Application Report
SLUA665 – November 2012

Going to Production With the bq34z1xx

Doug Williams
BMS – Multi-Cell Solutions

ABSTRACT

This application report presents a strategy for high speed, economical, calibration and production programming of the bq34z1xx fuel gauge family. Sample code and flowchart examples are provided, along with instructions for preparing an optimized golden image (.DFI or .ROM) data flash file. This file is programmed into all bq34z1xx devices at the pack maker production line.

Contents

Figures

Introduction

The bq34z1xx 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 bq34z1xx in production. Also, the calibration method is quick and simple because most of the calibration routines can be based on average values.

The methods in this document are presented as VB6 (Visual Basic 6) functions. These functions were copied directly from working code. In order to read from and write to the data flash, they use four types of I2C read and write functions. These can be duplicated in any software environment that has I2C Bus communication capabilities.

1. WriteI2CByte( ) has three arguments – the Command Address, Byte Data, and Device Address.
2. WriteI2CInteger( ) has three arguments – the Command Address, Integer Data, and Device Address. Internally, this function separates the integer into two bytes for transmission by the I2C 1-byte write protocol.
3. WriteI2CByteArray( ) has four arguments – the Command Address, Byte Array to Write, Length of Byte Array, and Device Address. Internally, this function separates the byte array into separate bytes for transmission by the I2C 1-byte write protocol.
4. ReadI2CByteArray( ) has four arguments– the Command Address, Returned Byte Array, Length of Byte Array, and Device Address. It is internally implemented with the I2C Incremental read protocol.

Error handling is not implemented in this sample code, because requirements are unique and varied. Also, constants are hard-coded into the functions to improve clarity rather than documenting them in code elsewhere as would normally be good coding practice.

A good strategy for bq34z1xx production is an eight-step process flow:

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

Step 2. Calibrate the voltage (optional for <= 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 and convert to HDQ communication if required.

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. The method for converting to HDQ is described in Appendix A.

Pre-Production Preparation

To configure the bq34z1xx for a given application, the data flash set of constants must be programmed depending on the cell type, application, system, and charger. The application report entitled "Configuring the bq34z100 Data Flash" presents a detailed description of all the data flash constants that the user can modify. Similar application reports are available for other members of the bq34z1xx family. All bq34z1xx ICs for an application will contain the same data flash, except for pack specific parameters such as serial number, manufacture date, voltage calibration, and others as required by the producer.

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 binary file with either a .DFI or an .ROM file extension. The .DFI is programmed into the gauge using I2C™ communication with the bq34z1xx using a programming platform developed by the customer. The .ROM is similar to the .DFI, but contains a special header identifying the device, and is programmed using I2C™ communication with the TI "bq Multi Station Tester" mass production program. Creating the .DFI or .ROM can be summarized with the process depicted in Figure 1. Sample code (using VB6) for creating the .DFI file is presented in Figure 2.

If it is desired to use the "bq Multi Station Tester" program, a .ROM file may be easily created with the .ROM DataFlash Reader utility as shown in Figure 3.

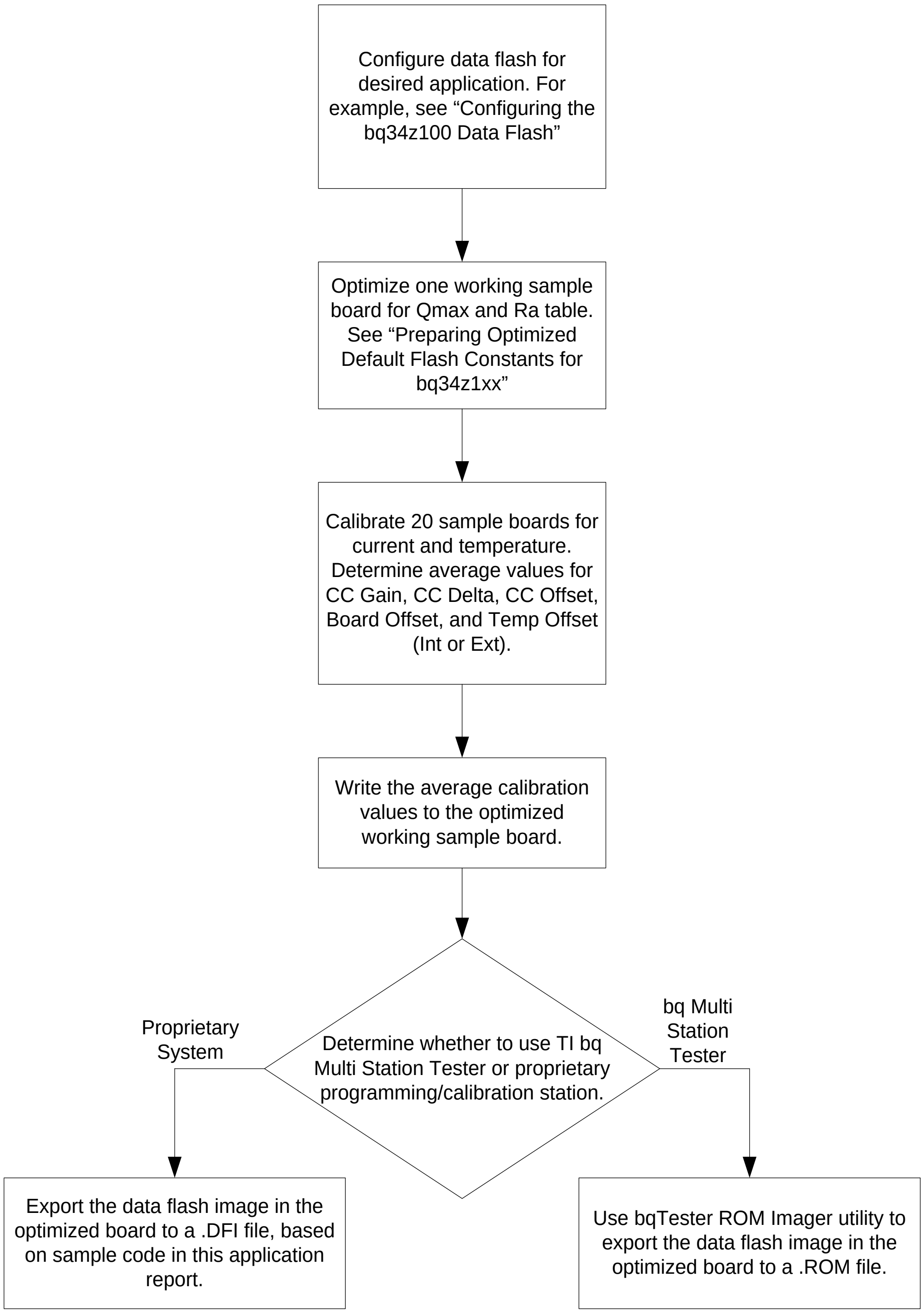


Figure 1. Pre-production tasks to create the Golden Image File.

```
Function Save_DFI_to_File(sFilename As String) As Long

    Dim iNumberOfRows As Integer
    Dim iBaseAddr As Integer
    Dim iAddrMS As Integer
    Dim iAddrLS As Integer
    Dim lError As Long
    Dim yRowData(&H20) As Byte
    Dim yDataFlashImage(&H400) As Byte
    Dim iRow As Integer
    Dim iIndex As Integer
    Dim iFileNumber As Integer

    '// FOR CLARITY, WITHOUT USING CONSTANTS. 32 Rows
    iNumberOfRows = &H20
    iBaseAddr = &H4000

    '// PUT DEVICE INTO ROM MODE
    lError = WriteI2CInteger(0, &HFFFF, &HAA)
    lError = WriteI2CInteger(0, &HFFFF, &HAA)
    DoDelay 0.5 '// Wait 0.5 seconds
    lError = WriteI2CInteger(0, &HF00, &HAA)
    DoDelay 0.1 '// Wait 0.1 seconds

    '// READ THE DATA FLASH, ROW BY ROW
    '//Note that ROM mode uses I2C address 0x16 instead of 0xAA
    For iRow = 0 To iNumberOfRows - 1

      '// INITIATE PEEK-BYTES COMMAND
      lError = WriteI2CByte(&H0, &H7, &H16)

      '// SET ROW ADDRESS.
      iAddrLS = (iBaseAddr + (iRow * &H20)) Mod &H100
      iAddrMS = (iBaseAddr + (iRow * &H20)) \ &H100
      lError = WriteI2CByte(&H1, iAddrLS, &H16) '//Low address byte
      lError = WriteI2CByte(&H2, iAddrMS, &H16) '//High address byte

      '// SET NUMBER OF BYTES TO READ
      lError = WriteI2CByte(&H4, &H20, &H16) '//High address byte

      '// WRITE CHECKSUM
      lError = WriteI2CByte(&H64, (&H7 + &H20 + iAddrLS + iAddrMS) Mod &H100, &H16) '//lsb of checksum
      lError = WriteI2CByte(&H65, (&H7 + &H20 + iAddrLS + iAddrMS) \ 256, &H16) '//msb of checksum

      '// READ THE ROW.
      lError = ReadI2CByteArray(&H5, yRowData, &H20, &H16)

      '//ADD ROW INTO BIG ARRAY
      For iIndex = 0 To &H20 - 1
        yDataFlashImage((iRow * &H20) + iIndex) = yRowData(iIndex)
      Next iIndex

      DoDelay 0.1
    Next iRow
    End With

    '// WRITE DATA FLASH IMAGE TO FILE
    ChDir App.Path
    iFileNumber = FreeFile
    Open sFilename For Binary Access Write As #iFileNumber
    Put #iFileNumber, , yDataFlashImage
    Close #iFileNumber
    ChDir App.Path

    '// EXECUTE GAS GAUGE PROGRAM
    lError = WriteI2CByte(0, &HF, &H16)
    lError = WriteI2CByte(&H64, &HF, &H16)
    lError = WriteI2CByte(&H65, 0, &H16)

    '// RETURN OK
    Save_DFI_to_File = 0
  End Function
```

Figure 2. DFI Export Sample Code

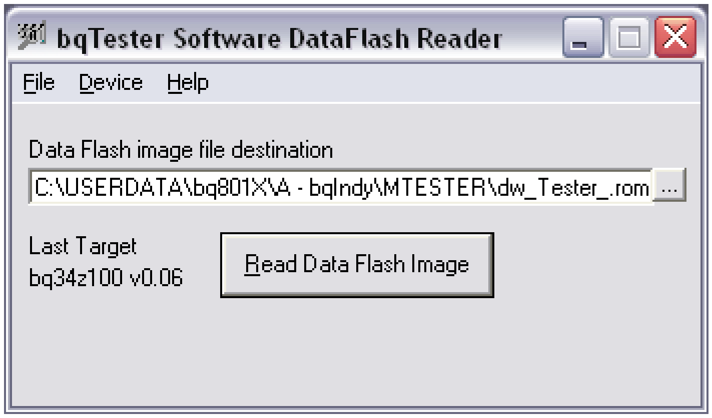


Figure 3. DataFlash Reader Utility (ROM Imager) for “bq Multi Station Tester” produces .ROM file

PRODUCTION STEP 1: Write the Data Flash Image to Each Device

Pack PCB designers must ensure that the I^2C lines of bq34z1xx are accessible at time of writing DFI in production.
If a proprietary system is used for writing the image, a routine based on the code in Figure 4 should be used.

```
'// PUT DEVICE INTO ROM MODE
   lError = WriteI2CInteger(0, &HFFFF, &HAA)
   lError = WriteI2CInteger(0, &HFFFF, &HAA)
   DoDelay 0.2 '// Wait 0.2 seconds
   lError = WriteI2CInteger(0, &HF00, &HAA)
   DoDelay 0.2 '// Wait 0.2 seconds

   '//Note that ROM mode uses I2C address 0x16 instead of 0xAA

   '// MASS ERASE DATA FLASH
   lError = WriteI2CByte(&H0, &HC, &H16)
   lError = WriteI2CByte(&H4, &H83, &H16)
   lError = WriteI2CByte(&H5, &HDE, &H16)
   iChecksum = (&HC + &H83 + &HDE) Mod &H10000
   lError = WriteI2CByte(&H64, iChecksum Mod &H100, &H16)
   lError = WriteI2CByte(&H65, iChecksum \ 256, &H16)
   DoDelay 0.5 '// Wait 0.5 seconds

   '// WRITE EACH ROW
   For iRow = 0 To iNumberOfRows - 1
      lError = WriteI2CByte(&H0, &HA, &H16) '//program row command

      '// WRITE TARGET ROW TO THE ROW LOW REGISTER
      lError = WriteI2CByte(&H1, iRow, &H16)
      iChecksum = (&HA + iRow) Mod &H10000

      '// COPY DATA FROM THE FULL ARRAY TO THE ROW ARRAY
      For iIndex = 0 To 31
         yRowData(iIndex) = yDataFlashImage((iRow * 32) + iIndex)
         iChecksum = (iChecksum + yRowData(iIndex)) Mod &H10000
      Next iIndex

      '// WRITE THE ROW DAYA REGISTERS
      lError = WriteI2CByteArray(&H4, yRowData, 32, &H16)

      '// WRITE THE ROW
      lError = WriteI2CByte(&H64, iChecksum Mod &H100, &H16)
      lError = WriteI2CByte(&H65, iChecksum \ 256, &H16)
      DoEvents '//allow Windows to catch up
      DoDelay 0.2 '// Wait 0.2 seconds
   Next iRow
   End With

   '// EXECUTE GAS GAUGE PROGRAM
   lError = WriteI2CByte(0, &HF, &H16)
   lError = WriteI2CByte(&H64, &HF, &H16)
   lError = WriteI2CByte(&H65, 0, &H16)

   '// RETURN OK OR ERROR
   Write_DFI_to_Flash = 0

End Function
```

Figure 4. Write_DFI_to_Flash Sample Code

Alternately, the Texas Instruments "bq Multi Station Tester" 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 bq34z1xx, the circuit board is not actually necessary or relevant, but could be purchased and used as-is or modified for custom voltage stacks.

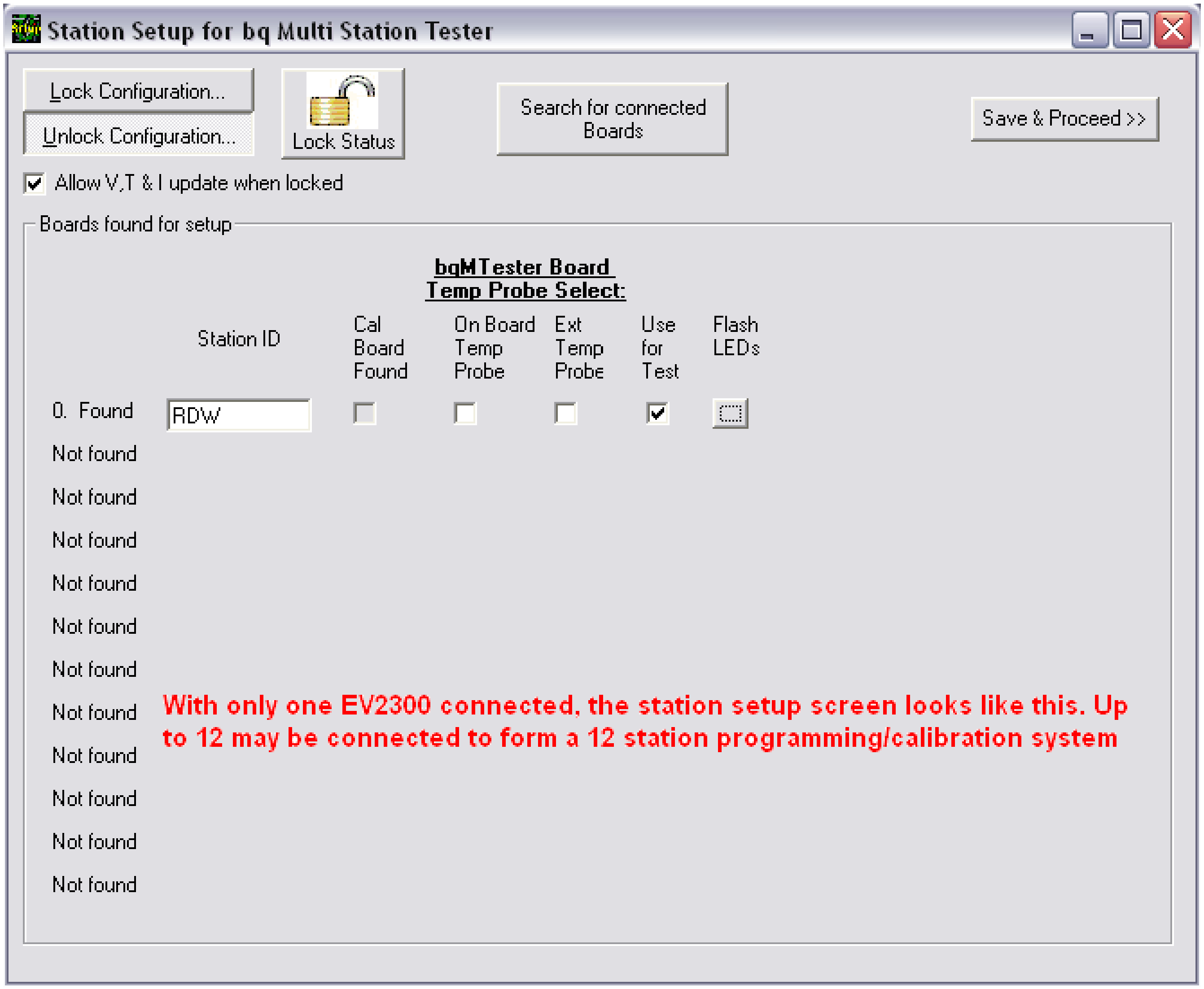


Figure 5. Initial Setup screen for "bq Multi Station Tester" discovers installed EV2300 USB adapters.

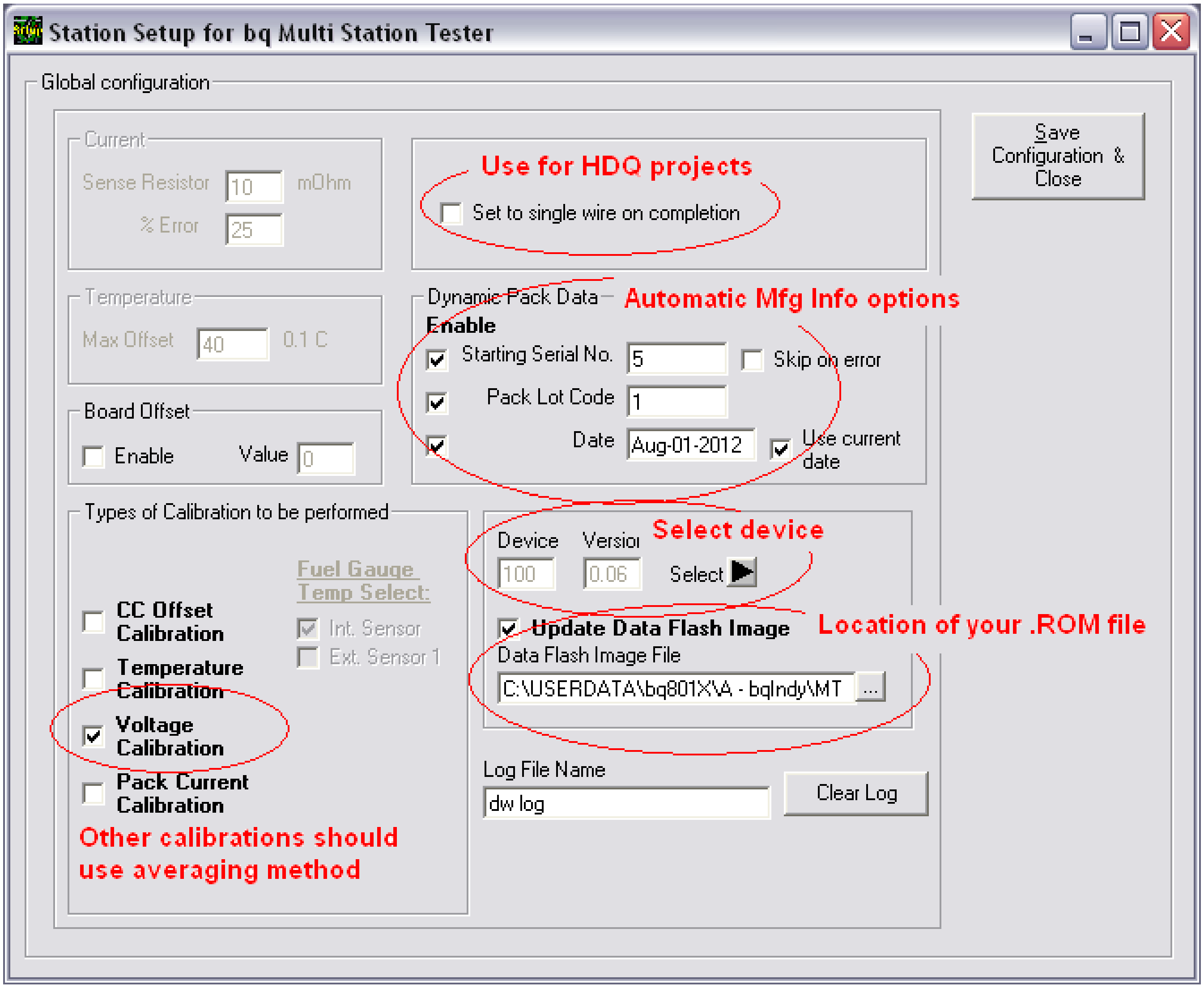


Figure 6. The "bq Multi Station Tester" can be used to calibrate voltage, write a serial number/Lot Code/Mfr Date, and program the golden image .ROM file.

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 Multi Station Tester” 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:

New Voltage Divider = Voltage Divider * Known Applied Voltage / 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 Multi Station Tester” program. The meter used for establishing this value should be accurate to less than one millivolt.

To write the new ***Voltage Divider*** value, use the same technique as demonstrated in Figure 8 for writing the serial number. The only difference is the subclass and offset as found in the device data sheet. To read the I2C Reported Voltage, use the technique demonstrated in Figure 7.

```
Function Read_Voltage(iVoltage As Integer) As Long

   Dim lError As Long
   Dim yData(2) As Byte

   '// READ TWO BYTES FROM COMMANDS  0x08 AND 0x09
   lError = ReadI2CByteArray(&H8, yData, 2, &HAA)

   '// LSB IS IN THE FIRST BYTE, MSB IN THE SECOND
   iVoltage = 256 * yData(1) + yData(0)

   '// RETURN OK OR ERROR CODE
   Read_Voltage = lError
End Function
```

Figure 7. Method to read the I2C Voltage from the gauge

PRODUCTION STEP 3: Update any Individual Flash Locations, such as Serial Number, Lot Code, and Date.

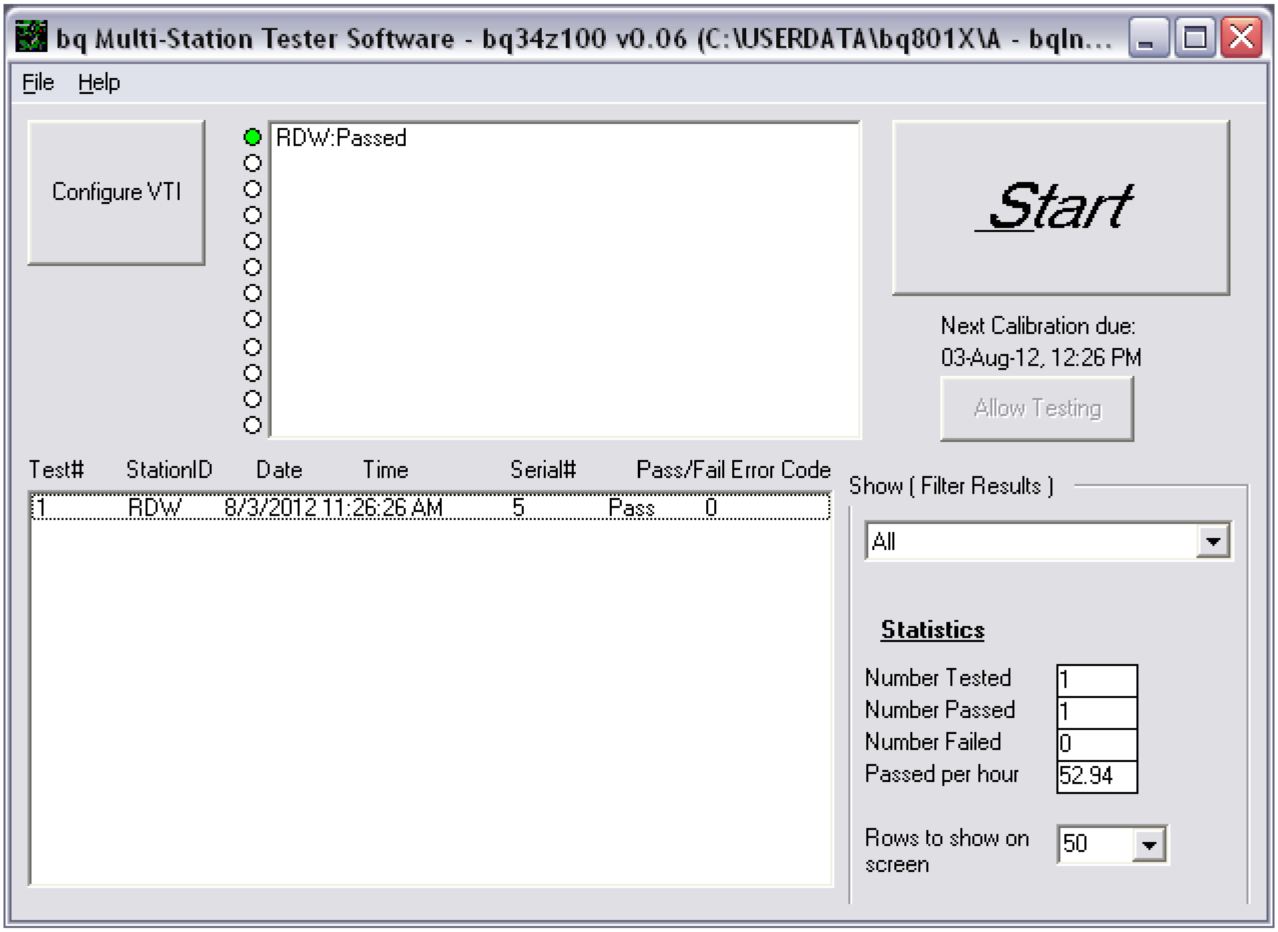
Other than the Voltage Divider value, there will usually be some data that is unique to each battery pack, or group of packs such as serial number, date of manufacture, etc. This data can be written using the technique below in Figure 8.

```
Function UpdateSerialNumber(iSerialNum As Integer) As Long

   Dim lError As Long
   Dim iSubClass As Integer
   Dim iTransferCode As Integer
   Dim yRowData(32) As Byte
   Dim iChecksum As Integer
   Dim iTmp As Integer
   Dim i As Integer

   iSubClass = 48 '// subclass for configuration data
   iTransferCode = 0

   '//ENABLE FLASH TRANSFER
   lError = WriteI2CByte(&H61, iTransferCode, &HAA)

   '//SPECIFY SUBCLASS
   lError = WriteI2CByte(&H3E, iSubClass, &HAA)

   '// ENABLE GENERAL PURPOSE BLOCK
   lError = WriteI2CByte(&H3F, iTransferCode, &HAA)

   '//READ 32 BYTE BLOCK
   lError = ReadI2CByteArray(&H40, yRowData, 32, &HAA)

   '// REPLACE SERIAL NUMBER RAM.  ROW OFFSETS ARE FOUND IN THE DATASHEET
   yRowData(15) = (iSerialNum \ 256) '//MSByte
   yRowData(16) = iSerialNum - (yRowData(0) * 256) '//LSByte

   '//CALCULATE THE CHECKSUM BYTE AND INVERT IT
   For i = 0 To 31
    iChecksum = iChecksum + yRowData(i)
   Next i

   iTmp = iChecksum \ 256 '//Integer divide
   iChecksum = iChecksum - (iTmp * 256)
   iChecksum = 255 - iChecksum

   '// MOVE THE SERIAL NUMBER INTO THE FUEL GAUGE BUFFER
   lError = WriteI2CByte(&H40 + 15, CInt(yRowData(15)), &HAA)
   lError = WriteI2CByte(&H40 + 16, CInt(yRowData(16)), &HAA)

   '// TRANSFER TO FLASH USING THE CHECKSUM
   lError = WriteI2CByte(&H60, iChecksum, &HAA)

   DoDelay 0.2 '// Wait 0.2 seconds

End Function
```

Figure 8. Method to Write a Unique Serial Number

PRODUTION STEPS 1, 2, and 3 - Using the bq Multi Station Tester.

The “bq Multi Station Tester” program is available as a free download and may be convenient for users without the time or resources to develop a proprietary programming and calibration station. The program can handle from one to twelve stations.



Appendix A. – Converting to HDQ Communication

If manufacturers develop proprietary tools to program the DFI and need to set the device to HDQ mode, the following steps are required.

After writing the DFI but before sending the commands to exit ROM mode, send the following commands:

(a) I2C Command 0x00: Data Byte 0x16

(b) I2C Command 0x04: Data Byte 0x05

(c) I2C Command 0x64: Data Byte 0x1B

(d) I2C Command 0x65: Data Byte 0x00

Finish the programming process by exiting ROM mode and sending the following commands:

(a) I2C Command 0x00: Data Byte 0x0F

(b) I2C Command 0x64: Data Byte 0x0F

(c) I2C Command 0x65: Data Byte 0x00

IMPORTANT NOTICE

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

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