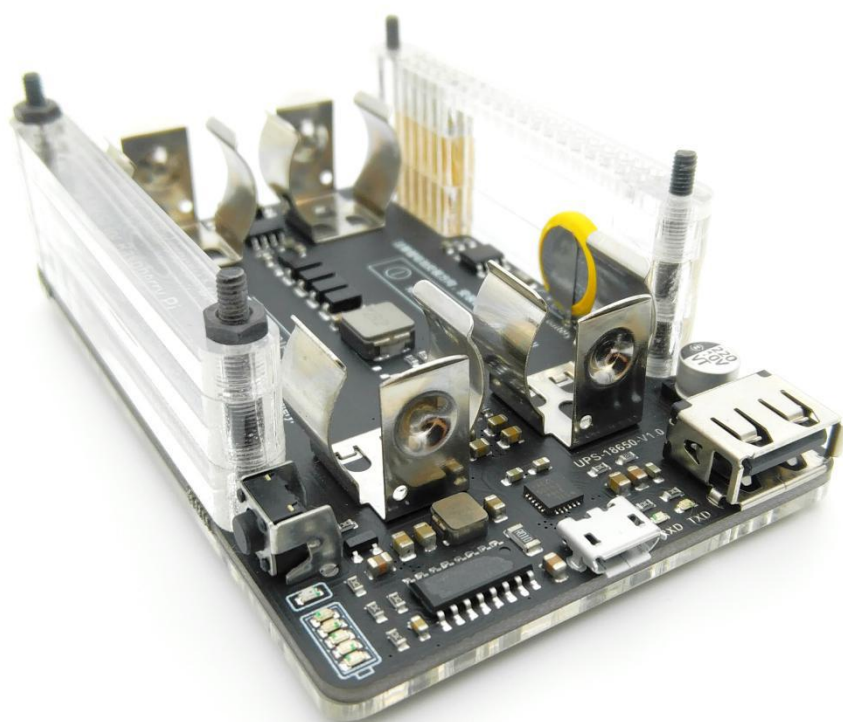


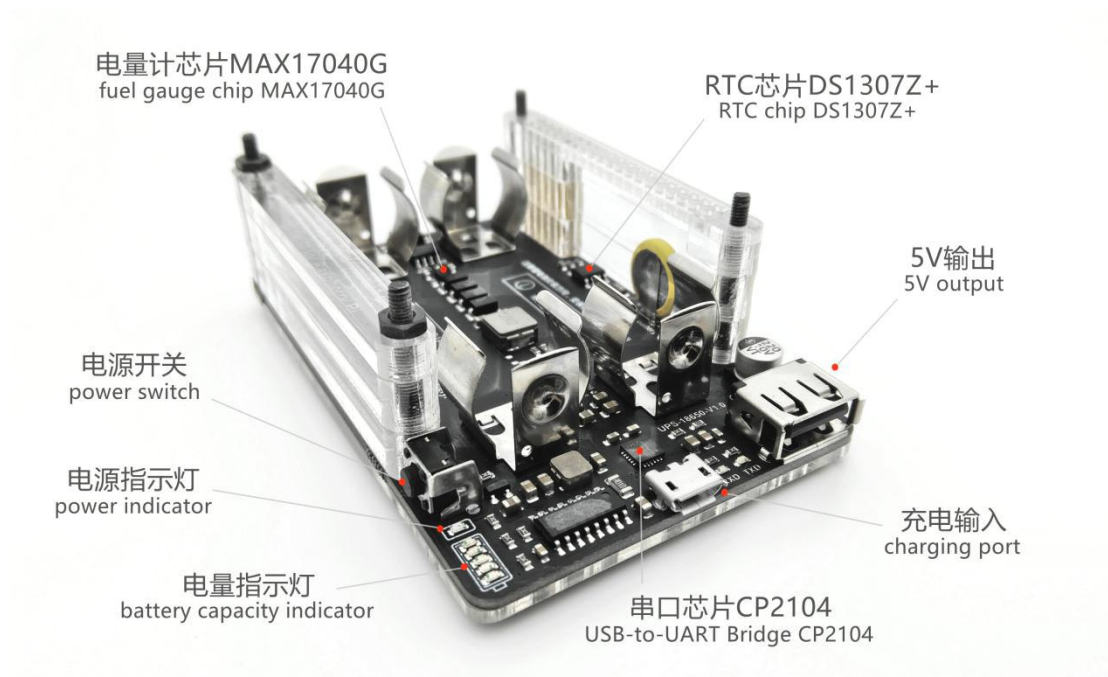
# UPS-18650 使用说明

for raspberry pi 1/2/3 V1.0 -- 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 18650 锂电池进行供电，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 转由锂电池供电。UPS-18650 通过 10 根大电流顶针与 pi 主板连接，所有使用时 pi 无需用 microusb 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：

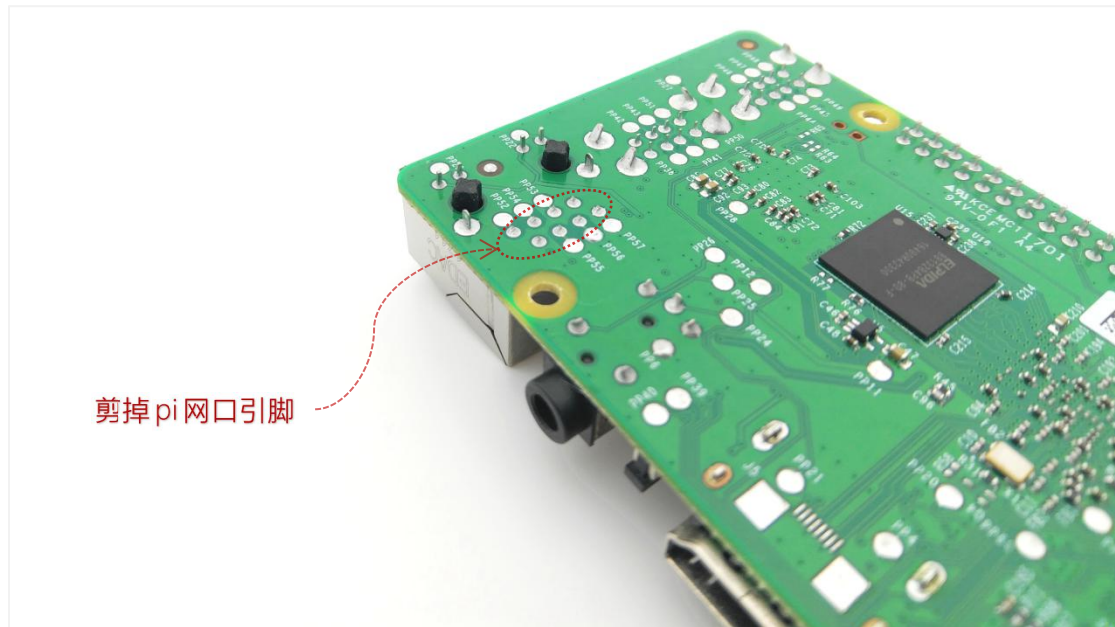
充电电流：**max 1A@5V**

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)

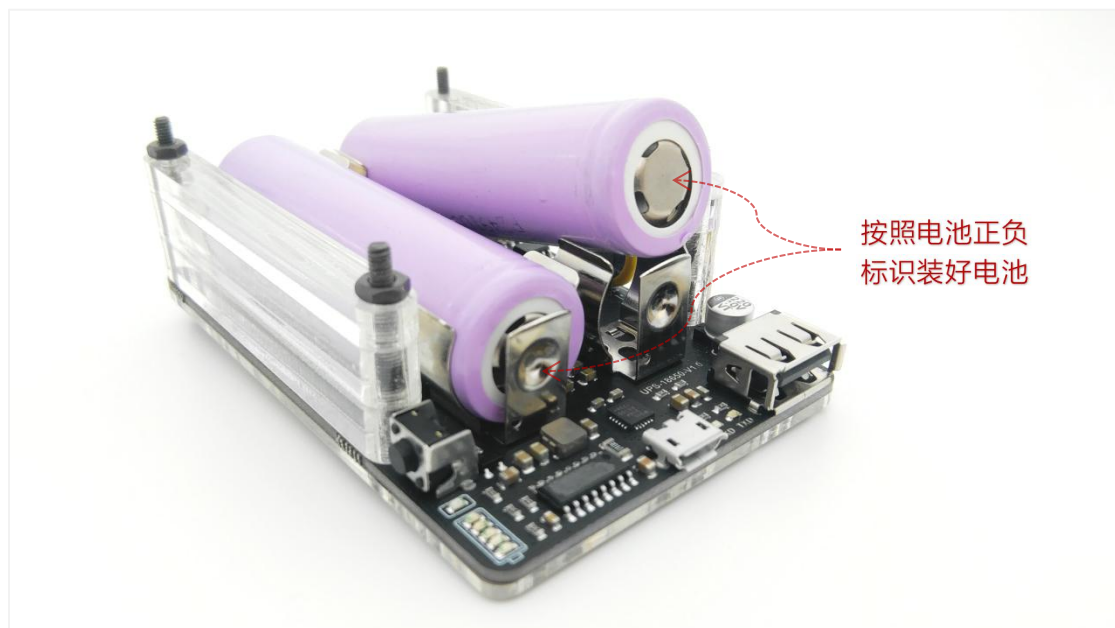
电量测量：输出电池电量百分比，误差 $\pm 1\%$ ，电压测量误差在 $\pm 3\text{mV}$

## 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. 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）。



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，请用镊子按照板子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 5V2A 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 io4(BCM 编号)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。







### 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：

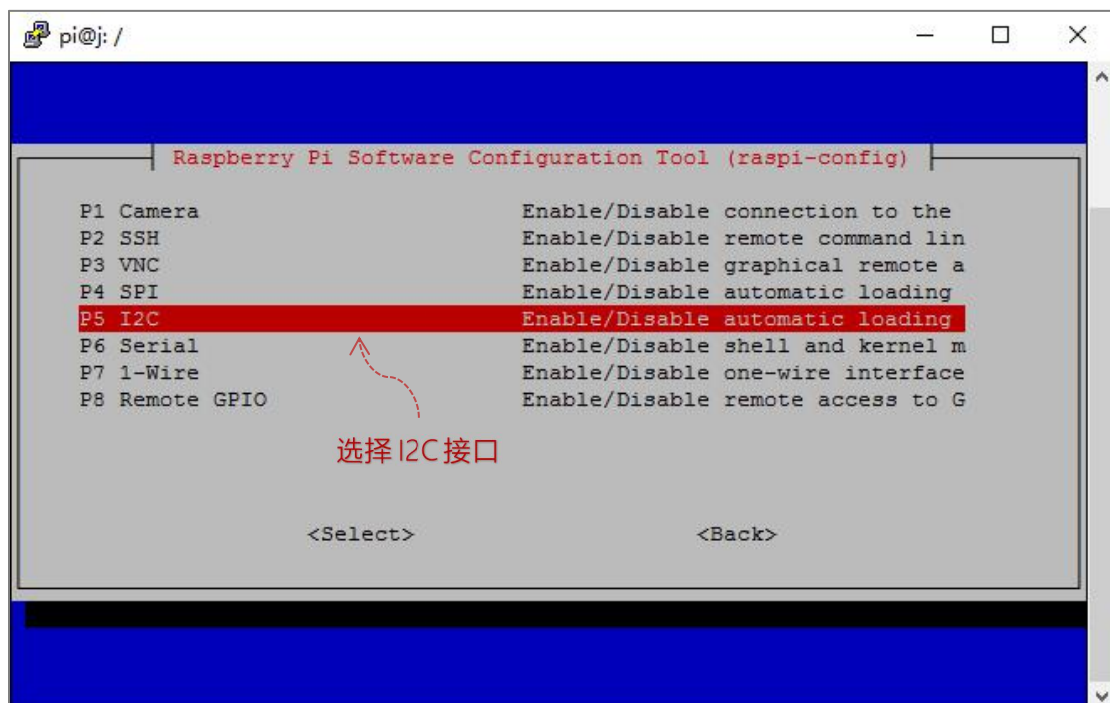
由于 pi 功耗比较大，一般的 USB 口输出电流（USB2.0 500mA, USB3.0 900mA）勉强够 pi 使用，但当 pi 进行复杂的运算时功耗会超过 USB 口的输出，所以进行串口调试时需要配合外部电源适配器使用，先把拨码开关全部拨到“ON”上，PC 上装好串口芯片 CP2104 的驱动，pi 插上 usb 线与 PC 连接，PC 上打开相应的串口软件如 PuTTY，之后再用电源适配器给 pi 供电，USP-18650 上的 5V 输出无须打开，既 USB 口现在只给锂电池充电不给 pi 供电。





## 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。

```
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:~$
```

运行 `i2cdetect -l` 查看 i2c 总线

地址 0x36 为 MAX17040G

地址 0x68 为 DS1307Z+

e. 获取 root 权限，运行 modprobe 命令加载 i2c-dev 模块，之后建立一个新的 i2c 从设备，名字为 ds1307，设备地址为 0x68，之后运行 hwclock -r 读取 RTC 硬件时钟读数。



```
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: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

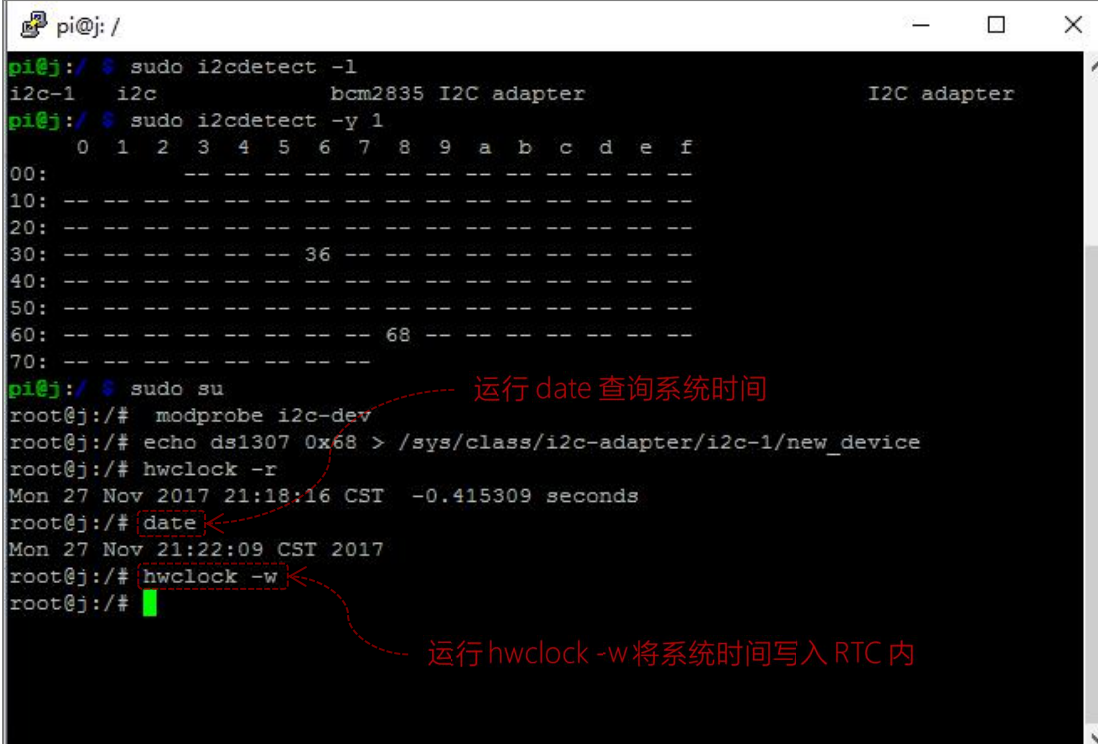
pi@j: / $ sudo su
root@j: /# modprobe i2c-dev
root@j: /# echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
root@j: /# hwclock -r
Mon 27 Nov 2017 21:18:16 CST -0.415309 seconds
root@j: /#
```

运行 modprobe 加载 i2c 模块

新建一个地址为 0x68 的 i2c 设备

运行 hwclock -r 读取 RTC 时钟读数

f. 运行 date 命令获取系统时钟，再运行 hwclock -w 将当期系统时钟写入 RTC 硬件时钟



```
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: / $ sudo su
root@j: /# modprobe i2c-dev
root@j: /# echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
root@j: /# hwclock -r
Mon 27 Nov 2017 21:18:16 CST -0.415309 seconds
root@j: /# date
Mon 27 Nov 21:22:09 CST 2017
root@j: /# hwclock -w
root@j: /#
```

运行 date 查询系统时间

运行 hwclock -w 将系统时间写入 RTC 内

g. 编辑系统启动文件 rc.local，设置开机时将 RTC 时钟同步到系统时钟去。

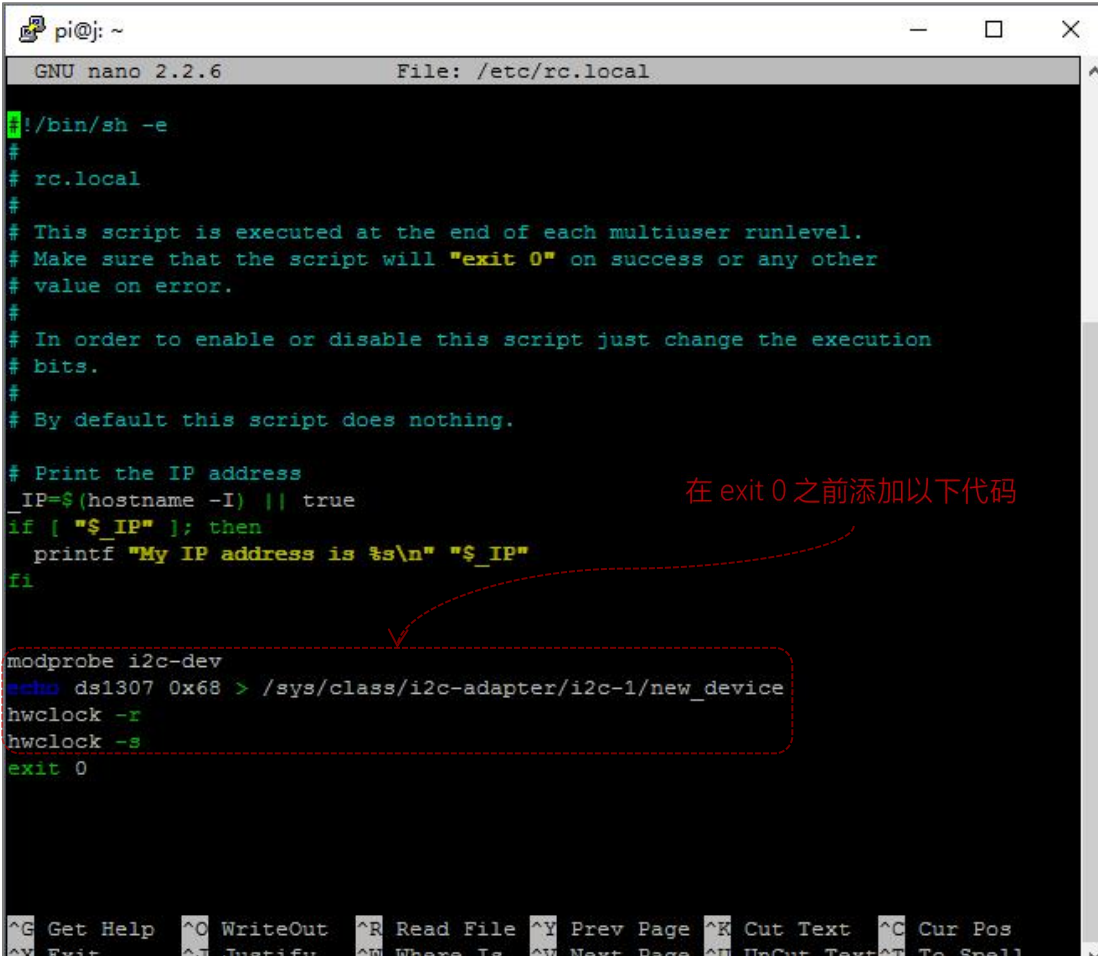


A terminal window titled 'pi@j: ~' showing the command `sudo nano /etc/rc.local` entered at the prompt. A red dashed arrow points from the text '用 nano 编辑器编辑系统启动文件' to the command.

```
pi@j: ~$ sudo nano /etc/rc.local
```

用 nano 编辑器编辑系统启动文件

h. 在系统启动文件 rc.local 中添加如下代码



A terminal window titled 'pi@j: ~' showing the contents of the file `/etc/rc.local` being edited with nano. The file content includes comments and a script to print the IP address. A red dashed box highlights the added code for i2c device and hwclock, with a red dashed arrow pointing from the text '在 exit 0 之前添加以下代码' to it.

```
GNU nano 2.2.6 File: /etc/rc.local

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

modprobe i2c-dev
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
hwclock -r
hwclock -s
exit 0
```

在 exit 0 之前添加以下代码



**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。

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 <sup>11</sup>	2 <sup>10</sup>	2 <sup>9</sup>	2 <sup>8</sup>	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	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 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	2 <sup>-1</sup>	2 <sup>-2</sup>	2 <sup>-3</sup>	2 <sup>-4</sup>	2 <sup>-5</sup>	2 <sup>-6</sup>	2 <sup>-7</sup>	2 <sup>-8</sup>
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  
+++++  
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

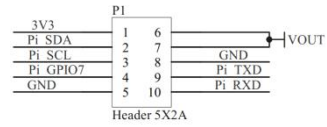
def readVoltage(bus):
    """This function returns as float the voltage from the Raspi UPS Hat via the provided SMBus object"""
    address = 0x36
    read = bus.read_word_data(address, 2)
    swapped = struct.unpack("<H", struct.pack(">H", read))[0]
    voltage = swapped * 1.25 / 1000 / 16
    return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

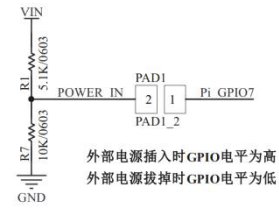
bus = smbus.SMBus(1) # 0 = /dev/i2c-0 (port I2C0), 1 = /dev/i2c-1 (port I2C1)

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)
```

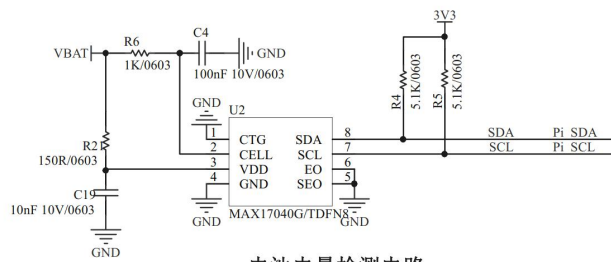
## 部分参考原理图：



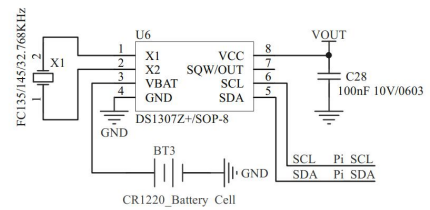
树莓派顶针接口



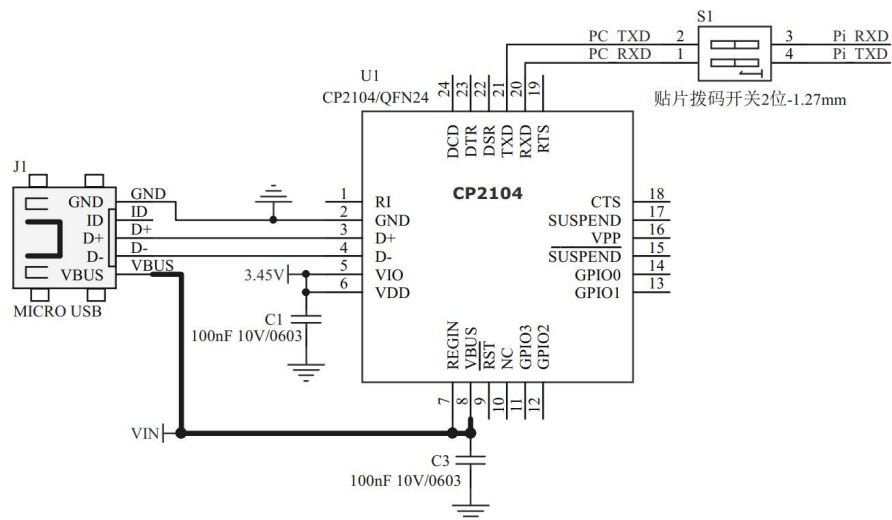
外部电源插入检测



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>

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、I<sup>2</sup>C 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>

## 联系我：



微信 扫一扫

感谢