*Programmer’s Guide*

# *BQ769x2 校准和 OTP 编程指南*

*Matt Sunna*

**摘要**

BQ769x2 电池监视器系列（包括 BQ76952、BQ76942 和 BQ769142）可通过主机在应用程序中进行配置，或者，用户需要在量产线上固定配置器件参数时，也可以使用 OTP 进行配置。BQ769x2 系列还包括可选功能，用于校准多个关键参数，以提高系统精度。本应用报告提供了校准和 OTP 编程过程所需的步骤。

## 内容



## 表格清单



## 商标

# 1 BQ769x2 器件系列的量产编程

虽然 BQ769x2 可通过主机在应用程序中进行配置，但使用 OTP 在量产线上配置器件更具优势。首先，预配置 OTP 使您能够配置在应用程序中难以配置的重要器件设置。例如，如果应用程序主机依赖 BQ769x2 LDO 供电，或需要选择与默认选项不同的通信类型，那么更实用的操作方法是在 OTP 中预配置这些设置，以便器件以所需设置上电。若要查看可申请预配置的通信类型和 LDO 配置，请参阅器件特定数据表中的器件选型表。

在量产线上配置器件的另一个优势是，支持校准电压、电流和温度测量值，以获得尽可能高的精度。校准仅适用于可严格控制所施加电压、电流和温度的量产线。

# 2 校准

BQ769x2 具有支持原始 ADC 读数的命令，因此可对电压、电流和温度读数进行校准。以下各部分将概述进行不同测量类型的校准所需的步骤、每个读数对应的命令和子命令，以及可存储校准数据的 RAM 参数。

## 2.1 校准精度

根据所需的精度、量产测试设备的能力和测试时间，在量产中实施校准时有几点需要考虑。我们以电芯电压校准为例展开介绍。下面列出了从精度最低 (1) 到精度最高 (4) 的几个不同选项。

1. 无量产校准，使用默认值。
2. 对固定数量的器件执行校准，并使用所有电池组校准参数的平均值。如果观察到每个电路板的差异很小且可以接受，这可能是一个好的方法。
3. 在量产过程中，通过向电芯输入端施加单个精确电压（如 3.5V），对每个器件进行单独校准。对测量的多个读数取平均值可进一步提高精度。
4. 在量产过程中，通过向电芯输入端施加两个精确的直流电压（如 2.5V 和 4.5V），对每个器件进行单独校准。对测量的多个读数取平均值可进一步提高精度。这不仅可以更精确地计算电芯增益，还可以校准电芯的电压偏移。

## 2.2 电芯电压增益校准

### 2.2.1 电芯电压增益校准步骤

以下步骤对于每个电芯都是相同的。在此示例中，显示了电芯 1 的两点校准步骤。

1. 禁用休眠模式（子命令 0x009A），以确保施加电压后电压计数快速更新。
2. 在器件引脚 VC1 和 VC0 之间施加已知电压 $V_{CELL1\_A}$。
3. 100ms 后，使用 DASTATUS1 子命令 0x0071 读取电芯 1 电压计数 ($ADC\_Counts_{CELL1\_A}$)。表 2-1 中提供了用于校准电压读数的命令的完整列表。为了获得最佳精度，请读取多个读数并计算平均值。
4. 在 VC1 和 VC0 之间施加第二个已知电压 $V_{CELL1\_B}$，并读取电芯 1 的电压计数 ($ADC\_Counts_{CELL1\_B}$)。
5. 计算电芯 1 的增益：

$$\textit{Cell 1 Gain} = \frac{2^{24} \times \left(V_{CELL1\_B} - V_{CELL1\_A}\right)}{\left(ADC\_Counts_{CELL1\_B} - ADC\_Counts_{CELL1\_A}\right)} \quad (1)$$

6. 将新的***电芯 1 增益*** 值写入 RAM。
   - 进入 CONFIG_UPDATE 模式（子命令 0x0090）。
   - 将***电芯 1 增益*** 的值写入 RAM 位置 0x9180。
   - 退出 CONFIG UPDATE 模式（子命令 0x0092）。
7. 重新检查电芯 1 的电压读数。如果读数不准确，重复步骤 1-6。

**表 2-1. 用于校准的电压和电流测量子命令**

| 子命令地址 | 名称 | 偏移 | 数据 | 类型 |
|---|---|---|---|---|
| 0x0071 | DASTATUS1 | 0 | 电芯 1 电压计数 | I4 |
| | | 4 | 电芯 1 电流计数 | I4 |
| | | 8 | 电芯 2 电压计数 | I4 |
| | | 12 | 电芯 2 电流计数 | I4 |
| | | 16 | 电芯 3 电压计数 | I4 |
| | | 20 | 电芯 3 电流计数 | I4 |
| | | 24 | 电芯 4 电压计数 | I4 |
| | | 28 | 电芯 4 电流计数 | I4 |
| 0x0072 | DASTATUS2 | 0 | 电芯 5 电压计数 | I4 |
| | | 4 | 电芯 5 电流计数 | I4 |
| | | 8 | 电芯 6 电压计数 | I4 |
| | | 12 | 电芯 6 电流计数 | I4 |
| | | 16 | 电芯 7 电压计数 | I4 |
| | | 20 | 电芯 7 电流计数 | I4 |
| | | 24 | 电芯 8 电压计数 | I4 |
| | | 28 | 电芯 8 电流计数 | I4 |
| 0x0073 | DASTATUS3 | 0 | 电芯 9 电压计数 | I4 |
| | | 4 | 电芯 9 电流计数 | I4 |
| | | 8 | 电芯 10 电压计数 | I4 |
| | | 12 | 电芯 10 电流计数 | I4 |
| | | 16 | 电芯 11 电压计数 | I4 |
| | | 20 | 电芯 11 电流计数 | I4 |
| | | 24 | 电芯 12 电压计数 | I4 |
| | | 28 | 电芯 12 电流计数 | I4 |
| 0x0074 | DASTATUS4 | 0 | 电芯 13 电压计数 | I4 |
| | | 4 | 电芯 13 电流计数 | I4 |
| | | 8 | 电芯 14 电压计数 | I4 |
| | | 12 | 电芯 14 电流计数 | I4 |
| | | 16 | 电芯 15 电压计数 | I4 |
| | | 20 | 电芯 15 电流计数 | I4 |
| | | 24 | 电芯 16 电压计数 | I4 |
| | | 28 | 电芯 16 电流计数 | I4 |
| 0xF081 | READ_CAL1 | 0 | 校准数据计数器 | U2 |
| | | 2 | CC2 计数 | I4 |
| | | 6 | PACK 引脚 ADC 计数 | I2 |
| | | 8 | 栈顶 (TOS) ADC 计数 | I2 |
| | | 10 | LD 引脚 ADC 计数 | I2 |

表 2-2. 电芯电压校准设置

| 参数名称 | 物理起始地址 | 类型 | 最小值 | 最大值 | 默认值 | 单位 |
|---|---|---|---|---|---|---|
| 电芯 1 增益 | 0x9180 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 2 增益 | 0x9182 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 3 增益 | 0x9184 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 4 增益 | 0x9186 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 5 增益 | 0x9188 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 6 增益 | 0x918A | I2 | -32767 | 32767 | 12409 | - |
| 电芯 7 增益 | 0x918C | I2 | -32767 | 32767 | 12409 | - |
| 电芯 8 增益 | 0x918E | I2 | -32767 | 32767 | 12409 | - |
| 电芯 9 增益 | 0x9190 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 10 增益 | 0x9192 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 11 增益 | 0x9194 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 12 增益 | 0x9196 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 13 增益 | 0x9198 | I2 | -32767 | 32767 | 12409 | - |
| 电芯 14 增益 | 0x919A | I2 | -32767 | 32767 | 12409 | - |
| 电芯 15 增益 | 0x919C | I2 | -32767 | 32767 | 12409 | - |
| 电芯 16 增益 | 0x919E | I2 | -32767 | 32767 | 12409 | - |
| Vcell 偏移 | 0x91B0 | I2 | -32767 | 32767 | 0 | - |

## 2.3 电芯的电压偏移校准

### 2.3.1 电芯的电压偏移校准步骤

1. 使用通过电芯电压增益校准得到的测量值计算每个电芯的偏移。例如，对于电芯 1：

$$\textbf{\textit{Cell 1 Offset}} = \frac{(\text{Cell 1 Gain}) \times \left(\text{ADC\_Counts}_{\text{CELL1\_A}}\right)}{2^{24}} - V_{\text{CELL1\_A}}$$

2. 计算各个电芯偏移的平均电芯偏移。
3. 将新的***电芯偏移***值写入 RAM。
   - 进入 CONFIG_UPDATE 模式（子命令 0x0090）。
   - 将***电芯偏移***写入 RAM 位置 0x91B0。
   - 退出 CONFIG UPDATE 模式（子命令 0x0092）。

## 2.4 TOS（栈顶）、PACK 和 LD 引脚电压校准

TOS（栈顶）、PACK 引脚和 LD 引脚增益的校准可以通过施加一个或两个（为提高精度）精确直流电压来实现。默认情况下，TOS、PACK 和 LD 电压的单位以 cV (10mV LSB) 表示。

### 2.4.1 TOS/PACK/LD 电压校准步骤

以下步骤对于三个电压测量值中的每一个都是相同的。在本例中，显示了 PACK 引脚电压两点校准的步骤。

1. 禁用休眠模式（子命令 0x009A），以确保施加电压后电压计数快速更新。
2. 在 PACK+ 和 VSS 之间施加已知电压 $V_{PACK\_A}$。
3. 100ms 后，使用 READ_CAL1 子命令 0xF081 读取 PACK 引脚 ADC 计数 ($ADC\_Counts_{PACK\_A}$)。为了获得最佳精度，请读取多个读数并计算平均值。
4. 在 PACK+ 和 VS 之间施加第二个已知电压 $V_{PACK\_B}$，并读取 PACK 引脚 ADC 计数。($ADC\_Counts_{PACK\_B}$)。

5. 计算 Pack 增益：

$$\textbf{\textit{Pack Gain}} = \frac{2^{16} \times \left(V_{PACK\_B} - V_{PACK\_A}\right)}{\left(ADC\_Counts_{PACK\_B} - ADC\_Counts_{PACK\_A}\right)} \quad (2)$$

6. 将新的 ***Pack 增益*** 值写入 RAM。
   - 进入 CONFIG_UPDATE 模式（子命令 0x0090）。
   - 将 ***Pack 增益*** 写入 0x91A0。
   - 退出 CONFIG UPDATE 模式（子命令 0x0092）。
7. 重新检查 Pack 电压读数。如果读数不准确，重复步骤 1-5。

**表 2-3. TOS、PACK、LD 电压校准设置**

| 参数名称 | 物理起始地址 | 类型 | 最小值 | 最大值 | 默认值 | 单位 |
|---|---|---|---|---|---|---|
| Pack 增益 | 0x91A0 | I2 | 0 | 65535 | 35507 | - |
| TOS 增益 | 0x91A2 | I2 | 0 | 65535 | 35507 | - |
| LD 增益 | 0x91A4 | I2 | 0 | 65535 | 35507 | - |
| Vdiv 偏移 | 0x91B2 | I2 | -32767 | 32767 | 0 | userV |
| ADC 增益 | 0x91A6 | I2 | -32767 | 32767 | 4166 | - |

## 2.5 ADC 增益校准

一些器件引脚可以配置为通用 ADC 输入。如果没有引脚配置为 ADC 输入，则无需校准 ADC 增益。

校准 ADCIN 电压读数的步骤与校准电芯电压测量值的步骤相同。可使用命令 *0x0076 DASTATUS6()* 或 *0x0077 DASTATUS7()*，具体取决于将哪个引脚配置为 ADC 输入。

## 2.6 电流校准

库仑计数器 ADC 测量 SRP 和 SRN 引脚之间的差分电压，以计算系统电流。库仑计数器偏移和电流增益校准步骤如下所示。只需根据系统中的检测电阻值调整该值（***CC 增益*** = 7.4768/（Rsense，以 mOhm 为单位））。如果需要进一步校准，可以使用以下步骤。

### 2.6.1 电路板偏移校准步骤

1. 禁用休眠模式（子命令 0x009A），以确保施加电压后电压计数快速更新。
2. 施加 0mA 的已知电流 $I_{CAL}$，并确保没有电流流过连接在 SRP 和 SRN 引脚之间的检测电阻。
3. 100ms 后，使用 READ_CAL1 子命令 0xF081 读取 CC2 计数。虽然 CC2 计数读数为四个字节，但只需读取中间的两个字节（16 位值）。为了获得最佳精度，请读取多个读数并计算平均值。
4. 读取 RAM 位置 0x91C6 处的***库仑计数器偏移样本*** 设置。此参数定义为这一多次库仑计数器转换过程中累积的偏移误差计数的数量。默认值是 64。
5. 计算电路板偏移：

$$\textbf{\textit{Board Offset}} = (\text{CC2 Counts}) \times \textbf{\textit{Coulomb Counter Offset Samples}} \quad (3)$$

6. 将新的***电路板偏移*** 值写入 RAM。
   - 进入 CONFIG_UPDATE 模式（子命令 0x0090）。
   - 将***电路板偏移*** 写入 0x91C8。
   - 退出 CONFIG_UPDATE 模式（子命令 0x0092）。
7. 重新检查 CC2 计数的读数。如果读数不接近零，重复步骤 1-5。

### 2.6.2 CC 增益校准步骤

1. 禁用休眠模式（子命令 0x009A），以确保施加电压后电压计数快速更新。
2. 施加已知电流 $I_{CAL}$（通常为 1A 至 2A），并确保电流流过连接在 SRP 和 SRN 引脚之间的检测电阻。
3. 100ms 后，使用 READ_CAL1 子命令 0xF081 读取 CC2 计数。为了获得最佳精度，请读取多个读数并计算平均值。
4. 计算 CC 增益：

$$CC\ Gain = \frac{(I_B - I_A)}{(CC\_Counts_B - CC\_Counts_A)} \quad (4)$$

5. 计算*容量增益*的值。*容量增益*等于 ***CC 增益*** 乘以 298261.6178。
6. 将新的 ***CC 增益*** 值写入 RAM。CC 增益必须使用 32 位 IEEE-754 浮点格式进行编码。本文档后面将提供一个代码示例，其中包括对此格式的计算。
   - 进入 CONFIG_UPDATE 模式（子命令 0x0090）。
   - 将 ***CC 增益*** 写入 0x91A8。
   - 将*容量增益*写入 0x91AC。
   - 退出 CONFIG_UPDATE 模式（子命令 0x0092）。
7. 重新检查 Current() 读数。如果读数不准确，重复步骤 1-5。

**表 2-4. 电流校准参数**

| 参数名称 | 物理起始地址 | 类型 | 最小值 | 最大值 | 默认值 | 单位 |
|---|---|---|---|---|---|---|
| CC 增益 | 0x91A8 | F4 | 1.00E-01 | 10.00E+00 | 7.4768 | - |
| 容量增益 | 0x91AC | I2 | 2.98262E+04 | 4.193046E+06 | 2230042.463 | - |
| 库仑计数器偏移样本 | 0x91C6 | U2 | 0 | 65535 | 64 | - |
| 电路板偏移 | 0x91C8 | I2 | -32767 | 32767 | 0 | - |

## 2.7 温度校准

内部和外部温度测量值可以在量产线上进行校准，方法是存储一个偏移值（该偏移值被添加到计算的测量值中），然后进行报告。每个温度测量值的单独偏移量可存储在配置寄存器中。每个温度测量值和配置寄存器的命令如下面的表中所示。

**表 2-5. 温度测量命令**

| 命令地址 | 名称 | 类型 | 单位 | 说明 |
|---|---|---|---|---|
| 0x68 | 内部温度 | I2 | 0.1K | 内部芯片温度 |
| 0x6A | CFETOFF 温度 | I2 | 0.1K | CFETOFF 引脚热敏电阻 |
| 0x6C | DFETOFF 温度 | I2 | 0.1K | DFETOFF 引脚热敏电阻 |
| 0x6E | ALERT 温度 | I2 | 0.1K | ALERT 引脚热敏电阻 |
| 0x70 | TS1 温度 | I2 | 0.1K | TS1 引脚热敏电阻 |
| 0x72 | TS2 温度 | I2 | 0.1K | TS2 引脚热敏电阻 |
| 0x74 | TS3 温度 | I2 | 0.1K | TS3 引脚热敏电阻 |
| 0x76 | HDQ 温度 | I2 | 0.1K | HDQ 引脚热敏电阻 |
| 0x78 | DCHG 温度 | I2 | 0.1K | DCHG 引脚热敏电阻 |
| 0x7A | DDSG 温度 | I2 | 0.1K | DDSG 引脚热敏电阻 |

表 2-6. 温度校准设置

| 参数名称 | 物理起始地址 | 类型 | 最小值 | 最大值 | 默认值 | 单位 |
|---|---|---|---|---|---|---|
| 内部温度偏移 | 0x91CA | I1 | -128 | 127 | 0 | 0.1°C |
| CFETOFF 温度偏移 | 0x91CB | I1 | -128 | 127 | 0 | 0.1°C |
| DFETOFF 温度偏移 | 0x91CC | I1 | -128 | 127 | 0 | 0.1°C |
| ALERT 温度偏移 | 0x91CD | I1 | -128 | 127 | 0 | 0.1°C |
| TS1 温度偏移 | 0x91CE | I1 | -128 | 127 | 0 | 0.1°C |
| TS2 温度偏移 | 0x91CF | I1 | -128 | 127 | 0 | 0.1°C |
| TS3 温度偏移 | 0x91D0 | I1 | -128 | 127 | 0 | 0.1°C |
| HDQ 温度偏移 | 0x91D1 | I1 | -128 | 127 | 0 | 0.1°C |
| DCHG 温度偏移 | 0x91D2 | I1 | -128 | 127 | 0 | 0.1°C |
| DDSG 温度偏移 | 0x91D3 | I1 | -128 | 127 | 0 | 0.1°C |

### 2.7.1 温度校准步骤

对于每个启用的温度测量值，以下步骤相同。在此示例中，仅显示了内部温度偏移和 TS1 温度偏移的步骤。

1. 将已知温度 $TEMP_{CAL}$ 应用于器件以及与外部热敏电阻引脚相连的热敏电阻。
2. 使用温度测量命令读取温度。例如，使用命令 0x68 读取内部温度 (TINT_measured)，使用命令 0x70 读取 TS1 温度 (TS1_measured)。温度命令返回的值以 0.1K 为单位，因此应将其转换为摄氏度。
3. 为了获得最佳精度，请读取多个读数并计算平均值。
4. 如果先前已写入温度偏移，则将偏移写回其默认值，即温度校准设置。默认情况下，这些值设置为零。
5. 计算每个测量值的温度偏移。在下面的公式中，温度单位为 0.1K。例如，2981 表示 25C。
   - ***内部温度偏移*** = TEMPCAL - TINT_measured
   - ***TS1 温度偏移*** = TEMPCAL - TS1_measured
6. 将新的***内部温度偏移*** 和 ***TS1 温度偏移*** 值写入 RAM。
   - 进入 CONFIG_UPDATE 模式（子命令 0x0090）。
   - 将***内部温度偏移*** 写入 0x91CA。
   - 将 ***TS1 温度偏移*** 写入 0x91CE。
   - 退出 CONFIG UPDATE 模式（子命令 0x0092）。
7. 使用温度测量命令重新检查温度读数。如果读数不准确，重复步骤 1-6。

## 2.8 COV 和 CUV 校准

该器件具有校准 COV（电芯过压）和 CUV（电芯欠压）保护阈值的可选功能。这可用于提高阈值精度或设置介于可用预设阈值之间的阈值。

表 2-7. COV/CUV 校准子命令

| 子命令地址 | 名称 | 说明 |
|---|---|---|
| 0xF090 | CAL_CUV | 使用顶部电芯输入端校准 CUV，以设置 ***CUV 阈值覆盖***。仅在 CONFIG_UPDATE 模式下可用。 |
| 0xF091 | CAL_COV | 使用顶部电芯输入端校准 COV，以设置 ***COV 阈值覆盖***。仅在 CONFIG_UPDATE 模式下可用。 |

表 2-8. COV/CUV 校准设置

| 参数名称 | 物理起始地址 | 类型 | 最小值 | 最大值 | 默认值 | 单位 |
|---|---|---|---|---|---|---|
| CUV 阈值覆盖 | 0x91D4 | H2 | 0x0000 | 0xFFFF | 0xFFFF | 十六进制 |
| COV 阈值覆盖 | 0x91D6 | H2 | 0x0000 | 0xFFFF | 0xFFFF | 十六进制 |

### 2.8.1 COV 校准步骤

1. 进入 CONFIG_UPDATE 模式 -（子命令 0x0090）。
2. 在位于器件顶部的两个电芯引脚之间施加精确的外部电压（具有所需 COV 值）。（对于 BQ76952，为 VC16 和 VC15；对于 BQ76942，为 VC10 和 VC9）。

3. 发送 *CAL_COV()* - 子命令 0xF091。
4. 退出 CONFIG UPDATE 模式（子命令 0x0092）。

***COV 阈值覆盖*** 设置现在在 RAM 中的地址 0x91D6 处进行更新。

### 2.8.2 CUV 校准步骤

1. 进入 CONFIG_UPDATE 模式 -（子命令 0x0090）。
2. 在器件的顶部两个电芯引脚之间施加精确的外部电压（具有所需 CUV 值）。
3. 发送 *CAL_CUV()* - 子命令 0xF090。
4. 退出 CONFIG UPDATE 模式（子命令 0x0092）。

***CUV 阈值覆盖*** 设置现在在 RAM 中的地址 0x91D4 处进行更新。

## 2.9 校准代码示例

下面的 Python 代码显示了一个示例校准过程。此代码演示了对电芯增益、电芯偏移、TOS 增益、PACK 增益、LD 增益、电路板偏移、CC 增益、容量增益、内部温度偏移和 TS1 偏移的校准。还提供了此示例的输出以供参考。

### 2.9.1 代码示例

```
'''
/* BQ769x2 Example Program for Calibration
'''
import pywinusb
import bqcomm
import sys
import time
from time import sleep
import sets
import math

I2C_ADDR = 0x10  # BQ769x2 slave address
numCells = 10    # Set to 10 for BQ76942############################################################
## 检查 EV2400 是否已连接
#####################################################
try:
    a = bqcomm.Adapter()  # This will use the first found EV2400
except:
    print "No EV2400 Available"
    sys.exit(1)

#####################################################
## 定义命令函数
#####################################################
def I2C_Read(device_addr, reg_addr, length):
    '''
    Uses global I2C address and returns value read
    '''
    try:
        value = a.i2c_read_block(device_addr, reg_addr, length)
    except:
        print "Nack received"
        return
    return value

def I2C_Write(device_addr, reg_addr, block):
    '''
    Uses global I2C address
    '''
    try:
        a.i2c_write_block(device_addr, reg_addr, block)
    except:
        print "Nack received"
    return

def DataRAM_Read(addr, length):
    '''
    Write address location to 0x3E and read back from 0x40
    Used to read dataflssh and for subcommands
    '''
    addressBlock = [(addr%256), (addr/256)]
    I2C_Write(I2C_ADDR, 0x3E, addressBlock)
```

```
    value = I2C_Read(I2C_ADDR, 0x40,length)
    return value

def DataRAM_Write(addr, block):
    '''
    Write address location to 0x3E and Checksum,length to 0x60
    Used to write dataflssh
    Add 2 to length for Rev A0 of Maximo2
    '''
    addressBlock = [(addr%256), (addr/256)]
    wholeBlock = addressBlock + block
    I2C_Write(I2C_ADDR, 0x3E, wholeBlock)             # Write Data Block
    # Write Data Checksum and length to 0x60, required for RAM writes
    I2C_Write(I2C_ADDR, 0x60, [~sum(wholeBlock) & 0xff, len(wholeBlock)+2])
    return

def ReadCellVoltage(cell):
    '''
    Reads a specific cell voltage
    '''
    cmd_addr = 0x12 + (cell * 2)
    result = I2C_Read(I2C_ADDR, cmd_addr, 2)
    print "Cell", cell, " = ", (result[1]*256 + result[0]), " mV"
    return


def Dec2Flash(value):
    '''

    '''
    if value == 0:
        value += 0.0000001    #avoid log of zero
    if value < 0:
        bNegative = 1
        value *= -1
    else:
        bNegative = 0

    exponent = int( (math.log(value)/math.log(2)) )
    MSB = exponent + 127        #exponent bits
    mantissa = value / (2**exponent)
    mantissa = (mantissa - 1) / (2**-23)

    if (bNegative == 0):
        mantissa = int(mantissa) & 0x7fffff   #remove sign bit if number is positive

    result = hex(int(round(mantissa + MSB * 2**23)))
    print result
    return result


def Flash2Dec(value):
    '''

    '''
    exponent = exponent = 0xff & (value/(2**23))  #exponent is most significant byte after sign bit
    mantissa = value % (2**23)
    if (0x80000000 & value == 0):   #check if number is positive
        isPositive = 1
    else:
        isPositive = 0
    mantissa_f = 1.0
    mask = 0x400000
    for i in range(0,23):
        if ((mask >> i) & mantissa):
            mantissa_f += 2**(-1*(i+1))
    result = mantissa_f * 2**(exponent-127)
    if not(isPositive):
        result *= -1
    print result
    return result

########################################
#  主脚本开始
########################################

################ 电压校准 ####################

# In this example we will apply the same reference voltage to all cells at once
```

```
print "CELL VOLTAGE CALIBRATION\n"
print "Apply 2.5V to all cells.\n"
print "This step will also enable the FETs to calibrate Top-of-Stack, PACK, and LD voltages.\n"
print "Press enter when voltage is applied..."
keypress = raw_input("")

# Make sure FETs are closed for PACK and LD measurements
I2C_Write(I2C_ADDR, 0x3E, [0x9A, 0x00]) #Sleep Disable 0x009A to prevent CHG FET from opening
status = I2C_Read(I2C_ADDR, 0x7F, 2) #Check FET Status
print "FET Status = ", hex(status[0])
if ((status[0] & 5) == 0):
    I2C_Write(I2C_ADDR, 0x3E, [0x22, 0x00]) #FET_ENABLE command 0x0022
sleep(0.5)

Cell_Voltage_Counts_A = [0]*numCells
Cell_Voltage_Counts_B = [0]*numCells
Cell_Gain = [0]*numCells
TOS_Voltage_Counts_A = 0
PACK_Voltage_Counts_A = 0
LD_Voltage_Counts_A = 0
TOS_Voltage_Counts_B = 0
PACK_Voltage_Counts_B = 0
LD_Voltage_Counts_B = 0

for i in range(0,10):
    sleep(0.1)
    DAStatus1 = DataRAM_Read(0x0071,32)  # Read DAStatus1
    DAStatus2 = DataRAM_Read(0x0072,32)  # Read DAStatus2
    DAStatus3 = DataRAM_Read(0x0073,32)  # Read DAStatus3
    READ_CAL1 = DataRAM_Read(0xF081,12)  # Read READ_CAL1    Cell_Voltage_Counts_A[0] +=
DAStatus1[0] + DAStatus1[1]*256 + DAStatus1[2]*256**2
    Cell_Voltage_Counts_A[1] += DAStatus1[8] + DAStatus1[9]*256 + DAStatus1[10]*256**2
    Cell_Voltage_Counts_A[2] += DAStatus1[16] + DAStatus1[17]*256 + DAStatus1[18]*256**2
    Cell_Voltage_Counts_A[3] += DAStatus1[24] + DAStatus1[25]*256 + DAStatus1[26]*256**2
    Cell_Voltage_Counts_A[4] += DAStatus2[0] + DAStatus2[1]*256 + DAStatus2[2]*256**2
    Cell_Voltage_Counts_A[5] += DAStatus2[8] + DAStatus2[9]*256 + DAStatus2[10]*256**2
    Cell_Voltage_Counts_A[6] += DAStatus2[16] + DAStatus2[17]*256 + DAStatus2[18]*256**2
    Cell_Voltage_Counts_A[7] += DAStatus2[24] + DAStatus2[25]*256 + DAStatus2[26]*256**2
    Cell_Voltage_Counts_A[8] += DAStatus3[0] + DAStatus3[1]*256 + DAStatus3[2]*256**2
    Cell_Voltage_Counts_A[9] += DAStatus3[8] + DAStatus3[9]*256 + DAStatus3[10]*256**2
    TOS_Voltage_Counts_A += READ_CAL1[8] + READ_CAL1[9]*256
    PACK_Voltage_Counts_A += READ_CAL1[6] + READ_CAL1[7]*256
    LD_Voltage_Counts_A += READ_CAL1[10] + READ_CAL1[11]*256print "Apply 4.20V to all cells.Press
enter when voltage is applied..."
keypress = raw_input("")

for i in range(0,10):
    sleep(0.1)
    DAStatus1 = DataRAM_Read(0x0071,32)  # Read DAStatus1
    DAStatus2 = DataRAM_Read(0x0072,32)  # Read DAStatus2
    DAStatus3 = DataRAM_Read(0x0073,32)  # Read DAStatus3
    READ_CAL1 = DataRAM_Read(0xF081,12)  # Read READ_CAL1

    Cell_Voltage_Counts_B[0] += DAStatus1[0] + DAStatus1[1]*256 + DAStatus1[2]*256**2
    Cell_Voltage_Counts_B[1] += DAStatus1[8] + DAStatus1[9]*256 + DAStatus1[10]*256**2
    Cell_Voltage_Counts_B[2] += DAStatus1[16] + DAStatus1[17]*256 + DAStatus1[18]*256**2
    Cell_Voltage_Counts_B[3] += DAStatus1[24] + DAStatus1[25]*256 + DAStatus1[26]*256**2
    Cell_Voltage_Counts_B[4] += DAStatus2[0] + DAStatus2[1]*256 + DAStatus2[2]*256**2
    Cell_Voltage_Counts_B[5] += DAStatus2[8] + DAStatus2[9]*256 + DAStatus2[10]*256**2
    Cell_Voltage_Counts_B[6] += DAStatus2[16] + DAStatus2[17]*256 + DAStatus2[18]*256**2
    Cell_Voltage_Counts_B[7] += DAStatus2[24] + DAStatus2[25]*256 + DAStatus2[26]*256**2
    Cell_Voltage_Counts_B[8] += DAStatus3[0] + DAStatus3[1]*256 + DAStatus3[2]*256**2
    Cell_Voltage_Counts_B[9] += DAStatus3[8] + DAStatus3[9]*256 + DAStatus3[10]*256**2
    TOS_Voltage_Counts_B += READ_CAL1[8] + READ_CAL1[9]*256
    PACK_Voltage_Counts_B += READ_CAL1[6] + READ_CAL1[7]*256
    LD_Voltage_Counts_B += READ_CAL1[10] + READ_CAL1[11]*256

#Take the average of the 10 measurements and calculate gains
for i in range(0,numCells):
    Gain = 2**24 * (4200 - 2500) / (Cell_Voltage_Counts_B[i]/10 - Cell_Voltage_Counts_A[i]/10)
    Cell_Gain[i] = int(round(Gain))
    print "Cell ",i+1," Gain = ", Cell_Gain[i]

#Calculate Cell Offset based on Cell1
Cell_Offset = ((Cell_Gain[0] * (Cell_Voltage_Counts_A[0] / 10)) / 2**24) - 2500
print "Cell Offset = ", Cell_Offset
if Cell_Offset < 0:
    Cell_Offset = 0xFFFF + Cell_Offset
```

```
TOS_Gain = int(round(2**16 * (4200 - 2500) / (TOS_Voltage_Counts_B/10 - TOS_Voltage_Counts_A/10)))
PACK_Gain = int(round(2**16 * (4200 - 2500) / (PACK_Voltage_Counts_B/10 - PACK_Voltage_Counts_A/
10)))
LD_Gain = int(round(2**16 * (4200 - 2500) / (LD_Voltage_Counts_B/10 - LD_Voltage_Counts_A/10)))
print "TOS Gain = ", TOS_Gain
print "PACK Gain = ", PACK_Gain
print "LD Gain = ", LD_Gain################ 电流校准 ####################
print "Current Calibration\n"
print "Apply 0mA through sense resistor for Board Offset Calibration.\n"
print "Press enter when current is applied..."
keypress = raw_input("")

value = 0

for i in range(0,10):
    sleep(0.1)
    READ_CAL1 = DataRAM_Read(0xF081,12)  # Read READ_CAL1
    value += READ_CAL1[3] + READ_CAL1[4]*256

value = 64 * int(round(value/10)) #take the average
Board_Offset = -(value & 0x8000) | (value & 0x7fff) #Get decimal value for printing

print "Board Offset = ", Board_Offset
if Board_Offset < 0:
    Board_Offset = 0xFFFF + Board_Offset

print "Apply 1A (discharge current) through sense resistor for Board Offset Calibration.\n"
print "Press enter when current is applied..."
keypress = raw_input("")

value = 0
for i in range(0,10):
    sleep(0.1)
    READ_CAL1 = DataRAM_Read(0xF081,12)  # Read READ_CAL1
    value += READ_CAL1[3] + READ_CAL1[4]*256
value = int(round(value/10)) #take the average
CC_Counts_A = -(value & 0x8000) | (value & 0x7fff) #Get decimal value for printing
print "CC_Counts_A = ", CC_Counts_A

print "Apply 2A (discharge current) through sense resistor for Board Offset Calibration.\n"
print "Press enter when current is applied..."
keypress = raw_input("")

value = 0
for i in range(0,10):
    sleep(0.1)
    READ_CAL1 = DataRAM_Read(0xF081,12)  # Read READ_CAL1
    value += READ_CAL1[3] + READ_CAL1[4]*256

value = int(round(value/10)) #take the average
CC_Counts_B = -(value & 0x8000) | (value & 0x7fff) #Get decimal value for printing
print "CC_Counts_B = ", CC_Counts_B

CC_Gain_float = (-2000.0 + 1000.0)/(CC_Counts_B - CC_Counts_A)
CC_Gain = Dec2Flash(CC_Gain_float)
print "CC_Gain = ", CC_Gain_float
Capacity_Gain = Dec2Flash(298261.6178 * CC_Gain_float)
print "Capacity Gain = ", Capacity_Gain

################ 温度校准 ####################
print "Temperature Calibration\n"
print "Set the device temperature to 25C.(~298.1K)"
print "This example will calibrate TS1 and the Internal Temperature.\n"
print "Press enter when temperature is applied..."
keypress = raw_input("")

Int_Temp = I2C_Read(I2C_ADDR, 0x68,2) #Read Internal Temp
TS1_Temp = I2C_Read(I2C_ADDR, 0x70,2) #Read TS1 Temp
Internal_Temp_Offset = 2981 - (Int_Temp[1]*256 + Int_Temp[0])
if Internal_Temp_Offset < 0:
    Internal_Temp_Offset = 0xFFFF + Internal_Temp_Offset
print "Internal Temp Offset = ", Internal_Temp_Offset
TS1_Offset = 2981 - (TS1_Temp[1]*256 + TS1_Temp[0])
if TS1_Offset < 0:
    TS1_Offset = 0xFFFF + TS1_Offset
print "TS1 Offset = ", TS1_Offset

################ COV/CUV 校准####################
print "COV Calibration\n"
```

```
print "Apply the desired value for the cell over-voltage threshold to device cell inputs.\n"
print "Calibration will use the voltage applied to the top cell of the device.\n"
print "For example, Apply 4350mV\n"
print "Press enter when voltage is applied..."
keypress = raw_input("")

I2C_Write(I2C_ADDR, 0x3E, [0x90, 0x00]) #Enter CONFIG_UPDATE Mode
I2C_Write(I2C_ADDR, 0x3E, [0x91, 0xF0]) #Execute CAL_COV() 0xF091
I2C_Write(I2C_ADDR, 0x3E, [0x92, 0x00]) #Exit CONFIG_UPDATE Mode

print "CUV Calibration\n"
print "Apply the desired value for the cell under-voltage threshold to device cell inputs.\n"
print "Calibration will use the voltage applied to the top cell of the device.\n"
print "For example, Apply 2400mV\n"
print "Press enter when voltage is applied..."
keypress = raw_input("")

I2C_Write(I2C_ADDR, 0x3E, [0x90, 0x00]) #Enter CONFIG_UPDATE Mode
I2C_Write(I2C_ADDR, 0x3E, [0x90, 0xF0]) #Execute CAL_COV() 0xF091
I2C_Write(I2C_ADDR, 0x3E, [0x92, 0x00]) #Exit CONFIG_UPDATE Mode

################ 将校准数据写入 RAM ####################
print "Writing Calibration to Data RAM\n"

I2C_Write(I2C_ADDR, 0x3E, [0x90, 0x00]) #Enter CONFIG_UPDATE Mode
#Cell Voltage Gains
DataRAM_Write(0x9180, [(Cell_Gain[0]%256),(Cell_Gain[0]/256)] )
DataRAM_Write(0x9182, [(Cell_Gain[1]%256),(Cell_Gain[1]/256)] )
DataRAM_Write(0x9184, [(Cell_Gain[2]%256),(Cell_Gain[2]/256)] )
DataRAM_Write(0x9186, [(Cell_Gain[3]%256),(Cell_Gain[3]/256)] )
DataRAM_Write(0x9188, [(Cell_Gain[4]%256),(Cell_Gain[4]/256)] )
DataRAM_Write(0x918A, [(Cell_Gain[5]%256),(Cell_Gain[5]/256)] )
DataRAM_Write(0x918C, [(Cell_Gain[6]%256),(Cell_Gain[6]/256)] )
DataRAM_Write(0x918E, [(Cell_Gain[7]%256),(Cell_Gain[7]/256)] )
DataRAM_Write(0x9190, [(Cell_Gain[8]%256),(Cell_Gain[8]/256)] )
DataRAM_Write(0x9192, [(Cell_Gain[9]%256),(Cell_Gain[9]/256)] )
DataRAM_Write(0x91B0, [(Cell_Offset%256),(Cell_Offset/256)] )
#PACK, LD, TOS Gains
DataRAM_Write(0x91A0, [(PACK_Gain%256),(PACK_Gain/256)] )
DataRAM_Write(0x91A2, [(TOS_Gain%256),(TOS_Gain/256)] )
DataRAM_Write(0x91A4, [(LD_Gain%256),(LD_Gain/256)] )
#CC Offset, CC Gain, Capacity Gain
DataRAM_Write(0x91C8, [(Board_Offset%256),(Board_Offset/256)])
DataRAM_Write(0x91A8,
[int(CC_Gain[8:10],16),int(CC_Gain[6:8],16),int(CC_Gain[4:6],16),int(CC_Gain[2:4],16)])
DataRAM_Write(0x91AC,
[int(Capacity_Gain[8:10],16),int(Capacity_Gain[6:8],16),int(Capacity_Gain[4:6],16),int(Capacity_Gai
n[2:4],16)])
#Temperature Offsets
DataRAM_Write(0x91CA, [(Internal_Temp_Offset%256),(Internal_Temp_Offset/256)])
DataRAM_Write(0x91CE, [(TS1_Offset%256),(TS1_Offset/256)])

I2C_Write(I2C_ADDR, 0x3E, [0x92, 0x00]) #Exit CONFIG_UPDATE Mode
print("End of calibration")

# Close the EV2400
a.close()
```

### 2.9.2 代码输出

```
CELL VOLTAGE CALIBRATION

Apply 2.5V to all cells.

This step will also enable the FETs to calibrate Top-of-Stack, PACK, and LD voltages.

Press enter when voltage is applied...

FET Status =  0x30
Apply 4.20V to all cells.Press enter when voltage is applied...

Cell  1  Gain =  12170
Cell  2  Gain =  12122
Cell  3  Gain =  12142
Cell  4  Gain =  12141
Cell  5  Gain =  12125
Cell  6  Gain =  12143
```

```
Cell  7  Gain =  12114
Cell  8  Gain =  12138
Cell  9  Gain =  12110
Cell  10  Gain =  12118
Cell Offset =  0
TOS Gain =  33977
PACK Gain =  34783
LD Gain =  33598
Current Calibration

Apply 0mA through sense resistor for Board Offset Calibration.

Press enter when current is applied...

Board Offset =  -64
Apply 1A (discharge current) through sense resistor for Board Offset Calibration.

Press enter when current is applied...

CC_Counts_A =  -130
Apply 2A (discharge current) through sense resistor for Board Offset Calibration.

Press enter when current is applied...

CC_Counts_B =  -258
0x40fa0000
CC_Gain =  7.8125
0x4a0e38e3
Capacity Gain =  0x4a0e38e3
Temperature Calibration

Set the device temperature to 25C.(~298.1K)
This example will calibrate TS1 and the Internal Temperature.

Press enter when temperature is applied...

Internal Temp Offset =  65534
TS1 Offset =  65509
COV Calibration

Apply the desired value for the cell over-voltage threshold to device cell inputs.

Calibration will use the voltage applied to the top cell of the device.

For example, Apply 4350mV

Press enter when voltage is applied...

CUV Calibration

Apply the desired value for the cell under-voltage threshold to device cell inputs.

Calibration will use the voltage applied to the top cell of the device.

For example, Apply 2400mV

Press enter when voltage is applied...

Writing Calibration to Data RAM

End of calibration
```

# 3 OTP 编程

配置所有器件设置并完成校准后，可以将设置长久写入器件 OTP。理想情况下，OTP 应只写入一次，但有一定的灵活性：

- OTP 存储器包括数据存储器配置设置的两个完整映像。这意味着能够更改设置的默认值，并将其更改回默认值一次，然后再也无法修改该设置。更实际地说，此特性可用于在各个部分中编写 OTP。例如，主设置可以写入 OTP，在以后的量产步骤中，校准设置可以写入 OTP。
- 然而，对 OTP 的部分更改并非不受限制。这是因为每次写入 OTP 时，都会计算 OTP 签名并将其存储在 OTP 中。该器件最多支持 8 个不同的签名值，因此 OTP 部分写入的最大次数为 8。

### 3.1 在量产中写入 OTP 的建议步骤

为了写入 OTP，应向 BAT 引脚施加 10 至 12V 之间的电压，并且器件必须处于 FULLACCESS（完全访问）模式。下面列出了写入 OTP 的建议步骤。

1. 通过读取其中一个已编程寄存器，检查器件上是否已完成 OTP 编程。通电时，寄存器将报告默认值或 OTP 中编程的值（如果 OTP 已编程）。如果尚未完成 OTP 编程，则转至后续步骤。
2. 读取 *0x12 电池状态[SEC1,SEC0]* 位，以验证器件是否处于 FULL ACCESS（完全访问）模式 (0x01)。
3. 如果器件处于 FULL ACCESS（完全访问）模式，则进入 CONFIG_UPDATE 模式-（子命令 0x0090）。
4. 在数据存储器中配置寄存器设置。
5. 退出 CONFIG_UPDATE 模式 -（子命令 0x0092）。
6. 读取数据存储器寄存器以验证所有参数均已成功写入。
7. 进入 CONFIG_UPDATE 模式。
8. 检查*电池状态[OTPB]* 位是否清除，以验证是否满足 OTP 编程条件。
9. 读取 OTP_WR_CHECK()（子命令 0x00A0）。如果返回值为 0x80，则满足 OTP 编程条件。
10. 如果 OTP_WR_CHECK 指示满足条件，则发送 OTP_WRITE() 子命令 (0x00A1)。
11. 等待 100 ms。从 0x40 读取以检查 OTP 编程是否成功（0x80 表示成功）。
12. 退出 CONFIG_UPDATE 模式 -（子命令 0x0092）。

## 4 参考文献


- 德州仪器 (TI)，*BQ76952 3-16 芯串联电池监测器和保护器数据表*
- 德州仪器 (TI)，*BQ76942 3-16 芯串联电池监测器和保护器数据表*
- 德州仪器 (TI)，*BQ76952 技术参考手册*
- 德州仪器 (TI)，*BQ76942 技术参考手册*
- 德州仪器 (TI)，BQ76952 评估模块用户指南
- 德州仪器 (TI)，BQ76942 评估模块用户指南
- 德州仪器 (TI)，BQ76942、BQ76952 软件开发指南

# 5 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

| Changes from Revision * (October 2020) to Revision A (August 2021) | Page |
|---|---|
| • 更新了本文档的“摘要” | 1 |
| • 更新了整个文档中的表格、图和交叉参考的编号格式。 | 2 |
| • 对节 3 进行了更新。 | 13 |

# 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/legal/termsofsale.html) 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。

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

# 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