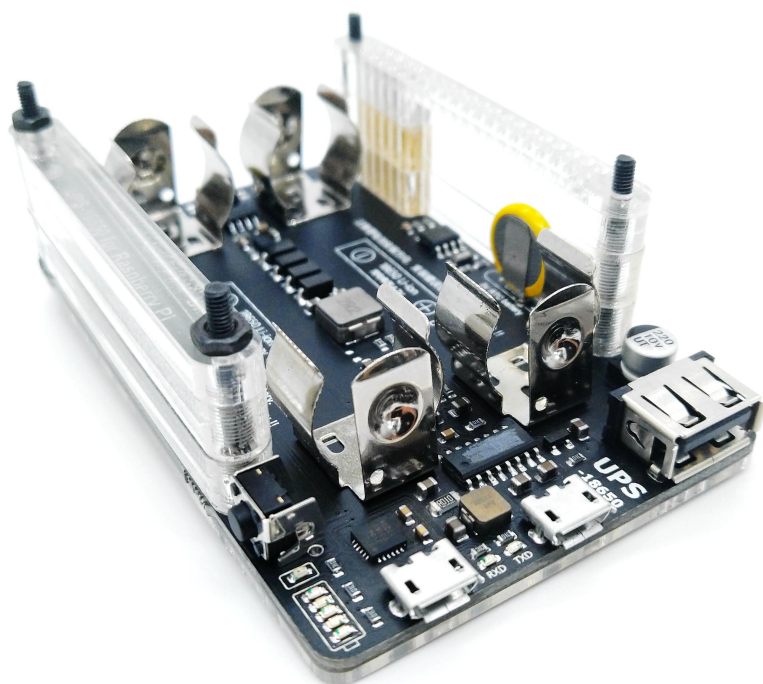


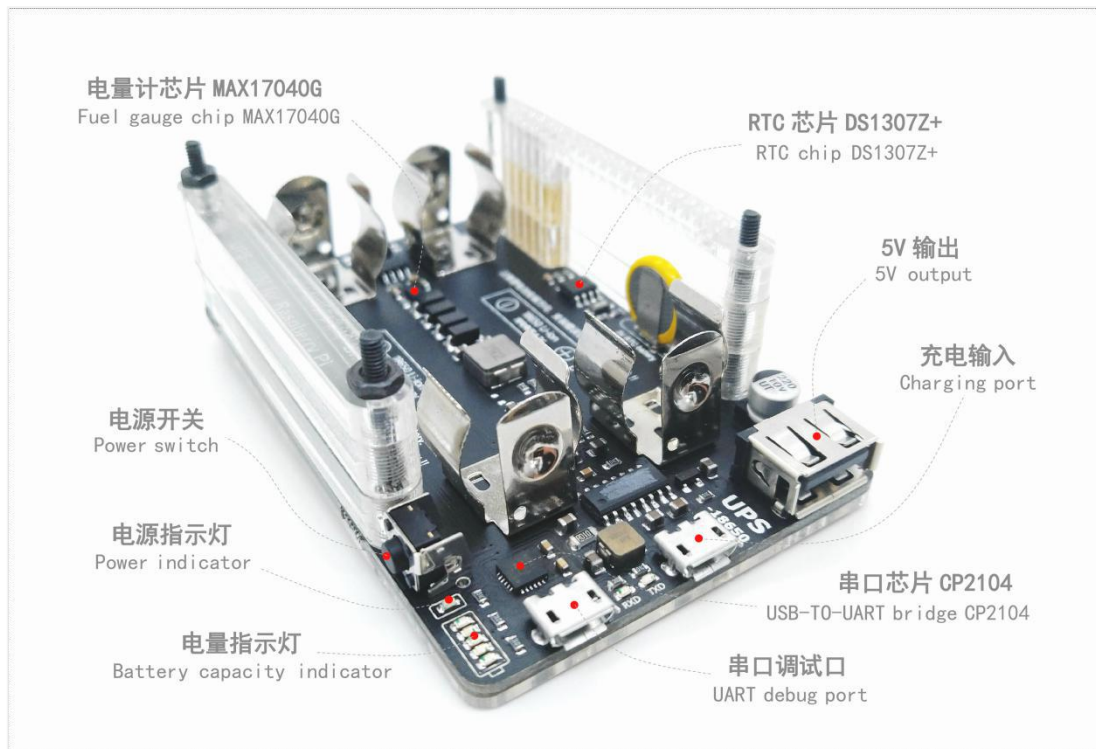
UPS-18650 使用说明

for Raspberry pi 1/2/3 V1.1 -- by XiaoJ



1. 简介

UPS-18650 是一个专门为树莓派（以下简称 pi）所设计的 UPS 电源，采用两颗标准的平头 18650 锂电池（不带保护板的电芯）进行供电，支持外部电源插入检测，支持边充边放，既插上外部电源时，pi 由外部电源供电，拔掉外部电源时，pi 无缝转由锂电池供电。UPS-18650 通过 10 根大电流顶针与 pi 主板连接，所以使用时 pi 无需用 micro usb 数据线与 UPS-18650 连接。另外 UPS-18650 还集成了专业电量计芯片 MAX17040G、RTC 实时时钟芯片 DS1307Z+、USB-to-UART 串口芯片 CP2104、电量指示灯、以及额外的 5V 输出口供外部其他设备使用。



参数：

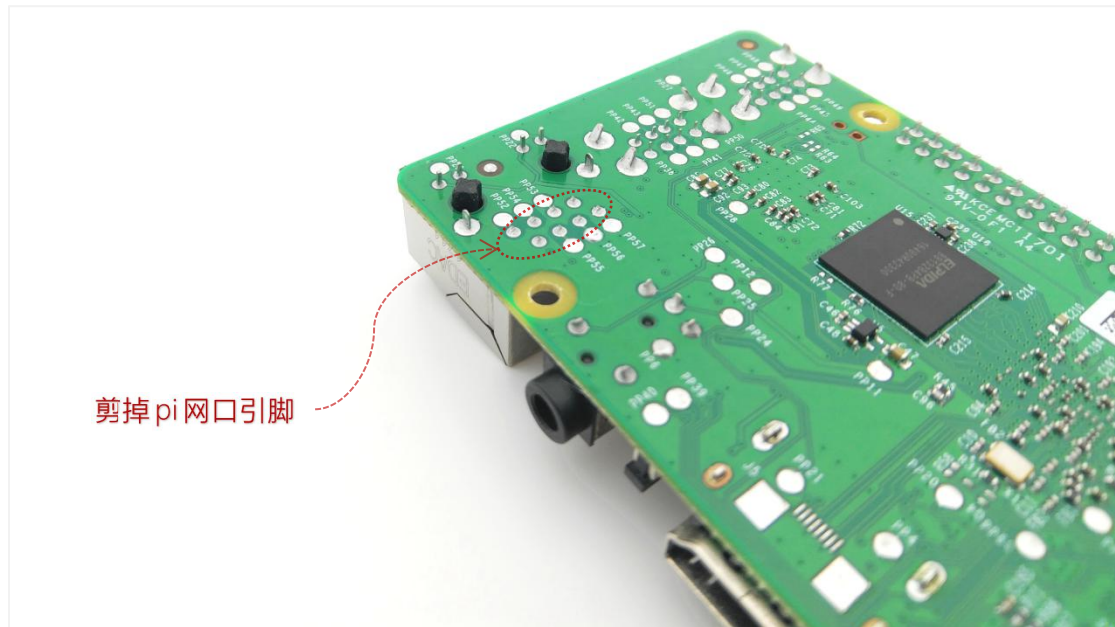
充电电流：**max 1A@5V**

输出电流：**max 3A@5V** (在使用两节 3300mAh 18650 锂电池或者外部电源适配器功率大于 5V3A 的情况下)

电量测量：输出电池电量百分比，误差 $\pm 1\%$ ，电压测量误差在 $\pm 3\text{mV}$ ，测量误差因电池而异

2. 安装步骤

a. 剪掉 pi 背面网口的引脚，否则网口引脚会顶到锂电池。



b. 请按照板子上的电池标识装好两节 18650 锂电池（备注：两节锂电池必须是同个型号且内阻及容量相差不多，不可混用两节不同参数的锂电池，更不能使用带保护板的锂电池，有可能会损坏充电芯片）。



备注：电池正负极识别



c. 对准 4 个螺丝，把 pi 主板装上去，然后拧上 4 个固定铜柱。



d. 电池拆卸更换，请用镊子按照板上箭头指示的方向插入后往上撬出电池

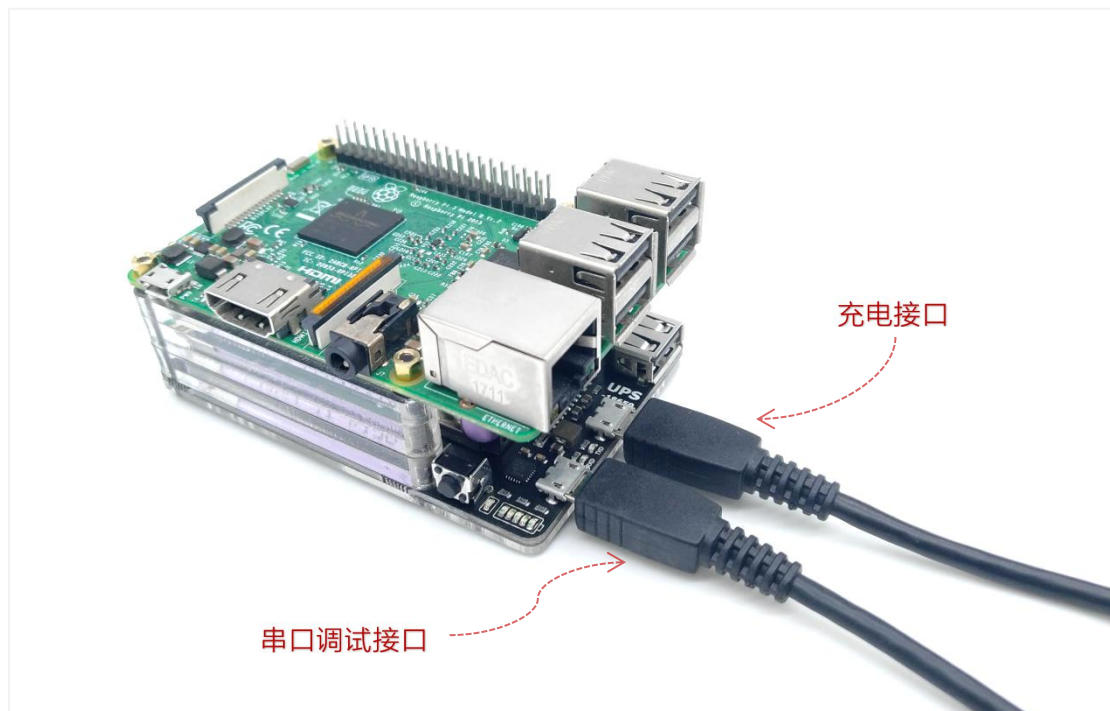


3. 功能操作

充电操作：

请用功率在 5V2.5A 及以上电源适配器给 UPS-18650 充电。因为当锂电池电量较低时，外部电源适配器不仅要给 pi 供电，还需要提供部分电流供锂电池充电，当锂电池充满电时，5 个绿电量指示灯会全亮，表示电池已经充满电。5 个电量指示灯表示的电池容量依次为 20%、40%、60%、80%、100%。另外 UPS-18650 带有电源适配器插入检测功能，插入电源时 pi 的 **IO7**(BCM 编号 IO4)会检测到高电平，拔出时为低电平，使能该功能需要短接 UPS 背面的两个焊盘，详细见下图。

充电时请将外部电源适配器插到 UPS 充电接口上，不要插到串口调试接口，这个接口不能对 UPS 进行供电，仅供串口调试使用。





5V 输出操作:

在有接外部电源适配器的情况下，长按电源开关 3 秒以上，红色电源指示灯亮，表示输出 5V 电压。再一次长按 3 秒以上，红色电源指示灯灭，表示关闭 5V 电压输出。

在没有接外部电源适配器的情况下，长按电源开关 3 秒以上，红色电源指示灯亮，表示输出 5V 电压。再一次长按 3 秒以上，红色电源指示灯灭，表示关闭 5V 电压输出，也可快速短按电源开关两次关闭 5V 电压输出。短按电源开关一次，电量指示灯显示当前电池电量，稍后会熄灭。



串口功能操作:

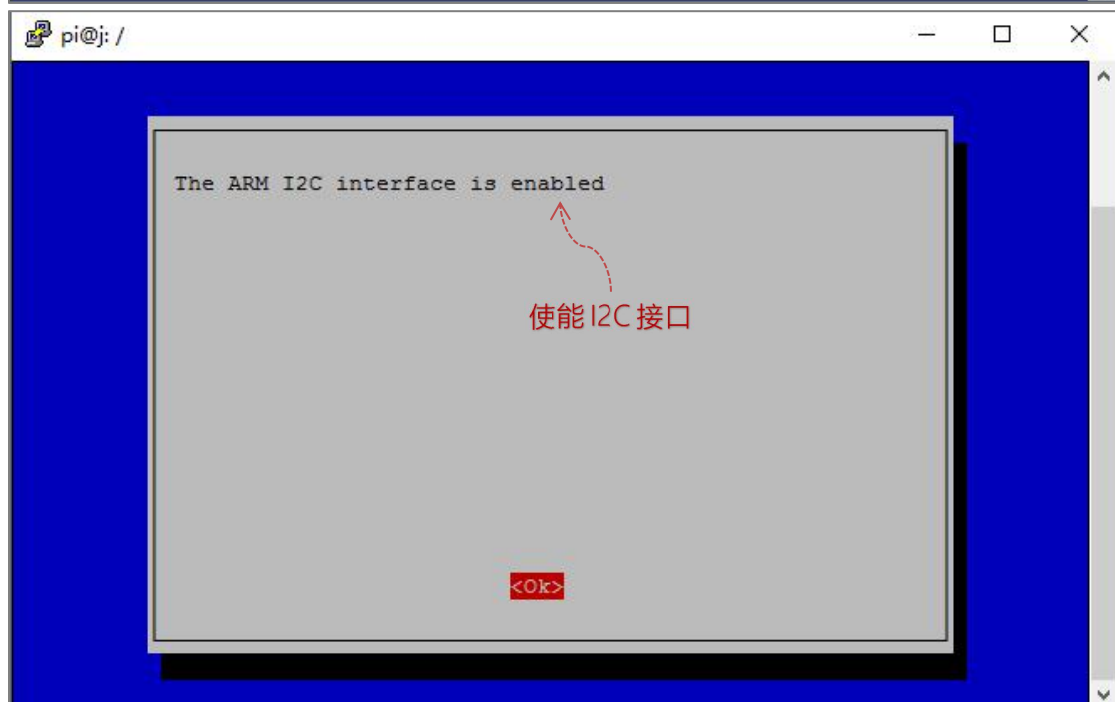
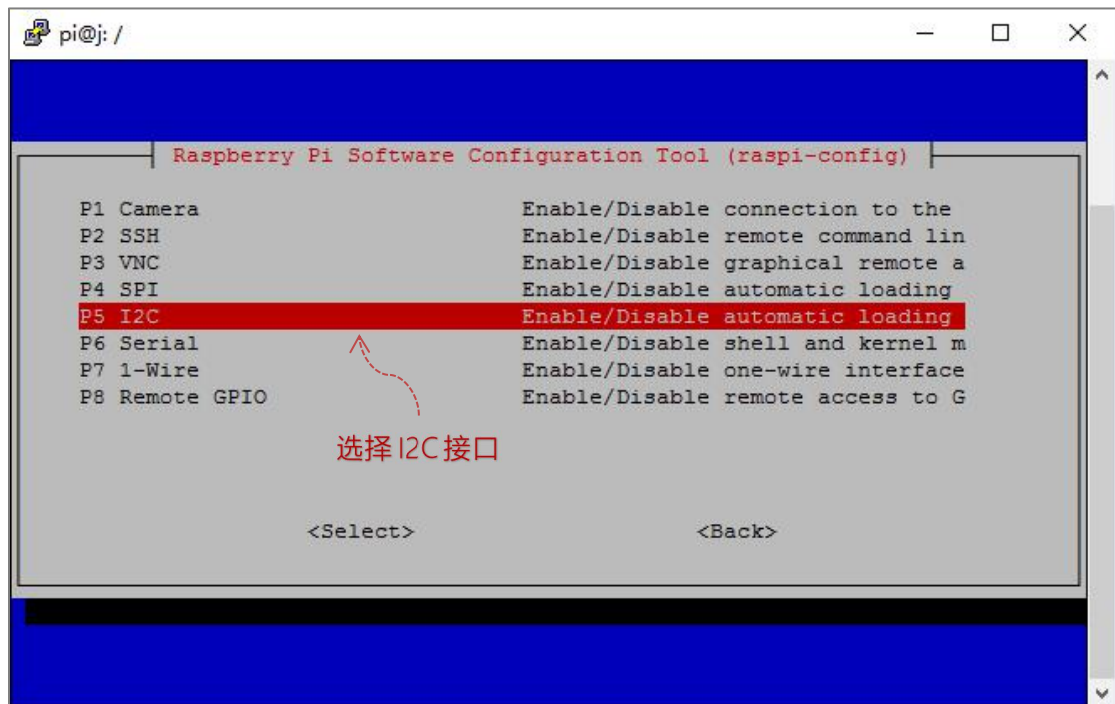
UPS-18650 采用一颗 CP2104 串口芯片来实现串口调试功能，使用前需要在 PC 端安装相应的驱动软件，然后把拨码开关全部拨到“ON”上，pi 插上 usb 线（插在串口调试接口上）与 PC 连接，PC 上打开相应的串口软件如 PuTTY，设置好相关参数，即可进行串口通信调试。



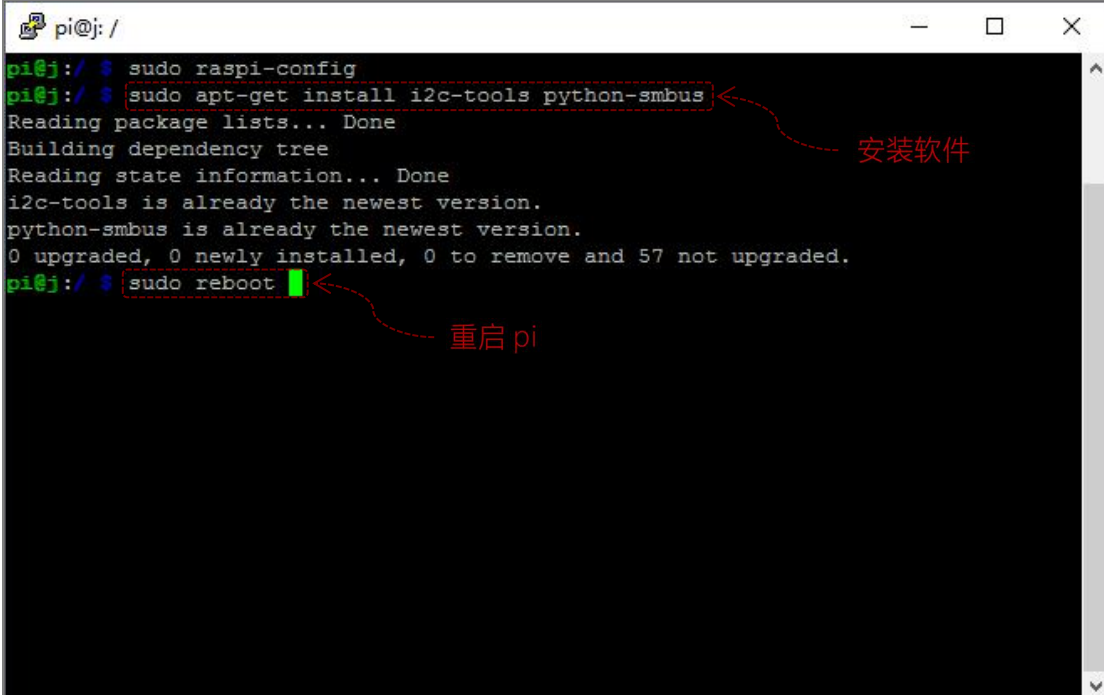
RTC 功能操作:

- a. 打开 pi 配置工具 raspi-config, 使能 I2C 接口





b. 安装 i2c-tools 和 python-smbus, 安装完成后重启一下 pi



```
pi@j: /  
pi@j:~$ sudo raspi-config  
pi@j:~$ sudo apt-get install i2c-tools python-smbus  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
i2c-tools is already the newest version.  
python-smbus is already the newest version.  
0 upgraded, 0 newly installed, 0 to remove and 57 not upgraded.  
pi@j:~$ sudo reboot
```

安装软件

重启 pi

c. 运行 `sudo i2cdetect -l` 查看当前 pi 是采用哪个 i2c 总线。



```
pi@j: /  
pi@j:~$ sudo i2cdetect -l  
i2c-1    i2c          bcm2835 I2C adapter      I2C adapter  
pi@j:~$
```

运行 i2cdetect -l 查看 i2c 总线

确定系统 i2c 总线为 1

d. 运行 `sudo i2cdetect -y 1` 查看当前 pi 的 i2c 总线上挂载的设备。



The terminal window shows the execution of `sudo i2cdetect -y 1`. The output is a grid representing the I2C bus. Annotations with red dashed arrows point to specific values:

- An arrow points to the command `sudo i2cdetect -y 1` with the text "运行 i2cdetect -l 查看 i2c 总线".
- An arrow points to the value `36` in the output grid with the text "地址 0x36 为 MAX17040G".
- An arrow points to the value `68` in the output grid with the text "地址 0x68 为 DS1307Z+".

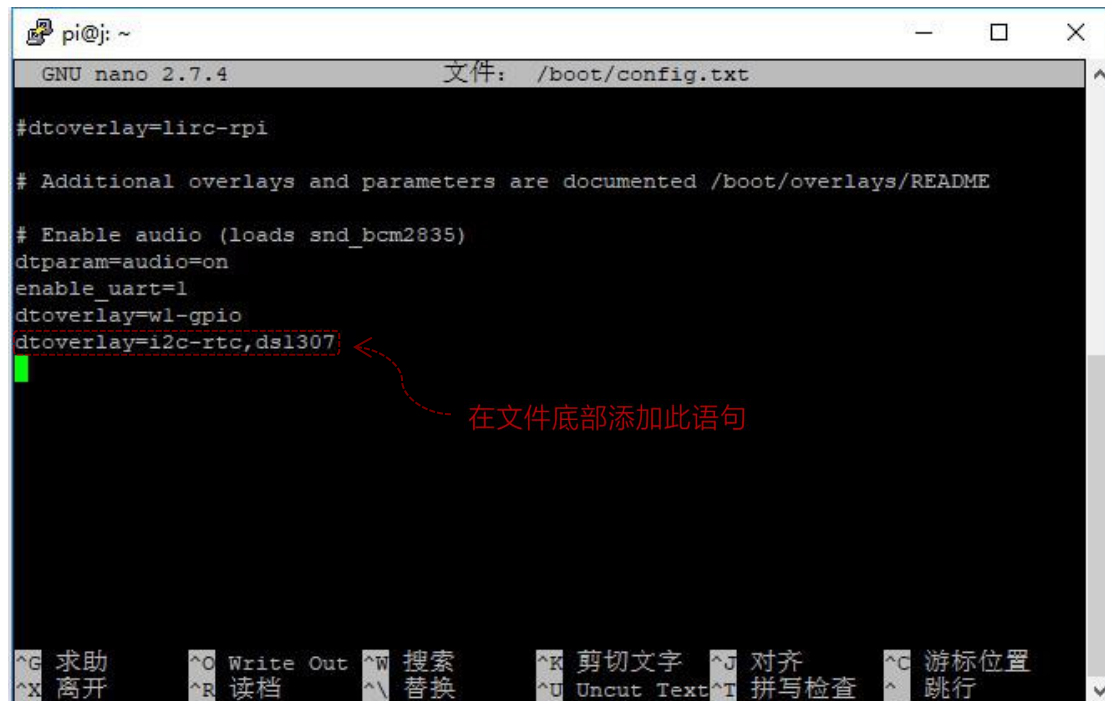
```
pi@j: /  
pi@j:~$ sudo i2cdetect -l  
i2c-1      i2c          bcm2835 I2C adapter      I2C adapter  
pi@j:~$ sudo i2cdetect -y 1  
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
30: -- -- -- -- -- 36 -- -- -- -- -- -- -- -- --  
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
60: -- -- -- -- -- 68 -- -- -- -- -- -- -- -- --  
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
pi@j:~$
```

e. 编辑配置文件，在文件底部添加以下语句 `dtoverlay=i2c-rtc,ds1307` 并保存，然后重启 pi。



The terminal window shows the command `sudo nano /boot/config.txt` being entered. An annotation with a red dashed arrow points to the command with the text "用 nano 编辑器打开配置文件".

```
pi@j: ~  
pi@j:~$ sudo nano /boot/config.txt
```



```
pi@j: ~
GNU nano 2.7.4 文件: /boot/config.txt

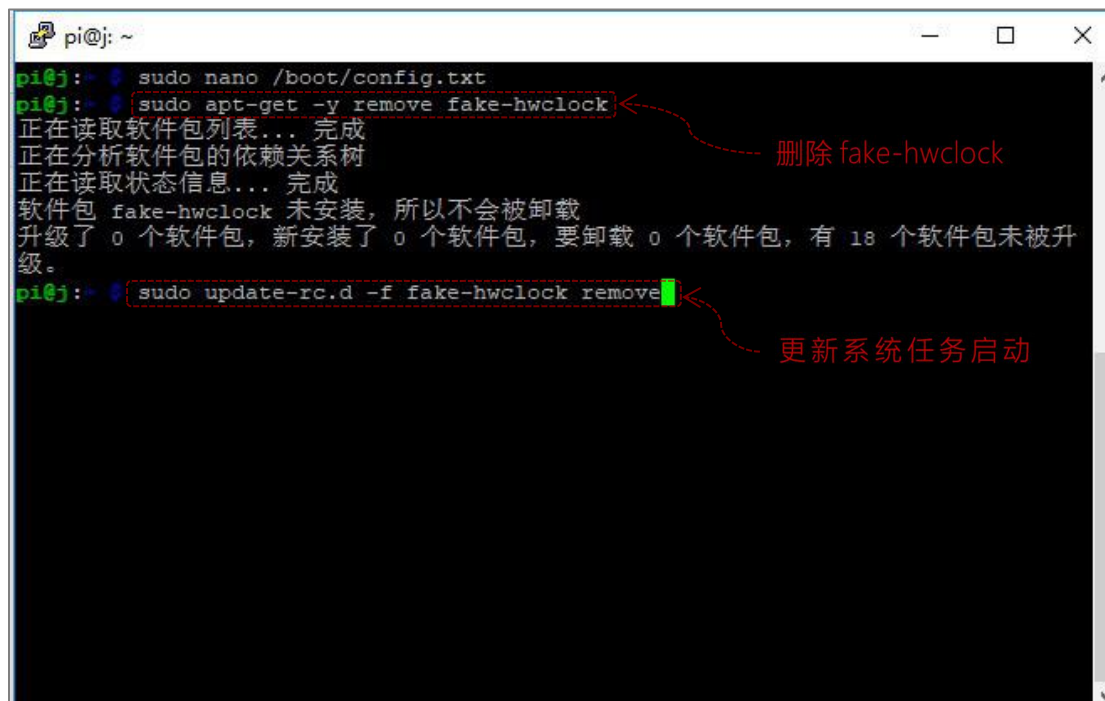
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtparam=audio=on
enable_uart=1
dtoverlay=wl-gpio
dtoverlay=i2c-rtc,dsl307


^G 求助      ^O Write Out ^W 搜索      ^K 剪切文字  ^J 对齐      ^C 光标位置
^X 离开      ^R 读档      ^\ 替换      ^U Uncut Text ^T 拼写检查  ^_ 跳行
```

f. 删除 fake-hwclock 以及更新系统任务启动项



```
pi@j: ~
pi@j:~$ sudo nano /boot/config.txt
pi@j:~$ sudo apt-get -y remove fake-hwclock
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
软件包 fake-hwclock 未安装，所以不会被卸载
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 18 个软件包未被升级。
pi@j:~$ sudo update-rc.d -f fake-hwclock remove
```

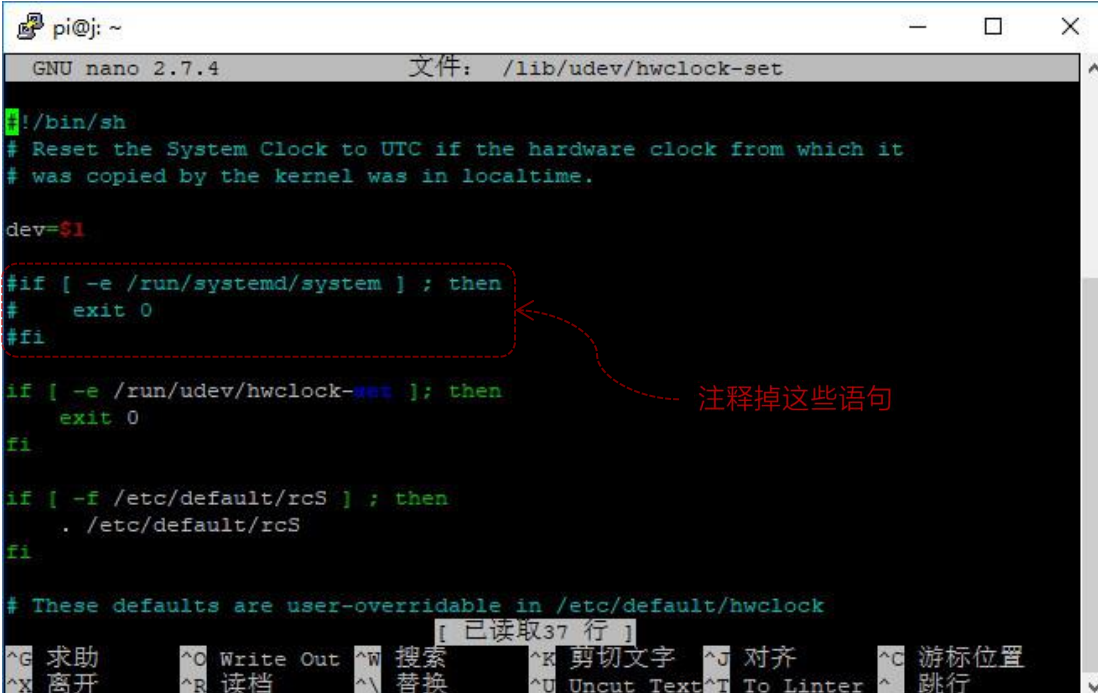

g. 编辑 hwclock 设置文件，注释掉相关语句，并保存。



A terminal window with the prompt `pi@j: ~`. The command `sudo nano /lib/udev/hwclock-set` is entered and highlighted in green. A red dashed arrow points from the text "用 nano 编辑器编辑 hwclock 设置文件" to the command.

```
pi@j:~$ sudo nano /lib/udev/hwclock-set
```

用 nano 编辑器编辑 hwclock 设置文件



A terminal window showing the nano editor editing the file `/lib/udev/hwclock-set`. The file content is displayed in green text. A red dashed arrow points from the text "注释掉这些语句" to a block of code that should be commented out. The bottom of the window shows a status bar with various keyboard shortcuts and a line count of 37.

```
GNU nano 2.7.4 文件: /lib/udev/hwclock-set

#!/bin/sh
# Reset the System Clock to UTC if the hardware clock from which it
# was copied by the kernel was in localtime.

dev=/dev

if [ -e /run/systemd/system ] ; then
#   exit 0
#fi

if [ -e /run/udev/hwclock-set ]; then
    exit 0
fi

if [ -f /etc/default/rcS ] ; then
    . /etc/default/rcS
fi

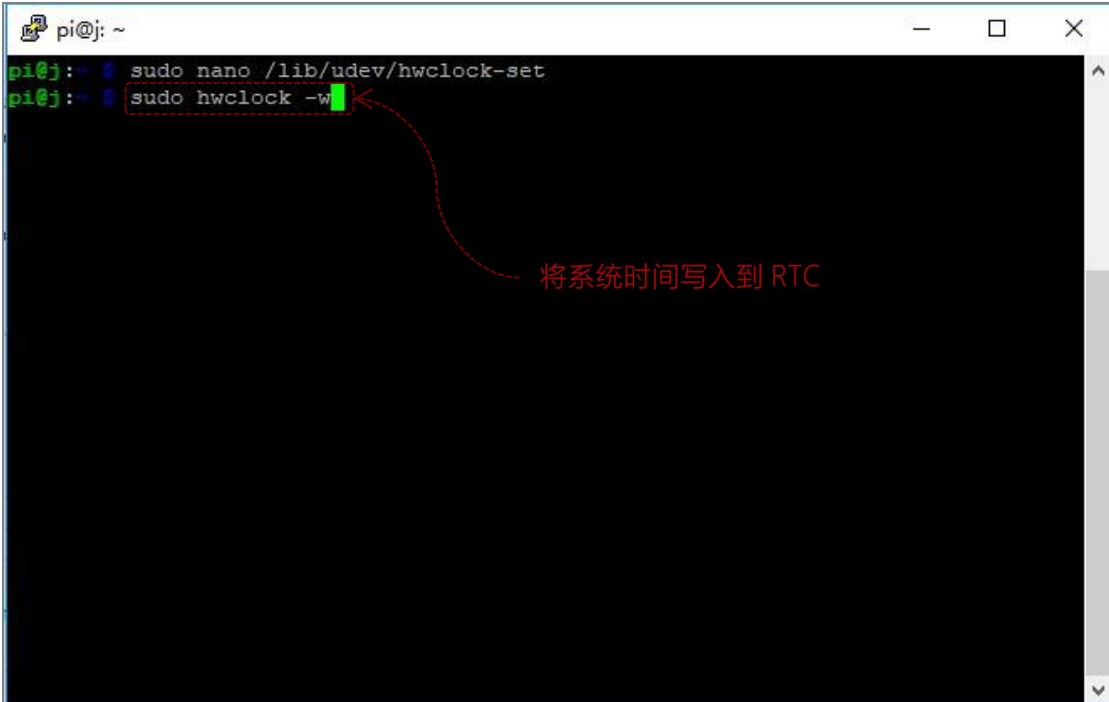
# These defaults are user-overridable in /etc/default/hwclock

[ 已读取 37 行 ]

^G 求助      ^O Write Out ^W 搜索      ^K 剪切文字  ^J 对齐      ^C 光标位置
^X 离开      ^R 读档      ^\ 替换      ^U Uncut Text ^T To Linter ^_ 跳行
```

注释掉这些语句

h. 运行 `hwclock -w` 把当前系统时间写入到 RTC 里，即可完成 RTC 设置。



```
pi@j: ~  
pi@j:~$ sudo nano /lib/udev/hwclock-set  
pi@j:~$ sudo hwclock -w
```

将系统时间写入到 RTC

电池计量功能操作：

用 nano 编辑器新建一个名字为 UPS_18650.py 的脚本程序，代码如下（详细代码见附录）。这个脚本是利用 Python 的 smbus 库对 MAX17040G 进行 i2c 读操作。MAX17040G 设备地址为 0x36，寄存器 VCELL 为 12bit 电池电压 ADC 测量值，地址为 02h-03h，ADC 测量精度单位为 1.25mV。寄存器 SOC 为 16bit 电池容量百分比读数，地址为 04h-05h，SOC 的高 8bit 单位为电池容量的 1%，低 8bit 单位为 1/256%，提供电池容量百分比小数位的读数。更多芯片的信息请查看 MAX17040G 芯片手册。

MAX17040G 寄存器地址以及功能介绍

ADDRESS (HEX)	REGISTER	DESCRIPTION	READ/ WRITE	DEFAULT (HEX)
02h-03h	VCELL	Reports 12-bit A/D measurement of battery voltage.	R	—
04h-05h	SOC	Reports 16-bit SOC result calculated by ModelGauge algorithm.	R	—
06h-07h	MODE	Sends special commands to the IC.	W	—
08h-09h	VERSION	Returns IC version.	R	—
0Ch-0Dh	RCOMP	Battery compensation. Adjusts IC performance based on application conditions.	R/W	9700h
FEh-FFh	COMMAND	Sends special commands to the IC.	W	—

VCELL 寄存器

MSB—ADDRESS 02h								LSB—ADDRESS 03h							
2 ¹¹	2 ¹⁰	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	0	0	0	0
MSB								LSB							
0: BITS ALWAYS READ LOGIC 0															
UNITS: 1.25mV FOR MAX17040 2.50mV FOR MAX17041															

SOC 寄存器

MSB—ADDRESS 04h								LSB—ADDRESS 05h							
2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴	2 ⁻⁵	2 ⁻⁶	2 ⁻⁷	2 ⁻⁸
MSB								LSB							
UNITS: 1.0%															

UPS_18650.py 脚本程序 (详细代码见附录)

```
pi@j: ~
GNU nano 2.2.6 File: UPS_18650.py

#!/usr/bin/env python
import struct
import smbus
import sys
import time

def readVoltage(bus):

    "This function returns as float the voltage from the Raspi UPS Hat via $
    address = 0x36
    read = bus.read_word_data(address, 2)
    swapped = struct.unpack("<H", struct.pack(">H", read))[0]
    voltage = swapped * 1.25 / 1000 / 16
    return voltage

def readCapacity(bus):

    "This function returns as a float the remaining capacity of the battery$
    address = 0x36
    read = bus.read_word_data(address, 4)
    swapped = struct.unpack("<H", struct.pack(">H", read))[0]
    capacity = swapped / 256
    return capacity

bus = smbus.SMBus(1) # 0 = /dev/i2c-0 (port I2C0), 1 = /dev/i2c-1 (port I2C1)

while True:

    print "+++++"
    print "Voltage:%5.2fV" % readVoltage(bus)

    print "Battery:%5i%%" % readCapacity(bus)

    if readCapacity(bus) == 100:

        print "Battery FULL"

    if readCapacity(bus) < 20:

        print "Battery LOW"
    print "+++++"
    time.sleep(2)

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

保存 UPS_18650.py 脚本程序到一个自己知道的目录下(如以下保存到 /home/pi/ 目录下), 然后用 Python 运行该程序, 可以看到程序每隔两秒就会输出当前电池的电压值以及电池容量百分比。另外需要说明一下, 在装好电池后, MAX17040G 需要花几个小时来学习电池的特性, 所以几个小时后读取的电池测量数值才是比较准确的。另外由于 MAX17040G 电池容量的计算方式, 当电池容量读数为 1% 时, 此时电池电压的读数约为 3.5V。由于锂电池一般电压为 3.5V 时, 对应的电池容量已经很低了, 过度放电会损坏电池, 所以用户后续编写低电量自动关机程序时, 建议电池容量为 1% 时就自动关闭 pi, 如果继续运行的话, 当电池电压下降到 3.0V 时, UPS-18650 会自动停止供电。

```
pi@j: ~  
pi@j:~ $ cd /home/pi/  
pi@j:~ $ ls  
Desktop Downloads j Pictures python_games UPS_18650.py  
Documents example.mp3 Music Public Templates Videos  
pi@j:~ $ python UPS_18650.py  
+++++  
Voltage: 4.19V  
Battery: 100%  
Battery FULL  
+++++  
+++++  
Voltage: 4.19V  
Battery: 100%  
Battery FULL  
+++++  
+++++  
Voltage: 4.19V  
Battery: 100%  
Battery FULL  
+++++  
+++++  
Voltage: 4.19V  
Battery: 100%  
Battery FULL  
+++++
```

运行脚本程序

保存的脚本程序

打印出电压值及电池容量百分比

附录:

UPS_18650.py 的脚本程序代码:

```
#!/usr/bin/env python
import struct
import smbus
import sys
import time

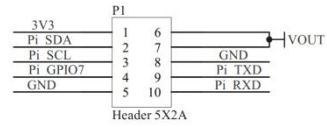
def readVoltage(bus):
    """This function returns as float the voltage from the Raspi UPS Hat via the provided SMBus object"""
    address = 0x36
    read = bus.read_word_data(address, 2)
    swapped = struct.unpack("<H", struct.pack(">H", read))[0]
    voltage = swapped * 1.25 / 1000 / 16
    return voltage

def readCapacity(bus):
    """This function returns as a float the remaining capacity of the battery connected to the Raspi UPS
    Hat via the provided SMBus object"""
    address = 0x36
    read = bus.read_word_data(address, 4)
    swapped = struct.unpack("<H", struct.pack(">H", read))[0]
    capacity = swapped / 256
    return capacity

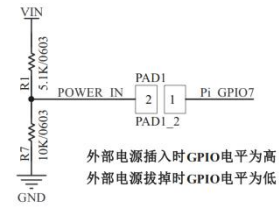
bus = smbus.SMBus(1) # 0 = /dev/i2c-0 (port I2C0), 1 = /dev/i2c-1 (port I2C1)

while True:
    print "+++++"
    print "Voltage:%5.2fV" % readVoltage(bus)
    print "Battery:%5i%" % readCapacity(bus)
    if readCapacity(bus) == 100:
        print "Battery FULL"
    if readCapacity(bus) < 20:
        print "Battery LOW"
    print "+++++"
    time.sleep(2)
```

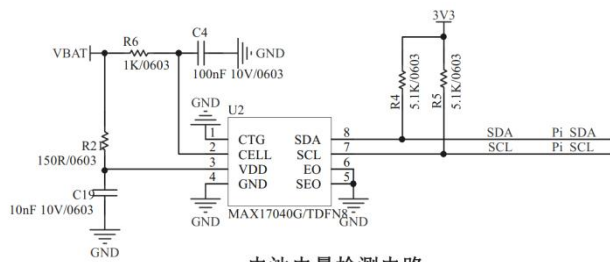
部分参考原理图：



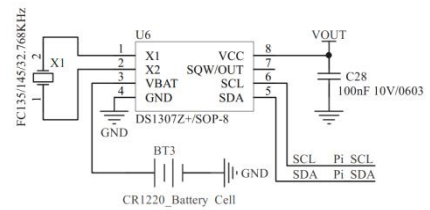
树莓派顶针接口



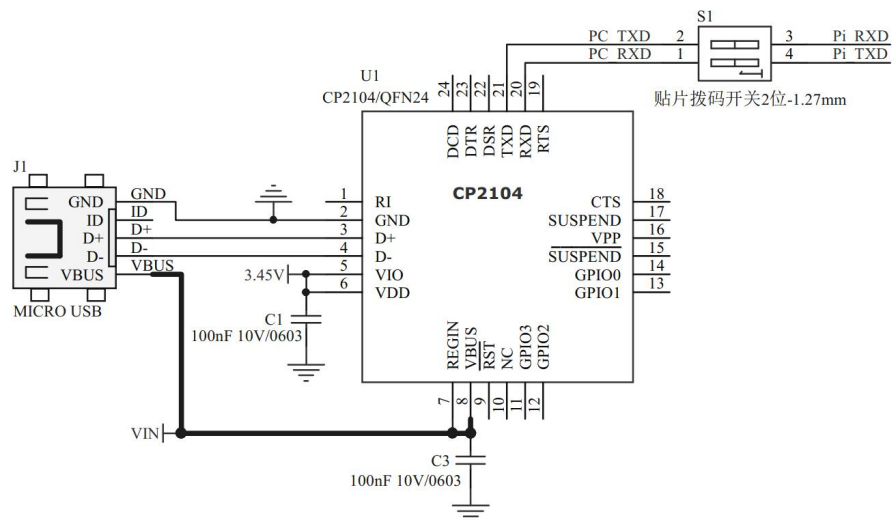
外部电源插入检测



电池电量检测电路



RTC时钟电路



USB转UART

参考资料:

《边学边用树莓派-4》I2C 总线的使用和 DS1307 RTC 设置.

<http://www.dfrobot.com.cn/community/forum.php?mod=viewthread&tid=2636&extra=page=3>

UPS-18650 RTC 设置建议.

<https://github.com/linshuqin329/UPS-18650/issues/1>

树莓派添加 RTC 时钟模块的方法 - CSDN 博客

<http://blog.csdn.net/huayucong/article/details/49982053>

树莓派学习笔记——I2C Tools 学习笔记 - CSDN 博客

<http://blog.csdn.net/xukai871105/article/details/15029843>

MAX17040 结构紧凑的低成本 1S/2S 电量计 - Maxim 美信

<https://www.maximintegrated.com/cn/products/power/battery-management/MAX17040.html>

DS1307 64 x 8、串行、I²C 实时时钟 - Maxim 美信

<https://www.maximintegrated.com/cn/products/digital/real-time-clocks/DS1307.html>

CP2104 驱动下载 USB to UART Bridge VCP Drivers | Silicon Labs

<https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>

为 UPS-18650 添加电池电量图标

<https://github.com/mezl/pi-battery-widget>

联系我:



微信 扫一扫

感谢