Application Note

使用智能电池充电器估计充电状态

Nick Brylski

摘要

电池的充电状态 (SOC) 是一种度量，它表示电池的剩余容量占其可用容量的百分比。电量计通常用于计算 SOC，方法是测量电池电压、电流和温度，作为计量算法的输入。使用电量计通常会实现准确的 SOC 预测。但是，使用电量计也存在一些缺点，例如系统成本增加、解决方案尺寸更大以及额外的功耗。在不需要高 SOC 精度的应用中，简单的基于电压的测量方法就足够了。本应用手册讨论了一种使用电池充电器 IC（例如 BQ25155）的内置 ADC 实现基于电压的 SOC 的方法。本应用手册也适用于同一系列中的其他充电器，包括 BQ25150 和 BQ25157，或任何允许测量电池电压的充电器。

内容

插图清单

表格清单

商标

1 引言

通过使用将电池电压与 SOC 相关联的静态查找表，可以简单地了解产品中的电池 SOC。该查找表可以通过在测量电压和电流时对满电量电池进行放电来生成。

2 电池特征

必须采取以下步骤才能生成电压与 SOC 关系表。此设置需要电压表、电流表、电子负载和适当的数据记录设备。图 2-1 显示了测试设置图。

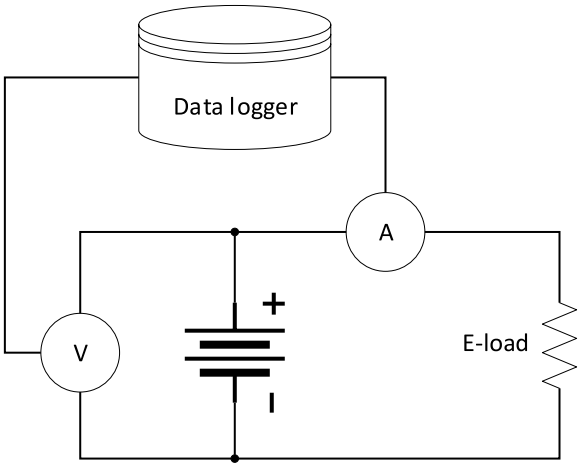

图 2-1. 测试设置

1. 确保按照制造商的规格将电池充满电。
2. 将电压表连接到电池正极端子。
3. 用电流表串联电子负载到电池正极端子。
4. 确保记录软件已打开，并定期读取电压和电流读数（每 5 到 10 秒一次即可）。
5. 将负载设置为您的应用典型放电速率，并将电池放电至放电终止电压。

C/10 或更低的值是理想的，可以更大限度减少弛豫的影响。请注意，此过程需要一段时间（> 10 小时）。*最佳用例* 部分对此假设有更多评论。

3 生成查找表

计算 SOC 的常用方法是将电池剩余容量 (RemCap) 除以电池最大容量 Q_{max}，其中这两个参数均以毫安时 (mAh) 为单位。

首先，计算测试过程中通过的总电量（电池最大容量）。

$$Q_{max} = \sum_{k=1}^{m} i[k] \times \Delta t \quad (1)$$

i[k] 是读数 k 时的电流，Δt 是读数之间的时间差，m 是读数的总个数。

可以在每个读数 n 处计算剩余容量。

$$\mathrm{RemCap[n]} = Q_{max} - \sum_{k=1}^{n} i[k] \times \Delta t \quad (2)$$

然后可以计算 SOC。

$$\mathrm{SOC[n]} = \frac{RemCap[n]}{Q_{max}} \times 100 \quad (3)$$

绘制电池电压与 SOC 的关系图会生成一个锂离子电池的典型 SOC 曲线。

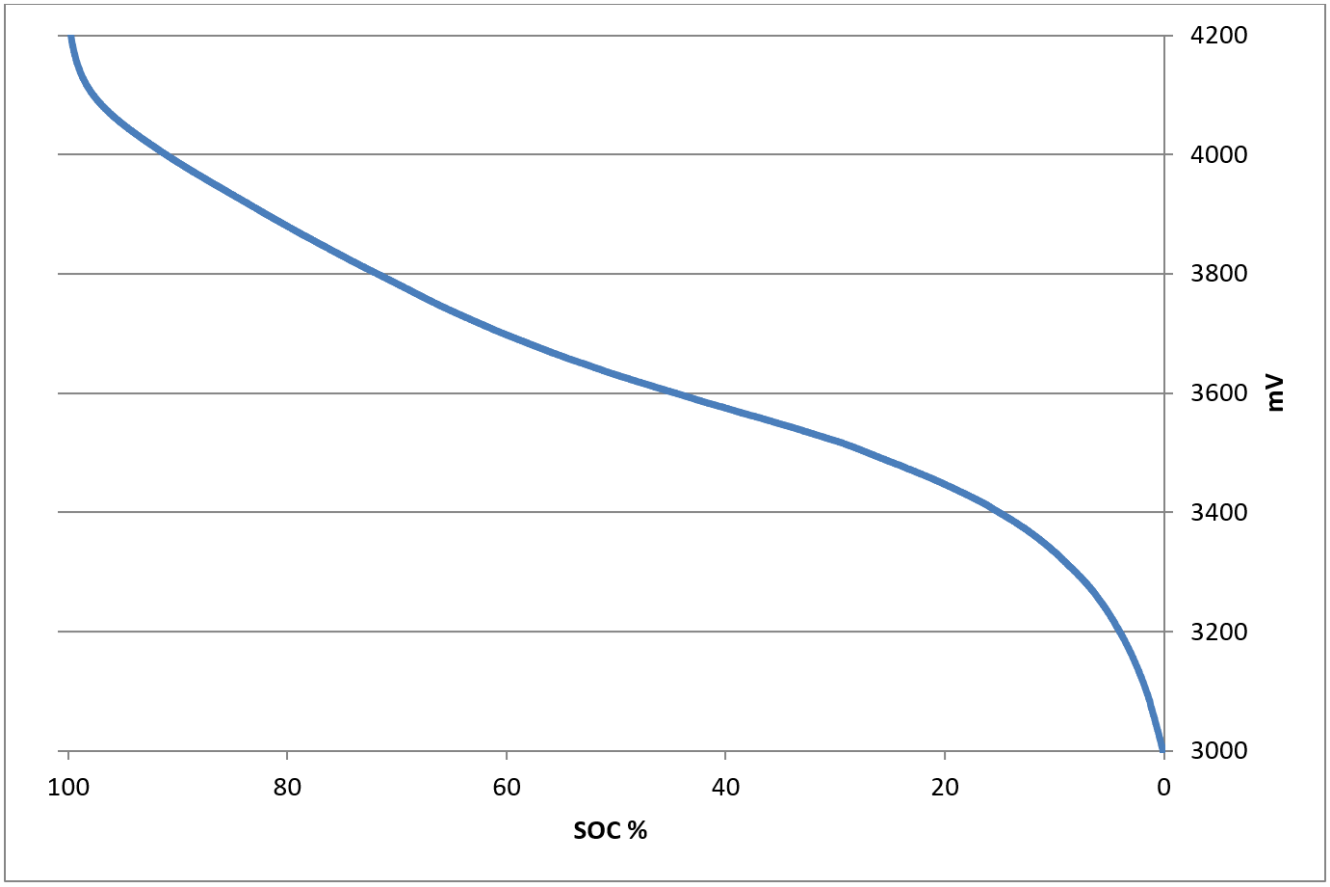

图 3-1. Vbat 与 SOC 关系图

此方法已用于确定 TI 电量计的精度，如此处所示。上面的曲线代表给定放电的确切 SOC。

示例代码已包含在 *Python 查找表生成器* 部分中，该部分根据 SOC 表征生成的数据生成多项式回归。然后将该数据映射到 101 点十六进制查找表，以便轻松导入 MCU 应用中。使用 16 位分辨率，该表只占用 202 字节的内存。

4 BQ25155 寄存器配置

BQ25155 是一款高度集成的电池充电管理 IC，集成了用于可穿戴设备和便携式设备的常用功能，即充电器、用于系统电源的稳压输出电压轨、用于电池和系统监控的 16 位 ADC、LDO 以及按钮控制器。BQ25155 IC 集成了具有 PowerPath 的线性充电器，可实现对小型电池进行快速准确的充电，同时为系统提供稳定电压。稳定系统电压 (PMID) 输出可根据下游 IC 和系统负载的建议运行条件通过 I2C 来配置，以实现卓越的系统运行。

为了限制 ADC 转换的次数，从而降低功耗，在电池运行模式下的 ADC 转换可以限制在由 ADC_READ_RATE 位确定的周期内。在 ADC_READ_RATE 设置为手动模式的情况下，主机必须设置 ADC_CONV_START 位，以启动 ADC 转换。ADC 转换完成且数据准备好后，设置 ADC_READY 标志并向主机发送中断。在低功耗模式下，ADC 保持关闭状态，以实现极低 IC 功耗。在执行 ADC 测量之前，主机必须切换到电池运行模式 (将 LP 设置为高电平) 。

BQ25155 允许通过其寄存器创建高级电池监控和测量固件。相关寄存器在下方列于表 4-1 中。

MSP430 代码片段 部分中还包括一个 MSP430 代码片段，它显示了与 BQ25155 的接口。

表 4-1. BQ25155 寄存器

	地址	R 或 R/W	注释
CHRG_CV_STAT	0x0	R	恒压充电模式 (锥度模式) 状态。
CHARGE_DONE_STAT	0x0	R	充电完成状态。可用于在 FW 中将 SOC 强制为 100%。

表 4-1. BQ25155 寄存器 (continued)

	地址	R 或 R/W	注释
CHRG_CV_FLAG	0x3	R	恒压充电模式（锥度模式）标志。可配置为中断，以提醒主机 MCU。
CHARGE_DONE_FLAG	0x3	R	充电完成标志。可配置为中断，以提醒主机 MCU。
BAT_UVLO_FAULT_FLA G	0x4	R	电池欠压标志。 可用于在 FW 中将 SOC 强制为 0%，阈值可配置为 2.4 – 3V。可配置为中断，以提醒主机 MCU。
ADC_READY_FLAG	0x5	R	ADC 就绪标志。 可配置为中断，以提醒主机 MCU。
ADC_READ_RATE_1:0	0x40	R/W	仅 BAT 操作中 ADC 测量的读取速率。可配置为手动或自动转换。
ADC_CONV_START	0x40	R/W	ADC 转换开始触发器。转换完成时，位回到 0。用于手动读数。
ADC_COMP1_2:0	0x40	R/W	比较器 1 的 ADC 通道。可配置为 VBAT 通道。
1_ADCALARM_15:4	0x53	R/W	ADC 警报 1 阈值，可配置为在 VBAT 达到或低于指定阈值时触发中断。如果需要不止一个阈值，则可以再提供两个警报（都可以设置为监视 VBAT）。
EN_VBAT_READ	0x58	R/W	为电池电压 (VBAT) 通道启用测量。

5 最佳用例

必须注意讨论的方法有一些限制。已知 SOC 是电芯开路电压 (OCV) 的函数。该方法假设 SOC 是电芯终止电压的函数。在以下条件下，此近似值最准确：

1. 当系统负载曲线类似于恒流放电时（就像用于表征电池的那样）。变化的电流会引入一些误差，这些误差可以根据应用和电流变化的幅度来容忍。
2. 低周期计数电池。高周期计数电池的电阻会发生变化，从而导致不同的负载电压曲线。
3. 低电流 (< C/10)。以低 C 速率放电的电池经历较少的电压弛豫。如果电池的低 C 速率放电停止（系统进入睡眠模式），则由于向上弛豫而引入的 SOC 误差量小于电池以高速率放电。
4. 室温。在极端情况下，高温和低温会显著改变电芯的 OCV。随着电芯温度偏离其表征的温度，预计误差会增加。

BQ27621-G1 是一款基于电压的电量计。该测量仪表可根据端子电压估算电流，从而实现更高的精度。该测量仪表适用于低电流应用，而库仑计数测量仪表的电流测量精度不符合此应用要求。可以考虑与 BQ27421-G1 类似的库仑计数测量仪表。

6 Python 查找表发生器

```
#Command Line Arguments: [TI format gauge output csv file], [Polynomial order for regression]
#Output: Plots VBAT vs SOC reported by TI gauge and creates a polynomial regression of the
specified order.
#        This regression is plotted on the same graph as the data and is mapped to a 101-pt
hexadecimal lookup table given in the "lookup_table.txt" file.import matplotlib.pyplot as plt
import numpy as np
import csv
import sys
vbat_arr = []
num_reads = 0
read_arr = []
soc_arr = []

poly4x = []
poly4y = []
poly3y = []
poly3x = []

bin_vals = []

#reads in the TI Gauge csv file and puts the data into the corresponding list.
#Adjust this section if not using TI gauge
with open (sys.argv[1]) as csv_file1:
    csv_reader = csv.reader(csv_file1, dialect='excel',delimiter=',')
    line_count = 0
    for row in csv_reader:
        if (line_count > 8): #To get rid of the labels and other unnecessary stuff
            vbat_arr.append(float(row[6]))
            soc_arr.append(int(row[16]))
            num_reads += 1
        line_count += 1


#create a polynomial regression of the order specified in cmd line
polyfunc = np.polyfit(soc_arr, vbat_arr, int(sys.argv[len(sys.argv)-1]))
poly4 = np.poly1d(polyfunc)
#This for loop creates an x and y list from the regression such that it can be plotted later.
#It also calculates the hex values for the battery voltages needed to create the lookup table.
for i in range (0, 101):
    poly4y.append(poly4(i))
    poly4x.append(i)
    vbat_16 = int(round(((poly4(i)/1000)*(2**16))/6)) #Vbat formula found in datasheet
    bin_vals.append(hex(vbat_16))

#This for loop outputs the lookup table to the file called "lookup_table.txt"
with open ("lookup_table.txt","w+") as outfile:
    for i in range(0, 101):
        outfile.write(str(bin_vals[i])[0] + str(bin_vals[i])[1] + str(bin_vals[i])[2].upper() +
str(bin_vals[i])[3].upper() + str(bin_vals[i])[4].upper() + str(bin_vals[i])[5].upper() + ",\n")
#Ensures that hex letters are all uppercase
outfile.close()

#The rest of this is for plotting the data collected and the calculated regression
plt.plot(soc_arr, vbat_arr, 'r', label='Battery Data')
plt.yticks([3000, 3200, 3400, 3600, 3800, 4000, 4200, 4400])
plt.plot(poly4x, poly4y,'b',label='Regression')

plt.xlabel('SOC (%)')
plt.ylabel('Battery Voltage (mV)')
plt.gca().invert_xaxis() #Reverses x axis so that 100% is shown as the leftmost value
plt.title('Battery Voltage vs SOC')
plt.legend()
plt.grid(b=True, which='major', axis='both')

plt.show()
```

7 MSP430 代码片段

```
{...

    //Disable Watchdog and Enable TS
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_CHARGERCTRL0, 0x90, &Err);

    // Disable interrupts for all the rest except ADC comparator
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_MASK0, 0xFE, &Err); //Mask all but VIN_PGOOD
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_MASK1, 0xBF, &Err); //Mask all
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_MASK2, 0xF7, &Err); //Mask all
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_MASK3, 0xFF, &Err); //Mask all

    // Enable ADC channels for VBAT only
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_ADC_READ_EN, 0x08, &Err);

    // Enable ADC
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_ADCCTRL0, 0x80, &Err); //Set ADC to perform conversion
every 1s at 24ms conversion speed
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_ADCCTRL1, 0x00, &Err); //Disables comparator channels

    //Set PG pin as GPIO for discharge
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_ICCTRL1, 0x08, &Err);
    StdI2C_P_TX_Single(BQ25150_ADDR, BQ25150_ICCTRL2, 0x10, &Err);

    //GPIO_setAsInputPinWithPullUpResistor(BQ_INT2);
    GPIO_setAsInputPin(BQ_INT2);
    GPIO_enableInterrupt(BQ_INT2);
    GPIO_selectInterruptEdge(BQ_INT2, GPIO_HIGH_TO_LOW_TRANSITION);
    GPIO_clearInterrupt(BQ_INT2);

    StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VBAT_M, &VBAT_Meas_M, &Err); //Finding current
battery voltage
    StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VBAT_M, &VBAT_Meas_L, &Err);
    VBAT_Meas = (uint16_t)(VBAT_Meas_M << 8) | VBAT_Meas_L; //Converting to 16-bit integer
    cur_SOC = find_initial_SOC(VBAT_Meas); //Finding initial SOC
    sprintf(SOC_string, "SOC = %d %%        ", cur_SOC);

    while(1) {

        waitms(1000); //Period between SOC updates
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VBAT_M, &VBAT_Meas_M, &Err);
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VBAT_L, &VBAT_Meas_L, &Err);
        VBAT_Meas = (uint16_t)(VBAT_Meas_M << 8) | VBAT_Meas_L;
        cur_SOC = update_SOC_discharge(cur_SOC, VBAT_Meas); //Update SOC
        sprintf(SOC_string, "SOC = %d %%        ", cur_SOC);    }
}

uint8_t find_initial_SOC(uint16_t VBAT_Meas){
    int i;

    for (i = 99; i >= 0; i--){
        if (VBAT_Meas >= SOC_lookup_table[i]){
            return (uint8_t)(i + 1);
        }
    }
    return 0;
}

uint8_t update_SOC_discharge(uint8_t cur_SOC, uint16_t VBAT_Meas){
    int i;

    for (i = cur_SOC - 1; i >= 0; i--){ //Begins at current SOC so it doesn't have to go through
the whole array each time.
        if (VBAT_Meas >= SOC_lookup_table[i]){
            return (uint8_t)(i + 1); //Must add 1 to the SOC found because array is zero-indexed.
        }
    }

    return 0; //Default is SOC = 0.
}
```

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265