NanoPi M3 编程开发教程

2016-07-14

(本手册适用于 NanoPi M3 发板)

简介

本手册由广州友善之臂计算机科技有限公司(简称“友善之臂”)创建和维护，友善之臂目前并不对本手册的内容提供任何解释和解答服务，用户可以在论坛中反馈你所遇到的问题和疑问，我们将在以后的更新中修正或者采纳您的建议，本手册主要以首页日期为版本标志。

本手册由友善之臂开发工程师编写制作，专门为嵌入式爱好者准备，同时也适用于产品开发功能，因此读着需要具备一些 Linux 及 ARM 的基本知识。本手册以 NanoPi M3 作为开发平台，讲解如何建立 Ubuntu 开发环境，以及如何使用 NanoPi M3 **开发板作为真机调试程序**，如何在程序中访问硬件设备，包括 I2C, GPIO, SPI, 串口等硬件接口，并且包含了对文件系统的操作和开发板的启动过程等。友善之臂特意介绍了很多极具挑战性和实用性的各种案例，做到真正为你的项目开发助一臂之力。

本手册由广州友善之臂计算机科技有限公司发布，转载请注明出处，手册内难免有遗漏和不足之处，欢迎大家提出宝贵意见。

更新说明

2016-07-14	本手册第一次发布

目录

第一章　初尝 NanoPi M3 体验

NanoPi M3（以下简称 M3）是友善之臂团队面向创客、极客、嵌入式爱好者、电子艺术家、发烧友等群体推出的又一款强劲的完全开源的掌上创客神器。好玩，趣味，突破是我们的目标，随时打开 NanoPi M3，你不只是实验室的技术控，你也可以是生活中的 HACK。它不只是一块开发板，它将会陪你度过漫长开发岁月，与你的思维连接，成为你忠实的朋友，帮你把想法变成现实。

本章节将介绍 NanoPi M3 资源特性，以及如何快速启动 NanoPi M3。

1.1 NanoPi　M3 概述

NanoPi M3 采用三星八核 A53 高性能处理器 S5P6818，动态运行主频 400M-1.4GHz，集成千兆以太网卡，WiFi、蓝牙，并采用了 AXP228 电源管理芯片。可支持软件开关机，并采用 MicroUSB 供电，板载陶瓷天线。体积足够小，接口却丰富，NanoPi M3 引出了视频输入/输出接口（DVP Camera/LVDS/HDMI/LCD 接口），千兆以太网口，I2S 接口，3.5mm 音频输出接口，更是留出了四个 USB 接口，带串口调试功能。

NanoPi M3 资源特性：

- CPU: S5P6818, 动态运行主频 400Mhz--1.4GHz
- PMU 电源管理：AXP228，支持软件关机和睡眠唤醒等
- DDR3 RAM: 1GB
- SD: microSD 卡槽一个
- 网口: 千兆以太网接口(RTL8211E)
- Wireless：802.11 b/g/n
- Bluetooth：4.0 dual mode
- 天线: Wi-Fi 和蓝牙共用, 板载陶瓷天线, 同时提供 IPX 接口
- 音频：3.5mm 耳机座/Via HDMI
- I2S：板载 I2S 接口,7Pin,2.54mm 排针
- 麦克风: 通过 3.5mm 耳机孔输入
- USB Host: 4 x USB 2.0 Host，其中两个是标准 A 型接口，另外两个是 2.54mm 排针
- Micro USB: 1 x USB 2.0 Client
- LCD 接口: 45pin, 0.5mm 间距 FPC 贴片座，支持全彩 TFT LCD (RGB:8-8-8)
- HDMI: HDMI 1.4a, Type A 型口，1080P 高清显示
- LVDS 接口：20Pin,2.0mm 排针
- DVP Camera 接口: 24pin, 0.5mm 间距，FPC 贴片竖座
- GPIO 扩展接口: 40 Pin,2.54mm 排针,兼容 NanoPi M1,M2,树莓派等创客板
- 调试串口：4Pin，2.5mm 单排针
- 按键：K1（电源按键），Reset
- LED：电源指示灯,系统状态指示灯
- RTC: 支持 RTC, 带纽扣电池引线孔
- PCB Size:64x60mm，6 层，沉金工艺
- 供电: DC 5V/2A
- OS/Software: u-boot, Android5.1, Debian8

1.2 准备工作

要开启你的 NanoPi M3 新玩具，请先准备好以下硬件

- NanoPi M3 主板
- microSD 卡/TF 卡: Class10 或以上的 8GB SDHC 卡
- 一个 microUSB 接口的外接电源，要求输出为 5V/3A
- 一台支持 HDMI 输入的显示器或者电视
- 一套 USB 键盘鼠标，同时连接还需要 USB HUB （或选购串口转接板，要 PC 上进行操作）
- 一台电脑，需要联网，建议使用 Ubuntu 14.04 64 位系统

制作一张带运行系统的 TF 卡

1.2.1 载系统固件

首先访问此处的下载地址下载需要的固件文件：
您需要准备一张 4G 或以上容量的 SDHC 卡，该卡的已有数据将会被破坏，因此请先对 SD 卡上的数据进行备份。

使用LCD或HDMI作来输出的用户，使用以下固件：	
s5p6818-debian-sd4g-20160426.img	Debian系统固件
s5p6818-android-sd4g-20160426.img	Android系统固件
s5p6818-core-qte-sd4g-20160426.img	Ubuntu Core + QT系统固件
烧写工具：	
win32diskimager.rar	Windows平台下的烧写工具，Linux系统可以用dd命令

1.2.2 制作启动 NanoPi M3 的 TF 卡

1、将固件 nanopi-m1-debian-sd4g.img.zip 和烧写工具 win32diskimager.rar 分别解压，在 Windows 下插入 TF 卡（限 4G 及以上的卡），以管理员身份运行 win32diskimager 工具， 在 win32diskimager 工具的界面上，选择你的 TF 卡盘符，选择系统固件，点击 Write 按钮烧写即可。

2、当制作完成 TF 卡后，拔出 TF 卡插入 NanoPi M3 的 BOOT 卡槽，上电启动（注意，这里需要 5V/3A 的供电），你可以看到绿灯常亮以及蓝灯闪烁，这时你已经成功启动 NanoPi M1。

1.3 Debian 系统的使用

1.3.1 连接有线网络

NanoPi M3 支持千兆网络，Debian 或者 Android 系统在启动前，只要接上网线，系统启动后则会自动分配 IP 地址，不需要额外去配置。

1.3.2 连接无线网络

用 vi 或在图形界面下用 gedit 编辑文件 /etc/wpa_supplicant/wpa_supplicant.conf，在文件末尾填入路由器信息如下所示：

```
network={
        ssid="YourWiFiESSID"
        psk="YourWiFiPassword"
}
```

其中，YourWiFiESSID 和 YourWiFiPassword 请替换成你要连接的无线 AP 名称和密码。
保存退出后，执行以下命令即可连接 WiFi:

```
ifdown wlan0
ifup wlan0
```

如果你的 WiFi 密码中有特殊字符，或者你不希望明文存放密码，你可以使用 wpa_passphrase 命令为 WiFi 密码生成一个密钥(psk)，用密钥来代替密码 ，在命令行下，可输入以下命令生成密钥:

```
wpa_passphrase YourWiFiESSID
```

在提示输入密码时，输入你的 WiFi 密码，再打开 /etc/wpa_supplicant/wpa_supplicant.conf 文件你会发现密钥已经被更新，你可以删除明文的密码了。

如果你的 WiFi 当前处于无线热点模式，你需要先退出该模式方可连接到路由器，使用以下命令退出无线热点模式：

```
su
turn-wifi-into-apmode no
```

1.3.3 配置 Wi-Fi 无线热点

可以通过以下命令，将 Wi-Fi 切换至无线热点模式：

```
turn-wifi-into-apmode yes
```

按提示重启即可，默认的热点名称为 nanopi2-wifiap，密码为 123456789。

现在，你可以在电脑上搜索并连接 nanopi2-wifiap 这个无线热点，连接成功后，可以通过 ssh 到 192.168.8.1 这个地址来登录 NanoPi M3:

```
ssh root@192.168.8.1
```

在提示输入密码时，输入预设的密码 fa，即可登入。

为了保证 ssh 的流畅，我们用以下命令关闭 wifi 的省电模式:

```
iwconfig wlan0 power off
```

WiFi 工作模式可通过以下命令查询：

```
cat /sys/module/bcmdhd/parameters/op_mode
```

输出为数字 2 则表示当前处于无线热点模式，要切换回普通的 Station 模式，输入如下命令：

```
turn-wifi-into-apmode no
```

1.3.4 使用蓝牙传输

点击右下角的蓝牙图标，会弹出一个操作菜单，其中 Make discoverable 菜单项是打开 NanoPi M3 蓝牙的可发现属性，这样其他设备（例如手机）就可以搜索到 NanoPi M3 并进行配对了；

Devices... 菜单项可以打开搜索界面，主动搜索周边的蓝牙设备（注：需要这个设备先打开可发现属性）；Send Files to Device...菜单项则可以通过蓝牙发送文件到已配对的指定设备上。

1.3.5 安装 Debian 软件包

我们提供的是标准的 Debian jessie 系统，你可以使用 apt-get 等命令来安装软件包，如果板子是首次运行，需要先用以下命令更新软件包列表：

```
apt-get update
```

然后就可以安装软件包了，例如要安装 ftp 服务器，使用以下命令：

```
apt-get install vsftpd
```

如果软件包下载速度不理想，你可以编辑 /etc/apt/sources.list 更换一个更快的源服务器，这个网址[1]

第二章　准备工作

为方便用户开发前了解硬件传感器等模块的特性，这章节主要介绍串口控制台和嵌入式设备 GPIO 的使用。

2.1 串口控制台

使用开发板的过程中，最主要的是调试部分，NanoPi M3 可以直接使用串口线跟 PC 机端（系统为 Ubuntu14.04）进行通讯和调试，方便你的开发。

2.1.1 Minicom 的使用方法

这里介绍的是使用 Minicom 工具进行串口调试。Linux 下的 Minicom 的功能与超级终端功能相似，适用于在通过超级终端对设备的管理以及对嵌入式操作系统的升级。

Step1：安装 Minicom：

```
sudo apt-get install minicom
```

Step2：使用 Minicom 连接 NanoPi M3 UART 串行设备，这时你的开发板跟 PC 机应该通过标配的串口线连接起来，如果你的 PC 机为笔记本，则需要使用 USB 转串口让你的开发板跟电脑连接好，运行 Minicom 工具：

```
#minicom    /dev/ttyS0      （假如你没有使用 USB 转串口）
#minicom    /dev/ttyUSB0        （使用 USB 转串口）
```

配置 Minicom：
1：在终端输入 minicom 以启动 minicom（可以根据你的配置使用以上两个命令）；
2：先按下 Ctrl+a ， 放开，再按 o ， 出现配置菜单；
3：选择 serial port setup ，此时所示图标在“change which setting”中，键入“A”，此时光标移到第 A 项：串口 COM1 对应 ttyS0，COM2 对应 ttyS1，根据你自己的配置选择；
具体的配置信息如下所示：

```
Serial port setup [Enter]
+-----------------------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0                  |
| B - Lockfile Location     : /var/lock                     |
| C -   Callin Program      :                               |
| D - Callout Program       -:                              |
| E -    Bps/Par/Bits       : 115200 8N1                    |
```

```
| F - Hardware Flow Control : No                   |
| G - Software Flow Control : No                   |
|                                                  |
|   Change which setting?                          |
+--------------------------------------------------+
```

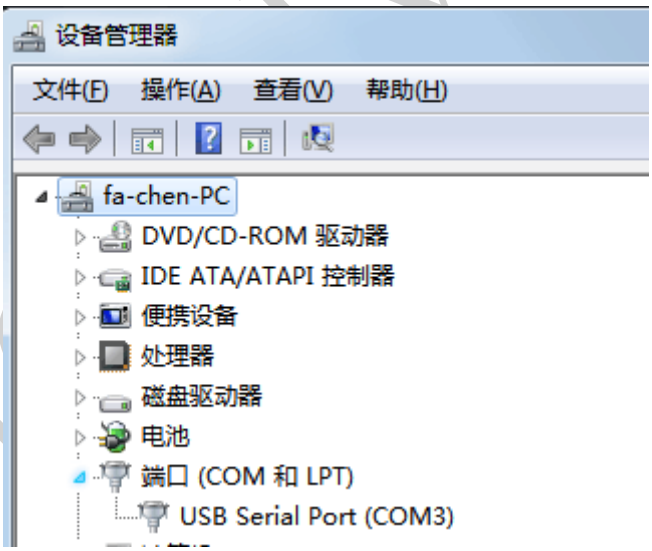
注意：如果没有使用 USB 转串口，而是直接使用串口，那么 serial device 要配置为/dev/ttyS0，如果使用 USB 转串口，则需要配置串口为 ttyUSB0。对波特率，数据位及停止位，键入 E，波特率为 115200　8N1（奇偶校验无，停止位 1），硬、软件流控制都选择 NO。确认更改成功之后，选择“save setup as dfl”之后重启 minicom 让你的配置生效，再连上开发板后，即可在 minicom 中输出正确的串口信息。
如需退出 minicom，可以使用 control-A X ，或者使用 control-A Z 进入帮助菜单。

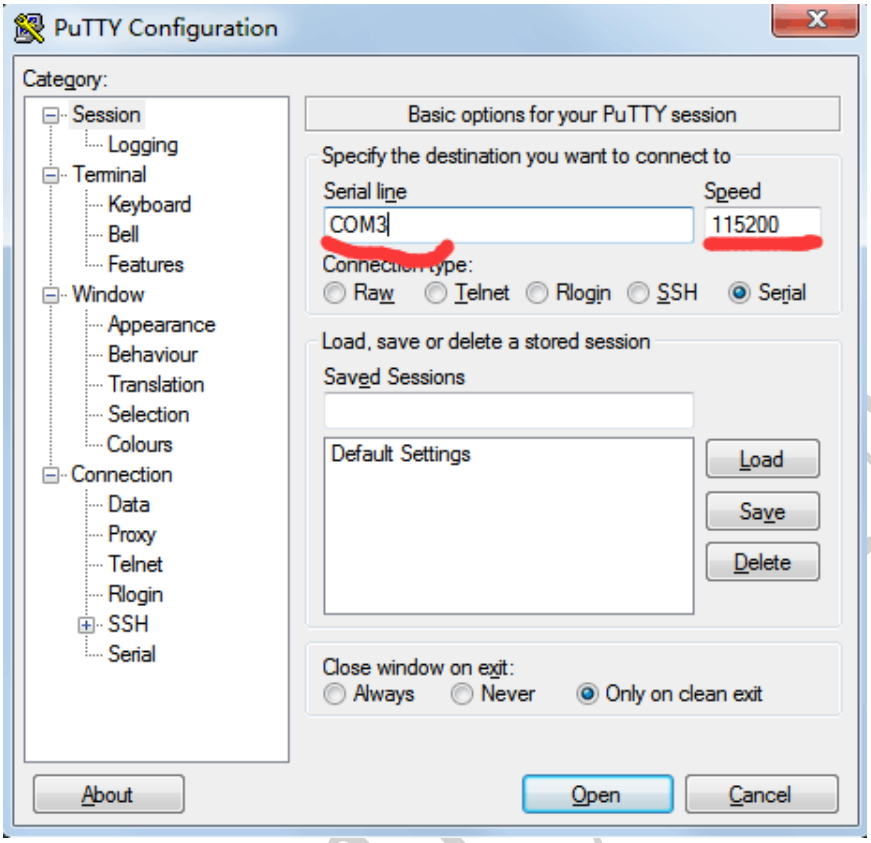
Step3：打开开发板，进入根文件系统。

2.1.2 putty 的使用

如果不想使用 Ubuntu 系统的 Minicom，你可以直接使用 putty 超级终端，这里介绍的 putty 的用法，不再介绍 putty 的概述。

下载 putty 回来，双击 putty.exe，即可运行 putty；从设备管理器里查看串口线占用的是端口几，如果找不到端口号，需要先安装串口驱动。运行 putty 后，波特率设置为 115200，点击 open 之后，打开开发板，即可看到输出信息。





2.2 支持硬件设备模块

NanoPi M3 支持使用市面上常见的传感器模块和外扩硬件设备，经友善之臂开发并且能使用的硬件模块如下：重力加速度模块、通用 GPIO、板载 KEY、摇杆、OLED、adc 转换、8*8 矩阵 LED、土壤温湿度模块、舵机、土壤湿度模块、红外对射技术传感器、LED、红外遥控模块、蓝牙模块、声音传感器、继电器、串口、摇杆、电机模块、无源蜂鸣器、光敏电阻、循迹模块、温湿度传感器模块等。后续我们会继续添加更多传感器模块和硬件设备的支持，本教程将不断更新，请留意友善官方最新动态。

2.2.1 嵌入式设备 GPIO 口介绍

在嵌入式设备中对 GPIO 的操作，一般的做法是写一个单独驱动程序。其实 linux 下面有一个通用的 GPIO 操作接口，那就是 “/sys/class/gpio” 方式。使用这种方法，你不需要写驱动，直接调用 gpio 引脚，更方便开发，内核更小。

首先，通过串口进入根文件系统，查看系统中有没有“/sys/class/gpio”这个文件夹。如果没有请在编译内核的时候加入　Device Drivers　—>　GPIO Support　—>　　/sys/class/gpio/… (sysfs interface)。

1：**gpio_operation** 通过/sys/文件接口操作 IO 端口 GPIO 到文件系统的映射；
2：控制 GPIO 的目录位于**/sys/class/gpio** 中；
3： **/sys/class/gpio/export** 文件用于通知系统需要导出控制的 GPIO 引脚编号；
4：**/sys/class/gpio/unexport** 用于通知系统取消导出；
5：**/sys/class/gpio/gpiochipX** 目录保存系统中 GPIO 寄存器的信息，包括每个寄存器控制引脚的起始编号 base，寄存器名称，引脚总数 导出一个引脚的操作步骤；（X 表示数字）；
6：首先计算此引脚编号，引脚编号 = 控制引脚的寄存器基数 + 控制引脚寄存器位数；
7： 向**/sys/class/gpio/export** 写入此编号，比如 12 号引脚，在 shell 中可以通过以下命令实现，命令成功后生成**/sys/class/gpio/gpio12** 目录，如果没有出现相应的目录，说明此引脚不可导出；
8：**echo 12 > /sys/class/gpio/export;**
9：direction 文件，定义输入输出方向，可以通过下面命令定义为输出；
10：**echo out > direction;**
11：direction 接受的参数：**in, out, high, low**。**high/low** 同时设置方向为输出，并将 value 设置为相应的 1/0；
12：value 文件是端口的数值，为 **1** 或 **0**；
13：**echo 1 & > value**。

NanoPi M3 GPIO 管脚如图：

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
37	AliveGPIO3	38	SPI2_MOSI/GPIOC12
39	DGND	40	SPI2_CLK/GPIOC9

/sys/class/gpio 的使用说明：

测试：由于我们需要通过 sysfs 的方式来操作 GPIO，因此，我们还需要找到这组引脚在内核中所对应的编号（每一个 GPIO 引脚，内核都会赋于它一个编号），查询内核编号的方法如下：

1）NanoPi M3 开机后，进入根文件系统；

2）在超级终端中输入以下命令（因命令比较长，建议直接复制运行）：

```
# cd /sys/class/gpio
# for i in gpiochip* ; do echo `cat $i/label`: `cat $i/base` ; done
nxp-gpio.0: 0
nxp-gpio.4: 128
nxp-gpio.5: 160
nxp-gpio.1: 32
nxp-gpio.2: 64
nxp-gpio.3: 96
```

其实这里的gpio.0表示的是GPIOA，gpio.1表示GPIOB，其他是也是这样计算。
假如，我们需要使用NanoPi M3的GPIOB30引脚，想在应用程序使用，我们可以表示为：
64+30=94 也就是说我们使用的是第 94 根 GPIO。

2.2.2 安装 Matrix 函数库

Matrix 配件专门为创客而生，是友善之臂设计生产并发行的开源硬件传感器，它包括各种常用传感器及硬件配件，非常适合初学者入门嵌入式知识。它适用于友善之臂所有 ARM 开发板，兼容树莓派、arduino 等主流开发板，秉着开源创新的精神，友善之臂专门提供了完整的示例教程和开源的程序，并且有完整的维基，你可以随心所欲发挥你的创意，做出有趣、实用的东西。

启动开发板并运行 Debian 系统，进入系统后克隆 Matrix 代码仓库：

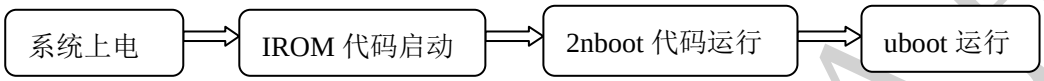
```
$ apt-get update && apt-get install git
$ git clone https://github.com/friendlyarm/matrix.git
```

第三章　NanoPi M3 启动过程分析

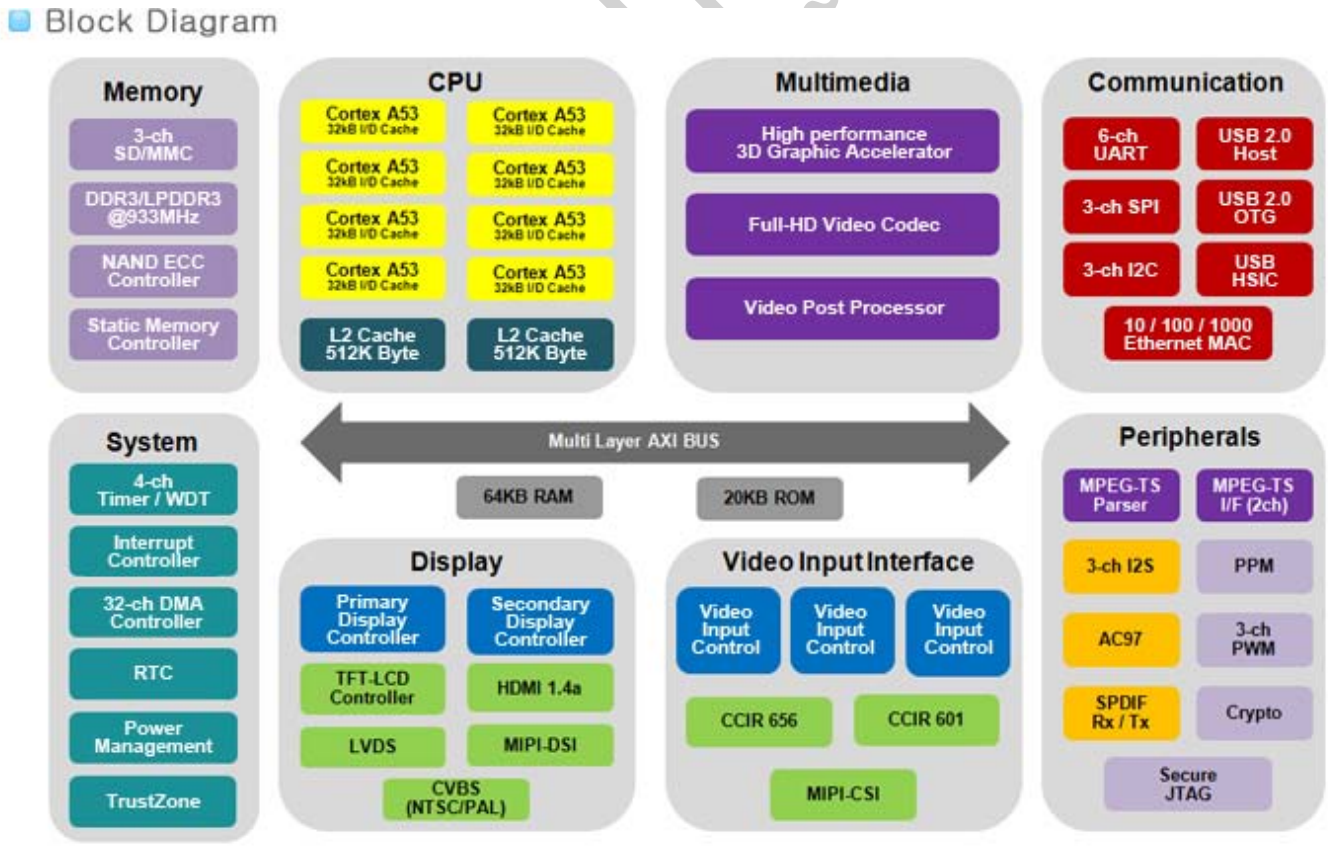
这章节主要讲解 NanoPi M3 开发板的启动过程，详细讲解烧写脚本的介绍和 uboot 的启动分析。

3.1 S5P6818 启动过程

S5P6818启动过程如图



1、系统上电；
2、系统上电后，会首先运行 IROM 里的代码；
3、加载 2ndboot 并运行之；
4、2ndboot 会去加载 uboot 并运行，然后根据环境变量加载内核并运行；



下面是一个镜像在 EMMC 中分布图：

Block 0	Sd/emmc 卡信息
Block 1	Nsih.bin
Block2	2nboot.bin
Block63	Nsih.bin
Block64	u-boot.bin

这样看就比较清楚看镜像所以具体位置。

U-boot.bin 可能大家都知道，这是用 u-boot 代码编译出来的，这个我就不介绍了。我们来简单说一个 nsih.bin 这个文件。

Nsih.bin 这个文件其实就是对芯片一些简单寄存器的配制，包括（启动方式，PLL,DDR）。

第四章　Uboot 启动过程分析

1、首先从链接脚本文件 uboot.lds 代码开始，找到代码的起始：

从代码可知程序的入口点是_start，位于 arch/arm/cpu/slsiap/s5p6818/start.o，且执行的入口点是_stext，uboot 启动的第一阶段。

```
#cd   s5p6818/uboot_nanopi2
#cat   u-boot.lds
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_stext)
SECTIONS
{
 . = 0x00000000;
 . = ALIGN(4);
 .text :
 {
  *(.__image_copy_start)
  arch/arm/cpu/slsiap/s5p6818/start.o (.text*)
  *(.text*)
 }
```

2、找到 uboot 启动的第一阶段，以下是 start.S 的源码，s5p6818 的数据手册有对该源码的详细解释：

* 将 CPU 设置为 SVC32 模式；
* 初始化 CP15 协处理器（禁用一级 I-Cache、D-Cache、MMC）；
* 在 bl cpu_init_crit 中进行跳转，进入 lowlevel_init 获取 CPU 、ID、内存信息等；
* 确定 uboot 运行运行在 flash 中还是 RAM 中；
* 如果是运行在 DRAM 中，则初始化 bss 段为 0，开启 mmu（开启的代码在 arch/arm/cpu/slsiap/s5p6818/mmu_asm.S 和 arch/arm/cpu/slsiap/s5p6818/mmu.c），重新初始化栈指针，并在第 140 行跳转到 board_init_f 入口执行；
* board_init_f 入口在 commom/board_f.c。函数返回时，返回至 start.S 的第 142 行继续执行。初始化 led，重新初始化串口，初始化堆，初始化 mmc 通道，初始化环境变量，初始化控制台，初始化中断系统并启动它；
*运行至第 156 行时，跳转至 board_init_r 入口。该入口在 commom/board_r.c 中。uboot 识别 flash 空间长度，初始化 led，重新初始化串口，初始化堆，初始化 mmc 通道，初始化环境变量，初始化控制台，初始化中断系统并启动它。最后到 927 行运行 run_main_loop()，最终调用 main_loop()。

```
#cd   s5p6818/uboot_nanopi2
#cat   arch/arm/cpu/slsiap/s5p6818/start.S
/*
 * armboot - Startup Code for NXPxxxx/ARM Cortex CPU-core
 */
```

```
#include <asm-offsets.h>
#include <config.h>
#include <version.h>
#include <asm/system.h>
#include <linux/linkage.h>

/*
 *************************************************************************
 *
 * Exception vectors as described in ARM reference manuals
 *
 * replace arm/lib/vectors.S
 *
 *************************************************************************
 */
    .globl  _stext
_stext:
    b    reset    //uboot 的主入口，复位向量入口，跳入了后面的 reset
    ldr pc, _undefined_instruction
    ldr pc, _software_interrupt
    ldr pc, _prefetch_abort
    ldr pc, _data_abort
    ldr pc, _not_used
    ldr pc, _irq
    ldr pc, _fiq

/* ldr 语句的意思是将第二个操作数（如: _undefined_instruction）指向的地址数据传给 PC，
undefined_instruction 为地址， 即后面标号所对的偏移地址数据  */

_undefined_instruction:    .word undefined_instruction
_software_interrupt:    .word software_interrupt
_prefetch_abort:    .word prefetch_abort
_data_abort:        .word data_abort
_not_used:          .word not_used
_irq:               .word irq
_fiq:               .word fiq

    .balignl 16,0xdeadbeef    //16 字节对齐，并以 0xdeadbeef 填充，它是个 Magic number

/*
```

```
 ************************************************************************
 *
 * Text and section base
 *
 ************************************************************************
 */

.globl TEXT_BASE
TEXT_BASE:
    .word   CONFIG_SYS_TEXT_BASE

/*
 * These are defined in the board-specific linker script.
 */
.globl _bss_start_ofs
_bss_start_ofs:
    .word __bss_start - _stext

.globl _bss_end_ofs
_bss_end_ofs:
    .word __bss_end - _stext

.globl _end_ofs
_end_ofs:
    .word _end - _stext

/*
 ************************************************************************
 *
 * Reset handling  代码从这里开始执行
 *
 ************************************************************************
 */

    .globl reset

reset:
    bl  save_boot_params
    /*
     * set the cpu to SVC32 mode   让系统进入 SVC 模式
     */
```

```
    mrs r0, cpsr
    bic r0, r0, #0x1f
    orr r0, r0, #0xd3
    msr cpsr,r0

    /* the mask ROM code should have PLL and others stable */
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
    bl  cpu_init_cp15
    bl  cpu_init_crit
#endif

#ifdef CONFIG_RELOC_TO_TEXT_BASE
relocate_to_text:
    /*
     * relocate u-boot code on memory to text base
     * for nexell arm core (add by jhkim)
     */
    adr r0, _stext              /* r0 <- current position of code   */
    ldr r1, TEXT_BASE           /* test if we run from flash or RAM */
    cmp r0, r1                  /* don't reloc during debug         */
    beq clear_bss

    ldr r2, _bss_start_ofs
    add r2, r0, r2              /* r2 <- source end address         */

copy_loop_text:
    ldmia   r0!, {r3-r10}       /* copy from source address [r0]    */
    stmia   r1!, {r3-r10}       /* copy to   target address [r1]    */
    cmp r0, r2                  /* until source end addreee [r2]    */
    ble copy_loop_text

    ldr r1, TEXT_BASE           /* restart at text base */
    mov pc, r1

clear_bss:
#ifdef CONFIG_MMU_ENABLE
    bl  mmu_turn_on
#endif
    ldr r0, _bss_start_ofs
    ldr r1, _bss_end_ofs
    ldr r4, TEXT_BASE           /* text addr */
```

```
    add r0, r0, r4
    add r1, r1, r4
    mov r2, #0x00000000         /* clear */

clbss_l:str r2, [r0]            /* clear loop... */
    add r0, r0, #4
    cmp r0, r1
    bne clbss_l

    ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)
    bic sp, sp, #7                  /* 8-byte alignment for ABI compliance */
    sub sp, #GD_SIZE                /* allocate one GD above SP */
    bic sp, sp, #7                  /* 8-byte alignment for ABI compliance */
    mov r9, sp                      /* GD is above SP */
    mov r0, #0
    bl  board_init_f

    mov sp, r9                      /* SP is GD's base address */
    bic sp, sp, #7                  /* 8-byte alignment for ABI compliance */
    sub sp, #GENERATED_BD_INFO_SIZE    /* allocate one BD above SP */
    bic sp, sp, #7                  /* 8-byte alignment for ABI compliance */

    mov r0, r9                      /* gd_t *gd */
    ldr r1, TEXT_BASE               /* ulong text */
    mov r2, sp                      /* ulong sp */
    bl  gdt_reset

    /* call board_init_r(gd_t *id, ulong dest_addr) */
    mov r0, r9                          /* gd_t */
    ldr r1,  =(CONFIG_SYS_MALLOC_END)  /* dest_addr for malloc heap end */
    /* call board_init_r */
    ldr pc, =board_init_r               /* this is auto-relocated! */

#else   /* CONFIG_RELOC_TO_TEXT_BASE */

    bl  _main
#endif

/*------------------------------------------------------------------------
-----*/
```

```
ENTRY(c_runtime_cpu_setup)
/*
 * If I-cache is enabled invalidate it
 */
#ifndef CONFIG_SYS_ICACHE_OFF
    mcr p15, 0, r0, c7, c5, 0  @ invalidate icache
#ifndef CONFIG_MACH_S5P6818
    mcr     p15, 0, r0, c7, c10, 4 @ DSB
    mcr     p15, 0, r0, c7, c5, 4  @ ISB
#endif
#endif
/*
 * Move vector table
 */
    /* Set vector address in CP15 VBAR register */
    ldr     r0, =_stext
    mcr     p15, 0, r0, c12, c0, 0  @Set VBAR

    bx  lr

ENDPROC(c_runtime_cpu_setup)

/*************************************************************************
 *
 * void save_boot_params(u32 r0, u32 r1, u32 r2, u32 r3)
 *  __attribute__((weak));
 *
 * Stack pointer is not yet initialized at this moment
 * Don't save anything to stack even if compiled with -O0
 *
 *************************************************************************/
ENTRY(save_boot_params)
    bx  lr      @ back to my caller
ENDPROC(save_boot_params)
    .weak   save_boot_params

/*************************************************************************
 *
 * cpu_init_cp15
 *
 * Setup CP15 registers (cache, MMU, TLBs). The I-cache is turned on unless
```

```
 * CONFIG_SYS_ICACHE_OFF is defined.
 *
 *************************************************************************/
ENTRY(cpu_init_cp15)
    /*
     * Invalidate L1 I/D
     */
    mov r0, #0          @ set up for MCR
    mcr p15, 0, r0, c8, c7, 0  @ invalidate TLBs
    mcr p15, 0, r0, c7, c5, 0  @ invalidate icache
    mcr p15, 0, r0, c7, c5, 6  @ invalidate BP array
#ifndef CONFIG_MACH_S5P6818
    mcr p15, 0, r0, c7, c10, 4 @ DSB
    mcr p15, 0, r0, c7, c5, 4  @ ISB
#endif

    /*
     * disable MMU stuff and caches
     */
    mrc p15, 0, r0, c1, c0, 0
    bic r0, r0, #0x00002000@ clear bits 13 (--V-)
    bic r0, r0, #0x00000007@ clear bits 2:0 (-CAM)
    orr r0, r0, #0x00000002@ set bit 1 (--A-) Align
    orr r0, r0, #0x00000800@ set bit 11 (Z---) BTB
#ifdef CONFIG_SYS_ICACHE_OFF
    bic r0, r0, #0x00001000@ clear bit 12 (I) I-cache
#else
    orr r0, r0, #0x00001000@ set bit 12 (I) I-cache
#endif
    mcr p15, 0, r0, c1, c0, 0
    mov pc, lr          @ back to my caller
ENDPROC(cpu_init_cp15)

#ifndef CONFIG_SKIP_LOWLEVEL_INIT
/*************************************************************************
 *
 * CPU_init_critical registers
 *
 * setup important registers
 * setup memory timing
 *
```

```
 *************************************************************************/
ENTRY(cpu_init_crit)
    /*
     * Jump to board specific initialization...
     * The Mask ROM will have already initialized
     * basic memory. Go here to bump up clock rate and handle
     * wake up conditions.
调用 board/lowlevel_init.S 中的 lowlevel_init 函数，对系统总线的初始化，初始化了连接存储器的位宽、速度、刷新率等重要参数。经过这个函数的正确初始化，Nor Flash、SDRAM 才可以被系统使用。
     */
    b   lowlevel_init       @ go setup pll,mux,memory
ENDPROC(cpu_init_crit)
#endif
```

3、第二阶段入口函数在 arch/arm/lib/board.c 中，board_init_r：

```
/************************************************************************
 *
 * This is the next part if the initialization sequence: we are now
 * running from RAM and have a "normal" C environment, i. e. global
 * data can be written, BSS has been cleared, the stack size in not
 * that critical any more, etc.
 *
 ************************************************************************
 */

void board_init_r(gd_t *id, ulong dest_addr)
{
    ulong malloc_start;
#if !defined(CONFIG_SYS_NO_FLASH)
    ulong flash_size;
#endif

    gd->flags |= GD_FLG_RELOC;    /* tell others: relocation done */
    bootstage_mark_name(BOOTSTAGE_ID_START_UBOOT_R, "board_init_r");

    monitor_flash_len = (ulong)&__rel_dyn_end - (ulong)_start;

    /* Enable caches */
    enable_caches();

    debug("monitor flash len: %08lX\n", monitor_flash_len);
```

```
    board_init();    /* Setup chipselects */
    /*
     * TODO: printing of the clock inforamtion of the board is now
     * implemented as part of bdinfo command. Currently only support for
     * davinci SOC's is added. Remove this check once all the board
     * implement this.
     */
#ifdef CONFIG_CLOCKS
    set_cpu_clk_info(); /* Setup clock information */
#endif
    serial_initialize();

    debug("Now running in RAM - U-Boot at: %08lx\n", dest_addr);

#ifdef CONFIG_LOGBUFFER
    logbuff_init_ptrs();
#endif
#ifdef CONFIG_POST
    post_output_backlog();
#endif

    /* The Malloc area is immediately below the monitor copy in DRAM */
    malloc_start = dest_addr - TOTAL_MALLOC_LEN;
    mem_malloc_init (malloc_start, TOTAL_MALLOC_LEN);

#ifdef CONFIG_ARCH_EARLY_INIT_R
    arch_early_init_r();
#endif

#ifndef CONFIG_PMIC_REG_DUMP
    power_init_board();
#endif

#if !defined(CONFIG_SYS_NO_FLASH)
    puts("Flash: ");

    flash_size = flash_init();
    if (flash_size > 0) {
# ifdef CONFIG_SYS_FLASH_CHECKSUM
        print_size(flash_size, "");
        /*
```

```
         * Compute and print flash CRC if flashchecksum is set to 'y'
         *
         * NOTE: Maybe we should add some WATCHDOG_RESET()? XXX
         */
        if (getenv_yesno("flashchecksum") == 1) {
            printf("  CRC: %08X", crc32(0,
                (const unsigned char *) CONFIG_SYS_FLASH_BASE,
                flash_size));
        }
        putc('\n');
# else    /* !CONFIG_SYS_FLASH_CHECKSUM */
        print_size(flash_size, "\n");
# endif /* CONFIG_SYS_FLASH_CHECKSUM */
    } else {
        puts(failed);
        hang();
    }
#endif

#if defined(CONFIG_CMD_NAND)
#if defined(CONFIG_NAND_MTD)
    puts("NAND:  ");
    nand_init();        /* go init the NAND */
#elif defined(CONFIG_NAND_FTL)
#include <nand_ftl.h>
    puts("NAND FTL:  ");
    nand_ftl_init();        /* go init the NAND */
#endif
#endif /* CONFIG_CMD_NAND */

#if defined(CONFIG_CMD_ONENAND)
    onenand_init();
#endif

#ifdef CONFIG_GENERIC_MMC
    puts("MMC:   ");
    mmc_initialize(gd->bd);
#endif

#ifdef CONFIG_CMD_SCSI
    puts("SCSI:  ");
```

```
    scsi_init();
#endif

#ifdef CONFIG_HAS_DATAFLASH
    AT91F_DataflashInit();
    dataflash_print_info();
#endif

    /* initialize environment */
    if (should_load_env())
        env_relocate();
    else
        set_default_env(NULL);

#if defined(CONFIG_CMD_PCI) || defined(CONFIG_PCI)
    arm_pci_init();
#endif

    stdio_init();    /* get the devices list going. */

    jumptable_init();

#if defined(CONFIG_API)
    /* Initialize API */
    api_init();
#endif

    console_init_r();    /* fully init console as a device */

#ifdef CONFIG_PMIC_REG_DUMP
    power_init_board();
#endif

#ifdef CONFIG_DISPLAY_BOARDINFO_LATE
# ifdef CONFIG_OF_CONTROL
    /* Put this here so it appears on the LCD, now it is ready */
    display_fdt_model(gd->fdt_blob);
# else
    checkboard();
# endif
#endif
```

```
#if defined(CONFIG_ARCH_MISC_INIT)
    /* miscellaneous arch dependent initialisations */
    arch_misc_init();
#endif
#if defined(CONFIG_MISC_INIT_R)
    /* miscellaneous platform dependent initialisations */
    misc_init_r();
#endif

     /* set up exceptions */
    interrupt_init();
    /* enable exceptions */
    enable_interrupts();

    /* Initialize from environment */
    load_addr = getenv_ulong("loadaddr", 16, load_addr);

#ifdef CONFIG_BOARD_LATE_INIT
    board_late_init();
#endif

#ifdef CONFIG_BITBANGMII
    bb_miiphy_init();
#endif
#if defined(CONFIG_CMD_NET)
    puts("Net:   ");
    eth_initialize(gd->bd);
#if defined(CONFIG_RESET_PHY_R)
    debug("Reset Ethernet PHY\n");
    reset_phy();
#endif
#endif

#ifdef CONFIG_POST
    post_run(NULL, POST_RAM | post_bootmode_get(0));
#endif

#if defined(CONFIG_PRAM) || defined(CONFIG_LOGBUFFER)
    /*
     * Export available size of memory for Linux,
```

```
     * taking into account the protected RAM at top of memory
     */
    {
        ulong pram = 0;
        uchar memsz[32];

#ifdef CONFIG_PRAM
        pram = getenv_ulong("pram", 10, CONFIG_PRAM);
#endif
#ifdef CONFIG_LOGBUFFER
#ifndef CONFIG_ALT_LB_ADDR
        /* Also take the logbuffer into account (pram is in kB) */
        pram += (LOGBUFF_LEN + LOGBUFF_OVERHEAD) / 1024;
#endif
#endif
        sprintf((char *)memsz, "%ldk", (gd->ram_size / 1024) - pram);
        setenv("mem", (char *)memsz);
    }
#endif

    /* main_loop() can return to retry autoboot, if so just run it again. */
    for (;;) {
        main_loop();
    }
    /* NOTREACHED - no way out of command loop except booting */
}
```

第五章　烧写脚本解释

Shell 脚本：

```
$# 是传给脚本的参数个数
$0 是脚本本身的名字
$1 是传递给该 shell 脚本的第一个参数
$2 是传递给该 shell 脚本的第二个参数
$@ 是传给脚本的所有参数的列表
$* 是以一个单字符串显示所有向脚本传递的参数，与位置变量不同，参数可超过 9 个
$$ 是脚本运行的当前进程 ID 号
$? 是显示最后命令的退出状态，0 表示没有错误，其他表示有错误
$- 最后运行的命令结束代码（返回值）
```

```
elif 可以有多个

在“判断条件”这个字段里可以直接写入 bash 下的命令、也可以写成条件测试
在判断条件中要进行条件测试：
条件测试方式：
“[ expression ]”一个中括弧里写表达式，
“[[ expression ]]”两个中括弧里写上表达式
“test expression”
“bash 命令”
条件测试的类型：
整数测试
expression:[ 数值 1 比较符 数值 2 ]
比较符一般有以下几种：
大于：-gt(greater than)，大于等于：-ge(greater equal)，小于：-lt(less than)，小于等于：le(less equal)，不等于：-ne(not equal)
字符测试
"\>"：大于
"\<"：小于
"=="或者"="：等于
"=~"：判断左边的字符串能否被右边的模式所匹配，通常用于[[ expression ]]
单目测试：
```

-z：格式为"[-z $STRING]"，表示为空值时则为真，不为空值时则为假
-n：格式为"[-n $STRING]"，表示为空值时则为假，不为空值时则为真
文件、目录测试
-d：测试目录是否存在
-f：测试档案是否存在
组合条件测试：
当有多个测试条件时，我们可以把这些测试条件组合起来使用：
-a：逻辑与
-o：逻辑或
！：逻辑非，这是单目操作
当是 bash 命令之间组合测试时，则：
&&：逻辑与
||：逻辑或
！：逻辑非

文件状态测试是指根据给定的路径名称，判断该名称对应的是文件还是目录，或者判断文件是否可读，可写，可执行等。根据判断的状态不同，在条件表达式中需要使用不同的操作选项。
-d：测试是否为目录（Directory）。
-e：测试目录或文件是否存在（Exist）。
-f：测试是否为文件（File）。
-r：测试当前用户是否有权限读取（Read）。
-w：测试当前用户是否有权限写入（Write）。
-x：测试当前用户是否可执行（Excute）该文件。
-L：测试是否为符号连接（Link）文件。

整数值比较是指根据给定的两个整数值，判断第一个数是否大于、等于、小于第 2 个数，可以使用的操作选项如下：
-eq:第一个数等于（Equal）第二个数。
-ne：第一个数不等于（Not Equal）第二个数。
-gt：第一个数大于（Greater Than）第二个数。
-lt：第一个数小于（Lesser Than）第二个数。
-le：第一个数小于或等于（Lesser or Equal）第二个数。
-ge：第一个数大于或等于（Greater or Equal）第二个数。

以下是对 sd-fuse_s5p6818.sh 脚本的注释：

```
if [ $(id -u) -ne 0 ]; then
```

```
    echo "Re-running script under sudo..."   //以 sudo 权限运行烧写脚本
    sudo "$0" "$@" //0:脚本本身的名字;@传给脚本的所有参数列表
    exit
fi
# ------------------------------------------------------------
# Checking device for fusing
if [ -z $1 ]; then   //$1 传递给脚本的第一个参数,-z 表示空值，空值时退出
    echo "Usage: $0 DEVICE [android|debian]"
    exit 0
fi
case $1 in   //SD 卡设备传递给 shell 脚本的设备名
/dev/sd[a-z] | /dev/loop[0-9] | /dev/mmcblk1)
    if [ ! -e $1 ]; then     //检查烧写文件是否存在
        echo "Error: $1 does not exist."   //文件不存在则退出
        exit 1
    fi
    DEV_NAME=`basename $1`  //指定给 DEV_NAME 文件第一个位置上的参数（$1）的
值（就是 SD 卡）
    BLOCK_CNT=`cat /sys/block/${DEV_NAME}/size` ;;&
/dev/sd[a-z])   //读 SD 卡块设备容量
    DEV_PART=${DEV_NAME}2  //如果路径读到第二块设备，则表示读到硬盘
    REMOVABLE=`cat /sys/block/${DEV_NAME}/removable` ;;
/dev/mmcblk1 | /dev/loop[0-9])    //退出，不读
    DEV_PART=${DEV_NAME}p2
    REMOVABLE=1 ;;
*)
    echo "Error: Unsupported SD reader"
    exit 0
esac //以上表示,插上 SD 卡后,脚本回去读取设备的内容,如果读取到的设备大小大于 500G,
则表示读取到的可能是硬盘分区，这样的话，就会退出不读取
if [ ${REMOVABLE} -le 0 ]; then
    echo "Error: $1 is non-removable device. Stop."
    exit 1
fi   //-le：SD 卡设备小于或等于 0，则退出
if [ -z ${BLOCK_CNT} -o ${BLOCK_CNT} -le 0 ]; then
    echo "Error: $1 is inaccessible. Stop fusing now!"
    exit 1
fi   //SD 卡设备为空值或小于等于 0，则退出
let DEV_SIZE=${BLOCK_CNT}/2
if [ ${DEV_SIZE} -gt 64000000 ]; then
    echo "Error: $1 size (${DEV_SIZE} KB) is too large"
```

```
    exit 1
fi   //512byte 为一个单位，计算 block 大小，如果 block 大小超过 64000000 则退出
      //一个 block=512byte,block x 512byte/1024 等于 kb
if [ ${DEV_SIZE} -le 3800000 ]; then
    echo "Error: $1 size (${DEV_SIZE} KB) is too small"
    echo "       At least 4GB SDHC card is required, please try another card."
    exit 1
fi    //如果小于 3800000 则退出，必须使用大于 4G 的 SD 卡
# ----------------------------------------------------------
# Get target OS //从 ext4 分区的 SD 卡中获取烧写文件，检查 SD 卡是否存在以下文件
case ${2,,} in
debian)
    TARGET_OS=debian
    PARTMAP=./${TARGET_OS}/partmap.txt
    ;;
eflasher)
    TARGET_OS=eflasher
    PARTMAP=./${TARGET_OS}/partmap.txt
    ;;
*)
    TARGET_OS=android
    PARTMAP=./${TARGET_OS}/partmap.txt
    if [ ! -z ${ANDROID_PRODUCT_OUT} ]; then
        PARTMAP=${ANDROID_PRODUCT_OUT}/partmap.txt
    fi
    ;;
esac
if [ ! -d ${TARGET_OS} ]; then   //查找是否存在目标文件路径
    echo -n "Warn: Image not found for ${TARGET_OS^}, download now (Y/N)? "//
提示用户敲入 Y 或者 N，判断是否进入烧写
    while read -r -n 1 -t 10 -s USER_REPLY; do    //读取用户输入值
        if [[ ${USER_REPLY} = [Nn] ]]; then
            echo ${USER_REPLY}
            exit 1  //如果输入 N 则退出不烧写
        elif [[ ${USER_REPLY} = [Yy] ]]; then
            echo ${USER_REPLY}
            break;    //如果输入 Y 则进入下一步骤
        fi
    done
    if [ -z ${USER_REPLY} ]; then
        echo "Cancelled."
```

```
        exit 1    //如果什么都不输入，则退出
    fi
    ./tools/get_rom.sh ${TARGET_OS} || exit 1    //下载目标烧写文件
fi
# ------------------------------------------------------------
# Get host machine
if grep 'ARMv7 Processor' /proc/cpuinfo >/dev/null; then
    EMMC=.emmc
    ARCH=armv7/
fi
# ------------------------------------------------------------
# Fusing 2ndboot, bootloader to card
true ${BOOT_DIR:=./prebuilt}
BL2_BIN=${BOOT_DIR}/2ndboot.bin${EMMC}
BL2_POSITION=1
TBI_BIN=${BOOT_DIR}/boot.TBI
TBI_POSITION=64
BL3_BIN=${BOOT_DIR}/bootloader
BL3_POSITION=65  //这是用于指定烧写到 SD 的起始位置 = 512 bytes * 65
// SD 卡地址偏移量 0x200 之前（也就是 block0）是 SD 卡信息，不要修改，
[0x200-0x400)是 NSIH（iROM 程序会识别的 CPU 配置信息和其他信息），
[0x400-0x8000）是 2ndboot.bin 的所存放的地方，
[0x8000-0x8200）是第二个 NISH，
```

Block 0	Sd/emmc卡信息
Block 1	Nsih.bin
Block2	2ndboot.bin
Block63	Nsih.bin
Block64	u-boot.bin

```
# umount all at first
umount /dev/${DEV_NAME}* > /dev/null 2>&1
echo "--------------------------------"
echo "2ndboot fusing"
dd if=${BL2_BIN} of=/dev/${DEV_NAME} bs=512 seek=${BL2_POSITION}
echo "--------------------------------"
echo "bootloader fusing"
dd if=${TBI_BIN} of=/dev/${DEV_NAME} bs=512 seek=${TBI_POSITION} count=1
dd if=${BL3_BIN} of=/dev/${DEV_NAME} bs=512 seek=${BL3_POSITION}
```

```
#<Message Display>
echo "--------------------------------"
echo "Bootloader image is fused successfully."
echo ""
# ------------------------------------------------------------
# partition card & fusing filesystem   //给 SD 卡分区及烧写文件系统

true ${FW_SETENV:=./tools/${ARCH}fw_setenv}   //正确读取 uboot 的环境变量
true ${SD_UPDATE:=./tools/${ARCH}sd_update}  //更新 SD 卡
true ${SD_TUNEFS:=./tools/sd_tune2fs.sh}     //给 SD 卡分区
echo "--------------------------------"
echo "${TARGET_OS^} filesystem fusing"    //烧写目标文件---文件名
echo "Image root: `dirname ${PARTMAP}`"    //烧写目标文件的路径
echo
if [ ! -f ${PARTMAP} ]; then    //如果找不到烧写文件，则退出
    echo "Error: ${PARTMAP}: File not found"
    echo "       Stop fusing now"
    exit 1
fi
# set uboot env, like cmdline
if [ -f ./${TARGET_OS}/env.conf ]; then   //env.cof 是命令行的配置
    ${FW_SETENV} /dev/${DEV_NAME} -s ./${TARGET_OS}/env.conf
else
    ${FW_SETENV} /dev/${DEV_NAME} -s ${BOOT_DIR}/${TARGET_OS}_env.conf
fi
//uboot 的命令行信息为：bootargs console=ttyAMA0,115200n8 root=/dev/mmcblk0p2
rootfstype=ext4    rootwait    init=/sbin/init    systemd.show_status=false
g_ether.host_addr=82:cf:ce:fa:44:18 console=tty1
# write ext4 image
${SD_UPDATE} -d /dev/${DEV_NAME} -p ${PARTMAP}     //SD 卡更新分区表，SD 卡更
新的系统存在该路径下，并且文本文件列表有列出
if [ $? -ne 0 ]; then     //最后命令的退出状态不等于 0，则烧写错误，停止烧写
    echo "Error: filesystem fusing failed, Stop."
    exit 1
fi
if [ -z ${ARCH} ]; then   //如果 ARCH 为空值：判断烧写的系统是在 arm 架构还是 x86
架构上运行
    partprobe /dev/${DEV_NAME} -s 2>/dev/null    //如果不在板子上运行，则判断
为 PC 机运行
fi
if [ $? -ne 0 ]; then    //最后命令的退出状态不等于 0，则读取分区 table 失败
```

```
    echo "Warn: Re-read the partition table failed"
else
    # optional: update uuid & label
    if [ "x${TARGET_OS}" = "xandroid" ]; then    //判断烧写 TARGET_OS 是否等
于 android,加上 x 表示一字符串
        sleep 1
        ${SD_TUNEFS} /dev/${DEV_NAME}
    elif [ "x${TARGET_OS}" = "xdebian" ]; then
        sleep 1
        resize2fs -f /dev/${DEV_PART}
    fi
fi
echo "--------------------------------"
echo "${TARGET_OS^} is fused successfully."
echo "All done."
```

上面我们已经讲解了 fusing.sh 脚本文件，这次我们来讲解一下 mkimage.sh 脚本文件，以下是 mkimage.sh 脚本文件的解释说明。

```
if[$(id -u) -ne 0];then
echo "Re-running script under sudo..."
//在 sudo 权限下运行
    sudo "$0" "$@" //0:脚本本身的名字 ;@ 传给脚本的所有参数列表
    exit
    fi

    # ------------------------------------------------------------
    # Get target OS
    case ${1,,} in
    debian)
    TARGET_OS=debian ;;
    eflasher)
    TARGET_OS=eflasher ;;
    *)
    TARGET_OS=android ;;
    esac
    //从 SD 卡获得内核名称
    # ------------------------------------------------------------
    # Create zero file 创建一个零文件
```

```
RAW_FILE=nanopi2-${TARGET_OS}-sd4g.img  //文件名为 NanoPi2-xxx-sd4g.img
RAW_SIZE_MB=3900    //RAW 文件大小
BLOCK_SIZE=1024     //每个 block 为 1024byte
let RAW_SIZE=(${RAW_SIZE_MB}*1000*1000)/${BLOCK_SIZE}
echo "Creating RAW image: ${RAW_FILE} (${RAW_SIZE_MB} MB)"
echo "--------------------------------"
if [ -f ${RAW_FILE} ]; then   //如果已经存在 RAW 文件，则删除，重新 dd 一个 RAW 文件
rm -f ${RAW_FILE}
fi
dd if=/dev/zero of=${RAW_FILE} bs=${BLOCK_SIZE} count=0 \
seek=${RAW_SIZE} || exit 1
sfdisk -u S -L -q ${RAW_FILE} 2>/dev/null << EOF
2048,,0x0C,-
EOF  //sfdisk 是在 raw 文件上创建分区，这是因为后续的 sd_update 这个程序如果检测到没有分区表就会直接退
先随便创建一个
if [ $? -ne 0 ]; then    //$? 是显示最后命令的退出状态，0 表示没有错误，其他表示有错误
//-ne：第一个数不等于（Not Equal）第二个数。
echo "Error: ${RAW_FILE}: Create RAW file failed"
exit 1
fi
# -------------------------------------------------------------
# Setup loop device
Setup loop device // 所谓 loop device 指的就是获取文件,如文件系统映像(ISO、IMG 等)来模拟块设备
// 名称
losetup - 设 定 与 控 制 loop devices

语 法
losetup [ -e encryption ] [ -o offset ] loop_device file losetup [ -d ] loop_device

描 述
losetup 用 来 将 loop device 与 档 案 或 block device 联 结 、 分 离 . 以 及 查 询 loop device
状 况 , 如 只 给 定 loop_device 的 参 数 . 则 秀 出 loop device 目 前 的 状 况 .
在 Linux 中，有一种特殊的块设备叫 loop device,这种 loop device 设备是通过影射操作系统上的正常的文件而
块设备。因为这种设备的存在，就为我们提供了一种创建一个存在于其他文件中的虚拟文件系统的机制。示例第一步
命令创建文件
dd if=/dev/zero of=FS_on_file bs=1k count=10000
第二步：使用 losetup 命令创建一个 loop device
losetup /dev/loop0 FS_on_file

LOOP_DEVICE=$(losetup -f)  //判断 losetup 是否为文件
echo "Using device: ${LOOP_DEVICE}"
```

```
if losetup ${LOOP_DEVICE} ${RAW_FILE}; then
USE_KPARTX=1
PART_DEVICE=/dev/mapper/`basename ${LOOP_DEVICE}`
sleep 1
else
echo "Error: attach ${LOOP_DEVICE} failed, stop now."
rm ${RAW_FILE}
exit 1
fi
# ------------------------------------------------------------
# Fusing all
FUSING_SH=./fusing.sh   //fusing.sh 脚本文件已经在上一节介绍过
${FUSING_SH} ${LOOP_DEVICE} ${TARGET_OS}
RET=$?  //判断 RET 等于哪个文件
if [ "x${TARGET_OS}" = "xeflasher" ]; then   //如果烧写的文件等于目标文件系统
mkfs.vfat ${LOOP_DEVICE}p1 -n FriendlyARM   //创建 vfat 格式的文件系统
fi
# cleanup
losetup -d ${LOOP_DEVICE}   //判断是否为目录
if [ ${RET} -ne 0 ]; then  //判断 RET 是否等于 0
echo "Error: ${RAW_FILE}: Fusing image failed, cleanup"
rm -f ${RAW_FILE}
exit 1
fi
echo "--------------------------------"
echo "RAW image successfully created (`date +%T`)."
ls -l ${RAW_FILE}
echo "Tip: You can compress it to save disk space."
```

第六章　Debian 文件系统的基本操作

6.1 制作 Debian 系统

参考：https://wiki.debian.org/Debootstrap

1、安装 qemu、debootstrap 工具

```
#sudo apt-get install binfmt-support qemu qemu-user-static debootstrap
```

2、下载一个完整的 debian rootfs

```
#qemu-debootstrap --arch armhf jessie jessie-chroot
http://ftp.cn.debian.org/debian --verbose --include=wpasupplicant,openssh-clie
nt,openssh-server
```

3、验证签名

```
#apt-get install debian-archive-keyring
```

4、至此，你已经成功下载了一个 debian 的文件系统包，把它替换掉你 SD 卡的 debian 文件系统即可。

6.2 Debian 系统基本配置

6.2.1 分配固定 IP 地址

```
#vim /etc/network/interfaces.d/eth0
auto eth0 inet dhcp
allow-hotplug eth0
#ifcace eth0 inet dhcp
iface eth0 inet static
            address 192.168.1.x
            netmask 255.255.255.0
            gateway 192.168.1.1
            hwaddress ether fa:44:18:40:0c:01
#/etc/init.d/networking restart
```

6.2.2 自动更新时间

```
#vim /etc/cron.daily/timeupdate
ntpdate ntp.ubuntu.com

#chmod 755  /etc/cron.daily/timeupdate
#sync
#date ( Type 'date' to check the time )
```

开机测试：

- 在没有接网线的情况下， 开机后会停在
  eth0: device MAC address fa:44:18:40:0c:01

是因为没有接网线，文件系统在等待分配 IP 地址，如接了网线启动，则不会出现此问题。

- MAC 地址的分配：

```
#cat /etc/network/interfaces.d/eth0
```

```
auto eht0
allow-hotplug eth0
iface eth0 inet dhcp
hwaddress ether fa:44:18:40:0c:01
```

6.2.3 Debian/Android 系统截图方法

Debian:

```
apt-get  install  fbgrab
fbgrab  -d  /dev/fb0  screen.png
```

Android:

板子接 ADB shell，ADB shell 登录板子后，终端执行：

```
#screencap -p screen.png
```

退出 ADB shell，终端执行：

```
#adb pull /screen.png
```

6.2.4 设置防火墙

1、首先确定你的 NanoPi M3 已经连接上网络；

2、安装 iptables：

```
#apt-get install iptables
```

3、查看 Iptables 目前的配置信息：

```
#iptables -L
```

4、配置/etc/iptables.test.rules：

```
#vi   /etc/iptables.test.rules
```

```
<br>*filter

# Allows all loopback (lo0) traffic and drop all traffic to 127/8 that doesn't use lo0
-A INPUT -i lo -j ACCEPT
-A INPUT ! -i lo -d 127.0.0.0/8 -j REJECT

# Accepts all established inbound connections
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
# Allows all outbound traffic
# You could modify this to only allow certain traffic
-A OUTPUT -j ACCEPT

# Allows HTTP and HTTPS connections from anywhere (the normal ports for websites)
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT

# Allows SSH connections
# The --dport number is the same as in /etc/ssh/sshd_config
-A INPUT -p tcp -m state --state NEW --dport 22 -j ACCEPT

# Now you should read up on iptables rules and consider whether ssh access
# for everyone is really desired. Most likely you will only allow access from certain IPs.

# Allow ping
#   note that blocking other types of icmp packets is considered a bad idea by some
#   remove -m icmp --icmp-type 8 from this line to allow all kinds of icmp:
#    https://security.stackexchange.com/questions/22711
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT

# log iptables denied calls (access via 'dmesg' command)
-A INPUT -m limit --limit 5/min -j LOG --log-prefix  "iptables denied: "  --log-level 7

# Reject all other inbound - default deny unless explicitly allowed policy:
-A INPUT -j REJECT
-A FORWARD -j REJECT

COMMIT
```

5、配置文件写好之后就可以运行下面的命令来使用这些 rules：
（iptables 提供了保存当前运行的规则功能）

```
#iptables-save > /etc/iptables.up.rules

#iptables-restore < /etc/iptables.test.rules
```

6.2.5 配置有线网络

```
#cat /etc/network/interfaces
auto lo
iface lo inet loopback
```

```
#vim  /etc/network/interfaces
增加:
auto eth0
iface eth0 inet dhcp
```

```
#/etc/init.d/networking restart
```

6.2.6 搭建流媒体服务器 minidlna

- 安装编译 minidlna 所依赖的包

```
#apt-get build-dep minidlna -y
```

- 下载最新的 minidlna 的源码包

```
#wget
http://sourceforge.net/projects/minidlna/files/latest/download?source=files -O
minidlna.tar.gz
#tar -xvzf minidlna.tar.gz
```

- 进入主目录，解压，编译和安装

```
#./configure && make && make install
```

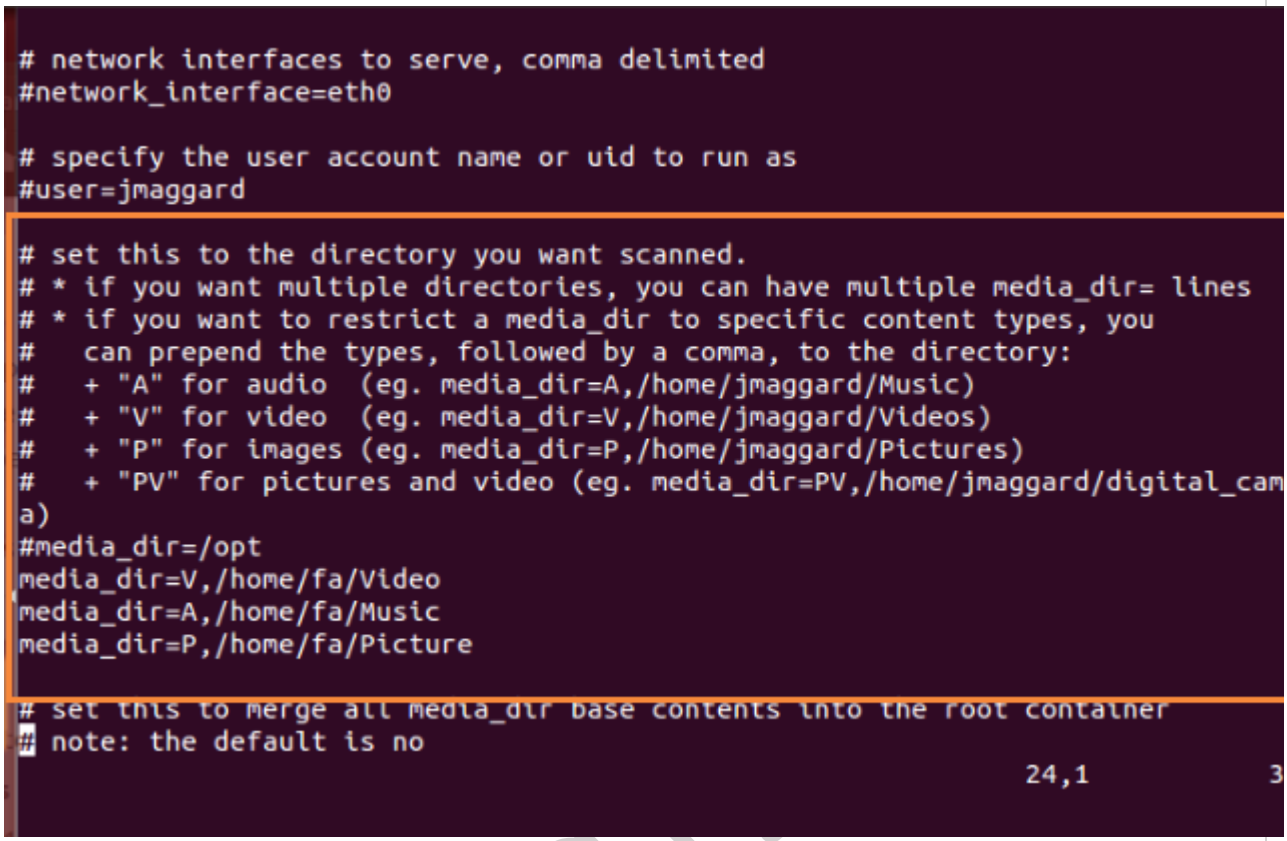
- 复制配置文件到相应的文件夹下，并且修改启动脚本权限

```
#cp  minidlna.conf    /etc/
#cp minidlna.init.d.script    /etc/init.d/minidlna
#chmod +x /etc/init.d/minidlna
```

- 设置默认的 miniDLNA 使用设置，并且编辑配置文件

```
#update-rc.d minidlna defaults
```

#vim /etc/minidlna.conf



- 重启服务，查看状态，显示正在运行的话，即可使用搭建好的环境

#service minidlna restart
#service minidlna status

6.2.7 配置 FTP 服务器

1、安装 vsftsps
#apt-get install vsftpd

2、创建 FTP 目录
#mkdir /var/ftp
#chmod 700 /var/ftp

3、对 FTP 的配置和使用

vsftpd 的去除匿名用户登录问题

#vi /etc/vsftpd.conf

anonymous_enable=YES
修改为：anonymous_enable=NO

允许本地用户登录
local_enable=YES

如何添加一个新用户

由于我是用本地用于登录的模式，所以确定你的 local_enable=YES 已经开启，再做下面的工作

首先添加一个用户组
#groupadd ftpgroup

然后添加用户

#useradd JJM -g ftpgroup -d /var/ftp -s /bin/bash
#passwd blogguy_cn

输入新密码，即可生效

如何限制新添加的用户不能使用 bash 登录

首先检查的你的 shells 文件

#vi /etc/shells

usermod -s /bin/false JJM

查看你的 passwd 文件是否生效

#cat /etc/passwd
#chown root:JJM /var/ftp
#/etc/init.d/vsftpd restart

第七章　搭建 Qt 开发环境及测试程序

7.1 搭建 Qt 开发环境

搭建 Qt 开发环境：
运行平台：Debian

1、做好启动卡；

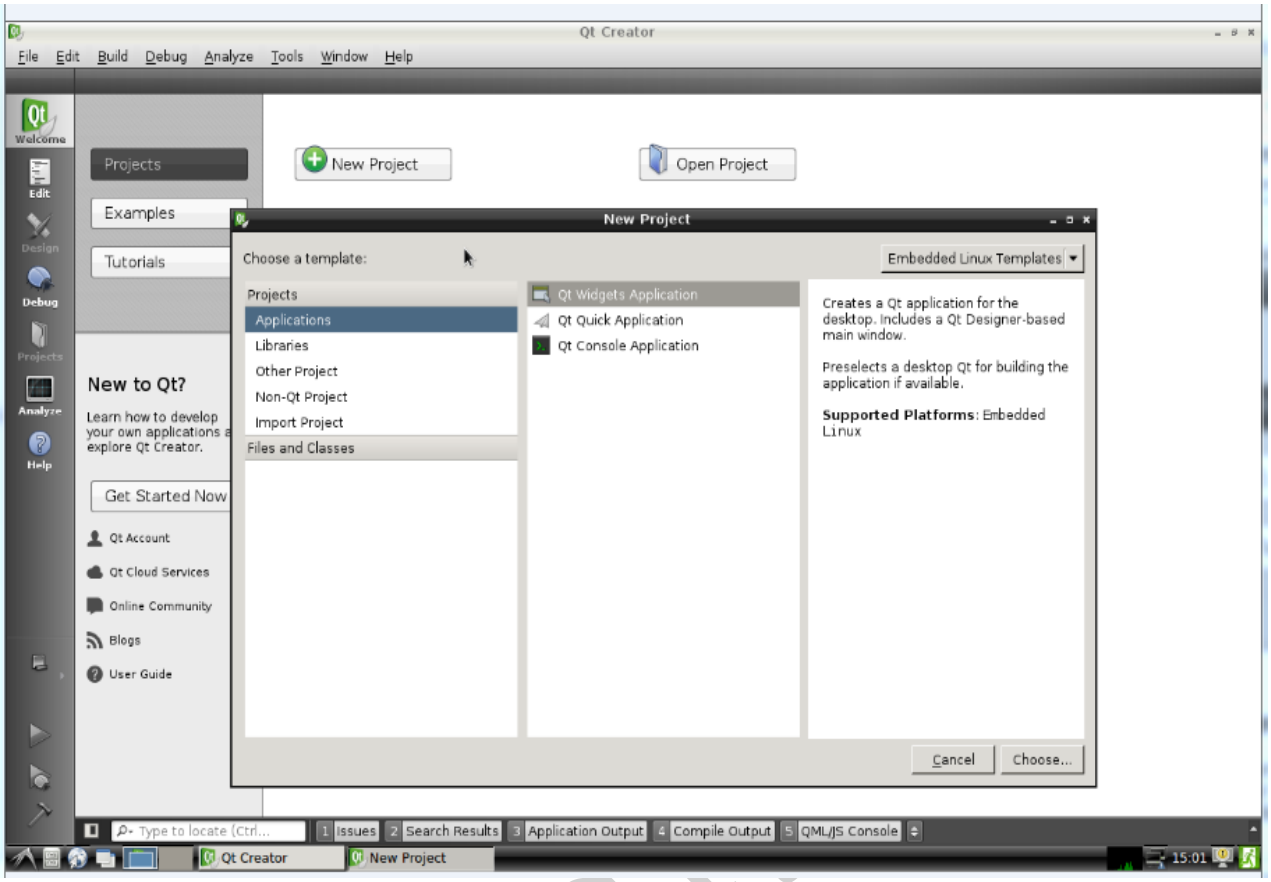
2、开发板启动后，配置好网络，确定网络正常使用；

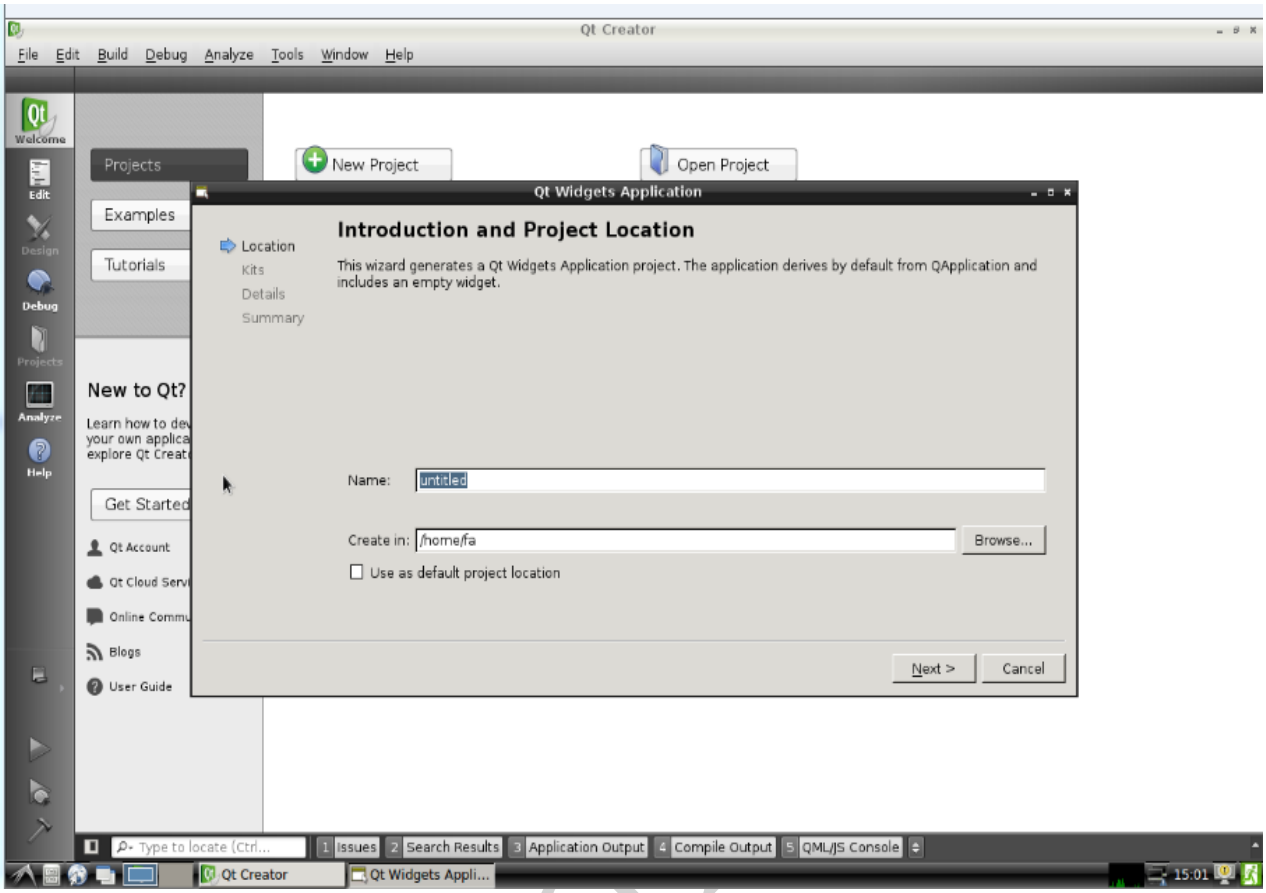
3、网络配置后，需要安装以下安装包：

```
apt-get update
apt-get install g++
apt-get install qt4-dev-tools
apt-get install Qtcreator
apt-get install libqt4-dev libqt4-gui libqt4-sql qt4-dev-tools qt4-doc qt4-designer qt4-qtconfig libqt4-dbg
```

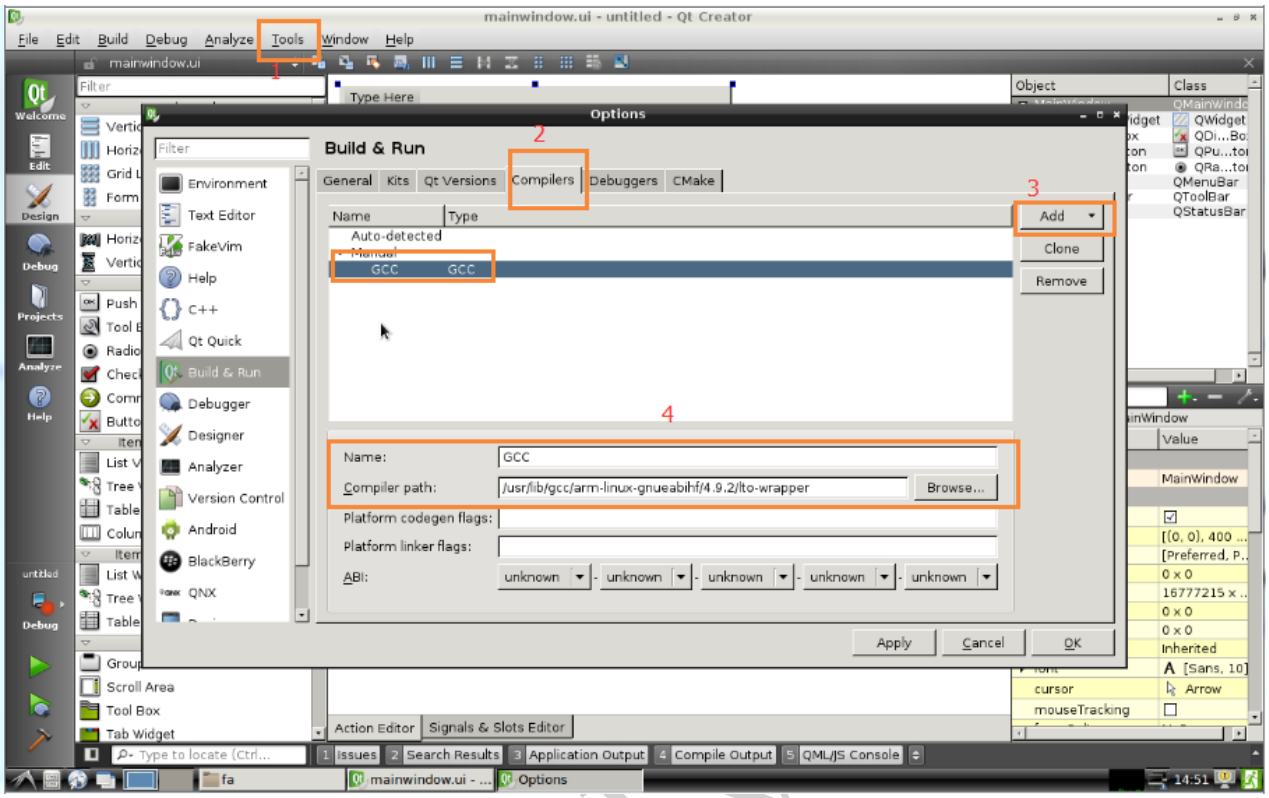
4、安装好后，开发板接上屏，即可在板子上点击 programing 看到 Qt 开发环境以及装好了：

- 点击-->Qt Creator-->New Project 新建一个 Qt Widgets Application;

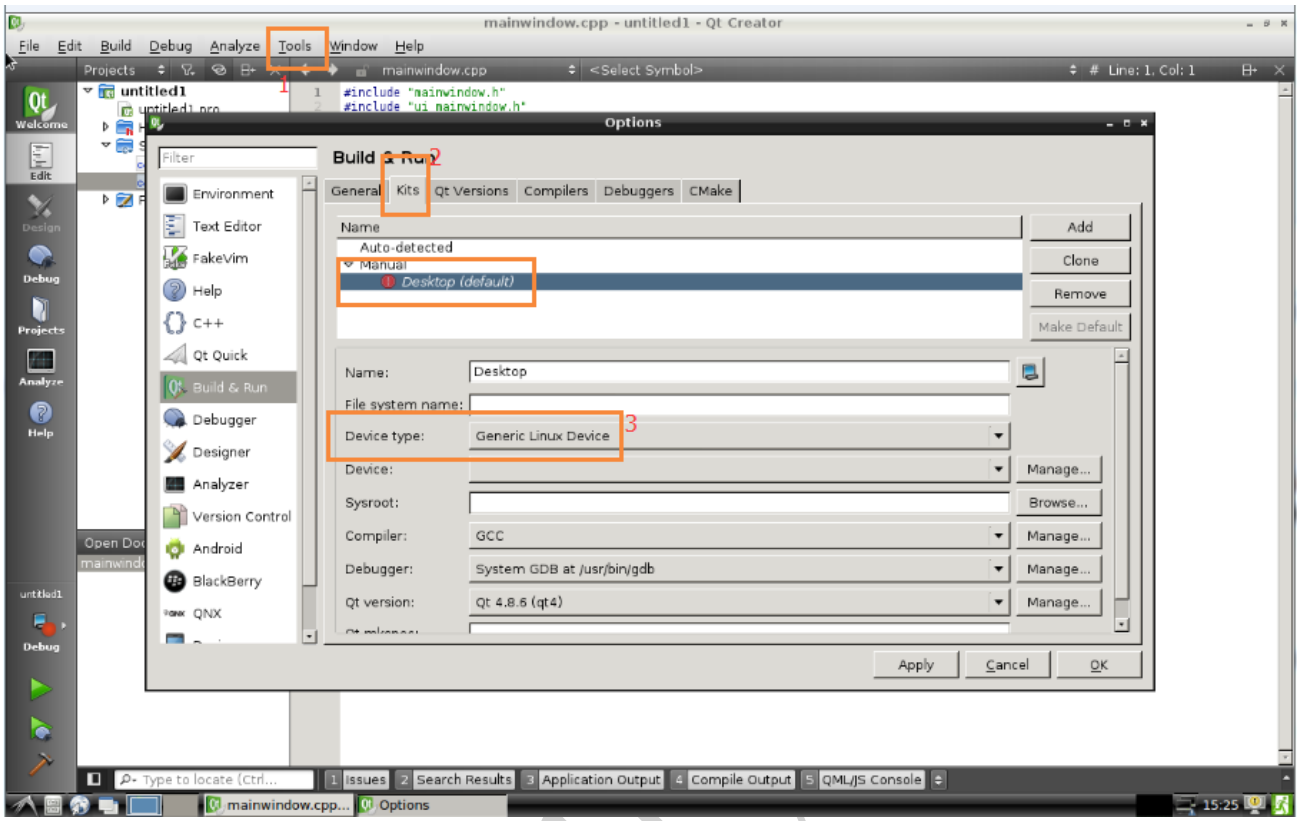




- 新建好工程之后，对编译器进行配置：点击 tools-->option-->Build & Run -->Compiler-->Add ，添加 GCC，并且选上编译器路径；

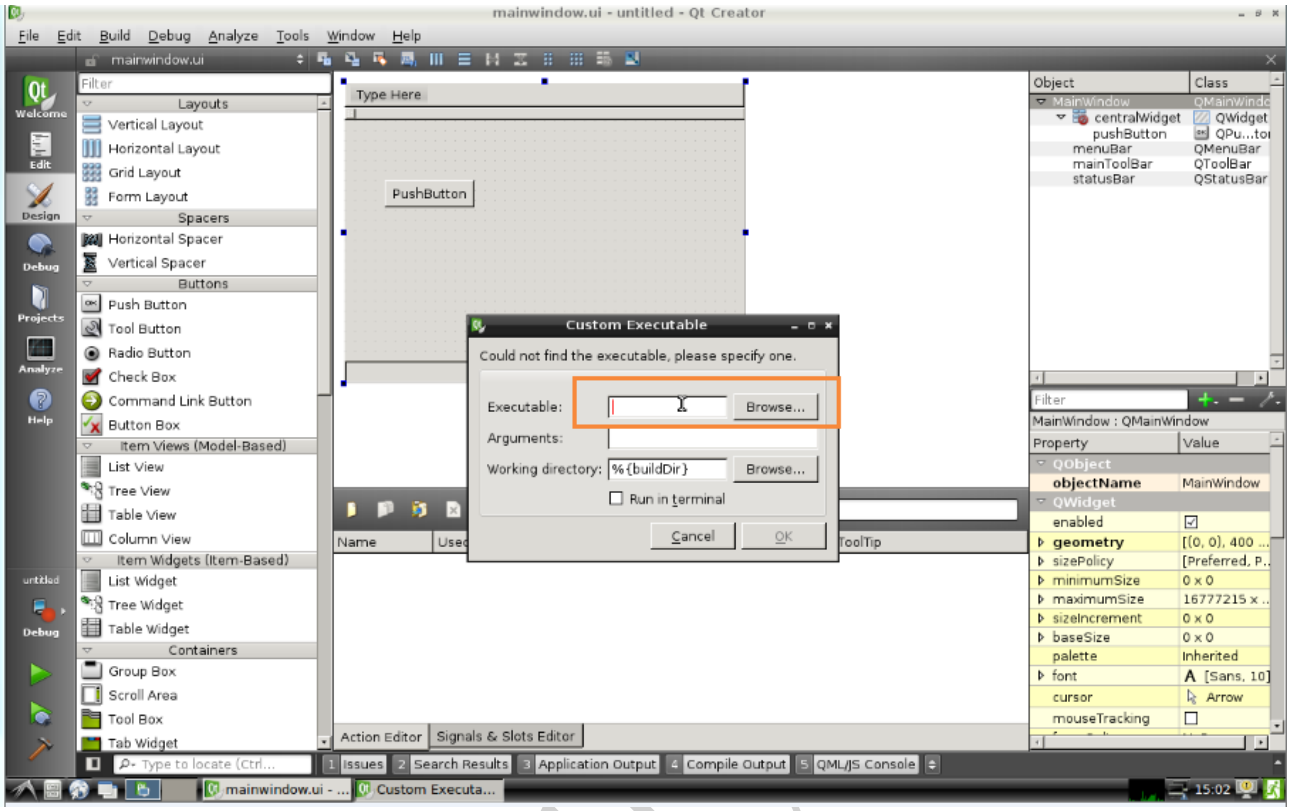


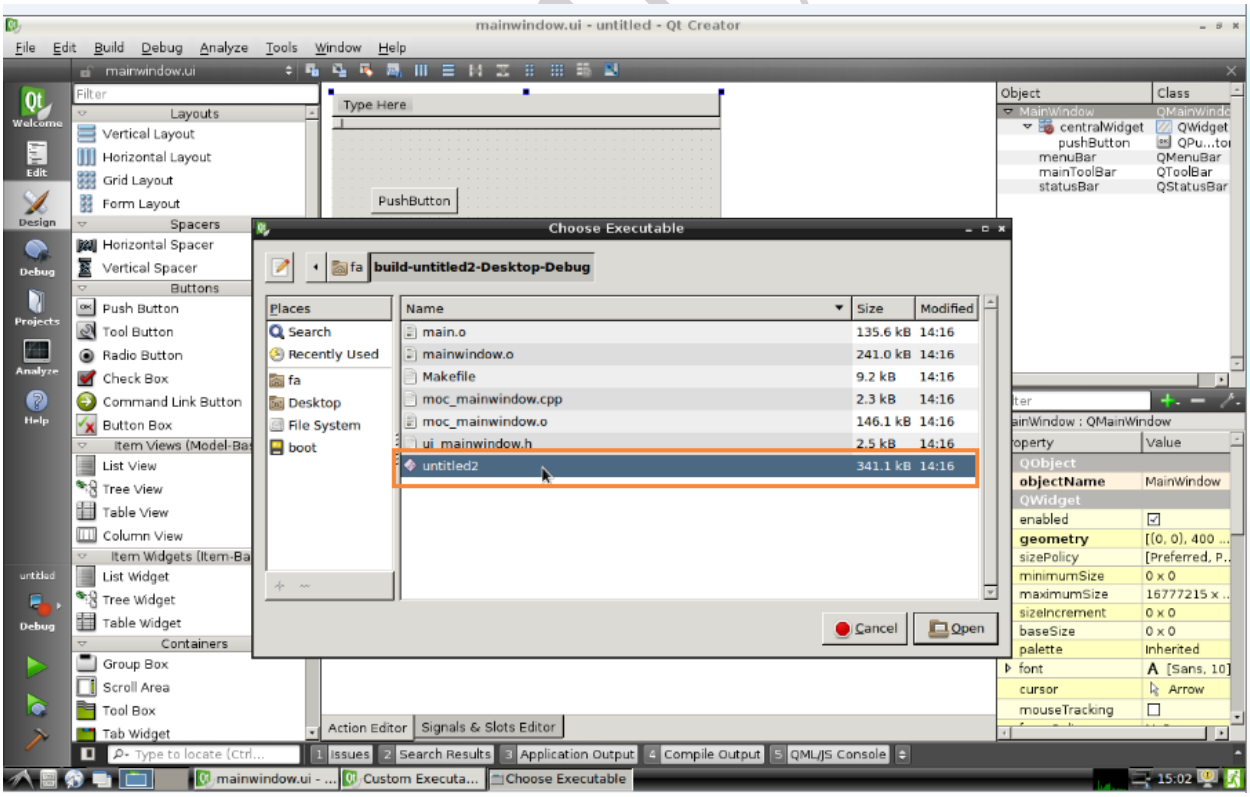
- 配置好编译器路径，需要再配置 Kits：tools-->option-->Build & Run-->Kits-->Desktop 选中 Generic Linux Device：

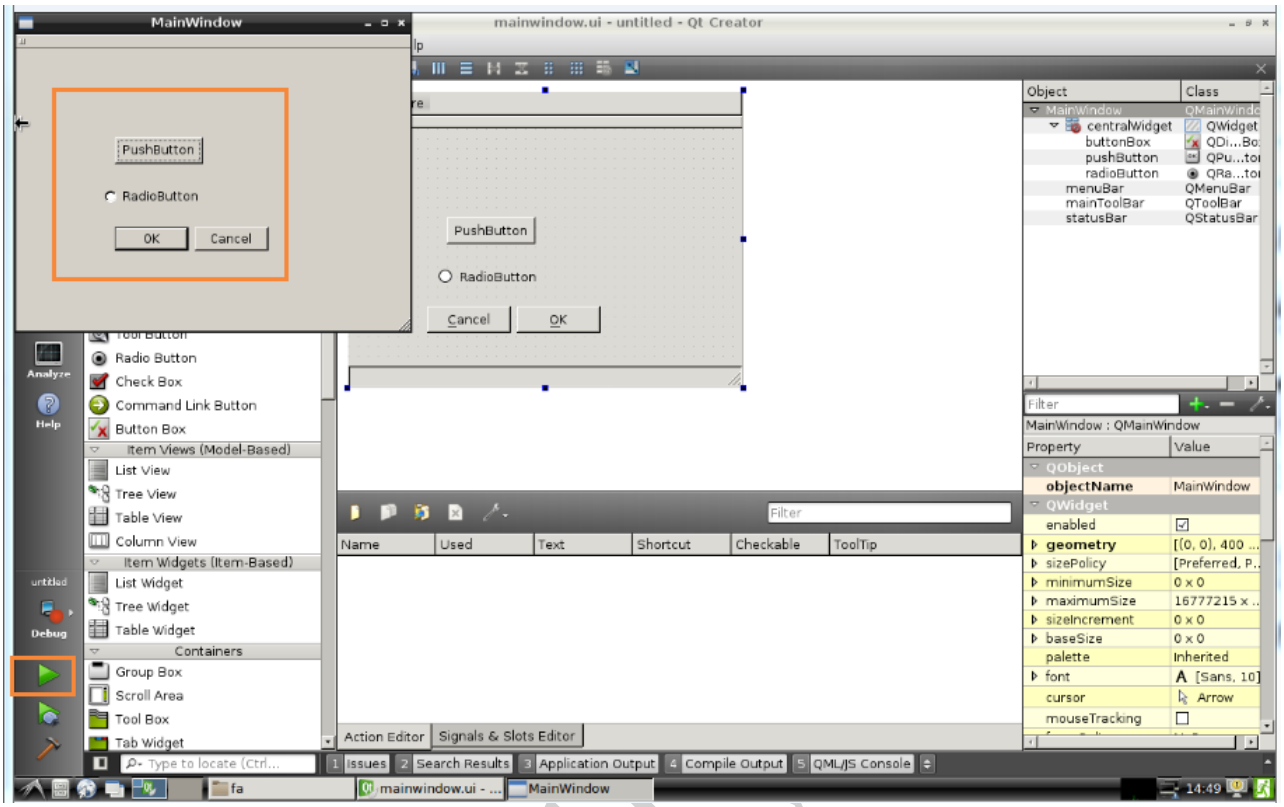


- 路径配置好之后，则点击运行按钮进行编译，编译过程中，会弹出

Custom Executable 窗口，点击 Browse 选中过程固件：


mainwindow.ui - untitled - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Custom Executable
Could not find the executable, please specify one.
Executable:
Browse...
Arguments:
Working directory:
%{buildDir}
Run in terminal
Cancel
OK
PushButton
Type Here
Action Editor
Signals & Slots Editor

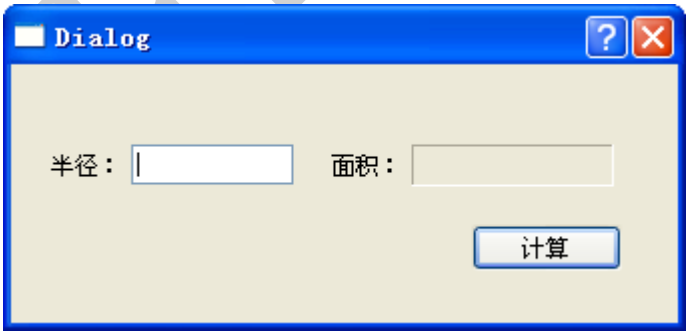

mainwindow.ui - untitled - Qt Creator
Choose Executable
build-untitled2-Desktop-Debug
Places
Search
Recently Used
fa
Desktop
File System
boot
Name
Size
Modified
main.o 135.6 kB 14:16
mainwindow.o 241.0 kB 14:16
Makefile 9.2 kB 14:16
moc_mainwindow.cpp 2.3 kB 14:16
moc_mainwindow.o 146.1 kB 14:16
ui_mainwindow.h 2.5 kB 14:16
untitled2 341.1 kB 14:16
Cancel
Open



7.2 编写第一个 Qt 程序

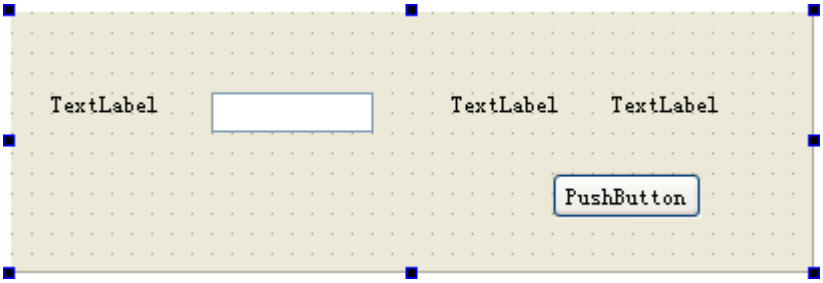
计算圆面积应用

Qt 是一个跨平台的 C++图形用户界面应用程序框架。它为应用程序开发者提供建立艺术级图形用户界面所需的所有功能。



1、创建一个新项目，命名为 Dialog，类名为 Dialog，基类为 QDialog；

2、新建项目后，在容器栏中选择以下控件，编辑 ui 如下：



3、编辑以上边框的内容为：

Class	text	objectName
QLabel	半径：	radiusLabel
QLineEdit		radiusLineEdit
QLabel	面积：	areaLabel_1
QLabel		areaLabel_2
QPushButton	计算	countBtn

4、编写相应的计算圆面积代码：
main.cpp 文件：

```
#include "dialog.h"
#include <QApplication>
int main(int argc, char *argv[])
{
QApplication a(argc, argv);
Dialog w;
w.show();
return a.exec();
}
```

5、在 LineEdit 文本框内输入半径值，然后单击“计算”按钮，则在 areaLabel_2 中显示对应的圆面积。编写代码步骤如下。

（1）在“计算”按钮上单击鼠标右键，在弹出的下拉菜单中选择“转到槽 ...”命令，在弹出的对话框中选择“ clicked()”信号；

（2）进入 dialog.cpp 文件中按钮单击事件的槽函数 on_countBtn_clicked()。信号与槽连接的具体说明参照本书后面提供的知识点链接部分。在此函数中添加如下代码：

```
void Dialog:: on_countBtn_clicked()
{
bool ok;
QString tempStr;
QString valueStr=ui->radiusLineEdit->text();
int valueInt=valueStr.toInt(&ok);
double area=valueInt*valueInt*PI;// 计算圆面积
ui->areaLabel_2->setText(tempStr.setNum(area));
}
```

（3）在此文件开始处添加以下语句：

```
const static double PI=3.1416;
```

（4）在上述工程的 dialog.h 中添加如下加黑代码：

```
class Dialog : public QDialog
{
Q_OBJECT
public:
Dialog(QWidget *parent = 0);
~Dialog();
private:
QLabel *label1,*label2;
QLineEdit *lineEdit;
QPushButton *button;
};
```

此时要在文件最开始加入头文件：

```
#include <QLabel>
#include <QLineEdit>
```

```
#include <QPushButton>
```

（5）在 dialog.cpp 中添加如下代码：

```
Dialog::Dialog(QWidget *parent)
: QDialog(parent)
{
label1=new QLabel(this);
label1->setText(tr(" 请输入圆的半径： "));
lineEdit=new QLineEdit(this);
label2=new QLabel(this);
button=new QPushButton(this);
button->setText(tr(" 显示对应圆的面积 "));
QGridLayout *mainLayout=new QGridLayout(this);
mainLayout->addWidget(label1,0,0);
mainLayout->addWidget(lineEdit,0,1);
mainLayout->addWidget(label2,1,0);
mainLayout->addWidget(button,1,1);
}
```

在此文件一开始添加头文件：

```
#include <QGridLayout>
```

（7）打开 dialog.h 文件，在类构造函数和控件成员声明后，添加如下代码：

```
class Dialog : public QDialog
{
... ...
QPushButton *button;
private slots:
void showArea();
};
```

（8）打开 dialog.cpp 文件，在构造函数中添加如下加黑代码：

```
Dialog::Dialog(QWidget *parent)
: QDialog(parent)
{
... ...
mainLayout->addWidget(button,1,1);
connect(button,SIGNAL(clicked()),this,SLOT(showArea()));
}
```

（9）在此文件开始处添加全局变量:

```
const static double PI=3.1416;
```

（10）在 LineEdit 中输入圆半径值，单击“显示对应圆的面积”按钮后，在 label2 中显示圆面积值.

第八章　对 NanoPi M3 开发板的基本操作

8.1 查看 CPU 温度

#cat　/sys/class/hwmon/hwmon0/device/temp_label

8.2 UbuntuCore+Qt 去掉 Qt 界面

修改 /etc/rc.local，去掉对 qt 的调用

8.3 Debian 系统去掉图形界面

NanoPi M3Debian 系统的使用跟普通的 Linux 系统一样　的方法，现在用的是 systemd
执行：
#systemctl　set-default　multi-user.target

8.4 修改 CPU 频率

这时候需要修改内核源码，并且重新编译内核。
内核源码位于：arch/arm/plat-s5p6818/nanopi3/include/cfg_main.h
CFG_I2C0_CLK
CFG_I2C1_CLK

8.5 降低 CPU 运行频率

/sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
用于锁定最高频率

8.6 命令行参数

```
-bootargs console=ttyAMA0,115200n8 root=/dev/mmcblk0p2 rootfstype=ext4 rootwait
init=/sbin/init systemd.show_status=false g_ether.host_addr=82:cf:ce:fa:44:18
```

```
+bootargs console=ttyAMA0,115200n8 root=/dev/mmcblk0p2 rootfstype=ext4 rootwait
init=/sbin/init systemd.show_status=false g_ether.host_addr=82:cf:ce:fa:44:18
console=tty1（把 frambuffer 当作控制台来用）
```

8.7 使用 lvds 接口

```
make  menuconfig
Device  Drivers    --->
    Graphics  support    --->
        Nexell  Graphics    --->
            [*]  LVDS
            (0)      Display  In    [0=Display  0,  1=Display  1]
```

这样就能启用 LVDS 了

8.8 Debian 系统使用 USB 转串口使用

```
1、开发板连接网络后通过串口进入系统终端界面（使用 minicom ttyS0）；
2、查看开发板 IP 地址，打开另外一个终端界面 ssh 登录开发板；
3、用 USB 转串口线接到电脑和开发板上；
4、ssh 端打开 minicom ttyUSB0，进入 USB 转串口（配置此终端为 ttyUSB0）；
5、这时候在两个终端之间互相发送，是可以接收到数据的。
```

8.9 Swap 分区

```
dd  if=/dev/zero  of=/var/swapfile  bs=1M  count=128
chmod  0600  /var/swapfile
mkswap  /var/swapfile
swapon  -f  /var/swapfile
```

1、以上命令即可创建一个 128MB 的文件来做为虚拟内存(swap)，大小自己可以修改那个第一条命令的 128 即可， 完成后使用 free 可查看到已启用 Swap。

2、需要使用开启 swap 分区的内核，使用开启 swap 分区的内核替换掉 SD 卡的内核启动后，再执行以上命令即可。

8.10 更换 logo

如果想更换 logo，直接更换 boot 分区下的文件即可;

第九章　Matrix 函数库说明

Matrix API reference manual 库是有友善之臂编写并维护的一个用 C 语言写成的类库，主要是给 Matrix 配件模块使用，集成库非常丰富，除了常用到的 GPIO 库，还包括 I2C 库、SPI 库、UART 库和软件 PWM 库等。由于友善推出的模块均以 Matrix 命令，故把此库称为 Matrix API reference manual 库。

Matrix API reference manual 库包含了一个命令行工具 gpio，它可以用来设置 GPIO 管脚，可以用来读写 GPIO 管脚，甚至可以在 Shell 脚本中使用来达到控制 GPIO 管脚的目的。

9.1 API 函数接口说明

9.1.1 GPIO 函数接口说明

接口名称	参数与返回值说明	功能说明
int pintoGPIO(int pin)	参数说明： pin：传递的 GPIO 引脚编号 返回值说明： 成功返回 0，失败返回负数	用于计算传递的 GPIO 对应于开发板的 GPIO 的索引号
int exportGPIOPin(int pin)	参数说明： pin: GPIO 引脚编号 返回值说明： 成功返回 0，失败返回负数	通知系统需要导出控制的 GPIO 引脚编号，相当于执行命令 echo pin > /sys/class/gpio/export
int unexportGPIOPin(int pin)	参数说明： pin: GPIO 引脚编号 返回值说明： 成功返回 0，失败返回负数	通知系统取消导出某个 GPIO 引脚，相当于执行命令 echo pin > /sys/class/gpio/unexport
int setGPIOValue(int pin, int value)	参数说明： pin: GPIO 引脚编号 value: 传入 GPIOEnum.LOW 表示输出低电平，传入 GPIOEnum.HIGH 表示输出高电平 返回值说明： 成功返回 0，失败返回负数	对某个引脚输出高或低电平

int getGPIOValue(int pin)	参数说明： pin：GPIO 引脚编号 返回值说明： 成功返回 GPIOEnum.LOW 表示输出低电平，返回 GPIOEnum.HIGH 表示输出高电平，失败返回负数	查询某个引脚的状态（高或低电平）
int setGPIODirection(int pin, int direction)	参数说明： pin：GPIO 引脚编号 direction：传入 GPIOEnum.IN 表示输入，GPIOEnum.OUT 表示输出 返回值说明： 成功返回 0，失败返回负数	配置引脚功能为输出或者输入
int getGPIODirection(int pin)	参数说明： pin：GPIO 引脚编号 成功返回 GPIOEnum.IN 表示输入，返回 GPIOEnum.OUT 表示输出，失败返回负数	查询引脚功能（为输出或者输入）

9.1.2 读取 ADC 转换结果的接口说明

接口名称	参数与返回值说明	功能说明
int setI2CSlave(int fd, int slave)	参数说明： fd：I2C 设备的文件描述符 slave：I2C 设备地址，例如 EEPROM 设备一般是 0x50 返回值说明： 成功返回 0，失败返回-1	设置要操作的 I2C 设备地址，例如 EEPROM 设备一般是 0x50
int setI2CTimeout(int fd, int timeout)	参数说明： fd：I2C 设备的文件描述符 timeout：超时时间 返回值说明： 成功返回 0，失败返回-1	设置超时时间（ioctl I2C_TIMEOUT）
int setI2CRetries(int	参数说明：	设置重试次数（ioctl

fd, int retries)	fd: I2C 设备的文件描述符 retries: 重试次数 返回值说明: 成功返回 0，失败返回-1	I2C_RETRIES)
int I2CWritreByteTo(int fd, int pos, unsigned char byteData, int wait_ms)	参数说明: fd: I2C 设备的文件描述符 pos: 字节位置 byteData:要写入的数据 wait_ms: 等待指定的时间(毫秒) 返回值说明: 成功返回 0，失败返回-1	写一个字节的数据到 I2C 设备的指定位置，并等待指定的时间(毫秒)
int I2CReadByteFrom(int fd, int pos, int wait_ms);	参数说明: fd: I2C 设备的文件描述符 pos: 字节位置 wait_ms: 等待指定的时间(毫秒) 返回值说明: 成功返回 0，失败返回-1	从 I2C 设备指定的位置读一个字节的数据，并等待指定的时间(毫秒)

9.1.3 SPI 接口说明

接口名称	参数与返回值说明	功能说明
int setSPIWriteBitsPerWord(int spi_fd, int bits)	参数说明: spi_fd: SPI 设备的文件描述符 bits: 字长，单位是比特 返回值说明: 成功返回 0，失败返回负数	设置每次读 SPI 设备的字长，单位是比特。虽然大部分 SPI 接口的字长是 8 或者 16，仍然会有一些特殊的例子。需要说明的是，如果这个成员为零的话，默认使用 8 作为字长 (ioctl SPI_IOC_WR_BITS_PER_WORD)
int setSPIReadBitsPerWord(int spi_fd, int bits)	参数说明: spi_fd: SPI 设备的文件描述符 bits: 字长，单位是比特 返回值说明: 成功返回 0，失败返回负数	设置每次写 SPI 设备的字长，单位是比特 (ioctl SPI_IOC_RD_BITS_PER_WORD)
int setSPIBitOrder(int spi_fd, int order)	参数说明: spi_fd: SPI 设备的文件描述符	设备 SPI 传输时是先传输低比特位还是高比特位，可选

	order: 传 SPIEnum.MSBFIRST 或 SPIEnum.LSBFIRST 返回值说明: 成功返回 0，失败返回负数	的参数有 SPIEnum.MSBFIRST 和 SPIEnum.LSBFIRST
int setSPIClockDivider(int spi_fd, int divider)	参数说明: spi_fd: SPI 设备的文件描述符 divider: 分频系数，传入在 SPIEnum.java 中定义的以 SPI_CLOCK_开头的常量，例如: SPIEnum.SPI_CLOCK_DIV128 返回值说明: 成功返回 0，失败返回负数	设置 SPI 的分频系数
int setSPIDataMode(int spi_fd, int mode)	参数说明: spi_fd: SPI 设备的文件描述符 mode: SPI 设备的模式，可传入 SPIEnum.SPI_MODE0 ~ SPIEnum.SPI_MODE3 返回值说明: 成功返回 0，失败返回负数	设置 SPI 设备的模式
int SPItransferOneByte(int spi_fd, byte byteData,int spi_delay,int spi_speed,int spi_bits)	参数说明: spi_fd: SPI 设备的文件描述符 byteData:要写入 SPI 设备的数据 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特 返回值说明: 成功返回读到的数据，失败返回负数	同时发送与接收一个字节的数据，调用示例: int byteRet = SPItransferOneByte(spi_fd , 0xAA , 0 /*delay*/ , 500000/*speed*/ ,8/*bits*/) ;
int SPItransferBytes(int spi_fd, byte[] writeData, byte[] readBuff, int spi_delay, int spi_speed, int spi_bits)	参数说明: spi_fd: SPI 设备的文件描述符 writeData:要写入的数据 readBuff: 存放读取数据的缓冲区 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特 返回值说明: 成功返回 0，失败返回负数	同时发送与接收多个字节的数据
int writeBytesToSPI(int	参数说明:	写多个字节的数据到 SPI 设

spi_fd, byte[] writeData, int spi_delay, int spi_speed, int spi_bits)	spi_fd: SPI 设备的文件描述符 writeData:要写入的数据 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特 返回值说明： 成功返回 0，失败返回负数	备
int readBytesFromSPI(int spi_fd, byte[] readBuff, int spi_delay, int spi_speed, int spi_bits)	参数说明： readBuff: 存放读取数据的缓冲区 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特 返回值说明： 成功返回 0，失败返回负数	从 SPI 设备读取多个字节

9.1.4 文件操作接口说明

接口名称	参数与返回值说明	功能说明
int openHW(String devName , int flags)	参数说明： devName: 要写入数据的 flags: 打开文件的方式，例如可读可写还是只读打开 返回值说明： 成功返回文件描述符，出错返回-1。	打开设备。
int ioctlWithIntValue (int fd, int cmd, int value)	参数说明： fd:设备文件描述符 cmd: ioctl 命令 value:命令参数，限整数 返回值说明： 成功返回 0，出错返回-1。	执行设备的 ioctl 操作
int writeHW(int fd, byte[] data)	参数说明： fd: 要写入数据的文件描述符 data: 要写入的数据 返回值说明： 成功返回写入的字节数，出错返回-1。	向打开的设备或文件中写数据。

int readHW(int fd, byte[] buf, int len)	参数说明： fd：要读出数据的文件描述符 buf：存储数据的缓冲区 len：要读取的字节数 返回值说明： 成功返回读取的字节数，出错返回-1，如果在调 read 之前已到达文件末尾，则这次 read 返回 0。	从打开的设备或文件中读取数据。
int selectHW(int fd, int sec, int usec)	参数说明： fd：要查询的文件描述符 sen：阻塞等待数据多长时间（单位：秒） usec：阻塞等待数据多长时间 （单位：纳秒，1 毫秒=1000 纳秒） 返回值说明： 如果 fd 有数据可读，返回 1，如果没有数据可读，返回 0，出错时返回-1。	查询打开的设备或文件是否有数据可读。
void closeHW(int fd)	参数说明： fd：要关闭的文件描述符 返回值说明： 无	关闭指定的文件描述符

9.1.5 ADXL345 芯片接口说明

接口名称	参数与返回值说明	功能说明
int adxl34xRead(char *position)	参数说明： position：读取模块的芯片数据 返回值说明： 成功返回 0，失败返回负数	读取传递进文件系统的模块芯片，读取到芯片数据，则对数据进行计算并且读取数据

9.1.6 BMP180 芯片接口说明

接口名称	参数与返回值说明	功能说明
int bmp180Read(int type,int *data)	参数说明： type：识别读取温度值或气压值 data：计算温度值或气压值	读取传递进文件系统的模块芯片，读取到芯片数据，则对数据进行计算并且读取数据

	返回值说明： 成功返回 0，失败返回负数	

9.1.7 Common 函数接口说明

接口名称	参数与返回值说明	功能说明
int boardInit()	参数说明： 返回值说明： 成功则返回开发板型号	初始化开发板 GPIO
void clearLastError()		清除上一次错误
void setLastError(const char *fmt,...)	参数说明： fmt： 返回值说明：	设置上一次错误信息
int getLastError(char* dest,int maxlen)	参数说明： dest: maxlen: 返回值说明：	

9.1.8 GPIO 传感器函数接口说明

接口名称	参数与返回值说明	功能说明
int sensorInit(struct sensor *dev, int num)	参数说明： dev：传递的传感器设备 num：传递的传感器数目 返回值说明： 成功返回文件描述符，出错返回-1	初始化接入开发板的传感器
void sensorRead(int devFD, char *buf, int len)	参数说明： devFD:要读取传感器的文件描述符 buf:存储数据的缓冲区 len：要读取的字节数 返回值说明： 成功返回字节数大小，出错返回-1	读取传感器数值
void sensorDeinit(int devFD)	参数说明： devFD：要读取传感器的文件描述符	停止或关闭传感器

	返回值说明： 读取成功后关闭文件描述符	

9.1.9hmc5883 芯片函数接口说明

接口名称	参数与返回值说明	功能说明
int hmc5883HWInit(int devFD)	参数说明： devFD：要写入数据的文件描述符 返回值说明： 成功返回写入的数值，出错返回-1。	初始化接入 hmc5883 芯片数据
double hmc5883Read(int devFD)	参数说明： devFD：要写入数据的文件描述符 返回值说明： 成功返回写入的数值，出错返回-1。	指南针计算出的方向值
int hmc5883Init(int i2cDev)	参数说明： i2cDev：要写入的芯片使用 i2c 设备 返回值说明： 成功返回写入的数值，出错返回-1。	打开 i2c 设备，设置 hmc5883
void hmc5883DeInit(int devFD)	参数说明： devFD：要关闭数据的文件描述符	关闭文件描述符

9.1.10　Led 函数接口说明

接口名称	参数与返回值说明	功能说明
int getLedState(int ledID)	参数说明： ledID：要写入数据的 led 引脚编号 返回值说明： 成功返回 LED 引脚编号，出错返回-1。	获得 LED 引脚编号
int setLedState(int ledID, int ledState)	参数说明： ledID：要写入数据的 led 引脚编号 ledState:LED 显示状态 返回值说明：	设置 LED 显示状态

	成功返回写入的数值，出错返回-1。	

9.1.11 LCD1602 函数接口说明

接口名称	参数与返回值说明	功能说明
int mcpWriteCmd4(int devFD, unsigned char command)	参数说明： devFD：要写入数据的文件描述符 command：要写入的指令 返回值说明： 成功返回写入的数值，出错返回-1	写指令
int mcpWriteCmd8(int devFD, unsigned char command)	参数说明： devFD：要写入数据的文件描述符 command：要写入的指令 返回值说明： 成功返回写入的数值，出错返回-1	写八位指令
int mcpWriteData4(int devFD, unsigned char data)	参数说明： devFD：要写入数据的文件描述符 data：要写入的数据 返回值说明： 成功返回写入的数值，出错返回-1	写数据
int mcpWriteData8(int devFD, unsigned char data)	参数说明： devFD：要写入数据的文件描述符 data：要写入的数据 返回值说明： 成功返回写入的数值，出错返回-1	写八位数据
int mcpInit(int i2cDev)	参数说明： i2cDev：i2c 设备描述符 返回值说明： 成功返回写入的数值，出错返回-1	初始化芯片数据
void mcpDeInit(int devFD)	参数说明： devFD：要写入数据的文件描述符	关闭文件描述符
int LCD1602KeyDispChar(int devFD, unsigned char x, unsigned char y, unsigned char data)	参数说明： devFD：要写入数据的文件描述符 x：第 x 列字符串 y：第 y 行字符串 data：要写入的数据 返回值说明： 成功返回写入的数值，出错返回-1	在第 y 行第 x 列显示字符

int LCD1602KeyDispChar(int devFD, unsigned char x, unsigned char y, unsigned char *str)	参数说明： devFD：要写入数据的文件描述符 x：第 x 列字符串 y：第 y 行字符串 str：要写入数据的地址 返回值说明： 成功返回写入的数值，出错返回-1	在第 y 行第 x 列开始写字符
int LCD1602KeyDispLines (int devFD, char* line1, char* line2)	参数说明： devFD：要写入数据的文件描述符 line1：LCD 显示数据的第一行 line2：LCD 显示数据的第二行 返回值说明： 成功返回写入的数值，出错返回-1	显示字符串的行数
int LCD1602KeyInit(i2cDev)	参数说明： i2cDev：i2c 设备描述符 返回值说明： 成功返回写入的数值，出错返回-1	初始化 LCD1602
int LCD1602KeyClear(int devFD)	参数说明： devFD：要清除数据的文件描述符 返回值说明： 成功返回写入的数值，出错返回-1	清除写入的数据
Void LCD1602KeyDeInit(int devFD)	参数说明： devFD：要关闭数据的文件描述符	关闭文件描述符
int LCD1602GetKey(int devFD)	参数说明： devFD：要写入数据的文件描述符	或者 LCD1602 设备的数据

9.1.12 OLED 函数接口说明

接口名称	参数与返回值说明	功能说明
int OLEDInit(int cmdDatPin, int resetPin)	参数说明： cmdDatPin：要写入命令的引脚 resetPin：复位引脚 返回值说明： 成功返回写入的数值，出错返回-1。	初始化 OLED
void OLEDDeInit(int devFD)	参数说明： devFD：要关闭数据的文件描述符	关闭文件描述符
void	参数说明：	配置写入 LCD 设备字符的位

OLEDDisp8x16Char(int devFD, int x, int y,char ch)	devFD：要写入数据的文件描述符 x：LCD 显示数据的行 y：LCD 显示数据的列 ch：显示的字符 返回值说明： 成功返回写入的数值，出错返回-1	置
void OLEDDisp8x16Char(int devFD, int x, int y,char ch[])	参数说明： devFD：要写入数据的文件描述符 x：LCD 显示数据的行 y：LCD 显示数据的列 ch：显示的字符串 返回值说明： 成功返回写入的数值，出错返回-1	写入一整串字符串数据到 LCD 设备
int OLEDScreen(int devFD)	参数说明： devFD：要清楚数据的文件描述符 返回值说明： 成功返回写入的数值，出错返回-1	关闭文件描述符

9.1.13 PCF8591 芯片函数接口说明

接口名称	参数与返回值说明	功能说明
int pcf8591Read(int channel, int *value)	参数说明： channel：模拟输入通道 value：文件描述符传递的值 返回值说明： 成功返回写入的数值，出错返回-1。	读取 pcf8591 芯片的值

9.1.14 PWM 函数接口说明

接口名称	参数与返回值说明	功能说明
int pwmtoGPIO(int pwm)	参数说明： pwm：定义使用的是第几路 pwm 返回值说明： 成功返回写入的数值，出错返回-1。	定义使用 pwm 的 gpio 管脚
int PWMPlay(int pwm, int freq, int duty)	参数说明： pwm：定义使用的是第几路 pwm freq：频率	输出 pwm

	duty：占空比 返回值说明： 成功返回写入的数值，出错返回-1。	
int PWMStop(int pwm)	参数说明： pwm：定义使用的是第几路 pwm 返回值说明： 成功返回写入的数值，出错返回-1。	停止输出 pwm

9.1.15　ds18b20 温度传感器函数接口说明

接口名称	参数与返回值说明	功能说明
int ds18b20Read(char * temperature)	参数说明： **temperature** ：采集到的温度值 返回值说明： 成功返回写入的数值，出错返回-1。	读取模块采集到的温度值

第十章　硬件编程示例

NanoPi M3 支持市面上大部分的传感器以及硬件设备模块，经友善之臂测试过的传感器模块适合用在某种特定的工控场合，同时更方便创客、学生学习使用。教程简单易懂，友善之臂已经把函数接口封装好，传感器等硬件模块直接使用杜邦线连接即可使用，这时你便可以随心所欲控制你身边的硬件模块了，极客的世界从这里开始。

本章主要介绍开发板支持的各种模块的原理、接线图以及程序函数的说明。

10.1 编译运行测试程序

启动开发板并运行 Debian 系统，进入系统后克隆 Matrix 代码仓库：

```
apt-get update && apt-get install git
git clone https://github.com/friendlyarm/matrix.git
```

克隆完成后会得到一个名为 matrix 的目录。
编译并安装 Matrix：

```
cd matrix
make && make install
```

10.1.1　点亮你的第一个 LED

发光二极管简称 LED，由电子与空穴复合时能辐射出可见光，因而可以用来制成发光二极管。

控制板上原带的 LED 非常简单，只需要调用函数 setGPIOValue ()接口，直接给高低电平，即可控制 LED 亮灭。

代码清单：

```
int main(int argc, char ** argv)
{
    int pin = GPIO_PIN(7);
    int i, value, board;
    int ret = -1;

    if ((board = boardInit()) < 0) {
        printf("Fail to init board\n");
```

```
        return -1;
    }
    if (board == BOARD_NANOPI_T2)
        pin = GPIO_PIN(15);

    if (argc == 2)
        pin = GPIO_PIN(atoi(argv[1]));
    if ((ret = exportGPIOPin(pin)) == -1) {
        printf("exportGPIOPin(%d) failed\n", pin);
    }
    if ((ret = setGPIODirection(pin, GPIO_OUT)) == -1) {
        printf("setGPIODirection(%d) failed\n", pin);
    }
    for (i = 0; i < STATUS_CHANGE_TIMES; i++) {
        if (i % 2) {
            value = GPIO_HIGH;
        } else {
            value = GPIO_LOW;
        }
        if ((ret = setGPIOValue(pin, value)) > 0) {
            printf("%d: GPIO_PIN(%d) value is  %d\n", i+1, pin, value);
        } else {
            printf("setGPIOValue(%d) failed\n", pin);
        }
        sleep(1);
    }
    unexportGPIOPin(pin);
    return 0;
}
```

运行测试程序：

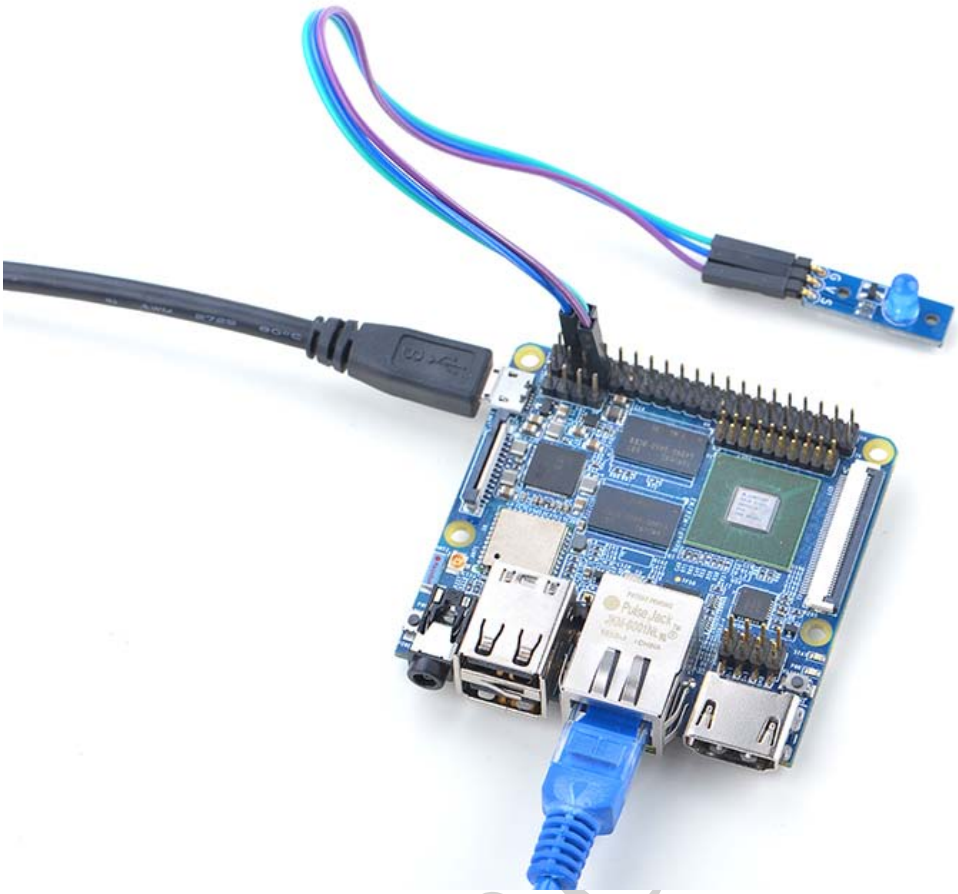
```
matrix-gpio_out
```

运行效果如下：

```
1: gpio status change
2: gpio status change
3: gpio status change
4: gpio status change
5: gpio status change
```

可以看到 LED 在不停地闪烁。

参考下图连接模块：



连接说明：

Matrix-LED	NanoPi M1
S	Pin7
V	Pin4
G	Pin6

10.1.2 内核操作 GPIO 示例

本示例演示 GPIO 接口的用法，将以操作在板 LED 为例，如果你外接的 LED，只需更换 pin 脚的序号即可。

与上一例子不同，本示例通过 GPIO 直接操作 LED，不再使用 Linux Kernel 的 LED 驱动，而且为了避免冲突，我们还需要将 Linux Kernel 的 LED 驱动先禁用，才能运行本示例。

禁用内核中的 LED：

你需要重新配置 Linux Kernel，在 make menuconfig 之后，进入以下路径 **Device Drivers**，找到 **LED Support** 驱动，去掉前面的钩选，然后重新编译内核并烧写到板子上。

10.2 使用 GPIO 示例

文件/sys/class/gpio 的使用前面已经详细介绍过，这里直接介绍各使用 gpio 引脚的模块原理和使用方法，以下介绍的模块均是使用 GPIO 引脚来驱动硬件模块。

10.2.1 无源蜂鸣器

一：无源蜂鸣器概述

一种一体化结构的电子讯响器，分为有源蜂鸣器与无源蜂鸣器。这里的“源”不是指电源，而是指震荡源，有源蜂鸣器内部带震荡源，所以只要一通电就会响，而无源内部不带震荡源，所以如果仅用直流信号无法令其鸣叫，必须用 2K-5K 的方波去驱动它。

无源蜂鸣器没有内部驱动电路，因此无源蜂鸣器工作的理想信号是方波。如果给预直流信号蜂鸣器是不响应的，因为磁路恒定，钼片不能振动发音。所以 GPIO 驱动无源蜂鸣器需要把 GPIO 的值拉高以后再拉低来产生振荡，而振荡的频率由 GPIO 从高拉低之间的时间决定，用户可以通过改变这个时间使无源蜂鸣器发出不同的声音。

二：硬件介绍

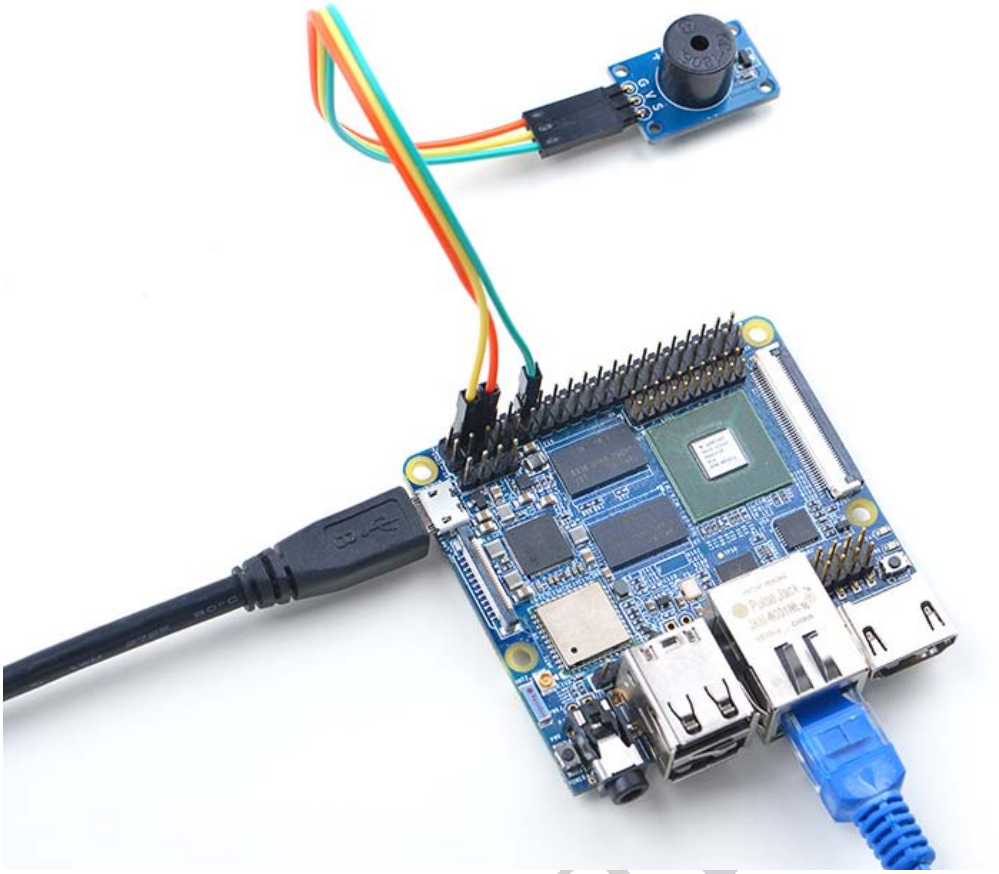
无源蜂鸣器的使用非常简单，直接使用板子上的 GPIO 管脚即可。通过软件设置不同的频率来控制无源蜂鸣器。

代码清单：

```
int main(int argc, char ** argv)
{
    int Hz, duty, board;
```

```
    if ((board = boardInit()) < 0) {
        printf("Fail to init board\n");
        return -1;
    }

    system("modprobe "DRIVER_MODULE);
    signal(SIGINT, intHandler);
    if (argc == 4) {
        // Usage:matrix-pwm channel freq duty[0~1000]
        pwm = atoi(argv[1]);
        Hz = atoi(argv[2]);
        duty = atoi(argv[3]);
    } else {
        Hz = 1000;
        duty = 500;
        printf("Using default config: channel=%d freq=%dHz duty=%d\n", pwm, Hz, duty);
    }
    if (PWMPlay(pwm, Hz, duty) == -1) {
        printf("Fail to output PWM\n");
    }
    printf("Press enter to stop PWM\n");
    getchar();
    PWMStop(pwm);
    system("rmmod "DRIVER_MODULE);

    return 0;
}
```

参考下图连接模块：



连接说明：

Matrix-Buzzer	NanoPi 2
G	Pin6
V	Pin4
S	Pin22

10.2.2 继电器

一：继电器概述

继电器是一种电子控制器件，它具有控制系统（又称输入回路）和被控制系统（又称输出回路），通常应用于自动控制电路中，它实际上是用较小的电流去控制较大电流的一种“自动开关”。故在电路中起

着自动调节、安全保护、转换电路等作用。

二：硬件设计

继电器的使用非常简单，直接给管脚供电即可工作。

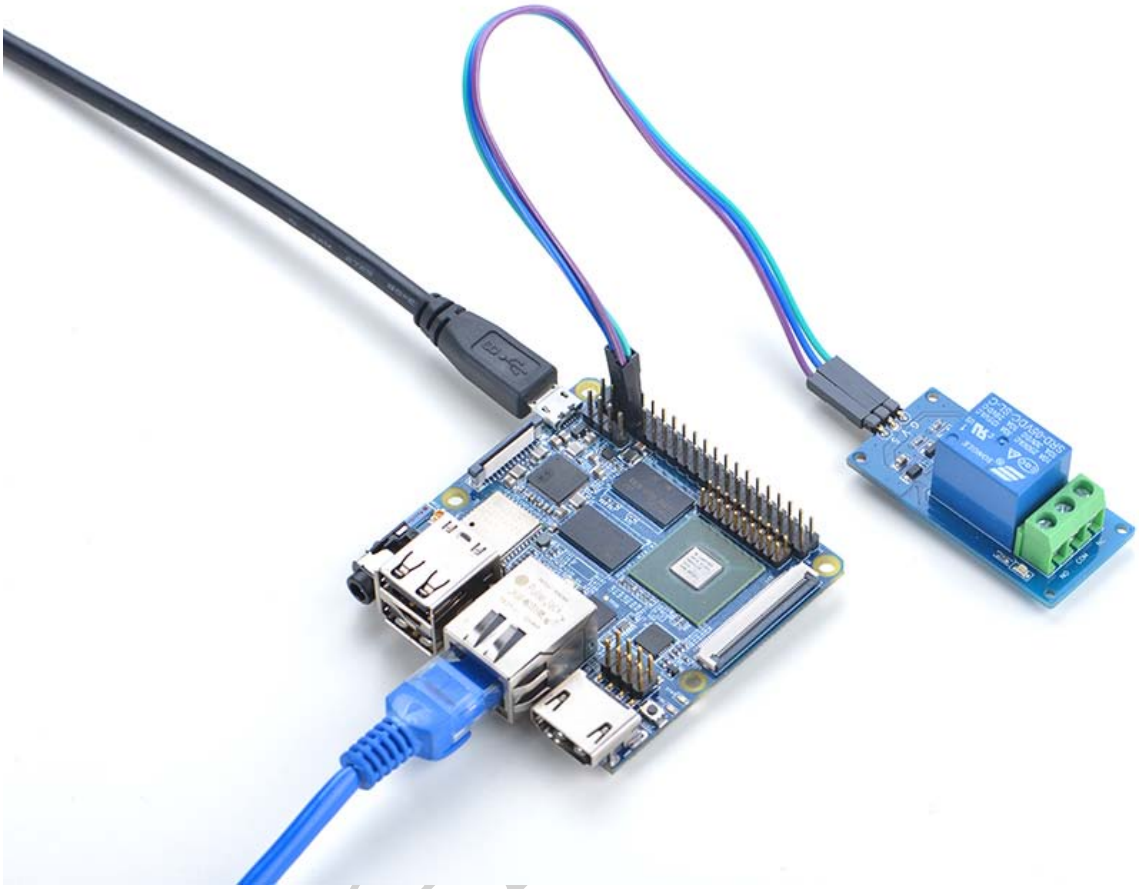
三：

代码清单：

```
int main(int argc, char ** argv)
{
    int pin = GPIO_PIN(7);
    int i, value, board;
    int ret = -1;

    if ((board = boardInit()) < 0) {
        printf("Fail to init board\n");
        return -1;
    }
    if (board == BOARD_NANOPI_T2)
        pin = GPIO_PIN(15);

    if (argc == 2)
        pin = GPIO_PIN(atoi(argv[1]));
    if ((ret = exportGPIOPin(pin)) == -1) {
        printf("exportGPIOPin(%d) failed\n", pin);
    }
    if ((ret = setGPIODirection(pin, GPIO_OUT)) == -1) {
        printf("setGPIODirection(%d) failed\n", pin);
    }
    for (i = 0; i < STATUS_CHANGE_TIMES; i++) {
        if (i % 2) {
            value = GPIO_HIGH;
        } else {
            value = GPIO_LOW;
        }
        if ((ret = setGPIOValue(pin, value)) > 0) {
            printf("%d: GPIO_PIN(%d) value is  %d\n", i+1, pin, value);
        } else {
            printf("setGPIOValue(%d) failed\n", pin);
        }
        sleep(1);
    }
```

```
    unexportGPIOPin(pin);
    return 0;
}
```

参考下图连接模块：



连接说明：

Matrix-Relay	NanoPi M1
S	Pin7
V	Pin4
G	Pin6

运行测试程序：

```
matrix-gpio_out
```

注意：此模块并不支持热插拔，启动系统前需要确保硬件连接正确。

运行效果如下：

```
1: gpio status change
```

```
2: gpio status change
3: gpio status change
4: gpio status change
5: gpio status change
```

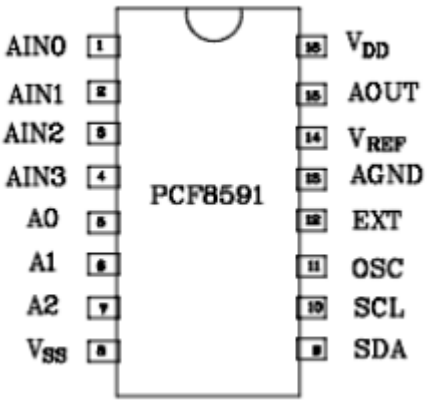
10.3 ADC 转换 PCF8591

一：PCF8591 概述

PCF8591 是单片、单电源低功耗 8 位 CMOS 数据采集器件，具有 4 路模拟输入通道、一路输出通道和一个串行 I2C 总线接口。3 个地址引脚 A0、A1 和 A2 用于编程硬件地址，允许将最多 8 个器件连接至 I2C 总线而不需要额外硬件。器件的地址、控制和数据通过两线双向 I2C 总线传输。器件功能包括多路复用模拟输入、片上跟踪和保持功能、8 位模数转换和 8 位数模拟转换。最大转换速率取决于 I2C 总线的最高速率。

二：硬件介绍

PCF8591 模块的工作电压为 2.5V-6V 之间，PCF8591 模块是用 i2c 总线进行串行输入/输出，因此模块上的 SDA、SCL 两个引脚可以接到开发板的 i2c-0 的 SDA、SCL 两个引脚上。



引脚描述：

Signal name	Description
AIN0	A/D 转换的模拟量输入通道
AIN1	
AIN2	
AIN3	
A1	地址引脚
A2　Vss	地

SDA	IIC 总线数据输入输出
SCL	IIC 总线时钟输入
OSC	时钟输入、输出
EXT	外部、内部时钟切换
AGND	模拟地
Vref	参考电压输入
AOUT	D/A 转换模拟量输出
Vdd	电源

模块引脚描述符：

Signal name	Type	Description
SDA	Bidirectional (双向)	IIC-bus data input/output
SCL		IIC-bus clock input

模块上还有 AINT0-AINT4 引脚为外部模拟输入可编程为单端或差分输入，要注意外接的模拟电压输入范围在 0—VDD。另外模块上还有三个红色短路帽，它们的作用分别如下：

P4：接上 P4 短路帽	选择热敏电阻接入电路
P5：接上 P5 短路帽	选择光敏电阻接入电路
P6：接上 P6 短路帽	选择 0-5V 可调电压接入电路

注意：如果需要使用 4 路外部电压输入，请将短路帽去取下.

三：代码清单

I2c 总线是通过发送有效的器件地址来对器件进行读写的，所以首先需要确定模块在 i2c 上的器件地址。该地址包括固定部分和可编程部分，如下图：

图 4 地址

模块中已经把可编程部分全接地置 0，而且 i2c 的器件地址只有 7 位，不包括 R/W 位，所以该模块的器件地址为 0x48。

在 pcf8591.c 文件中的语句：

```
unsigned char *devName = "/dev/i2c-0";
devFD = pcf8591_open(devName);
```

打开 i2c-0 设备并设置好器件地址了，之后就可以成功对模块进行读和写操作了。

在 pcf8591.c 文件中调用 pcf8591_AD_init(devFD); 语句把模块初始化为四个单端模拟电压输入，读

取 channel0 输出的 AD 转换。

当模块工作在 AD 转换时，调用语句：data = pcf8591_read_channel(devFD,mode,channel)把模块在你设定的模拟输入方式（mode 的值决定）和输出的通道（channel 的值决定）读到的数据存到 data 中。

代码清单：

```
int main(int argc, char ** argv)
{
    int i = 0;
    int value = 0;
    int channel = 0;

    if (boardInit() < 0) {
        printf("Fail to init board\n");
        return -1;
    }

    if (argc == 2)
        channel = atoi(argv[1]);
    system("modprobe "DRIVER_MODULE);
    signal(SIGINT, intHandler);
    for (i=0; i<ADC_READ_TIMES; i++) {
        if (pcf8591Read(channel, &value) != -1) {
            printf("The channel%d value is %d\n", channel, value);
        } else {
            printf("Fail to get channel%d value\n", channel);
        }
    }
    system("rmmod "DRIVER_MODULE);

    return 0;
}
```

连接说明：

Matrix-Analog_to_Digital_Converter	NanoPi M1
SDA	Pin3
SCL	Pin5
5V	Pin4
GND	Pin6

运行测试程序：

```
matrix-adc
```

注意：此模块并不支持热插拔，启动系统前需要确保硬件连接正确。

运行效果如下：

```
The channel0 value is 2460
```

当通道 0 接到 5V 时，能得到最大值 2550，当通道 0 接到地时，能得到最小值 0。

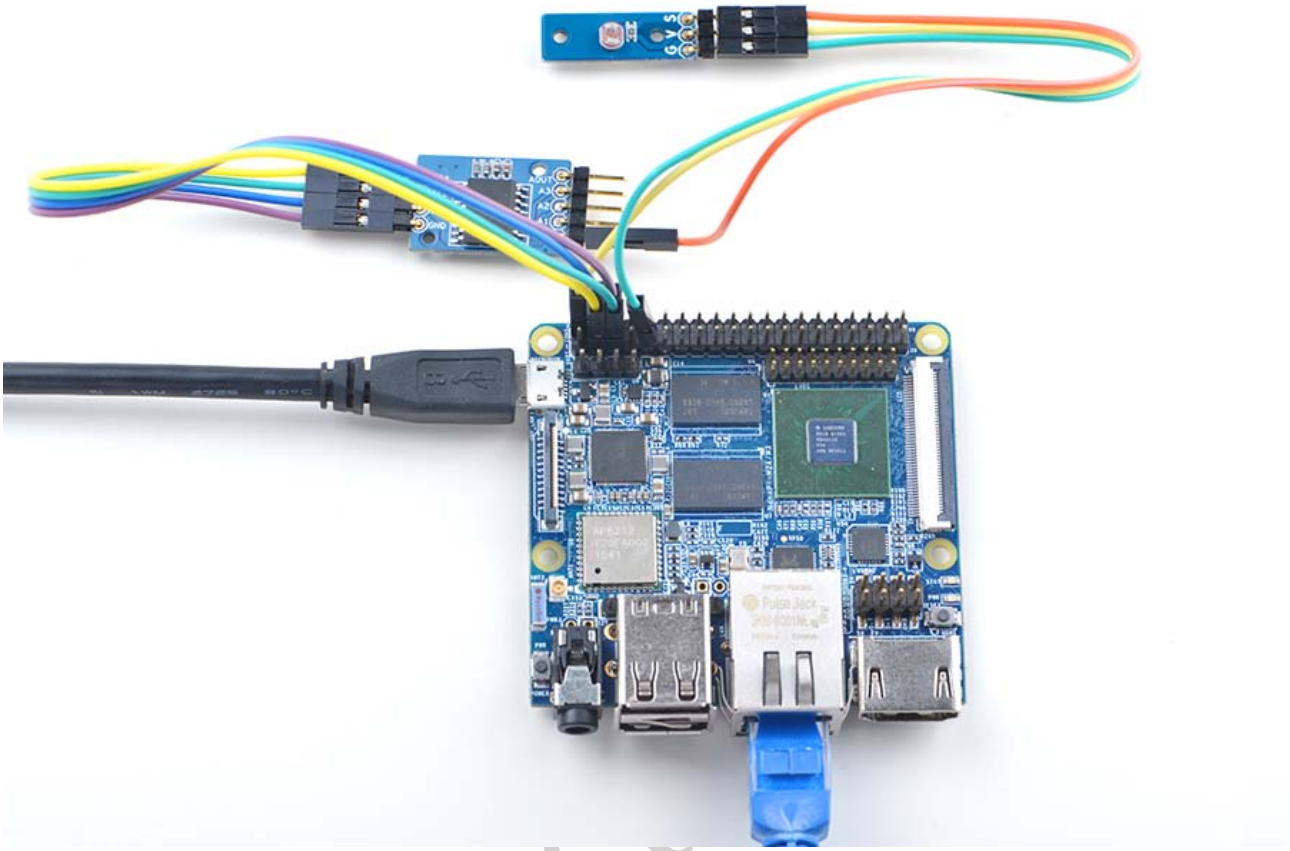
Matrix-Soil_Moisture_Sensor	
GND	NanoPi M1 Pin9
5V	NanoPi M1 Pin2
S	Matrix-Analog_to_Digital_Converter A0

10.3.1 光敏电阻

光敏电阻原理	工作形式	使用说明	读取值
利用光线亮度强弱检测原理，通过光线亮度来调节电阻大小。光强度增加，则电阻减小；光强度减小，则电阻增大。	比较器输出，配可调电位器调节检测光线亮度。工作电压在 3.3V-5V。数字开关量输出（0-1）	光敏电阻模块使用的是 adc 模拟输入输出，PCF8591 已经集成光敏电阻，直接使用即可读取信息。	four_signle_adc 文件用于使 pcf8591 工作于 AD 转换模式下 channel0-3 分别转换 AIN0-3 路的单端模拟输入，并在终端每秒采集一次数据

注意：光敏电阻使用的是 AD 转换，不用到 DA 转换，模块直接接 AO 口就可以。

参考下图连接模块：



10.3.2 土壤湿度传感器

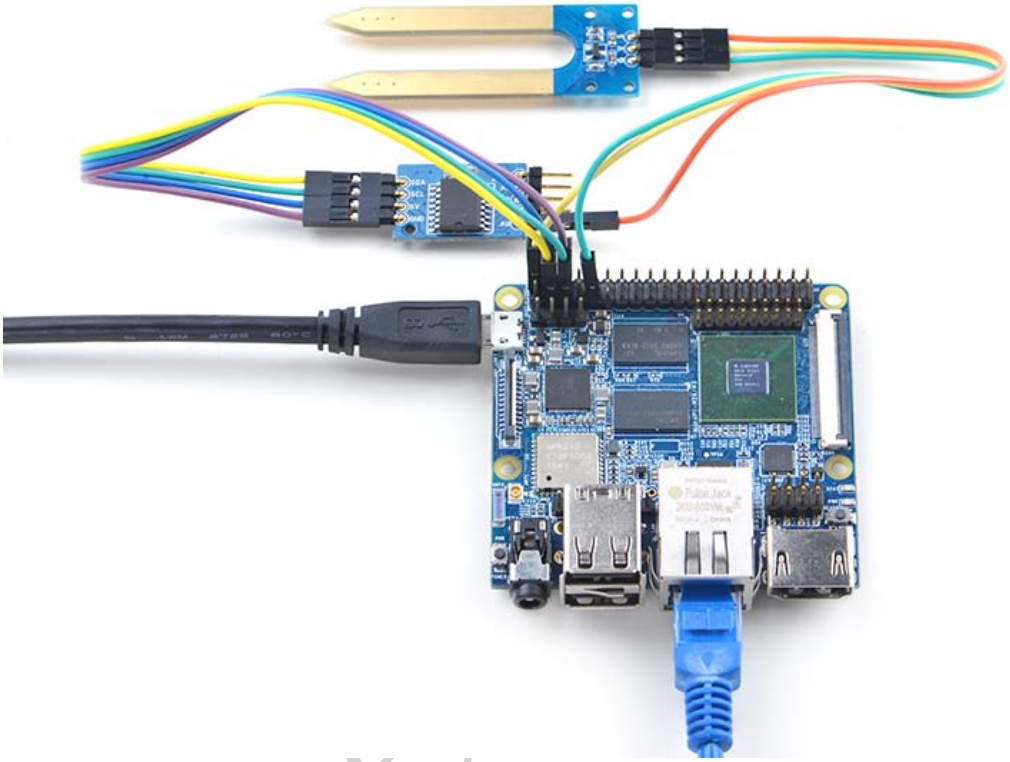
土壤湿度传感器	工作形式	使用说明	读取值
它利用电磁脉冲原理、根据电磁波在介质中传播频率来测量土壤的表观介电常数（ε），从而得到土壤容积含水量(θv)。	工作电压 3.3V-5V 模块双输出模式。土壤湿度传感器水分传感器可用于检测土壤的水分，当土壤缺水时，模块输出一个高电平，反之输出低电平。	传感器模块使用的是 adc 模拟输入输出，这里直接把传感器模块接到 PCF8591 转换模块上即可检查到土壤湿度值。	four_signle_adc 文件用于使 pcf8591 工作于 AD 转换模式下 channel0-3 分别转换 AIN0-3 路的单端模拟输入，并在终端每秒采集一次数据

注意：土壤湿度传感器使用的是 AD 转换，不用到 DA 转换，模块直接接 AO 口就可以。

土壤湿度传感器模块管脚：

土壤湿度传感器	
GND	接地
VCC	接电
AO	SIG

参考下图连接模块：



10.3.4 双轴按键摇杆

双轴按键摇杆原理	工作形式	使用说明	读取值
模块特设二路模拟输出和一路数字输出接口，输出值分别对应（X，Y）双轴偏移量，其类型为模拟量；按键表示用户是否在 Z	为了让客户更加方便地配合扩展板等标准接口，在设计上把 X,Y,Z 轴的电路都单独引出，以控制输入这个操纵杆模块的	传感器模块使用的是 adc 模拟输入输出，上面已经介绍过如何使用 ADC 转换模块，这里直接把传感器模块接到 PCF8591 转	

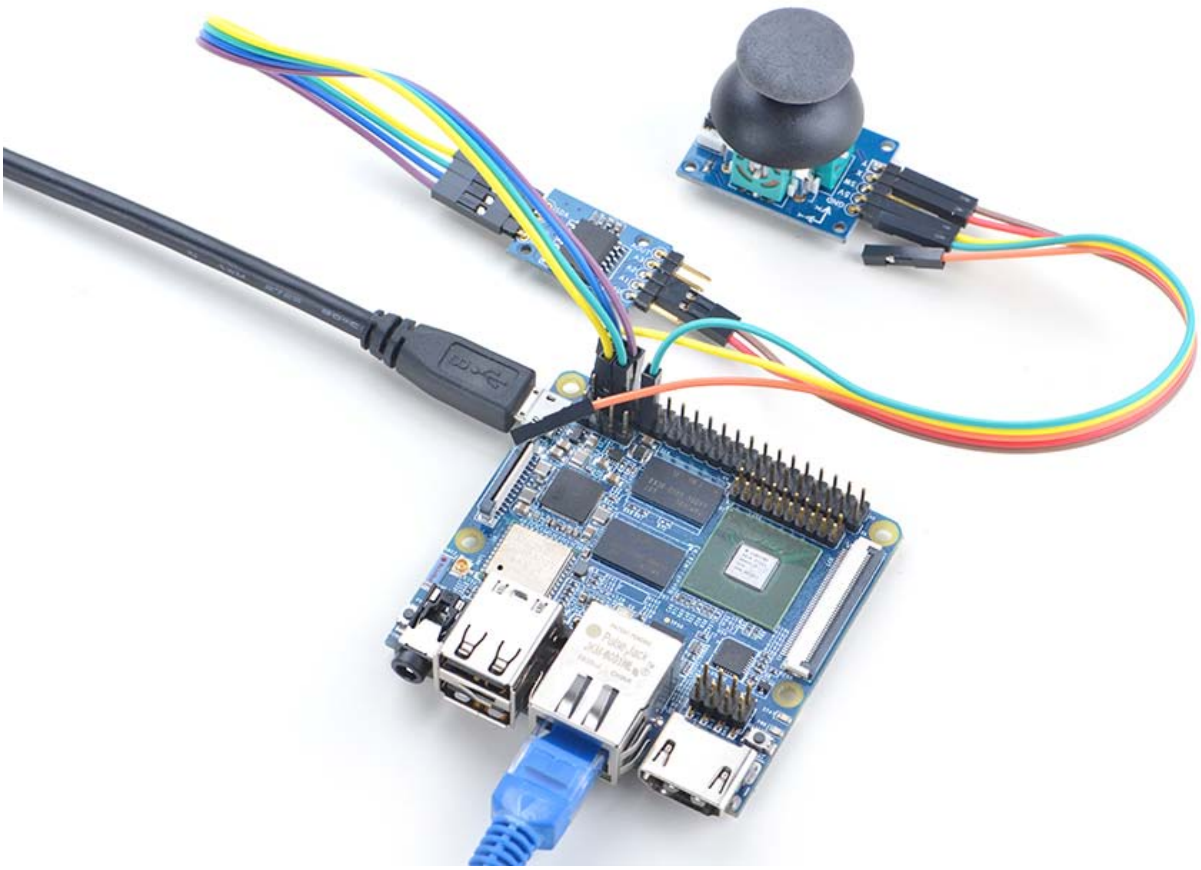
轴上按下，其类型为数字开关量。模块集成电源指示灯，可显示工作状态；坐标标识符清晰简明、准确定位。	x、y、z 的 PS2 摇杆戏摇杆模块 Joystick 值以及在特定的值下实现某种功能	换模块上即可对模块进行操作。	

摇杆模块管脚如下表所示：

PS2 摇杆	
GND	接地
VCC	+5V
VRX	PCF8951.AIN0
VRY	PCF8951.AIN1
SW	PCF8951.AIN2

PCF8951 模块对应四只脚连接到扩展板对应的 I2C 接口上，即可进行数据采集。

参考下图连接模块：



Matrix-Joystick	
GND	NanoPi M1 Pin9
5V	NanoPi M1 Pin2
X	Matrix-Analog_to_Digital_Converter A1
Y	Matrix-Analog_to_Digital_Converter A2

10.4 使用 IIC 通讯 LCD1602 液晶显示器

概述：
LCD1602 是指显示的内容为 16*2 ，即可显示两行，每行 16 个字符。

LCD1602 引脚介绍：

引脚名称	电平	输出/输出	作用
VSS			电源地
VCC			电源（+5V）
VO			对比调整电压
RS	0/1	输入	0=输入指令 1=输入数据
R/W	0/1	输入	0=向 LCD 写入指令或数据 1=从 LCD 读取数据
E	1,1--0	输入	使能信号，1=读取信息 1—0（下降沿）执行指令
D0	0/1	输入/输出	数据总线 line0（最低位）
D1	0/1	输入/输出	数据总线 line1
D2	0/1	输入/输出	数据总线 line2
D3	0/1	输入/输出	数据总线 line3
D4	0/1	输入/输出	数据总线 line4
D5	0/1	输入/输出	数据总线 line5
D6	0/1	输入/输出	数据总线 line6
D7	0/1	输入/输出	数据总线 line7（最高位）

A	+VCC		LCD 背光电源正极
K	接地		LCD 背光电源负极

注：更多详细的资料请查看 LCD1602 datasheet，这里不再详细的解释。

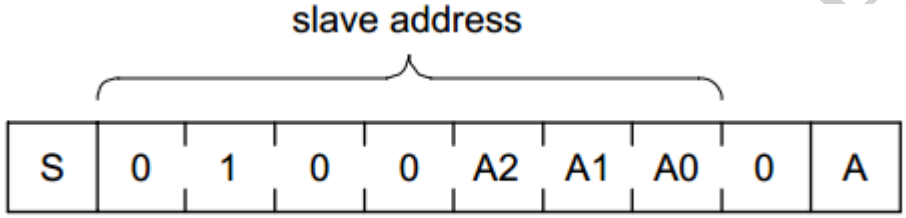
图 LCD1602：

由于 LCD1602 模块正常驱动时至少需要 4 条数据线，3 条控制线，为了减少 LCD1602 占用的 IO 口，使用以 PCF8574 为核心的 I2C 并行口扩展电路模块来驱动 LCD，可以利用 i2c 总线进行驱动。

PCF8574 转接模块：

PCF8574 模块为 i2c 并行口扩展电路，即对输入的 i2c 字节数据实现 8 位并行 io 输出（P0-P7），因此开发板可以可以通过 i2c 总线实现远程 io 扩展。

由于 PCF8574 需要与 i2c 总线进行通信，由 i2c 总线特性可以知道 i2c 总线传输数据到该模块时必须有该模块的器件地址。本模块使用的 PCF8574 芯片型号为 PCF8574T，其地址为如下图所示：



由模块的原理图可以看出 A2-A0 全部置 1，则 7 位器件地址为 0x27（0100111），RW 默认为 0，此时模块为写模式。	PCF8574 模块有 4 个引脚与开发板相连，其中 VCC（5V）、GND 与开发板的 VCC、GND 相连，而模块上的 SDA、SCL 两个引脚可以接到开发板的 i2c-0 的 SDA、SCL 两个引脚上
unsigned char *devName = "/dev/i2c-0"; devFD = pcf8574_open(devName);	打开 i2c-0 设备并设置好器件地址了，便可以成功对模块进行写操作了

当要对模块进行读操作需要把模块设置为读模式（RW 改置 1），由于这里没有用到，不详细介绍，具体可以参看 PCF8574 的 datasheet。

代码清单：

```
int main(int argc, char ** argv)
{
    int devFD, board;
    int i2cDev = 0;

    if ((board = boardInit()) < 0) {
        printf("Fail to init board\n");
        return -1;
    }

    if (argc == 2)
```

```
        i2cDev = atoi(argv[1]);
    if ((devFD = LCD1602Init(i2cDev)) == -1) {
        printf("Fail to init LCD1602\n");
        return -1;
    }
    if (LCD1602Clear(devFD) == -1) {
        printf("Fail to Clear\n");
    }
    printf("clearing LCD1602\n");
    sleep(1);
    if (LCD1602DispLines(devFD, "  B&G Char LCD", "--by FriendlyARM") == -1) {
        printf("Fail to Display String\n");
    }
    printf("displaying LCD1602\n");
    LCD1602DeInit(devFD);
    return 0;
}
```

运行测试程序：

```
$ matrix-lcd1602
```

注意：此模块并不支持热插拔，启动系统前需要确保硬件连接正确。

运行效果如下：

```
root@nanopi2:/# matrix-lcd1602
clearing LCD1602
displaying LCD1602
```

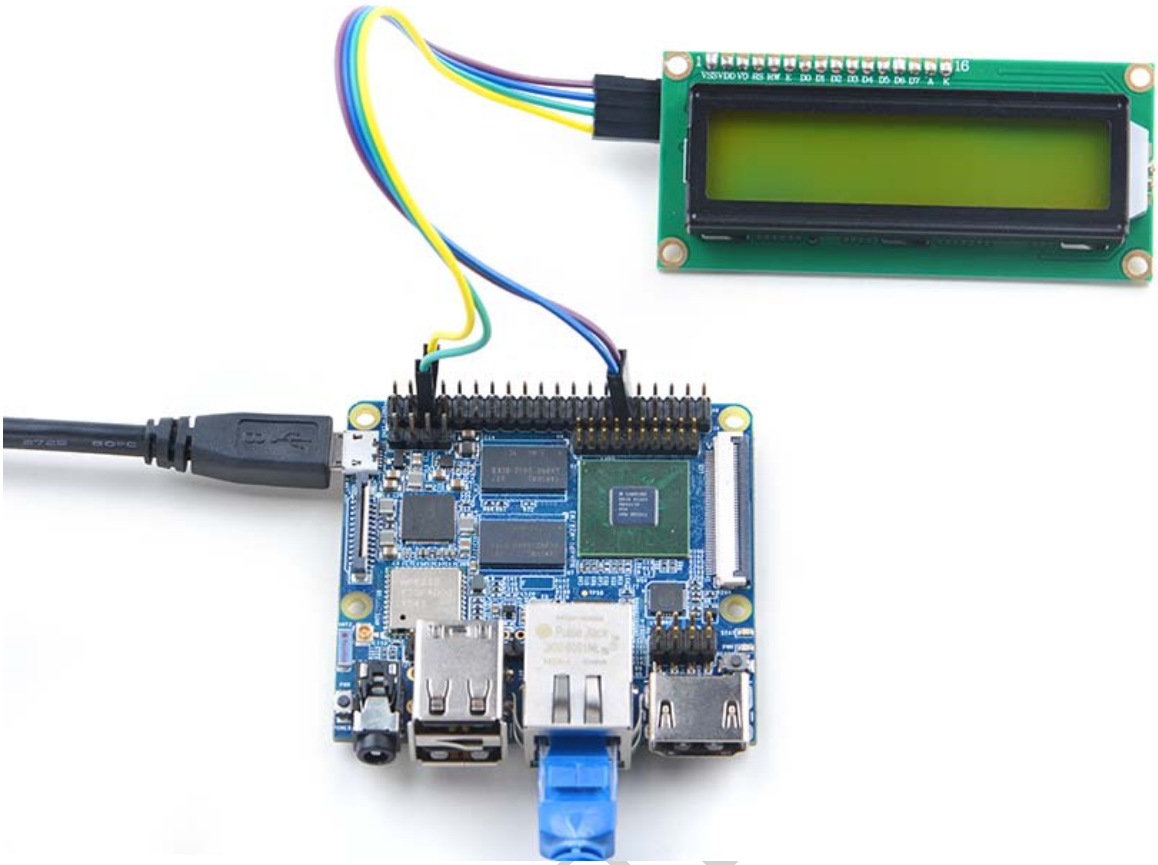
LCD 上会显示下列 2 行字符：

" B&G Char LCD"

"--by FriendlyARM"

如果 LCD 上没有显示，则需要旋转模块上的可调电阻以调节字体颜色的深浅。

参考下图连接模块：



连接说明：

Matrix-I2C_LCD1602	NanoPi M1
SDA	Pin3
SCL	Pin5
5V	Pin4
GND	Pin6

10.5 串口通讯

友善之臂已经给开发板集成串口测试模块，用户可以直接使用串口测试程序来传输自己设备的数据。具体使用步骤如下：

准备一个 1.8v 的串口转接板，四根杜邦线。

1)：用串口转接板把开发板引出的串口跟 PC 机接起来。转接板上面有标明 RX、DX、GND、5V，然后对照原理图用杜邦线把开发板的 RXD 接转接板的 RX，开发板的 TXD 接转接板的 TX,电源接电源,地接地.

3)：打开终端，输入：

```
#minicom  -s  115200  /dev/ttySAC1
```

即可让开发板跟电脑进行通讯。

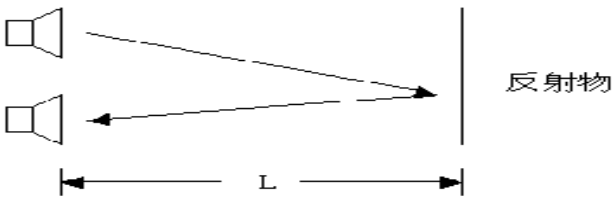
10.6 超声波模块

模块概述

超声波传感器是利用超声波的特性研制而成的传感器。超声波碰到杂质或分界面会产生显著反射形成反射成回波，碰到活动物体能产生多普勒效应。

初始化时将 trig 和 echo 端口都置低，首先向给 trig 发送至少 10 us 的高电平脉冲（模块自动向外发送 8 个 40K 的方波），然后等待，捕捉 echo 端输出上升沿，捕捉到上升沿的同时，打开定时器开始计时，再次等待捕捉 echo 的下降沿，当捕捉到下降沿，读出计时器的时间，这就是超声波在空气中运行的时间，按照 测试距离=(高电平时间*声速(340M/S))/2 就可以算出超声波到障碍物的距离。

$$L = \frac{340\ (m) \times (X2-X1)\ (S)}{2}$$



测试及使用

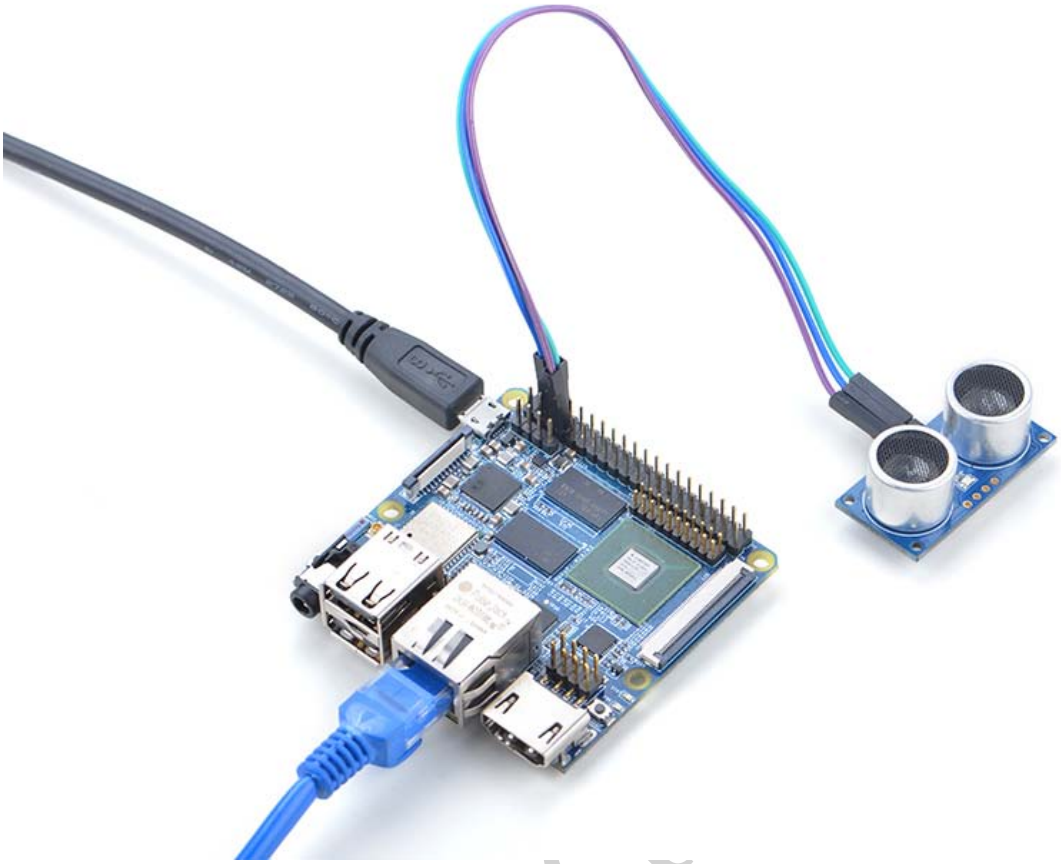
引脚介绍

Signal name	Type	Description
TRIGGER	One-way	Read access time
ECHO	One-way	transmit ultrasonic wave

代码清单：

```
int main(int argc, char ** argv)
{
    int distance = -1;
    int pin = GPIO_PIN(7);
    int board;

    if ((board = boardInit()) < 0) {
        printf("Fail to init board\n");
        return -1;
    }
    if (board == BOARD_NANOPI_T2)
        pin = GPIO_PIN(15);

    system("modprobe "DRIVER_MODULE);
    if (Hcsr04Init(pin) == -1) {
        printf("Fail to init hcsr04\n");
        goto err;
    }
    if (Hcsr04Read(&distance) != -1) {
        printf("The distance is  %3d cm\n", distance);
    } else {
        printf("Faid to get distance\n");
    }
    Hcsr04DeInit();
err:
    system("rmmod "DRIVER_MODULE);

    return 0;
}
```

参考下图连接模块：



连接说明：

Matrix-Ultrasonic_Ranger	NanoPi M1
S	Pin7
V	Pin4
G	Pin6

运行测试程序：

```
$ matrix-ultrasonic_ranger
```

注意：此模块并不支持热插拔，启动系统前需要确保硬件连接正确。
运行效果如下：

```
The distance is  24 cm
```

将超模波模块的正面对准待测距物体，测距完毕后程序会打印距离值。

10.8 RTC 串行时钟模块

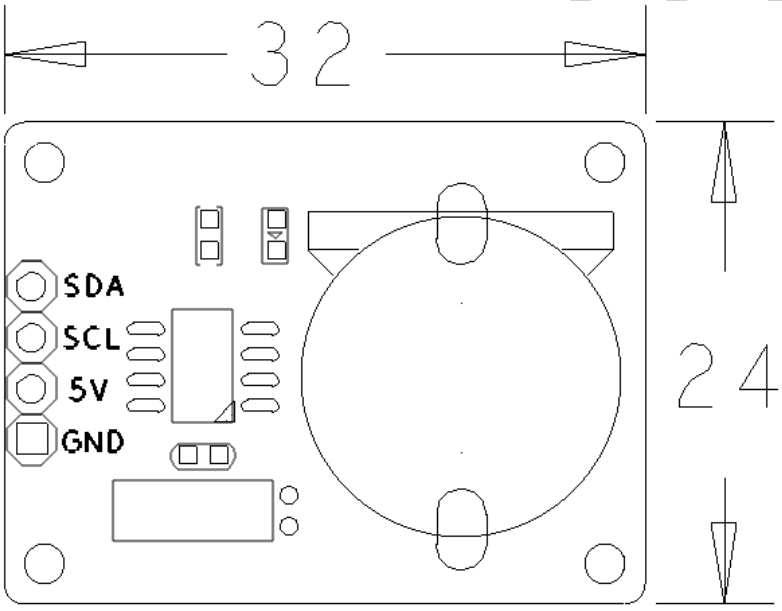
概述

DS1307 串行实时时钟是一种低功耗，完整的二进制编码的十进制（BCD）时钟/日历加 56 位字节的 NV SRAM。地址和数据通过 IIC 串行传输，双向总线。

时钟/日历提供秒、分、时、日、星期、月和年的信息。月的最后一天自动调整月的日数少于 31 天，包括闰年的修正。时钟运行 24 小时或者 12 小时格式与 AM/PM 指标。

特性

- I2C 串口接口
- 56 字节、电池支持、通用的 RAM 和无限写道
- 8-Pin DIP 和 8-Pin SO
- 操作温度在-40 度到 85 度
- PCB 尺寸（mm）：24x32



引脚说明:

名称 描述

SDA I2C SDA

SCL I2C SCL

5V　电源 5V

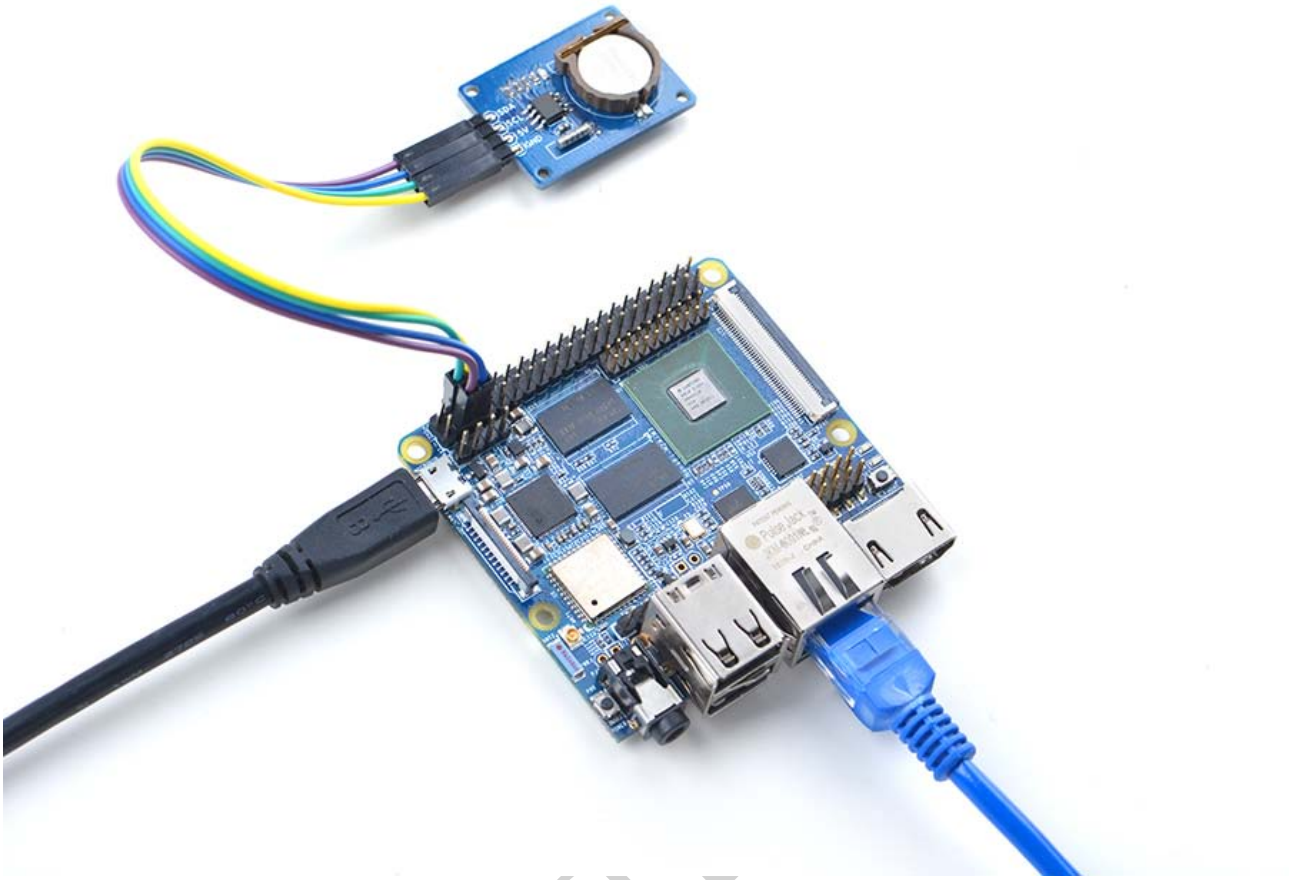
GND 地

代码清单：

```
int main(int argc, char **argv)
{
    int fd, retval, board;
    struct rtc_time rtc_tm;
    const char *rtc = default_rtc;
    const char *date_time = default_date_time;

    if ((board = boardInit()) < 0) {
        printf("Fail to init board\n");
        return -1;
    }

    switch (argc) {
    case 3:
        rtc = argv[1];
        date_time = argv[2];
        break;
    case 1:
        break;
    default:
        fprintf(stderr, "usage: rtctest [rtcdev] [year mon day hour min sec]\n");
        return 1;
    }
    system("modprobe "DRIVER_MODULE);
    fd = open(rtc, O_RDONLY);
    if (fd == -1) {
        perror(rtc);
        goto err;
    }
    fprintf(stderr, "RTC Driver Test Example.\n");

    sscanf(date_time, "%d %d %d %d %d %d",
            &rtc_tm.tm_year,
            &rtc_tm.tm_mon,
            &rtc_tm.tm_mday,
            &rtc_tm.tm_hour,
            &rtc_tm.tm_min,
```

```
            &rtc_tm.tm_sec);
    rtc_tm.tm_year -= 1900;
    rtc_tm.tm_mon -= 1;
    retval = ioctl(fd, RTC_SET_TIME, &rtc_tm);
    if (retval == -1) {
        perror("RTC_SET_TIME ioctl");
        goto err;
    }

    fprintf(stderr, "Set RTC date/time is  %d-%d-%d,  %02d:%02d:%02d.\n",
            rtc_tm.tm_mon + 1, rtc_tm.tm_mday, rtc_tm.tm_year + 1900,
            rtc_tm.tm_hour, rtc_tm.tm_min, rtc_tm.tm_sec);

    /* Read the RTC time/date */
    retval = ioctl(fd, RTC_RD_TIME, &rtc_tm);
    if (retval == -1) {
        perror("RTC_RD_TIME ioctl");
        goto err;
    }

    fprintf(stderr, "Read RTC date/time is  %d-%d-%d,  %02d:%02d:%02d.\n",
            rtc_tm.tm_mon + 1, rtc_tm.tm_mday, rtc_tm.tm_year + 1900,
            rtc_tm.tm_hour, rtc_tm.tm_min, rtc_tm.tm_sec);

    fprintf(stderr, "Test complete\n");
    close(fd);
err:
    system("rmmod "DRIVER_MODULE);
    return 0;
}
```

参考下图连接模块：



连接说明：

Matrix-RTC	NanoPi M1
SDA	Pin3
SCL	Pin5
5V	Pin4
GND	Pin6

运行测试程序：

```
$ matrix-rtc
```

注意：此模块并不支持热插拔，启动系统前需要确保硬件连接正确。

运行效果如下：

```
RTC Driver Test Example.
Set RTC date/time is 9-15-2015, 01:01:01.
Read RTC date/time is 9-15-2015, 01:01:01.
Test complete
```

该程序只是简单的读写硬件 RTC，如果想设置 Debian 的系统时间并将其保持在 Matrix-RTC 模块里，可执行以下命令，假设当前时间为"2016-11-17 17:26:01"：

```
$ modprobe rtc-ds1307
$ date -s "2016-11-17 17:26:01"
$ hwclock -w -f /dev/rtc-ds1307
```

然后修改/etc/modprobe.d/matrix-blacklist.conf，在"blacklist rtc_ds1307"前加上一个#，表示注释该行，这样下次开机就会自动加载驱动了。

重启系统，可以看到时间仍然是准确的：

```
$ hwclock -r -f /dev/rtc-ds1307
2016 年 11 月 18 日 星期五 08 时 29 分 48 秒  -0.492649 seconds
```

10.9 气压模块

一、概述

BMP180 是一款高精度、小体积、超低能的压力传感器，适用于移动电话、PDAs、GPS 导航设备和户外设备。在与仅仅 0.25 米低空噪音快速转换时，BMP180 提供卓越的性能。使用 IIC 接口可以很轻松的跟主控制器系统进行通讯。

BMP180 基于压阻式技术 EMC 稳健性、高精度以及线性长期稳定性，被设计用来直接连接带有 I2C 总线接口的移动设备小型控制器。

BMP180 的 EEPROM 值会补偿气压以及温度值。BMP180 包含压阻式传感器、模拟数字转换器、EEPROM 控制单元和 IIC 接口。

二、

I2C，3.3V

气压值 (16 to 19 bit)

温度值 (16 bit）

- 模式(超低功率、标准高,超高分辨率)可以选择通过变量过采样设置(0,1,2,3)的 C 代码。
- 计算真实的气压和温度值步长为 1pa(= 0.01 hpa = 0.01 mbar)，温度的步长为 0.1° C。
- 计算绝对高度，测量压力 p 和压力在海平面 1013.25 p0 例如 hpa,海拔米可以与国际气压公式计算:

$$altitude = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

- 计算海平面气压，使用测量压力 P 和绝对高度可以计算出海平面气压：

$$p_0 = \frac{p}{\left(1 - \frac{altitude}{44330}\right)^{5.255}}$$

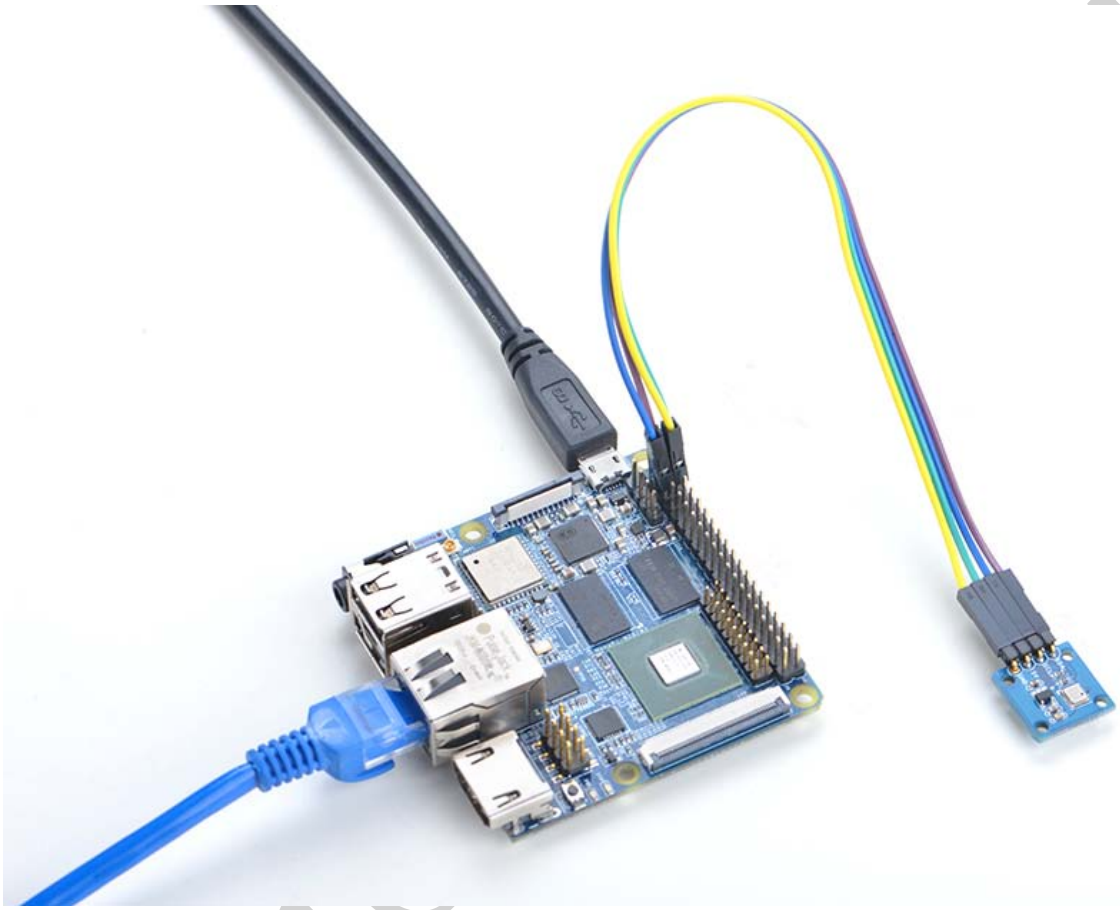
这样，不同海拔 Δaltitude = 10 米对应于 1.2hPa 海平面变化。

代码清单：

```
int main(int argc, char ** argv)
{
    int ret = -1;
    int bmpTemp=0, bmpPressure=0;
    int board;
    float altitude = 0;

    if ((board = boardInit()) < 0) {
        printf("Fail to init board\n");
        return -1;
    }

    system("modprobe "DRIVER_MODULE);
    if ((ret = bmp180Read(BMP180_TEMP, &bmpTemp)) != -1) {
        printf("The temperature is  %.1f C\n", (float)bmpTemp / 10);
    } else {
        printf("Faided to get humidity\n");
    }
    if ((ret = bmp180Read(BMP180_PRESSURE, &bmpPressure)) != -1) {
        printf("The pressure is  %.2f hPa\n", (float)bmpPressure / 100);
    } else {
        printf("Faided to get pressure\n");
    }

    altitude = 44330 * ( 1 - pow( ((float)bmpPressure / 100 / 1013.25), (1/5.255) ) );
```

```
    printf("The altitude is  %.2f m\n", altitude);
    system("rmmod "DRIVER_MODULE);
    return 0;
}
```

参考下图连接模块：



连接说明：

Matrix-Pressure_and_Temperature_Sensor	NanoPi M1
SDA	Pin3
SCL	Pin5
5V	Pin4
GND	Pin6

运行测试程序：

```
$ matrix-pressure_temp
```

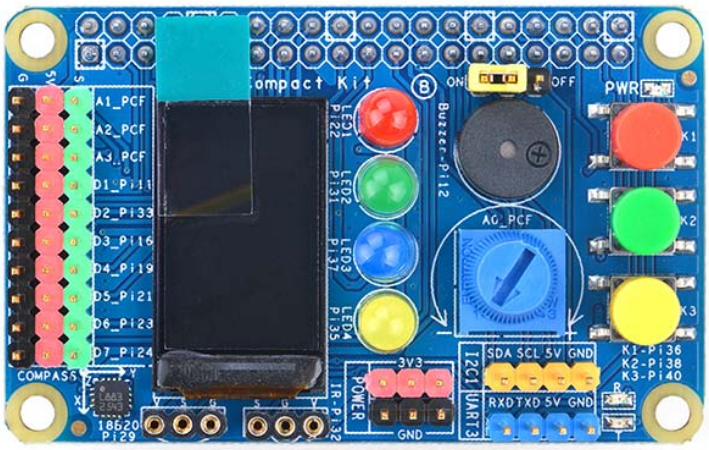
注意：此模块并不支持热插拔，启动系统前需要确保硬件连接正确。
运行效果如下：

```
The temperature is 26.6 C
The pressure is 983.91 hPa
The altitude is 247.18
```

第十一章　入门开发套件

11.1 资源

Matrix - Compact_Kit_B 是为 Nanopi M3 的接口开发的一款紧凑、简洁的多功能开发者套件，该套件由一系列常用电气元件经过精心构建而成，色彩斑斓，资源丰富，包括了按键、LED、无源蜂鸣器、ADC、指南针、温度传感器、红外接收器、TFT 等 14 种资源。该套件可以直接通过 40pin 排母和 NanoPi M3。除了丰富的板上资源以外，您还可以通过扩展的 IO 口外接模块，基于本产品开发出功能丰富的应用。



Matrix - Compact Kit B 集成的元件如下:

- 0.9 TFT LCD
- 轻触开关 3 个
- 5mm LED4 个

- 无源蜂鸣器
- ADC
- 滑动变阻器
- 40pin 排母
- 4pin 排针---I2C 接口
- 4pin 排针---UART 接口
- 3pin 双排针---3.3V 和 GND
- 18B20 温度传感器
- 红外接收器
- 3pin 排座 10 个---3 个模拟 IO,7 个数字 IO(其中两个可复用为 PWM,四个复用为 SPI 接口)
- 指南针

参考下图连接模块：

运行测试程序：

```
$ matrix-compact_kit
```

注意：此模块并不支持热插拔，启动系统前需要确保硬件连接正确。
运行效果如下：

```
LED blinking 1
LED blinking 2
Button:  1 1 1
The channel0 value is 2070
The angle is 336.3
Pwm start
Pwm stop
```

运行 Qt 程序，测试 TFT 屏：

```
cd matrix/demo/nanopi-status
./build.sh
./run.sh /dev/fb-st7735s
```

程序会显示出系统的基本信息，效果如下：

代码清单：

```
int main(int argc, char ** argv)
{
    int board;

    if ((board = boardInit()) < 0) {
        printf("Fail to init board\n");
        return -1;
    }
    testLED(board);
    readButton();
    readADC();
    readCompass();
    testPWM(board);
    // readTemp();
    // testIR();

    return 0;
```

```
}
```