LTC3335 - Nanopower Buck-Boost DC/DC with Integrated Coulomb Counter

DESCRIPTION

Demonstration Circuit DC2343A is a complete system level solution for a Nanopower Buck-Boost DC/DC with Integrated Coulomb Counter. The DC2343A contains a PIC16F1459 embedded processor which communicates with the LTC3335 via I²C, monitors VBAT and VOUT voltages, and communicates with a GUI via USB. The GUI is capable of reading and writing all the control registers on the LTC3335 as well as displaying and resetting its alarm register. Connection for an alternative microprocessor can be made through JP5 to bypass the PIC16. Reference the DC2343A demo manual for a description of the hardware, and the operation of the board with the GUI: http://www.linear.com/docs/47042

This document describes the source code provided for Demonstration Circuit DC2343A. Source code is provided for both the firmware (FW) programmed into the PIC16F1459 and for the GUI software (SW) run on a Windows PC. All source code is provided allowing the FW and SW to be built with free tools available from Microchip and Microsoft. The source code that most useful for LTC3335 applications are contained in the files LTC3335.c (FW) and LTC3335.vb (GUI).

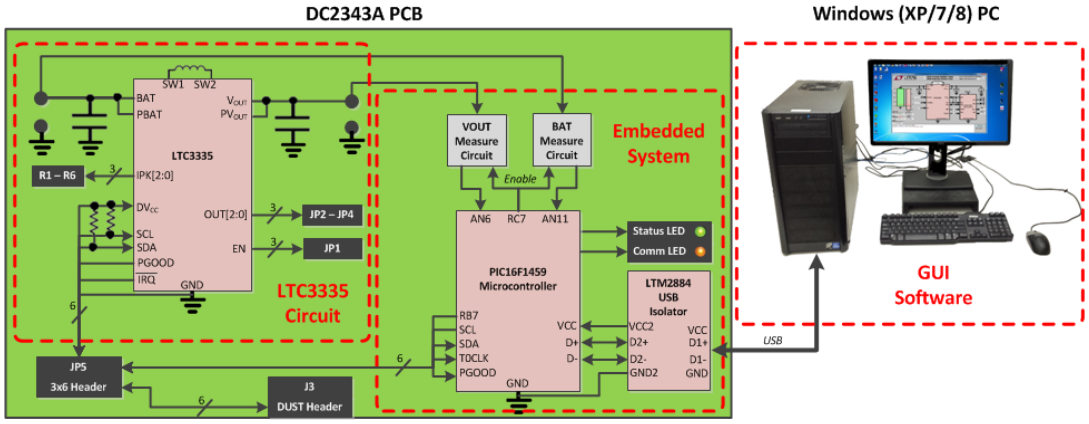

Figure 1 - DC2343A Hardware Block Diagram

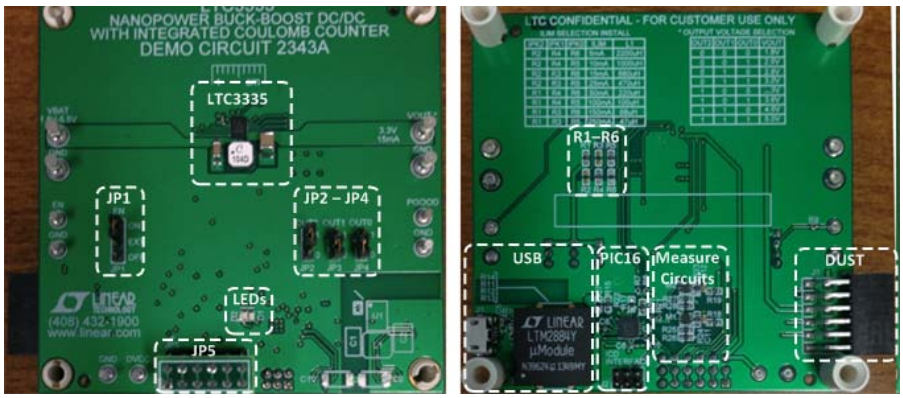

Figure 2 - DC2343A Hardware

SOFTWARE OVERVIEW

The DC2343A source files are pictured in Figure 3 and can be downloaded from http://www.linear.com/solutions/5948.

The PIC16F1459 Firmware (**FW**), executing in the microcontroller on the DC2343A PCB, is written in C for the XC8 compiler v1.33 with an MPLab X v2.26 project.

MPLab X can be freely downloaded from Microchip Technology: (http://www.microchip.com/pagehandler/en-us/devtools/dev-tools-parts.html). During the installation process, MPLab X will detect that a compiler is not installed and will offer to download and install the free version of XC8.

The GUI Software (**GUI SW**), developed for Windows XP/7/8 computers with v4.5 of the .NET framework, is written in Visual Basic using Microsoft Visual Studio 2010.

Visual Basic Express 2010 can be freely downloaded from Microsoft to allow builds of the GUI software: (https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx).

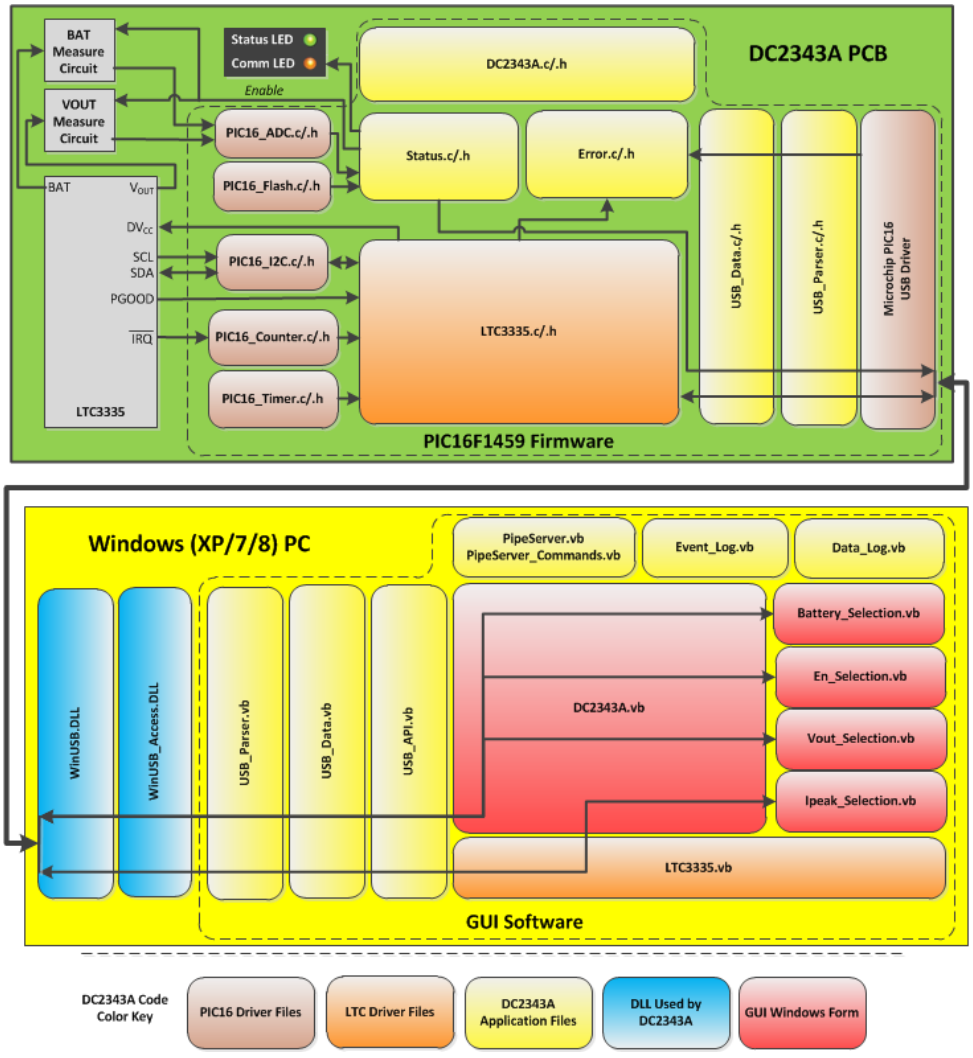

Figure 3 - DC2343A Code Architecture

Figure 3 shows the data path from the hardware on the DC2343A PCB, through the DC2343A FW, over the USB connection to a Windows XP/7/8 PC, for display through the DC2343A GUI SW. The files are color coded to indicate their usefulness in the end application of a customer. Only the LTC Driver files are intended to be used in an end application. The PIC16 Driver files are specific to the PIC16 microprocessor, where the DC2343A Application Files, the GUI Windows Forms, and the DC2343A DLLs are specific to creating a general purpose demo board.

HARDWARE INTERFACE AND PIC16 DRIVER FW

The most important hardware on the DC2343A is the LTC3335 Nanopower Buck-Boost DC/DC with Integrated Coulomb Counter. There are also two circuits to measure the voltages on the BAT and VOUT pins of the LTC3335 while consuming very little power. Two LEDs provide information about the status of the DC2343A.

The DC2343A FW interfaces to 5 pins of the LTC3335 through the PIC16 driver files:

1. The DVCC pin is driven directly with a GPIO.
2. The SDA and SCL pins interface to the I2C module of the PIC16, accessed through the PIC16_I2C.c/.h driver files.
3. PGOOD is read directly with a GPIO.
4. Although /IRQ can be read directly with a GPIO, it is also accessed through the PIC16 Counter module so that the number of transitions on this pin can be accurately counted when the LTC3335 Counter Test feature is active.

The DC2343A FW interfaces to the voltage measurement circuits through 3 pins:

1. A GPIO is used to activate the circuits only when the voltages are being read by the FW. This minimizes power consumption from the battery.
2. The two voltages are then read with the 10 bit ADC module of the PIC16, accessed through the PIC16_ADC.c/.h driver files.

The DC2343A FW interfaces to the LEDS through 2 pins:

1. A GPIO is used to toggle the Green LED at a rate indicating the status of the DC2343A. The GPIO will be toggled quickly if a battery is not present in an acceptable voltage range for the DC2343A.
2. A GPIO is used to toggle the Amber LED each time the PIC16F1459 communicates with the LTC3335.

LTC3335 DRIVER FW

The LTC3335.c/.h driver FW interacts with the LTC3335 IC through the PIC16 driver files. These are the FW files that are most useful to a customers end application. See Doxygen Documentation for a list of the functions available to interact with the LTC3335.

The LTC3335.c/.h driver provides the following services to application code:

1. **Perform the basic register commands necessary to access the features of the LTC3335:** The functions of the LTC3335 are accessed through an API that abstracts away from the register transactions.
2. **Minimize I2C communication in an effort to minimize power consumption:** When the same LTC33335 register is repeatedly accessed via I2C, it is not necessary to send the subaddress byte with each communication. The LTC3335.c/.h driver file uses static memory to track the last subaddress accessed, so as to minimize the amount of I2C traffic sent to the LTC3335.
3. **Use the Counter Test feature of the LTC3335 to provide a real time battery current measurement:** When the Counter Test feature of the LTC3335 is active, the Counter and Timer modules of the PIC16 are used to accurately count the number of edges on the /IRQ pin over time. The number of edges / time is proportional to the battery current measured by the LTC3335. See Appendix A for a description of using the Counter Test feature to estimate battery current.

APPLICATION FW

While these files are provided for completeness, it is not expected for these files to be used in a customer's end application.

1. DC2343.c/.h contains the main() and isr() functions for the DC2343A Firmware. Note that the PIC16 only has one ISR.
2. Status.c/.h enables the voltage measurement circuits, reads the ADC values, and uses calibration factors stored in PIC16 Flash to convert them to voltage measurements. It controls the flashing of the Status LEDs according to the value of these voltages. It pulses the Comm LED when notified by the LTC3335.c/.h driver that communication had occurred with the LTC3335 IC.
3. Error.c/.h collects information about errors in the DC2343A system so that they can be reported to the GUI. It was primarily intended to expose any I2C communication issues with the LTC3335, although it also reports USB communication errors.

USB FW and SW

While these files are provided for completeness, it is not expected for these files to be used in a customer's end application.

Data is passed from the FW to the GUI through a layered USB stack.

1. USB_Data.c/.h groups the data available from the FW into collections defined by unique DataIDs.
2. USB_Parser.c/.h frames the USB Data with a packet header, and passes this data to the USB Driver.
3. A USB Driver freely available from Microchip Technology was included with the DC2343A project. This driver was configured to operate as a WinUSB device.
4. DLLs are used on the Windows XP/7/8 PC to receive data from the LTC WinUSB device. These DLLs are installed by the LTC QuikEval software if they are not already included with the Windows installation.
5. USB_Parser.vb receives the framed USB Data data from the DLL and verifies the format.
6. USB_Data.vb converts the USB Data from the compact form used by the DC2343A FW into the more convenient data types available in the GUI Software.
7. USB_API.vb provides functions and callbacks for receiving the USB Data into a Windows Form.

LTC3335 SW

This file contains functions for the more complex mathematical functions that would not fit in the DC2343A FW due to Program Memory limitations in the PIC16F1459. See Doxygen Documentation for a list of the functions available to interact with the LTC3335.

The LTC3335.vb file provides the following services to application code:

1. **Calculate the range and resolution represented by the LTC3335 accumulator in Ah**: this is calculated from formula (1) in the LTC3335 datasheet, adjusting for the prescaler value.
2. **Calculate the optimal LTC3335 prescaler for a given IPeak and Battery Capacity:** this is calculated from formula (2) in the LTC3335 datasheet.
3. **Translate the Counter Test results in edges/second into a battery current estimate in Amps:** since the signal output to the $\overline{IRQ}$ pin in the Counter Test mode is the input clock to the LTC3335 ripple counter, the number of $\overline{IRQ}$ edges per second can be directly translated into battery current. See Appendix A for a description of using the Counter Test feature to estimate battery current.
4. **Correct the coulomb count and battery current estimate:** since the typical coulomb counter error is consistent for a given IPeak, VBAT, and VOUT, a lookup table allows the amount of coulombs per count to be corrected given the current LTC3335 conditions. See Appendix B for a description of using software correction to improve the accuracy of the LTC3335 coulomb count.

GUI WINDOWS FORM SW

While these files are provided for completeness, it is not expected for these files to be used in a customer's end application.

The DC2343A GUI consists of five windows:

1. DC2343A.vb contains the Control Panel Window, which is the main window in the GUI. The Control Panel Window displays the components on the DC2343A PCB and their relationship with the user's battery. The other four windows are shown by clicking or double clicking on the controls circled on the Control Panel.
2. Battery_Selection.vb contains the Battery Selection Window which allows the user to specify the battery that they have connected to the DC2343A.
3. En_Selection.vb contains the EN Jumper Window which displays the options for JP1 on the DC2343A.
4. Vout_Selection.vb contains the OUT Jumpers Window which displays the options for JP2 – JP4 on the DC2343A.
5. Ipeak_Selection.vb contains the IPEAK Selection Window which configures the GUI so that it will properly interpret the values read from the LTC3335.

GUI APPLICATION SW

While these files are provided for completeness, it is not expected for these files to be used in a customer's end application.

The following code modules extend the functionality of the DC2343A software beyond the five window GUI:

1. PipeServer.vb provides a Named Pipe Server in the DC2343A GUI to receive messages from another Windows program. When commands defined in PipeServer_Commands.vb are sent to the Named Pipe Server, they allow other programs to access the controls as if they were a human operating the GUI. LTC can provide C# and Python code that accesses the DC2343A Named Pipe Server.
2. Event_Log.vb provides a record of GUI events such as battery connection, modification of LTC3335 registers, and changes of the PGOOD and /IRQ outputs. The log is stored in a text file and can be maintained after the GUI is exited to create a record over a long time in which the GUI is periodically used to monitor a DC2343A.
3. Data_Log.vb provides a method of logging the data in the DC2343A GUI to a CSV file at regular intervals.

Appendix A – Estimating Battery Current using LTC3335 Counter Test

One of the more advanced capabilities of the LTC3335 is to provide a signal representing the real-time battery current on the $\overline{IRQ}$ pin. This feature is accessed by calling the LTC3335_Set_Counter_Test() function with the enabled argument set to TRUE. This causes the signal input to the LTC3335 ripple counter to be output on the $\overline{IRQ}$ pin instead of its typical alarm status information. This is shown in Figure 4.

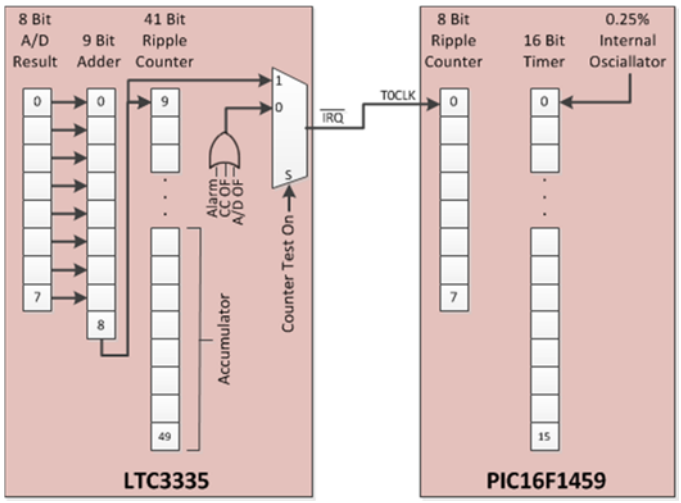

Figure 4 - LTC3335 Hardware in Counter Test Mode

As shown in Figure 5, the DC2343A FW uses a hardware counter to count each positive edge on the $\overline{IRQ}$ pin in PIC16_Counter.c. As the counter only has 8 bits of resolution, an interrupt is used to count overflows of this 8 bit counter, increasing the overall resolution to 16 bits. A hardware timer counts time in PIC16_Timer.c with a resolution of PIC16F_TIMER_TICKS_PER_SEC defined in PIC16_Timer.h. The PIC16 timer has 16 bits of resolution. Both the PIC16 Counter and Timer are allowed to free run. They are never reset after they are initialized, and their overflows are accounted for by the code in LTC3335c

.

LTC3335.c/.h contains two functions that interface to the PIC16 Counter and Timer files:

1. LTC3335_Reset_Counter_Test() records the PIC16 Counter and Timer values are the time it is called, and resets its own edge count and timer values to zero.

2. LTC3335_Counter_Test_Task() is periodically called in main() to read the new values of the PIC16 Counter and Timer. The difference between the new and last values of the PIC16 Counter and Timer are added to the LTC3335 edge count and timer values. This allows the LTC3335 values to counter higher than 16 bits, and account for overflows of the PIC16 values.

Figure 5 shows the relationship of the PIC16 Counter and Timer values during the Counter Test. The accuracy of the estimate for a constant current improves with the number of LTC3335_Counter_Test_Task() calls. In order to detect changes in the battery current, however, the LTC3335_Reset_Counter_Test() function should be called periodically to reset the edge count and timer value. As shown in Figure 3, the LTC3335 edge count and timer values are passed through several USB files as the data is passed to the GUI. When the data is presented to the GUI windows, the timer value is in the units of seconds where the counter value is in $\overline{IRQ}$ edge counts.

The LTC3335.vb file contains the Convert_Counter_Test_Results() function to convert the number of $\overline{IRQ}$ edge counts per second to mA with the equation:

$$\text{Battery Current} = I_{peak} \times t_{FS} \times (\text{IRQ Edges per Sec})$$
$$\text{Where } t_{FS} = 11.74\mu s$$

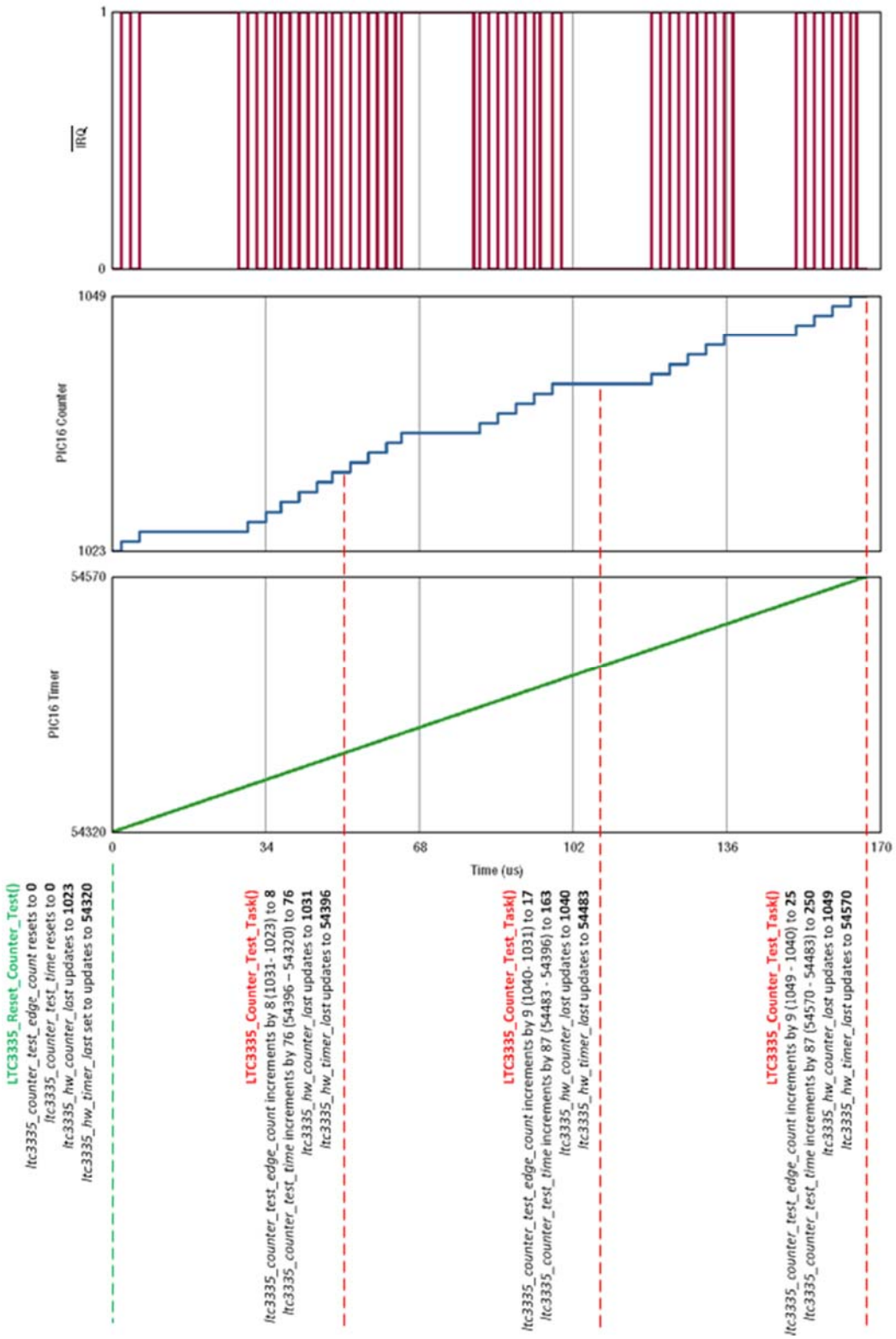

Figure 5 - PIC16 Counter and Timer values during Counter Test

Appendix B – Performing Software Correction using LTC3335 Typical Error

The LTC3335 accuracy is impressive when configured for high values of IPEAK. When configured with low values of IPEAK, however, there can be up to 40% error. The typical error for all configurations of the LTC3335 are shown in Figure 6. The typical error data is compiled as the COULOMB_COUNTER_CORRECTION_TABLE table in the LTC3335.vb file. This table is indexed by the Get_Software_Correction_Factor() function for a given IPEAK setting, VOUT value, and the VBAT value. Linear interpolation is used to estimate the error correction value for voltage conditions that are not contained in the table. Although the GUI uses the sampled VOUT and VBAT values to access this table, most applications could use fixed values for VOUT and VBAT if they are expected to be relatively constant.

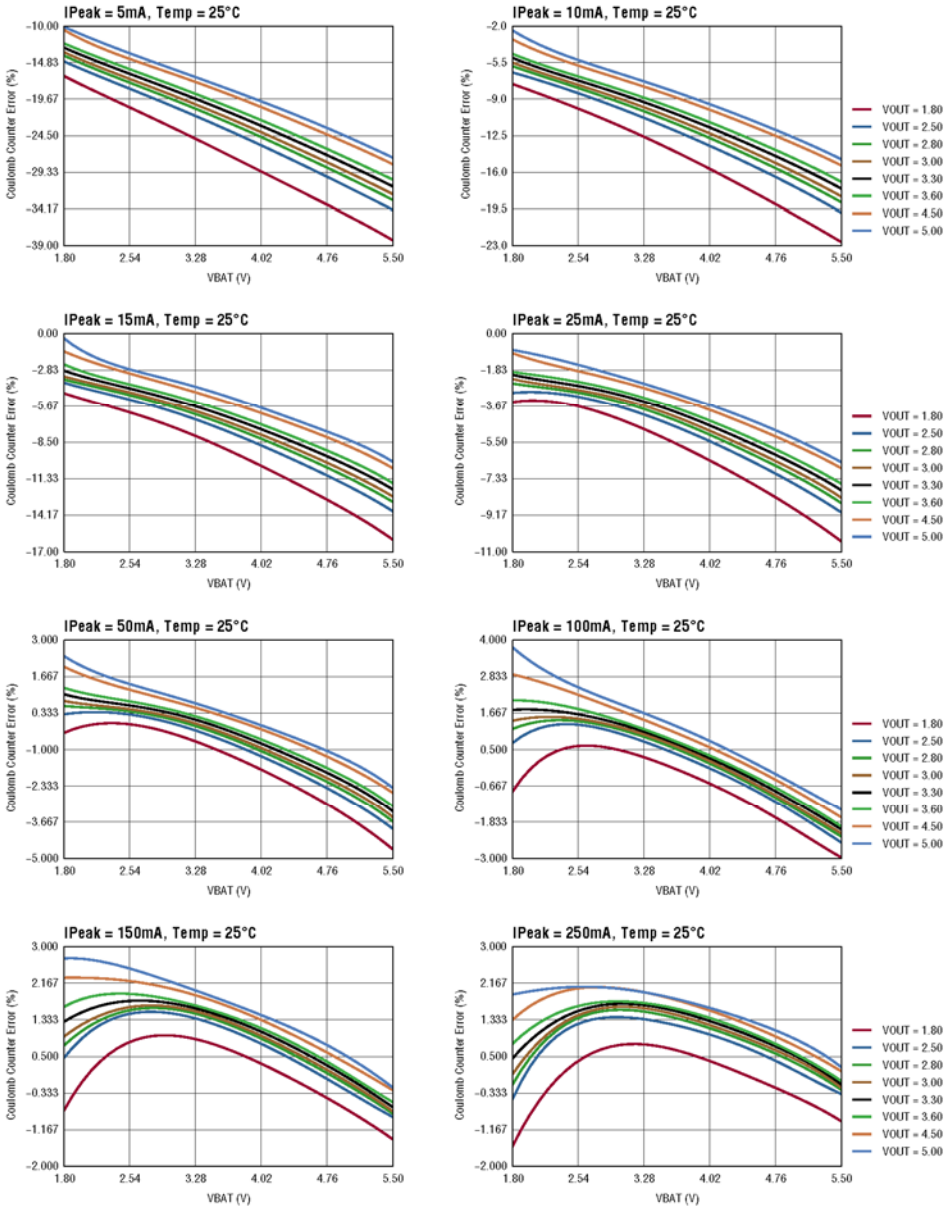

Figure 6 - Typical Error for LTC3335 Configurations Stored as COULOMB_COUNTER_CORRECTION_TABLE in LTC3335.vb

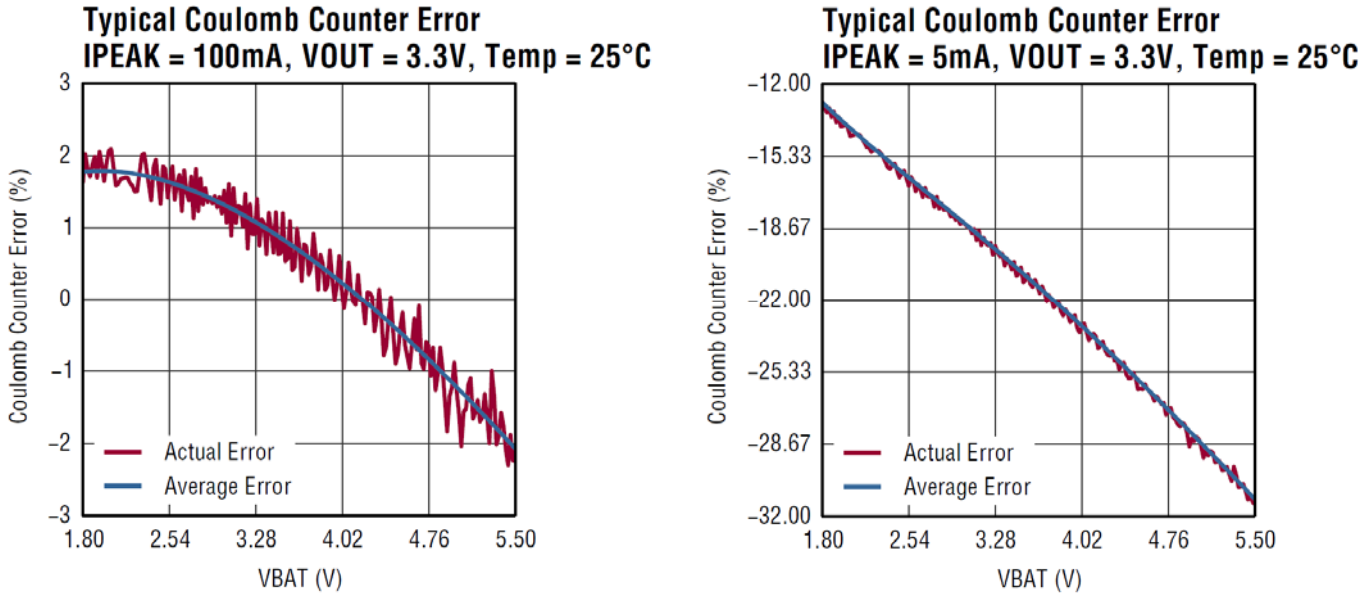

Figure 7 - Average and Actual Coulomb Counter Error

While Figure 4 shows the typical average error for all configurations of the LTC3335, Figure 5 shows that the actual error will vary about the average error. For the case where IPEAK = 100mA, the actual error can vary on the same order as the average error. In the case where IPEAK = 5mA, however, the actual error varies little in comparison to the average error. It is only recommended to use software correction of the coulomb count for conditions where the average error is more than +/-2% so that the effect of the actual error variation is small.

There are two approaches for how software correction can be used to correct the coulomb count of the LTC3335:

1. One Time Correction: The LTC3335 Alarm value can be corrected once by the correction factor, assuming that VBAT will be constant and quiescent current losses are minimal.

2. Continuous Correction: The software can maintain its own coulomb count that is updated each time the LTC3335 accumulator changes, and periodically to account for quiescent current losses.

One time correction of the LTC3335 Alarm value is the simplest method of using software correction of the coulomb count error.

Consider the example of a LTC3335 configured for an IPEAK value of 5mA, used with a 280mAh battery. The best prescaler for this battery would be 7, providing a nominal resolution of 1.1mAh per count of the LTC3335 accumulator. If the user wishes to set the Alarm value to 90% of the battery capacity, the nominal value would be:

$$\frac{280\text{mAh} \times 90\%}{1.1\ \text{mA/count}} = 229\ \text{counts}$$

If the LTC3335 output voltage is 3.3V and the battery has a nominal voltage of 3.7V, however, the Get_Software_Correction_Factor() function returns an expected error of -21.6%. Therefore, adjusting the resolution of the alarm value to reflect the additional 21.6% coulombs per count would result in the LTC3335 presenting an alarm at the desired level of battery discharge:

$$\frac{280\text{mAh} \times 90\%}{1.1\ \text{mA/count} / (100\% - 21.6\%)} = 179\ \text{counts}$$

Continuous correction of the coulomb count should be used if the battery voltage is not expected to be constant during a discharge or if quiescent current losses are not expected to be negligible.

With this method, the software maintains a high resolution coulomb count in software. Each time that the LTC3335 accumulator changes by one count, the software will add the amount of coulombs represented by one count at the current value of VBAT.

Consider the example of a LTC3335 configured for an IPEAK value of 5mA, used with a 1.1mAh battery. The best prescaler for this battery would be 15, providing a nominal resolution of 4.3 µAh per count of the LTC3335 accumulator. As the battery discharges and its voltage changes, different amounts of charge would be added to the software coulomb count. Some examples are shown in Table 1.

VOUT (V)	VBAT (V)	TYP Error (%)	Charge (µAh)
3.30	4.00	-23.10%	5.58
3.30	3.60	-21.13%	5.44
3.30	3.20	-19.26%	5.31
3.30	2.80	-17.47%	5.20

Table 1 - Charge Represented by 1 Accumulator Count

While the LTC3335 quiescent current is impressively low, it may not be negligible for very small batteries or for very long discharges. To account for this, the software should increment its coulomb count at the rate at which its resolution allows.

Consider if a 1.1mAh battery is being represented in software by a 16 bit number, resulting in a resolution of 16.8nAh per count. Since the LTC3335 quiescent current is 680nA, the software should increment its coulomb count by 1 every 88.8 seconds:

$$\frac{1.1\text{mAh} \times 3600\ \text{s/h}}{(2^{16}-1) \times 680\text{nA}} = 88.8\ \text{seconds}$$

Figure 6 shows an example of continuous correction being performed on the LTC3335 coulomb count as a 1.1mAh battery is discharged using an IPEAK value of 5mA and VOUT of 3.3V. If the accumulator was used without correction, the LTC3335 would under-count the discharged coulombs by -18%. Using software correction, however, the discharged coulombs are slightly over-counted by 1.5%

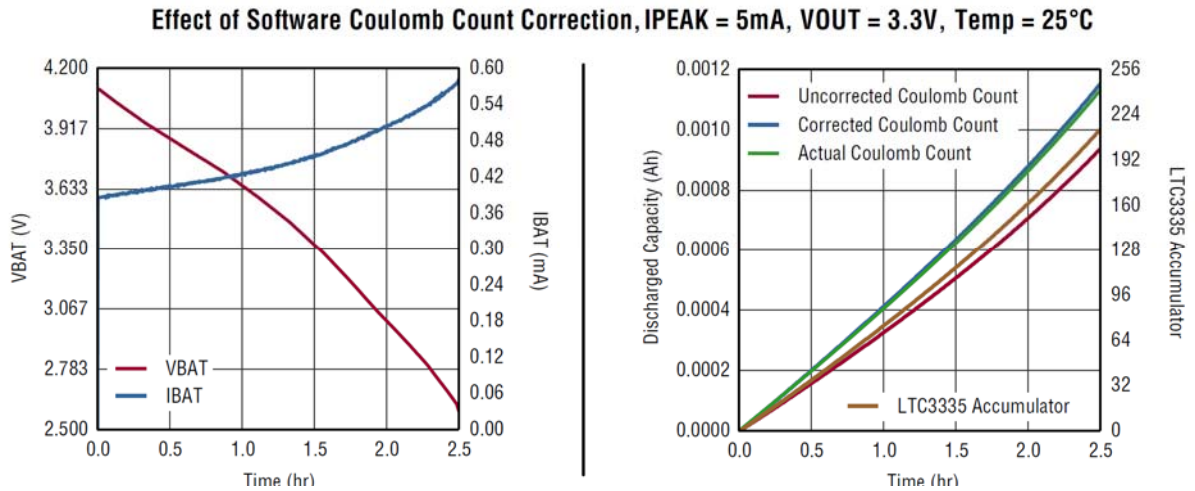

Figure 8 - Continuous Coulomb Count Correction during Discharge of 1.1mAh Battery