NanoPi M1 初学者入门开发教程

2017-02-13

（本手册适用于 NanoPi M1 开发板）

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 M1 作为开发平台，讲解如何建立 Ubuntu 开发环境，以及如何使用 NanoPi M1 **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。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

2016-06-21	本手册第一次发布

目录

第一章　初尝 NanoPi M1 体验

NanoPi M1 是友善之臂推出的一款四核开发板，采用了全志高性能处理器 Allwinner H3，集成以太网、红外接收、视频/音频输出等接口，支持 HDMI、AVOUT 视频输出等功能，受到广大用户的喜爱，你可尽情发挥自己的想象力，使用它作为中控，开发出高性能的智能产品。你可以用它来作为网络机顶盒、智能广告机、智能家居控制器、云服务器、3D 打印机、无人飞行器等，你可以随心所欲发挥你的创意，做出有趣的、实用的智能产品。

本章节将介绍 NanoPi M1 资源特性，以及如何快速启动 NanoPi M1。

1.1 NanoPi　M1 概述

NanoPi M1（以下简称 M1）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又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 Debian、Ubuntu-MATE、Ubuntu-Core、Android 等操作系统。

NanoPi M1 采用了全志高性能处理器 Allwinner H3，集成以太网、红外接收、视频/音频输出等接口，支持 HDMI、AVOUT 视频输出等功能。

尽管体积很小，设计却紧凑美观。NanoPi M1 引出了相当丰富的接口，包括 HDMI、以太网、USB-Host、USB-OTG、DVP camera 和 AVOUT（音频+视频）等。而且集成了板载麦克风，红外接收器，并且兼容树莓派 GPIO 口，并且拥有独立的调试串口等。

NanoPi M1 资源特性：

- CPU：Allwinner H3, Quad-core Cortex-A7@1.2GHz
- GPU：Mali400MP2@600MHz，Supports OpenGL ES2.0
- DDR3 RAM：512MB/1GB
- 网络：10/100M 以太网
- 音频：3.5mm 耳机座/Via HDMI
- 麦克风：板载麦克风
- 红外：板载红外接收模块
- USB Host：Type A 型号，USB 2.0 x 3
- MicroSD Slot：x1
- MicroUSB ：支持供电和数据传输，有 OTG 功能
- 视频输出：HDMI 1.4 1080P 高清显示，CVBS
- DVP Camera 接口：24pin，0.5mm 间距竖直贴片 FPC 座
- 调试串口：4Pin，2.54mm 排针
- GPIO：40pin，2.54mm 间距，兼容 RasberryPi2 的扩展 GPIO，含 UART，SPI，I2C，PWM，IO 等管脚资源
- 按键：电源按键 x1，复位按键 x1
- PC Size：64 x 50mm
- Power Supply：DC 5V/2A
- OS/Software：u-boot，Debian，Ubuntu-MATE，Ubuntu-Core

1.2 准备工作

要开启你的 NanoPi M1 新玩具，请先准备好以下硬件

- NanoPi M1 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/2A（可使用同规格的手机充电器）
- 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

1.2.1 制作一张带运行系统的 TF 卡

1.2.1.1 下载系统固件

首先访问 http://wiki.friendlyarm.com/wiki/nanopim1/download 下载需要的固件文件(officail-ROMs 目录)和烧写工具(tools 目录)：

使用以下固件：	
nanopi-m1-debian-sd4g.img.zip	Debian系统固件
nanopi-m1-android.img.zip	Android系统固件
烧写工具：	
win32diskimager.rar	Windows平台下的Debian系统烧写工具，Linux平台下可以用dd命令烧写Debian系统
PhoenixCard_V310.rar	Windows平台下的Android系统烧写工具，注意：Android系统固件禁止在Linux平台下用dd命令烧写
HDDLLF.4.40.exe	Windows平台下用于格式化TF卡的工具

1.2.1.2 制作 Debian 系统 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 M1 的 BOOT 卡槽，上电启动（注意，这里需要 5V/2A 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。

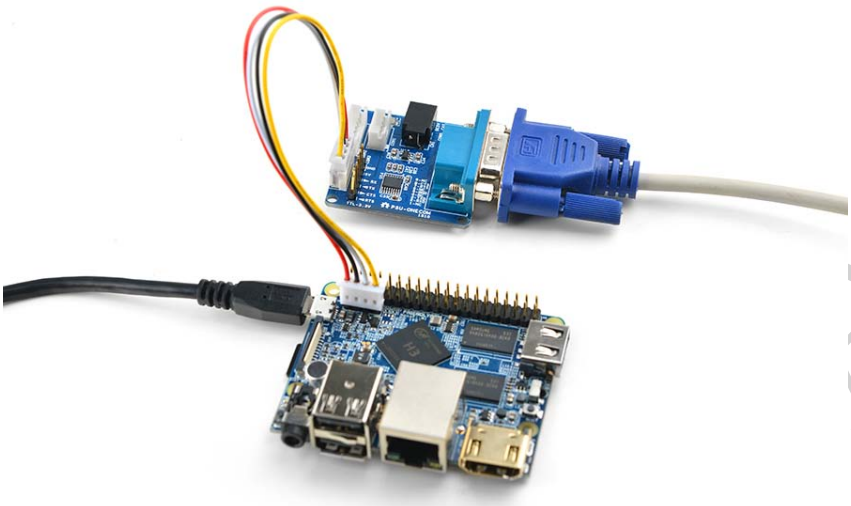
1.3 Debian 系统的使用

1.3.1 运行 Debian

将制作的好 TF 卡插入 NanoPi M1，连接 HDMI，最后连接电源(5V 2A)，NanoPi M1 会上电自动开机，看到板上的蓝色 LED 闪烁，这说明系统已经开始启动了，同时电视上也将能看到系统启动的画面。

1）要在电视上进行操作，你需要连接 USB 鼠标和键盘。
2）如果您需要进行内核开发，你最好选购一个串口配件，连接了串口，则可以通过终端对 NanoPi M1 进行操作。

- 以下是串口的接法，接上串口，即可调试。接上串口后你可以选择从串口模块的 DC 口或者从 M1 的 MicroUSB 口进行供电：



- 如果提示输入密码，Debian 的 root 和 fa 用户的默认密码都是两个字母 fa。
- 更新软件包：

```
$sudo apt-get update
```

1.3.2 扩展 TF 卡 文件系统

强烈建议做好系统运行卡之后立即进行文件系统 rootfs 分区的扩展，这将大大提升系统的性能，避免空间不足带来的各种繁琐问题。

- 在 PC 机上扩展 TF 卡的文件系统 rootfs 分区：

```
sudo umount /dev/sdx?
sudo parted /dev/sdx unit % resizepart 2 100 unit MB print
sudo resize2fs -f /dev/sdx2
```

(/dev/sdx 请替换为实际的 TF 卡设备文件名。)

- 在 M1 上扩展 TF 卡的文件系统 rootfs 分区，在 M1 上运行 Debian，然后执行命令：

```
sudo fs_resize
```

- 根据提示，输入 y 开始扩展文件系统，然后再输入 y 选择重启 M1。重启后执行下列命令查看新分区大小：

```
df -h
```

1.3.3 连接有线网络

NanoPi M1 在加电开机前如果已正确的连接网线，则系统启动时会自动获取 IP 地址，如果没有连接网线、没有 DHCP 服务或是其它网络问题，则会导致获取 IP 地址失败，同时系统启动会因此等待约 15~60 秒的时间。

- 配置 MAC 地址

板子没有提供有效的 Ethernet 的 MAC 地址，系统在连接网络时会自动生成一个随机的 MAC 地址，您可以修改 /etc/network/interfaces.d/eth0 ，配置一个固定的 MAC 地址：

```
vi /etc/network/interfaces.d/eth0
```

以下是配置文件的具体内容：

```
auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
hwaddress 76:92:d4:85:f3:0f
```

其中"hwaddress" 就是用来指定 MAC 地址，"76:92:d4:85:f3:0f"是一个随机生成的地址，为防止冲突导致网络问题，请修改为一个不同的且有效的地址。

需要注意的一点是，MAC 地址必须符合 IEEE 的规则，请不要随意指定，否则会出现无法获取 IP 地址、无法上网等问题。修改完配置文件并保存后，可重启板子或直接下列命令重启网络服务：

```
systemctl restart networking
```

1.3.4 通过 VNC 和 ssh 登录 Debian[

如果你不想连接HDMI，可以使用手机或电脑到这里下载并安装一个名为VNC Viewer的软件，用VNC连接

到NanoPi M1，默认的端口号为 1，密码为：fa123456 。
以下是在iPhone上用VNC登录NanoPi M1 的画面：

你也可以通过 ssh -l root 192.168.8.1 命令在终端上登录，默认的root用户密码是 fa 。请将 192.168.8.1 替换为实际IP地址。

1.3.5 修改 HDMI 分辨率

Debian 系统的 HDMI 分辨率是由 TF 卡 boot 分区根目录下的 script.bin 决定的，默认使用的是 720p-60Hz 的分辨率。boot 分区的 script 目录下已经有其他几种分辨率的 scrpit.bin，假设你需要 1080p-60Hz 的分辨率，只需用 script/script-1080p-60Hz.bin 替换掉根目录的 script.bin：

```
# in TF card boot partition
cp script/script-1080p-60Hz ./script.bin
```

1.3.6 HDMI 输出声音

Debian 系统默认从 3.5mm 耳机座输出声音，想从 HDMI 输出需要修改文件系统上的配置文件 /etc/asound.conf 如下：

```
pcm.!default {
	type hw
	card 1
	device 0
}

ctl.!default {
	type hw
	card 1
}
```

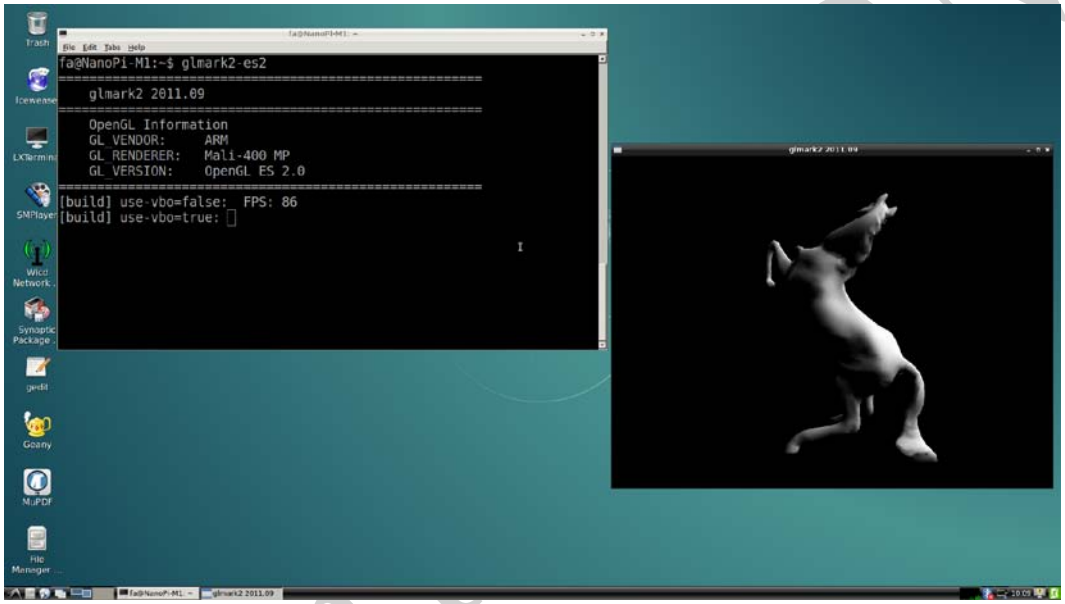
card 0 代表 3.5mm 耳机孔，card 1 代表 HDMI 音频。设置完成后需要重启系统才能生效。

1.3.7 测试 GPU

启动 Debian 系统，在 HDMI 界面下登录 Debian，打开终端并运行命令：

```
glmark2-es2
```

测试效果如下：



1.3.8 测试 VPU

访问此处下载地址下载视频文件，启动Debian系统，在HDMI界面下登录Debian，打开终端并运行命令：

```
sudo apt-get install mpv
video_play mpv ./big_buck_bunny_1080p_H264_AAC_25fps_7200K.MP4
```

经测试，可流畅硬解播放 1080p 视频。

1.3.9 连接 USB WiFi

Debian 系统已经支持市面上众多常见的 USB WiFi，想知道你的 USB WiFi 是否可用只需将其接在 M1 上即可，已测试过的 USB WiFi 型号如下：

序号	型号
1	RTL8188CUS/8188EU 802.11n WLAN Adapter
2	RT2070 Wireless Adapter
3	RT2870/RT3070 Wireless Adapter
4	RTL8192CU Wireless Adapter
5	小米WiFi mt7601

1.3.10 连接 USB 摄像头模块(FA-CAM202)

这里使用的是 200 万像素的 USB 摄像头模块 FA-CAM202。
启动 Debian 系统，在 HDMI 界面下登录 Debian，打开终端并运行命令：

```
xawtv 0
```

可以在 HDMI 界面下正常看到摄像头的预览内容。注：这里的"0"指的是接进板子的摄像头为/dev/video0 设备，请根据实际情况填写 video 索引号。

1.3.11 连接 USB 摄像头测试 OpenCV

OpenCV 的全称是 Open Source Computer Vision Library，是一个跨平台的计算机视觉库。
执行以下步骤测试 OpenCV：
连接网线，然后启动 Debian 系统，在 HDMI 界面下登录 Debian。

- 安装 opencv 库，执行命令：

```
apt-get update
apt-get install libcv-dev libopencv-dev
```

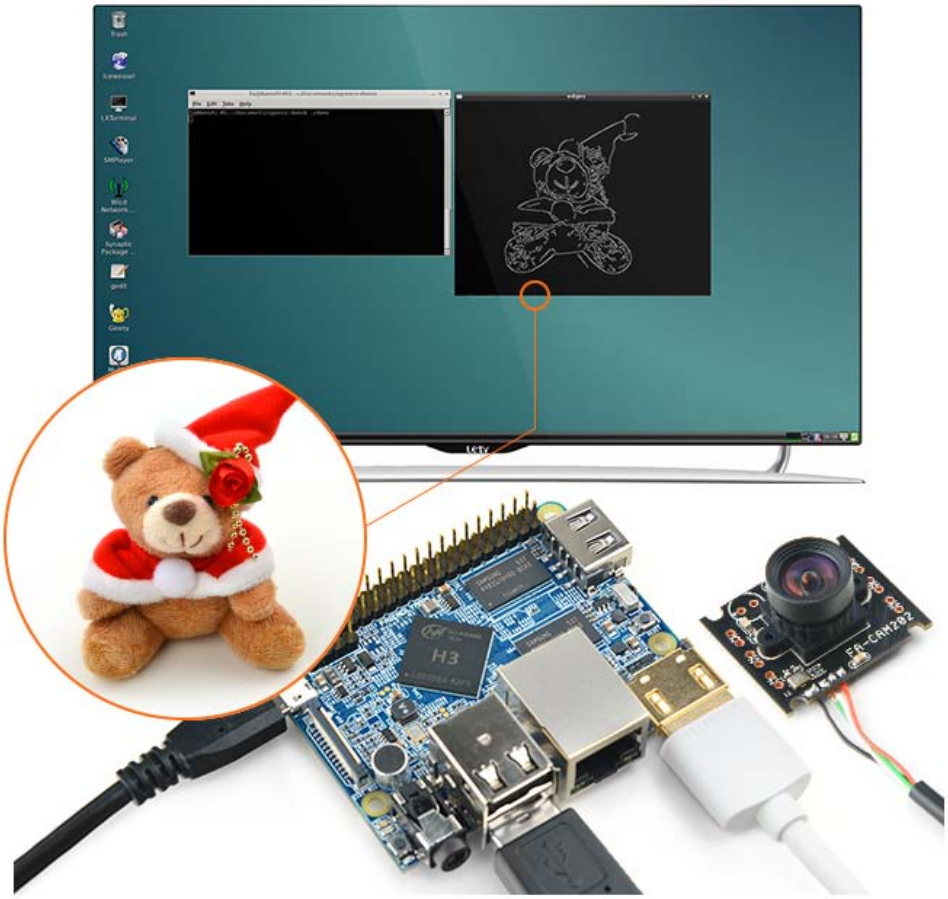
- 连接 USB 摄像头，使用 xawtv 确保摄像头工作正常：

```
xawtv 0
```

- 运行 OpenCV 官方 C++示例代码，执行下列命令编译运行：

```
cd /home/fa/Documents/opencv-demo
make
./demo
```

可以看到效果如下：



NanoPi M1图像开发
运行平台：Debian+OpenCV

1.3.12 查看 CPU 工作温度

获取 CPU 核心的当前温度值、运行频率等信息：

```
cpu_freq
```

1.4 编译 Debain 系统

访问此处下载地址的sources目录，下载源码nanopi-H3-bsp。
使用 7-Zip工具解压后得到两个目录：lichee和android，也可以从github上克隆lichee源码：

```
git clone https://github.com/friendlyarm/h3_lichee.git lichee
```

（注：lichee 是全志为其 CPU 的板级支持包所起的项目名称，里面包含了 U-boot，Linux 等源码和众多的编译脚本。）

1.4.1 编译 lichee 源码

编译全志 H3 的 BSP 源码包必须使用 64bit 的 Linux PC 系统，并安装下列软件包，下列操作均基于 Ubuntu-14.04 LTS-64bit：

```
sudo apt-get install gawk git gnupg flex bison gperf build-essential \
zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
libgl1-mesa-dev g++-multilib mingw32 tofrodos \
python-markdown libxml2-utils xsltproc zlib1g-dev:i386
```

为 Debian 系统编译 lichee 源码包，进入 lichee 目录，执行命令：

```
cd lichee
./build.sh -p sun8iw7p1 -b nanopi-h3
```

该命令会为 Debian 系统一次性编译好 U-boot、Linux 内核和模块。
注：lichee 目录里内置了交叉编译器，当使用 build.sh 脚本进行源码编译时，会自动使用该内置的编译器，所以无需手动安装编译器。

1.4.2 打包系统组件

```
./build.sh pack
```

该命令会将所有编译生成的可执行文件(包括U-boot、Linux内核)和系统配置文件拷贝到lichee/tools/pack/out/目录以便进行统一管理，并且会该目录下生成 script.bin文件。
script.bin是全志系列 CPU 的硬件板级配置文件，相关信息请查看script.bin。

下列命令可用于更新 TF 卡上的 U-boot：

```
./fuse_uboot.sh /dev/sdx
```

/dev/sdx 请替换为实际的 TF 卡设备文件名。

uImage 和内核模块均位于 linux-3.4/output 目录下，将 uImage 拷贝到 TF 卡的 boot 分区的根目录即可更新内核。

1.4.3 编译 U-boot

如果你想单独编译 U-boot，可以执行命令：

```
./build.sh -p sun8iw7p1 -b nanopi-h3 -m uboot
```

编译生成的可执行文件需打上全志系列CPU的硬件板级配置补丁后才能烧写到TF卡上运行，执行./build.sh pack能自动完成打补丁的操作。
如何手动为U-boot打补丁请查看H3_Manual_build_howto，执行下列命令更新TF卡上的U-boot：

```
./fuse_uboot.sh /dev/sdx
```

/dev/sdx 请替换为实际的 TF 卡设备文件名。

1.4.4 编译 Linux 内核

如果你想单独编译 Linux 内核，可以执行命令：

```
./build.sh -p sun8iw7p1 -b nanopi-h3 -m kernel
```

编译完成后 uImage 和内核模块均位于 linux-3.4/output 目录下，将 uImage 拷贝到 TF 卡的 boot 分区的根目录即可。

1.4.5 清理 lichee 源码

```
./build.sh -p sun8iw7p1_linux -b nanopi-h3 -m clean
```

第二章　畅游硬件开发世界

为方便用户开发前了解硬件传感器等模块的特性，这章节主要介绍支持的硬件模块和一些通用的函数接口的使用。友善之臂为用户开发了一个函数库(命名为 libfahw.so)，用于访问开发板的硬件资源，目前支持的硬件接口包括：I2C, SPI, GPIO 常用的接口。

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，Tiny4412 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。

2.1.1Minicom 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 Tiny4412 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?                                      |
+--------------------------------------------------------------+
```

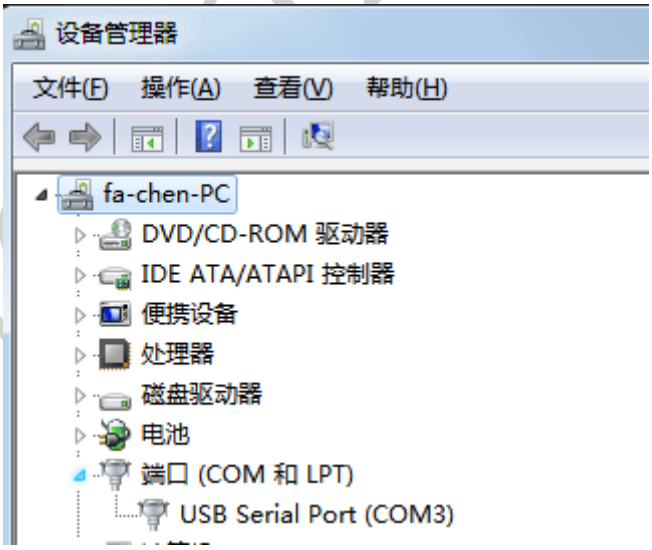
注意：如果没有使用 USB 转串口，而是直接使用串口，那么 serial device 要配置为/dev/ttyS0，如果使用 USB 转串口，则需要配置串口为 ttyUSB0。对波特率，数据位及停止位，键入 E，波特率为 115200　8N1（奇偶校验无，停止位 1），硬、软件流控制都选择 NO。确认更改成功之后，选择“save setup as dfl”之后重启 minicom 让你的配置生效，再连上开发板后，即可在 minicom 中输出正确的串口信息。
如需退出 minicom，可以使用 control-A X ，或者使用 control-A Z 进入帮助菜单。

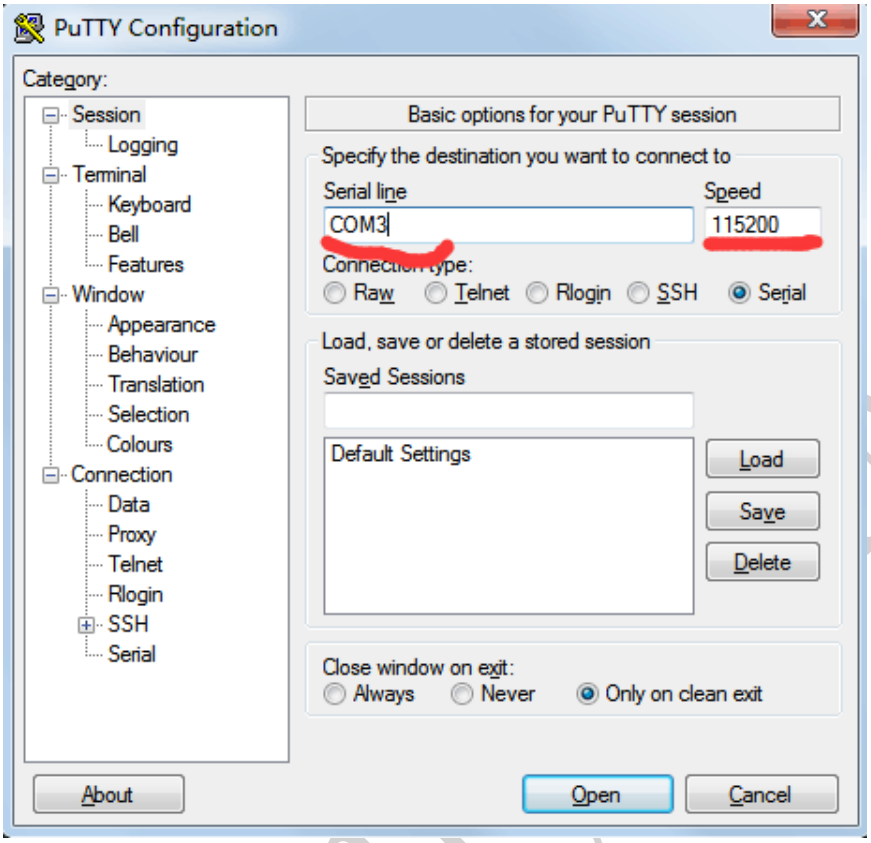
Step3：打开开发板，进入根文件系统。

2.1.2putty 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 M1 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。

2.3 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**。

2.3.1NanoPi M1 GPIO 操作说明

GPIO 管脚可以从用户空间使用的 sysfs 访问。要启用这个你必须启用以下内核选项 CONFIG_GPIO_SYSFS：

```
Device Drivers  ---> GPIO Support  ---> /sys/class/gpio/... (sysfs interface)
```

要访问一个 GPIO 引脚你首先需要将它导出：

```
echo XX > /sys/class/gpio/export
```

第 XX 是所希望的针的数目，要获得正确的 GPIO 编号，你必须从引脚名称开始计算（例如你要使用 NanoPi M1 的 GPIOG18 引脚，可通过以下公式计算： ）

```
(position of letter in alphabet - 1) * 32 + pin number
```

(全志芯片 H3 中，GPIOA 为 1，GPIOB 为 2，GPIOC 为 3，以此类推。)

例如要计算 GPIOG18，则 (7 - 1) * 32 + 18 = 256 + 18 = 274 (因为 G 在 H3 中为第七组 GPIO)。这时你已经计算出该 GPIO 位于文件系统 */sys/class/gpio/gpio242* 中，根据 With /sys/class/gpio/gpio*NUMBER*/ 路径你可配置该 GPIO 为输入/输出方向：

```
echo "out" > /sys/class/gpio/gpio*NUMBER*/direction
```

(同时，你可通过 /sys/class/gpio/gpio*NUMBER*/value 读/写该 GPIO 的值。)

当你需要通知系统取消导出引脚时，使用以下命令：

```
echo XX > /sys/class/gpio/unexport
```

(详情可参考：http://linux-sunxi.org/GPIO)

2.3.2 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　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。

3.1 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	从打开的设备或文件中读取数据。

	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。	
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

3.2 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 fd, int retries)	参数说明： 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	设置重试次数（ioctl 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)

3.3 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 order: 传 SPIEnum.MSBFIRST 或 SPIEnum.LSBFIRST 返回值说明： 成功返回 0，失败返回负数	设备 SPI 传输时是先传输低比特位还是高比特位，可选的参数有 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 spi_fd, byte[] writeData, 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 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特	写多个字节的数据到 SPI 设备

	返回值说明： 成功返回 0，失败返回负数	
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

3.4 GPIO 接口说明

接口名称	参数与返回值说明	功能说明
int initPinGPIO(int board)	参数说明： board：开发板选择 返回值说明： 成功返回 0，失败返回负数	用于识别所使用开发板的型号
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	对某个引脚输出高或低电平

	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	
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）

3.5 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

3.6 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	读取传递进文件系统的模块芯片，读取到芯片数据，则

	data：计算温度值或气压值 返回值说明： 成功返回 0，失败返回负数	对数据进行计算并且读取数据

3.7 common

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

3.8 gpio 传感器

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 返回值说明： 读取成功后关闭文件描述符	停止或关闭传感器

3.9 hmc5883

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

3.10 LED

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： 成功返回写入的数值，出错返回-1。	设置 LED 显示状态

3.11 LCD1602

接口名称	参数与返回值说明	功能说明
int mcpWriteCmd4(int devFD, unsigned char	参数说明：	写指令

command)	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	
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

3.12 OLED

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 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 返回值说明： 成功返回写入的数值，出错返回-1	配置写入 LCD 设备字符的位置
void	参数说明：	写入一整串字符串数据到

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：显示的字符串 返回值说明： 成功返回写入的数值，出错返回-1	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

3.13 PCF8591

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

3.14 pwm

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

	成功返回写入的数值，出错返回-1。	

3.15 ds18b20 温度传感器

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

3.16 PCF8574

接口名称	参数与返回值说明	功能说明
int pcf8574WriteCmd4(int devFD, unsigned char command)	参数说明： devFD：写入数据的文件描述符 command：写入的指令 返回值说明： 成功返回写入的数值，出错返回-1。	给模块写指令
int pcf8574WriteCmd8(int devFD, unsigned char command)	参数说明： devFD：写入数据的文件描述符 command：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 pcf8574WriteData4(int devFD, unsigned char data)	参数说明： devFD：写入数据的文件描述符 command：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 pcf8574WriteData8(int devFD, unsigned char data)	参数说明： devFD：写入数据的文件描述符 command：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 pcf8574Init(int i2cDev)	参数说明： i2cDev：写打开设备的 i2c 返回值说明：	初始化 i2c 设备

	成功返回写入的数值，出错返回-1。	
void pcf8574DeInit(int devFD)	参数说明： devFD:写入数据的描述符 返回值说明： 成功返回写入的数值，出错返回-1。	关闭 pcf8574 设备

第四章　玩转硬件模块

NanoPi M1 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。

4.1 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
```

4.2 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：

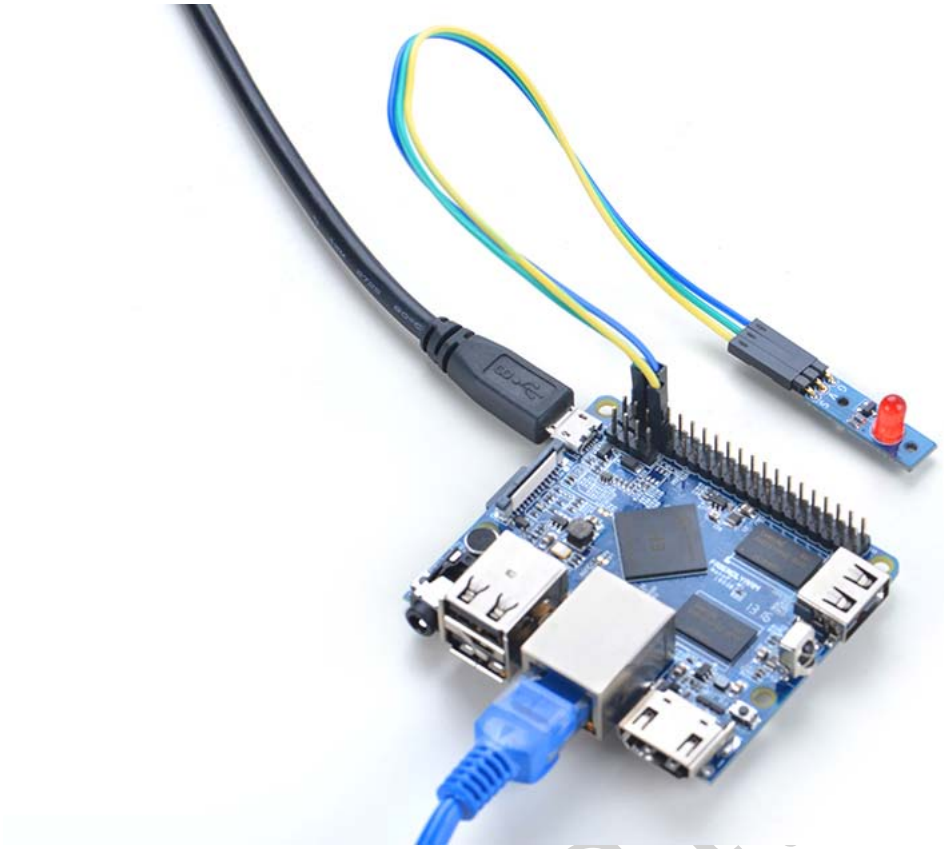
```
matrix-gpio_out
```

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

4.3 GPIO 使用示例

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。

4.4 自由控制 GPIO

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。

4.4.1 彩炫的 RGB LED

一、RGB LED 概述

RGBLED 是以三原色共同交集成像，其封装内有三个 LED，一个红色，一个绿色的，一个蓝色的。通过控制各个 LED 的亮度，你可以尽情发挥想象力，混合出几乎任何你想要的颜色。

二、管脚介绍

RGBLED 的使用原理很简单，直接给 LED 的 gpio 引脚供电即可控制三色 LED。

1	2	3	4
GPIO	Ground	GPIO	GPIO

注意：REB LED 管脚电压范围在 2.7V---5.5V ；

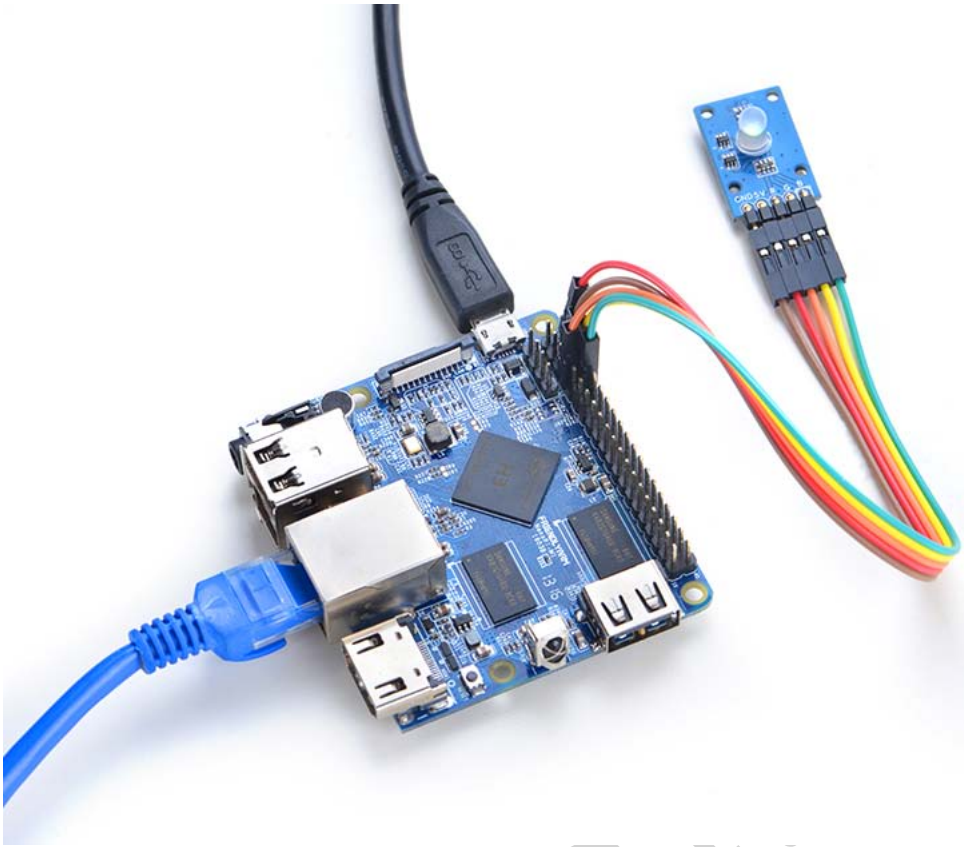
代码清单：

```
int main(int argc, char ** argv)
{
    int ret = -1;
    int val, board;

    if ((board = boardInit()) < 0) {
```

```
        printf("Fail to init board\n");
        return -1;
    }
    if (board == BOARD_NANOPI_T2) {
        ledPin1 = GPIO_PIN(15);
        ledPin2 = GPIO_PIN(16);
        ledPin3 = GPIO_PIN(17);
    }
    if ((ret = exportGPIOPin(ledPin1)) == -1) {
        printf("exportGPIOPin(%d) failed\n", ledPin1);
    }
    if ((ret = setGPIODirection(ledPin1, GPIO_OUT)) == -1) {
        printf("setGPIODirection(%d) failed\n", ledPin1);
    }
    if ((ret = exportGPIOPin(ledPin2)) == -1) {
        printf("exportGPIOPin(%d) failed\n", ledPin2);
    }
    if ((ret = setGPIODirection(ledPin2, GPIO_OUT)) == -1) {
        printf("setGPIODirection(%d) failed\n", ledPin2);
    }
    if ((ret = exportGPIOPin(ledPin3)) == -1) {
        printf("exportGPIOPin(%d) failed\n", ledPin3);
    }
    if ((ret = setGPIODirection(ledPin3, GPIO_OUT)) == -1) {
        printf("setGPIODirection(%d) failed\n", ledPin3);
    }
    signal(SIGINT, intHandler);
    for (val = 0; val < 8; val++) {
        printf("Set RGB LED:  %x\n", val);
        setRGBLED(val);
        usleep(1000 * 1000);
    }
    unexportGPIOPin(ledPin1);
    unexportGPIOPin(ledPin2);
    unexportGPIOPin(ledPin3);
    return 0;
}
```

连接 NanoPi　M1

参考下图连接模块：



连接说明：

Matrix-RGBLED	NanoPi M1
R	Pin7
G	Pin8
B	Pin10
V	Pin4
G	Pin6

运行测试程序：

```
matrix-rgb_led
```

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：

```
Set RGB LED: 0
Set RGB LED: 1
Set RGB LED: 2
```

```
Set RGB LED: 3
Set RGB LED: 4
Set RGB LED: 5
Set RGB LED: 6
Set RGB LED: 7
```

可以看到 RGBLED 在不停地变换颜色。

4.4.2 无源蜂鸣器

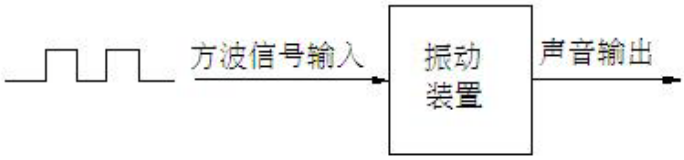
一：无源蜂鸣器概述

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，钳片不能振动发音。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;
}
```

4.4.3 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：

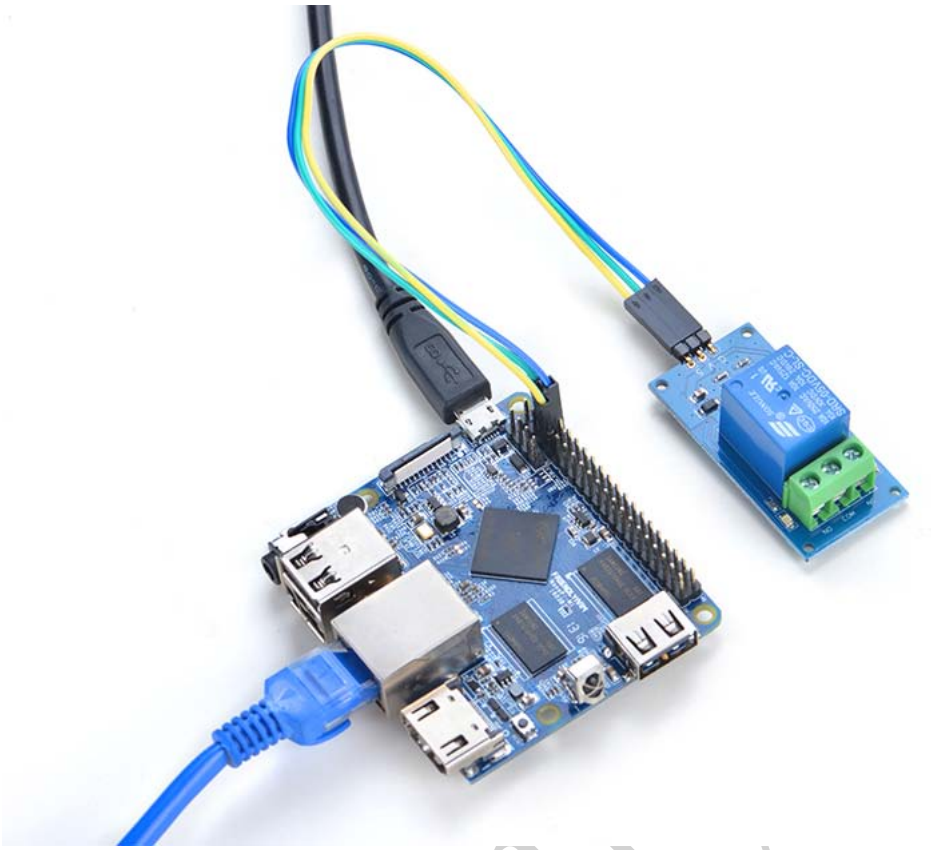
代码清单：

```
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;
}
```

连接 NanoPi M1

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
```

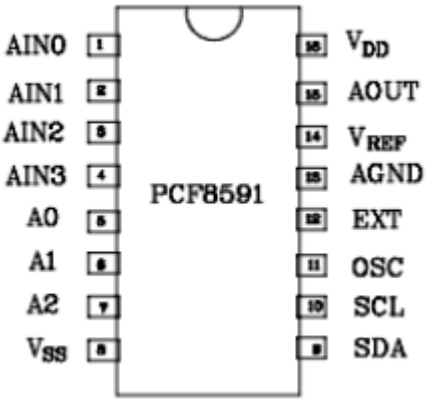
4.5　ADC 转换 PCF8591

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。

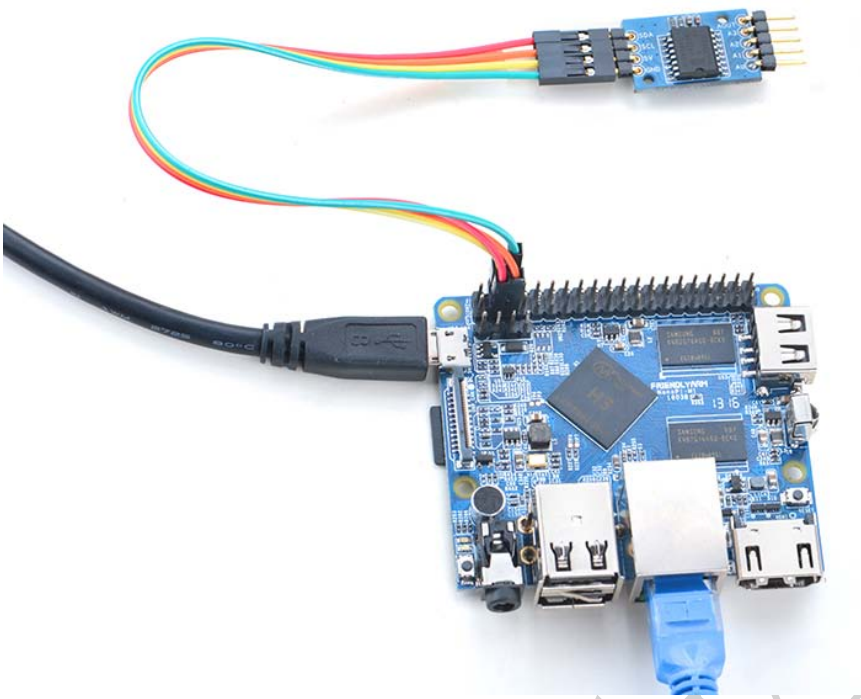
在 pcf8591.c 文件中调用 pcf8591_AD_init(devFD); 语句把模块初始化为四个单端模拟电压输入，读取 channel0 输出的 AD 转换。

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 NanoPi M1
参考下图连接模块：



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

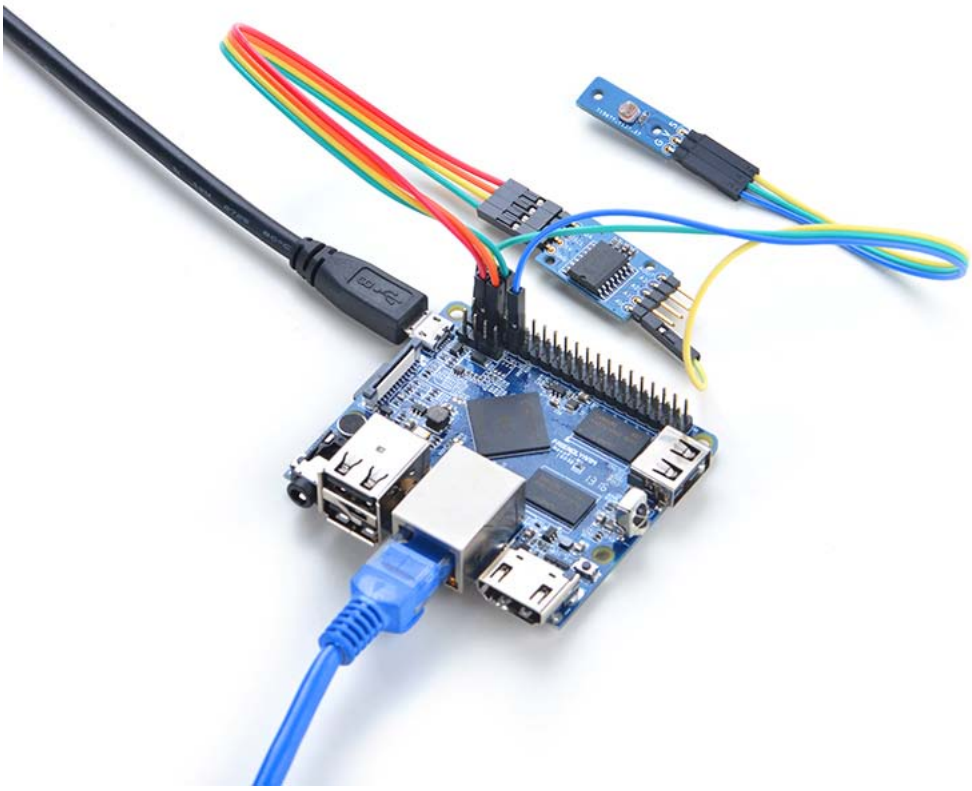
4.5.1 光敏电阻

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。

连接 NanoPi M1

参考下图连接模块：



4.5.2 土壤湿度传感器

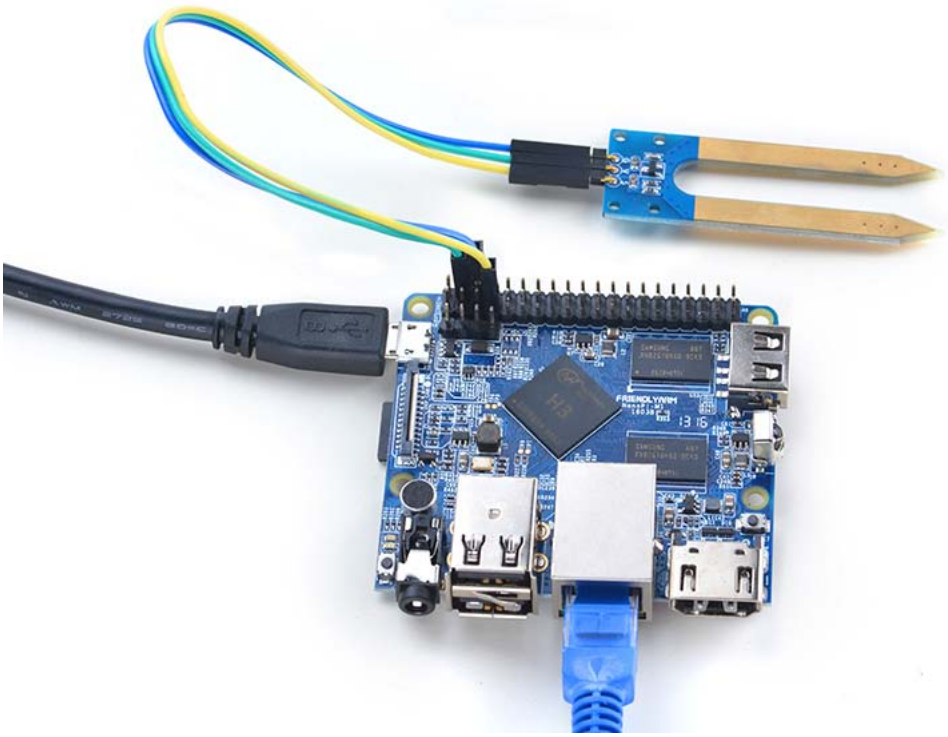
土壤湿度传感器	工作形式	使用说明	读取值
它利用电磁脉冲原理、根据电磁波在介质中传播频率来测量土壤的表观介电常数(ε)，从而得到土壤容积含水量(θv)。	工作电压 3.3V-5V 模块双输出模式。土壤湿度传感器水分传感器可用于检测土壤的水分，当土壤缺水时，模块输出一个高电平，反之输出低电平。	传感器模块使用的是 adc 模拟输入输出，这里直接把传感器模块接到 PCF8591 转换模块上即可检查到土壤湿度值。	four_signle_adc 文件用于使 pcf8591 工作于 AD 转换模式下 channel0-3 分别转换 AIN0-3 路的单端模拟输入，并在终端每秒采集一次数据

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：



4.5.3 双轴按键摇杆

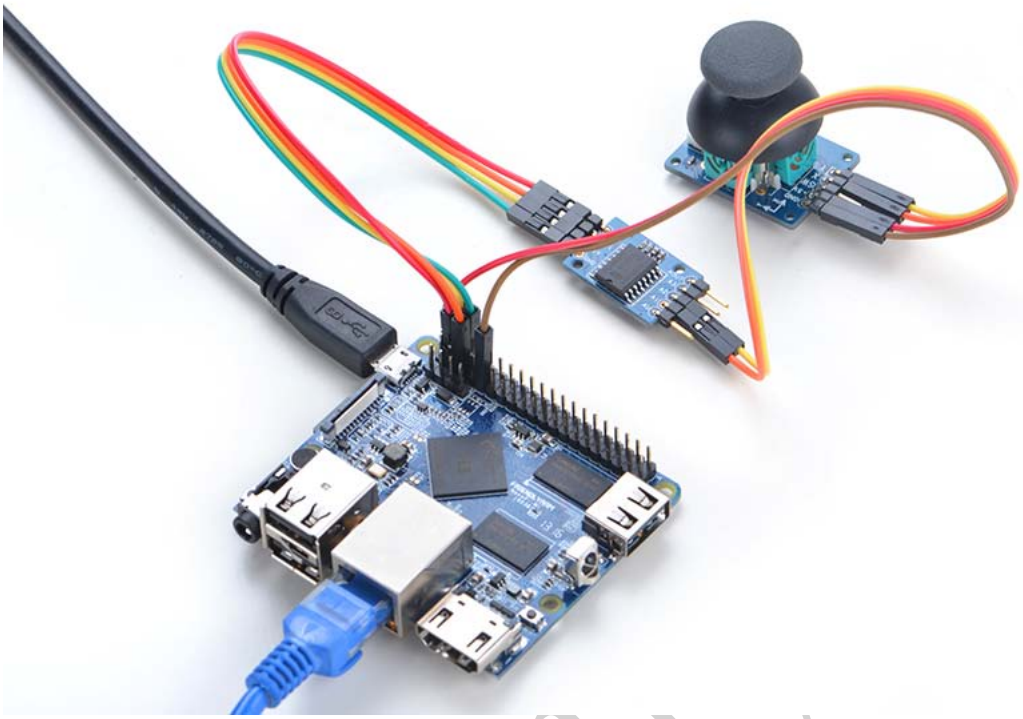
双轴按键摇杆原理	工作形式	使用说明	读取值
模块特设二路模拟输出和一路数字输出接口，输出值分别对应（X，Y）双轴偏移量，其类型为模拟量；按键表示用户是否在 Z 轴上按下，其类型为数字开关量。模块集成电源指示灯，可显示工作状态；坐标标识符清晰简明、准确定位。	为了让客户更加方便地配合扩展板等标准接口，在设计上把 X,Y,Z 轴的电路都单独引出，以控制输入这个操纵杆模块的 x、y、z 的 PS2 摇杆戏摇杆模块 Joystick 值以及在特定的值下实现某种功能	传感器模块使用的是 adc 模拟输入输出，上面已经介绍过如何使用 ADC 转换模块，这里直接把传感器模块接到 PCF8591 转换模块上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。

连接 NanoPi M1
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

4.6 LCD1602 液晶显示器

一、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。

二、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。

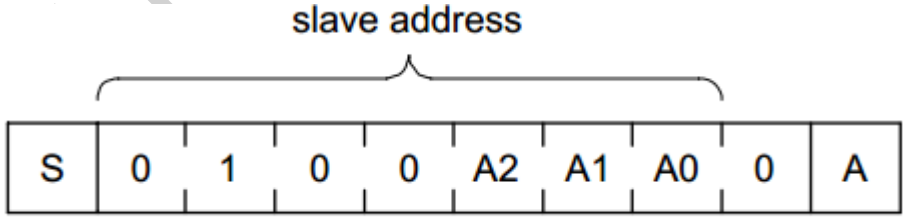
图 LCD1602：

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：



三、代码清单

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
```

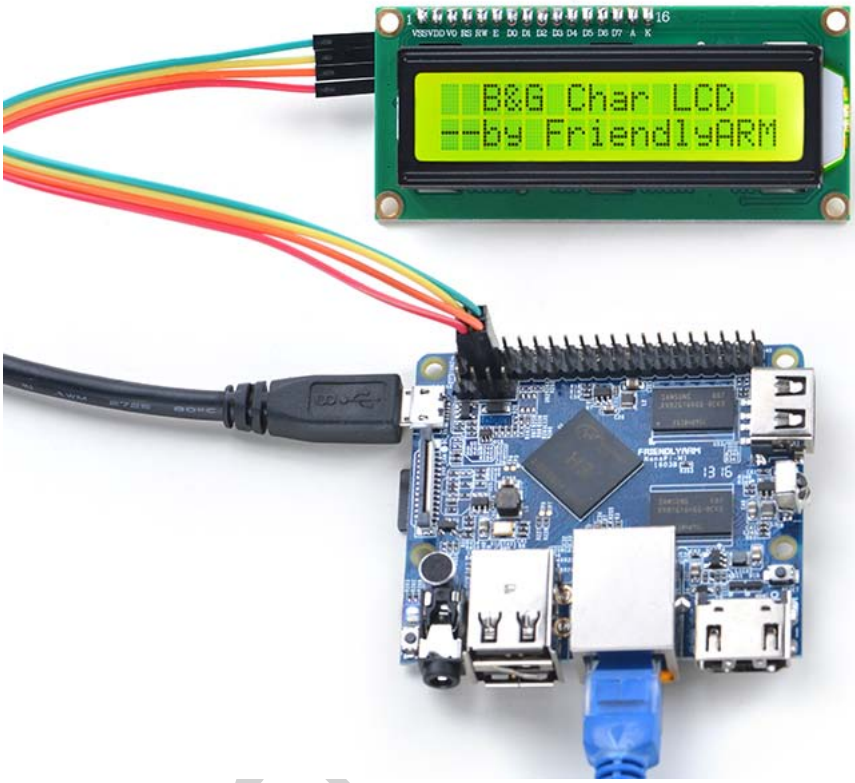
LCD 上会显示下列 2 行字符：

" 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。

连接 NanoPi M1

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

4.7 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。

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.
打开终端，输入：

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

即可让开发板跟电脑进行通讯。

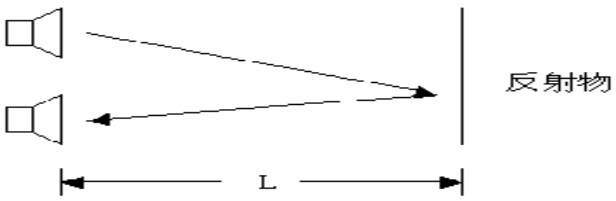
4.8 超声波模块

一、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。

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



二、引脚介绍

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

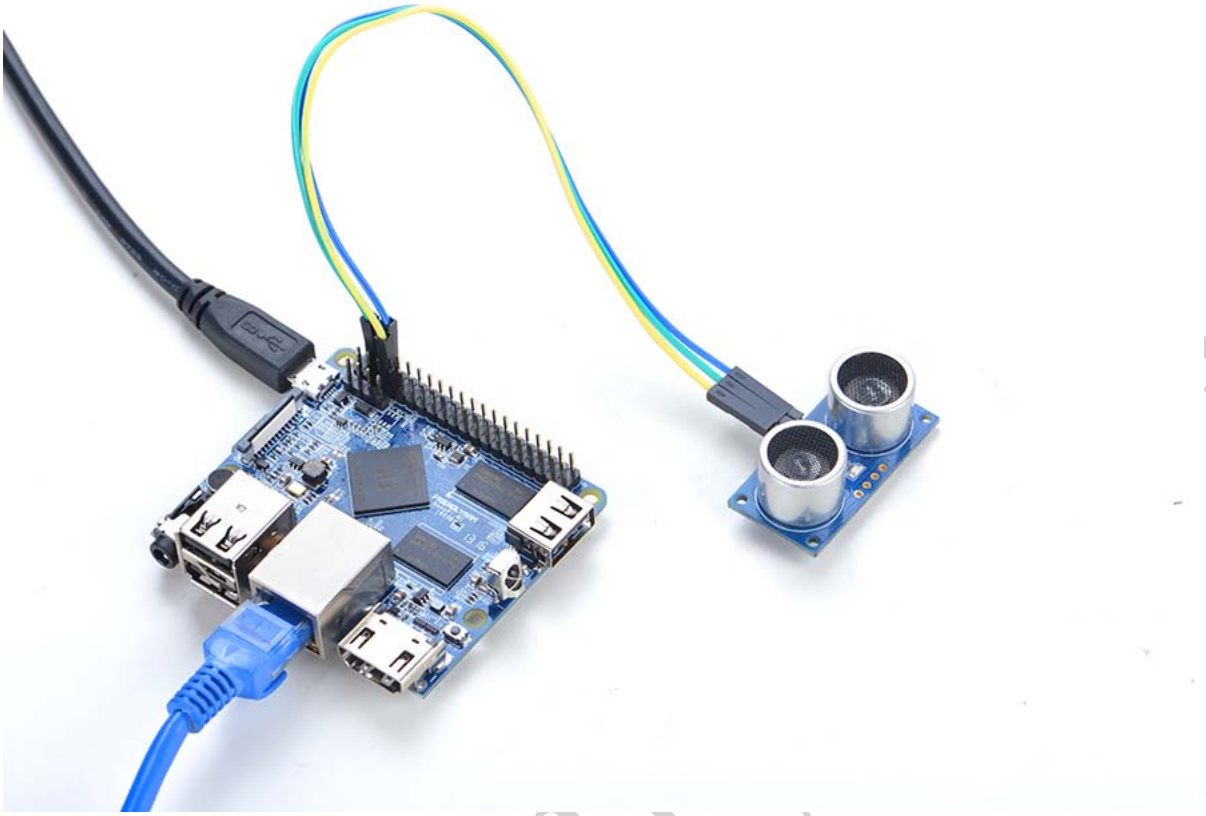
三、代码清单

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;
}
```

连接 NanoPi M1

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。

4.9 对射光计数模块

一、概述

具有信号输出指示，单路信号输出，输出有效信号为低电平，灵敏度不可调。可用于工件计数、电机测速。有遮挡物时输出高电平(LED 灯灭)，无遮挡物时输出低电平(LED 灯亮)灵敏度不可调。可用于工件计数、电机测速。电路板输出开关量！

二、引脚概述

OUT	直接接 GPIO 引脚

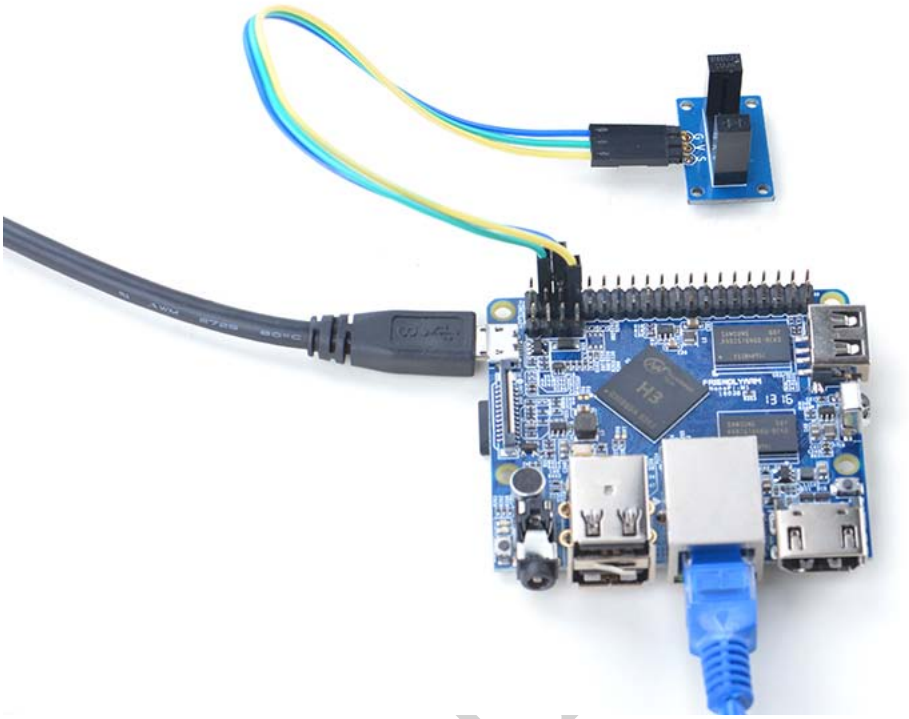
代码清单：

```
int main(int argc, char ** argv)
{
    int i, board;
    int retSize = -1;
    char value[ARRAY_SIZE(dev)];

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

    if (argc == 2)
        dev[0].pin = atoi(argv[1]);
    system("modprobe "DRIVER_MODULE);
    signal(SIGINT, intHandler);
    if (board == BOARD_NANOPI_T2)
        dev[0].pin = GPIO_PIN(15);
    printf("Use GPIO_PIN(%d)\n", dev[0].pin);
    if ((devFD =sensorInit(dev, ARRAY_SIZE(dev))) == -1) {
        printf("Fail to init sensor\n");
        goto err;
    }
    printf("Waiting event...\n");
    if ((retSize = sensorRead(devFD, value, ARRAY_SIZE(dev))) == -1) {
        printf("Fail to read sensors\n");
    }
    if (retSize > 0) {
        i = 0;
        for (i=0; i<retSize; i++) {
            printf("Device[%d] value is  %d\n", i, value[i]);
        }
    }
    sensorDeinit(devFD);
err:
```

```
    system("rmmod "DRIVER_MODULE);
    return 0;
}
```

连接 NanoPi M1

参考下图连接模块：



连接说明：

Matrix-IR_Counter	NanoPi M1
S	Pin7
V	Pin4
G	Pin6

运行测试程序：

```
matrix-gpio_int
```

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：

```
Waiting event...
Device[0] value is 1
```

当有物体阻挡在模块中间时会检测到事件。

4.10 RTC 串行时钟模块

一、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

二、引脚说明:

名称 描述

SDA I2C SDA

SCL I2C SCL

5V 电源 5V

GND 地

三、代码清单

代码清单：

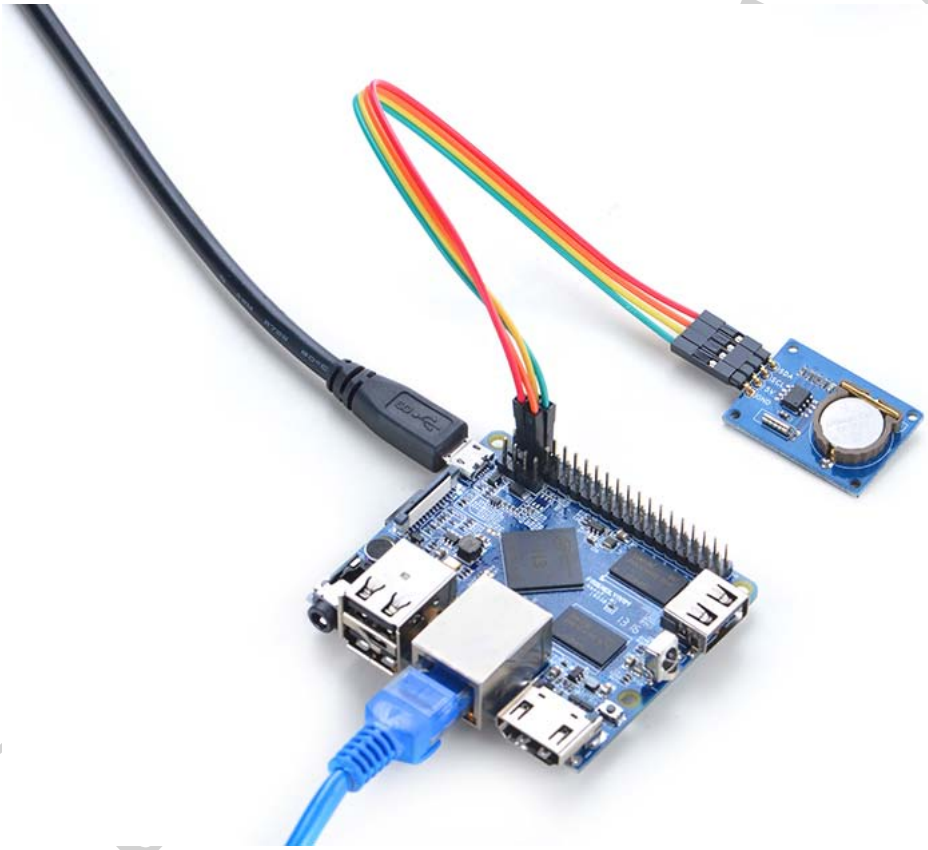
```
int main(int argc, char **argv)
{
    int fd, retval, board;
    struct rtc_time rtc_tm;
    const char *rtc = default_rtc;
    const char *date_time = default_date_time;

    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;
}
```

连接 NanoPi　M1

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

4.11 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、引脚名称

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。

三、代码清单

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;
}
```

连接 NanoPi M1
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
```

第五章　迷你扩展板初学者入门开发套件

Matrix - Compact_Kit_B 是为 Nanopi 2 Fire 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 NanoPi2 Fire 连接。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。

本章将介绍迷你扩展板初学者入门开发的基础教程。

5.1 资源

Matrix - Compact Kit B 集成的元件如下:

1. 0.9 TFT LCD
2. 轻触开关 3 个
3. 5mm LED4 个
4. 无源蜂鸣器
5. ADC
6. 滑动变阻器
7. 40pin 排母
8. 4pin 排针---I2C 接口
9. 4pin 排针---UART 接口
10. 3pin 双排针---3.3V 和 GND
11. 18B20 温度传感器
12. 红外接收器
13. 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)
14. 指南针

连接 NanoPi M1

参考下图连接模块：

5.2 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。

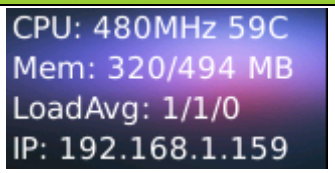
运行效果如下：

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

5.3 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;
}
```