Programmer's Guide

BQ769x2 Calibration and OTP Programming Guide

Matt Sunna

ABSTRACT

The BQ769x2 battery monitor family (which includes the BQ76952, BQ76942, and BQ769142) can be configured by the host in the application or OTP is available for those who want to permanently configure device parameters on their production line. The BQ769x2 family also includes the optional capability for you to calibrate several key parameters for improved system accuracy. This application report provides the steps needed for the calibration and the OTP programming process.

Table of Contents

List of Tables

Trademarks

1 Production Programming of BQ769x2 Device Family

While the BQ769x2 can be configured by the host in the application, there are some advantages to configuring the device on the production line using OTP. First, pre-configuring the OTP allows you to configure important device settings that may be difficult to configure in the application. For example, if the application host relies on the BQ769x2 LDO for power or needs to select a different communication type than the default selection, it may be more practical to have these settings pre-configured in OTP so that the device powers up with the desired settings. To see which available communication type and LDO configurations are available to order pre-configured, see the device-selection table in the device-specific data sheet.

Another advantage of configuring the device on the production line is it allows the opportunity to calibrate voltage, current, and temperature measurements for the best possible accuracy. Calibration is only practical on the production line where applied voltages, current, and temperature can be tightly controlled.

2 Calibration

The BQ769x2 has commands that support raw ADC readings to support calibration of voltage, current, and temperature readings. The following sections will outline steps required for the calibration of different measurement types, the commands and subcommands needed for each of the readings, and the RAM parameters where calibration data can be stored.

2.1 Calibration Accuracy

There are a few options to consider when implementing calibration in production depending on the needed accuracy and production test equipment capability and test time. Let's take cell voltage calibration as an example. Below are a few different options listed ranging from least accurate (1) to most accurate (4).

1. No production calibration, use default values.
2. Perform calibration on a fixed number of devices and use the average values of the calibration parameters for all packs. This might be a good option if the variation observed from board to board is small and acceptable.
3. Calibrate each device individually in production by applying a single accurate voltage applied to the cell input, such as 3.5 V. Averaging the measurement over several readings can further improve accuracy.
4. Calibrate each device individually in production by applying two precise DC voltages to the cell input, such as 2.5 V and 4.5 V. Averaging each of the measurements over several readings can further improve accuracy. This will not only allow a more accurate calculation of cell gain, but also enables calibration of cell voltage offset.

2.2 Cell Voltage Gain Calibration

2.2.1 Cell Voltage Gain Calibration Steps

The below steps are the same for each of the cells. In this example, the steps are shown for a two-point calibration of Cell 1.

1. Disable Sleep Mode (Subcommand 0x009A) to ensure voltage counts update quickly after applying a voltage.
2. Apply a known voltage, V_{CELL1_A}, between device pins VC1 and VC0.
3. After 100ms, read the Cell 1 Voltage Counts ($ADC\_Counts_{CELL1\_A}$) using the DASTATUS1 subcommand 0x0071. A full list of commands for calibration voltage readings is provided in Table 2-1. For best accuracy, take multiple readings and calculate the average.
4. Apply a second known voltage, V_{CELL1_B}, between VC1 and VC0 and read the Cell 1 Voltage Counts. ($ADC\_Counts_{CELL1\_B}$).
5. Calculate the Cell 1 Gain:

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

6. Write the new ***Cell 1 Gain*** value to RAM.
   - Enter CONFIG_UPDATE mode (Subcommand 0x0090).
   - Write ***Cell 1 Gain*** to RAM location 0x9180.
   - Exit CONFIG UPDATE mode (Subcommand 0x0092).
7. Re-check the Cell 1 voltage reading. If the reading is not accurate, repeat steps 1-6.

Table 2-1. Voltage and Current Measurement Subcommands for Calibration

Subcommand Address	Name	Offset	Data	Type
0x0071	DASTATUS1	0	Cell 1 Voltage Counts	I4
		4	Cell 1 Current Counts	I4
		8	Cell 2 Voltage Counts	I4
		12	Cell 2 Current Counts	I4
		16	Cell 3 Voltage Counts	I4
		20	Cell 3 Current Counts	I4
		24	Cell 4 Voltage Counts	I4
		28	Cell 4 Current Counts	I4
0x0072	DASTATUS2	0	Cell 5 Voltage Counts	I4
		4	Cell 5 Current Counts	I4
		8	Cell 6 Voltage Counts	I4
		12	Cell 6 Current Counts	I4
		16	Cell 7 Voltage Counts	I4
		20	Cell 7 Current Counts	I4
		24	Cell 8 Voltage Counts	I4
		28	Cell 8 Current Counts	I4
0x0073	DASTATUS3	0	Cell 9 Voltage Counts	I4
		4	Cell 9 Current Counts	I4
		8	Cell 10 Voltage Counts	I4
		12	Cell 10 Current Counts	I4
		16	Cell 11 Voltage Counts	I4
		20	Cell 11 Current Counts	I4
		24	Cell 12 Voltage Counts	I4
		28	Cell 12 Current Counts	I4
0x0074	DASTATUS4	0	Cell 13 Voltage Counts	I4
		4	Cell13 Current Counts	I4
		8	Cell 14 Voltage Counts	I4
		12	Cell 14 Current Counts	I4
		16	Cell 15 Voltage Counts	I4
		20	Cell 15 Current Counts	I4
		24	Cell 16 Voltage Counts	I4
		28	Cell 16 Current Counts	I4
0xF081	READ_CAL1	0	Calibration Data Counter	U2
		2	CC2 Counts	I4
		6	PACK pin ADC Counts	I2
		8	Top of Stack (TOS) ADC Counts	I2
		10	LD pin ADC Counts	I2

Table 2-2. Cell Voltage Calibration Settings

Parameter Name	Physical Start Address	Type	Min	Max	Default	Units
Cell 1 Gain	0x9180	I2	-32767	32767	12409	-
Cell 2 Gain	0x9182	I2	-32767	32767	12409	-
Cell 3 Gain	0x9184	I2	-32767	32767	12409	-
Cell 4 Gain	0x9186	I2	-32767	32767	12409	-
Cell 5 Gain	0x9188	I2	-32767	32767	12409	-
Cell 6 Gain	0x918A	I2	-32767	32767	12409	-
Cell 7 Gain	0x918C	I2	-32767	32767	12409	-
Cell 8 Gain	0x918E	I2	-32767	32767	12409	-
Cell 9 Gain	0x9190	I2	-32767	32767	12409	-
Cell 10 Gain	0x9192	I2	-32767	32767	12409	-
Cell 11 Gain	0x9194	I2	-32767	32767	12409	-
Cell 12 Gain	0x9196	I2	-32767	32767	12409	-
Cell 13 Gain	0x9198	I2	-32767	32767	12409	-
Cell 14 Gain	0x919A	I2	-32767	32767	12409	-
Cell 15 Gain	0x919C	I2	-32767	32767	12409	-
Cell 16 Gain	0x919E	I2	-32767	32767	12409	-
Vcell Offset	0x91B0	I2	-32767	32767	0	-

2.3 Cell Voltage Offset Calibration

2.3.1 Cell Voltage Offset Calibration Steps

1. Calculate the offset of each cell using the meausrements taken for cell voltage gain calibration. For example, for Cell 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. Calculate the average cell offset from the individual cell offsets.
3. Write the new ***Cell Offset*** value to RAM.
   - Enter CONFIG_UPDATE mode (Subcommand 0x0090).
   - Write ***Cell Offset*** to RAM location 0x91B0.
   - Exit CONFIG UPDATE mode (Subcommand 0x0092).

2.4 TOS (Top-of-Stack), PACK, and LD Pin Voltage Calibration

The calibration of the TOS (top-of-stack), PACK pin, and LD pin gain can be similarly implemented by applying one or two (for higher accuracy) precise DC voltages. The units for TOS, PACK, and LD voltages are reported in cV (10mV LSB) by default.

2.4.1 TOS / PACK / LD Voltage Calibration Steps

The below steps are the same for each of the three voltage measurements. In this example, the steps are shown for a two-point calibration of the PACK pin voltage.

1. Disable Sleep Mode (Subcommand 0x009A) to ensure voltage counts update quickly after applying a voltage.
2. Apply a known voltage, V_{PACK_A}, between PACK+ and VSS.
3. After 100ms, read the PACK Pin ADC Counts ($ADC\_Counts_{PACK\_A}$) using the READ_CAL1 subcommand 0xF081. For best accuracy, take multiple readings and calculate the average.
4. Apply a second known voltage, V_{PACK_B}, between PACK+ and VS and read the PACK Pin ADC Counts. ($ADC\_Counts_{PACK\_B}$).

5. Calculate the Pack Gain:

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

6. Write the new ***Pack Gain*** value to RAM.
   - Enter CONFIG_UPDATE mode (Subcommand 0x0090).
   - Write ***Pack Gain*** to 0x91A0.
   - Exit CONFIG UPDATE mode (Subcommand 0x0092).
7. Re-check the Pack voltage reading. If the reading is not accurate, repeat steps 1-5.

Table 2-3. TOS, PACK, LD Voltage Calibration Settings

Parameter Name	Physical Start Address	Type	Min	Max	Default	Units
Pack Gain	0x91A0	I2	0	65535	35507	-
TOS Gain	0x91A2	I2	0	65535	35507	-
LD Gain	0x91A4	I2	0	65535	35507	-
Vdiv Offset	0x91B2	I2	-32767	32767	0	userV
ADC Gain	0x91A6	I2	-32767	32767	4166	-

2.5 ADC Gain Calibration

Several device pins can be configured as a general purpose ADC input. If none of the pins are configured as an ADC input, then it is not necessary to calibrate the ADC Gain.

The steps for calibrating ADCIN voltage readings is the same as for calibrating cell voltage measurements. The commands *0x0076 DASTATUS6()* or *0x0077 DASTATUS7()* can be used depending on which pin is configured as an ADC input.

2.6 Current Calibration

The coulomb counter ADC measures the differential voltage between the SRP and SRN pins to calculate the system current. The coulomb counter offset and current gain calibration steps are shown below. This value can simply be adjusted based on the sense resistor value in the system (***CC Gain*** = 7.4768 / (Rsense in mOhm)). The steps below can be used if further calibration is required.

2.6.1 Board Offset Calibration Steps

1. Disable Sleep Mode (Subcommand 0x009A) to ensure voltage counts update quickly after applying a voltage.
2. Apply a known current I_{CAL} of 0 mA and ensure no current is flowing through the sense resistor connected between the SRP and SRN pins.
3. After 100ms, read CC2 Counts using the READ_CAL1 subcommand 0xF081. While the CC2 Counts reading is four bytes, only the middle two bytes need to be read (16-bit value). For best accuracy, take multiple readings and calculate the average.
4. Read the ***Coulomb Counter Offset Samples*** settings at RAM location 0x91C6. This parameter is defined as the number of counts of offset error that would accumulate over this many coulomb counter conversions. The default value is 64.
5. Calculate the Board Offset:

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

6. Write the new ***Board Offset*** value to RAM.
   - Enter CONFIG_UPDATE mode (Subcommand 0x0090).
   - Write ***Board Offset*** to 0x91C8.
   - Exit CONFIG_UPDATE mode (Subcommand 0x0092).
7. Re-check the CC2 Counts reading. If the reading is not close to zero, repeat steps 1-5.

2.6.2 CC Gain Calibration Steps

1. Disable Sleep Mode (Subcommand 0x009A) to ensure voltage counts update quickly after applying a voltage.
2. Apply a known current I_{CAL} (typically 1 A to 2 A) and ensure current is flowing through the sense resistor connected between the SRP and SRN pins.
3. After 100ms, read CC2 Counts using the READ_CAL1 subcommand 0xF081. For best accuracy, take multiple readings and calculate the average.
4. Calculate the CC Gain:

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

5. Calculate the value for ***Capacity Gain***. ***Capacity Gain*** is simply the ***CC Gain*** multiplied by 298261.6178.
6. Write the new ***CC Gain*** value to RAM. CC Gain must be encoded using the 32-bit IEEE-754 floating point format. A code example is provided later in this document that includes computing this format.
   - Enter CONFIG_UPDATE mode (Subcommand 0x0090).
   - Write ***CC Gain*** to 0x91A8.
   - Write ***Capacity Gain*** to 0x91AC.
   - Exit CONFIG_UPDATE mode (Subcommand 0x0092).
7. Re-check the Current() reading. If the reading is not accurate, repeat steps 1-5.

Table 2-4. Current Calibration Parameters

Parameter Name	Physical Start Address	Type	Min	Max	Default	Units
CC Gain	0x91A8	F4	1.00E-01	10.00E+00	7.4768	-
Capacity Gain	0x91AC	I2	2.98262E+04	4.193046E+06	2230042.463	-
Coulomb Counter Offset Samples	0x91C6	U2	0	65535	64	-
Board Offset	0x91C8	I2	-32767	32767	0	-

2.7 Temperature Calibration

Internal and external temperature measurements can be calibrated on the production line by storing an offset value which is added to the calculated measurement before reporting. A separate offset for each temperature measurement can be stored in the configuration registers. The command for each temperature measurement and the configuration registers are shown in the tables below.

Table 2-5. Temperature Measurement Commands

Command Address	Name	Type	Units	Description
0x68	Int Temperature	I2	0.1K	Internal die temperature
0x6A	CFETOFF Temperature	I2	0.1K	CFETOFF pin thermistor
0x6C	DFETOFF Temperature	I2	0.1K	DFETOFF pin thermistor
0x6E	ALERT Temperature	I2	0.1K	ALERT pin thermistor
0x70	TS1 Temperature	I2	0.1K	TS1 pin thermistor
0x72	TS2 Temperature	I2	0.1K	TS2 pin thermistor
0x74	TS3 Temperature	I2	0.1K	TS3 pin thermistor
0x76	HDQ Temperature	I2	0.1K	HDQ pin thermistor
0x78	DCHG Temperature	I2	0.1K	DCHG pin thermistor
0x7A	DDSG Temperature	I2	0.1K	DDSG pin thermistor

Table 2-6. Temperature Calibration Settings

Parameter Name	Physical Start Address	Type	Min	Max	Default	Units
Internal Temp Offset	0x91CA	I1	-128	127	0	0.1°C
CFETOFF Temp Offset	0x91CB	I1	-128	127	0	0.1°C
DFETOFF Temp Offset	0x91CC	I1	-128	127	0	0.1°C
ALERT Temp Offset	0x91CD	I1	-128	127	0	0.1°C
TS1 Temp Offset	0x91CE	I1	-128	127	0	0.1°C
TS2 Temp Offset	0x91CF	I1	-128	127	0	0.1°C
TS3 Temp Offset	0x91D0	I1	-128	127	0	0.1°C
HDQ Temp Offset	0x91D1	I1	-128	127	0	0.1°C
DCHG Temp Offset	0x91D2	I1	-128	127	0	0.1°C
DDSG Temp Offset	0x91D3	I1	-128	127	0	0.1°C

2.7.1 Temperature Calibration Steps

The below steps are the same for each of the enabled temperature measurements. In this example, the steps are shown for the Internal Temp Offset and TS1 Temp Offset only.

1. Apply a known temperature, $TEMP_{CAL}$, to the device and to thermistors connected to external thermistor pins.
2. Read the temperatures using the temperature measurement commands. For example, read Internal temperature (TINT_measured) with command 0x68, TS1 temperature (TS1_measured) with command 0x70. The values returned by the temperature commands are in 0.1K units, so these should be converted to Celsius.
3. For best accuracy, take multiple readings and calculate the average.
4. If temperature offsets have been previously written, write the offsets back to their default values the Temperature Calibration Settings. These values are set to zero by default.
5. Calculate the temperature offset for each of the measurements. In the equations below, the temperature units are in 0.1K. So for example, 2981 would represent 25C.
   - ***Internal Temp Offset*** = TEMPCAL - TINT_measured
   - ***TS1 Temp Offset*** = TEMPCAL - TS1_measured
6. Write the new ***Internal Temp Offset*** and ***TS1 Temp Offset*** values to RAM.
   - Enter CONFIG_UPDATE mode (Subcommand 0x0090).
   - Write ***Internal Temp Offset*** to 0x91CA.
   - Write ***TS1 Temp Offset*** to 0x91CE.
   - Exit CONFIG UPDATE mode (Subcommand 0x0092).
7. Re-check the temperature readings using the temperature measurement commands. If the reading is not accurate, repeat steps 1-6.

2.8 COV and CUV Calibration

The device includes the optional capability to calibrate the COV (cell overvoltage) and CUV (cell undervoltage) protection thresholds. This may be done to improve threshold accuracy or to set a threshold between the available preset threshold values.

Table 2-7. COV/CUV Calibration Subcommands

Subcommand Address	Name	Description
0xF090	CAL_CUV	Calibrates CUV using the top cell input to set ***CUV Threshold Override***. Only available in CONFIG_UPDATE mode.
0xF091	CAL_COV	Calibrates COV using the top cell input to set ***COV Threshold Override***. Only available in CONFIG_UPDATE mode.

Table 2-8. COV/CUV Calibration Settings

Parameter Name	Physical Start Address	Type	Min	Max	Default	Units
CUV Threshold Override	0x91D4	H2	0x0000	0xFFFF	0xFFFF	Hex
COV Threshold Override	0x91D6	H2	0x0000	0xFFFF	0xFFFF	Hex

2.8.1 COV Calibration Steps

1. Enter CONFIG_UPDATE mode - (Subcommand 0x0090).
2. Apply an accurate external voltage with the desired COV value between the top two cell pins of the device. (VC16 and VC15 for the BQ76952, VC10 and VC9 for BQ76942).
3. Send *CAL_COV()* - Subcommand 0xF091.
4. Exit CONFIG UPDATE mode (Subcommand 0x0092).

The ***COV Threshold Override*** setting is now updated in RAM at address 0x91D6.

2.8.2 CUV Calibration Steps

1. Enter CONFIG_UPDATE mode - (Subcommand 0x0090).
2. Apply an accurate external voltage with the desired CUV value between the top two cell pins of the device.
3. Send *CAL_CUV()* - Subcommand 0xF090.
4. Exit CONFIG UPDATE mode (Subcommand 0x0092).

The ***CUV Threshold Override*** setting is now updated in RAM at address 0x91D4.

2.9 Calibration Code Example

The following Python code shows an example calibration procedure. This code demonstrates calibration of the Cell Gains, Cell Offset, TOS Gain, PACK Gain, LD Gain, Board Offset, CC Gain, Capacity Gain, Internal Temp Offset, and TS1 Offset. The output of the example is also provided for reference.

2.9.1 Code Example

```
'''
/* 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


#####################################################
## Check to see if EV2400 is connected
#####################################################
try:
    a = bqcomm.Adapter()  # This will use the first found EV2400
except:
    print "No EV2400 Available"
    sys.exit(1)

#####################################################
## Define command functions
#####################################################
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

########################################
#  Start of Main Script
########################################

################ Voltage Calibration ###################

# 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]*256


print "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


################ Current Calibration ####################
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

################ Temperature Calibration ####################
```

```
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 Calibration ####################
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

################ Write Calibration Data to 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 Code Output

```
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 Programming

After all device settings are configured and calibration is complete, the settings can be permanently written to the device OTP. Ideally, OTP should be written only one time, but there is some flexibility:

- The OTP memory includes two full images of the Data Memory configuration settings. This means a setting can be changed from the default and changed back to the default once, then that settings can no longer be modified. More practically, this characteristic can be used to write the OTP in sections. For example, the main settings can be written to OTP and at a later production step, the calibration settings can be written to OTP.
- However, partial changes to OTP are not unlimited. This is because an OTP signature is calculated and also stored in OTP each time the OTP is written. The device supports up to 8 different signature values, so there is a maximum number of 8 OTP partial writes.

3.1 Recommended Steps for Writing OTP in Production

In order to write to OTP, a voltage between 10 to 12 V should be applied to the BAT pin and the device must be in FULLACCESS mode. The recommended steps are listed below to write to OTP.

1. Check whether OTP programming has already been done on the device by reading one of the programmed registers. When power is applied, registers will either report the default values or the values programmed in OTP if OTP has been programmed. If OTP programming has not been done, proceed to the next steps.
2. Read the *0x12 Battery Status[SEC1,SEC0]* bits to verify the device is in FULL ACCESS mode (0x01).
3. If the device is in FULL ACCESS mode, enter CONFIG_UPDATE mode - (Subcommand 0x0090).
4. Configure the register settings in data memory.
5. Exit CONFIG_UPDATE mode - (Subcommand 0x0092).
6. Read the data memory registers to verify all parameters were written successfully.
7. Enter CONFIG_UPDATE mode.
8. Check the *Battery Status[OTPB]* bit is clear to verify OTP programming conditions are met.
9. Read OTP_WR_CHECK() (Subcommand 0x00A0). If this returns a value of 0x80, then OTP programming conditions are met.
10. If OTP_WR_CHECK indicates conditions are met, send OTP_WRITE() subcommand (0x00A1).
11. Wait 100 ms. Read from 0x40 to check if OTP programming was successful (0x80 indicates success).
12. Exit CONFIG_UPDATE mode - (Subcommand 0x0092).

4 References

- Texas Instruments, *BQ76952 3S-16S Battery Monitor and Protector Data Sheet*
- Texas Instruments, *BQ76942 3S-160S Battery Monitor and Protector Data Sheet*
- Texas Instruments, *BQ76952 Technical Reference Manual*
- Texas Instruments, *BQ76942 Technical Reference Manual*
- Texas Instruments, BQ76952 Evaluation Module User's Guide
- Texas Instruments, BQ76942 Evaluation Module User's Guide
- Texas Instruments, BQ76942, BQ76952 Software Development Guide

5 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (October 2020) to Revision A (August 2021)	Page
• Update was made in the Abstract of this document	1
• Updated the numbering format for tables, figures and cross-references throughout the document	2
• Updates were made in Section 3	14

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (https:www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265