOC14/PWM2	78
	SYS_3.3V	17	18	GPIOB27	59
95	SPI0_MOSI/GPIOC31	19	20	DGND	
96	SPI0_MISO/GPIOD0	21	22	UART4_RX/GPIOB28	60
93	SPI0_CLK/GPIOC29	23	24	SPI0_CS/GPIOC30	94
	DGND	25	26	GPIOB26	58
	I2C1_SDA	27	28	I2C1_SCL	
72	GPIOC8	29	30	DGND	
71	GPIOC7	31	32	GPIOC28	92
77	GPIOC13/PWM1	33	34	DGND	
75	SPI2_MISO/GPIOC11	35	36	SPI2_CS/GPIOC10	74
35	AliveGPIO3	37	38	SPI2_MOSI/GPIOC12	76
	DGND	39	40	SPI2_CLK/GPIOC9	73

本次使用 3.3v 供电，小灯要接一个 1kΩ 的电阻，62 号[GPIOB30]作为输出控制小灯，63 号[GPIOB31]做为输入，接人体红外感应模块的 OUT 引脚，当检测到一个信号时，63 号[GPIOB31]便会读到一个高电平，于是触发小灯亮。这就实现了一个输入输出的控制。

四. 具体代码实现

```
#!/usr/bin/env python3
# -*- coding:UTF-8 -*-

import gpio
import time
#设置 62 为输出，63 为输入
gpio.setup(62,'out')
gpio.setup(63,'in')
```

```

gpio.setwarnings(False)#去除警告信息
print("NanoPi M2 Module Test!(Ctrl+C to exit)")
Current_State = 0 #当前状态
Previous_State = 0 #上一次的状态
try:
        print("Waiting for NanoPi M2 to settle....")
        while gpio.input(63) == 1:
                Current_State = 0
        print("Read")
        while True:
                Current_State = gpio.input(63)#检测状态
                if Current_State==1 and Previous_State==0:
                        gpio.set(62,1)#小灯亮
                        print("Motion detected!")
                        time.sleep(3)
                        gpio.set(62,0)#小灯灭
                        Previous_State = 1
                elif Current_State==0 and Previous_State==1:
                        print("Read!")
                        Previous_State = 0
                time.sleep(0.1)

except KeyboardInterrupt:
        print("Quit!")
        gpio.cleanup(62)
        gpio.cleanup(63)

```

复制代码

五. 测试效果如下所示

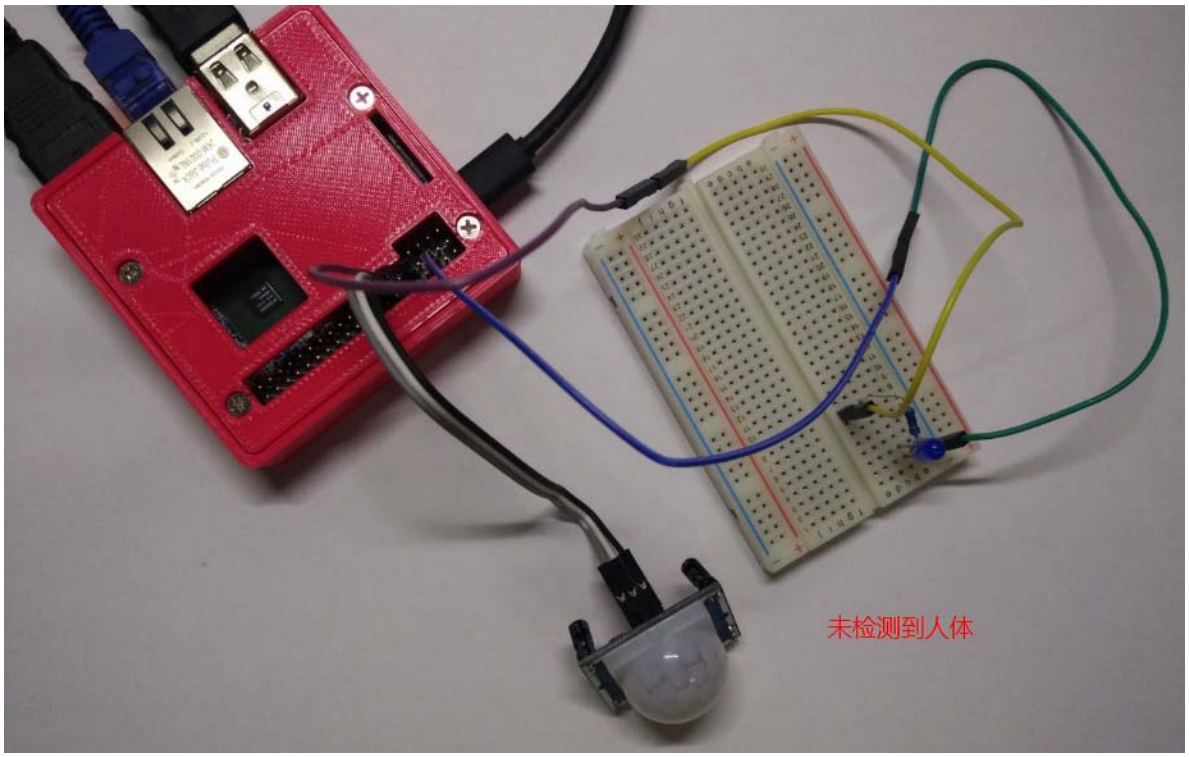


图 1: 未检测到时

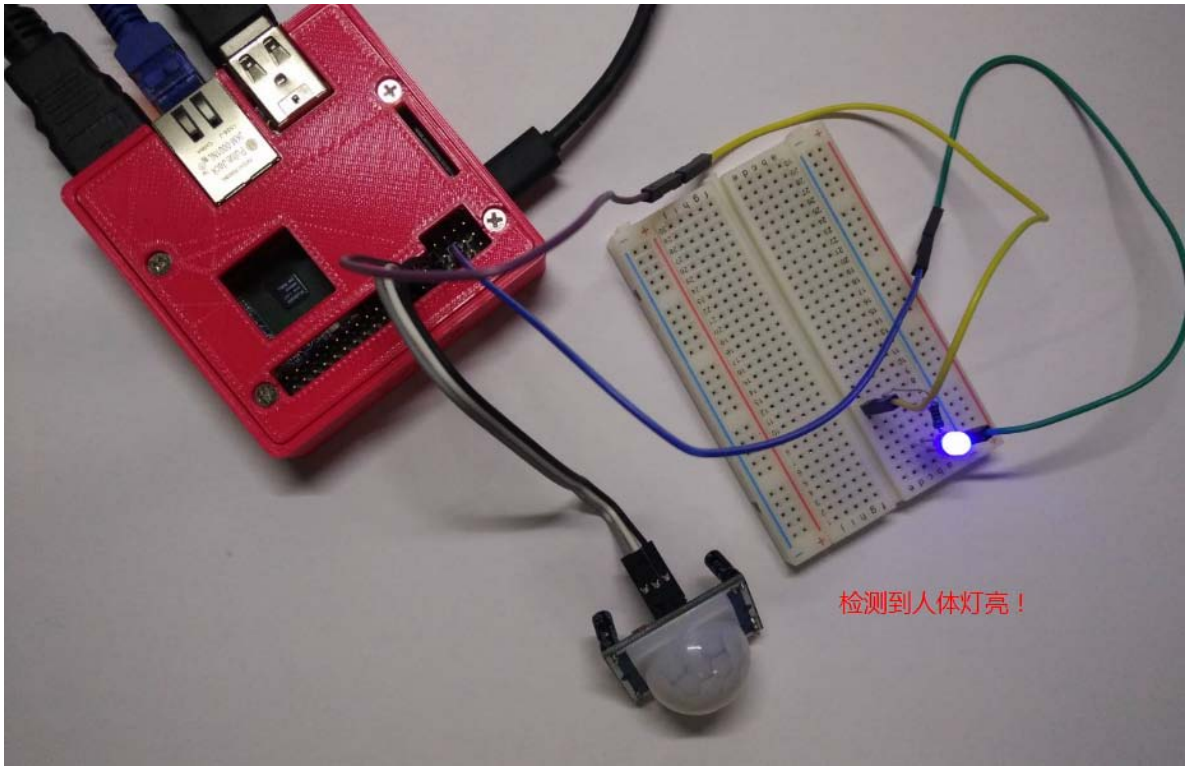

检测到人体灯亮！

图 2:检测到时小灯亮

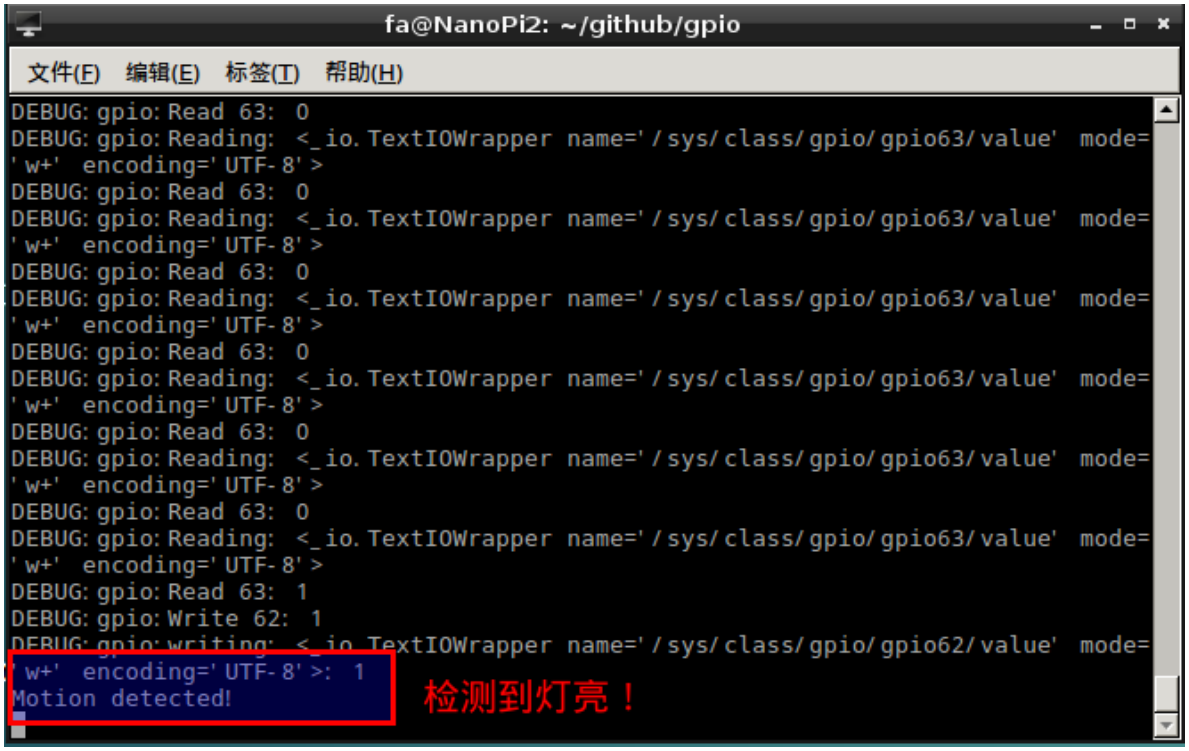

fa@NanoPi2: ~/github/gpio
文件(F) 编辑(E) 标签(T) 帮助(H)
DEBUG: gpio: Read 63: 0
DEBUG: gpio: Reading: <_io.TextIOWrapper name='/sys/class/gpio/gpio63/value' mode='w+' encoding='UTF-8'>
DEBUG: gpio: Read 63: 0
DEBUG: gpio: Reading: <_io.TextIOWrapper name='/sys/class/gpio/gpio63/value' mode='w+' encoding='UTF-8'>
DEBUG: gpio: Read 63: 0
DEBUG: gpio: Reading: <_io.TextIOWrapper name='/sys/class/gpio/gpio63/value' mode='w+' encoding='UTF-8'>
DEBUG: gpio: Read 63: 0
DEBUG: gpio: Reading: <_io.TextIOWrapper name='/sys/class/gpio/gpio63/value' mode='w+' encoding='UTF-8'>
DEBUG: gpio: Read 63: 0
DEBUG: gpio: Reading: <_io.TextIOWrapper name='/sys/class/gpio/gpio63/value' mode='w+' encoding='UTF-8'>
DEBUG: gpio: Read 63: 0
DEBUG: gpio: Reading: <_io.TextIOWrapper name='/sys/class/gpio/gpio63/value' mode='w+' encoding='UTF-8'>
DEBUG: gpio: Read 63: 1
DEBUG: gpio: Write 62: 1
DEBUG: gpio: writing: <_io.TextIOWrapper name='/sys/class/gpio/gpio62/value' mode='w+' encoding='UTF-8'>: 1
Motion detected!
检测到灯亮！

图 3:终端运行

技巧十：智能车多功能遥控器开发项目结题报告

来自：辉哥的歌 http://bbs.elecfans.com/jishu_866078_1_1.html

在第 7 讲“我的‘潘多拉魔盒’7—遥控器按键电平检测电路设计中，重新改进了遥控器的电路设计，解决了按键未按下时，对应引脚悬空导致的电平跳变问题。从而为本项目的结题做了关键的铺垫。

为了能够实现所开发的遥控器对小车的遥控，本节需要解决的问题有：

1） NanoPi M2 与 Ardunio 的无线通信问题；

2） 小车的控制流程设计及程序的开发；

3） 小车驱动电机控制。

所采用的硬件如下表所示：

序号	名称	数量	作用
1	NanoPi M2	1	遥控器控制根据控制操作形成控制指令
2	面包板遥控器	1	用于遥控操作

3	3DR 无线数传	1 对	发送/接收遥控指令
4	Ardunio	1	小车主控制器
5	L298N	1	小车直流电机驱动器
6	9V 方电池	1	给 Ardunio 供电
7	3s 锂电池	1	给小车驱动电机供电
8	6V UBEC	1	将 3s 电池电压降到 6V 给小车 L298N 供电
9	小车本体	1	承载

由于本项目的重点在于小车遥控器的开发，对于小车的建立和控制原理本报告就不细讲，感兴趣的同学，有问题可以跟我直接联系。

2. 查看 NanoPi M2 可用串口号

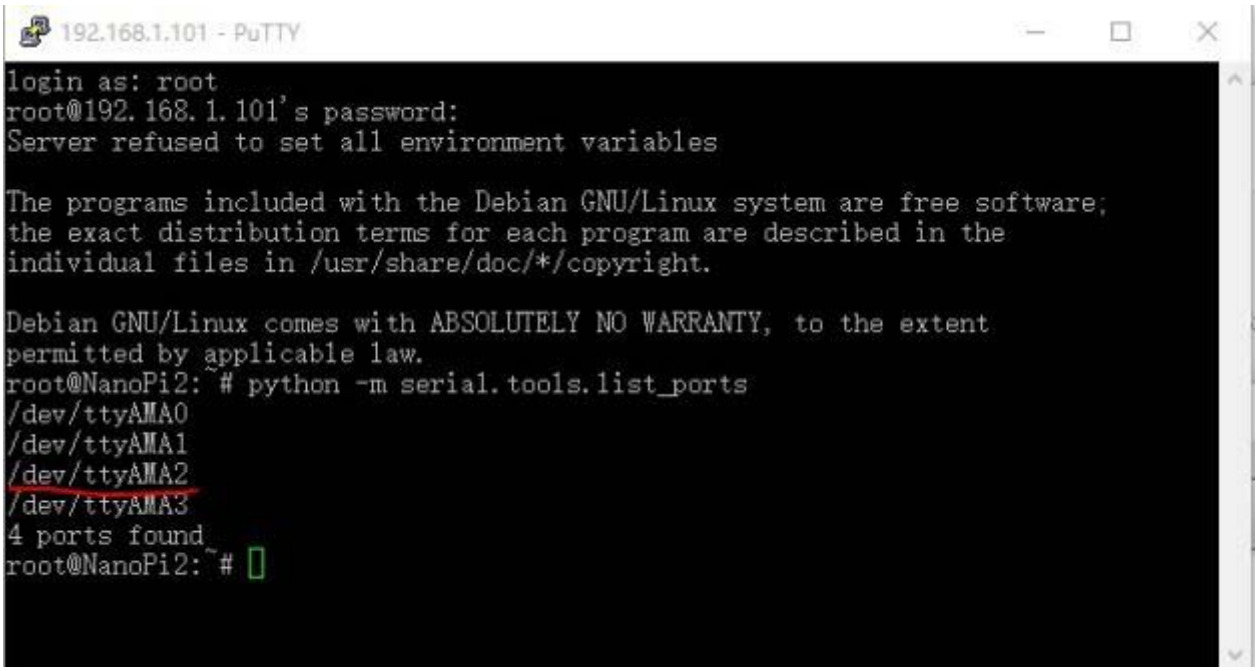


图 1 查看 NanoPi M2 可用的串口号

由图 1 可见，当前有四个串口可用，其中画红线的 ttyAMA2 就是 UART3 对应的串口，也即引脚 8 和引脚 10。而其中一个数传是 USB 接口（见图 2），所以可直接插在 USB 口上，此时再次查询可用串口发现多了一个 ttyUSB0 串口，也就是我所采用的插着无线数传的接口。

图 2 无线数传模块

```
root@NanoPi2: # python -m serial.tools.list_ports
/dev/ttyAMA0
/dev/ttyAMA1
/dev/ttyAMA2
/dev/ttyAMA3
/dev/ttyUSB0
5 ports found
root@NanoPi2:~#
```

图 3 插上带 USB 接口的无线数传后出现 ttyUSB 串口号

3. NanoPiM2 与 Ardunio 通信测试

（1） 开发环境及通信库安装

遥控器与 NanoPi M2 的接线在第 5 和第 7 讲中都有详细描述，这里不再赘述，可直接见图 4。主要讲遥控器端的控制程序的软件实现。

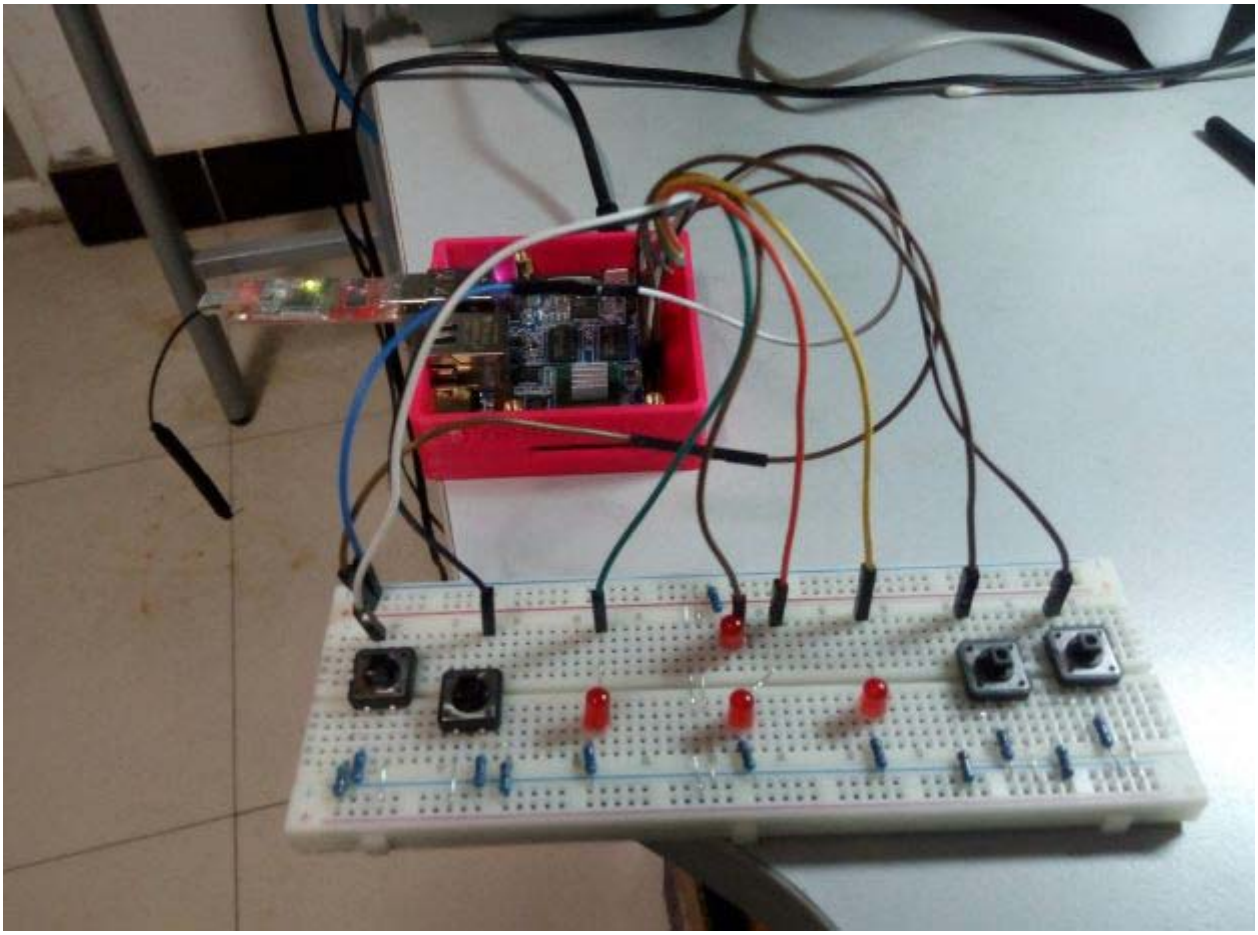


图 4 遥控器端连接实物图

这里采用的开发环境是 Python2.7 的 IDE，采用这个的开发环境的好处是格式简约大方，调试方便(见图 5 和图 6)，安装也很简单，采用如下指令：

安装 IDE 命令：sudoapt-get install idle

采用的开发语言当然也就是 python 了，之所以采用 python 主要是 python 的串口通信库很好用，也就是 pyserial-3.0.1。不过，该库需要安装，安装方法如下：

使用 apt-get 工具安装：sudoapt-get install python-serial

当然也可以下载后安装，下载后安装方法如下：

1） tar zxvf pyserial-3.0.1.tar.gz

2） cd pyserial-3.0.1

3） python setup.py install

pyserial 的下载地址如下，该库有详细的使用教程，虽然是英文的但简单易懂：

下载地址：https://pypi.python.org/pypi/pyserial/3.0.1

通信库使用手册连接：http://pythonhosted.org/pyserial/

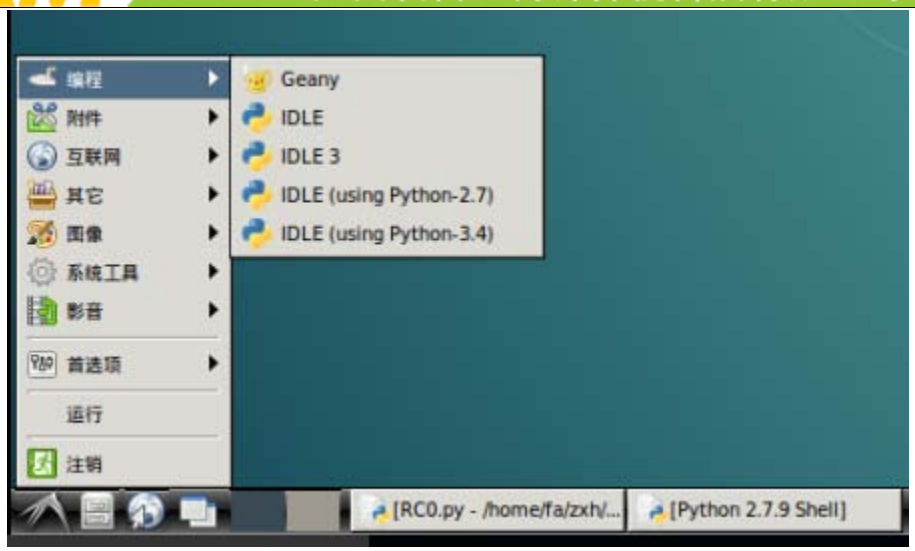


图 5 Python IDLE 开发环境安装后在编程菜单下

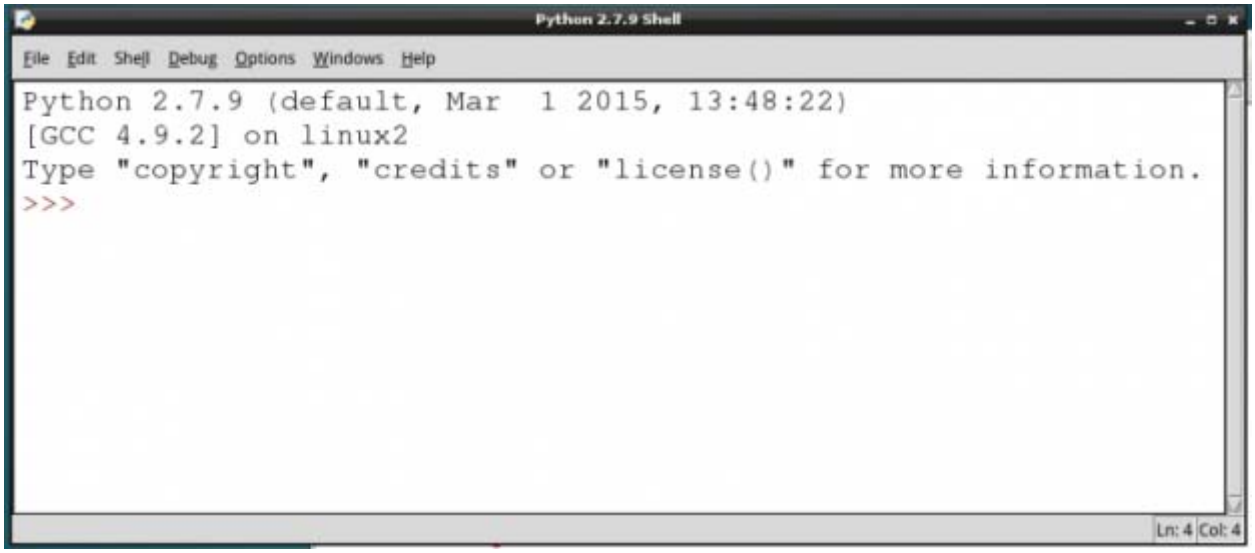


图 6 打开 python2.7 后进入开发环境界面

（2）　　**遥控器控制流程**

1）　导入串口通信库：importserial

2）　打开通信的串口及设置波特率：ser = serial.Serial('/dev/ttyUSB0',57600)

3）　判断是否打开正确：if(ser.isOpen()):print('Device open successfully!')

4）　定义所采用的引脚编码号，如 forward_in= 92

5）　定义引脚的输入输出模式，如 gpio.setup(forward_in,'in')

6）　初始化输出引脚为低电平，如 gpio.set(forward_out,0)

7）　判断按下的是哪个按键，点亮对应的 led 灯，并发送对应的控制指令，这里程序设计时滤除了多按键同时按下的情况，因为原则上是不允许的，比如说若同时按下前进和后退键，则认为是操作不当，不予反应。控制主体程序如下：

```
while True:
        #time.sleep(0.000001)
        Fin =gpio.input(forward_in)
```

```
Rin =gpio.input(right_in)
Lin = gpio.input(left_in)
Bin = gpio.input(back_in)
if((Fin==1) and (Rin==0)and (Lin==0) and (Bin==0)):
                gpio.set(forward_out,1)
                gpio.set(right_out,0)
                gpio.set(left_out,0)
                gpio.set(back_out,0)
                print("forward_in = " + str(Fin))
                 command =ser.write('F')    # 发送前进指令
elif((Fin==0) and(Rin==1) and (Lin==0) and (Bin==0)):
                 gpio.set(forward_out,0)
                gpio.set(right_out,1)
                gpio.set(left_out,0)
                gpio.set(back_out,0)
                print("right_in = " + str(Rin))
                 command = ser.write('R')    # 发送右转指令
elif((Fin==0) and(Rin==0) and (Lin==1) and (Bin==0)):
                gpio.set(forward_out,0)
                gpio.set(right_out,0)
                 gpio.set(left_out,1)
                gpio.set(back_out,0)
                print("left_in = " + str(Lin))
                 command =ser.write('L')    # 发送左转指令
elif((Fin==0) and(Rin==0) and (Lin==0) and (Bin==1)):
                gpio.set(forward_out,0)
                gpio.set(right_out,0)
                gpio.set(left_out,0)
                 gpio.set(back_out,1)
                print("back_in = " + str(Bin))
                 command =ser.write('B')    # 发送后退指令
else:
                gpio.set(forward_out,0)
                gpio.set(right_out,0)
                gpio.set(left_out,0)
                gpio.set(back_out,0)
                print("others operation" )
                 command =ser.write('P') # 发送无效操作指令
```

8） 强制停止设置：exceptKeyboardInterrupt: print("Qiut")

（3） 按键操作效果如下：

在进行通信之前，先进行了单遥控器的控制操作测试，已检验控制操作逻辑的正确性。从各个按键下的电平情况（图 7-11）来看，没有出现电平跳动情况，控制操作响应较快，也比较稳定。

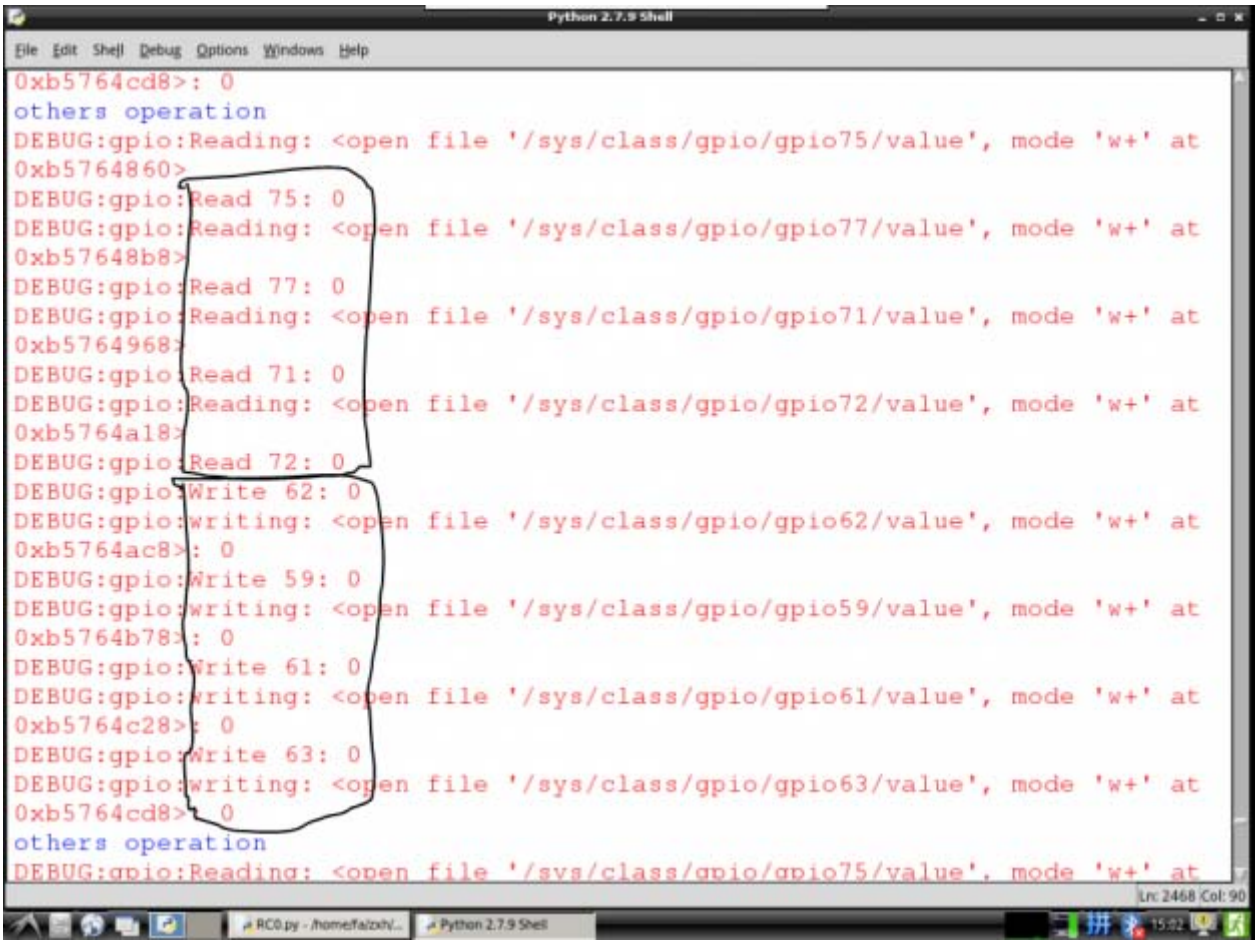


图 7 无控状态下各引脚电平都为零

图 8　　按下前进按键引脚电平情况

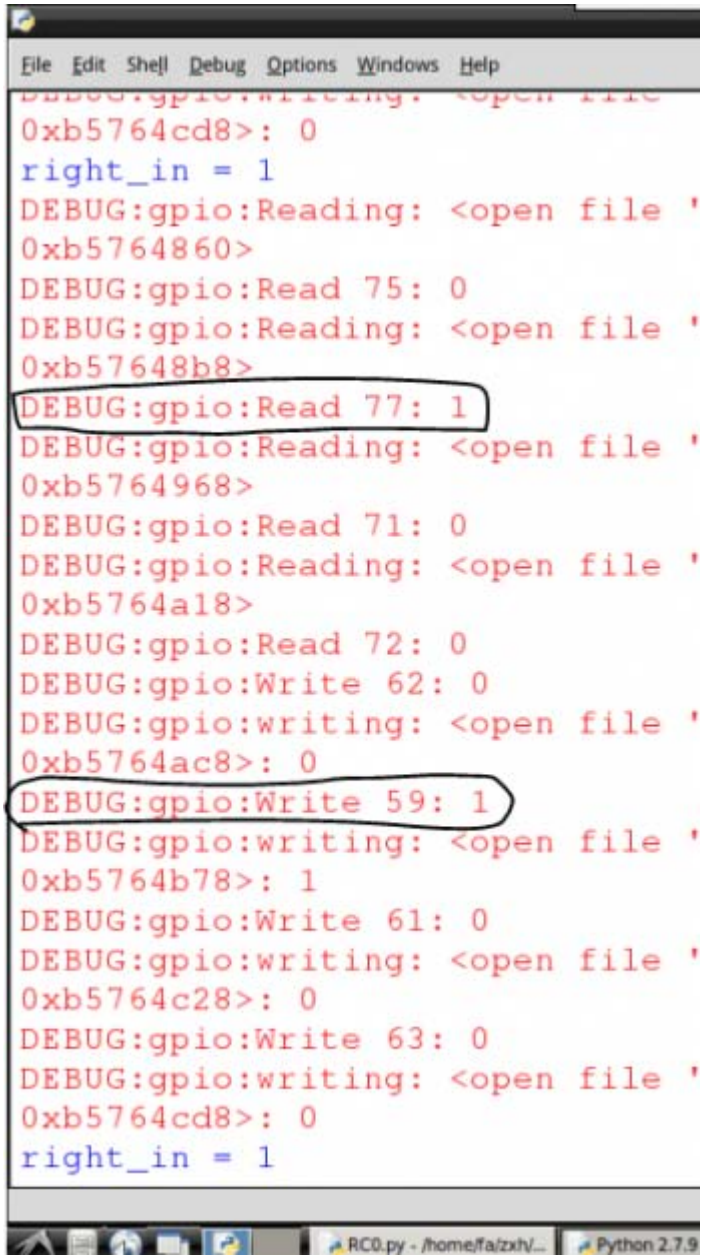


图 9 按下右转按键引脚电平情况

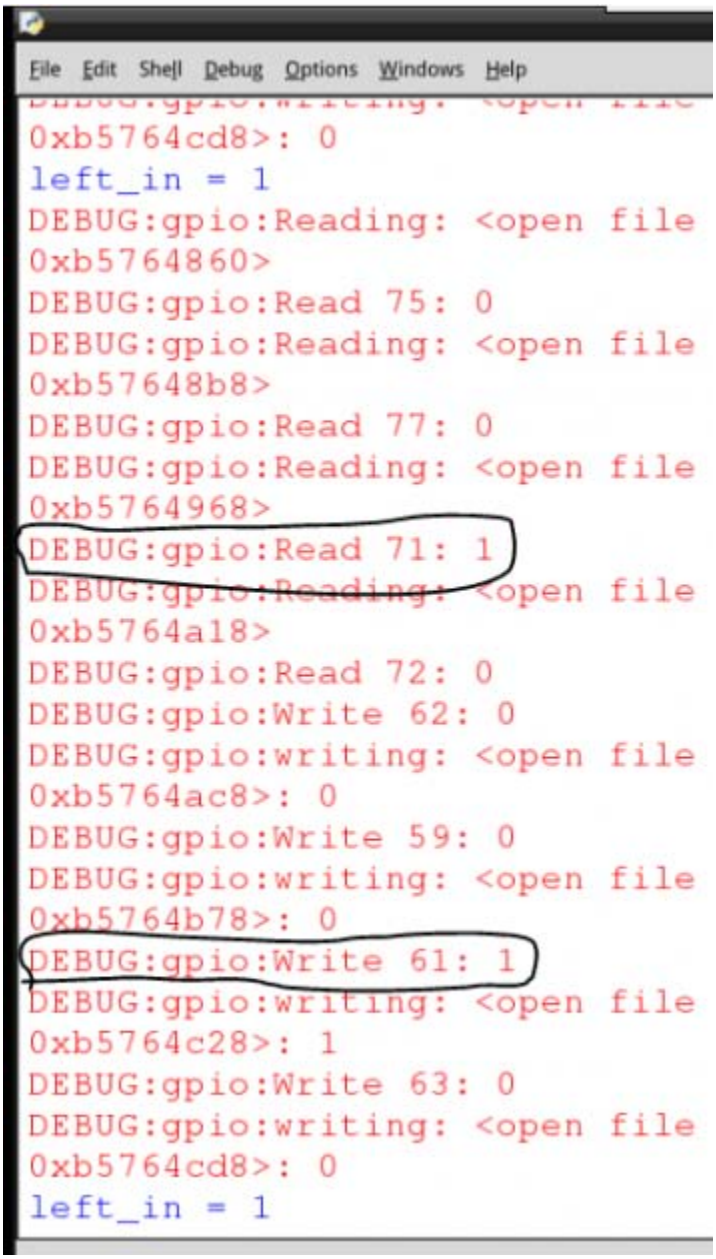


图 10　　按下左转按键引脚电平情况

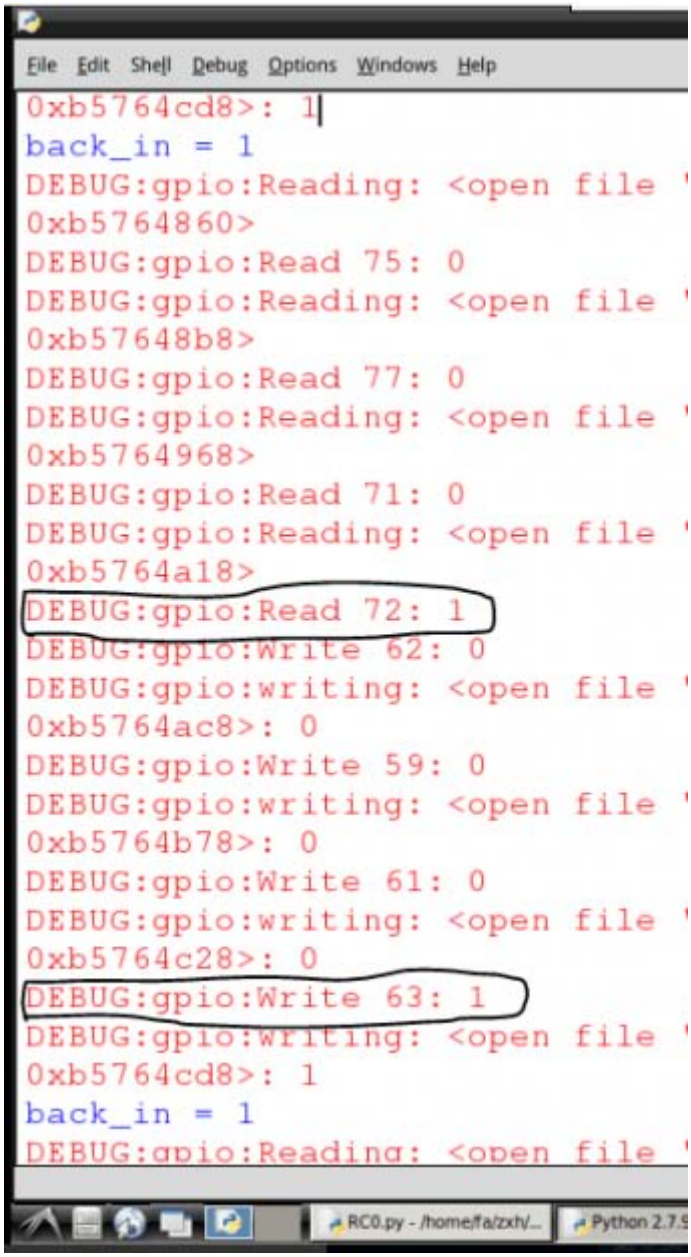


图 11　　按下后退按键引脚电平情况

（4）　　**接收端测试**

接收端采用的是 Arduino 的 IDE 开发环境，进行程序的开发，首先进行接收 NanoPi M2 发出的指令测试。接收端的测试程序如图 12 所示。

图 12 Ardunio 接收端界面及测试程序

将该程序下载到 Ardunio 板后，连接 Ardunio 与另一端无线数传模块（见图 13）后，打开 Ardunio 的串口监视窗口，然后进行按键操作，可见效果如图 14-16 所示。

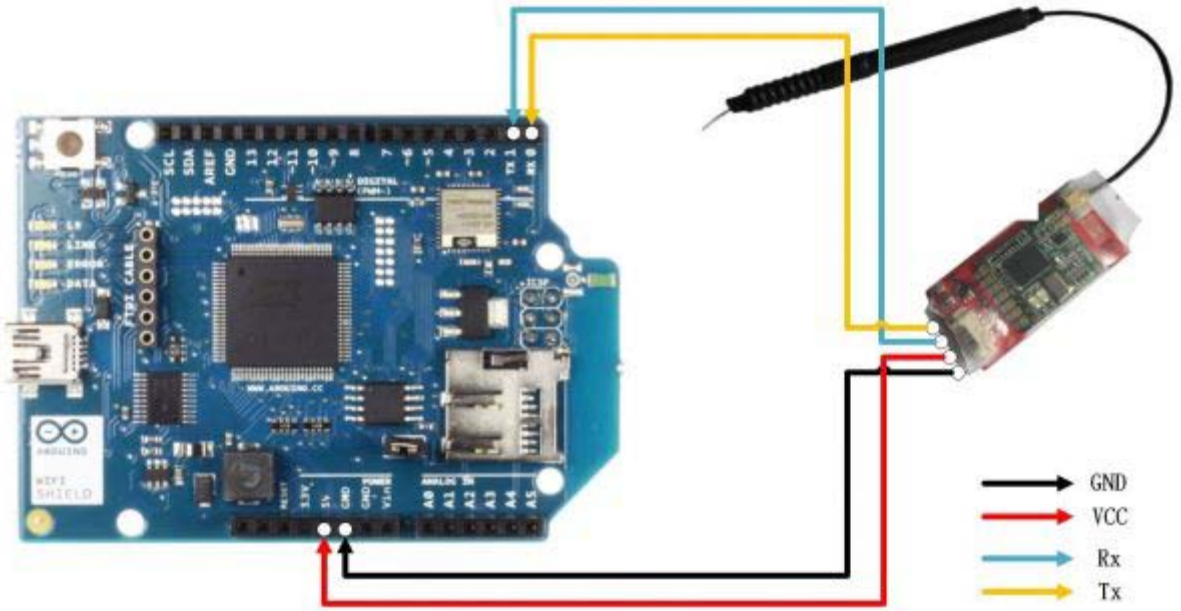


图 13　　无线数传接收端与 Ardunio 连接示意图

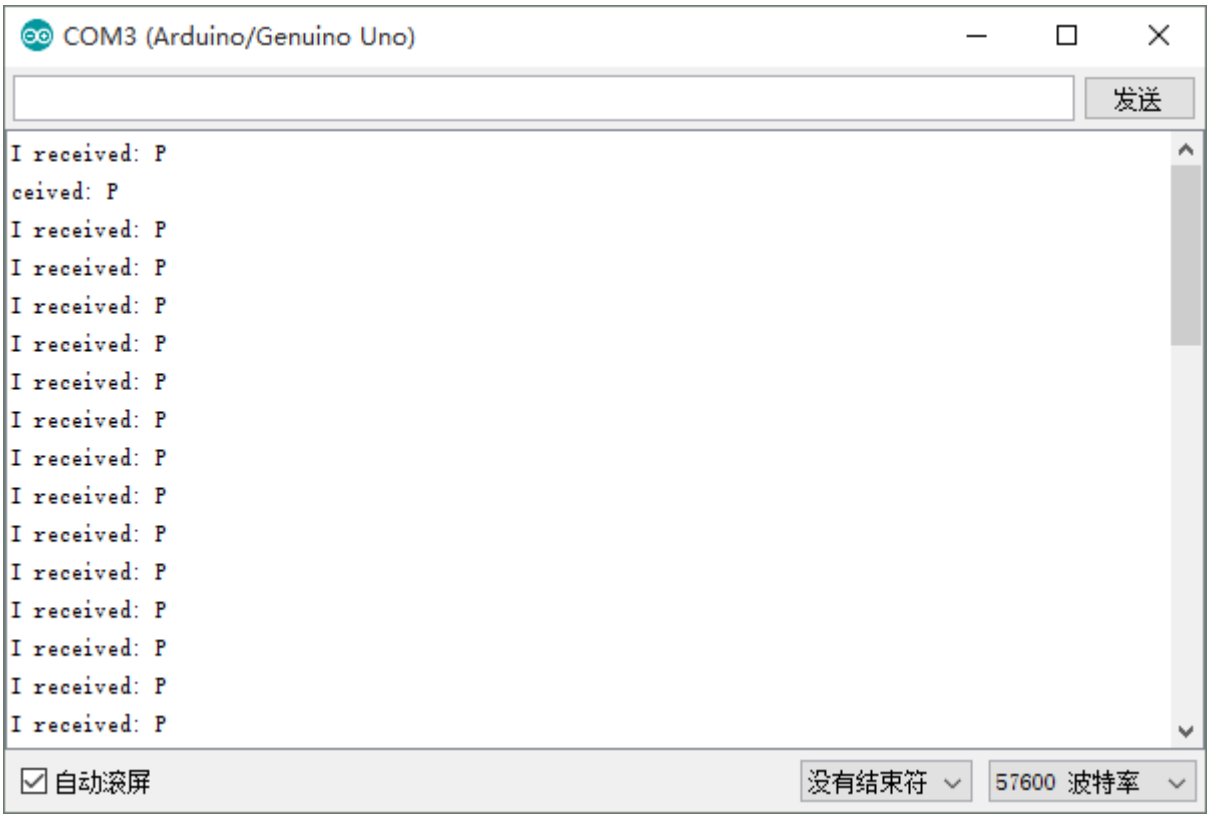


图 14　　无操作或多按键同时按下时接收到的是停车指令 P

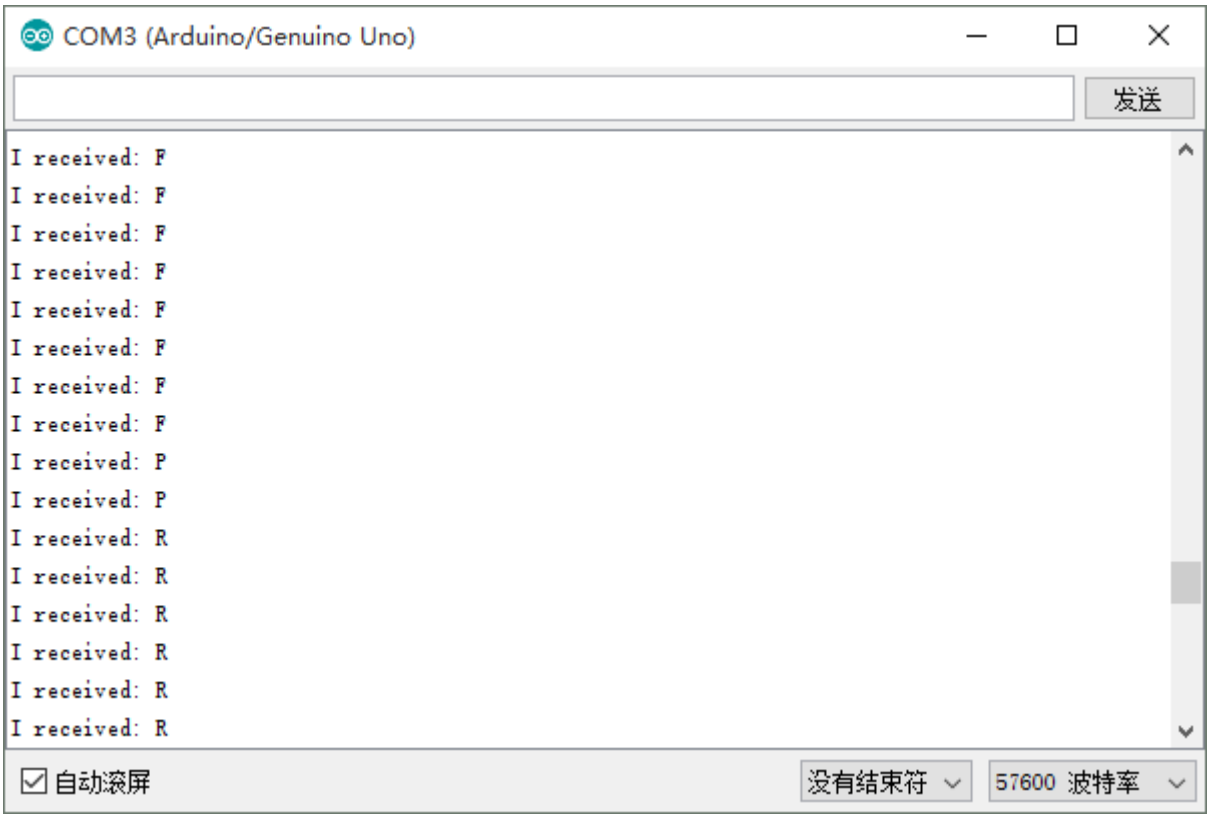


图 15　　接收到 F 前进和 R 右转指令

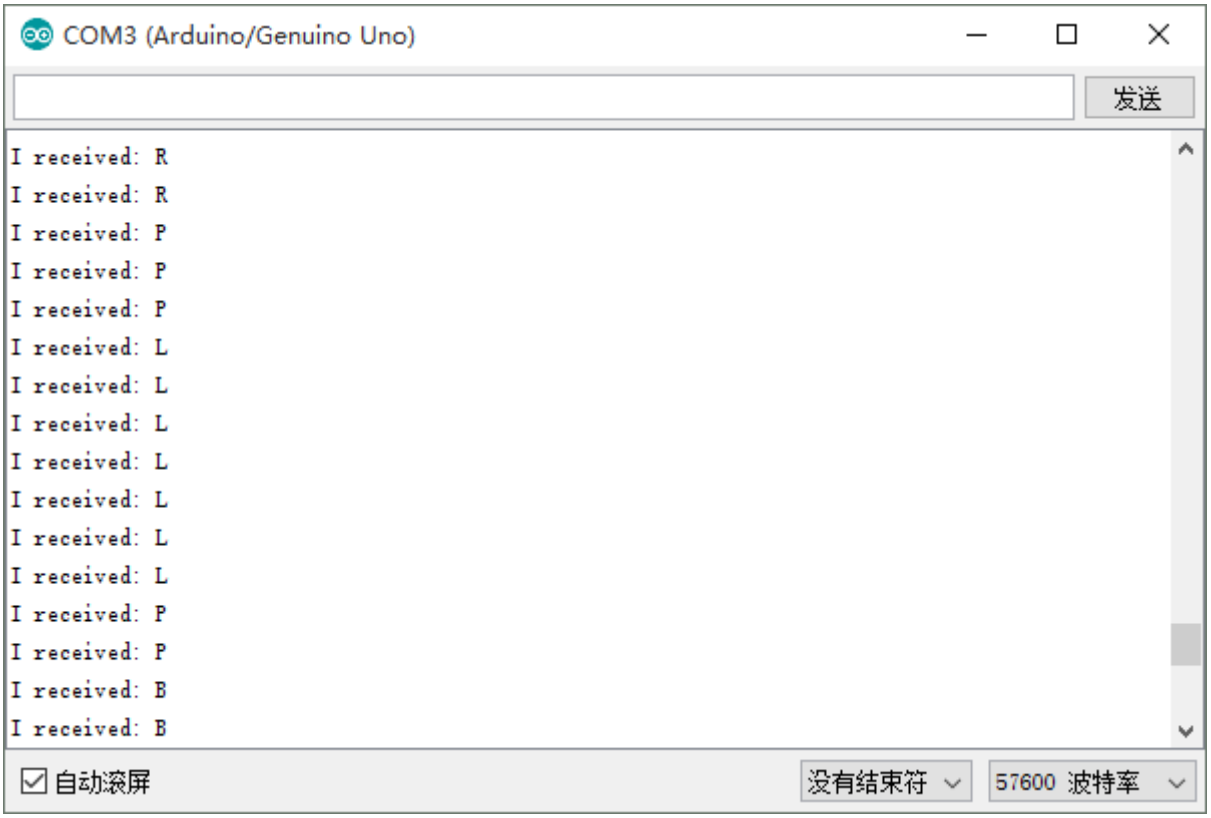


图 16　　接收到右转 R—停 P—左转 L—停 P—后退 B 指令

至此说明 NanoPi M2 和 Ardunio 已经通信成功，下一步只需判断所接收到的控制指令，控制小车即可。

4.　　小车端硬件连接和控制流程

（1）小车端硬件连接

对于小车端的硬件连接需要注意的主要是供电方式，这里选择 3s 锂电池作为小车的动力电源，而 9v 方块电池作为 Ardunio uno r3 的供电电源。选择 L298N 作为小车的直流电机的控制驱动器，以及 3DR 的无线数传作为无线控制的信号接收设备。整体连接方案如图 17 所示，实物连接如图 18 和 19 所示。

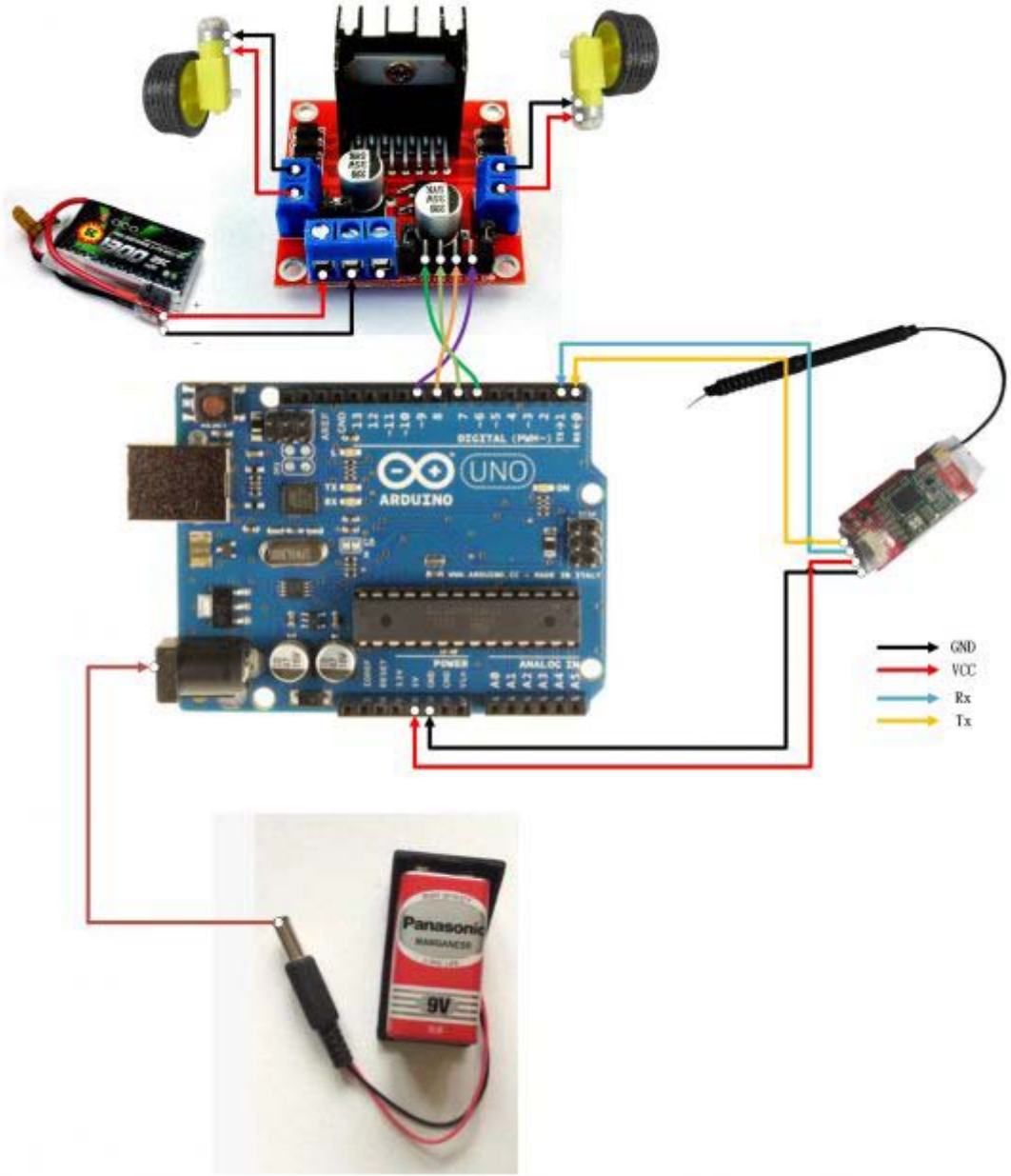


图 17 小车端硬件连接图

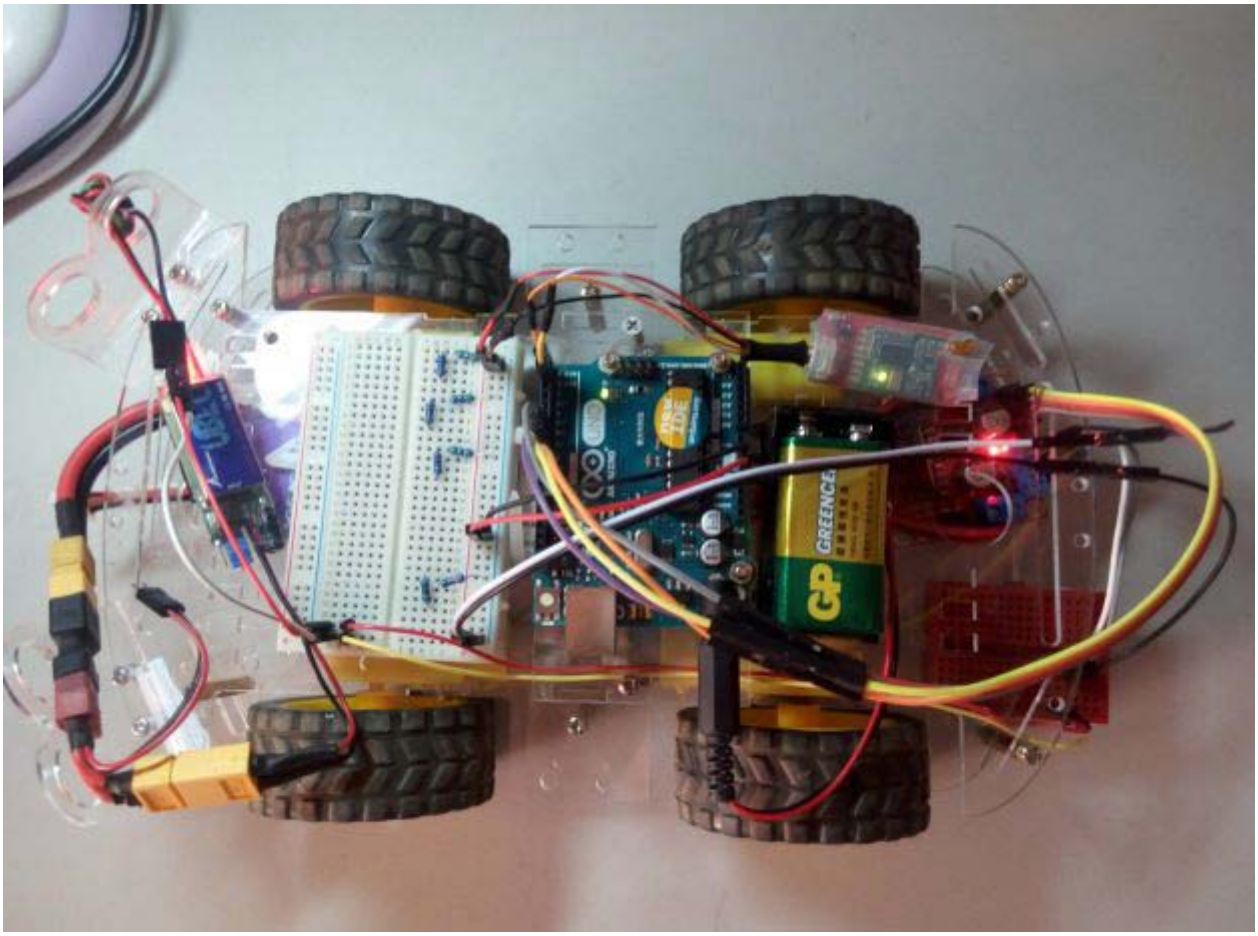


图 18 实物连接图

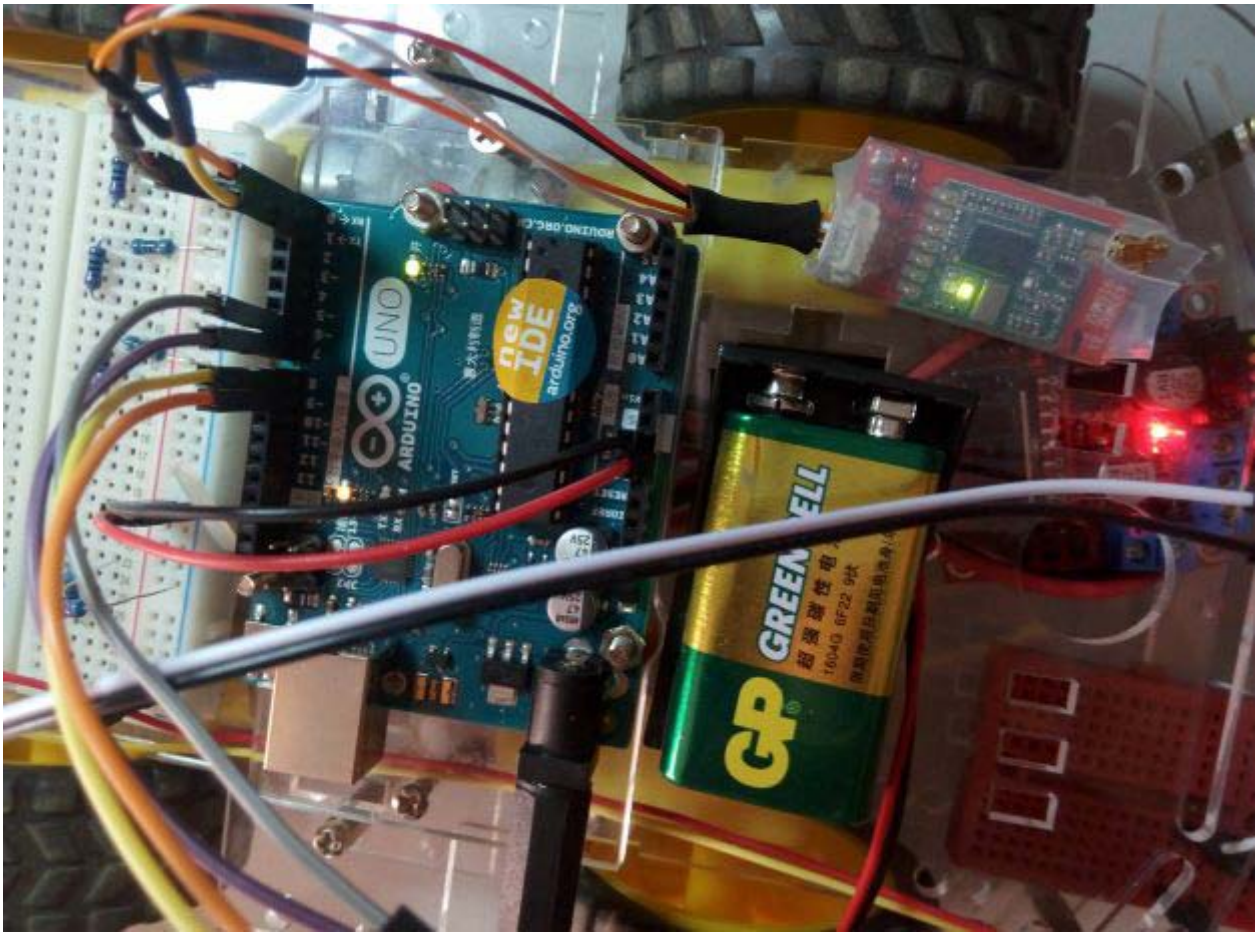


图 19　　局部连接图

（2）控制流程

此时，遥控器端控制软硬件都已经准备好了，小车控制端的连接也已经完成，下来就需要根据接收到的指令完成对应的小车运动的控制。而控制小车的运动其实就是控制直流电机的驱动器 L298N 使其驱动电机的转向，L298N 的控制逻辑如图 20 所示。

直流电机	旋转方式	IN1	IN2	IN3	IN4	调速PWM信号	
						调速端A	调速端B
M1	正转	高	低	/	/	高	/
	反转	低	高	/	/	高	/
	停止	低	低	/	/	高	/
M2	正转	/	/	高	低	/	高
	反转	/	/	低	高	/	高
	停止	/	/	低	低	/	高

图 20　　L298N 驱动电机的控制逻辑

软件控制流程如下：

1）　首先需要定义一个接受指令的字符变量：char command;

2）　然后定义接L298N控制引脚IN1,IN2,IN3,IN4的Ardunio引脚,这里选择Ardunio的6~9引脚与其对应;intpin_L1 = 6;　int pin_L2 = 7; int pin_R1 =8; int pin_R2 = 9;

3）　然后设置引脚的输入输出模式，如 pinMode(pin_L1,OUTPUT);

4） 设置串口波特率，需要注意的是这个波特率要与 NanoPi M2 的遥控端一致，参考第 3 小节第（2）部分的遥控器控制流程中的第 2）条，波特率的设置（57600）,因此，这里的波特率只能设置为 57600，即 Serial.begin(57600)；

5） 定义小车的运动模式函数，分别为 Foreward()、Turnleft()、Turnright()、Backward()和 Stop()；

6） 进入循环控制主体，读取 command，判断是什么指令，从而运行对应的函数即可，该部分程序见图 21。

完整的程序在附件中给出。

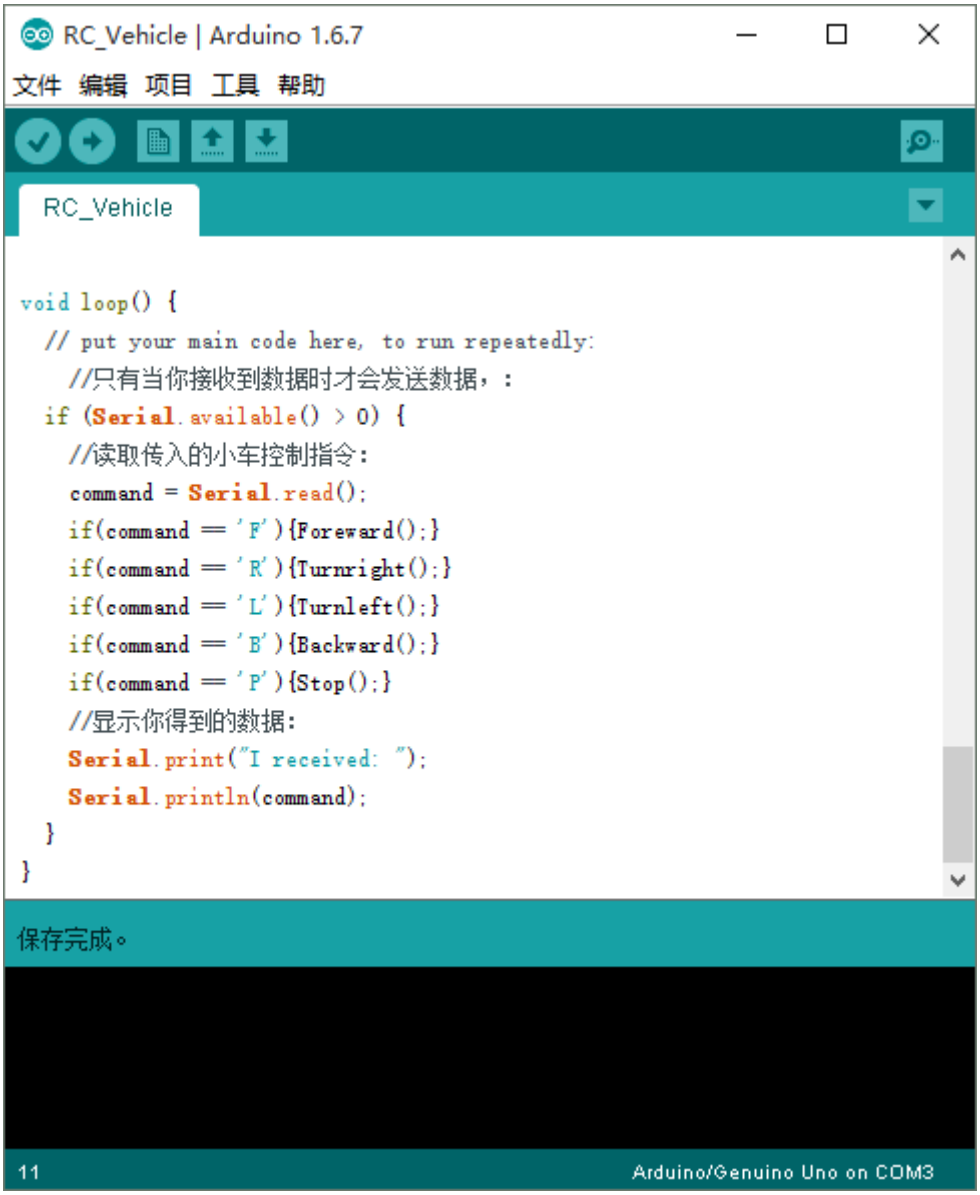


图 21 小车控制主循环部分程序

5. 遥控效果展示

遥控效果展示是一个比较头疼的问题，因为发烧友的平台没法投放视频，我只能把录制的视频上传到优酷上，大家可以点开链接去优酷上观看。

视频展示优酷链接：http://v.youku.com/v_show/id_XMTYwMTk0MTE4OA==.html

这里先给大家展示两图片：

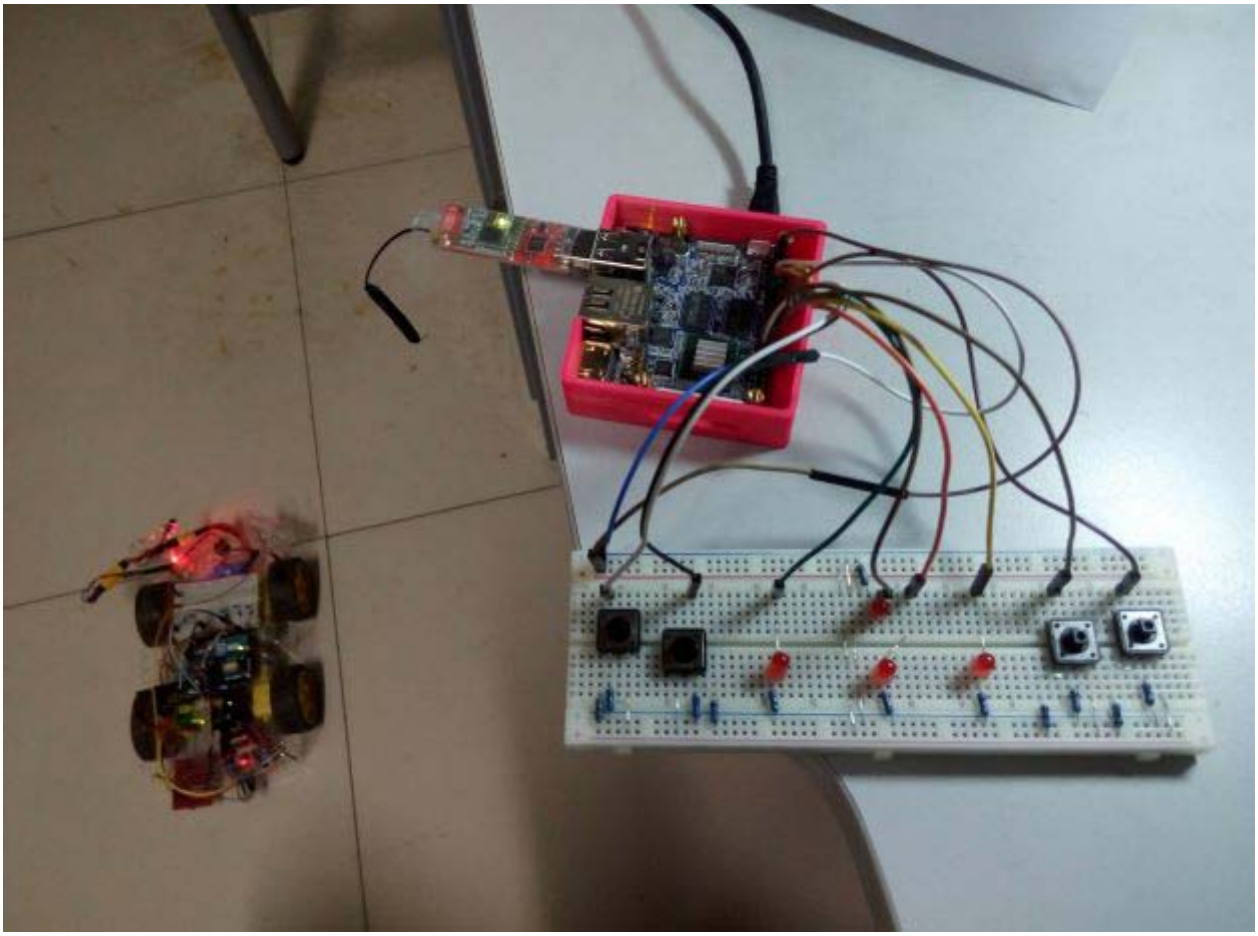


图 22　遥控器控制小车运动

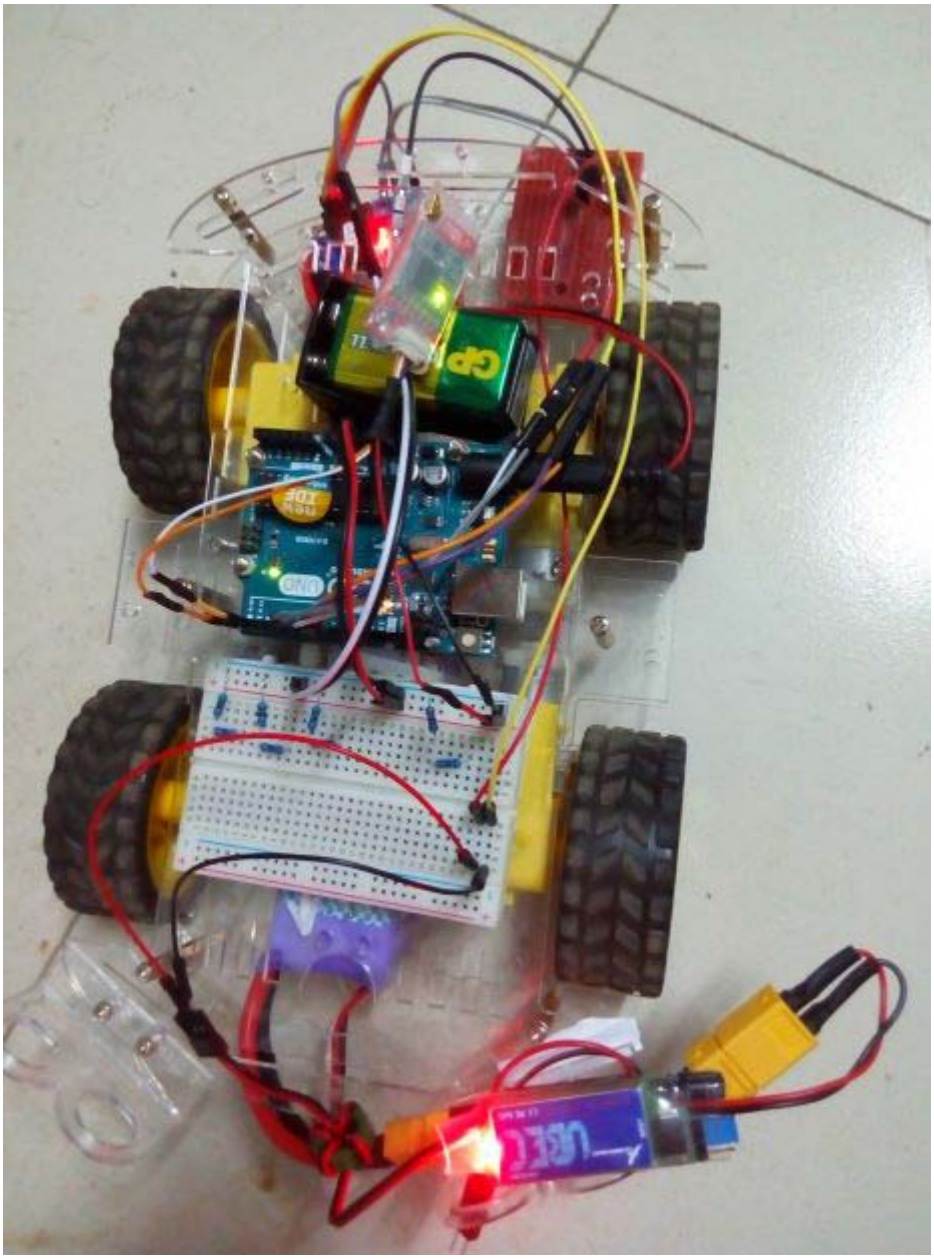


图 23　遥控小车

6. 结束语

至此，我的用 NanoPi M2 制作遥控器控制小车项目就结束了，所制作的遥控器能够完美的遥控我的小车，解决了遥控器制作过程的控制键电路设计和无线串口通信等关键技术，对于 NanoPi M2 GPIO 方面的开发有了较深的了解，我的试用就告一段落，特此申请结项。

如果后续还有进一步的改进和其它好玩的开发，也会继续给大家分享，还请大家对我的试用过程有什么问题的地方不吝赐教，一起探讨进步。

技巧十一： 搭建 WordPress 或者 Emlog 个人网站（基于 APAche2+PHP5+mysql）

来自：为她而狂野、 http://bbs.elecfans.com/jishu_613870_1_1.html

Apache2 的默认网页目录为/var/www/html/

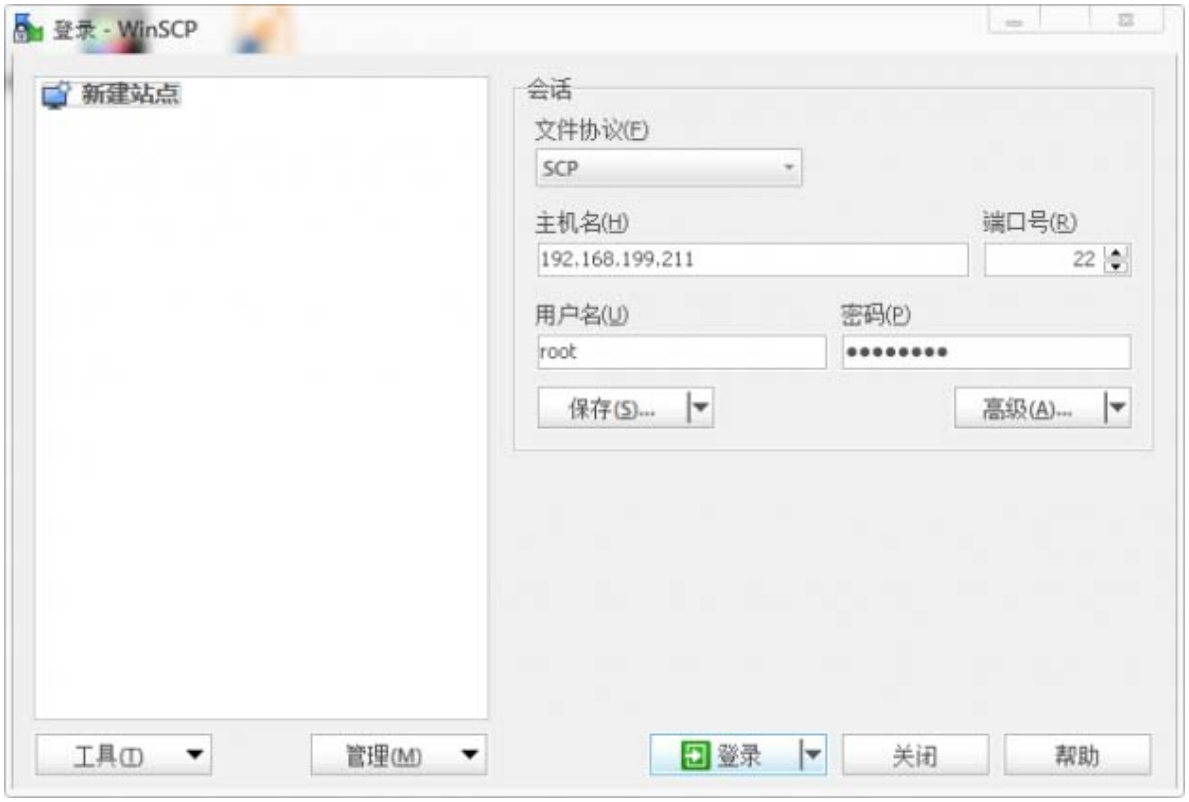
首先下载 WordPress 或 Emlog 的源码下载下来，然后解压

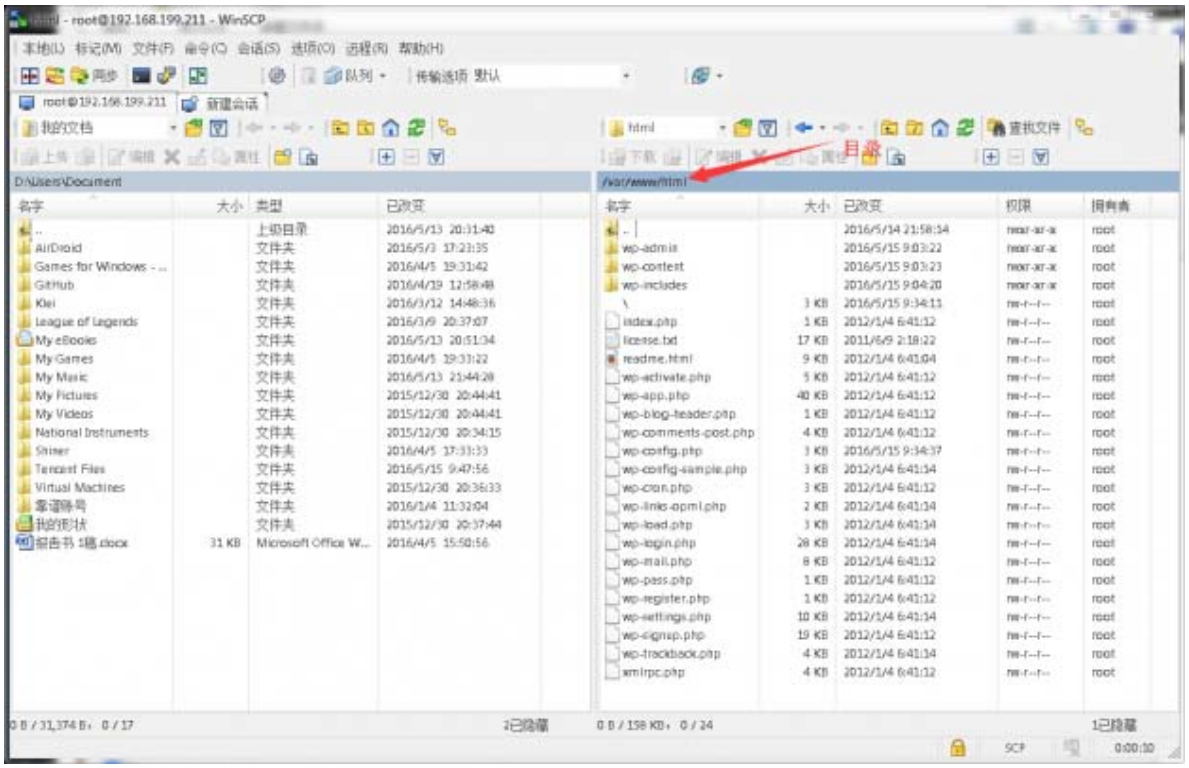
wordpress.zip (4.41 MB, 下载次数: 1)

emlog_5.3.1.zip (570.13 KB, 下载次数: 0)

然后使用 WinSCP，这个软件没有的小伙伴可以百度一下下载个破解版什么的～

解压网页源码，然后 SCP 连接到 NanoPiM2





就这样，然后用浏览器登录 NanoPi 的 IP ,就可以看到安装界面了，

我这里安装的是 Emlog

按照提示步骤就可以安装了～

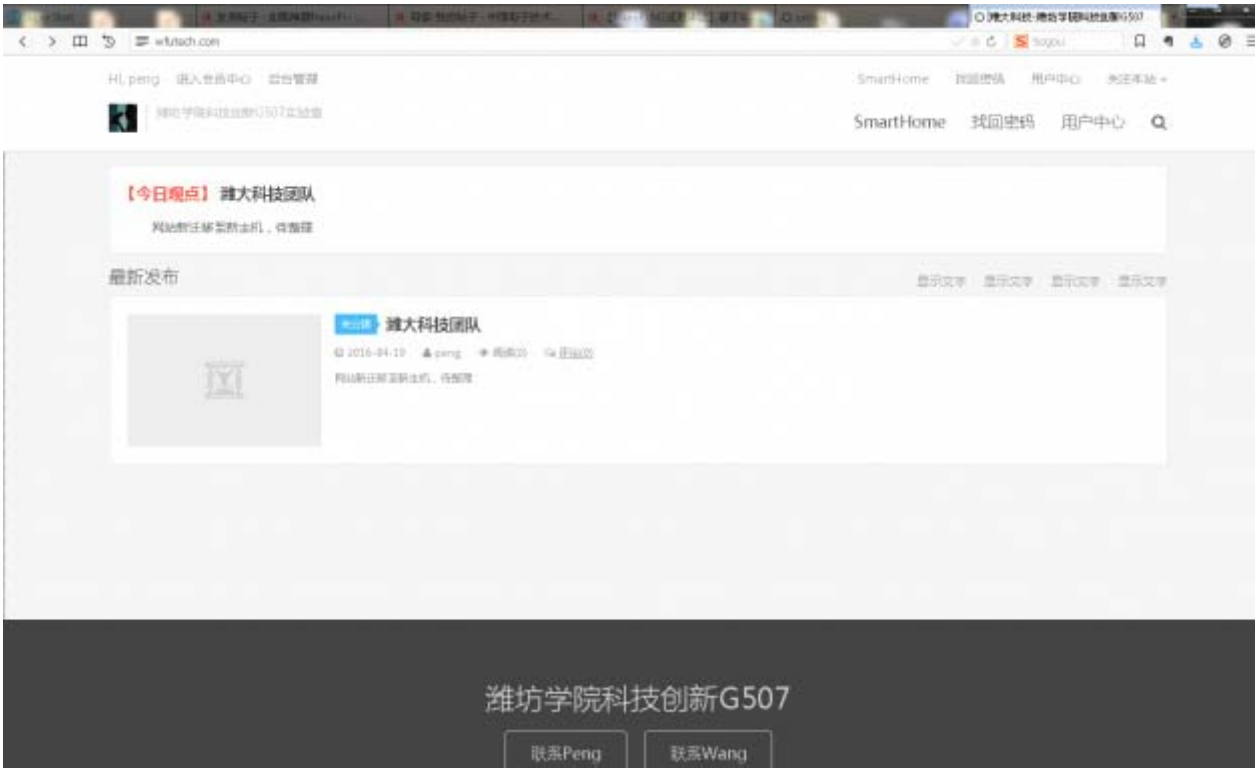
安装之前要新建一个空数据库，具体方法网上大把教程，就不罗嗦了～

安装完成后的一个页面：

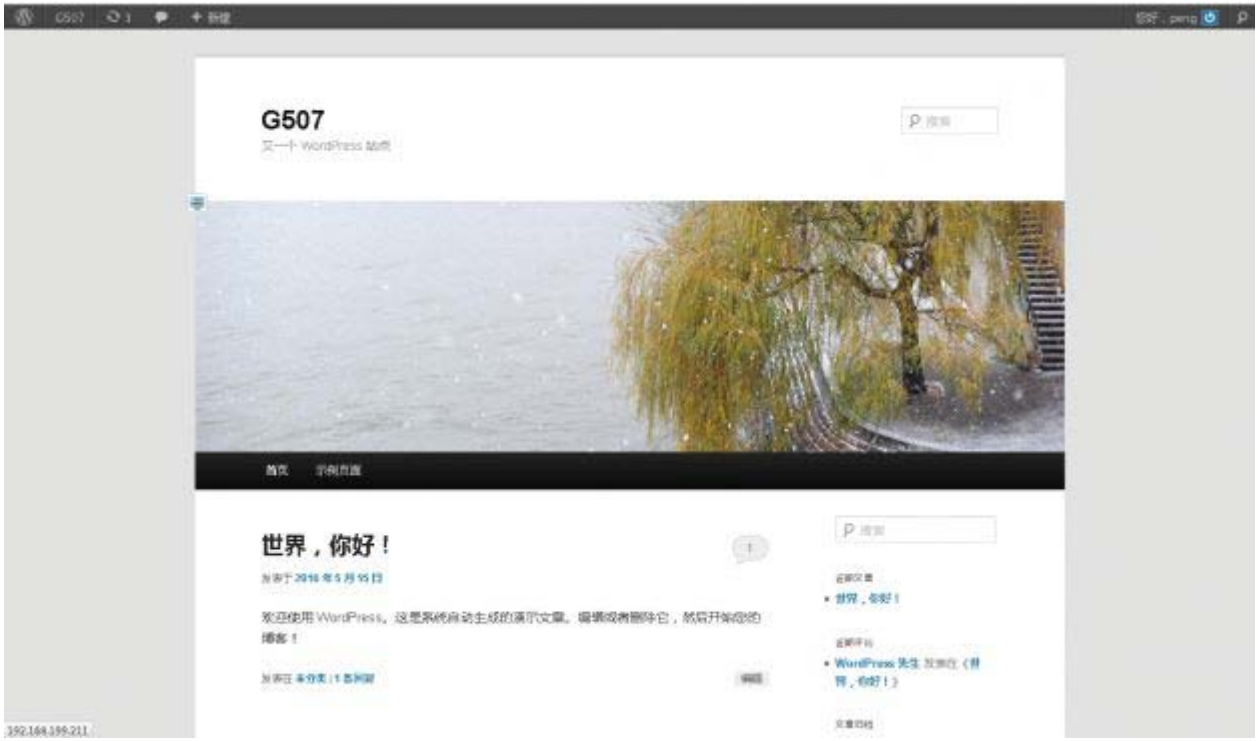
然后可以在万网申请个域名玩玩，解析到自己的 IP，就可以访问啦～

这些比较简单，就不写了..毕竟我比较懒，有问题留言吧～我也是新手，尽力而为~

我搭了一个网站试了一下



sd.jpg（76.06 KB，下载次数：0）



截屏图片.jpg（17.53 KB，下载次数：0）

技巧十二：Modbus-RTU 主站开发

来自：zxl_zxl http://bbs.elecfans.com/jishu_736180_1_1.html

NanoPi M2 试用体验之 Modbus-RTU 主站开发

本菜鸟之前的几篇使用报告介绍了基于 NanoPi M2 无线网卡设置、NFS 文件系统搭建、Mysql 安装、Mysql 可视化配置，以及基于 QT 的 Mysql 数据编程数据操作，以上这些报告当然不是独立的，每一篇报告都是基于本人关于板卡申请时，关于相关项目的一步一步实现过程，本篇报告介绍关于水泥搅拌站项目的一个关键核心部分：Modbus 通讯。

Modubus 通讯是工业应用领域使用非常广泛的一种基于串口的上层通讯协议，通过对底层串口信息的组合传输、解析、CRC 校验，大大提高数据传输的准确性和数据的可靠性。

自从 1979 年出现工业串行链路的事实标准以来， MODBUS 使成千上万的自动化设备能够通信。当前主流 PLC 控制器、传感器仪表均提供了对 Modus 协议支持，Modbus 通讯协议与 Profibus DP、CanOpen、DeviceNet 并列为事实上的工业标准通讯标准协议，而 Modbus 通讯协议又具有开发简单、承载要求低，从 51 单片机到 ARM Cortex 处理器，再到 X86 PC，只要以上处理器支持串口通讯，就可自定义编程实现 Modbus 通讯协议。

尽管我们可以 Modbus 协议标准自定义协议的实现，但是存在稳定性和可移植性的问题。目前开源 Modbus 协议源码的主要有 FreeModbus 和 libmodbus,通过应用，我个人感觉 libmodbus 的平台移植性可开发快速性更好，所以本篇报告主要讲解基于 libmodbus 的开发。

modbus 通讯协议又分为：Modbus-RTU、Modbus-ASCII、Modbus-TCP，由于多数控制仪表主要支持 Modbus-RTU、Modbus-TCP，本篇报告主要介绍 Modbus-RTU 通讯协议测试程序的开发。

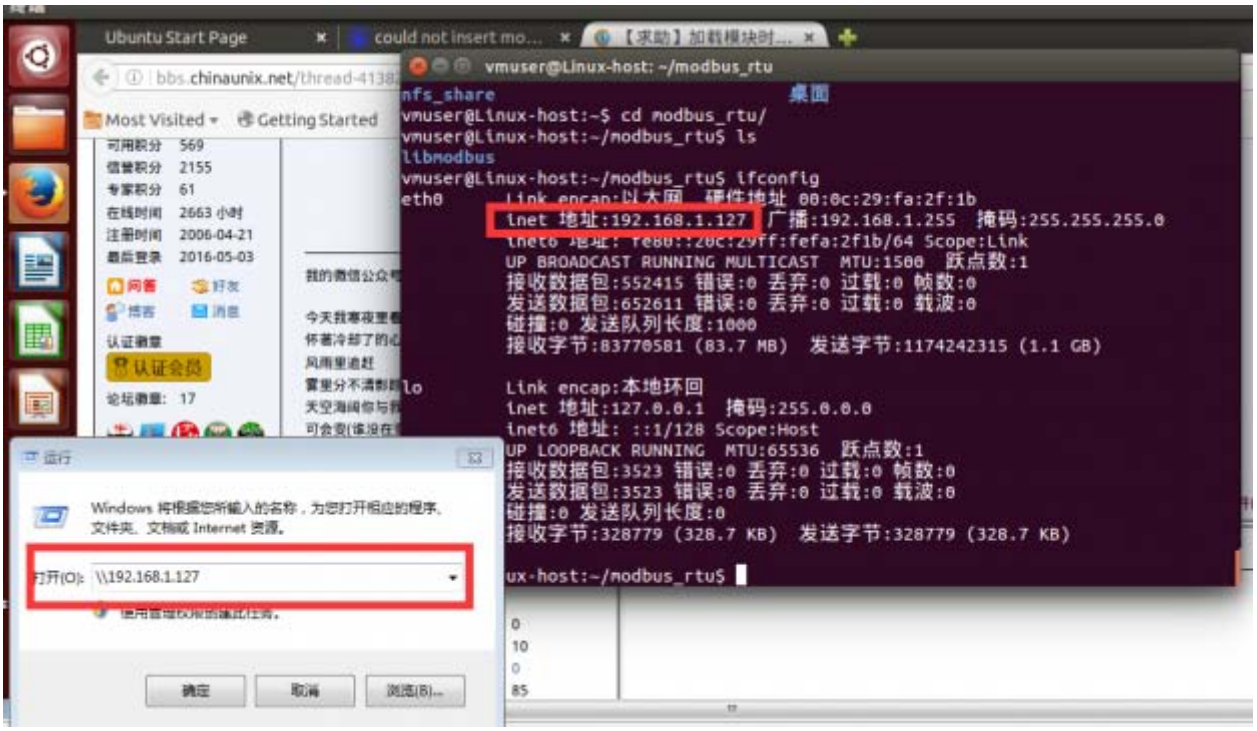
关于 Modbus-RTU 的更加详细介绍请自己搜索了解。

1.下载、配置 libmodbus

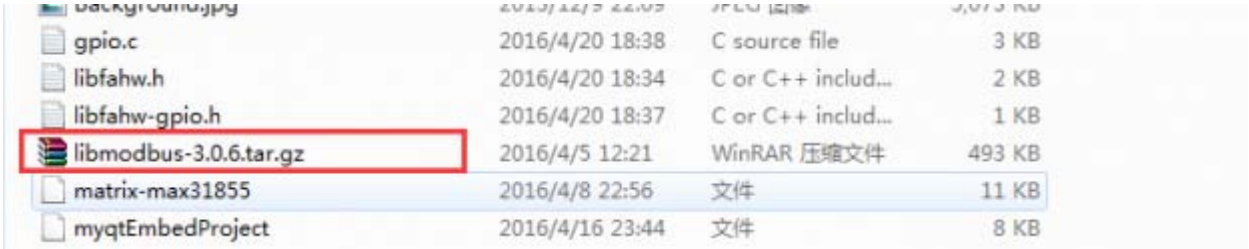
libmodbus 的下载地址：http://libmodbus.org/download/

当前稳定版本的最新版为 3.0.6，我们基于这个版本开发。

下载源码后，通过 samba 将文件拷贝到 Linux 虚拟机，如图所示：



将 libmodbus-3.0.6.tar.gz 拷贝到 samba 共享文件夹



在 Linux 命令行终端切入共享文件夹，输入命令：

tar -xzvf libmodbus-3.0.6.tar.gz,生成 libmodbus 源码文件夹，如图所示：

```
vmuser@Linux-host:~/modbus_rtu$ ls
libmodbus
vmuser@Linux-host:~/modbus_rtu$
```

输入命令：cd libmodbus，如图示所示：

```
vmuser@Linux-host:~/modbus_rtu/libmodbus$ ls
acinclude.m4  configure.ac     libmodbus.pc.in  MIGRATION  src
AUTHORS       COPYING.LESSER   m4               NEWS       tests
autogen.sh    doc              Makefile.am      README.md
```

输入命令：./autogen.sh

生成 configure 文件，如图示：

```
vmuser@Linux-host:~/modbus_rtu/libmodbus$ ls
acinclude.m4  autom4te.cache  configure.ac     m4            NEWS
aclocal.m4    build-aux       COPYING.LESSER   Makefile.am   README.md
AUTHORS       config.h.in     doc              Makefile.in   src
autogen.sh    configure       libmodbus.pc.in  MIGRATION     tests
vmuser@Linux-host:~/modbus_rtu/libmodbus$
```

输入命令：./configure && make install

命令完成后如图示：

```
vmuser@Linux-host:~/modbus_rtu/libmodbus$ ls
acinclude.m4    config.h        COPYING.LESSER   Makefile      src
aclocal.m4      config.h.in     doc              Makefile.am   stamp-h1
AUTHORS         config.log      libmodbus.pc     Makefile.in   tests
autogen.sh      config.status   libmodbus.pc.in  MIGRATION
autom4te.cache  configure       libtool          NEWS
build-aux       configure.ac    m4               README.md
vmuser@Linux-host:~/modbus_rtu/libmodbus$
```

生成 config.h 文件。

将 src 文件夹下的所有.h 和.c 文件，以及上一步生成的 config.h 文件拷贝通过 NFS 拷贝到 M2 的文件系统,如执行命令：

sudo mount -t nfs 192.168.1.xxx:nfsshare -onolock

sudo mkdir /libmodbus

cp ./src/*.h　/mnt/libmodbus

cp ./src/.c /mnt/libmodbus/

cp config.h /mnt/libmodbus

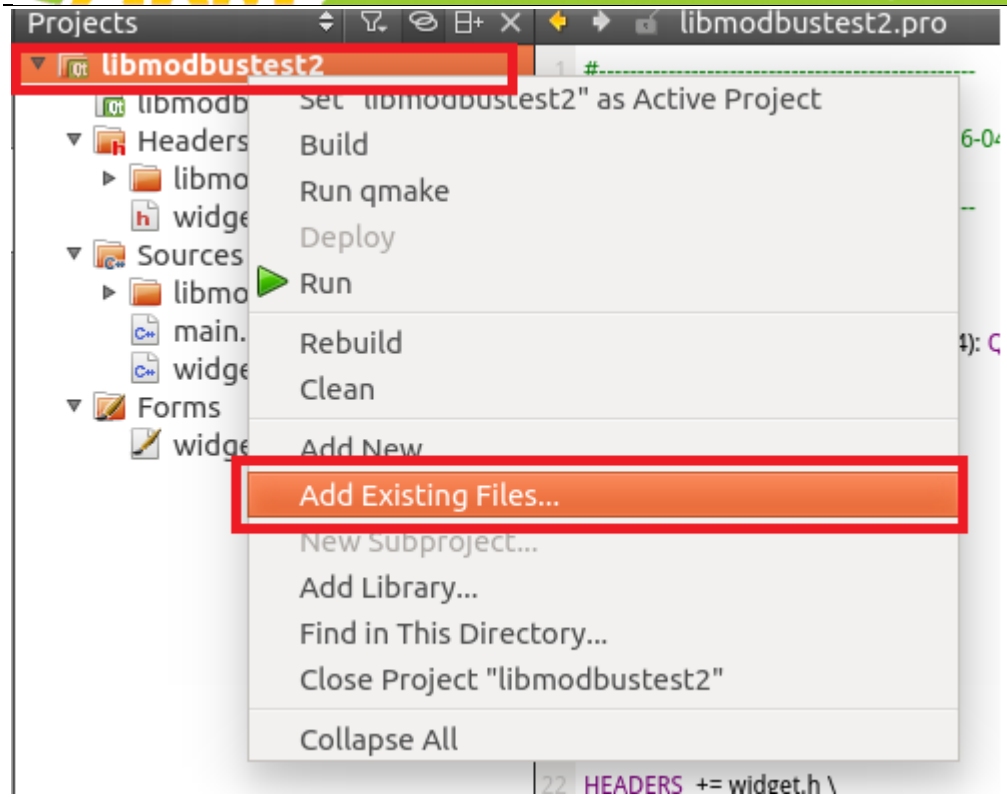
2. 创建 QT 测试项目

搭建 M2 的硬件运行环境，开机启动系统，启动 QTCreator，创建 libmodbustest 工程。项目创建结束后，通过命令行将 libmodbus 中的文件拷贝到项目目录下：

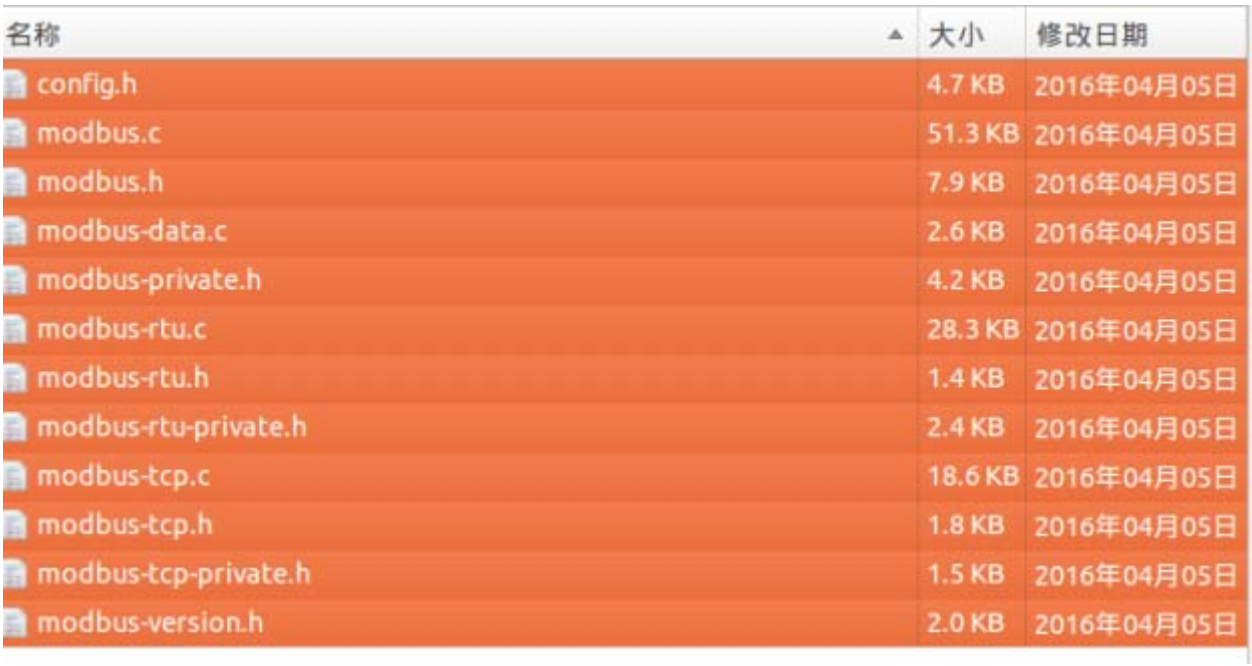
cp -r　/nfsshare/libmodbus　项目文件夹路径

将 libmodbus 文件夹中文件添加到项目中：

右击项目名称，选择“Add Existing Files...”，如图所示：



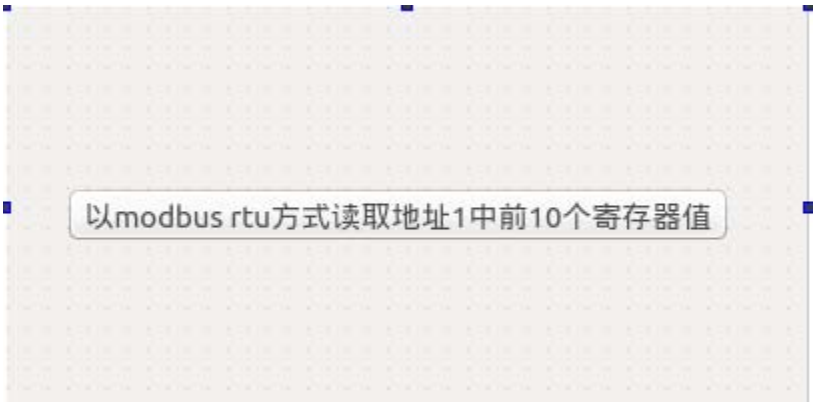
选择 libmodbus 文件夹：



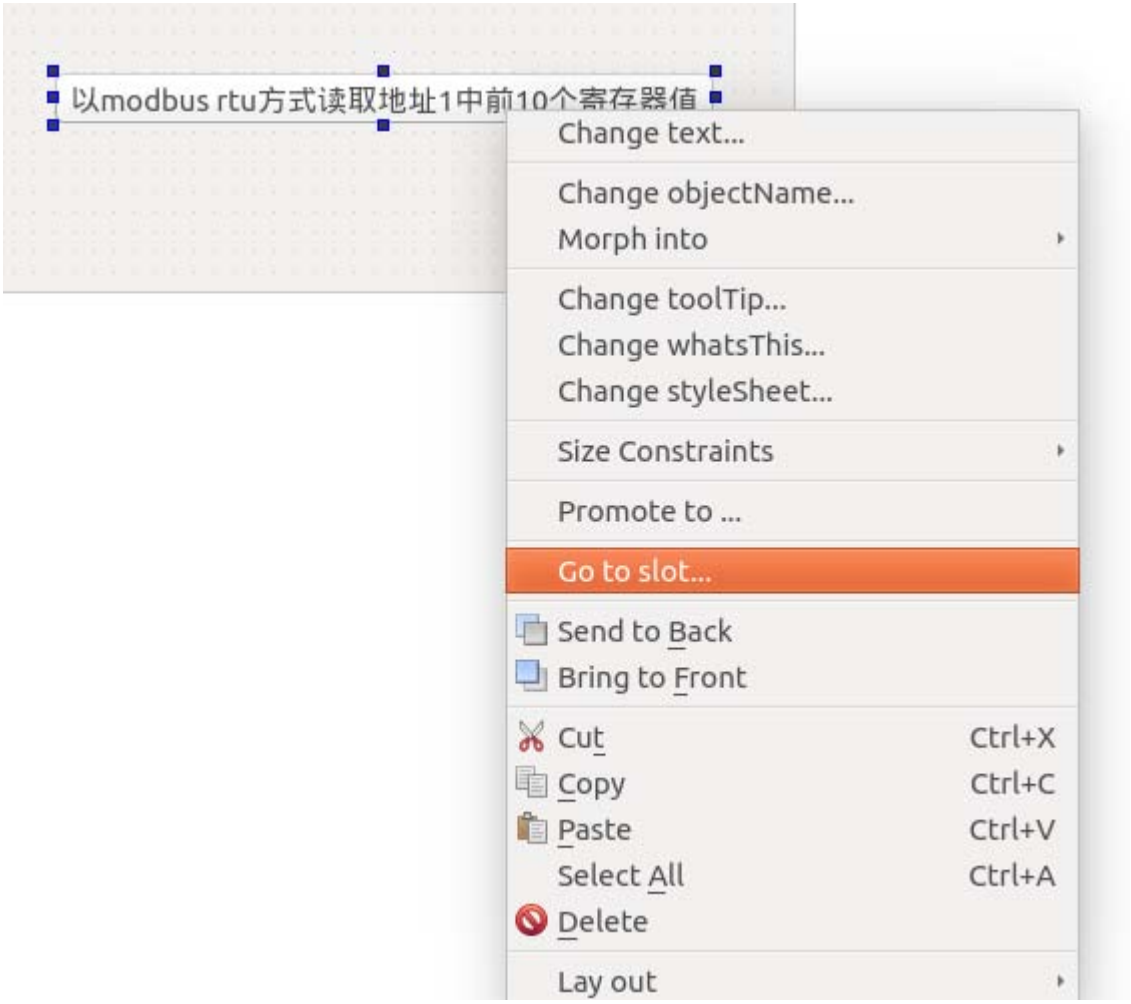
名称	大小	修改日期
config.h	4.7 KB	2016年04月05日
modbus.c	51.3 KB	2016年04月05日
modbus.h	7.9 KB	2016年04月05日
modbus-data.c	2.6 KB	2016年04月05日
modbus-private.h	4.2 KB	2016年04月05日
modbus-rtu.c	28.3 KB	2016年04月05日
modbus-rtu.h	1.4 KB	2016年04月05日
modbus-rtu-private.h	2.4 KB	2016年04月05日
modbus-tcp.c	18.6 KB	2016年04月05日
modbus-tcp.h	1.8 KB	2016年04月05日
modbus-tcp-private.h	1.5 KB	2016年04月05日
modbus-version.h	2.0 KB	2016年04月05日

修改 libmodbus 文件夹下的 modbus-private.h 和 modbus.c 代码中的”#include<config.h>”为”#include<libmodbus/config.h>”，保存。

设计界面如图示：



右击上图中的按钮，选择“goto slot...”，如下图所示：



弹出对话框，选择第一项，如所所示：

点击“OK”按钮，创建该按钮的命令执行函数，输入代码如下图所示：

```
void Widget::doRtuQuery()
{
    modbus_t *mb;
    uint16_t tab_reg[32]={0};

    mb = modbus_new_rtu("/dev/ttyAMA2", 9600, 'N', 8, 1);  //相同的端口只能同时打开一个
    modbus_set_slave(mb, 17);  //设置modbus从机地址

    modbus_connect(mb);

    struct timeval t;
    t.tv_sec=0;
    t.tv_usec=1000000;  //设置modbus超时时间为1000毫秒
    modbus_set_response_timeout(mb, &t);

    int regs=modbus_read_registers(mb, 0, 10, tab_reg);

    QMessageBox::about(NULL, "报告", QString("Rtu读取寄存器的个数:%1").arg(regs));
    QString qs = QString("%1\n%2\n%3\n%4\n%5\n%6\n%7\n%8\n%9\n%10\n").arg(tab_reg[0]).arg(tab_reg[1]).arg(tab_reg[2]).arg(tab_reg[3]).arg(tab_reg[4]).arg(t
    QMessageBox::information(this,"Info",qs);
    modbus_close(mb);
    modbus_free(mb);
}
```

波特率9600，M2上UART3接口的驱动对应文件为：/dev/ttyAMA2

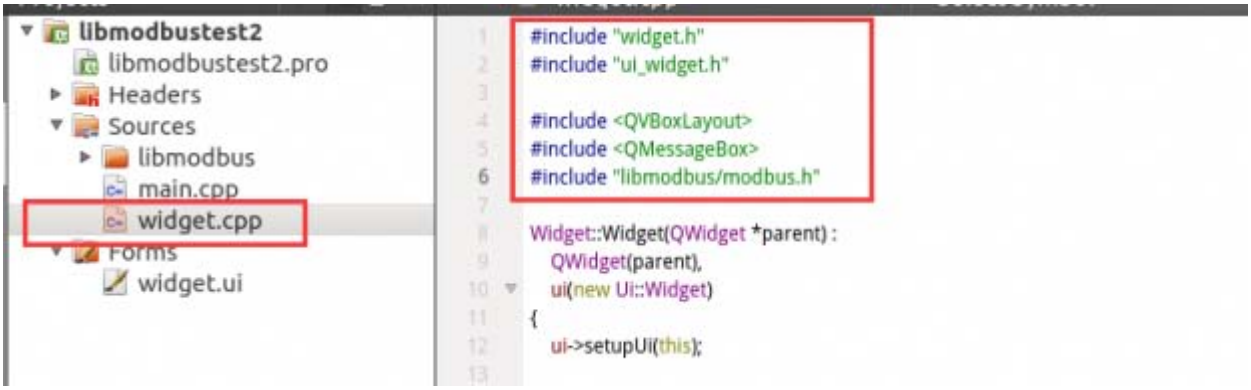
从站地址选为：17

详细代码如下：

```
modbus_t *mb;
        uint16_t tab_reg[32]={0};
        mb = modbus_new_rtu("/dev/ttyAMA2", 9600, 'N', 8, 1);    //相同的端口只能同时打开一个
        modbus_set_slave(mb, 17);    //设置 modbus 从机地址
        modbus_connect(mb);
        struct timeval t;
        t.tv_sec=0;
        t.tv_usec=1000000;    //设置 modbus 超时时间为 1000 毫秒
        modbus_set_response_timeout(mb, &t);
        int regs=modbus_read_registers(mb, 0, 10, tab_reg);
        QMessageBox::about(NULL, "报告", QString("Rtu 读取寄存器的个数:%1").arg(regs));
        QString qs=QString("%1\n%2\n%3\n%4\n%5\n%6\n%7\n%8\n%9\n%10\n").arg(tab_reg[0]).arg(tab_reg[1]).arg(tab_reg[2]).arg(tab_reg[3]).arg(tab_reg[4]).arg(tab_reg[5]).arg(tab_reg[6]).arg(tab_reg[7]).arg(tab_reg[8]).arg(tab_reg[9]);
        QMessageBox::information(this,"Info",qs);
        modbus_close(mb);
```

```
    modbus_free(mb);
```

输入以上代码后，在 widget.cpp 文件中输入头文件：



至此，QT 项目工程创建完毕，下面开始搭建硬件环境。

3. 硬件测试环境搭建

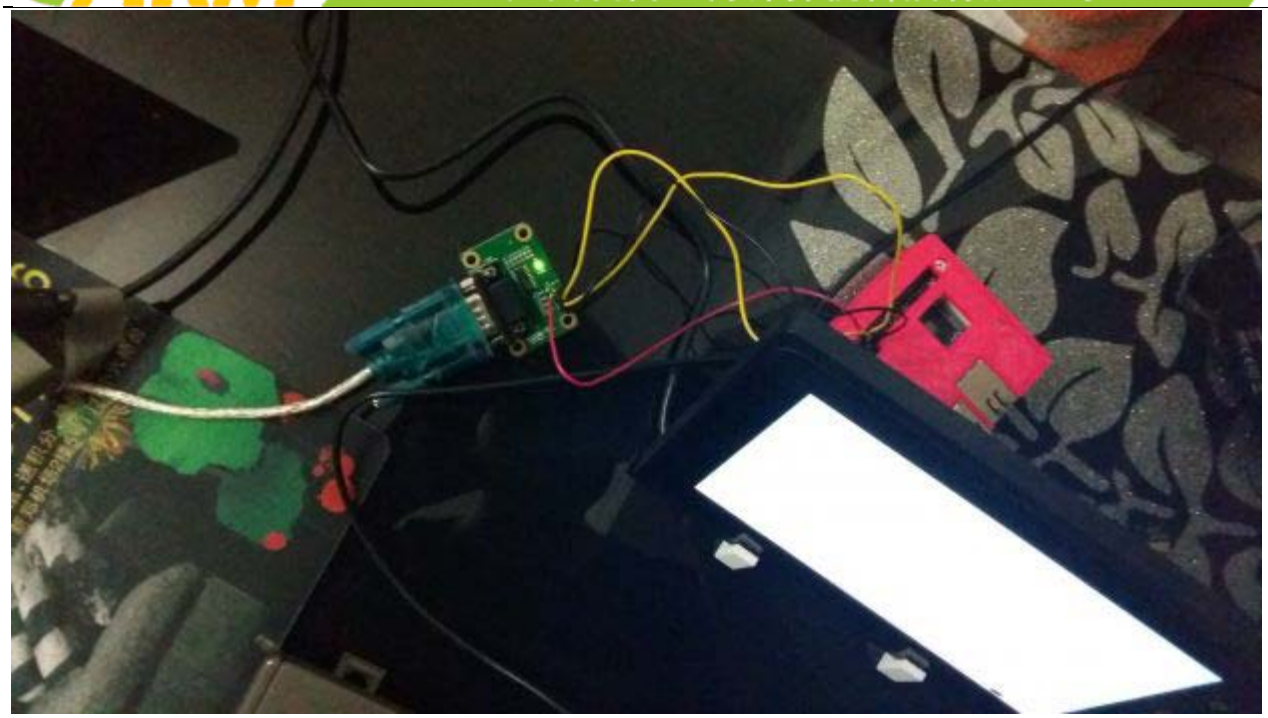
由于 Nanopi M2 40 针扩展接口中支持 UART3 接口，我们就将 UART3 接口与外围电路连接。（需要注意的是，UART3 接口的驱动程序文件为 ttyAMA2,所以我们在上述程序中直接将打开的文件设置为“/dev/ttyAMA2”，请根据自己的实际情况适当修改）。

M2 扩展的 UART3 接口为 TTL 电平，无法与 PC 机 RS232 接口电平直连，我们需要一个 UART TTL 电平转 RS232 电平的扩展板，可以使用友善官方的扩展板，也可以使用其他第三方的扩展板，本人采用的是自己手头现有的一块的 TTL 转 RS232 电平的接口板，如图示：

由于该转换板已经将 RXD TXD 做成交叉连接方式，所以本试验的连接如下

转换板	NanoPi M2
VCC	02(5VPower)
RXD	10(RX3)
TXD	08(TX3)
GND	06(DGND)

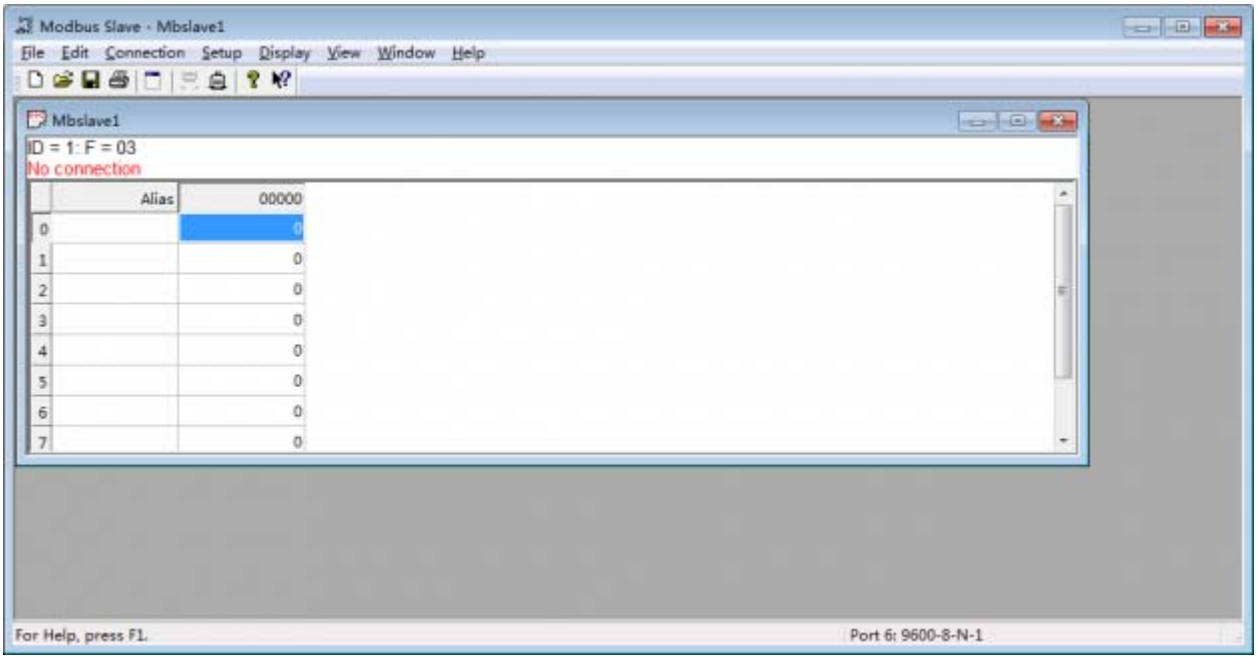
连接后的电路如图所示：



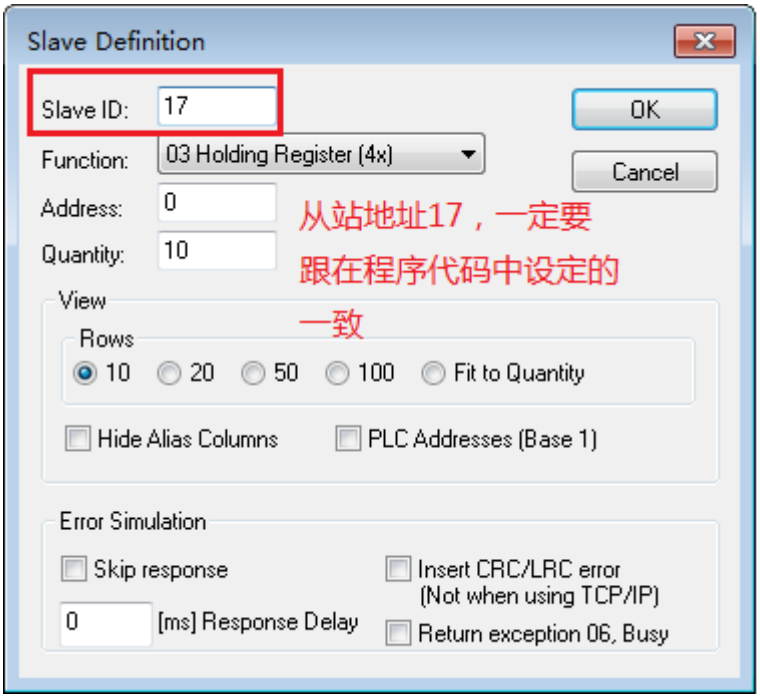
4. PC 端设置

由于手头没有相关的 Modbus 从站仪表，我们通过在 PC 端安装一个模拟软件，来测试我们以上的基于 QT 的测试程序是否可用。

下载一个 Modbus Slave 软件（试用版），然后启动该软件，如图所示：

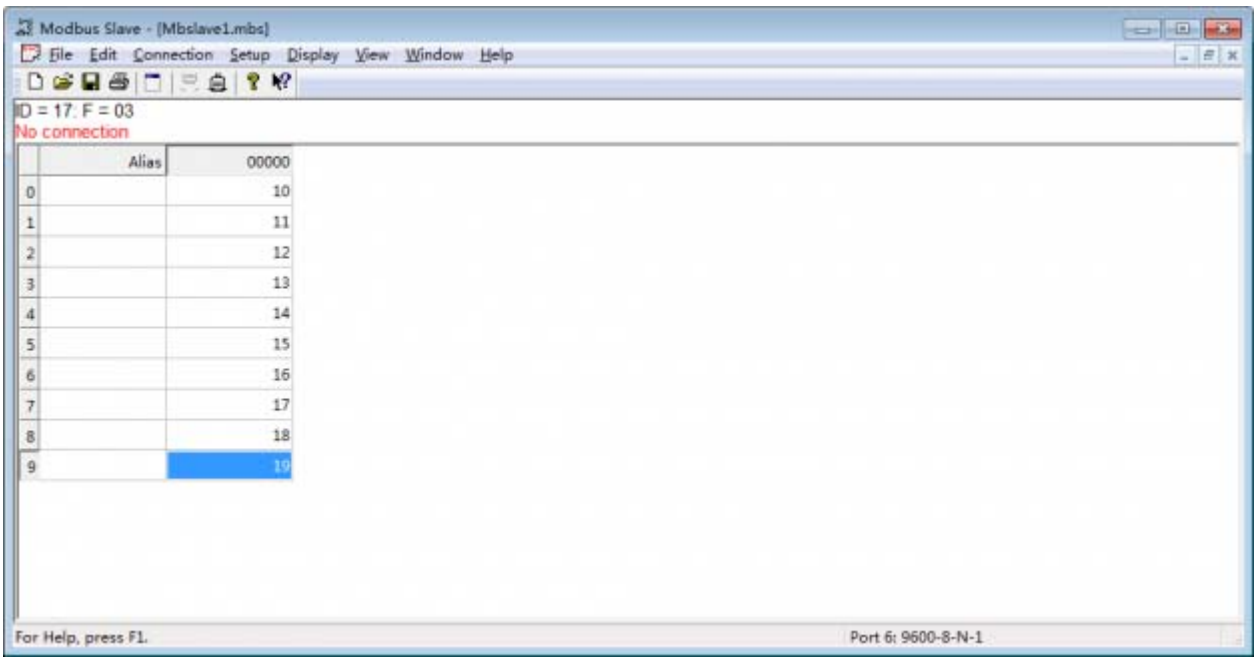


在菜单“Setup”中，选择“Slave Definition...”，设置如下：



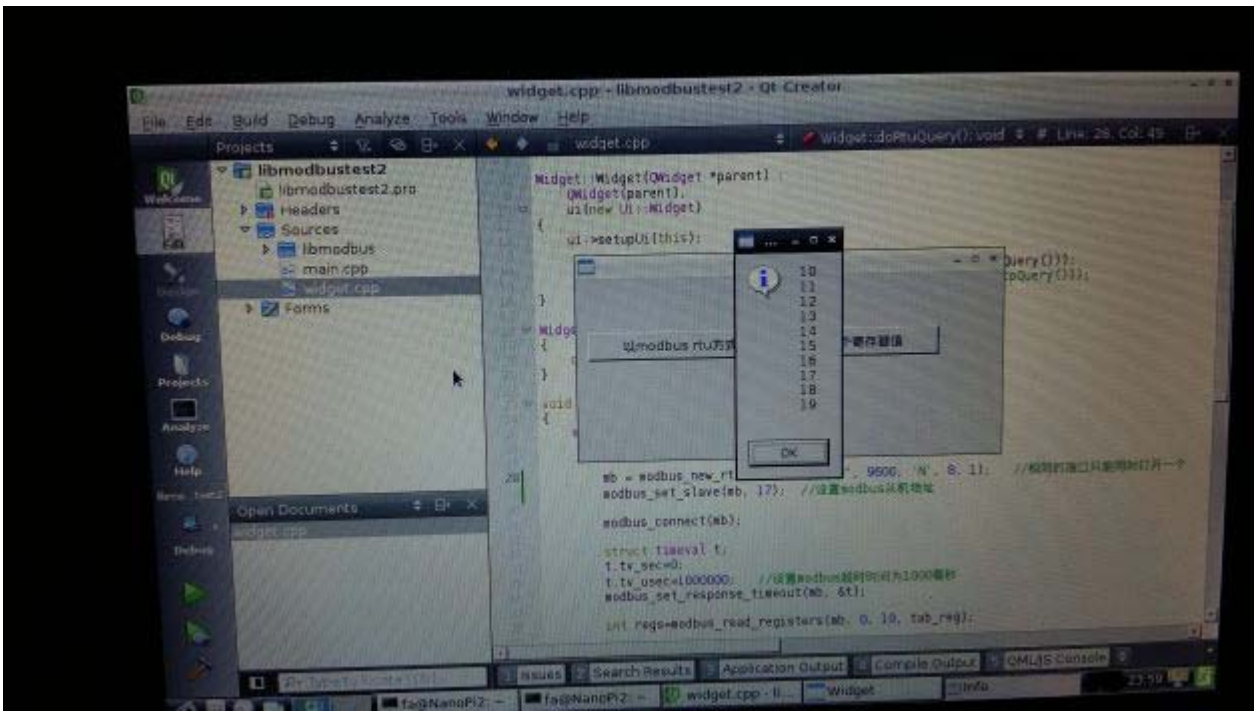
在“connection”菜单中点击“connect”，设置通讯参数如下：

然后设置从站的寄存器数据如下图示：



5. 启动测试

Modbus Slave 软件根据上述设置，处于连接状态后，我们启动 QT 测试项目，点击“以 modbus rtu 方式读取地址 1 中前 10 个寄存器值”按钮，通讯结果如图所示：



出现结果“10,11,12,13,14,15,16,17,18,19”的对话框，说明我们的测试成功了。

对于本项目而言，我就可以通过 Modbus-RTU 通讯协议，RS232 转 RS485 电气接口与市面上流行通用的称重仪表实现数据互联，达到了实时采集水泥配料的动态数据。而对于其他的应用，有了 Modbus-RTU 功能，基本上可以与车间内的绝大多数第三方设备、仪表通信，实现数据的共享，在通过 Nanopi M2 的有线或无线以太网，将采集的数据中继到其他主机或服务器，让车间的独立设备真正“动”起来，解决了车间级设备的“信息孤岛”问题。

技巧十三：Modbus-TCP 主站开发（Modbus 协议二）

来自：zxl_zxl http://bbs.elecfans.com/jishu_801460_1_1.html

基于 QT 的 Modbus-TCP 的开发

在上一篇的使用报告中，我们详细介绍了 libmodbus 的安装、配置，以及基于 QT 的 Modbus-RTU 应用程序开发。本篇报告介绍另外一种 Modbus 通讯协议，即 Modbus-TCP，Modbus-TCP 与 RTU 最大的不同是传输介质，以及中间协议转换。

Modbus-RTU 采用 RS232 或 RS485 电气接口，传输介质采用普通电缆或双绞线。Modbus-TCP 采用以太网芯片收发数据，可采用以太网双绞线或光线传输，Modbus-TCP 在普通的协议基础上将数据转换为 SOCKET 数据类型，接收端解析数据还原为普通 Modbus 协议数据类型。由于以太网高速传输，以及距离可无限扩展，近年来逐步得到了广泛应用，部分传感器仪表同时指出 Modbus-RTU 和 Modbus-TCP 两种协议。

关于 Modbus-RTU 的更加详细介绍请自己搜索了解。

本试用报告主要介绍 QT 工程创建和通讯测试过程。

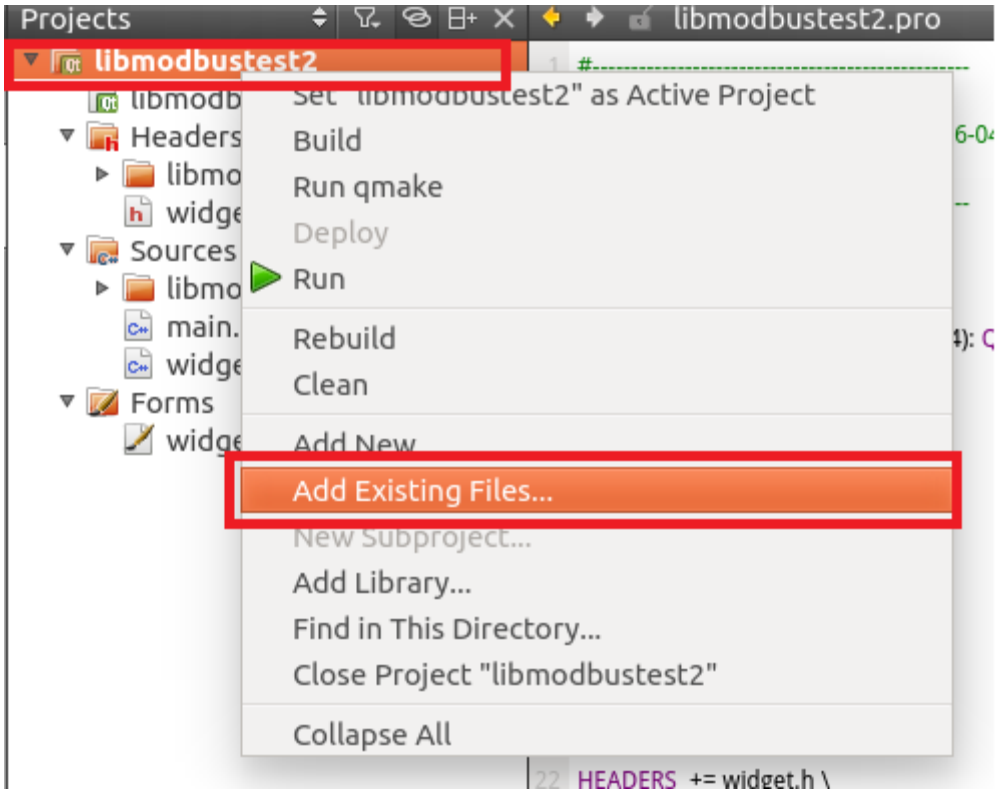
1. 创建 QT 测试项目

搭建 M2 的硬件运行环境，开机启动系统，启动 QTCreator，创建 libmodbustest2 工程。项目创建结束后，通过命令行将 libmodbus 中的文件拷贝到项目目录下：

cp -r　/nfsshare/libmodbus　项目文件夹路径

将 libmodbus 文件夹中文件添加到项目中：

右击项目名称，选择“Add Existing Files...”，如图所示：

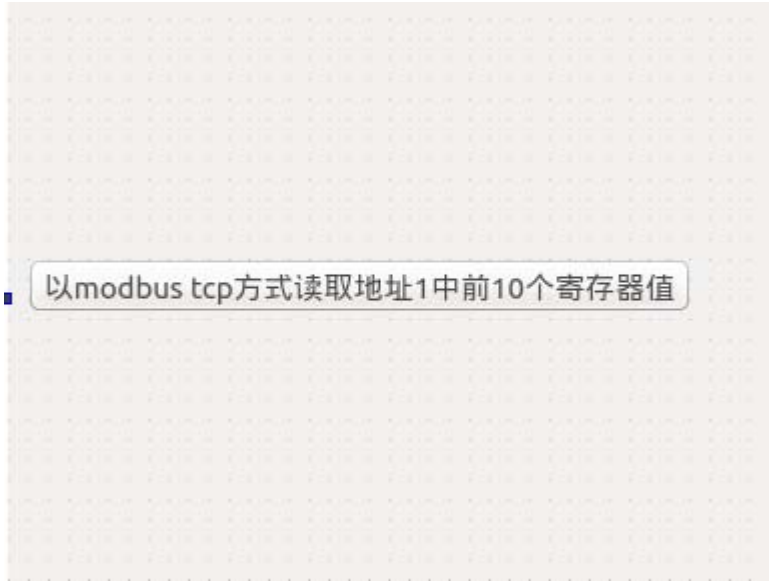


选择 libmodbus 文件夹：

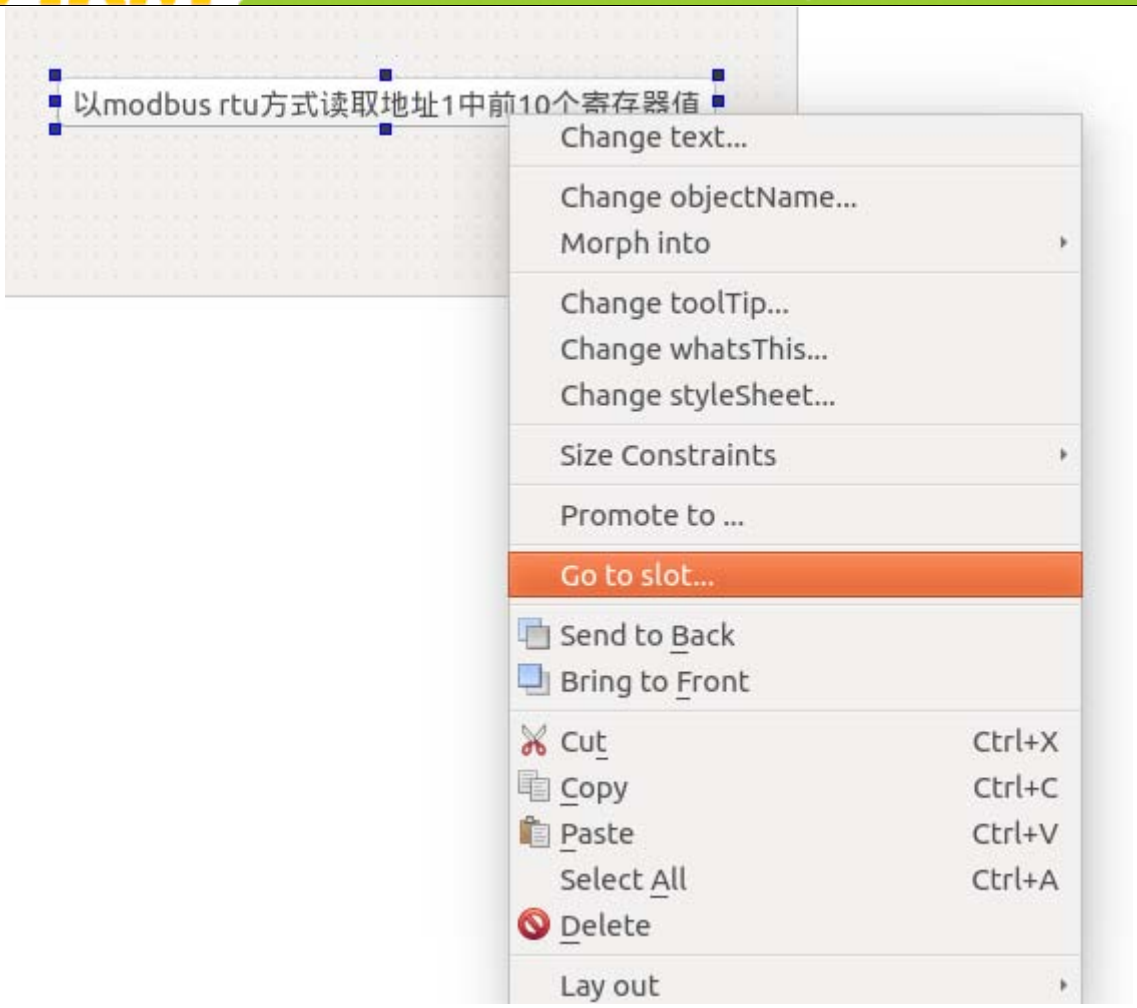
名称	大小	修改日期
config.h	4.7 KB	2016年04月05日
modbus.c	51.3 KB	2016年04月05日
modbus.h	7.9 KB	2016年04月05日
modbus-data.c	2.6 KB	2016年04月05日
modbus-private.h	4.2 KB	2016年04月05日
modbus-rtu.c	28.3 KB	2016年04月05日
modbus-rtu.h	1.4 KB	2016年04月05日
modbus-rtu-private.h	2.4 KB	2016年04月05日
modbus-tcp.c	18.6 KB	2016年04月05日
modbus-tcp.h	1.8 KB	2016年04月05日
modbus-tcp-private.h	1.5 KB	2016年04月05日
modbus-version.h	2.0 KB	2016年04月05日

修改 libmodbus 文件夹下的 modbus-private.h 和 modbus.c 代码中的”#include<config.h>”为”#include<libmodbus/config.h>”，保存。

设计界面如图示：



右击上图中的按钮，选择“goto slot...”，如下图所示：



弹出对话框，选择第一项，如图所示：

点击“OK”按钮，创建该按钮的命令执行函数，输入代码如下图所示：

```
void Widget::doTcpQuery()
{
    modbus_t *mb;
    uint16_t tab_reg[32]={0};

    mb = modbus_new_tcp("192.168.1.110", 1502);  //由于是tcp client连接，在同一个程序中相同的端口可以连接多次。
    modbus_set_slave(mb, 17);  //从机地址

    modbus_connect(mb);

    struct timeval t;
    t.tv_sec=0;
    t.tv_usec=1000000;  //设置modbus超时时间为1000毫秒，注意：经测试，如果没有成功建立tcp连接，则该设置无效。
    modbus_set_response_timeout(mb, &t);

    int regs=modbus_read_registers(mb, 0, 10, tab_reg);

    QMessageBox::about(NULL, "报告", QString("Tcp读取寄存器的个数:%1").arg(regs));
    QString qs = QString("%1\n%2\n%3\n%4\n%5\n%6\n%7\n%8\n%9\n%10\n").arg(tab_reg[0]).arg(tab_reg[1]).arg(tab_reg[2]).arg(tab_reg[3]).arg(tab_reg[4]).arg(t
    QMessageBox::information(this,"Info",qs);
    modbus_close(mb);
    modbus_free(mb);
}
```

IP地址设置为PC机Windows下的IP地址，端口号与Modbus slave软件设置的一致，从站地址也必须与Modbus slave端设置的一致

详细代码如下：

modbus_t *mb;

uint16_t tab_reg[32]={0};

mb = modbus_new_tcp("192.168.1.110", 1502);　//由于是 tcp client 连接，在同一个程序中相同的端口可以连接多次。IP 地址根据自己的 PC 地址设置，1502 端口也可根据实际情况自己选择

modbus_set_slave(mb, 17);　//从机地址

modbus_connect(mb);

struct timeval t;

t.tv_sec=0;

t.tv_usec=1000000;　//设置 modbus 超时时间为 1000 毫秒，注意：经测试，如果没有成功建立 tcp 连接，则该设置无效。

modbus_set_response_timeout(mb, &t);

int regs=modbus_read_registers(mb, 0, 10, tab_reg);

QMessageBox::about(NULL, "报告", QString("Tcp 读取寄存器的个数:%1").arg(regs));

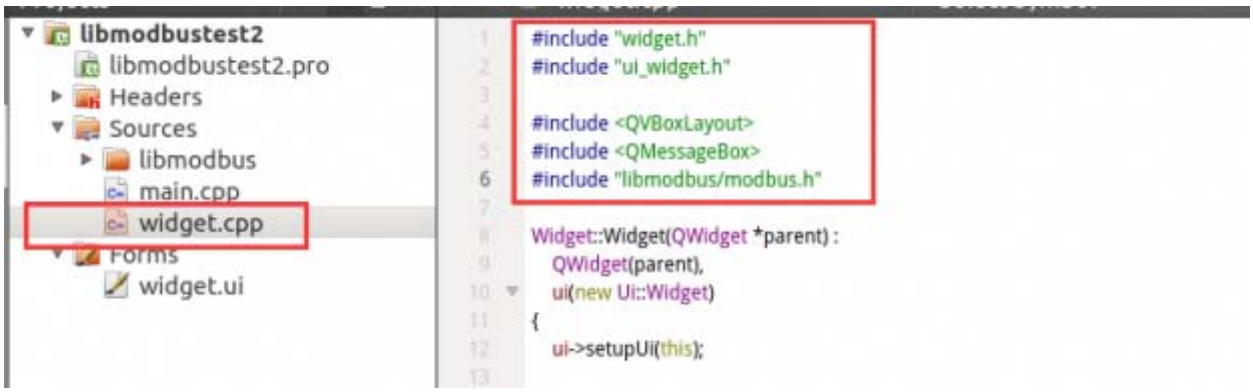
QString qs =QString("%1\n%2\n%3\n%4\n%5\n%6\n%7\n%8\n%9\n%10\n").arg(tab_reg[0]).arg(tab_reg[1]).arg(tab_reg[2]).arg(tab_reg[3]).arg(tab_reg[4]).arg(tab_reg[5]).arg(tab_reg[6]).arg(tab_reg[7]).arg(tab_reg[8]).arg(tab_reg[9]);

QMessageBox::information(this,"Info",qs);

modbus_close(mb);

modbus_free(mb);

输入以上代码后，在 widget.cpp 文件中输入头文件：



至此，QT 项目工程创建完毕，下面开始搭建硬件环境。

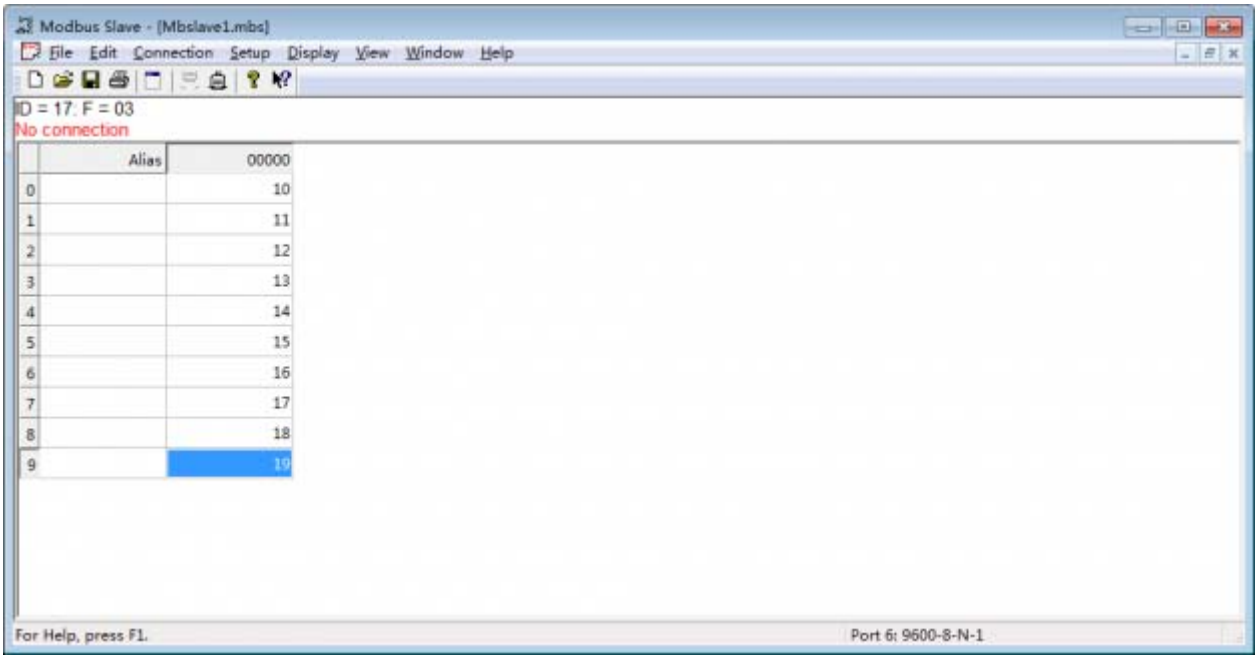
2. 硬件测试环境搭建

为了便于调试，我们采用无线连接，Nanopi M2 配置无线网卡，连接到无线路由，在 PC 机端 Modbus slave 软件配置为 TCP Modbus 监听模式。

进入 Modbus slave 软件主界面，主菜单“Setup”——“Slave Definition...”，弹出对话框，设置如下：

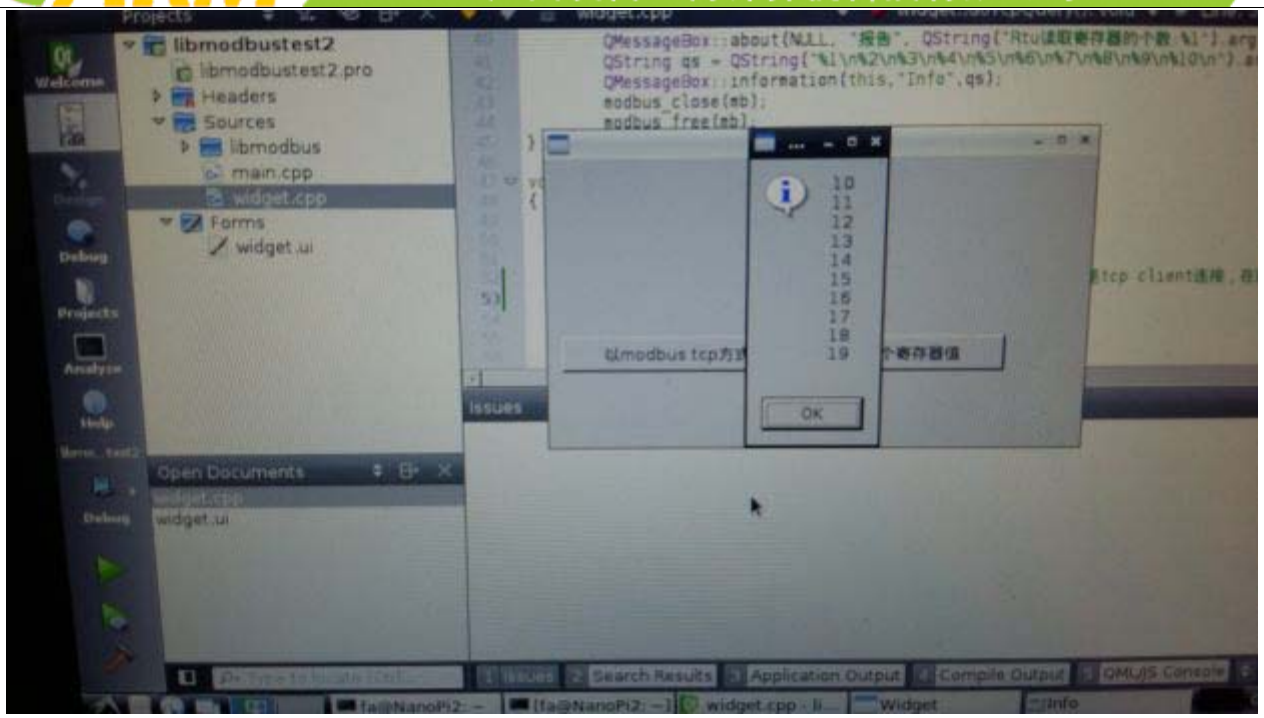
主菜单“Connection”——“Connect...”，在弹出的对话框中设置如下，

然后设置从站的寄存器数据如下图示：



3. 启动测试

Modbus Slave 软件根据上述设置，处于连接状态后，我们启动 QT 测试项目，点击“以 modbus tcp 方式读取地址 1 中前 10 个寄存器值”按钮，通讯结果如图所示：



出现结果“10,11,12,13,14,15,16,17,18,19”的对话框，说明我们的测试成功了。

NanoPi M2 结合 QT、串口、MySQL、Modbus-RTU、Modbus-TCP,已经可是实现功能强大的工业现场数据采集了，只要我们认真去发掘，其实小小的一个卡片式电脑完全可以取代传统工控机，实现传统工控机不具备的功能，我们将继续探索。

技巧十四：基于 gsoap 的 Webservice 开发

来自：zxl_zxl http://bbs.elecfans.com/jishu_801462_1_1.html

NanoPiM2 试用体验之基于 gsoap 的 WebService 开发

Webserive 是不同开发平台、开发语言间数据共享的一种行业标准，我们开发出基于 websevice 的服务 API，这些 API 对外以函数的形式提供接口，而函数本身实现具体的功能，其他开发语言设计的软件可以通过 Webservice 协议远程调用我们开发的服务器端函数提供的不同功能，并将运算结果反馈到调用端，实现不同开发平台、开发语言的数据共享，形成基于 Webservice 协议的 C/S（Client/Server）架构。

对于基于 C/S 架构的数据库软件开发，我们通常不会直接让客户端软件直接通过用户名和密码去连接数据库，而是采用客户端调用服务器端的 Webserice 提供的相关函数，Webservice 提供的函数去在服务器本地执行对数据库的 SQL 操作，这样可以大大提高对数据库操作的安全性，避免了在每一个客户端设置服务器端数据库的登陆名和密码。

本篇试用报告就是要实践如何安装、配置 gsoap，以及如何配置和开发服务器端的 Webservice 程序和客户端的 Webservice 程序。

本篇报告分别在 PC 机端的 Ubuntu 系统下安装和配置 gsaop，交叉编译可以在 Nanopi M2 Debian 系统下运行的 WebService 服务端程序，本地 X86 体系架构下 gcc 编译的可以在 Ubuntu 系统中运行的 WebService 客户端程序。

首先下载 gsoap2.8 源码：http://sourceforge.net/projects/gsoap2/

然后通过 samba 将下载到的 gsoap 源码拷贝到 PC 机端的 Ubuntu 系统和 Nanopi M2 端的 debian 系统下。

1. PC 机 Ubuntu 端

1.1 PC 机端编译安装 gsoap

先安装 gsoap 依赖软件：

命令：sudo apt-get install bison

命令：sudo apt-get install flex

命令：sudo apt-get install libssl-dev

命令：sudo apt-get install openssl

在自己的用户目录下创建 gsoap 目录，以下命令均基于我本人的环境，请根据自己的实际情况替换成你自己的目录。

命令：mkdir /hom/xxx/gsoap (xxx 代表你自己的用户目录)

命令：cp yyy/gsoap_2.8.32.zip/ /hom/xxx/gsoap (yyy 代表你自己的 samba 目录路径)

命令：cd home/xxx/gsoap

命令：unzip gsoap_2.8.32.zip

命令：./configure --prefix=/usr/local/gSOAP

命令：make

命令：sudo make install

（1）将源代码目录下 gsoap 子目录中的 import 目录拷贝到 gSOAP 目录下来；

（2）gSOAP 目录下建一个 env 目录，将 gsoap/samples/link 下的所有文件拷贝到 env 目录，并且生成 envC.o（C 版本要用到），方法是：

cd 到 env 目录，然后：

```
/usr/local/gSOAP/bin/soapcpp2 -penv -cenv.h
g++ -c -I /usr/local/gSOAP/include envC.c
```

1.2 编写 PC 机端的程序

（1）头文件：SmsWBS.h

```
//gsoap ns service name: SmsWBS
          //gsoap ns service style: rpc
          //gsoap ns service namespace: http://192.168.1.88:9000/SmsWBS.wsdl
          //gsoap ns service location: http://192.168.1.88:9000
          //gsoap ns service encoding:encoded
          //gsoap ns schema namespace:urn:SmsWBS
int ns__add(int num1, int num2, int *sum);
```

这个头文件需要注意的是，前面的 // 部分是有意义的，可以在上面修改，如果完全去掉，将会导致生成的中间文件不同，由此会引起要修改 Makefile 文件

（2）Makefile 文件(本地编译)：

比较关键，如果头文件中的 // 部分有修改，要检查是否要修改 Makefile 文件

```
GSOAP_ROOT=/usr/local/gSOAP
WSNAME0=soap
WSNAME=SmsWBS
CC=g++ -g -DWITH_NONAMESPACES
INCLUDE=-I $(GSOAP_ROOT)/include
SERVER_OBJS=$(WSNAME0)C.o $(WSNAME0)Server.o stdsoap2.o
CLIENT_OBJS=$(GSOAP_ROOT)/env/envC.o $(WSNAME0)ClientLib.o stdsoap2.o
```

```
ALL_OBJS=${WSNAME}server.o $(WSNAME0)C.o $(WSNAME0)Server.o ${WSNAME}test.o$(WSNAME0)ClientLib.o
#GSOAP_SRC=/usr/local/gsoap-2.7/gsoap
all:server
${WSNAME}.wsdl:${WSNAME}.h
		$(GSOAP_ROOT)/bin/soapcpp2 -c$(GSOAP_ROOT)/import ${WSNAME}.h
#stdsoap2.o:$(GSOAP_ROOT)/src/stdsoap2.c
stdsoap2.o: stdsoap2.c
		$(CC) -c $? $(INCLUDE)
$(ALL_OBJS):%.o:%.c
		$(CC) -c $? $(INCLUDE)
server:Makefile ${WSNAME}.wsdl ${WSNAME}server.o$(SERVER_OBJS)
		$(CC) ${WSNAME}server.o$(SERVER_OBJS) -o ${WSNAME}server
client:Makefile ${WSNAME}.wsdl${WSNAME}test.c $(ALL_OBJS) stdsoap2.o
		$(CC) ${WSNAME}test.o $(CLIENT_OBJS)-o ${WSNAME}test
clean:
		rm -f *.o *.xml *.a *.wsdl *.nsmap$(WSNAME0)H.h $(WSNAME0)C.c $(WSNAME0)Server.c
$(WSNAME0)Client.c$(WSNAME0)Stub.* $(WSNAME)$(WSNAME)Proxy.* $(WSNAME)$(WSNAME)Object.*$(WSNAME0)ServerLib.c
$(WSNAME0)ClientLib.c $(WSNAME)server ns.xsd$(WSNAME)test
```

（3）服务端程序 SmsWBSserver.c：

```
#include "soapH.h"
#include "SmsWBS.nsmap"
int main(int argc, char **argv)
{
		int m,s;		/* master and slave sockets */
		struct soap SmsWBS_soap;
	soap_init(&SmsWBS_soap);
	soap_set_namespaces(&SmsWBS_soap, namespaces);
	if (argc < 2)
		{
			printf("usage: %s <server_port> \n", argv[0]);
			exit(1);
		}
		else
		{
			m = soap_bind(&SmsWBS_soap, NULL, atoi(argv[1]), 100);
			if (m < 0)
			{
				soap_print_fault(&SmsWBS_soap, stderr);
				exit(-1);
			}
			fprintf(stderr, "Socket connection successful: master socket = %d\n",m);
```

```
                        for (;;)
                        {
                                s = soap_accept(&SmsWBS_soap);
                                if (s < 0)
                                {
                                        soap_print_fault(&SmsWBS_soap, stderr);
                                        exit(-1);
                                }
                                fprintf(stderr, "Socket connection successful: slave socket = %d\n",s);
                                soap_serve(&SmsWBS_soap);
                                soap_end(&SmsWBS_soap);
                        }
            }
            return 0;
}
int ns__add(struct soap *add_soap, intnum1, int num2, int *sum)
{
            *sum = num1 + num2;
            return 0;
}
```

（4）客户端程序 SmsWBStest.c：

```
#include <stdio.h>
#include <stdlib.h>
#include "soapStub.h"
#include "SmsWBS.nsmap"
int add(const char *server, int num1, intnum2, int *sum);
int add(const char *server, int num1, intnum2, int *sum)
{
            struct soap SmsWBS_soap;
            int result = 0;
            soap_init(&SmsWBS_soap);
            soap_set_namespaces(&SmsWBS_soap, namespaces);
            soap_call_ns__add(&SmsWBS_soap, server, "", num1, num2, sum);
            if(SmsWBS_soap.error)
            {
                        printf("soap error:%d, %s, %s ", SmsWBS_soap.error,*soap_faultcode(&SmsWBS_soap),
*soap_faultstring(&SmsWBS_soap));
                        result = SmsWBS_soap.error;
            }
            soap_end(&SmsWBS_soap);
            soap_done(&SmsWBS_soap);
```

```
        return result;
}
int main(int argc, char **argv)
{
            int result = -1;
            char* server="192.168.1.xxx:9000";              //192.168.1.xxx 为 Nanopi M2 Debian 系统下载的 IP 地址
        int num1 = 0;
            int num2 = 0;
            int sum = 0;
        if( argc < 3 )
            {
                    printf("usage: %s num1 num2 \n", argv[0]);
                    exit(0);
        }
        num1 = atoi(argv[1]);
            num2 = atoi(argv[2]);
        result = add(server, num1, num2, &sum);
            if (result != 0)
            {
                    printf("soap err, errcode = %d \n", result);
            }
            else
            {
                    printf("%d + %d = %d \n", num1, num2, sum);
            }
        return 0;
}
```

（5）拷贝 stdsoap2.c 拷贝到当前目录

命令：cp　源码目录/gsoap/stdsoap2.c　./

（6）编译和运行：

前面都已经准备好了，现在只需要：

make　---得到服务端程序 SmsWBSserver

makeclient　---得到客户端程序 SmsWBStest

./SmsWBSserver9000　----运行服务端程序

出来类似下面的显示就表示运行正常

Socket connection successful: master socket= 3

如图所示：

```
vmuser@Linux-host:~/gsoap/test/add$ ls
Makefile            SmsWBS.nsmap     SmsWBS.wsdl       soapServer.c
Makefile~           SmsWBSserver     soapC.c           soapServerLib.c
ns.xsd              SmsWBSserver.c   soapClient.c      soapServer.o
SmsWBS.add.req.xml  SmsWBSserver.o   soapClientLib.c   soapStub.h
SmsWBS.add.res.xml  SmsWBStest       soapClientLib.o   stdsoap2.c
SmsWBS.h            SmsWBStest.c     soapC.o           stdsoap2.o
SmsWBS.h~           SmsWBStest.o     soapH.h
vmuser@Linux-host:~/gsoap/test/add$ ./SmsWBSserver 9000
Socket connection successful: master socket = 3
```

再运行客户端程序：

./SmsWBStest 67 78

显示：

67 + 78 = 145

调用 Web Service 成功，如图示：

```
vmuser@Linux-host:~/gsoap/test/add$ ./SmsWBStest 67 78
67 + 78 = 145
vmuser@Linux-host:~/gsoap/test/add$
```

在浏览器中输入命令：192.168.1.124:9000 （124 为我自己的 IP 地址），如果出现如图所示的界面说明我们运行的 Webservice 服务程序是成功的。

以上说明我们在 Ubuntu 环境中已经搭建好了 gsoap Webservice 开发环境，下面开始在 Nanopi M2 上搭建。

2. Nanopi M2 端

(1)将上述程序拷贝到另外一个文件夹，比如：

```
sudo mkdir add_arm
cp ./add/*    ./add_arm
```

(2)修改 makefile

```
GSOAP_ROOT=/usr/local/gSOAP
WSNAME0=soap
WSNAME=SmsWBS
CC=arm-linux-g++ -g -DWITH_NONAMESPACES
INCLUDE=-I $(GSOAP_ROOT)/include
SERVER_OBJS=$(WSNAME0)C.o $(WSNAME0)Server.o stdsoap2.o
CLIENT_OBJS=$(GSOAP_ROOT)/env/envC.o $(WSNAME0)ClientLib.o stdsoap2.o
ALL_OBJS=${WSNAME}server.o $(WSNAME0)C.o $(WSNAME0)Server.o ${WSNAME}test.o $(WSNAME0)ClientLib.o
#GSOAP_SRC=/usr/local/gsoap-2.7/gsoap
all:server
```

```
${WSNAME}.wsdl:${WSNAME}.h
        $(GSOAP_ROOT)/bin/soapcpp2 -c$(GSOAP_ROOT)/import ${WSNAME}.h
#stdsoap2.o:$(GSOAP_ROOT)/src/stdsoap2.c
stdsoap2.o: stdsoap2.c
        $(CC) -c $? $(INCLUDE)
$(ALL_OBJS):%.o:%.c
        $(CC) -c $? $(INCLUDE)
server:Makefile ${WSNAME}.wsdl${WSNAME}server.o $(SERVER_OBJS)
        $(CC) ${WSNAME}server.o$(SERVER_OBJS) -o ${WSNAME}server
client:Makefile ${WSNAME}.wsdl${WSNAME}test.c $(ALL_OBJS) stdsoap2.o
        $(CC) ${WSNAME}test.o $(CLIENT_OBJS)-o ${WSNAME}test
clean:
        rm -f *.o *.xml *.a *.wsdl *.nsmap$(WSNAME0)H.h $(WSNAME0)C.c $(WSNAME0)Server.c
$(WSNAME0)Client.c$(WSNAME0)Stub.* $(WSNAME)$(WSNAME)Proxy.* $(WSNAME)$(WSNAME)Object.*
$(WSNAME0)ServerLib.c$(WSNAME0)ClientLib.c $(WSNAME)server ns.xsd $(WSNAME)test
```

主要是把编译工具链 g++修改为 arm-linux-g++，以编译可以在 ARM 平台上运行的服务端程序。

然后执行命令：

make ----------生成可以在 Nanopi M2 上运行的服务器端程序 SmsWBSserver

将 SmsWBSserver 拷贝到/nfsshare 目录下，以供 Nanopi M2 系统调用：

sudo cp ./add_arm/SmsWBSserver /nfsshare Ubuntu 虚拟机下执行

sudo mount -t nfs192.168.1.xxx:/nfsshare /mnt -onolock //M2 Debian 系统下执行

cd /mnt

./SmsWBSserver 9000

WebService 服务器端程序启动执行

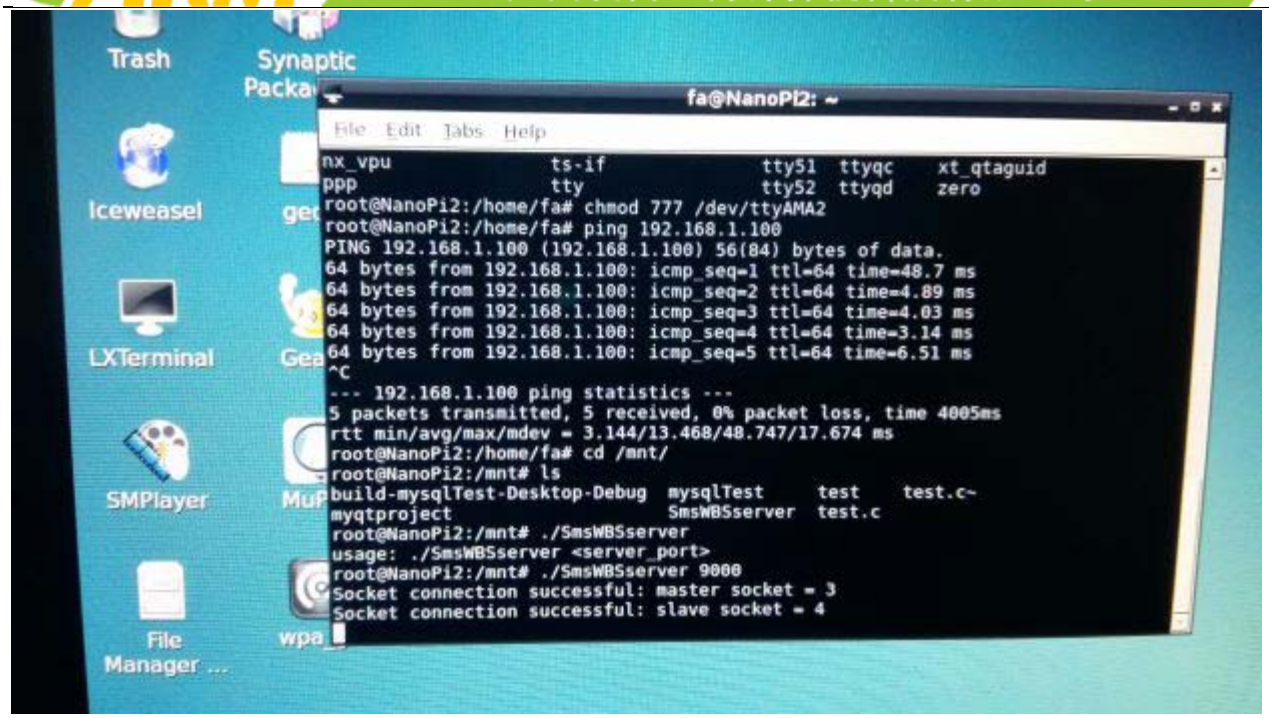
3. Nanopi M2 作为服务器端、PC 机作为客户端的功能测试

我们在 PC 机和 Nanopi M2 上均搭建好了 gsoap 的开发环境，并且都本地启动服务和启动客户程序均成功执行，下面我们让 Nanopi M2 作为服务器端，而 PC 机作为客户端，远程访问一下 Webservice 提供的 Add 函数。

在 Nanopi M2 环境中执行命令：

./SmsWBSserver 9000

此时 Nanopi M2 处于服务器监听状态。



在 Nanopi M2 的命令行终端输入命令：

ifconfig wlan0

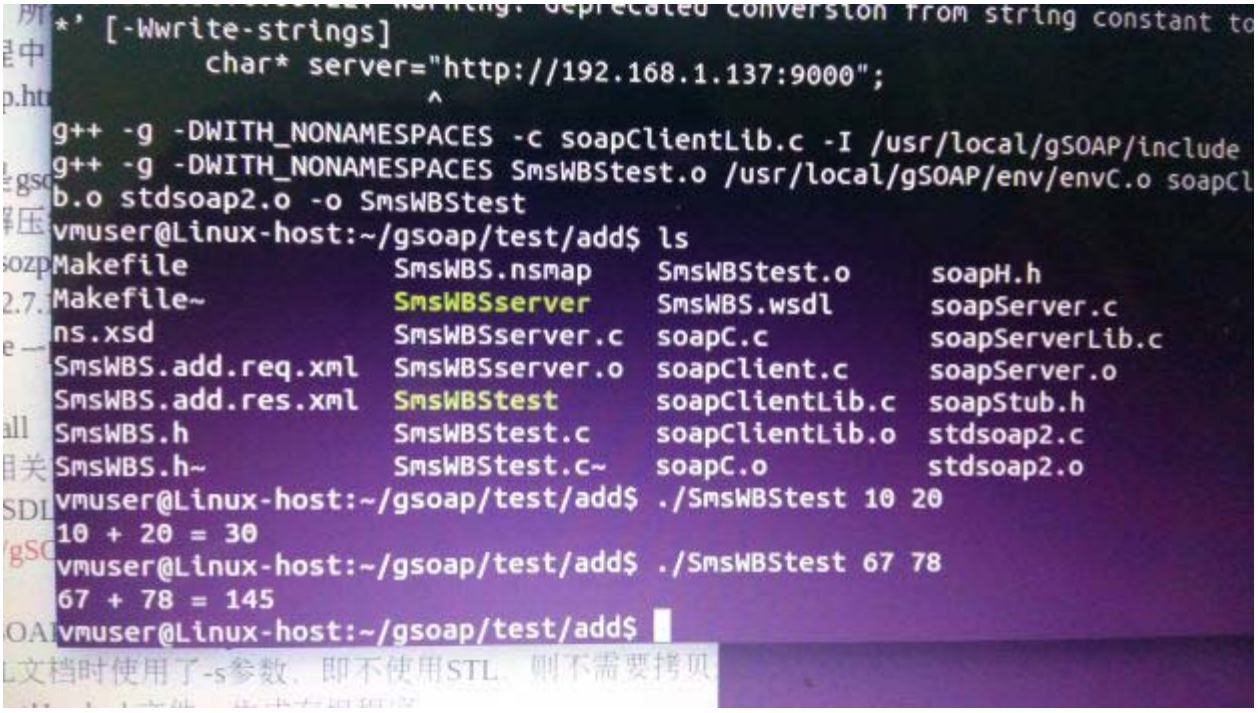
查看 M2 的 IP 地址，然后在 PC 机端的 Ubuntu 系统中，修改 msWBStest.c 中的 char* server="http://localhost:9000";

为 char* server="http://192.168.1.xxx:9000";　　(xxx 为 M2 的 IP 地址)

再次执行 make client

执行./SmsWBStest 67 78

如果返回 67 + 78 = 145



则说明 Nanopi M2 作为 Webservice 服务器，Ubuntu 做为 Webservice 的客户端应用测试成功。

4. 结论

尽管上述的功能测试很简单，只是让服务器的 Webservice 实现了一个加法操作，但是如果我们把这个加法函数修改成读取 Nanopi M2 的 GPIO 端口状态、设置 GPIO 输出，是不是就可以在其他计算机上通过远程调用，实现对 Nanopi M2 的状态读取以及 I/O 命令发送了呢。如果这一步做通了，我们完全可以把 Nanopi M2 作为一个远程数据采集与控制服务器，通过以太网控制。

技巧十五：基于 gsaop 的 Webservice 远程控制 LED 灯

来自：zxl_zxl http://bbs.elecfans.com/jishu_806846_1_1.html

基于 gsaop 的 Webservice 远程控制 LED 灯

在【Nanopi M2 试用】之基于 gsaop 的 Webservice 开发，这篇试用报告中，详细介绍了如何开发运行在 Nanopi M2 上的 Webservice 程序，写了一个简单 Add 加法运算的 Webservice 服务端例程，在 PC 机 ubuntu 中编写了一个远程调用服务端 Add 函数的客户端程序，服务器端和客户端通过 TCP/IP 协议，经无线网络实现了 produc-consume（生产与消费）工作模式的软件开发尝试。

本篇试用报告是上一篇试用的报告的基础上，开发基于 Webserice 的远程控制 LED 亮灭的实验，进而实现在远端，通过网络控制 Nanopi M2 的 I/O 输出。

服务器端为 Nanopi M2，由 M2 承载 Webservice 服务，即 M2 实时监听客户端发出的服务请求，当 M2 根据客户端调用的函数为 on 时，M2 控制 gpio 口输出低电平，使 LED 导通；当 M2 根据客户端调用的函数为 off 时，M2 控制 gpio 口输出高电平，使 LED 关断。服务模型如图所示：

1. 搭建硬件环境

本实验采用 Nanopi M2 中的 GPIOB30 作为控制 LED 灯亮灭的输出引脚，如所示：

Pin#	Name	Pin#	Name
1	SYS_3.3V	2	VDD_5V
3	I2C0_SDA	4	VDD_5V
5	I2C0_SCL	6	DGND
7	GPIOD8/PPM	8	UART3_TXD/GPIOD21
9	DGND	10	UART3_RXD/GPIOD17
11	UART4_TX/GPIOB29	12	GPIOD1/PWM0
13	GPIOB30	14	DGND
15	GPIOB31	16	GPIOC14/PWM2
17	SYS_3.3V	18	GPIOB27
19	SPI0_MOSI/GPIOC31	20	DGND
21	SPI0_MISO/GPIOD0	22	UART4_RX/GPIOB28
23	SPI0_CLK/GPIOC29	24	SPI0_CS/GPIOC30
25	DGND	26	GPIOB26
27	I2C1_SDA	28	I2C1_SCL
29	GPIOC8	30	DGND
31	GPIOC7	32	GPIOC28
33	GPIOC13/PWM1	34	DGND
35	SPI2_MISO/GPIOC11	36	SPI2_CS/GPIOC10

限流电阻采用 300Ω，3.3V 供电，硬件连接如图所示：

2. 设计服务器端例程

2.1 GPIO 配置函数

Nanopi M2 系统已经移植好了基于 sysfs 的 GPIO 驱动程序，编译内核时直接将 GPIO 驱动静态集成到了系统中，根据 sysfs 导出 GPIO 操作接口的方法，编写了 GPIO 设置通用函数如下：

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <dirent.h>
#include <time.h>
#define A    0
#define B    1
#define C    2
#define D    3
#define E    4
/*******************************************************************
gpio 初始化 gpio_init(B,30,0);
* 0    输出
* 1    输入
*******************************************************************/
void gpio_init( char port, char pin, chardirec )
{
    char buf[100];
    int fd=-1,i=0;
```

```
    DIR *dir;

    fd = open( "/sys/class/gpio/export", O_WRONLY );
    snprintf( buf, sizeof( buf ), "/sys/class/gpio/gpio%d", port *32 + pin );
    if( ( dir = opendir( buf ) ) == NULL )
     {
          snprintf( buf, sizeof( buf ), "%d", port * 32 + pin );
          write( fd, buf, sizeof( buf ) );
     }
    closedir( dir );
    close( fd );
    snprintf( buf, sizeof( buf ),"/sys/class/gpio/gpio%d/direction", port * 32 + pin );
    if( fd = open( buf, O_WRONLY ) )
     {
          printf("%d\n",fd);
          if( direc == 0 )
          {
                  i=write( fd, "out", sizeof("out") );
                  printf( "GPIO %d.%d Output\n", port, pin );
          }
          else
          {
               i= write( fd, "in", sizeof("in") );
                  printf( "GPIO %d.%d Input\n", port, pin );
          }
     }
    close( fd );
    printf( "GPIO %d.%d Init Success..\n", port, pin    );
}
/*******************************************************************
gpio 电平配置
实例：gpio_write(B,30,1);
*******************************************************************/
void gpio_write(char port, char pin, charvalue)
{
   char buf[100];
     int fd=-1,i=0;
    snprintf( buf, sizeof( buf ), "/sys/class/gpio/gpio%d/value",port * 32 + pin );
    if( fd = open( buf, O_WRONLY) )
     {
          if( value == 0 )
          {
```

```
                i=write( fd, "0", sizeof("0") );
                printf( "GPIO %d.%d Output low\n", port, pin );
        }
        else
        {
                i=write( fd, "1", sizeof("1") );
                printf( "GPIO %d.%d Output high\n", port, pin );
        }
        close(fd);
    }
}
/*******************************************************************
gpio 输入读取
实例：int zhage=gpio_read(B,30);
0：低电平
1：高电平
*******************************************************************/
int gpio_read( char port, char pin )
{
    int keyValue=1;
    char buf[100];
    int fd=-1;
    snprintf( buf, sizeof( buf ), "/sys/class/gpio/gpio%d/value",port * 32 + pin );
    fd = open( buf, O_RDONLY );
    printf("%d\n",fd);
    if( read( fd, &keyValue, 1) == -1 )
     {
        printf( "PIN%d.%d read error\n", port, pin );
     }
    close(fd);
    return keyValue-48;
}
```

2.2 编写 Webservice 服务器端程序

1）编写头文件 SmsWBS.h

```
//gsoap ns service name: SmsWBS
//gsoap ns service style: rpc
//gsoap ns service namespace:http://192.168.1.124:9000/SmsWBS.wsdl
//gsoap ns service location:http://192.168.1.124:9000
//gsoap ns service encoding: encoded
//gsoap ns schema namespace: urn:SmsWBS
int ns_led_on(int* r);          //返回值为 0 说明 LED 打开成功，返回-1 说明 LED 打开失败
```

```
int ns_led_off(int* r);            //返回值为 0 说明 LED 关闭成功，返回-1 说明 LED 关闭失败
int ns_get_B31(int* r);          //获取 GPIO B31 的引脚输入状态
```

2）编写服务实现函数文件 SmsWBSserver.c

```
#include "soapH.h"
#include "SmsWBS.nsmap"
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <dirent.h>
#include <time.h>
/*******************************************************************/
#define A    0
#define B    1
#define C    2
#define D    3
#define E    4
/*******************************************************************
gpio 初始化 gpio_init(B,30,0);
* 0    输出
* 1    输入
*******************************************************************/
void gpio_init( char port, char pin, chardirec )
{
     char buf[100];
      intfd=-1,i=0;
     DIR *dir;

     fd = open( "/sys/class/gpio/export", O_WRONLY );
     snprintf( buf, sizeof( buf ), "/sys/class/gpio/gpio%d", port *32 + pin );
     if( ( dir = opendir( buf ) ) == NULL )
      {
           snprintf( buf, sizeof( buf ), "%d", port * 32 + pin );
           write( fd, buf, sizeof( buf ) );
      }
     closedir( dir );
     close( fd );
     snprintf( buf, sizeof( buf ),"/sys/class/gpio/gpio%d/direction", port * 32 + pin );
     if( fd = open( buf, O_WRONLY ) )
      {
           printf("%d\n",fd);
           if( direc == 0 )
           {
```

```
                i=write( fd, "out", sizeof("out") );
                printf( "GPIO %d.%d Output\n", port, pin );
        }
        else
        {
            i= write( fd, "in", sizeof("in") );
                printf( "GPIO %d.%dInput\n", port, pin );
        }
    }
    close( fd );
    printf( "GPIO %d.%d Init Success..\n", port, pin    );
}
/*******************************************************************
gpio 电平配置
实例：gpio_write(B,30,1);
*******************************************************************/
void gpio_write(char port, char pin, charvalue)
{
                char buf[100];
    int fd=-1;
    int i=0;
    snprintf( buf, sizeof(buf), "/sys//class/gpio/gpio%d/value",port * 32 + pin );
    if( fd = open( buf, O_WRONLY) )
    {
        if( value == 0 )
        {
                i=write( fd, "0", sizeof("0") );
                printf( "GPIO %d.%d Output low\n", port, pin );
        }
        else
        {
                i=write( fd, "1", sizeof("1") );
                printf( "GPIO %d.%d Output high\n", port, pin );
        }
        close(fd);
    }
}
/*******************************************************************
gpio 输入读取
实例：int zhage=gpio_read(B,30);
0：低电平
1：高电平
```

```
*******************************************************************/
int gpio_read( char port, char pin )
{
     int keyValue=1;
     char buf[100];
     int fd=-1;
     snprintf( buf, sizeof( buf ), "/sys/class/gpio/gpio%d/value",port * 32 + pin );
     fd = open( buf, O_RDONLY );
     printf("%d\n",fd);
     if( read( fd, &keyValue, 1) == -1 )
      {
           printf( "PIN%d.%d read error\n", port, pin );
      }
     close(fd);
     return keyValue-48;
}
int main(int argc, char **argv)
{
           int m, s;                        /* masterand slave sockets */
           struct soap SmsWBS_soap;
           soap_init(&SmsWBS_soap);
           soap_set_namespaces(&SmsWBS_soap, namespaces);

           //初始化 B30 引脚为 GPIO 输出
           gpio_init(B,30,0);
           gpio_write(B,30,1);              //初始化为高电平输出
           //初始化 B31 引脚为 GPIO 输入
           gpio_init(B,31,1);
           if (argc < 2)
           {
                          printf("usage: %s<server_port> \n", argv[0]);
                          exit(1);
           }
           else
           {
                          m = soap_bind(&SmsWBS_soap,NULL, atoi(argv[1]), 100);
                          if (m < 0)
                          {
                                        soap_print_fault(&SmsWBS_soap, stderr);
                                         exit(-1);
                          }
                          fprintf(stderr, "Socketconnection successful: master socket = %d\n", m);
```

```
                        for (;;)
                        {
                                s =soap_accept(&SmsWBS_soap);
                                if (s < 0)
                                {
                                        soap_print_fault(&SmsWBS_soap, stderr);
                                        exit(-1);
                                }
                                fprintf(stderr,"Socket connection successful: slave socket = %d\n", s);
                                soap_serve(&SmsWBS_soap);
                                soap_end(&SmsWBS_soap);
                        }
        }
         return 0;
}
/*
int ns__add(struct soap *add_soap, intnum1, int num2, int *sum)
{
         *sum = num1 + num2;
         return 0;
}
*/
int ns_led_on(struct soap *add_soap,int* r)
{
        gpio_write(B,30,0);                     //输出低电平,点亮 LED 灯
        *r= 0;
        return0;
}
int ns_led_off(struct soap *add_soap,int*r)
{
        gpio_write(B,30,1);                     //输出高电平,熄灭 LED 灯
        *r= 0;
        return0;
}
int ns_get_B31(struct soap *add_soap,int*r)
{
        *r= gpio_read(B,31);                    //返回 B31 引脚状态
        return0;
}
```

3）编写 Makefile 文件

```
GSOAP_ROOT=/usr/local/gSOAP
WSNAME0=soap
```

```
WSNAME=SmsWBS
CC=arm-linux-g++ -g    -DWITH_NONAMESPACES
INCLUDE=-I $(GSOAP_ROOT)/include
SERVER_OBJS=$(WSNAME0)C.o$(WSNAME0)Server.o stdsoap2.o
CLIENT_OBJS=$(GSOAP_ROOT)/env/envC.o$(WSNAME0)ClientLib.o stdsoap2.o
ALL_OBJS=${WSNAME}server.o $(WSNAME0)C.o$(WSNAME0)Server.o ${WSNAME}test.o $(WSNAME0)ClientLib.o
#GSOAP_SRC=/usr/local/gsoap-2.7/gsoap
all:server
${WSNAME}.wsdl:${WSNAME}.h
        $(GSOAP_ROOT)/bin/soapcpp2-c $(GSOAP_ROOT)/import ${WSNAME}.h
#stdsoap2.o:$(GSOAP_ROOT)/src/stdsoap2.c
stdsoap2.o:stdsoap2.c
        $(CC)-c $? $(INCLUDE)
$(ALL_OBJS):%.o:%.c
        $(CC)-c $? $(INCLUDE)
server:Makefile ${WSNAME}.wsdl${WSNAME}server.o $(SERVER_OBJS)
        $(CC)${WSNAME}server.o $(SERVER_OBJS) -o ${WSNAME}server
client:Makefile ${WSNAME}.wsdl${WSNAME}test.c $(ALL_OBJS) stdsoap2.o
        $(CC)${WSNAME}test.o $(CLIENT_OBJS) -o ${WSNAME}test

clean:
        rm-f *.o *.xml *.a *.wsdl *.nsmap $(WSNAME0)H.h $(WSNAME0)C.c $(WSNAME0)Server.c$(WSNAME0)Client.c
$(WSNAME0)Stub.* $(WSNAME)$(WSNAME)Proxy.*$(WSNAME)$(WSNAME)Object.* $(WSNAME0)ServerLib.c
$(WSNAME0)ClientLib.c$(WSNAME)server ns.xsd $(WSNAME)test
```

注意，该程序要在基于 ARM 架构的 Nanopi M2 上运行，一定要确保编译器为 CC=arm-linux-g++

4）执行 make 命令，生成 SmsWBSserver 可执行程序。

5）将 SmsWBSserver 拷贝到/nfsshare 文件夹，以备 Nanopi M2 上电后挂载该网络文件系统，以在 M2 上直接执行/nfsshare/SmsWBSserver 程序。

3. 设计客户端例程

1）将上一节创建代码的文件夹中创建客户端程序 SmsWBStest.c

```
#include <stdio.h>
#include <stdlib.h>
#include "soapStub.h"
#include "SmsWBS.nsmap"
int ns_led_on(const char *server, int *r)
{
        structsoap SmsWBS_soap;
    int result = 0;

    soap_init(&SmsWBS_soap);
    soap_set_namespaces(&SmsWBS_soap, namespaces);
```

```
    soap_call_ns_led_on(&SmsWBS_soap, server, "", r);
    if(SmsWBS_soap.error)
     {
          printf("soap error:%d, %s, %s ", SmsWBS_soap.error,*soap_faultcode(&SmsWBS_soap),
*soap_faultstring(&SmsWBS_soap));
          result = SmsWBS_soap.error;
     }
    soap_end(&SmsWBS_soap);
    soap_done(&SmsWBS_soap);
    return result;
}
int ns_led_off(const char *server, int *r)
{
         structsoap SmsWBS_soap;
    int result = 0;
         soap_init(&SmsWBS_soap);
    soap_set_namespaces(&SmsWBS_soap, namespaces);
    soap_call_ns_led_off(&SmsWBS_soap, server, "", r);
    if(SmsWBS_soap.error)
     {
          printf("soap error:%d, %s, %s ", SmsWBS_soap.error,*soap_faultcode(&SmsWBS_soap),
*soap_faultstring(&SmsWBS_soap));
          result = SmsWBS_soap.error;
     }
    soap_end(&SmsWBS_soap);
    soap_done(&SmsWBS_soap);
    return result;
}
int ns_get_B31(const char *server, int *r)
{
         structsoap SmsWBS_soap;
    int result = 0;
         soap_init(&SmsWBS_soap);
    soap_set_namespaces(&SmsWBS_soap, namespaces);
    soap_call_ns_get_B31(&SmsWBS_soap, server, "", r);
    if(SmsWBS_soap.error)
     {
          printf("soap error:%d, %s, %s ", SmsWBS_soap.error,*soap_faultcode(&SmsWBS_soap),
*soap_faultstring(&SmsWBS_soap));
          result = SmsWBS_soap.error;
     }
    soap_end(&SmsWBS_soap);
```

```
    soap_done(&SmsWBS_soap);
    return result;
}
int main(int argc, char **argv)
{
        int result = -1;
        char* server="http://192.168.1.137:9000";
                intcmd = 0;
                intr = -1;

        if( argc < 2 )
        {
                printf("usage: %s cmdnum\n", argv[0]);
                exit(0);
          }
        cmd = atoi(argv[1]);

        switch(cmd)
        {
                case 0:
                {
                        result =ns_led_on(server, &r);
                        if (result != 0)
                        {
                        printf("soap err, errcode = %d \n", result);
                        }
                        else
                        {
                        printf("led on\n");
                        }
                        break;
                }
                case 1:
                {
                        result =ns_led_off(server, &r);
                        if (result != 0)
                        {
                        printf("soap err, errcode = %d \n", result);
                        }
                        else
                        {
                        printf("led off\n");
```

```
                        }
                        break;
                }
                case 2:
                {
                        result =ns_get_B31(server, &r);
                        if (result != 0)
                        {
                        printf("soap err, errcode = %d \n", result);
                        }
                        else
                        {
                        printf("Pin B31's status is %d \n",r);
                        }
                        break;
                }
                default:
                        printf("The cmd youinput is not supported!\n");
        }
        return 0;
}
```

2）修改 Makefile

由于客户端程序运行在 PC 机中的 ubuntu 系统下，为 X86 架构，所编译器也要选为 X86 架构下的 gcc,只需将上一节中的 CC=arm-linux-g++ 修改为 CC=g++

3）编译程序

顺序执行命令：make

make client

至此，服务器端和客户端的程序就都建立起来了。

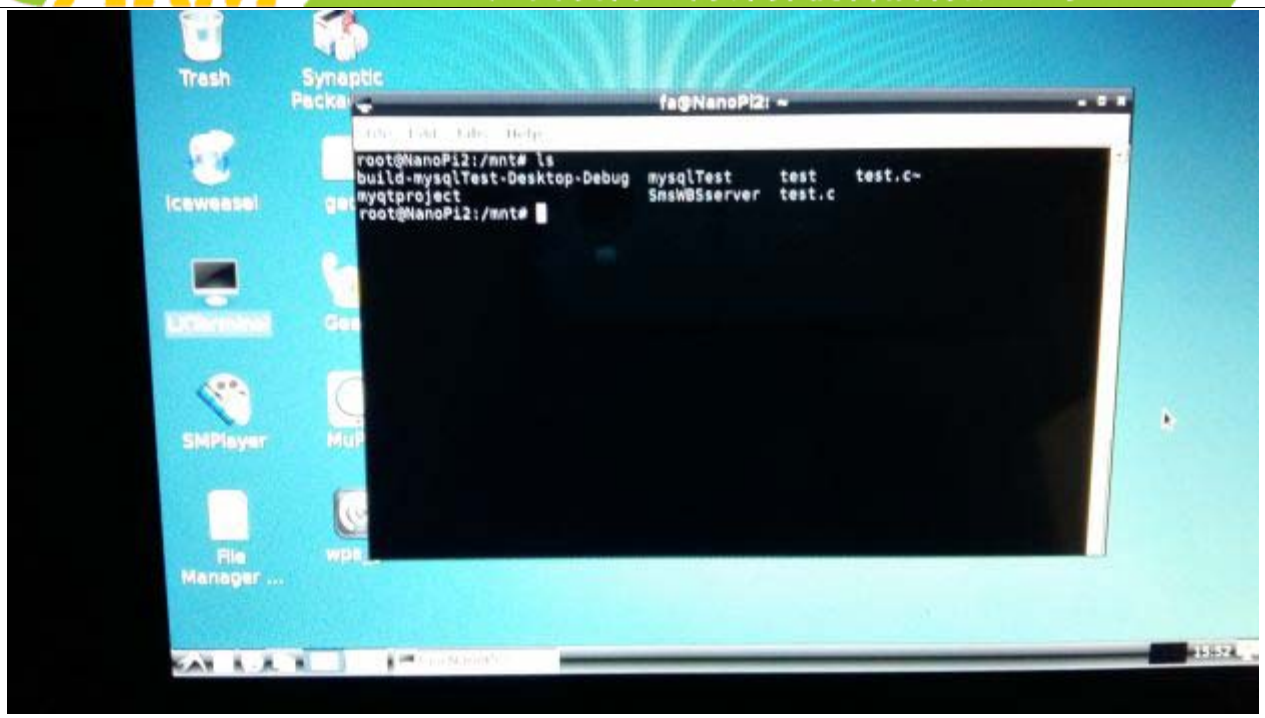
4. 测试

在 Nanopi M2 执行命令：

sudo mount -t nfs 192.168.1.xxx:/nfsshare /mnt -o nolock

ls /mnt

如下图所示：



可以看到 SmsWBSserver 已经出现在 M2 挂载的网络文件系统中了。

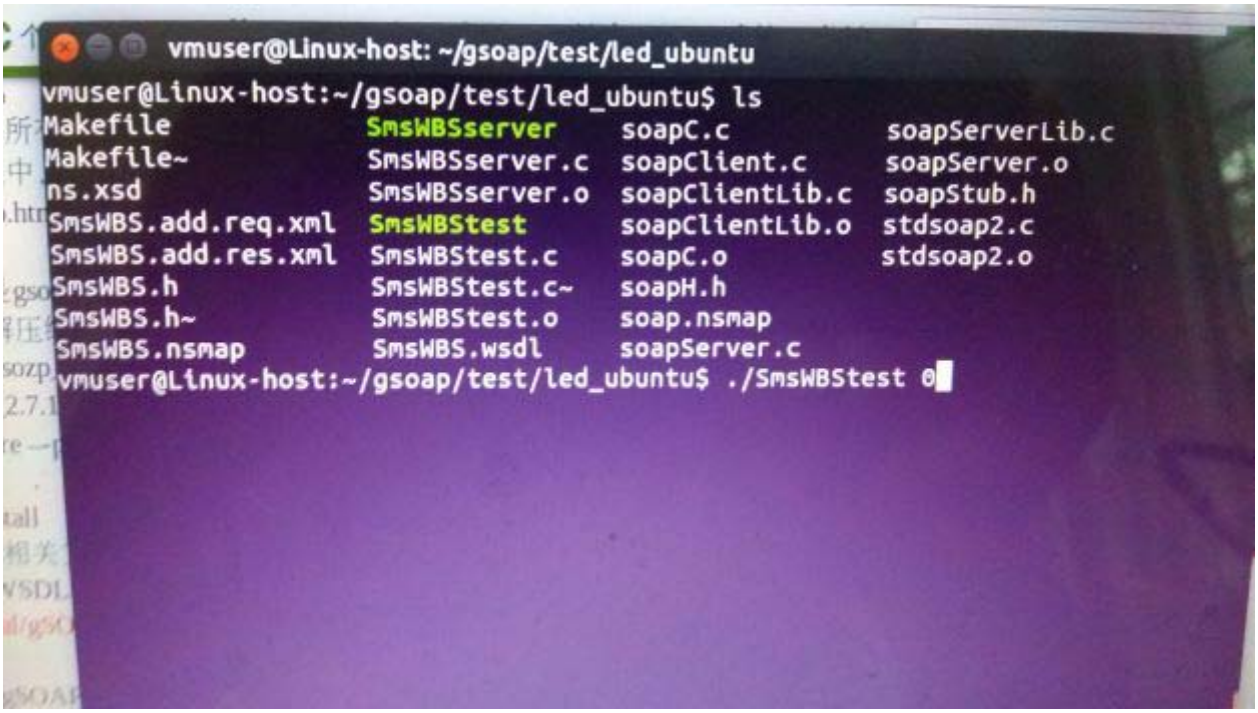
执行命令：./SmsWBSserver 9000

NanopiM2 启动服务监听，如图所示：

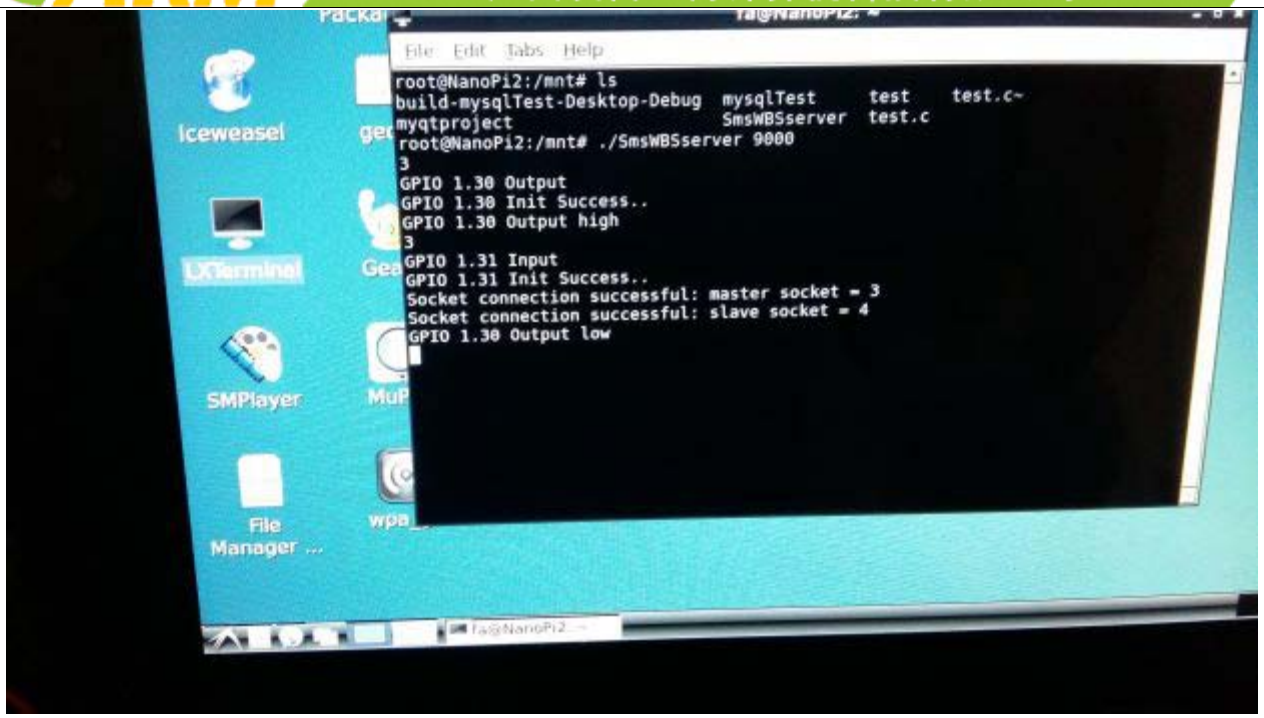
在 PC 机 ubuntu 客户端没有发出命令请求时，面包板上的 LED 灯不亮，如下图所示：

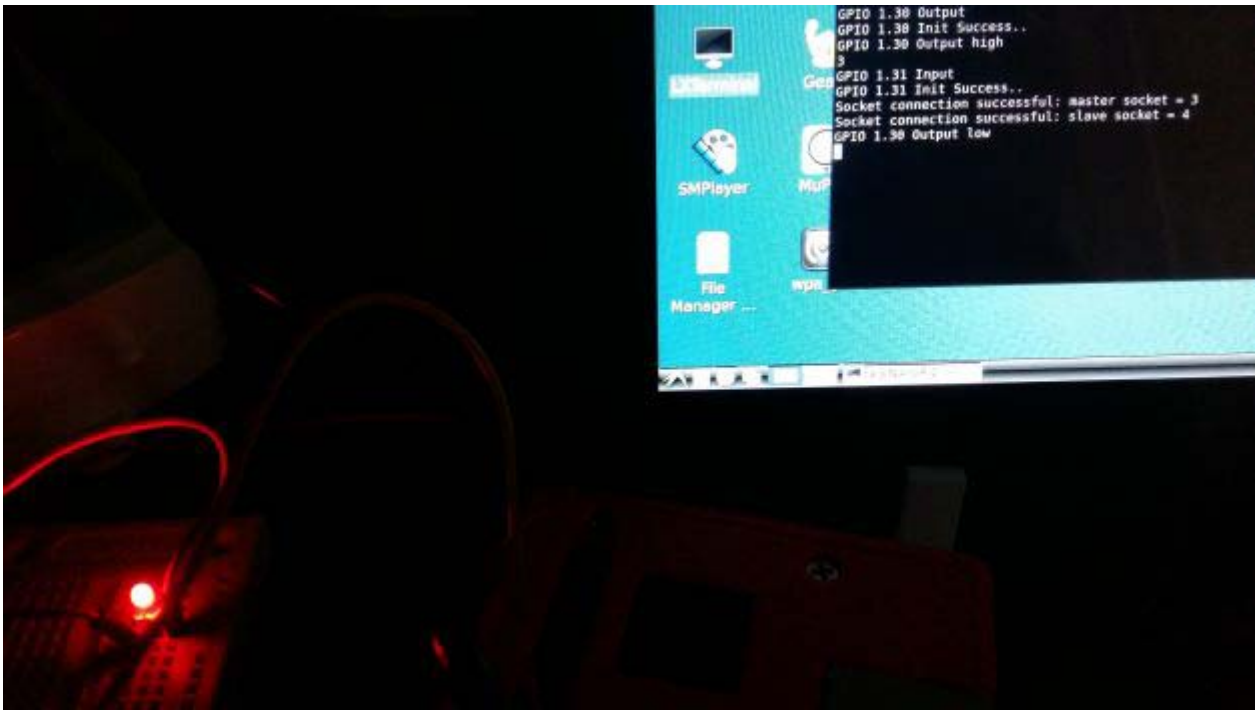
在 PC 的 ubuntu 系统下自行如下命令：

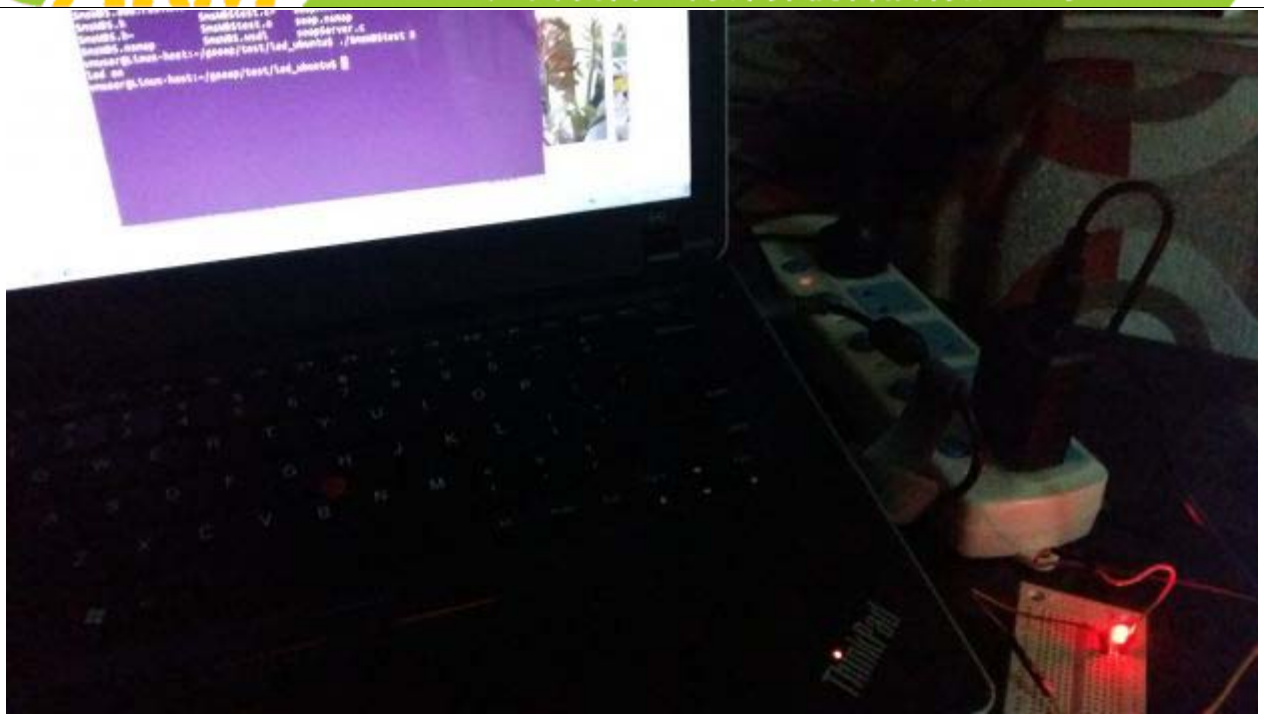
./SmsWBStest 0　　（参数为 0 说明，让 Nanopi M2 的 GPB30 端口输出低电平）



按下 Enter 键后，Nanopi M2 根据客户端的请求，GPB30 输出低电平，点亮 LED 灯，如图所示：


File Edit Tabs Help
root@NanoPi2:/mnt# ls
build-mysqlTest-Desktop-Debug mysqlTest test test.c~
myqtproject SmsWBSserver test.c
root@NanoPi2:/mnt# ./SmsWBSserver 9000
3
GPIO 1.30 Output
GPIO 1.30 Init Success..
GPIO 1.30 Output high
3
GPIO 1.31 Input
GPIO 1.31 Init Success..
Socket connection successful: master socket = 3
Socket connection successful: slave socket = 4
GPIO 1.30 Output low
Iceweasel
LXTerminal
SMPlayer
File Manager ...

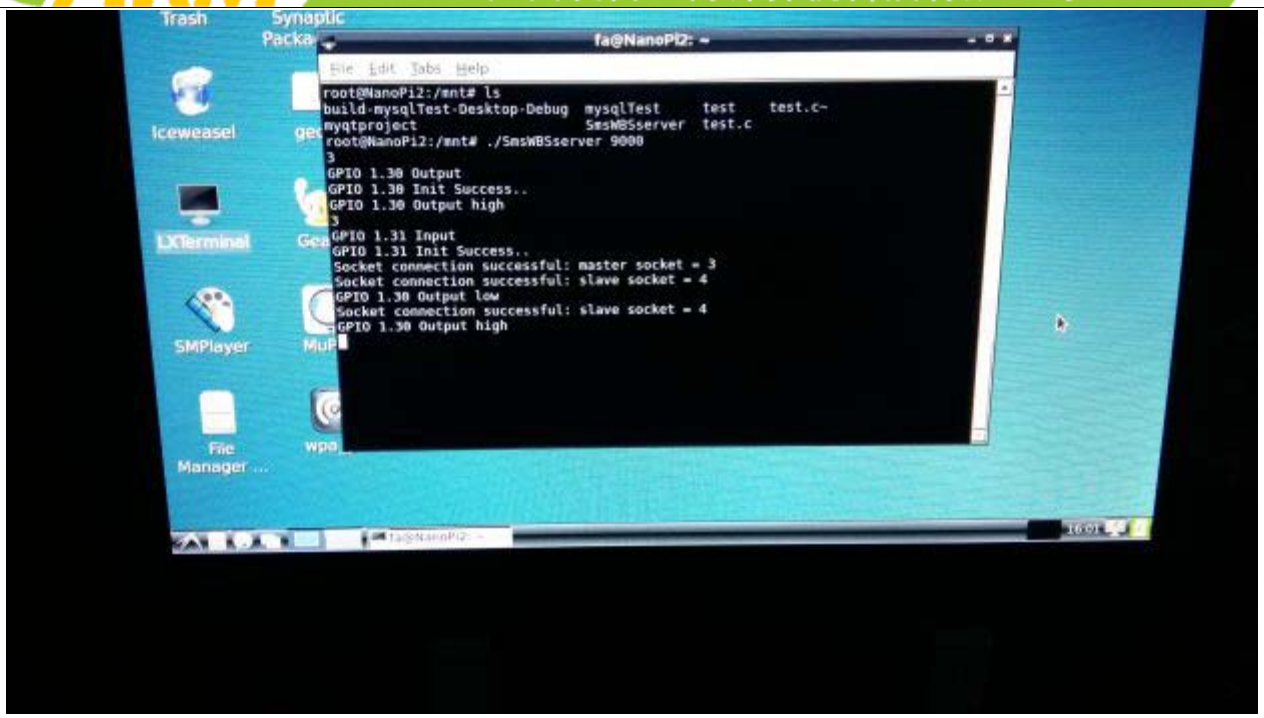

GPIO 1.30 Output
GPIO 1.30 Init Success..
GPIO 1.30 Output high
3
GPIO 1.31 Input
GPIO 1.31 Init Success..
Socket connection successful: master socket = 3
Socket connection successful: slave socket = 4
GPIO 1.30 Output low
SMPlayer
File Manager ...



在 PC 机的 ubuntu 系统输入如下命令熄灭 LED 灯

./SmsWBStest 1

如图所示：



Nanopi M2 上的控制终端显示 GPIO1.30 Output high，输出高电平，熄灭 LED 灯。

5. 总结

我们通过无线网络，采用基于 gsoap 的 Webservice 服务实现了对 Nanopi M2 的远程控制，实验虽然简单，但是充分说明我们除了控制 I/O 输出实现 LED 的亮灭外，我们还是实现对继电器的控制，进而控制车间的电机启停，Nanopi M2 完全可以作为一个车间的小的前端工控机服务器。

技巧十六：OpenCV3.1 搭建

来自：jinyi7016 http://bbs.elecfans.com/jishu_866090_1_1.html

OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库，可以运行在 Linux、Windows 和 Mac OS 操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口

通过 BT 下载工具与 DLAN 的组合，完成了类似于下载宝的功能，接下来就是对于图像处理 OpenCV 的学习了。
网上大多都是 OpenCV2.94 的教程，但是 3.1 的版本已经发布好久了，这个版本相对于原版本也有了一些更新，此次就以 3.1 的版本来学习。

下载 OpenCV 3.1 的源码。
下载地址：http://opencv.org/downloads.html

```
fa@NanoPi2:~$ ls -al opencv-3.1.0.zip
-rwxrwxrwx 1 fa fa 79186131 May 22 16:17 opencv-3.1.0.zip
```

解压源码

系统中默认是没有 unzip 命令的，这里要安装一下

Sudo apt-get install unzip

解压：

Unzip opencv-3.1.0.zip

```
ileMediumLarge.png
 extracting: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/Assets/Tiles/IconicT
ileSmall.png
  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/LocalizedStrings.cs

  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/MainPage.xaml
  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/MainPage.xaml.cs
  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/OpenCVXaml.csproj
   creating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/Properties/
  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/Properties/AppManife
st.xml
  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/Properties/AssemblyI
nfo.cs
  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/Properties/WMAppMani
fest.xml
   creating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/Resources/
  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/Resources/AppResourc
es.Designer.cs
  inflating: opencv-3.1.0/samples/wp8/OpenCVXaml/OpenCVXaml/Resources/AppResourc
es.resx
  inflating: opencv-3.1.0/samples/wp8/readme.txt
fa@NanoPi2:~$
```

安装 cmake 及一些依赖库

sudo apt-get install cmake

sudo apt-get install build-essential

sudo apt-get install libgtk2.0-dev

sudo apt-get install libavcodec-dev

sudo apt-get install libavformat-dev

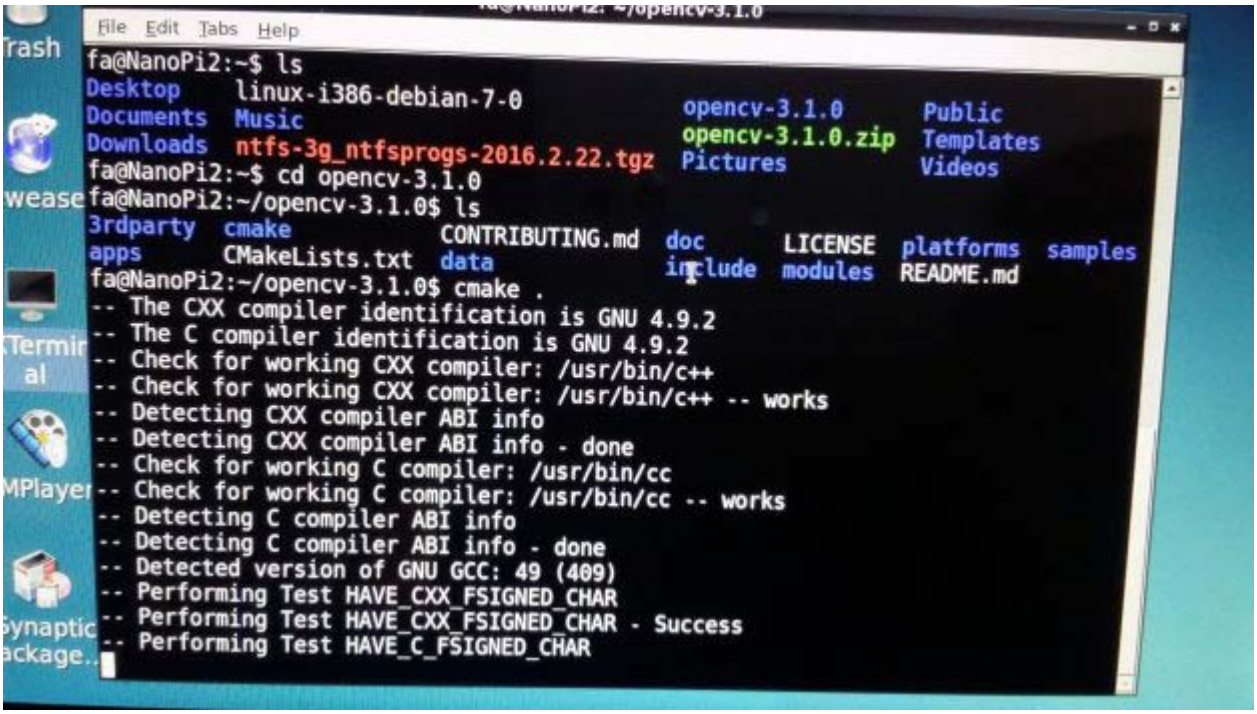
sudo apt-get install libjpeg-dev

sudo apt-get install libtiff5-dev

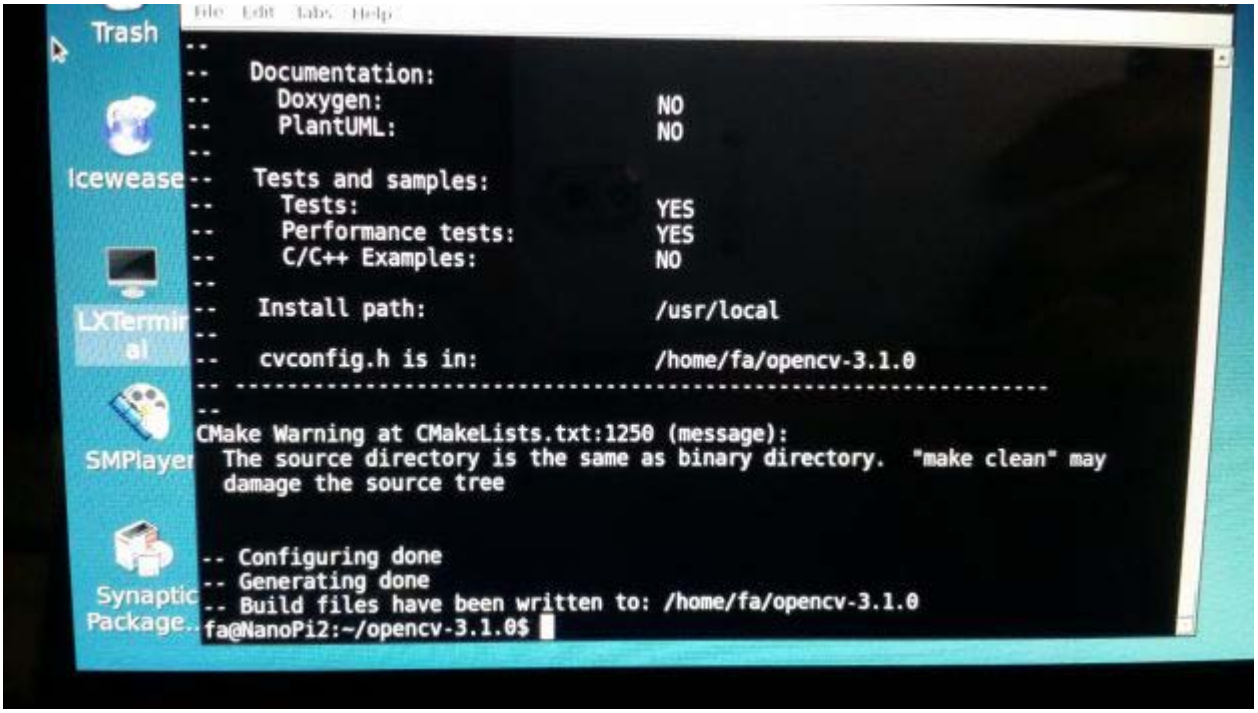
sudo apt-get install libswscale-dev

sudo apt-get install libjasper-dev

进入解压出的源码目录下运行 cmake

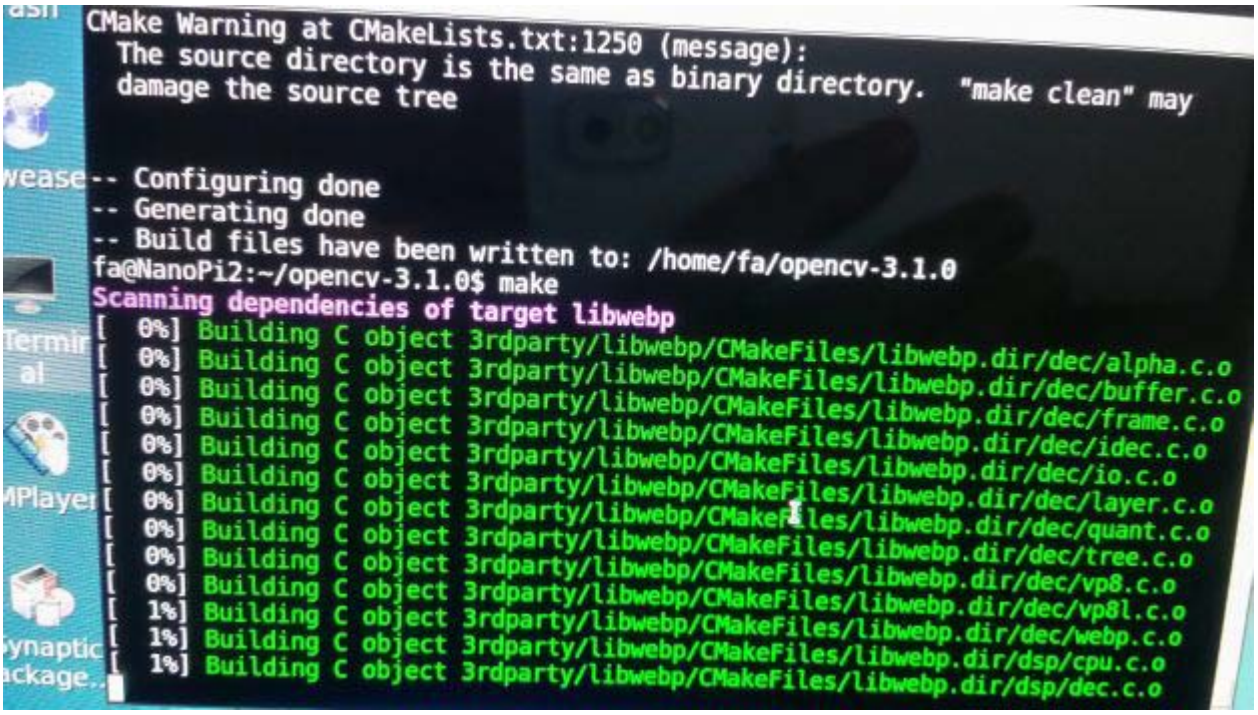


cmake 完成后。

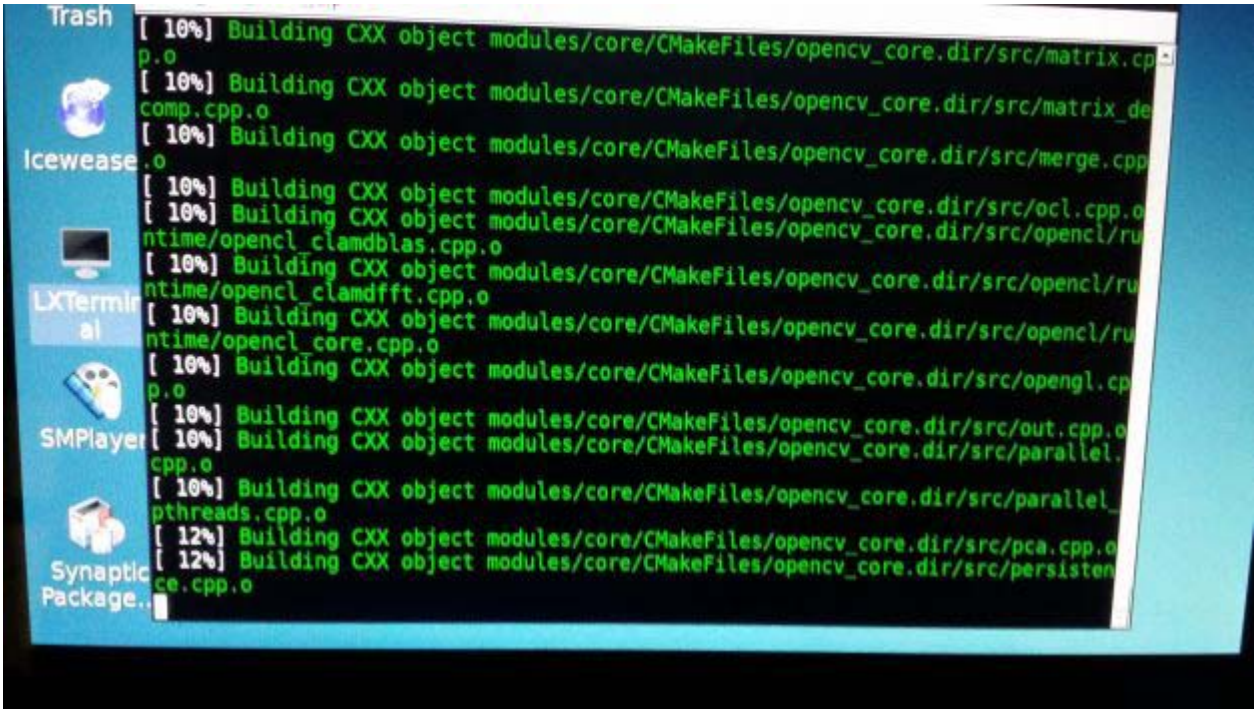


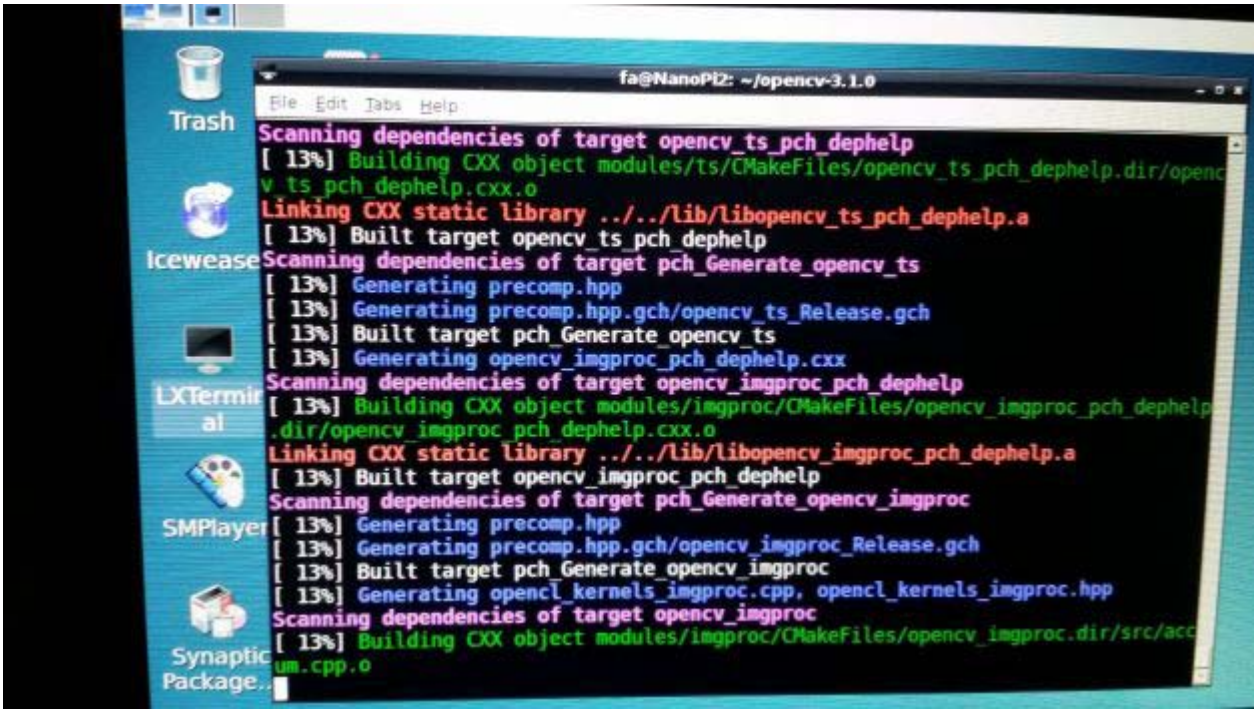
编译：make

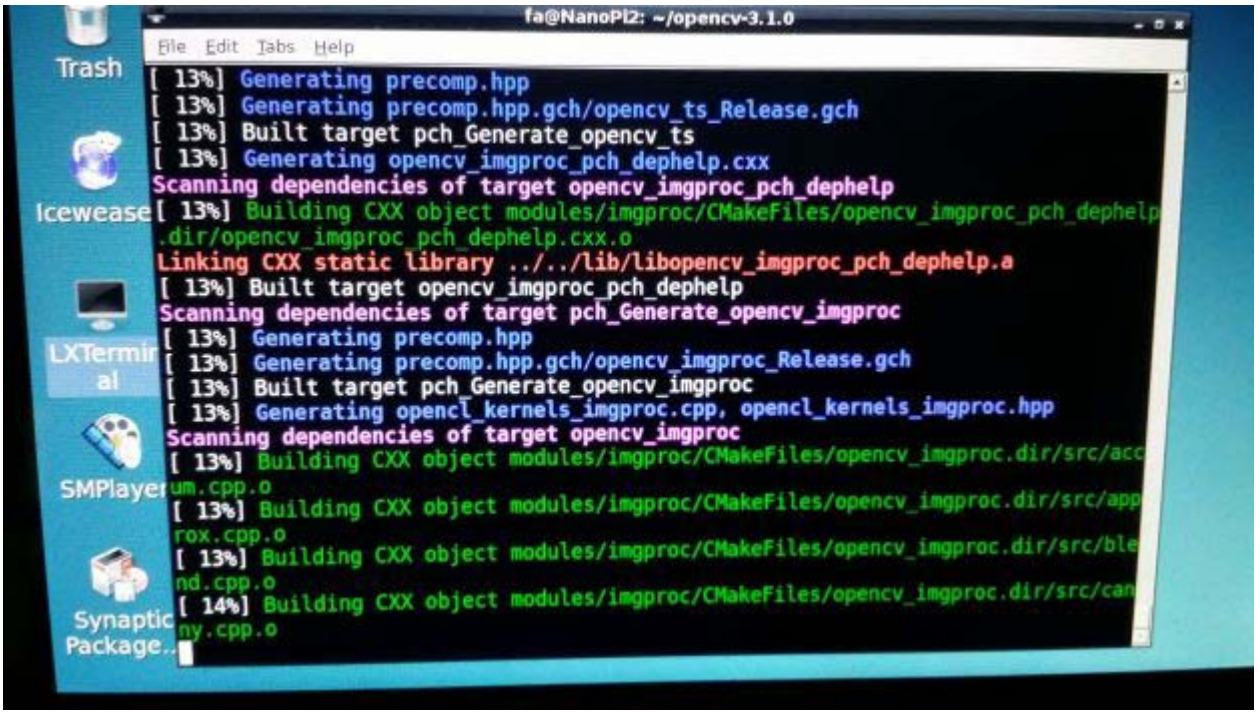
这里的时间会很长，如下刚开始时。



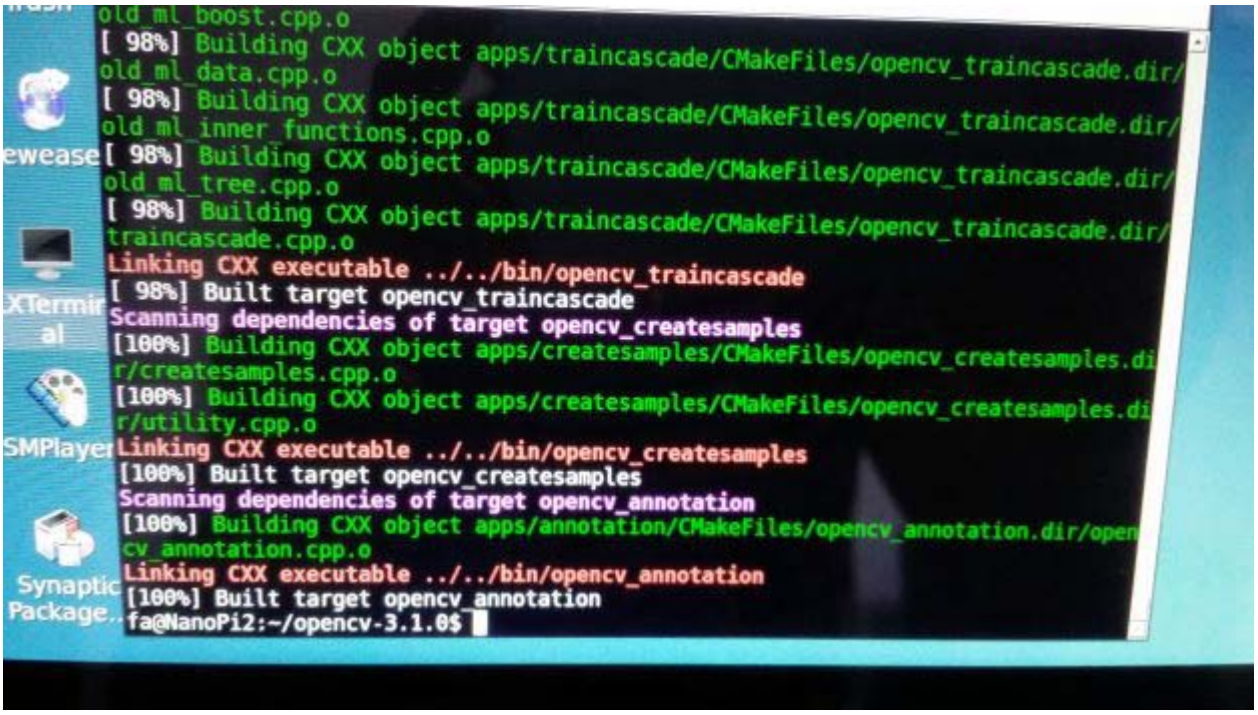
到 10% 时。


Trash
Icewease
LXTermi al
SMPlayer
Synaptic Package..
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/matrix.cp p.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/matrix_de comp.cpp.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/merge.cpp .o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/ocl.cpp.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/opencl/ru ntime/opencl_clamdblas.cpp.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/opencl/ru ntime/opencl_clamdfft.cpp.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/opencl/ru ntime/opencl_core.cpp.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/opengl.cp p.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/out.cpp.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/parallel. cpp.o
[10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/parallel_ pthreads.cpp.o
[12%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/pca.cpp.o
[12%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/persisten ce.cpp.o


Trash
Icewease
LXTermi al
SMPlayer
Synaptic Package..
fa@NanoPi2: ~/opencv-3.1.0
File Edit Tabs Help
Scanning dependencies of target opencv_ts_pch_dephelp
[13%] Building CXX object modules/ts/CMakeFiles/opencv_ts_pch_dephelp.dir/openc v_ts_pch_dephelp.cxx.o
Linking CXX static library ../../lib/libopencv_ts_pch_dephelp.a
[13%] Built target opencv_ts_pch_dephelp
Scanning dependencies of target pch_Generate_opencv_ts
[13%] Generating precomp.hpp
[13%] Generating precomp.hpp.gch/opencv_ts_Release.gch
[13%] Built target pch_Generate_opencv_ts
[13%] Generating opencv_imgproc_pch_dephelp.cxx
Scanning dependencies of target opencv_imgproc_pch_dephelp
[13%] Building CXX object modules/imgproc/CMakeFiles/opencv_imgproc_pch_dephelp .dir/opencv_imgproc_pch_dephelp.cxx.o
Linking CXX static library ../../lib/libopencv_imgproc_pch_dephelp.a
[13%] Built target opencv_imgproc_pch_dephelp
Scanning dependencies of target pch_Generate_opencv_imgproc
[13%] Generating precomp.hpp
[13%] Generating precomp.hpp.gch/opencv_imgproc_Release.gch
[13%] Built target pch_Generate_opencv_imgproc
[13%] Generating opencl_kernels_imgproc.cpp, opencl_kernels_imgproc.hpp
Scanning dependencies of target opencv_imgproc
[13%] Building CXX object modules/imgproc/CMakeFiles/opencv_imgproc.dir/src/acc um.cpp.o

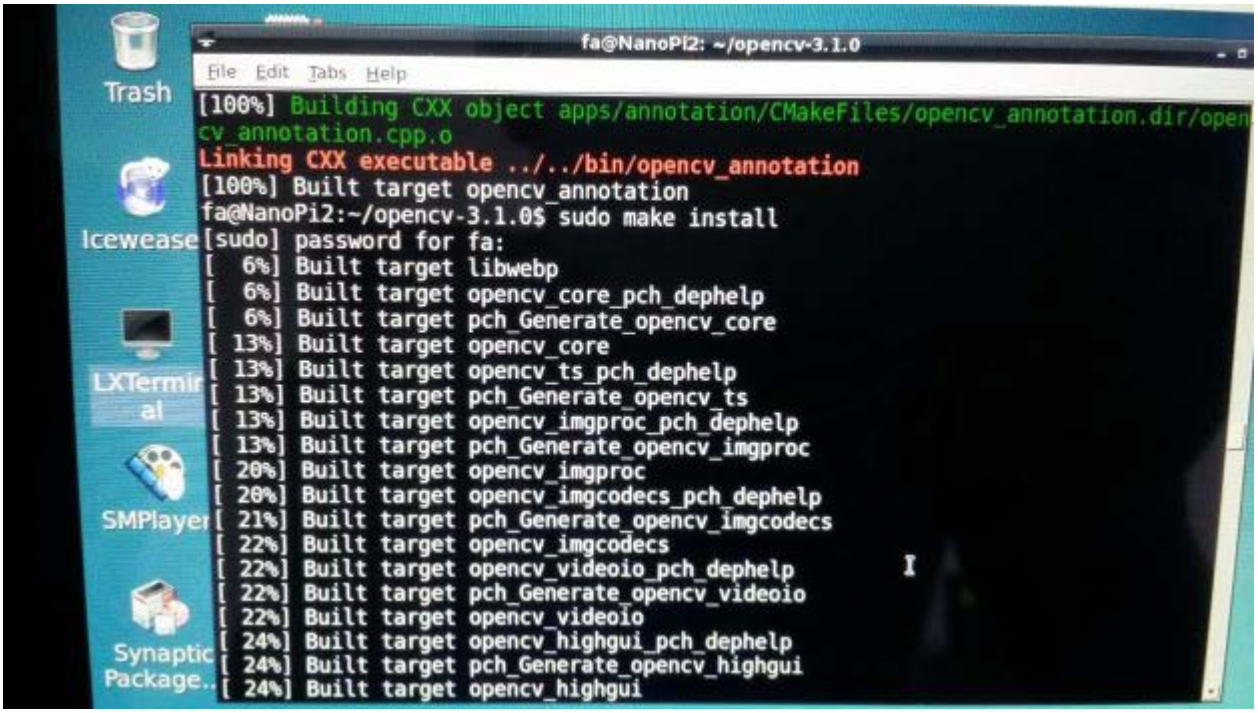


最后花了差不多三个小时，编译完成了。



安装

sudo make install

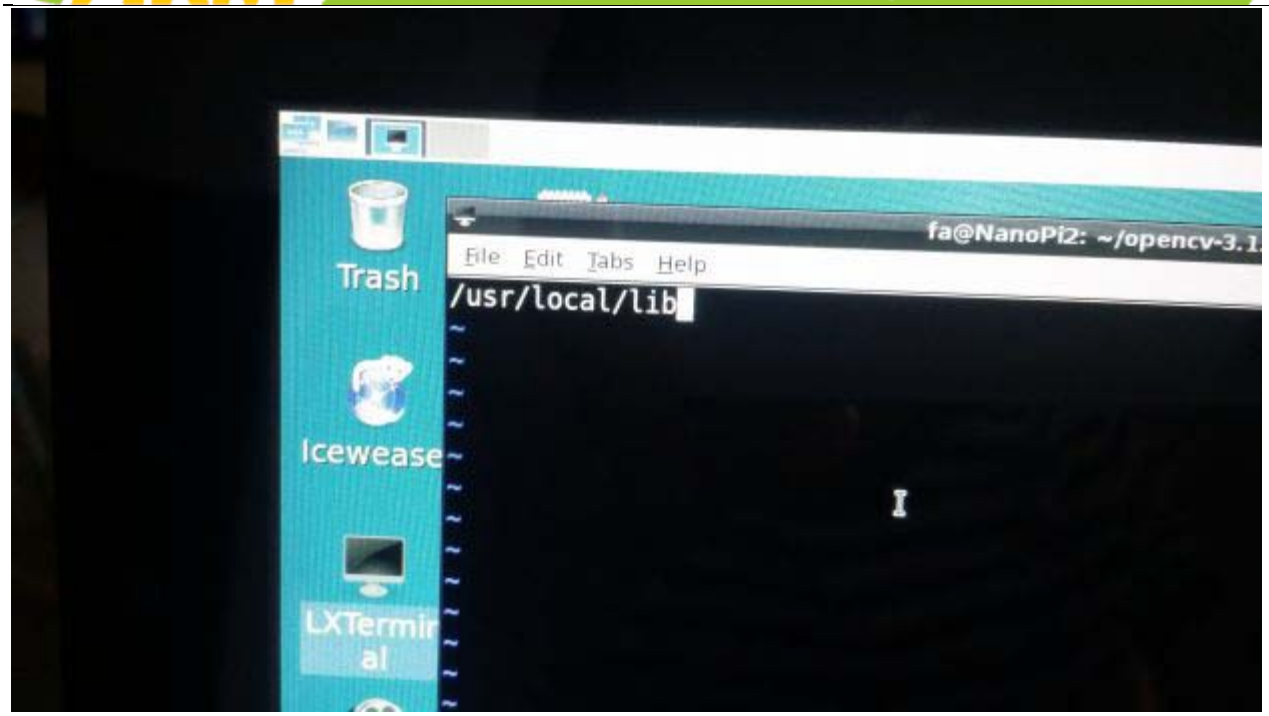


配置

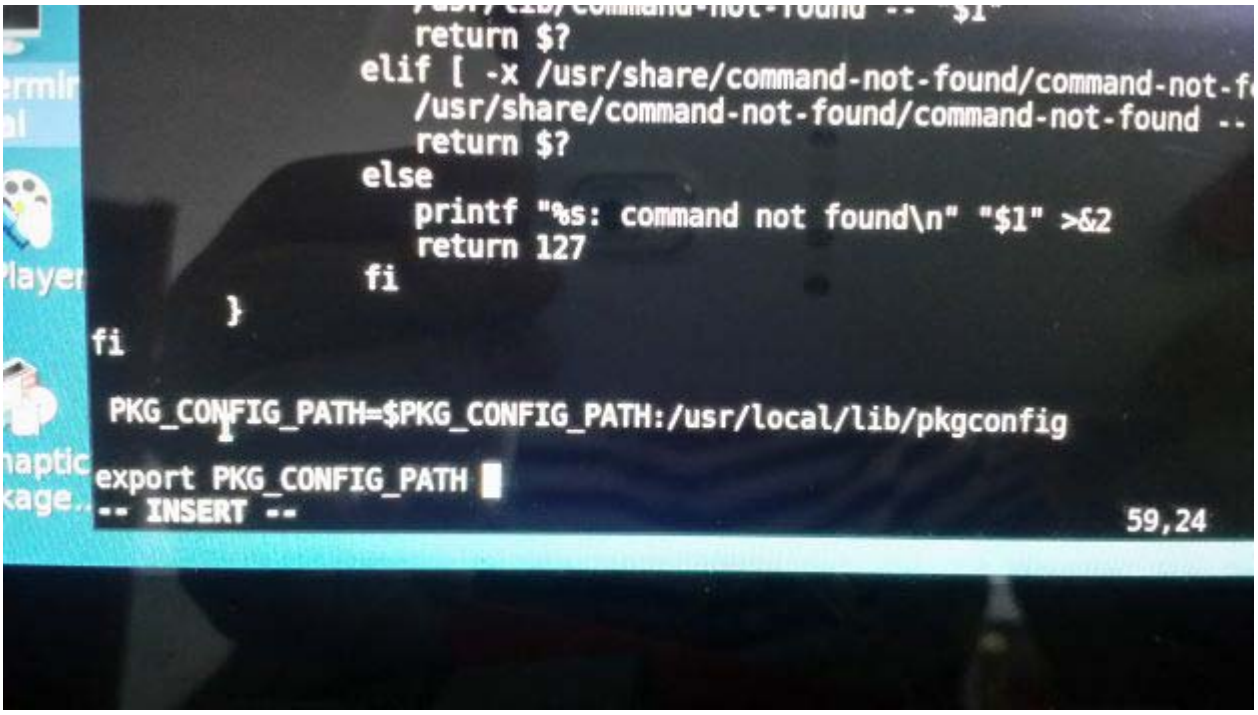
将 opencv 的库加入到路径，从而让系统可以找到

sudo vim /etc/ld.so.conf.d/opencv.conf

末尾加入如下图的语句，保存退出。



sudo ldconfig #使配置生效
sudo vim /etc/bash.bashrc
末尾加入内容如下图。



此次 OpenCV 3.1 的开发环境就搭建好了。

技巧十七：OpenCV3.1 例程编译测试

来自：jinyi7016 http://bbs.elecfans.com/jishu_866091_1_1.html

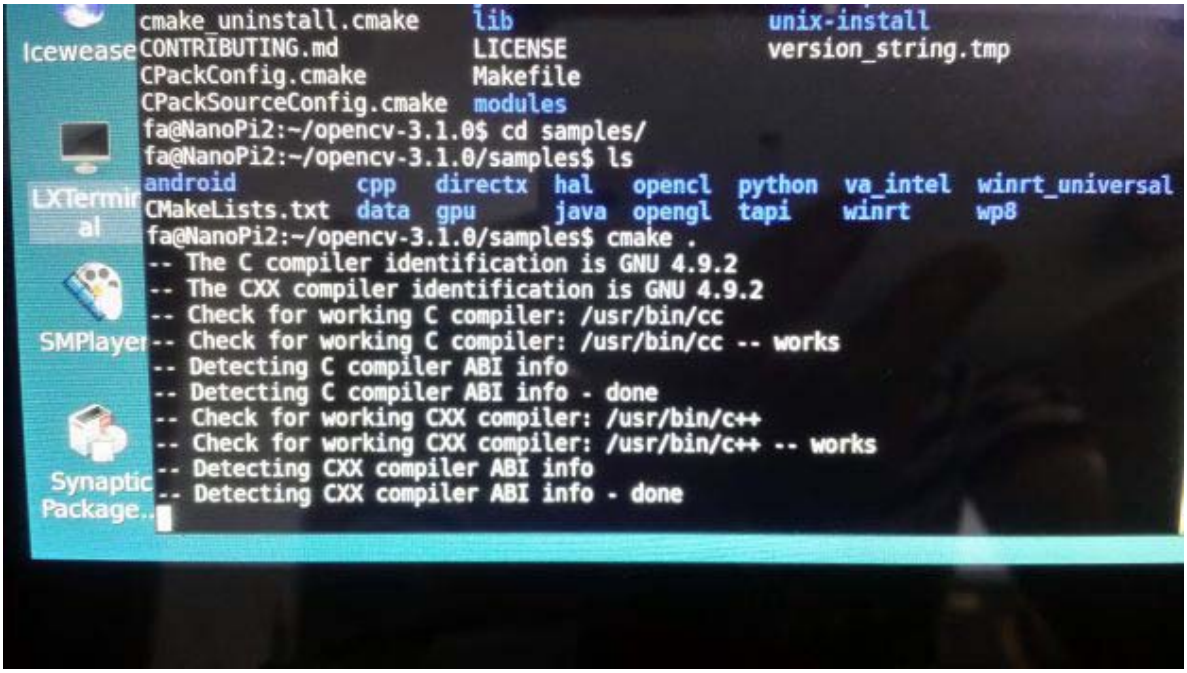
上一帖，我们把 OpenCV 3.1 的开发环境搭建好了。那么就可以开始编程，学习图像处理了，不过在开始学习前，可以运行一下 OpenCV 3.1 自带的一些例程，来了解下，具体的开发流程。

例程的目录也是在下载的源码包里。

目录为：~/opencf3.1.0/samples

进入到目录中。

运行 cmake ，生成 Makefile 与配置文件。



再使用 make 编译所有的例程源码。

```
-- Detecting CXX compiler ABI info - done
-- OpenCL samples require CMakes 3.1+
-- Configuring done
-- Generating done
-- Build files have been written to: /home/fa/opencv-3.1.0/samples
fa@NanoPi2:~/opencv-3.1.0/samples$ make
Scanning dependencies of target cpp-tutorial-pnp_detection
[  0%] Building CXX object cpp/CMakeFiles/cpp-tutorial-pnp_detection.dir/tutorial_code/calib3d/real_time_pose_estimation/src/main_detection.cpp.o
[  1%] Building CXX object cpp/CMakeFiles/cpp-tutorial-pnp_detection.dir/tutorial_code/calib3d/real_time_pose_estimation/src/CsvReader.cpp.o
[  1%] Building CXX object cpp/CMakeFiles/cpp-tutorial-pnp_detection.dir/tutorial_code/calib3d/real_time_pose_estimation/src/CsvWriter.cpp.o
[  2%] Building CXX object cpp/CMakeFiles/cpp-tutorial-pnp_detection.dir/tutorial_code/calib3d/real_time_pose_estimation/src/ModelRegistration.cpp.o
[  3%] Building CXX object cpp/CMakeFiles/cpp-tutorial-pnp_detection.dir/tutorial_code/calib3d/real_time_pose_estimation/src/Mesh.cpp.o
[  3%] Building CXX object cpp/CMakeFiles/cpp-tutorial-pnp_detection.dir/tutorial_code/calib3d/real_time_pose_estimation/src/Model.cpp.o
[  4%] Building CXX object cpp/CMakeFiles/cpp-tutorial-pnp_detection.dir/tutorial_code/calib3d/real_time_pose_estimation/src/PnPProblem.cpp.o
[  5%] Building CXX object cpp/CMakeFiles/cpp-tutorial-pnp_detection.dir/tutorial_code/calib3d/real_time_pose_estimation/src/Utils.cpp.o
```

编译完成 后，可以在目录中看到生成的可执行文件。

```
fa@NanoPi2: ~/opencv-3.1.0
-rwxr-xr-x  1 fa fa  34824 May 22 22:53 cpp-tutorial-how_to_scan_images
-rwxr-xr-x  1 fa fa  59404 May 22 22:53 cpp-tutorial-hull_demo
-rwxr-xr-x  1 fa fa  64816 May 22 22:53 cpp-tutorial-imageSegmentation
-rwxr-xr-x  1 fa fa  32056 May 22 22:53 cpp-tutorial-interoperability_with_OpenCV_1
-rwxr-xr-x  1 fa fa  61900 May 22 22:53 cpp-tutorial-introduction_to_pca
-rwxr-xr-x  1 fa fa  35120 May 22 22:53 cpp-tutorial-introduction_to_svm
-rwxr-xr-x  1 fa fa  14328 May 22 22:53 cpp-tutorial-introduction_windows_vs
-rwxr-xr-x  1 fa fa  15900 May 22 22:51 cpp-tutorial-Laplace_Demo
-rwxr-xr-x  1 fa fa   8204 May 22 22:51 cpp-tutorial-LATCH_match
-rwxr-xr-x  1 fa fa  20936 May 22 22:51 cpp-tutorial-MatchTemplate_Demo
-rwxr-xr-x  1 fa fa  31516 May 22 22:53 cpp-tutorial-mat_mask_operations
-rwxr-xr-x  1 fa fa  67808 May 22 22:53 cpp-tutorial-mat_the_basic_image_container
-rwxr-xr-x  1 fa fa  61156 May 22 22:54 cpp-tutorial-moments_demo
-rwxr-xr-x  1 fa fa  19264 May 22 22:51 cpp-tutorial-Morphology_1
-rwxr-xr-x  1 fa fa  18008 May 22 22:51 cpp-tutorial-Morphology_2
-rwxr-xr-x  1 fa fa  22640 May 22 22:51 cpp-tutorial-Morphology_3
-rwxr-xr-x  1 fa fa  40408 May 22 22:54 cpp-tutorial-non_linear_svms
-rwxr-xr-x  1 fa fa  17240 May 22 22:54 cpp-tutorial-npr_demo
-rwxr-xr-x  1 fa fa  27572 May 22 22:54 cpp-tutorial-objectDetection
-rwxr-xr-x  1 fa fa  27592 May 22 22:54 cpp-tutorial-objectDetection2
```

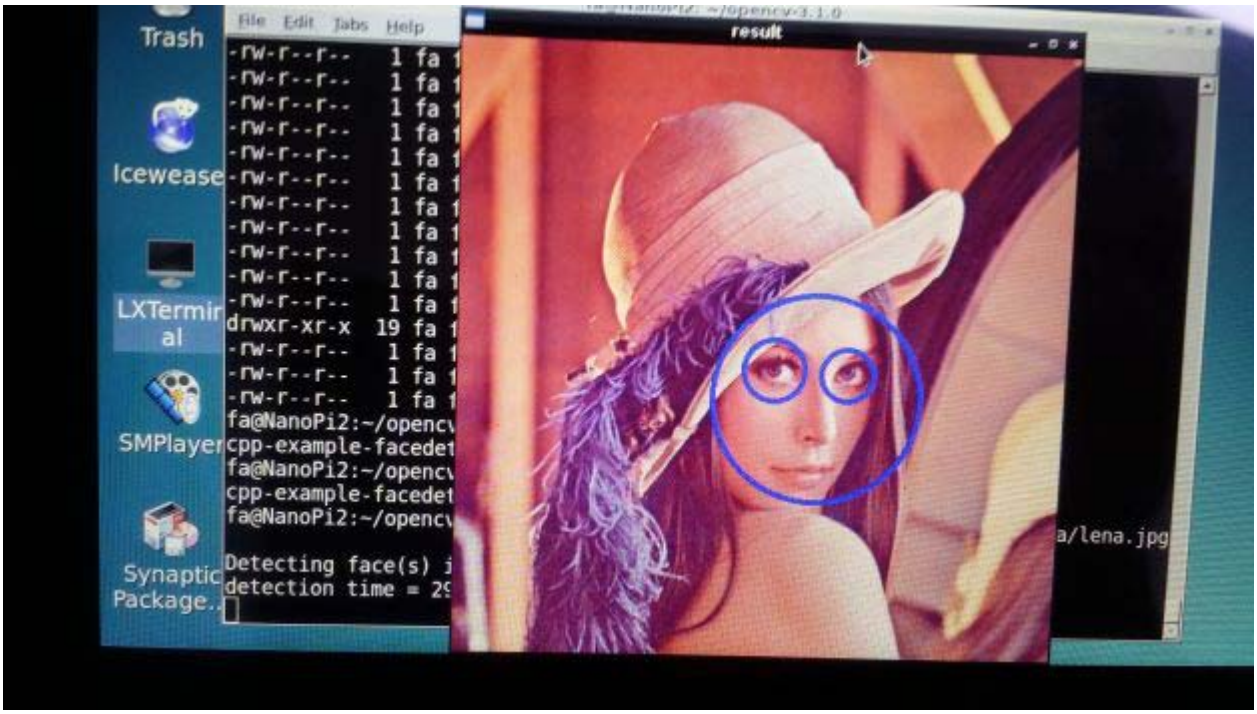
我们就以人脸识别的例程开始吧。

使用到的可执行文件为 cpp-example-facedetect。

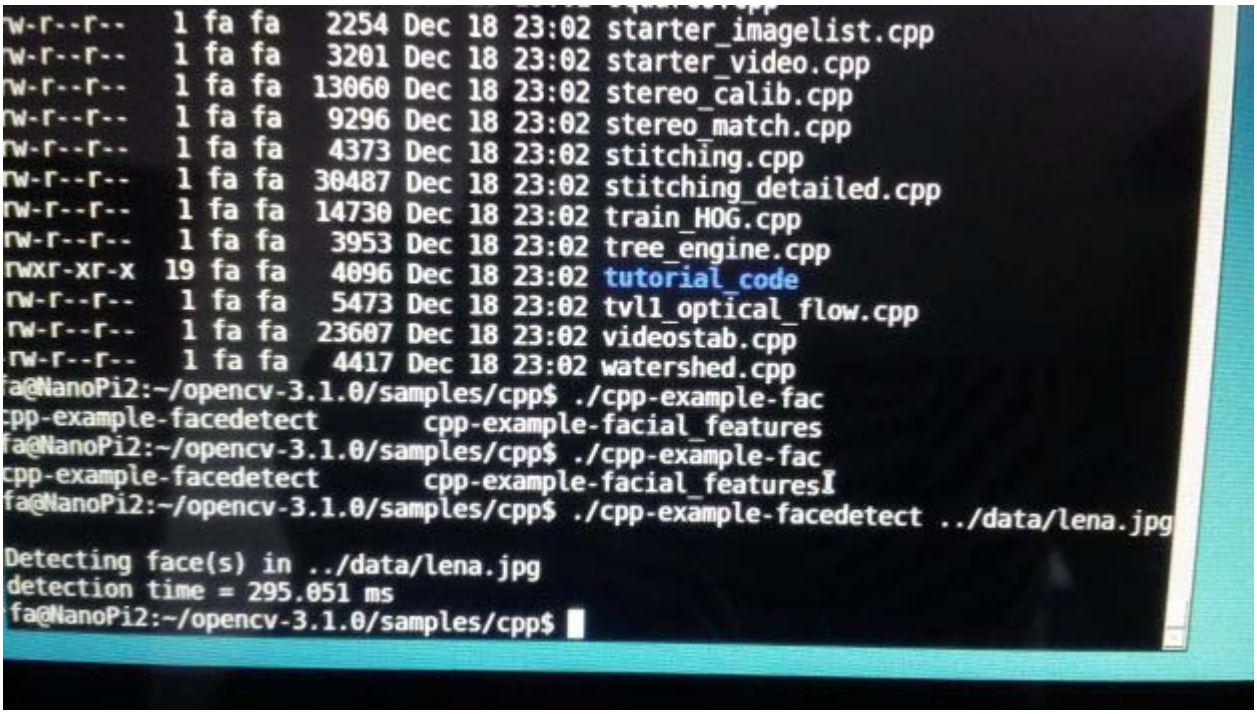
要识别的图片是../data/lena.jpg。

那么，运行如下的命令。

./cpp-example-facedetect ../data/lena.jpg



同时在终端可以看到使用的时间是 295.051ms



以此类推，大家可以试试其他的例程。

技巧十八：POLL 机制

来自：@kidsure http://bbs.elecfans.com/jishu_702777_1_1.html

POLL 机制在我看来啊，有点像刚学单片机时候的定时器（可能内部差不多），设置定时的时长，然后让这个应用程序或者硬件休眠，毕竟没有休眠的话，即使一个很简单的程序也能让 cpu 占用率飙到很高。我今天主要还是讲 POLL 机制在低版本内核中的实现。下一个帖子会以 IO 中断的方式把 POLL 机制加到驱动和应用程序中去。好了，来看看程序和内核把。

首先我们在驱动程序中调用 POLL，其实就是在 file_operations 结构体中，加一个成员.poll

```
static struct file_operations sencod_drv_fops = {
    .owner    =    THIS_MODULE,    /* 这是一个宏，推向编译模块时自动创建的__this_module 变量 */
    .open     =    third_drv_open,
        .read        =        third_drv_read,
        .release =    third_drv_close,
        .poll        =    third_drv_poll,
};
```

一般这样加到这个结构体中，系统就认为你这个驱动是支持 POLL 机制的了。

```
static unsigned int third_drv_poll(struct file *file, struct poll_table_struct *wait)
{
```

```
return 0;
}
```

然后呢，我们要写一个这样东西，让系统能找到 poll 对应的操作函数，这里先返回 0，之后再具体往里面写休眠的程序。

我们来看看应用层和系统里是怎么找到驱动中的 POLL 的。
在应用程序中我们可以写一个很简单的测试。

```
ret = poll(fds, 1, 5000);
                        if (ret == 0)
                        {
                                printf("time out\n");
                        }
```

大概就是让他调用驱动中的 POLL，时钟到了 5000 的时候这个函数的返回值就会等于 0，时间到了之后我们就打印一句 time out。然后我们来从上层慢慢向下看哈。
内核中我们先找到 sys_poll,然后他会执行 do_sys_poll(.....,timeout_jiffies),前面的参数咱们先不看，后面是 poll 的延时时间，在往里面看，有个 poll_initwait(&table),函数里面主要是构造一个结构体 table，然后 init_poll_funcptr(&pwq->pt,__pollwait);就相当于让 table->qproc =__pollwait
构造好了这个结构体之后，要把他加到内核里面去就是执行 do_poll(nfds,head,&table,timeout)，里面就是各种累加

```
if(do_pollfd(pfd,pt))
{
//mask = file->f_op->poll(file ,pwait);          返回值 mask
//return mask;                                                                    这
些是 do_pollfd 的具体实现
//__pollwait(...,&button_waitq,...);                    把进程挂到队列中
count++;
pt = NULL;
}
if(count || !*timeout || signal_pending(current))
break;
```

这个大概意思就是 break 的条件，count 非 0，超时，或者有信号在等待处理的时候。否则呢，就正常休眠。

```
__timeout = schedule_timeout(__timeout);
```

如果休眠时间到了_timeout 的时间，就从 do_poll 函数重新执行，重新执行之后会把 timeout 清零然后再把我们驱动里的当前进程挂到队列里面去，当有中断发生的时候，我们的中断程序再把这个进程从队列中唤醒。
内核中的实现过程基本就是这样，我们来看看驱动中怎么写我们的 static unsigned int third_drv_poll(struct file *file, struct poll_table_struct *wait)
我们在里面用 poll_wait(file,&button_waitq,wait);把当前的进程挂到队列里面去，像我们刚刚看内核里面写的一样，它不会立即休眠，而是等判断完 count 是否等于 0 之后进入休眠。

我们的 poll 函数里初始化 unsigned int mask = 0;

```
if (ev_press)                                //ev_press 由中断程序置一，由 read 函数清零
            mask |= POLLIN | POLLRDNORM;
      return mask;
```

如果发生中断，mask 不等于 0，则唤醒休眠进程。

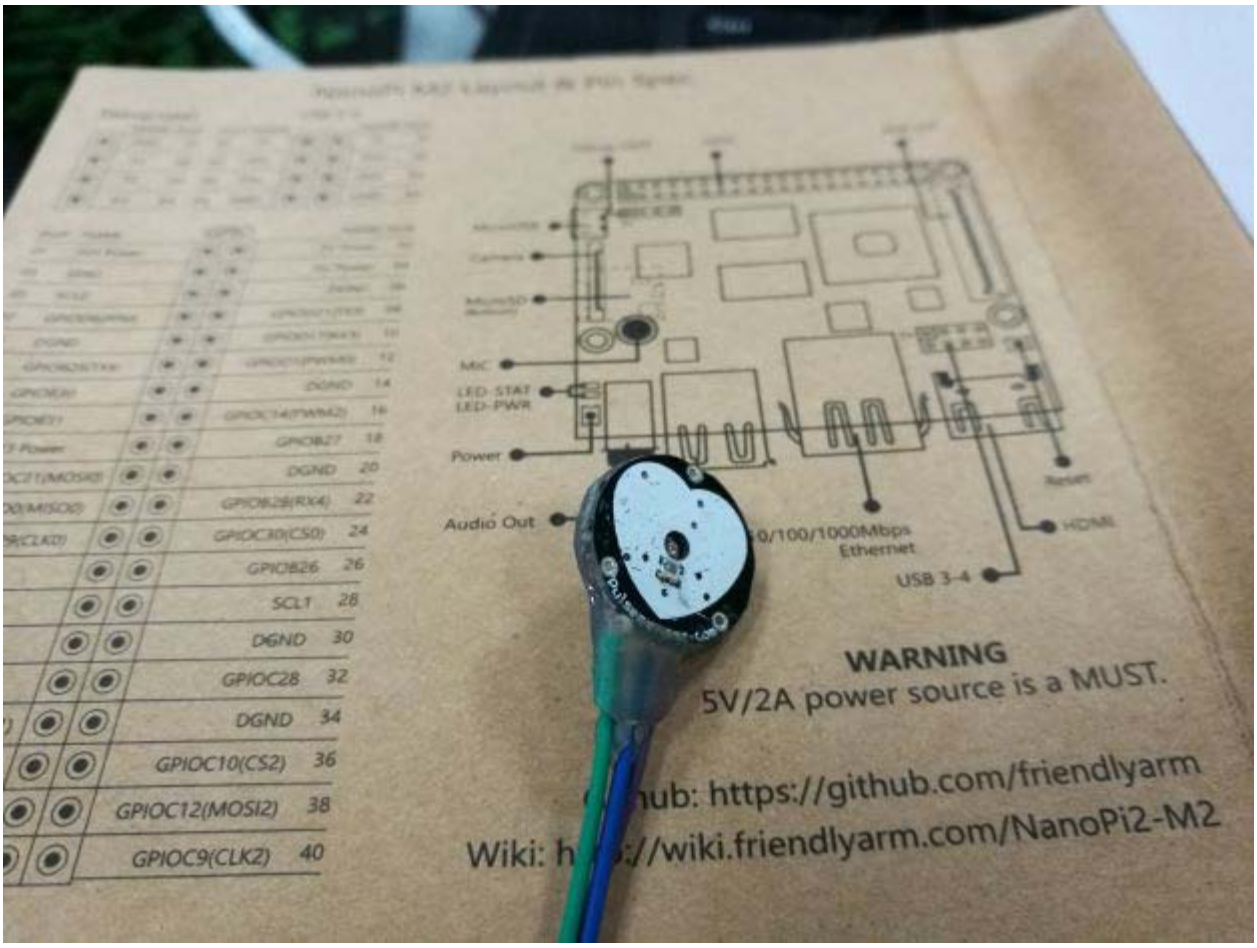
大概说了一下 POLL 机制，其实我对这个也不是很明白，也是半瓶醋就给别人瞎讲了，如果哪里说得不对，还请各位指正批评。下一个帖子，我本来准备用高电平中断 IO 的方式写一个 GPIO 中断加入 POLL 机制。

技巧十九：ad 转换，I2c

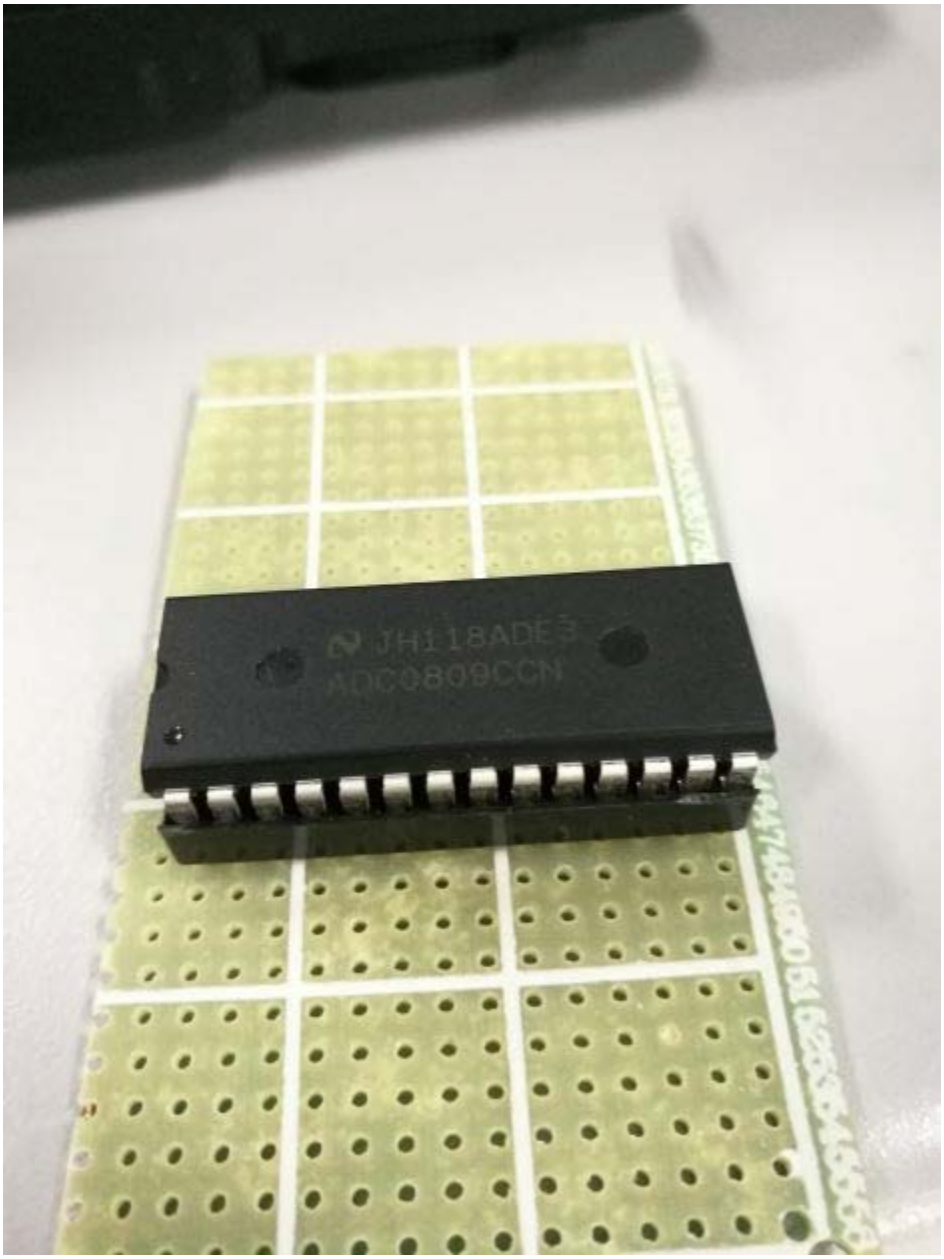
来自 kidsure http://bbs.elecfans.com/jishu_747126_1_1.html

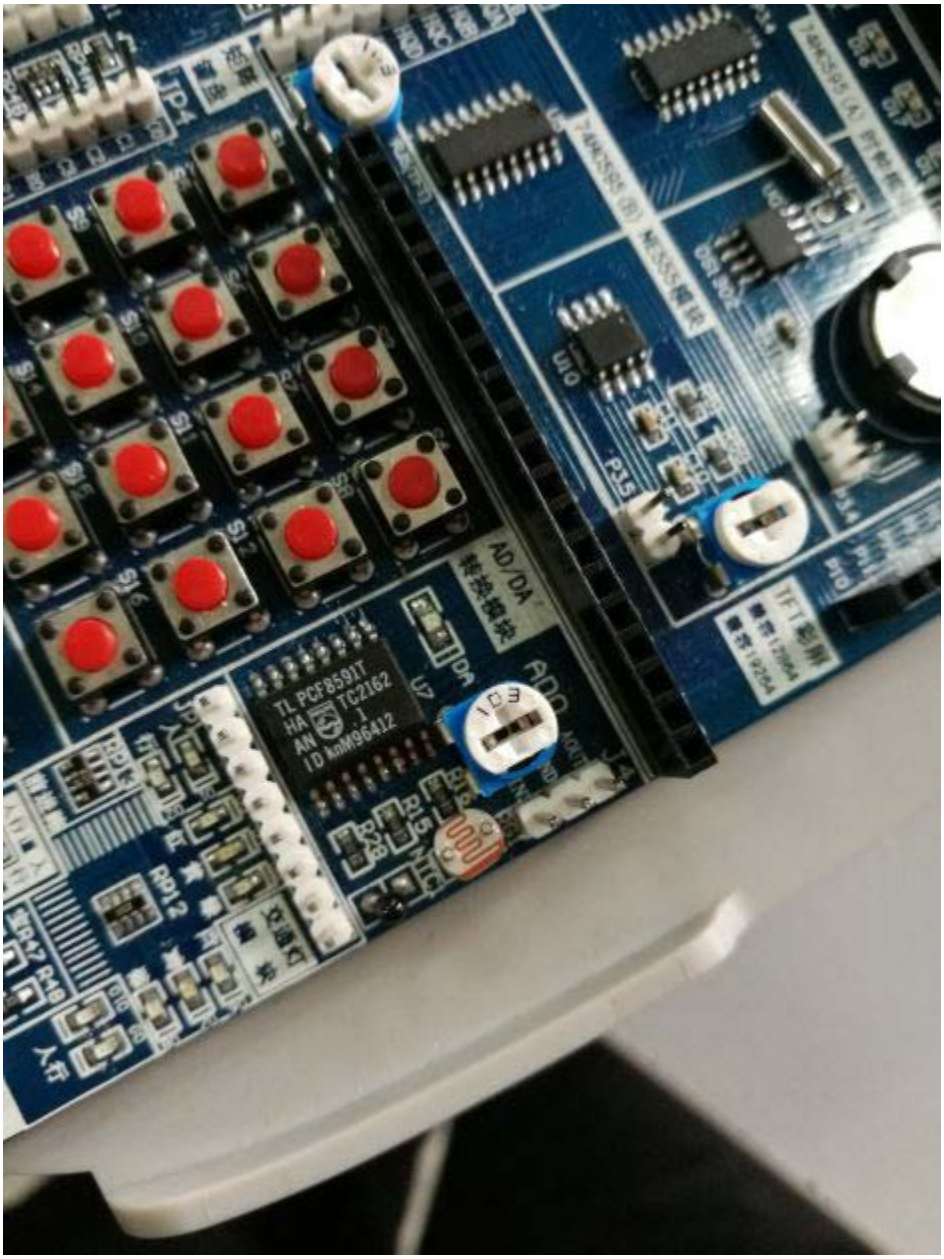
俗话说的好，自己挖的坑，跪着也得填。。这话还得从申请开发板说起啊，当时脑袋一热啊，就寻思做一个智能医疗的什么，板子拿到手之后啊，查遍了引脚图，没有 ad 转换的引脚，当时啊，我抱着芯片手册在厕所里哭晕了一次又一次啊。还好有大神提点啊，这里鸣谢一下小狂同学@3guoyangyang7 ，对嘛，片上没有 ad 可以外挂嘛。。所以，无尽折磨刚刚开始了。

嗯。。从哪里说起呢，应该是从心率传感器说起吧，当时觉得自己学校有这玩意，可以直接从老师那里磨叽一个拿来玩耍，查了好多的芯片手册，还看了看各种例程，总而言之，就是没看懂，最后看了看引脚图，一个电源一个 GND，一个 ADC，一种得了，先把 ad 读出来再说的感觉油然而生。


GPIOC9(CLK2) 40
GPIOC12(MOSI2) 38
GPIOC10(CS2) 36
DGND 34
GPIOC28 32
DGND 30
SCL1 28
GPIOB26 26
GPIOC30(CS0) 24
GPIOB28(RX4) 22
DGND 20
GPIOB27 18
DGND 14
MIC
LED-STAT
LED-PWR
Power
Audio Out
Ethernet
USB 3-4
HDMI
WARNING
5V/2A power source is a MUST.

然后就到了，怎么外挂 AD 芯片的问题，手上呢有两种解决方案，一个是 ADC0809，一个是 51 单片机开发板上面的 AD 转换模块（偷的学弟的开发板，嘿嘿）。像我这种焊东西全靠杵的人啊，查了一下 adc0809 的东西，感觉难度好大的，另外最主要的是占用 IO 口资源太多了，一共没几个 I/O 口还让拿出 7 个引脚来读 ad，而且精度不高，怎么想都很亏啊，所以啊。。最后还是选择 51 开发板上的东西吧，I2C 虽然驱动程序写的费劲，但是用的资源少么，所以啊，这样吧。


JH118ADE3
ADC0809CCN



看着这些东西，不管哪个都好不想整啊。
这篇帖子还是写一半吧。先把 I2C 的驱动框架和这个东西的驱动加进去，复习一下吧。。之前也没认真写过 I2C 的驱动

又到了我们最不喜欢的看内核环节，从我开始时候的 2440 到现在的 4418。。真是天差地别啊。
我们先来看看简单易懂的 2440 的 I2C 驱动的初始化程序

```
static int __init i2c_adap_s3c_init(void)
{
        int ret;
```

```
        ret = platform_driver_register(&s3c2410_i2c_driver);
        if (ret == 0) {
                ret = platform_driver_register(&s3c2440_i2c_driver);
                if (ret)
                        platform_driver_unregister(&s3c2410_i2c_driver);
        }

        return ret;
}
```

这个看起来容易很多的样子，基本就是一个 platform_driver_register(&s3c2440_i2c_driver);这个函数的事，注册一个平台设备驱动。。然后我们再看看 4418（我要哭了）

```
CBOOL     NX_I2C_Initialize( void )
{
    //    @modified Gamza static variable(__g_ModuleVariables) is automatically filled by '0'
    //                                父距 檬扁拳 苞沥俊 傈开函荐甫 0 栏肺 檬扁拳 窍绰 累诀
捞寇狼 老阑
    //                                秦具茄促搁 bInit 蔼阑 CFALSE 肺 荐沥秦具茄促.
    static CBOOL bInit = CTRUE;
    U32 i;

    if( CFALSE == bInit )
    {
        for( i=0; i < NUMBER_OF_I2C_MODULE; i++ )
        {
            __g_ModuleVariables.pRegister = CNULL;
        }

        bInit = CTRUE;
    }

    return CTRUE;
}
```

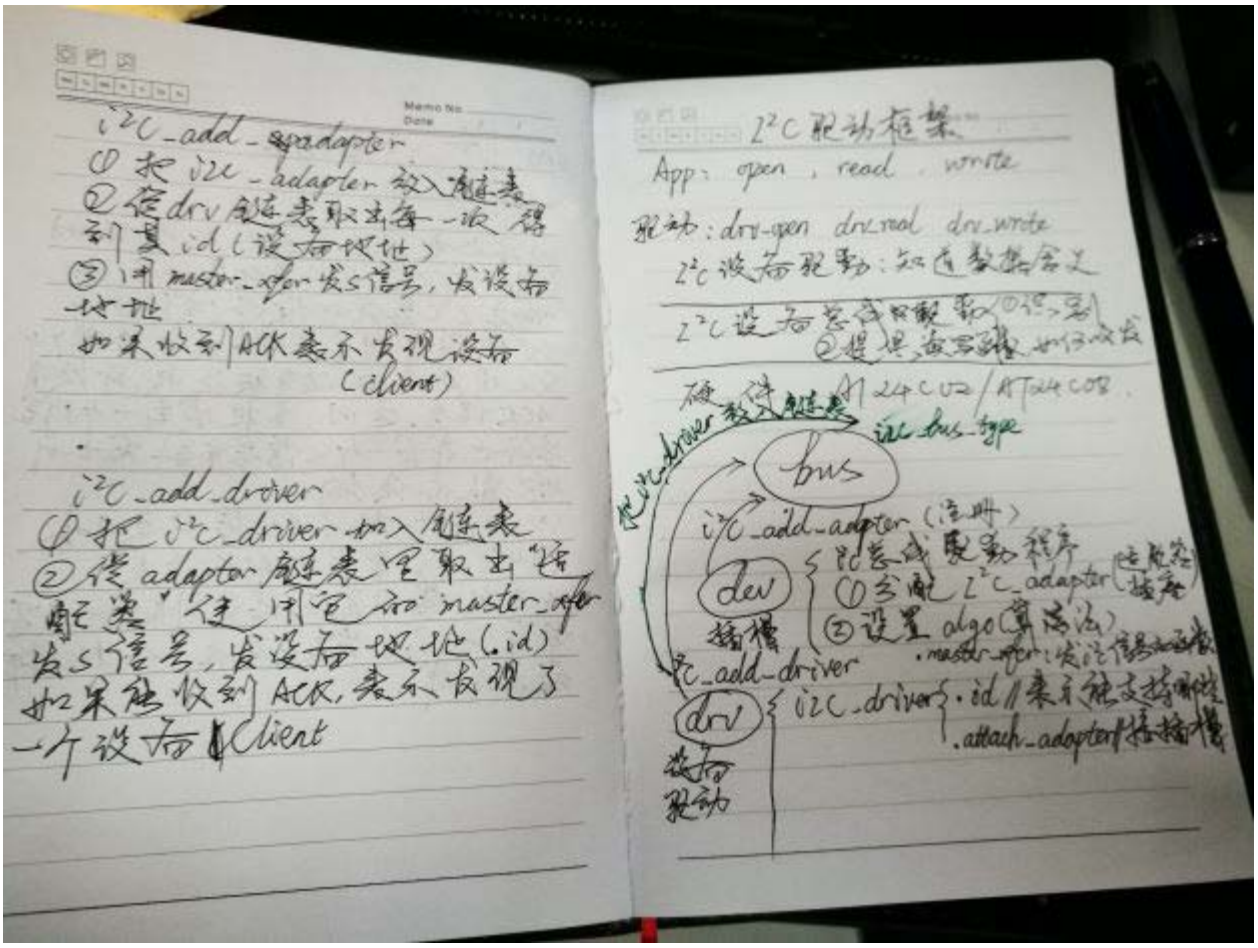
整个人都思密达了。。但是__g_ModuleVariables.pRegister 结构体 最后无非也就是注册这个驱动各个参数

```
    struct    NX_I2C_RegisterSet
    {
        volatile U32 ICCR;                          ///< 0x00 : I2C Control Register
        volatile U32 ICSR;                          ///< 0x04 : I2C Status Register
```

```
        volatile U32 IAR;                    ///< 0x08 : I2C Address Register
        volatile U32 IDSR;                   ///< 0x0C : I2C Data Register
        volatile U32 ICLC;                   ///< 0x10 : I2C Line Control Register
    };
```

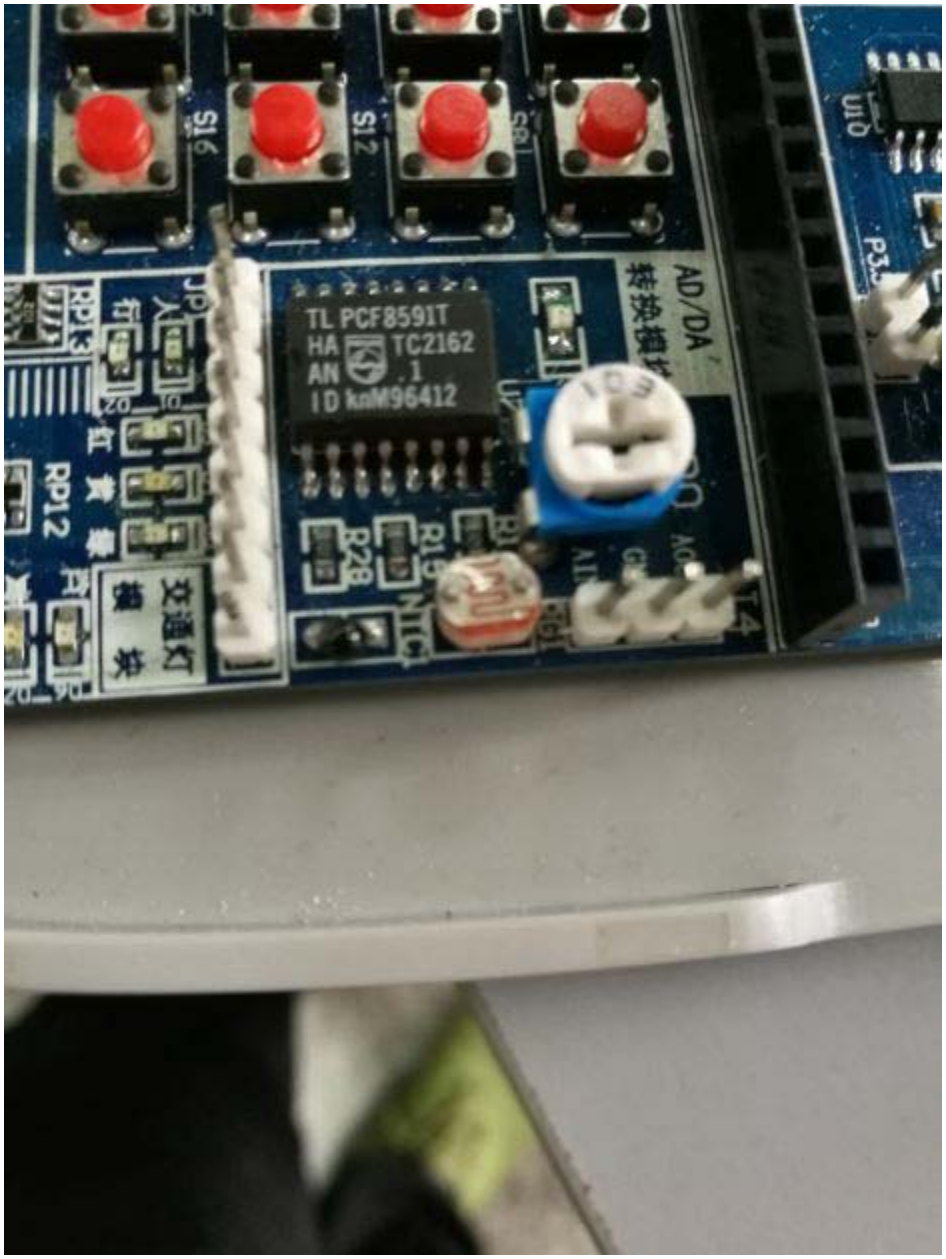
好像更复杂一些，但是似乎还是差不多的了。好吧 ，这只是个初始化，似乎难的在后面呢。

写的有点乱，这个是三星给的驱动程序，内核里还有总线的驱动程序。我也不知道怎么分开来说,算了，还是先说低版本内核的东西吧，等我把新内核整明白了再来改。。。大家仅仅做个参考就好。下面是我韦东山大神画的。。我的笔记大家可以忽略。。



艾玛，这篇帖子我从早晨 9 点开始写的。。一直写到晚上 6 点了。。我的天哪，菜鸟就是菜鸟，明天或者今天晚上，把新内核和 4418 的驱动研究明白一些之后和大家分享，有驱动方面的大神的话，小弟在这里求教了，自己整好费劲啊，在这里献丑了。

本来输入子系统就整得我头大了，现在这个 I2C 简直不要太凶残，大家给我点时间，我相信会更好的。算是开始结项的第一个帖子，磕磕绊绊，真的很难，今天就到这。。有想交流的，请加我的 QQ：504777097


S16
S12
S8
U10
P33
AD/DA
转换模块
TL PCF8591T
HA TC2162
AN .1
1D kaM96412
JP
RP13
RP12
交通灯
蜂鸣器