Application Note
ZHCAAE0 – 5. 2021

BQ40Z80 量产校准代码示例

Mason Liu *Sales and Marketing/East China*

ABSTRACT

近年来，为了方便使用，随着越来越多的工具均采用无线化设计。因此，电池的需求也与日俱增。在电池的使用过程中， 我们会需要用到 Gauge（电量计）来监测电池的状态，以及提前预知电池的剩余工作时间以及剩余的容量。在电量计的使用过程中，会需要对 gauge 内部的参数进行校准，以符合当前实际使用的情况。BQ40Z80 是一款支持 2 到 6 节电池的电量计，同时也能够提供保护等功能。本应用报告基于 BQ40Z80，给出了在实际量产过程中该如何通过代码来对这些参数进行校准，从而更好地监测电池状态。

Contents

Figures

1 典型应用

1.1 Gauge 典型应用及算法

BQ40Z80 为一款支持 2 到 6 节电池的电量计，采用了已获专利的 Impedance Tracking 算法，是一款基于电池组的单芯片全集成解决方案，并且利用其集成的高性能模拟外设，测量锂离子电池或者锂聚合物电池的可用容量，电压，电流，温度和其他关键参数，保留准确的数据记录，并通过 SMBus 接口将这些信息报告给系统主机控制器。运用了简化原理图如下图所示：

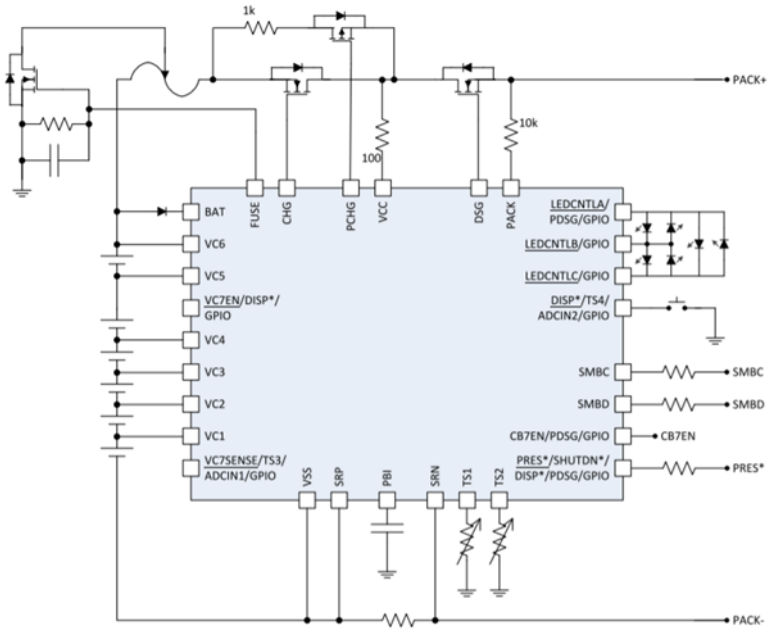

Figure 1.　简化的典型应用图

BQ40Z80 采用的 Impedance Tracking 算法（简称 IT 算法），该算法采用以下三种信息来计算电池的剩余容量（RM）以及电池的满充容量（FCC）：

1） Chemical: 放电深度（DOD）和电池化学容量（Qmax）。

2） Electrical: 电池内阻（R）。

3） External: 负载和温度。

关于 IT 算法的具体细节，可以参考《Theory and implementation of Impedance Track Battery》，这里不再赘述。

1.2 Gauge 校准方法

在实际使用过程中，为了获得准确的测量电压，电流，温度等，会需要对 Gauge 进行校准。校准的参数主要包括电压，电流，温度等等。

校准主要分为两个部分，第一部分为研发过程中的校准，及在项目初期研发过程中的校准，这一部分校准较为简单，具体校准方法可以查看 BQ40Z80EVM User Guide 中的 3.3 Calibration Screen 部分。

第二部分为量产过程中的校准，由于量产过程中无法像研发过程中通过 GUI 针对每一个寄存器实时进行调整，所以我们需要通过代码实现校准。

1.2.1 硬件平台搭建

按照下图所示，完成 EVM 以及硬件平台的连接，主要连接的部分分成三块，电池（battery Stack），负载和充电回路（Load and Charger）以及 EV2400。

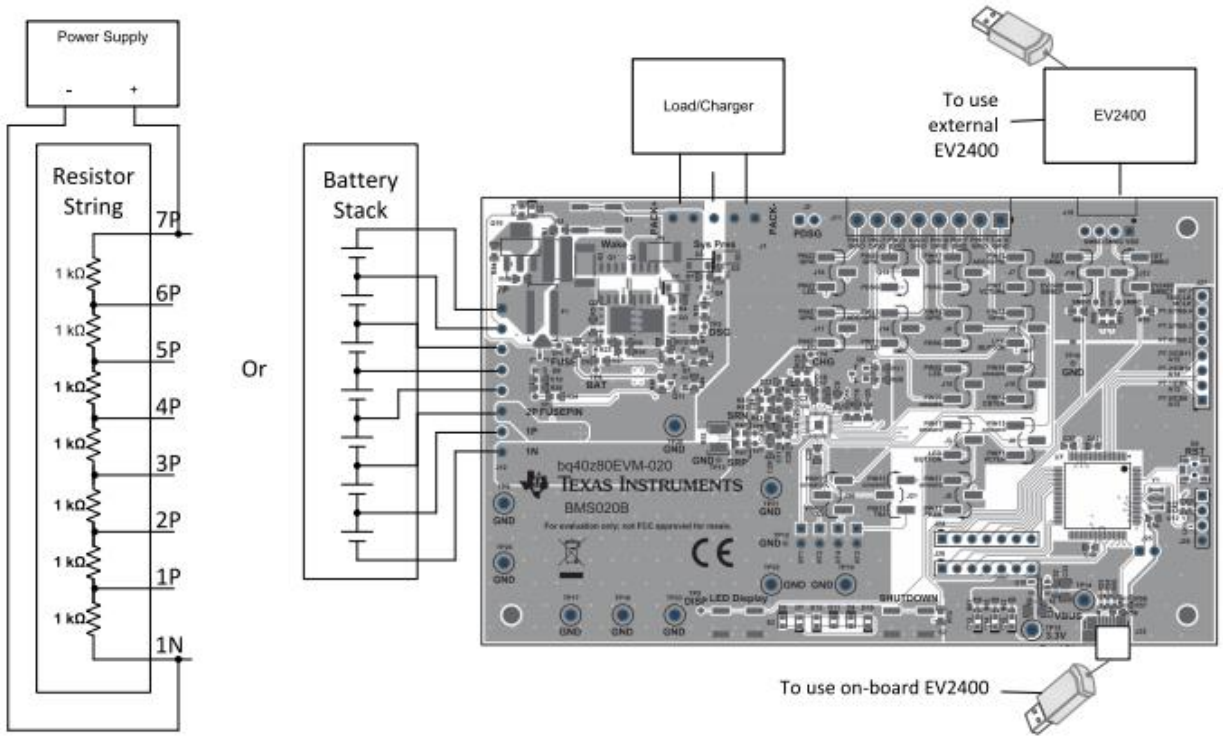

Figure 2. EVM 连接方法

1.2.2 软件平台搭建

由于需要将 EV2400 用作 USB to I2C/SMBus/HDQ 接口，并在此基础上进行编程，所以需要 TI 提供的 SDK 文件，具体的软件环境搭建可以参考《A Quick Start Guide to Program with BQTools SDK》这篇文章，里面有详细的关于如何搭建环境的介绍。

1.2.3 校准流程及构成

本文所提到的代码是基于《BQ40Z80 Manufacture, Production, and Calibration》完成， calibration 可以分成以下几个部分：

1) BQ40Z80 进入 calibration 模式。

2) 电压校准, 这里分为单个电池 Vcell1 至 Vcell7 的校准以及 PACK Voltage 的校准，分别对应程序中的不同寄存器: Cell Gain 和 Pack Gain。

3) 电流校准，这里主要分为 CC offset 校准，Board offset 校准以及 CC Gain 校准。CC Gain 的校准比较好理解，就是针对实际电流的校准，更新内部的寄存器 CC Gain 和 Capacity Gain。

而 CC offset 和 Board offset 都是在 0A 情况下的校准，区别在于 CC offset 是对于内部库伦计数器进行校准，Board offset 是对整版进行校准，一般来说，我们建议先对 CC offset 进行校准，再对 Board offset 进行校准。

4) 温度校准，分为内部的温度传感器校准以及外部的温度传感器校准。

5) BQ40Z80 退出 calibration 模式。

下面就以 Vcell1 校准为例，通过以下的程序例程，来详细说明该如何用代码来做校准。

2 量产校准代码例程

本例程基于《BQ40Z80 Manufacture, Production, and Calibration》的 2.1 节”Cell Voltage 1-6 Calibration”完成，以下结合实际代码，分析如何完成校准。

2.1 开启校准模式

```
1.      printf("Enable Calibration . . .\n");
2.      returnValue = SDKWriteSMBBlock(comChannel.dataPipeName, NMAC, cal_en, 2, 10,
GGTGTADDRESS);
3.      if (returnValue != 0)
        {
4.                  PrintError(returnValue);
5.                  puts("\nPress any key to exit.");
6.                  _getch();
7.                  return 1;
        }
8.      else
        {
9.                  printf("\tDone\n");
        }
```

第一行为在界面中显示出”Enable Calibration”。

第二行为开启校准模式，将 cal_en(0x002D)写入 ManufacturerAcess()寄存器，就会开启 ManufacturingStatus()中的[CAL], 进入校准模式。

第三行到第九行为判断是否通过 SMBus 读到数据，当未出现 error 或者 warning 时，程序中的 SDKWriteSMBBlock 函数会返回 0，进而主程序可以继续。如果出现 error 或者 warning 时，SDKWriteSMBBlock 函数会返回非 0 值，界面上显示 Press any key to exit，键入任意信息主程序都会结束。

2.2 读取 ADC 数据

```
1.      printf("Get Raw Data From ADC_Voltage_Current_Temperature . . .\n");
2.      returnValue = SDKWriteSMBBlkReadBlk(comChannel.dataPipeName, NMAC, output_raw_data, 2,
10, NMAC, dataBlock_cal, BUFSIZE, &numBytesRead_cal, GGTGTADDRESS);
3.      if (returnValue != 0)
        {
4.                  PrintError(returnValue);
5.                  puts("\nPress any key to exit.");
6.                  _getch();
7.                  return 1;
        }
```

第一行为在界面中显示出” Get Raw Data From ADC_Voltage_Current_Temperature . . .”。

第二行为将 output_raw_data(0xF081)写入 ManufacturerAcess()寄存器，就可以从 ManufacturerData()寄存器中读出原始的 ADC 数据，这里还建立了一个数组 dataBlock_cal，用来存放从 ADC 中读取出来的数据。读出数据的排列顺序可以参照下图所示，在 BQ40Z80 TRM 或者上文提到的《BQ40Z80 Manufacture, Production, and Calibration》也都可以找到。

The *ManufacturerData()* output format for 0xF081 and 0xF082 is:
ZZYYaaAAbbBBccCCddDDeeEEffFFggGGhhHHiiIIjjJJkkKKllLLmmMMnnNNooOO,

where:

Value	Format	Description
ZZ	byte	8-bit counter, increments when raw ADC values are refreshed (every 250 ms)
YY	byte	Output status *ManufacturerAccess()* = 0xF081: 1 *ManufacturerAccess()* = 0xF082: 2
AAaa	2's comp	Current (coulomb counter)
BBbb	2's comp	Cell voltage 1
CCcc	2's comp	Cell voltage 2
DDdd	2's comp	Cell voltage 3
EEee	2's comp	Cell voltage 4
FFff	2's comp	Cell voltage 5
GGgg	2's comp	Cell voltage 6
HHhh	2's comp	PACK voltage
IIii	2's comp	BAT Voltage
JJjj	2's comp	Cell current 1
KKkk	2's comp	Cell current 2
LLll	2's comp	Cell current 3
MMmm	2's comp	Cell current 4
NNnn	2's comp	Cell current 5
OOoo	2's comp	Cell current 6

Figure 3. ADC raw data

2.3 检查 ADC 数据

```
1.      printf("\tBytes returned: %d\n", numBytesRead_cal);
2.      returnedAddress_cal = dataBlock_cal[1] << 8 | dataBlock_cal[0];
3.      requestedReadAddress_cal = 0xF081;
4.      if (returnedAddress_cal != requestedReadAddress_cal)
        {
5.                  printf("\tReturned address (0x%04X) does not match the requested address
(0x%04X)!\n", returnedAddress_cal, requestedReadAddress_cal);
        }
6.      else
        {
7.                  printf("\t8-bit counter number: %d\n", dataBlock_cal[2]);
8.                  counter_num = dataBlock_cal[2];
9.                  printf("wait until 8-bit counter number increased by at least 2\n");
10.                 while (1)
                    {
11.                             returnValue =
SDKWriteSMBBlkReadBlk(comChannel.dataPipeName, NMAC, output_raw_data, 2, 10, NMAC, dataBlock_cal,
BUFSIZE, &numBytesRead_cal, GGTGTADDRESS);
```

```
12.                                 printf("\t8-bit counter number: %d\n", dataBlock_cal[2]);
13.                                 bit_counter_number_increment = dataBlock_cal[2] -
counter_num;
14.                                 cycle_times++;
15.                                 if (bit_counter_number_increment > 2)
                                    {
16.                                             break;
                                    }
                        }
17.                     printf("\tCycle Times: %d\n", cycle_times);
18.                     printf("Get all the Raw ADC Data:\n");
19.                     printf("\tReturned Address: 0x%04X\n", returnedAddress_cal);
20.                     printf("\tFinal 8-bit counter number: %d\n", dataBlock_cal[2]);
21.                     printf("\tOutput Status: %d\n", dataBlock_cal[3]);
22.                     printf("\tData:\n");
23.                     printf("\t");
24.                     for (i = 1; i < 30; i++)
                        {
25.                                 printf("0x%02X ", dataBlock_cal[3 + i]);
                        }
26.                     printf("\n");
```

第一行为把数组 dataBlock_cal 的 raw data 均显示在界面上，方便查阅。

由于我们之前读取 raw data 时是采用的 read block, 所以 raw data 中前两位是写入的 command 信息，也就是 0xF081，第二行到第五行代码就是判断读出的 raw data 的前两位 command 值是为 0Xf081，如果不相符，会在界面上显示：Returned address (0x%04X) does not match the requested address. 这里需要注意的一点是在第二行中，我们需要将 raw data 中 LSB MSB 格式（81 F0）的数据转换为 MSB LSB 格式（F0 81）再进行判断。

第六行到十六行为等待 8-bit counter number 增加值大于 2 之后，再来进行下一步操作。这里的要求在量产的 application note 中有明确规定，程序中第六行为在界面中显示出目前 counter number 的数值，之后第十行到第十六行通过 while(1)循环函数，判断当增加值大于 2 时，跳出循环，继而进行后续的计算。

之后的第十七行到第二十六行为在界面显示出 raw data 的各种数值，方便分析，可以按照图三来配置，这里不再一一赘述。

2.4 获得 ADCVcell1 值

```
1.      printf("Get Avg Cell1Voltage\n");
2.      returnValue = SDKWriteSMBBlkReadBlk(comChannel.dataPipeName, NMAC, output_raw_data, 2,
10, NMAC, dataBlock_cal, BUFSIZE, &numBytesRead_cal, GGTGTADDRESS);
3.      counter_num_avgvol = dataBlock_cal[2];
4.      while (1)
        {
5.                  returnValue = SDKWriteSMBBlkReadBlk(comChannel.dataPipeName, NMAC,
output_raw_data, 2, 10, NMAC, dataBlock_cal, BUFSIZE, &numBytesRead_cal, GGTGTADDRESS);
6.                  Cell1Voltage = dataBlock_cal[7] << 8 | dataBlock_cal[6];

7.                  if (Cell1Voltage < 0x8000)
                    {
8.                              ADCcell1 = Cell1Voltage;
```

```
                        }
 9.                     else
                        {
10.                                 ADCcell1 = Cell1Voltage - 0xFFFF - 0x0001;
                        }

11.                     SumCell1Voltage += Cell1Voltage;
12.                     bit_counter_number_increment_avgvol = dataBlock_cal[2] -
counter_num_avgvol;
13.                     cycle_times_avgvol++;
14.                     if (bit_counter_number_increment_avgvol>1)
                        {
15.                                 break;
                        }
                }
16.             AvgCell1Voltage = SumCell1Voltage / cycle_times_avgvol;
17.             printf("\tCycle Times: %d\n", cycle_times_avgvol);
18.             printf("\tAverage Cell 1 Voltage(2's complement) = 0x%04X\n", AvgCell1Voltage);
```

第一行同样是在界面上显示出 Get Avg Cell1Voltage。

第二行到第三行为读取 8-bit counter number 的值，并且存入到 counter_num_avgvol 变量中，保存 8-bit counter number 初始值。

第四行到第十五行为 while(1)循环函数，目的在于获得 ADC 中的 Vcell1 值并且多次取样，获得其平均值，提高精度。

其中第五行和第六行是获得 ADCVcell1 值的代码。

第七行到第十行是进行判断，如果 Vcell1 小于 0x8000, 就采用 ADC 中原始的 Vcell1 值，否则就采用 Vcell1= – (0xFFFF – BBbb + 0x0001)。这一点在《BQ40Z80 Manufacture, Production, and Calibration》有详细描述。

第十一行到第十五行为多次取值，直到 8-bit counter number 增加 1 才停止，并且通过 cycle_times_avgvol 来记录循环次数。方便之后平均值计算。

第十六行为平均值计算，并将结果存入到 AvgCell1Voltage 中。

第十七行和第十八行为在界面上显示出循环次数以及平均后的 ADCVcell1 值。

2.5 获得实际测量 Vcell1 值

```
1.      printf("Please Input Your Cell1Voltage\n");
2.      printf("\tCell1Voltage(mV) = ");
3.      scanf_s("%d", &input_cell1voltage);
4.      Vcell1 = input_cell1voltage;
5.      printf("\tTurn into HEX(mV): %X\n", Vcell1);
```

校准是将 ADCVcell1 值修正为实际的，真实的 Vcell1 值，可以采用万用表或者其他仪器实际测量 Vcell1，将得到的值输入到界面中，也就是第一行到第三行代码执行的功能，第四行和第五行将 Vcell1 转换为十六进制，方便对比和计算。

2.6 计算新 Cell Gain

```
printf("Calculate Cell Gain Value\n");
Cell_Gain = (Vcell1*1.0 / AvgCell1Voltage*1.0) * 65536;
Cell_Gain_int = (unsigned int)(Cell_Gain);
printf("\tCell Gain(HEX) = : %X\n", Cell_Gain_int);
Cell_Gain_intH = Cell_Gain_int / 256;
Cell_Gain_intL = Cell_Gain_int % 256;
write_cell_gain[0] = 0x00;
write_cell_gain[1] = 0x40;
write_cell_gain[2] = Cell_Gain_intL;
write_cell_gain[3] = Cell_Gain_intH;
printf("\tCell Gain(DEC) = : %d\n", Cell_Gain_int);
```

这一段代码的第二行就是通过将实际测试的 Vcell1 除以 ADC 中平均 Vcell1 的值，得到了符合实际测试条件的 Cell Gain.

代码的第四行到第十一行是在界面上显示出来十进制下和十六进制下的 Cell Gain 的值。其中的第七行和第八行提前写入的 0x00 和 0x40 是按照 LSB MSB 顺序排列， 对应的寄存器为 0x4000，也就是 BQ40Z80 的 Data Flash 中 Cell Gain 的地址。

2.7 读取原始 Cell Gain

```
printf("Get Original Cell Gain\n");
returnValue = SDKWriteSMBBlkReadBlk(comChannel.dataPipeName, NMAC,
DF_Data_0x4000_0x4020, 2, 10, NMAC, dataBlock_origin_0x4000_0x4020, BUFSIZE, &numBytesRead_cal,
GGTGTADDRESS);
Origin_Cell_Gain = dataBlock_origin_0x4000_0x4020[3] << 8 |
dataBlock_origin_0x4000_0x4020[2];
printf("\tOringin Cell Gain(HEX) = : %X\n", Origin_Cell_Gain);
printf("\tOringin Cell Gain(DEC) = : %d\n", Origin_Cell_Gain);
```

这一段代码的目的是读取 Gauge 的 Data Flash 中原始的 Cell Gain 的值，实际校准过程可以将这一段变为注释，对实际校准没有影响。

2.8 写入新 Cell Gain

```
printf("Get Modified Cell Gain\n");
returnValue = SDKWriteSMBBlock(comChannel.dataPipeName, NMAC, write_cell_gain, 4, 80,
GGTGTADDRESS);
returnValue = SDKWriteSMBBlkReadBlk(comChannel.dataPipeName, NMAC,
DF_Data_0x4000_0x4020_modified, 2, 80, NMAC, dataBlock_modified_0x4000_0x4020, BUFSIZE,
&numBytesRead_cal, GGTGTADDRESS);
Modified_Cell_Gain = dataBlock_modified_0x4000_0x4020[3] << 8 |
dataBlock_modified_0x4000_0x4020[2];
printf("\tModified Cell Gain(HEX) = : %X\n", Modified_Cell_Gain);
printf("\tModified Cell Gain(DEC) = : %d\n", Modified_Cell_Gain);
```

这一段代码的目的就是写入新的 Cell Gain，至此，Vcell1 的校准工作已经全部完成。

3 例程校准结果

```
        8-bit counter number: 109
        8-bit counter number: 110
        Cycle Times: 17
Get all the Raw ADC Data:
        Returned Address: 0xF081
        Final 8-bit counter number: 110
        Output Status: 1
        Data:
        0x01 0x00 0x6C 0x56 0x60 0x56 0x6A 0x56 0x64 0x56 0x66 0x56 0x5B 0x56 0x9A 0x47 0x2A 0x55 0x00 0x00 0x01 0x00 0x
05 0x00 0x00 0x00 0x02 0x00 0xFB
Get Avg Cell1Voltage
        Cycle Times: 15
        Average Cell 1 Voltage(2's complement) = 0x566D
Please Input Your Cell1Voltage
        Cell1Voltage(mV) = 3400
        Turn into HEX(mV): D48
Calculate Cell Gain Value
        Cell Gain(HEX) = : 2757
        Cell Gain(DEC) = : 10071
Get Original Cell Gain
        Oringin Cell Gain(HEX) = : 2E4B
        Oringin Cell Gain(DEC) = : 11851
Get Modified Cell Gain
        Modified Cell Gain(HEX) = : 2757
        Modified Cell Gain(DEC) = : 10071
Disable Calibration . . .
        Done

Completed. Press any key to exit.
```

Figure 4. 代码校准结果

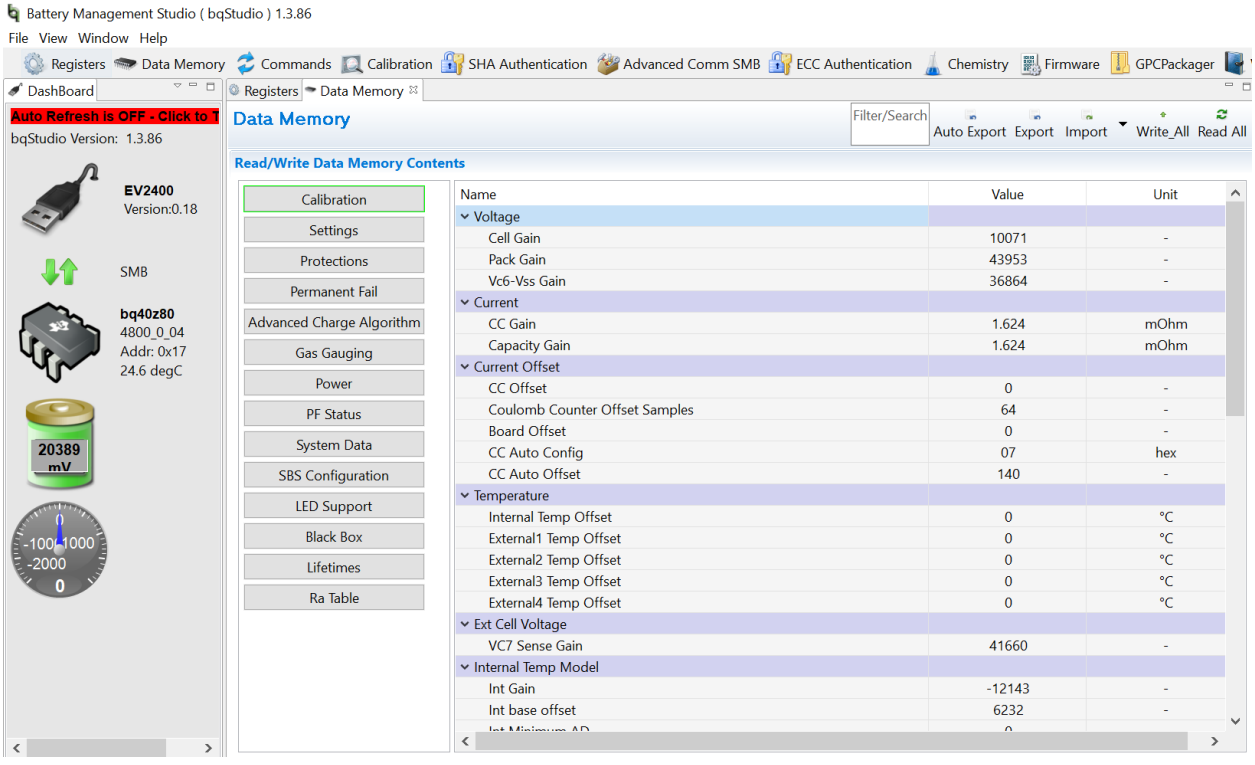

Figure 5. BqStudio 校准结果

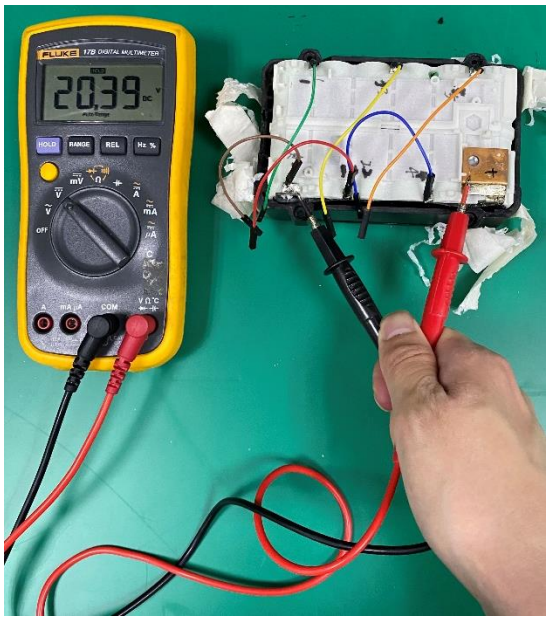

Figure 6.　实际测试结果

图四显示的就是通过代码来校准完成后的界面， Modified Cell Gain(DEC) = 10071 表明校准后的 Cell Gain 为 10071。相应的在 BqStudio 中打开 Data Memory 界面，确认 Cell Gain 确实修改为了 10071，证明校准成功。并且如图六所示，最终实际测试为 20.39V，和上位机读出来的数据相吻合。以上就是针对 Vcell1 的校准，其他 Cell 电压的校准以及电流和温度的校准可以参照本文所述的代码示例，按照《BQ40Z80 Manufacture, Production, and Calibration》的要求来完成。

4　参考文献

1. *BQ40Z80 datasheet (SLUSBV4B)*
2. *BQ40Z80EVM User Guide (SLUUBZ5)*
3. *Theory and implementation of Impedance Track Battery (SLUA450)*
4. *BQ40Z80 Manufacture, Production, and Calibration (SLUA868)*

重要声明和免责声明

TI 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安全、安保或其他要求。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 (https:www.ti.com.cn/zh-cn/legal/termsofsale.html) 或 ti.com.cn 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。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122