

ABSTRACT

This application report presents a strategy for high speed, economical, calibration and production programming of the bq34z100-G1 fuel gauge. Sample code and flowchart examples are provided, along with instructions for preparing an optimized golden image (.dffs, .bqfs or .srec) flash file. This file is programmed into all bq34z100-G1 devices at the pack maker production line.

Contents

Introduction

The bq34z100-G1 fuel gauge family is built with new technology and a new architecture for both data flash access and calibration. With this new architecture, unit production cost and capital equipment investment can be minimized, as it is no longer necessary to perform a learning cycle on each pack. A single golden image file can be used to program each bq34z100-G1 in production. Also, the calibration method is quick and simple because most of the calibration routines can be based on average values.

A good strategy for bq34z100-G1 production is an eight-step process flow:

Step 1. Write the data flash image to each device.

Step 2. Calibrate the voltage (optional for $\leq 5V$ applications).

Step 3. Update any individual flash locations, such as serial number, lot code, and date.

Step 4. Perform any desired board level tests

Step 5. Connect the cells.

Step 6. Perform any desired pack level tests.

Step 7. Send 0x0021 to Manufacturer Access 0x00 command, to enable Impedance Track, Lifetime, and Permanent Fail functions.

Step 8. Send 0x0020 to Seal the pack. In this document, pre-production, and the first three production steps are examined in detail.

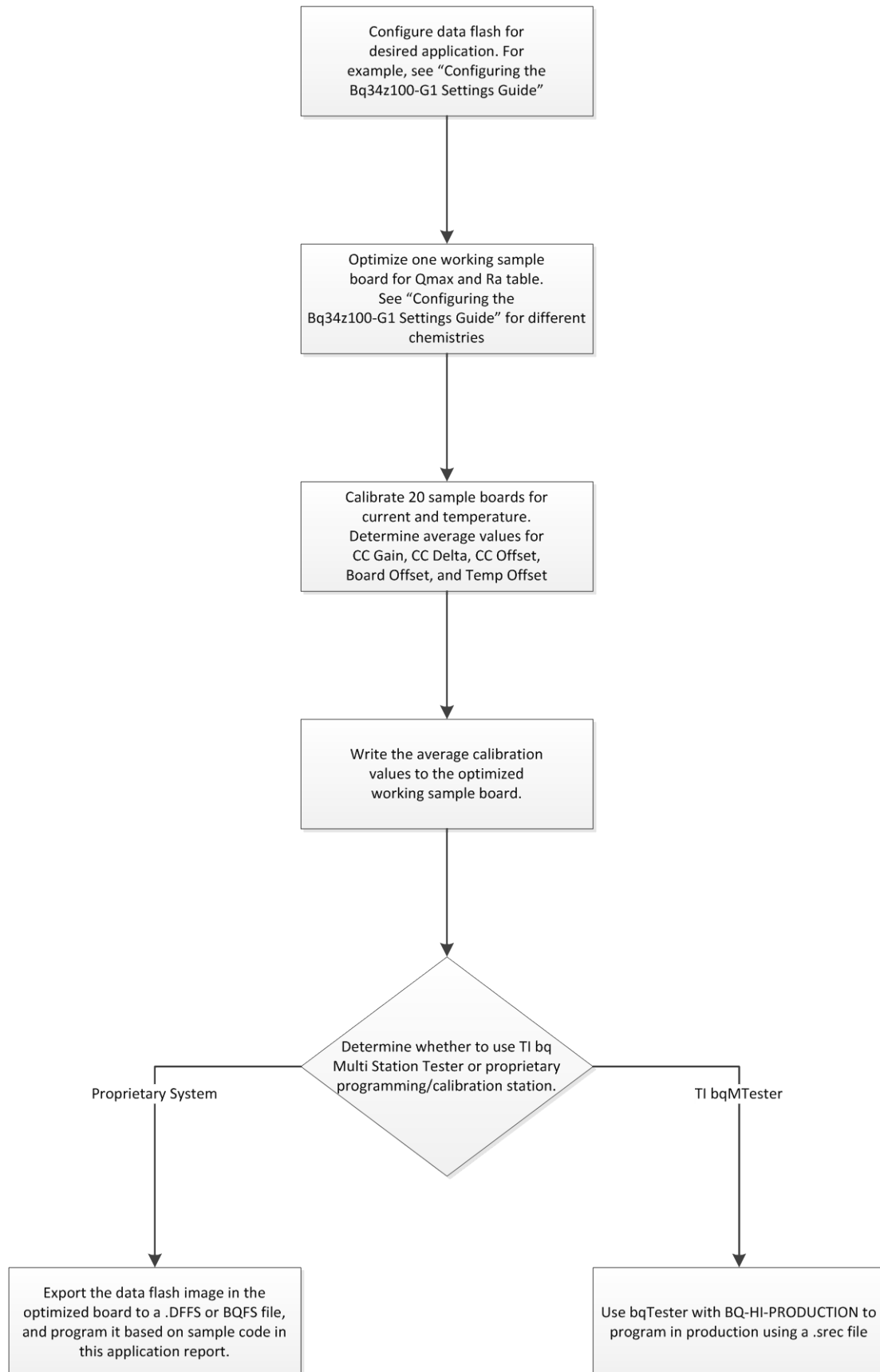
Pre-Production Preparation

To configure the bq34z100-G1 for a given application, the data flash set of constants must be programmed based on the cell type, application, system, and charger. The series of application reports

entitled “Quick Start Guide for the bq34z100-G1” presents a detailed description of all the data flash constants that the user can modify.

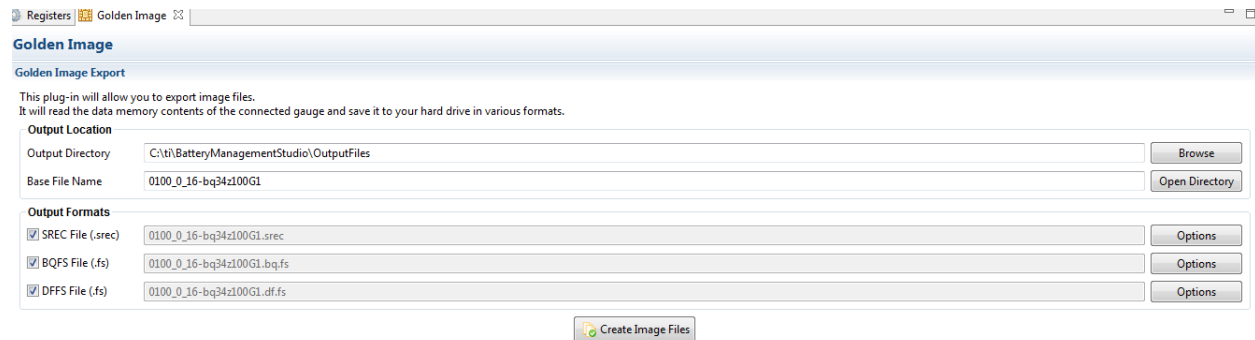
The “golden image file” is a binary file containing the data flash image from an optimized and validated fuel gauge containing average current and temperature calibration values obtained from at least 20 sample units.

It is a file with either a .dffs, .bqfs or .srec file extension. The .dffs or .bqfs files are programmed into the gauge using I2C™ communication with the bq34z100-G1 using a programming platform developed by the customer. The .bqfs is similar to the .dffs, but contains the firmware in addition to all the data flash settings. The .srec file can be programmed using the Advanced bqMTester using the bq-Hi-Production software available from Texas Instruments.



Exporting the golden image:

Using bqStudio is the easiest method to export the golden image. In the tool bar of bqStudio there is a “Golden Image” tool. Clicking on this will open the golden image tab as follows:



The screenshot shows the 'Golden Image' tab in bqStudio. The 'Golden Image Export' section contains the following fields and options:

- Output Location:**
 - Output Directory: C:\ti\BatteryManagementStudio\OutputFiles (with a 'Browse' button)
 - Base File Name: 0100_0_16-bq34z100G1 (with an 'Open Directory' button)
- Output Formats:**
 - ☒ SREC File (.srec): 0100_0_16-bq34z100G1.srec (with an 'Options' button)
 - ☒ BQFS File (.fs): 0100_0_16-bq34z100G1.bq.fs (with an 'Options' button)
 - ☒ DFFS File (.fs): 0100_0_16-bq34z100G1.dff.fs (with an 'Options' button)

At the bottom center is a 'Create Image Files' button.

All three or the chosen golden image format can be exported from this screen.

PRODUCTION STEP 1: Write the Data Flash Image to Each Device Pack PCB designers must ensure that the I 2 C lines of bq34z100-G1 are accessible at time of writing flash in production.

Instructions to program .dffs or .bqfs golden files:

File Structure The .bqfs and .dffs file is an ASCII text file which contains both commands and data. Each line of the file represents one command and potentially 96 bytes of data, as described in the following text. No row contains more than 96 data bytes. The first two characters of each row represent the command, followed by a ":".

"W:" – indicates that the row is a command to write one or more bytes of data.

"R:" – indicates that the row is a command to read one or more bytes of data.

"C:" – indicates that the row is a command to read and compare one or more bytes of data.

"X:" – indicates that the row is a command to wait a given number of milliseconds before proceeding. White space is used to separate fields within the .bqfs and .dffs files. Each row contains one and only one of the four commands.

The commands discussed in this section can be implemented by a system that can perform multibyte operations for I2C.

Write Command

The write command "W:" instructs the I2C master to write one or more bytes to a given I2C address and given register address. The I2C address format used throughout this document is based on an 8-bit representation of the address. The format of this sequence is:

"W: I2CAddr RegAddr Byte0 Byte1 Byte2...".

For example, the following:

W: AA 55 AB CD EF 00

indicates that the I2C master writes the byte sequence 0xAB 0xCD 0xEF 0x00 to register 0x55 of the device addressed at 0xAA.

More precisely, it indicates to write the following data to the device address

0xAA: 0xAB to register 0x55

0xCD to register 0x56

0xEF to register 0x57

0x00 to register 0x58

Read Command

The read command "R:" instructs the I2C master to read a given number of bytes from a given I2C address and given register address. The format of this sequence is:

"R: I2CAddr RegAddr NumBytes"

For example, the following:

R: AA 55 100

indicates that the I2C master reads 100 (decimal) bytes of data from register address 0x55 of device address 0xAA.

Read and Compare Command

The read and compare command is formatted identically to the write command. The data presented with this command matches the data read exactly, or the operation ceases with an error indication to the user.

The format of this sequence is:

"C: i2cAddr RegAddr Byte0 Byte1 Byte2".

An example of this command is as follows:

C: AA 55 AB CD EF 00

This example expects the master to read back 4 bytes from the register address 0x55 of the device addressed at 0xAA and then compare the data to the values given on the line command in this same order as 0xAB, 0xCD, 0xEF, and 0x00.c.

Wait command

The wait command indicates that the host waits a minimum of the given number of milliseconds before continuing to the next row of the flash stream.

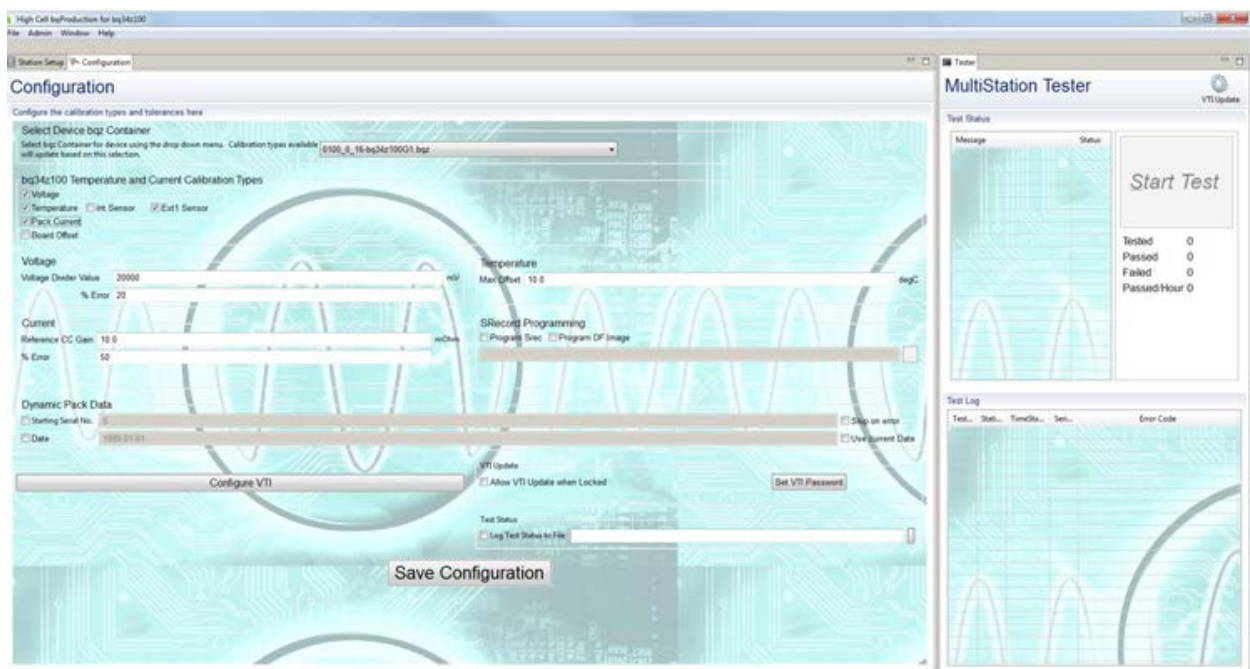
For example, the following:

X: 200

indicates that the I2C or HDQ master must wait at least 200 ms before continuing.

Programming .srec file

Alternately, the Texas Instruments “Advanced bqMTester” along with “bq-Hi-Production” program may be used for writing the image as well as performing voltage calibration quickly and inexpensively. With more complex fuel gauge types, this program is used in conjunction with a hardware platform available from TI, which performs current, temperature, and voltage calibration. In the case of bq34z100-G1, the circuit board is not actually necessary or relevant, but could be purchased and used as-is or modified for custom voltage stacks.



PRODUCTION STEP 2: Calibrate the Voltage

While it is recommended that current and temperature calibration factors be derived from an average of 20 production units, the same is not true for voltage calibration. However, in applications where the peak battery voltage does not exceed 5V, the internal TI factory-calibrated divider network may be used to avoid an external divider and calibration altogether. See the device datasheet for additional details.

Voltage calibration, using the “bq-Hi-Production” program or proprietary system is based on modifying the Voltage Divider data flash constant to achieve best possible accuracy. The formula for calibration is as follows:

$$\text{New Voltage Divider} = \text{Voltage Divider} * \text{Known Applied Voltage} / \text{I2C Reported Voltage}$$

The Known Applied Voltage, as measured by an agency-traceable DMM, is typed into one of the configurations screens on the “bq-Hi-Production” program. The meter used for establishing this value should be accurate to less than one millivolt.

To write the new Voltage Divider value can be programmed as follows:

- SubclassID for “Voltage Divider” is 104 or 0x68 hex, and offset is 14
- Write 0x00 using BlockDataControl() command (0x61) to enable block data flash control.

(wr 0x61 0x00)

- Write 0x68 (Calibration Subclass) using the DataFlashClass() command (0x3E) to access the Registers subclass.

(wr 0x3E 0x68)

- Write the block offset location using DataFlashBlock() command (0x3F). To access data located at offset 0 to 31, use offset = 0x00.

For example, Voltage Divider (offset = 14) is in the first block so use (wr 0x3F 0x00).

- To read the data of a specific offset, use address 0x40 + mod(offset, 32). So to read the old Voltage divider (rd 0x4E old_Voltage Divider_MSB) (rd 0x4F old_Voltage Divider_LSB)
- To write the data of a specific offset, use address 0x40 + mod(offset, 32). So to write the new Voltage divider (rd 0x4E new_Voltage Divider_MSB) (rd 0x4F new_Voltage Divider_LSB)
- The data is actually transferred to the data flash when the correct checksum for the whole block (0x40 to 0x5F) is written to BlockDataChecksum() (0x60).

(wr 0x60 NEW_checksum)

The checksum is (255-x) where x is the 8-bit summation of the BlockData() (0x40 to 0x5F) on a byte-by-byte basis.

A quick way to calculate the new checksum is to make use of the old checksum:

(a) temp = mod (255 – OLD_checksum – old_Voltage Divider_MSB - old_Voltage Divider_LSB), 256)

(b) $\text{NEW_checksum} = 255 - \text{mod}(\text{temp} + \text{new_Voltage Divider_MSB} + \text{new_Voltage Divider_LSB}, 256)$

Step 3. Update any individual flash locations, such as serial number, lot code, and date:

In a similar manner to the Voltage divider, any pack specific data such as serial number, lot code and date can be changed during the production process.