User's Guide

MSPM0 Gauge L2 Solution Guide

ABSTRACT

This application note describes the level2 gauge implementation based on MSPM0. The document provides a leading long time state-of-charge (SOC) accuracy by utilizing a data fusion method to combines the SOC information sourced from coulometer (current) and OCV (voltage). This algorithm can also automatically compensate for cell aging, temperature, and current rate while providing accurate state-of-charge (SOC) in percentage (%) over a wide range of operating conditions.

Implementation features, hardware introduction, GUI introduction, software introduction, and evaluation are included in this application note. Project collateral detailed in this application note can be downloaded from MSPM0 Gauge L2 Development Package.

Table of Contents

List of Figures

List of Tables

Trademarks

LaunchPad™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

There are different gauge implementations based on MSPM0. Table 1-1 shows a comparison for customers to choose.

Table 1-1. MSPM0 Gauge Implementation Comparison

	MSPM0 Gauge L1	**MSPM0 Gauge L2**
Detected parameters	Voltage; temperature	Voltage; temperature; current
Output key parameters	SOC	SOC; SOH; Remain capacity; Cycles
Used methods	Volt Gauge	Coulomb counting + volt gauge + mixing + capacity learn
Key technologies	Battery model	Battery model + data fusion + empty or full compensation+ core temperature evaluation
Application	Output step data with low SOC accuracy	Output percentage data with high SOC accuracy
Battery type	LiCO2, LiMn2O4	LiCO2, LiMn2O4, LiFePO4

MSPM0 Gauge L2 is a pure software algorithm, provided in software lib type and has large flexibility on MCU platform, the AFE or the battery selection. Some key features are shown below:

- Support SOC, SOH, capacities and warning flag output with limited parameters input
- Work after MCU power-on without factory calibration or learning cycles
- Have residual SOC and Full SOC learning and compensation for cell aging, temperature, and current rate
- Have SOC and FullCap measurement error reduction with the data fusion method
- Have battery core temperature estimation to handle low temperature discharge
- Support data saving for reloading

Here is the summary of the MCU resource requirement for this Gauge L2 algorithm.

	Single Cell	**Multiple Cells**
Flash	Optimization level 0: approx. 14.8k Optimization level 0: approx. 13.6k	Optimization level 0: approx. 14.8k Optimization level 0: approx. 13.6k
RAM	1.54k	2 cells: 2.04k 3 cells: 2.54k N cells: 1.04k + 0.5*Nk
Algorithm running time based on M0L	3ms	N*3ms
MSPM0 platform	MSPM0L, MSPM0C, MSPM0H, MSPM0G	
Examples	MSPM0L1306 Gauge board + 1 LiCO2 battery (Section 5.1) MSPM0G3507 Launchpad+ BQ76952 EVM + 4 LiFePO4 batteries (Section 5.2) MSPM0L1306 Gauge board + BQ76905 EVM (Section 5.3)	

The design is combined of three parts: hardware, software and GUI. These can be found at MSPM0 Gauge L2 Development Package.

- The hardware board is used to detect voltage, current and temperature, which are input into algorithm to calculate SOC. As described for different hardware setup details, refer to Section 5.
- The software project includes the used gauge algorithm, MCU control and AFE communications. For the description of algorithm, refer to Section 2. For the typical usage case, see Section 5.
- The GUI is written by python, which can be used to communicate with the gauge board, run test pattern, and do data analysis. For GUI introduction, refer to Section 3.

2 Algorithm Introduction

In the following section, a basic introduction on battery knowledge, the algorithm used and how to get the wanted SOC is presented before giving a description to the software and the detailed algorithm description,

2.1 Battery Basic Knowledge Introduction

The gauge algorithm is mostly used to tell users the battery working conditions, and reach a balance between outputting max capacity and extending the battery life. The basic control strategy and the battery performance under these two conditions are shown.

Figure 2-1 shows a battery discharge pattern for a one-cell LiCO2 battery and the related concept to introduce. The red line represents the open circuit voltage (OCV), which means the potential difference between the positive electrode (PE) and the negative electrode (NE) when no current flows and the electrode potentials are at equilibrium. OCV can normally be treated to equal to battery voltage after resting the battery for 1-2 hours. The blue line means the run-time cell voltage (Vcell). As the battery has internal resistance, there is a voltage drop between OCV and Vcell with a constant load.

For a LiCO2 battery, due to the chemical limitation, a full charge voltage threshold (for example, 4.2V) and an end of discharge voltage threshold (for example, 3V) is set to avoid irreversible damage on the battery. That means with different discharge current, users can get different capacities from the battery. The output capacity is also influenced by the temperature, as the Rcell gets reduced while the temperature is increasing. In this gauge design, the unchangeable capacity is called NomFullCap, which represents the movable lithium ions in the electrode structure and the quantity does not vary with temperature or C-rate of cell usage. The changeable capacity is called CusFullCap, which means the usable capacity by end users and affected by the battery running conditions and threshold setting.

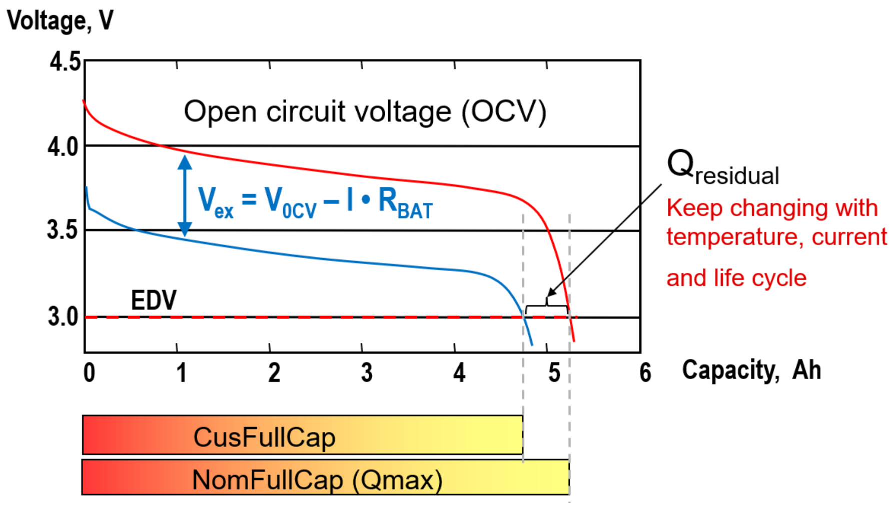


Figure 2-1. Battery Discharge Pattern

Figure 2-2 shows a battery charge pattern for a normal one cell battery. A charging condition can be simplified into a constant current (CC) window and then a constant voltage (CV) window. At the end of charge, if the charge voltage is constant, then the current is reduced in an exponential order. If the current turns to 0, then the NomFullCap is obtained. To avoid waiting for a long time, a terminating charge current is set (for example 1/20 capacity (1/20C)), which causes a little reduction on CusFullCap compared with NomFullCap.

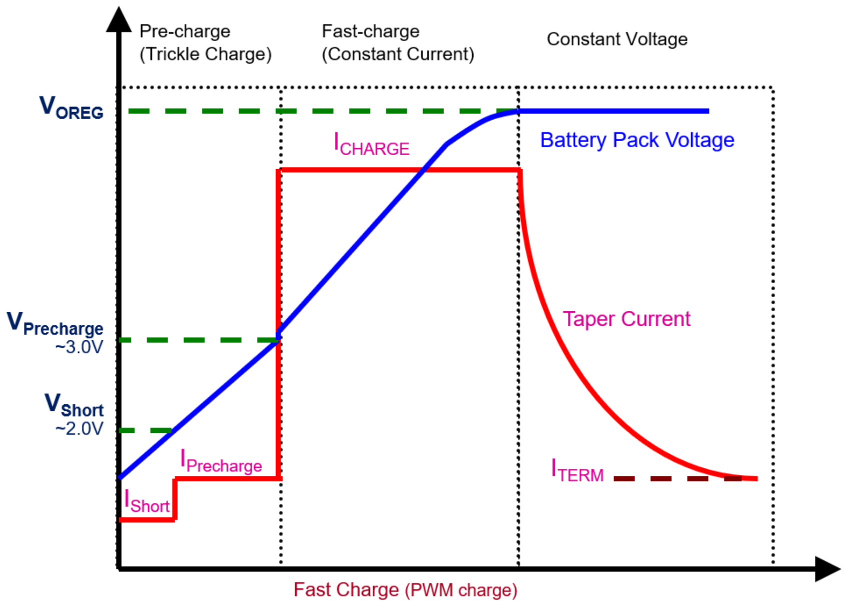


Figure 2-2. Battery Charge Pattern

NomFullCap is obtained from one OCV to another OCV. CusFullCap is from one Vcell to another Vcell. The NomFullCap always covers the CusFullCap.

In this gauge algorithm, the NomFullCap range is based on the per saved OCV range in the SOC-OCV table (circuitParamsTable). The CusFullCap range is based on the MaxFullChgVoltThd and EmptyDhgVoltThd setting, and also changes after self-learning. To leave some margin, TI recommends to make the OCV range for NomFullCap to be a little larger that the voltage range for CusFullCap.

2.2 Different SOCs and Used Technologies

The previous section shows a basic idea about Capacity and OCV. The following sections introduce State-of-charge (SOC), which is finally wanted by end users. Equations are also used to help you under the SOC concept differences. For more about the SOC behaviors, please refer to Section 4.5.

2.2.1 NomAbsSoc Calculation

If users use the NomFullCap as the normalization factor of the available capacity to the capacity use from a full battery, then get the normalized absolute State of Charge (NomAbsSoc). This represents the remain percentage of moveable lithium ions in the negative-electrode solid particles. In this algorithm, two strategies are used to calculate NomAbsSoc, which is detailed in the next section. The first is a coulometer with OCV calibration. The second is a battery model filter. Use a data fusion method to further reduce the NomAbsSoc evaluation error.

2.2.1.1 Coulometer With OCV Calibration

The common method to update NomAbsSoc is to use coulometer, which is shown in Equation 1 and Equation 2.

$$\mathrm{Quse} = \mathrm{I(t)}^{*}\Delta\mathrm{t} \tag{1}$$

$$\mathrm{NomAbsSoc} = \frac{\mathrm{NomFullCap} - \mathrm{Quse}}{\mathrm{NomFullCap}} \tag{2}$$

As coulometer has error accumulation problems. NomAbsSoc is purely calibrated by using the OCV, which is determined after the battery is rested for enough time. An OCV to SOC search table example is shown in Figure 2-3.

$$\mathrm{NomAbsSoc} = \mathrm{f(OCV)} \tag{3}$$

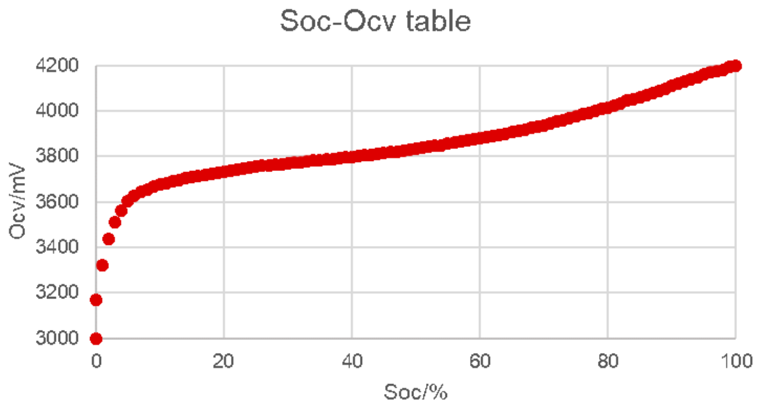


Figure 2-3. SOC-OCV Table

Equation 2 is used for run time output NomAbsSoc and Figure 2-1 is used to periodically calibrate NomAbsSoc. After two more calibrations, users can get the delta capacity and delta NomAbsSoc. Then, calculate the NomFullCap, as shown in Equation 4.

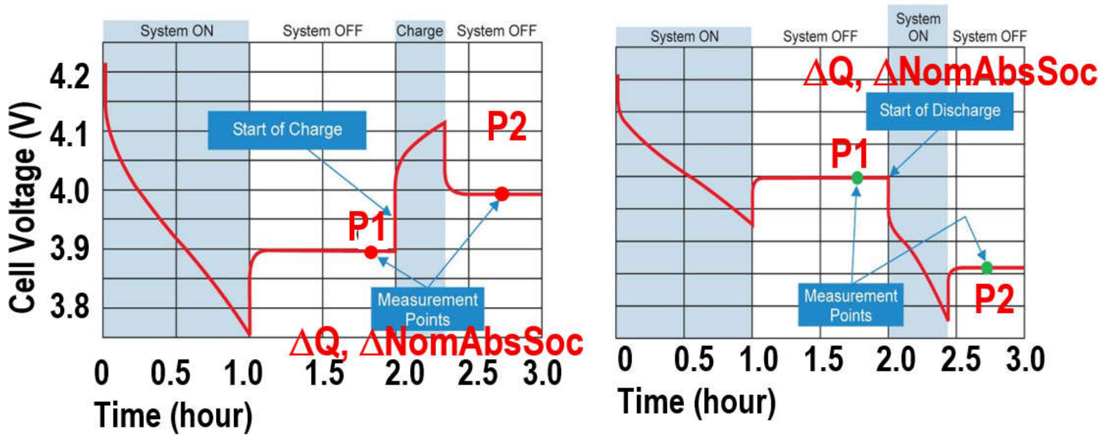


Figure 2-4. OCV Calibration and Capacity Accumulation

$$\mathrm{NomFullCap} = \frac{\sum \mathrm{ABS}(\Delta \mathrm{Q})}{\sum \mathrm{ABS}(\Delta \mathrm{NomAbsSoc})} \tag{4}$$

For a real battery, the NomFullCap slightly decreases due to the battery getting old. To track the capacity decline issue, periodically calibrate the NomFullCap. Equation 5 is used to represent the capacity decline, named with State-of-Health (SOH). However, in real applications, as the obtained NomFullCap has error, use the maximum obtained NomFullCap is used as the Max NomFullCap.

$$\mathrm{SOH} = \frac{\mathrm{NomFullCap[n]}}{\mathrm{Max(NomFullCap[n])}} \tag{5}$$

2.2.1.2 Data Fusion

As seen in Equation 4, in a real application, the calibrated NomAbsSoc is affected by the voltage detection accuracy, battery rest time and SOC-OCV table accuracy, which is especially critical on the LiFePO4 battery. The DeltaQ is influenced by the shunt resistor accuracy and ADC performance. This means the calibrated NomAbsSoc and the NomFullCap all include some errors. In the Gauge Level2, use a data fusion method, which has the same concept with the kalman filter. TI gives weight to all the error sources, including measured current, measure voltage, evaluated OCV and so forth. After that, users know the weight of the SOC generated through coulometer and the SOC generated through OCV calibration. At last, users can obtain a more accurate NomAbsSoc after combining these two SOCs together, as shown in Figure 2-5.

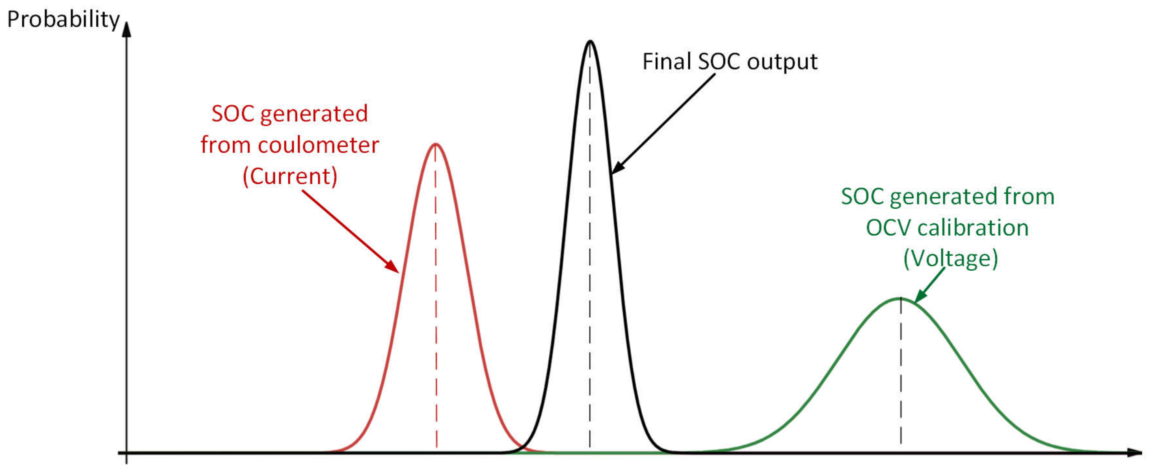

Probability
SOC generated from coulometer (Current)
Final SOC output
SOC generated from OCV calibration (Voltage)

Figure 2-5. Data Fusion

Figure 2-6 shows the data fusion algorithm performance based on a LiFePO4 battery simulation model, with 3000 random DHG and CHG points, after considering the detection error. The AbsNomSoc error across the battery life is controlled within 3.5%, and the NomFullCap error is controlled within 4%. Remember that the result is only used to show the algorithm capability to detect NomAbsSoc with a limited condition and this does not make sure of the error range of the algorithm in a real application to detect CusRltSoc.

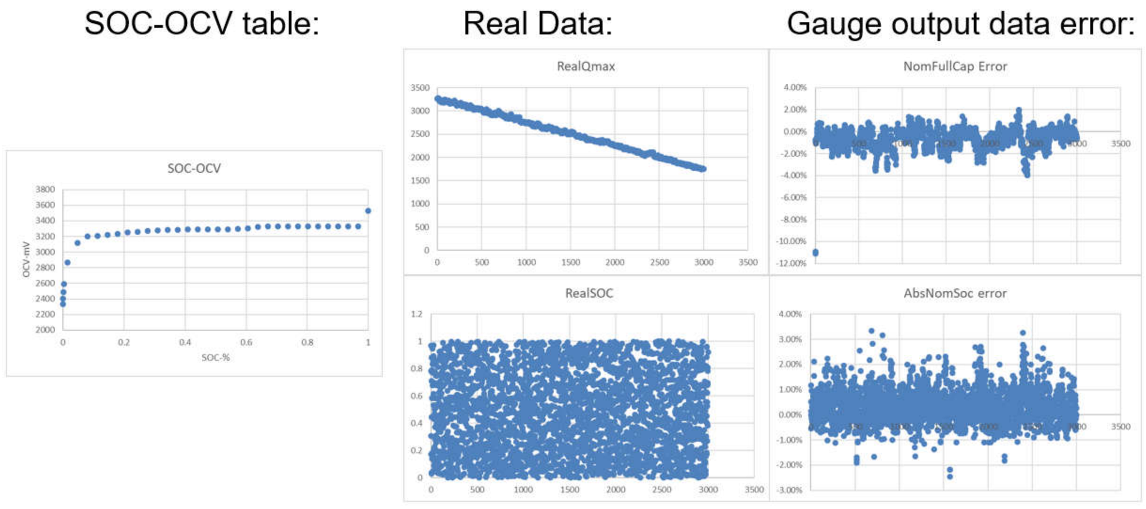


Figure 2-6. Algorithm Performance (By Simulation)

2.2.1.3 Battery Model Filter

In Section 2.2.1.1, NomAbsSoc is evaluated based on purely one battery parameter (voltage or current accumulation). One method is to use the relationship between OCV (positive and negative electrodes balance) and NomAbsSoc (different lithium ion concentrations). Another method is to use the relationship between coulometer (electron numbers) and capacity (movable lithium ion numbers). This strategy requires less computing resources, as only a coulometer is needed to work in every cycle. However, this needs to wait until the NomFullCap is generated for 1-2 battery cycles.

Another strategy is used to create a model or a filter to evaluate the NomAbsSoc or even the CusRltSoc based on all the input parameters, like current, voltage, time and temperature. The common methods are equivalent-circuit model, Kalman-filter, neural network and so on. The accuracy of the SOC output mostly lies on the model accuracy. However, more complex model means more MCU computing resources.

An economic method is to use a simplest equivalent-circuit model (a first order RC model), shown in Figure 2-7, to output NomAbsSoc with only voltage input.

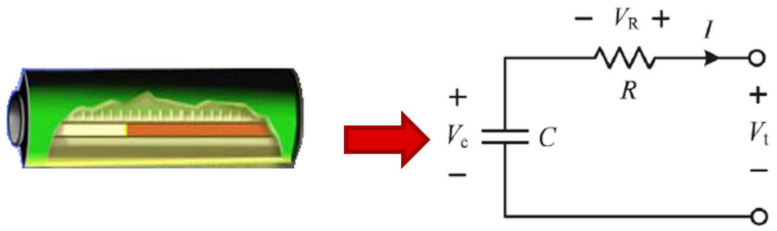


Figure 2-7. Battery Model

Figure 2-8 shows the software flow chart of VGauge. However, in this design, the SOC lookup table function only needs to be used to search the wanted parameters in this battery model. The SOC part comes out from IGauge.

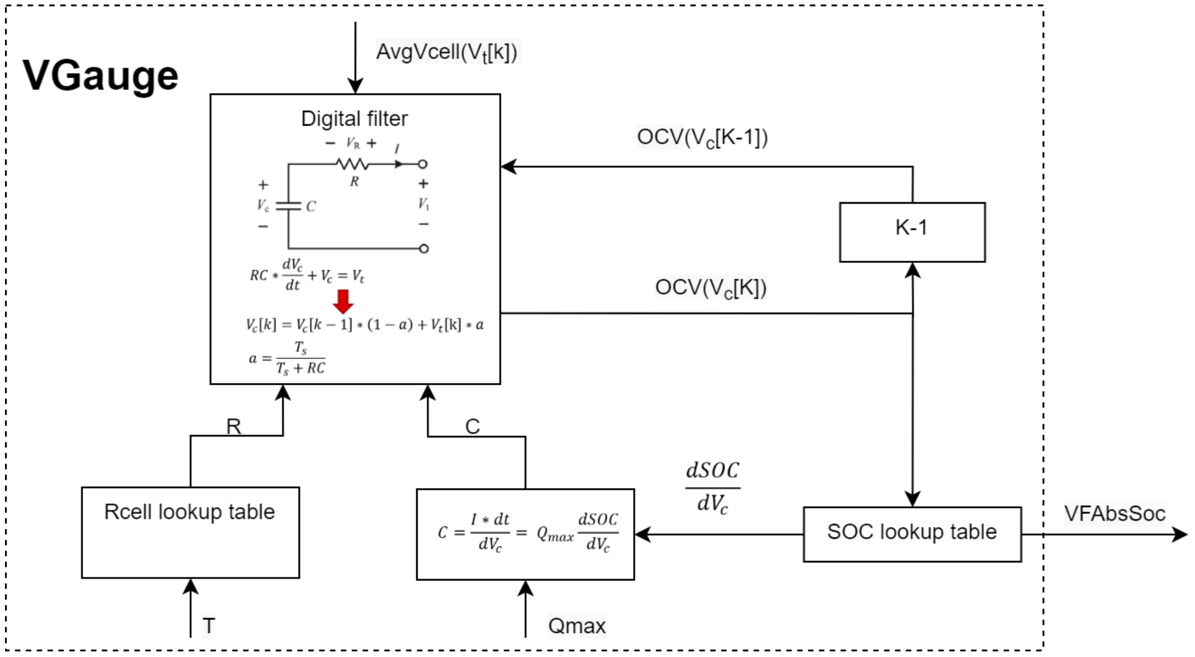

VGauge
AvgVcell(V_t[k])
Digital filter
$RC * \frac{dV_c}{dt} + V_c = V_t$
$V_c[k] = V_c[k-1] * (1-a) + V_t[k] * a$
$a = \frac{T_s}{T_s + RC}$
OCV(V_c[K-1])
K-1
OCV(V_c[K])
R
C
Rcell lookup table
$C = \frac{I * dt}{dV_c} = Q_{max} \frac{dSOC}{dV_c}$
$\frac{dSOC}{dV_c}$
SOC lookup table
VFAbsSoc
T
Qmax

Figure 2-8. VGauge Software Flow

For more about the NomAbsSoc accuracy outputted from VGauge, see MSPM0 Gauge L1 Solution Guide, application note.

2.2.2 CusRltSoc Calculation

For real applications, the wanted SOC by customer is not NomAbsSoc, because the current cannot be controlled at 0 when the battery full and battery empty state is reached, see in Figure 2-2 .

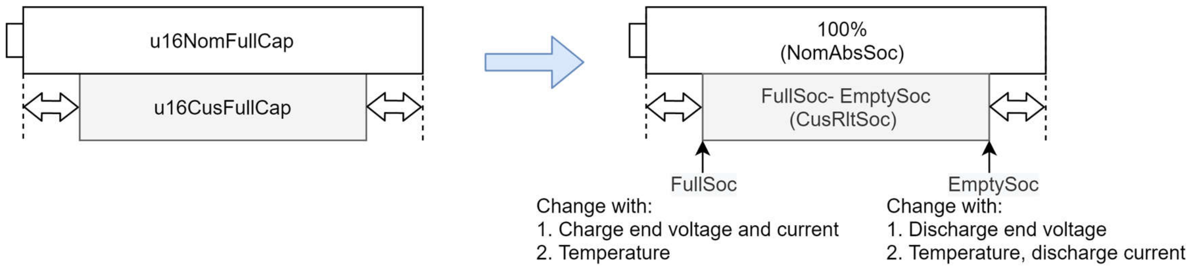


Figure 2-9. Capacity to SOC

Based on the normalization factor: NomFullCap, three new SOCs are gotten to represent the unchangeable capacity, see in Figure 2-9. FullSoc is used to represent the uncharged NomAbsSoc threshold and EmptySoc is used to represent the undischarged NomAbsSoc threshold. CusRltSoc represents the relative SOC, normalized by FullSoc and EmptySoc based on battery conditions. The equations to calculate the CusRltSoc is shown in Equation 6.

$$\text{CusRltSoc} = \frac{\text{NomAbsSoc} - \text{EmptySOC}}{\text{FullSoc} - \text{EmptySoc}} \tag{6}$$

2.2.2.1 EmptySoc and FullSoc

To get the CusRltSoc, get FullSoc and EmptySoc at first. As seen in Figure 2-1, the FullSoc and EmptySoc (Residual Soc) are influenced by the battery resistance and current. That means in different conditions, there are different FullSoc and EmptySoc.

In this gauge algorithm, update and compensate these two SOCs under different battery conditions, including cell aging, temperature, and current rate. A current-temperature table *AbsEmptySocMatrix* is made to simulate the influence of current and temperature on EmptySoc as shown in Figure 2-10. A temperature table *AbsFullSocMatrix* is made to simulate the influence of temperature on FullSoc.

In the *AbsEmptySocMatrix*, one EmptySoc value is used to cover all the real EmptySoc when the battery works in a CT table block range. For example, if TempThd[1]<Tcell< TempThd[0], and CurtThd[0]<Icell<CurtThd[1], EmptySoc[4] is used to represent all the EmptySoc under this condition. With that setting, in one block, the real EmptySoc of the left bottom corner is minimum and the real EmptySoc of the right top corner is maximum. Users need to adjust TempThd[] and CurtThd[] in battParamsCfg according to the application or test results.

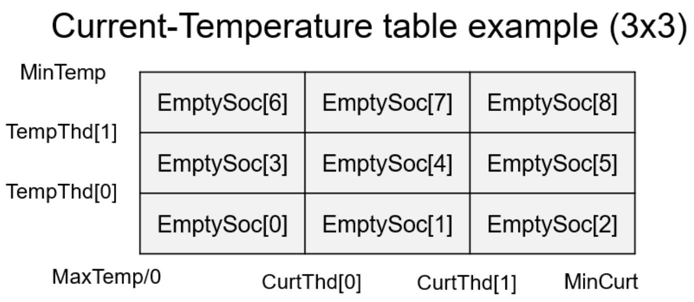


Figure 2-10. CT Table Example

For FullSoc, when the battery is fully charged, the calibrated NomAbsSoc is recognized as the new FullSoc updated into the current-temperature table *AbsFullSocMatrix* under a different temperature, which same as EmptySoc.

EmptyOcvMatrix[] and FullOcvMatrix[] are provided for users to set the beginning values for *AbsEmptySocMatrix* and *AbsFullSocMatrix* if low SmooRltSoc error in the learning cycles is needed. Otherwise, *AbsEmptySocMatrix* uses the SOC related to the OCV equals to EmptyDhgVoltThd as the default value. *AbsFullSocMatrix* uses the SOC related to the OCV equals to MaxFullChgVoltThd as the default value.

Note

To get the fast response to the full charge and empty discharge threshold, the gauge algorithm uses the raw current, voltage and temperature to handling EmptySoc and FullSoc output. If the raw data has large noise, then users need to consider to add additional filters on the battery information input like an IIR filter.

2.2.2.2 Core Temperature Evaluation

In common conditons, the measured temperature is the battery surface temperature. However, the core temperature is more related to the battery resistance change. This temperature gap between the battery core temperature and the battery surface temperature causes the error on FullSoc and EmptySoc evaluation, especially in low temperature discharge.

To get a more accurate EmptySoc and FullSoc, a thermal model is used to evaluate battery core temperature. Users need to input battModelUse parameters and the battWeight parameters. The supported batteries are 18650, 21700, 26650, pouch and prismatic types. A result example is shown in Figure 2-11.

When a module is used, the suggested iq15Tcell_C temperature input resolution is 0.1°C. With 1°C resolution, the iq15EvlCoreTemp_C error is about ±1.5°C with a high discharge current. When no module is used, surface temperature is used as the battery core temperature.

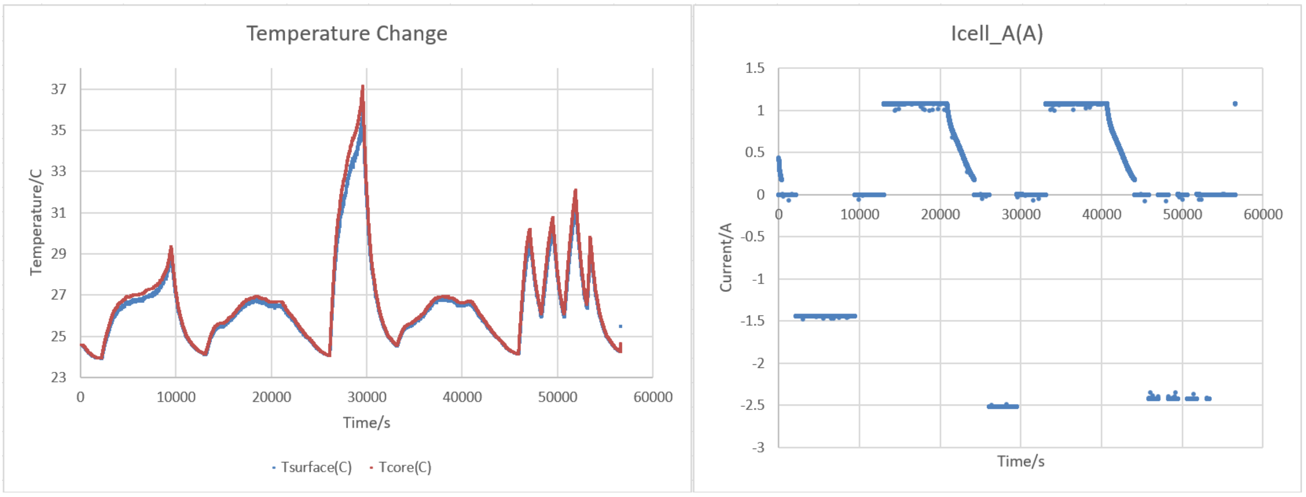


Figure 2-11. Battery Core Temperature Evaluation

2.2.3 SmoothRltSoc Calculation

The blue line in Figure 2-12 shows a CusRltSoc reaction under a battery charge and discharge test pattern. See the data jumps caused from OCV calibration or EmptySoc change.

To make the output SOC more acceptable for customers, a SmoothRltSoc is created to make the SOC output to be constant and no sudden jump. For the realizing method, see Section 2.3.4.

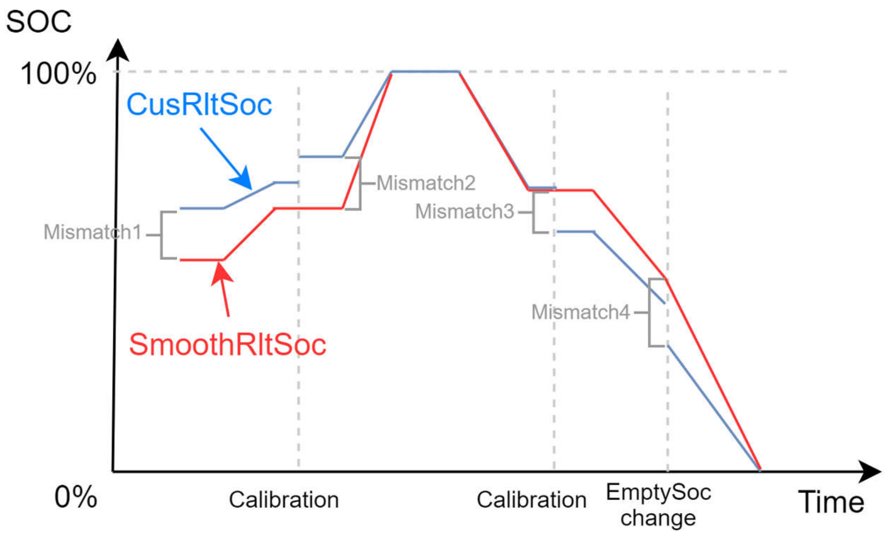


Figure 2-12. CusRltSoc to SmoothRltSoc

The SmoothRltSoc is obtained by tacking the change of the CusRltSoc with the same target 0% or 100% without sudden change. However, if CusRltSoc suddenly changed to 0% or 100%, then SmoothRltSoc changes to 0% and 100% at the same time. CusRltSoc can be above 0% and 100%. SmoothRltSoc is limited between 0% and 100%.

2.3 Algorithm Overview

This section provides an overview of the introduced gauge algorithm as shown in Figure 2-13. This is only for the battery cell algorithm. Battery pack algorithms are just a combination of battery cell algorithms.

This algorithm is based on the coulometer, paired with other methods described before to solve the limitation. This is a combination of four parts. The Capacity Learn part is used to detect the battery rest, do OCV calibration and calculate SOH. The VGauge part is used to output the related parameters from the saved class-one battery model. IGauge is a coulometer used to accumulate the capacity. Mixing part is used to calculate and output NomAbsSoc, CusRltSoc and SmoothRltSoc to customers.

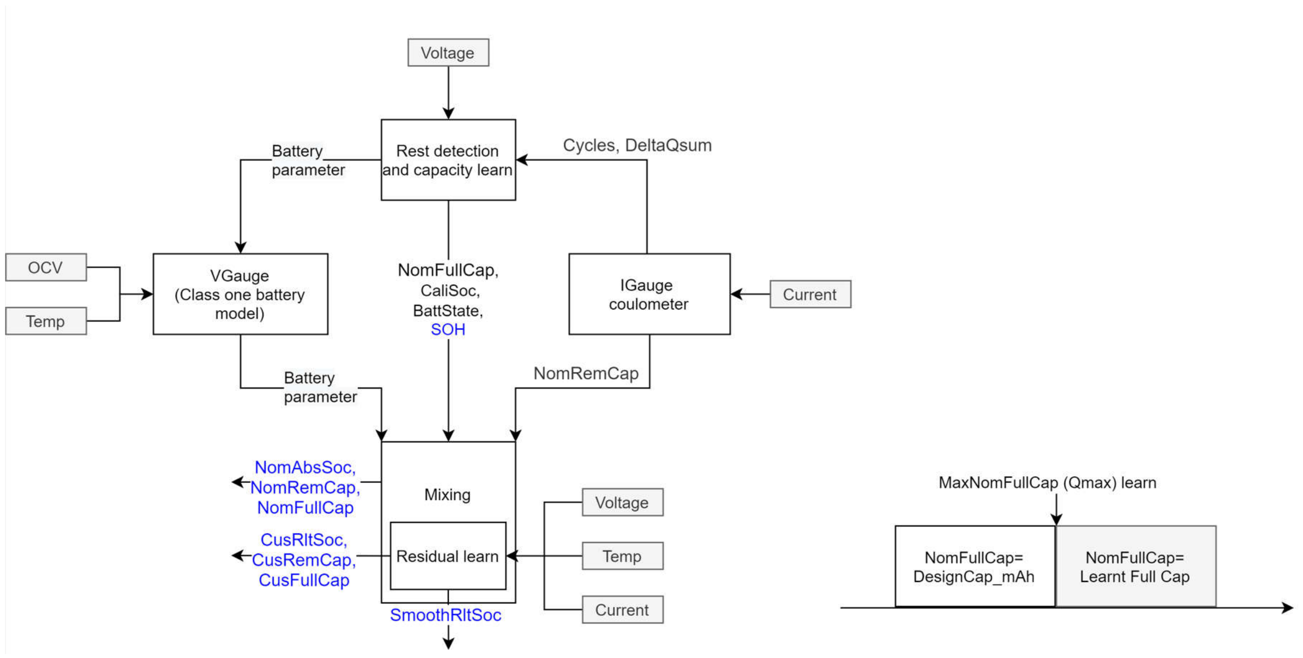

Voltage
Rest detection and capacity learn
Battery parameter
Cycles, DeltaQsum
OCV
Temp
VGauge (Class one battery model)
NomFullCap, CaliSoc, BattState, SOH
IGauge coulometer
Current
Battery parameter
NomRemCap
NomAbsSoc, NomRemCap, NomFullCap
Mixing
CusRltSoc, CusRemCap, CusFullCap
Residual learn
Voltage
Temp
Current
SmoothRltSoc
MaxNomFullCap (Qmax) learn
NomFullCap= DesignCap_mAh
NomFullCap= Learnt Full Cap

Figure 2-13. Algorithm Overview

In the following section, a description for every algorithm part and the key parameters outputted to GUI or other functions are provided.

2.3.1 Voltage Gauge Introduction

The VGauge is just used for other function block diagrams to search the battery parameters from a saved battery model: *circuitParamsTable*. For more details about the battery model, refer to Section 4.2.2.

2.3.2 Current Gauge Introduction

The current gauge is just a simple coulometer and accumulates different types of capacities used for different functions.

The key output parameters of the function are shown in Table 2-1.

Table 2-1. IGauge Key Parameters

Parameter	Comment
u16DhgCycles	Used to tell the user how many cycles the battery has experienced. This only accumulates current when discharging the battery. The value increases when the accumulated capacity is above the NomFullCap.
iq15DeltaQsum	Used for NomFullCap (Qmax) calculation in CapacityLearn algorithm part.
16IF_NomRemCap_mAh	Runtime calculate and report NomRemCap based on the detected current. This is calibrated after getting the OCV.

2.3.3 Capacity Learn Introduction

The capacity learn part has three functions:

- First, is to update battery state, which is used to tell when to do OCV or SOC calibration.
- Second, is to calibrate OCV and SOC. This does the calibration when the battery rest for some time and the voltage drop goes to an acceptable range.
- Third, is to get NomFullCap and SOH using the method shown in Section 2.2.1.1.

The key output parameters of the function are listed in Table 2-2.

Table 2-2. Capacity Learn Key Parameters

Parameter	Comment
iGuageDominationFlg	When this flag is set, the u16MaxNomFullCap_mAh is generated and the SOC accuracy goes to an acceptable accuracy
iq15CaliSoc_DEC	This parameter is used by Capacity learn part to record as the calibrated NomAbsSoc for further NomFullCap calculation, after ocvCaliFinishFlg is set. This is also used by Current gauge part to recalculate NomRemCap in the same cycle.
u16CaliOcv_mV	Used to calibrate NomAbsSoc after ocvCaliFinishFlg is set.
u16NomFullCap_mAh	This parameter is used by Capacity learn part to calculate SOH. This is also used by Current gauge part to calculate NomRemCap and discharge or charge cycles.
u16MaxNomFullCap_mAh	This parameter is fixed once the first u16NomFullCap is get and used to calculate SOH.
iq15SOH_DEC	Used to show the capacity decrease.

2.3.4 Mixing Introduction

Mixing algorithm is used to calculate different types of SOC and capacity.

NomAbsSoc and NomFullCap directly comes from IGauge or CapacityLearn.

The key output parameters of the function are listed in Table 2-3.

Table 2-3. Mixing Key Parameters

Parameter	Comment
iq15CusRltSoc	The relative SOC that closely tack the change of EmptySoc and FullSoc, based on NomAbsSoc.
SmoothRltSoc	Used to remove the sudden jump of CusRltSoc.
i16CusRemCap_mAh	The remain capacity can be used by customer with current load and temperature.
u16CusFullCap_mAh	The full capacity can be used by customer with current load and temperature.

Table 2-3. Mixing Key Parameters (continued)

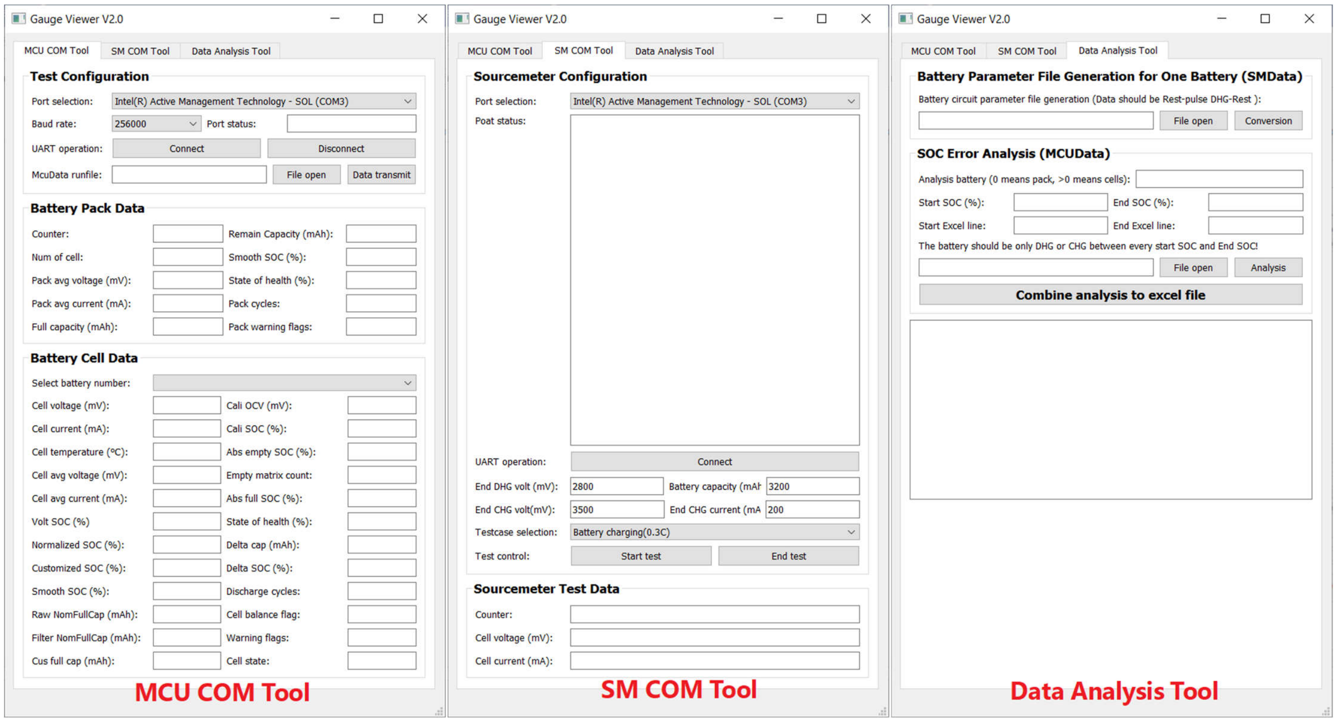
Parameter	Comment
i16NomRemCap_mAh	The remain capacity of battery based on circuitParamsTable range.
u16NomFullCap_mAh	The full capacity of battery based on circuitParamsTable range.
iq15AbsEmptySocMatrix	The matrix to record different emptySoc under different temperatures and current. If not initialization, then this learned automatically.
iq15AbsFullSocMatrix	The matrix to record different fullSoc under different temperatures and is learned automatically.

3 Gauge GUI Introduction

Gauge GUI is also an important part of this implementation as shown in Figure 3-1. This can be used to record MCU data, run battery test case, and do data conversion. This GUI has three pages.

- First, is MCU COM Tool, used to communicate with MSPM0 and record the MCU transmitted battery running data.
- Second, is SM COM Tool, used to communicate with the source meter, run battery test case and record the test data sent from the source meter.
- Third, is Data Analysis Tool, used to generate battery parameters and evaluate the SOC error.

Be attentive when using GUI executing file to evaluate the implementation. The GUI takes 2-3 minutes to start due to Python's limitation. Instead, the GUI source file has a faster GUI start-up speed. If users want to customize the battery test cases under the SM COM Tool, then TI recommends to use the source code. For more details about how to use the GUI, refer to the next sections.



MCU COM Tool:
Port selection: Intel(R) Active Management Technology - SOL (COM3)
Baud rate: 256000

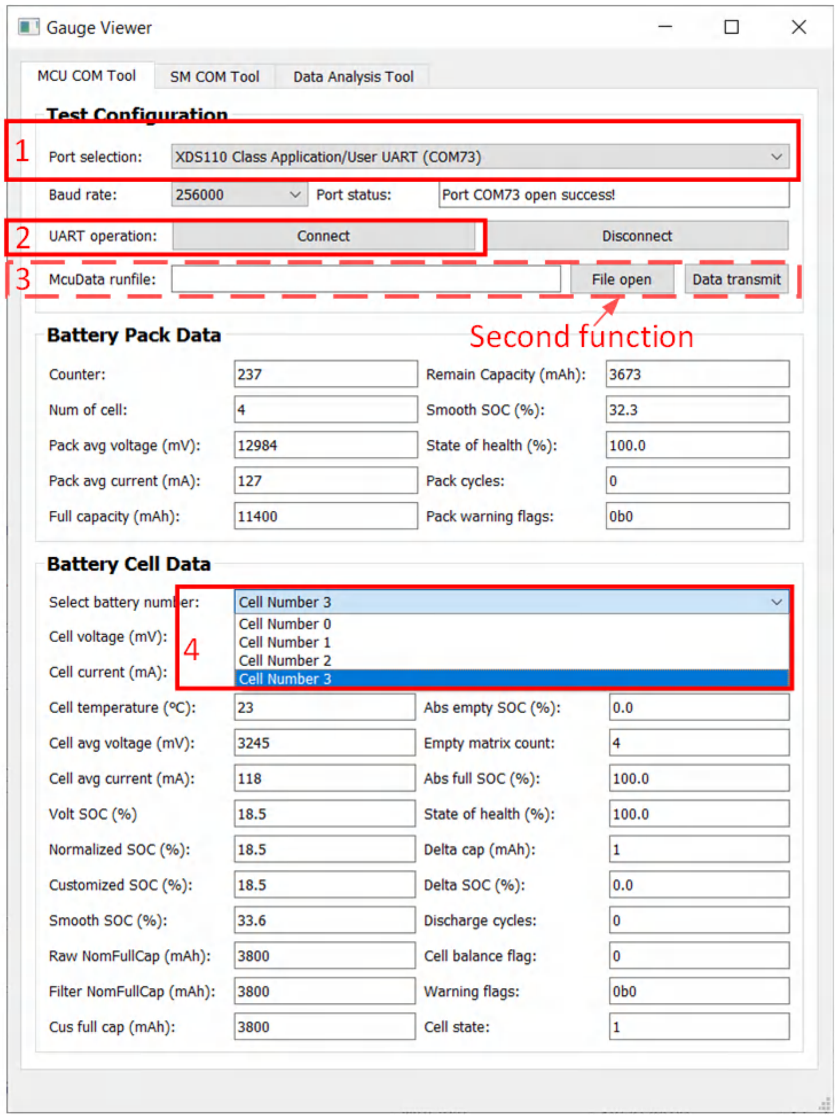
SM COM Tool:
Port selection: Intel(R) Active Management Technology - SOL (COM3)
End DHG volt (mV): 2800
Battery capacity (mAh: 3200
End CHG volt(mV): 3500
End CHG current (mA: 200
Testcase selection: Battery charging(0.3C)

Figure 3-1. Gauge GUI Functions

3.1 MCU COM Tool

The MCU COM Tool, shown in Figure 3-2, is commonly used by end users. This has two functions:

- The default function is to receive the battery running data from MCU. The default shown data is pack data and cell 1 data. If cell number above 1, then users can choose the wanted number following the steps in Figure 3-2. The data is saved automatically in CSV with a name *time-McuData.csv* after the test is finished or the test is stopped.
- The second function is to load the selected *time-McuData.csv* CSV file and transmit the cell current, cell voltage and cell temperature data in this file to MCU for algorithm running when paired with the related gauge mode (communication data input mode).



Test Configuration

Port selection:	XDS110 Class Application/User UART (COM73)		
Baud rate:	256000	Port status:	Port COM73 open success!
UART operation:	Connect		Disconnect
McuData runfile:			

Battery Pack Data

Counter:	237	Remain Capacity (mAh):	3673
Num of cell:	4	Smooth SOC (%):	32.3
Pack avg voltage (mV):	12984	State of health (%):	100.0
Pack avg current (mA):	127	Pack cycles:	0
Full capacity (mAh):	11400	Pack warning flags:	0b0

Battery Cell Data

Select battery number: Cell Number 3
Cell Number 0
Cell Number 1
Cell Number 2
Cell Number 3

Cell voltage (mV):
Cell current (mA):

Cell temperature (°C):	23	Abs empty SOC (%):	0.0
Cell avg voltage (mV):	3245	Empty matrix count:	4
Cell avg current (mA):	118	Abs full SOC (%):	100.0
Volt SOC (%)	18.5	State of health (%):	100.0
Normalized SOC (%):	18.5	Delta cap (mAh):	1
Customized SOC (%):	18.5	Delta SOC (%):	0.0
Smooth SOC (%):	33.6	Discharge cycles:	0
Raw NomFullCap (mAh):	3800	Cell balance flag:	0
Filter NomFullCap (mAh):	3800	Warning flags:	0b0
Cus full cap (mAh):	3800	Cell state:	1

Figure 3-2. MCU COM Tool Functions

3.2 SM COM Tool

The SM COM Tool, shown in Figure 3-1, is used to control the source meter to run the battery test case, and show the data measured by source meter. The record data is saved in the CSV file with a name *time-SmData.csv*. If users want to recreate this part for software, then install NI_VISA. For hardware, users need a USB to rs232 wire and a keithley 2602A source meter.

As the source meter is not an equipment solely made for BMS, consider the limitations on the BMS test. For example, the source meter cannot support high current control with high voltage battery. This can trigger the AFE into protection mode with constant current control because the battery voltage changes above the threshold set in the AFE.

3.3 Data Analysis Tool

Data Analysis Tool is used to do data conversion and data analysis work.

The first function is the battery parameter file generation as shown in Figure 3-3. This is used to abstract battery circuit information from the pulse discharge datas (SMData file). Follow the steps to get the battery parameter file. For more details, refer to the Section 4.2.1.

The second function is to analyze the gauge accuracy performance. After a real test, the MCUData file is generated. The SOC is outputted by the gauge at the start of the **Excel** line and there are prior values at the end of the Excel line. The input of the Start SOC and End SOC by users are posterior values. After using the analysis button, the Qmax is calculated by the Start SOC and End SOC input and generates the posterior value for every excel line. In a test pattern, there can be more than one DHG and CHG. That means users also need to do the input in step 2 and step 3 for the same times. Finally, click on step 4 and the GUI outputs the final result.

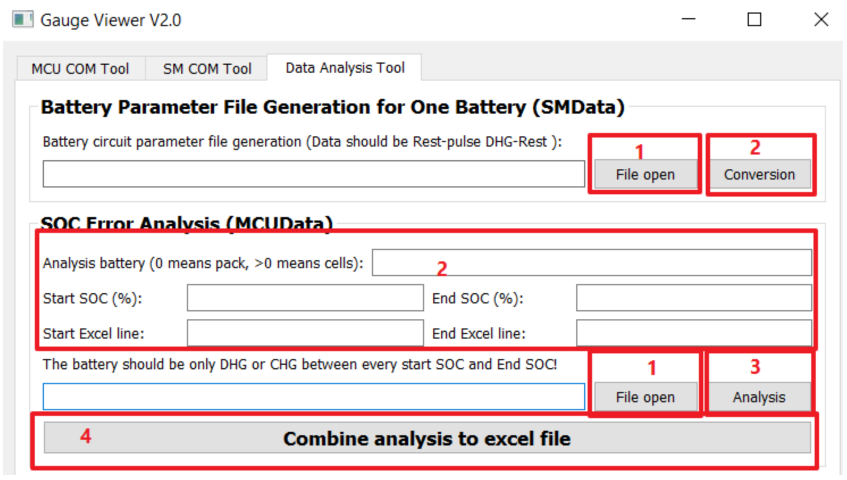

Gauge Viewer V2.0
MCU COM Tool
SM COM Tool
Data Analysis Tool
Battery Parameter File Generation for One Battery (SMData)
Battery circuit parameter file generation (Data should be Rest-pulse DHG-Rest):
1
File open
2
Conversion
SOC Error Analysis (MCUData)
Analysis battery (0 means pack, >0 means cells):
2
Start SOC (%):
End SOC (%):
Start Excel line:
End Excel line:
The battery should be only DHG or CHG between every start SOC and End SOC!
1
File open
3
Analysis
4
Combine analysis to excel file

Figure 3-3. Data Analysis Tool

4 MSPM0 Gauge Evaluation Steps

4.1 Step 1: Hardware Preparation

Hardware board:

If users want to evaluate this implementation, then do the hardware setup first. If users only want to evaluate the gauge software, then only a LaunchPad™ is needed and input the prepared voltage and temperature data into the MSPM0 gauge algorithm.

Test setup:

To do the test and evaluate MSPM0 Gauge performance, users need to prepare a source meter or other battery test machines to control the battery charge and discharge. This is also helpful if users have a thermo stream to evaluate the gauge performance under a different temperature.

4.2 Step 2: Get a Battery Model

The battery model is obtained by the battery parameter calculation from pulse discharge test case. However, for MSPM0 Gauge L1 with low-discharge current in real application, users do not need to do the test. Users can reuse the default model in the code or get a model related to the battery chemistry. For higher level MSPM0 Gauge implementation, as the SOC calibration accuracy lies on the battery model, TI recommends to get the dedicated battery model.

4.2.1 Battery Test Pattern

For the test machine, users can use any machine that can charge and discharge the battery, and the tested data can be recorded. The paired test machine with the supplied GUI is keithley 2602A source meter, which is controlled through a USB to rs232 wire, paired with NI_VISA.

To get a more accurate model, users need to discharge the battery with low current (for example, 0.1C for 20 minutes). The rest time after each pulse needs to be 1-2 hours. Then, users can take the Vcell as OCV. Finally, with this setting, users get about 30 points, which is the minimum data size of SOC-OCV table. TI recommends to reduce the discharge current and discharge time at the beginning and at the end to catch the voltage rapid change and increase accuracy, especially for LiFePO4 battery.

Note

When doing a battery test, the tested battery needs to take the PCB and battery socket influence into consideration. Otherwise the tested resistor is smaller than the real circuit resistor.

Table 4-1 shows a suggested test pattern for LiCO2 and LiMn2O4. For LiFePO4, refer to this as well.

Table 4-1. Battery Test Pattern

Parameter	Value	Comment
Test temperature	Approx. 25°C	
Start voltage	Approx. 4.3-4.4V	Make sure the start voltage is no lower than the application max charge voltage
End voltage	Approx. 2.5-3.0V	Make sure the rest voltage is no higher than the application min discharge voltage
Discharge current	Approx. 0.05C-0.1C (Capacity)	Low current means more points. Recommended to use 0.05C for first and last 5% capacity
Discharge time	Approx. 10-20 minutes	Low discharge time means more point. Recommended to use 10 minutes for first and last 5% capacity
Rest time	1-2 hours	Longer is better. However, 1 hour is enough

Figure 4-1 shows a battery model example test case. This charges the battery to full (4350mV) and rests for 1 hour, with the voltage drops to 4322mV. Then, the battery does a pulse discharge with 20 minutes and rests for 1 hour to get the OCV under different SOC. The test is terminated at 2450mV. After 1 hour rest, the voltage increases to 2864mV. So, the OCV range of the SOC-OCV table is from 4322mV to 2864mV.

Note

For battery test pattern, make sure the tested OCV range is wide enough to avoid that the calibrated OCV beyond the SOC-OCV table range in real applications. The simplest way is to let the OCV range of the SOC-OCV table to cover the battery operation range (MaxFullChgVoltThd to EmptyDhgVoltThd). For example, the SOC-OCV table (OCV range is from 4322mV to 2864mV) can be an excellent choice for a battery which operates between 4200mV and 3000mV.

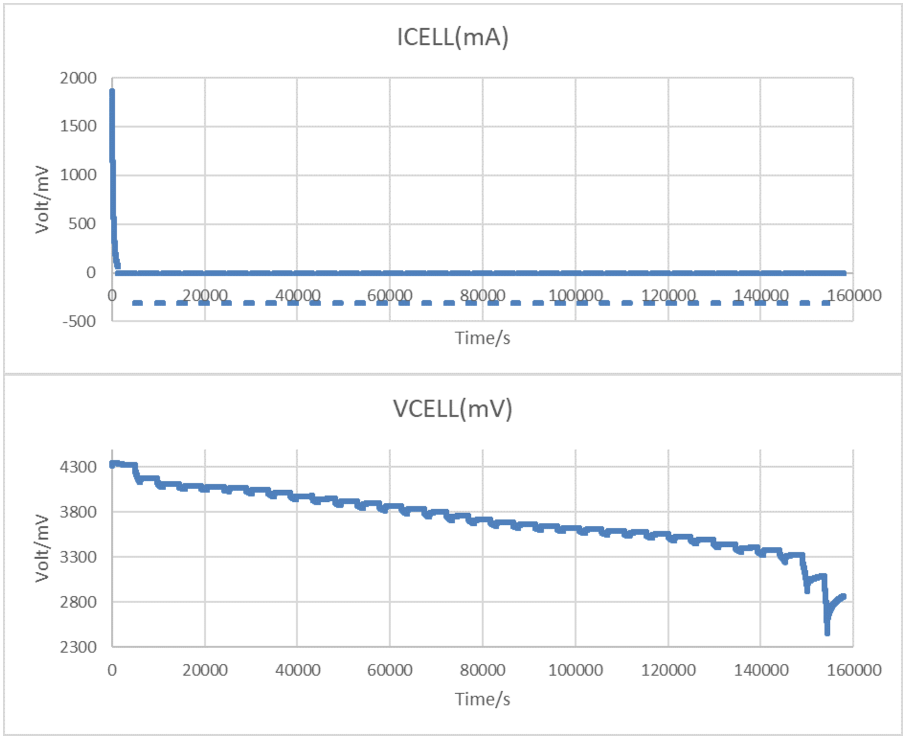


Figure 4-1. Pulse Discharge Test Case

If you use the GUI and the suggested source meter to do the battery test, remember to use the source meter in four wire mode, which can reduce the voltage detection error caused from line resistance. The suggested setup is shown in Figure 4-2. The MCU COM tool is used to get the battery run data. The SM COM tool is used to control the source meter to generate pulse battery charge and collect the voltage and current data to generate the battery parameters later.

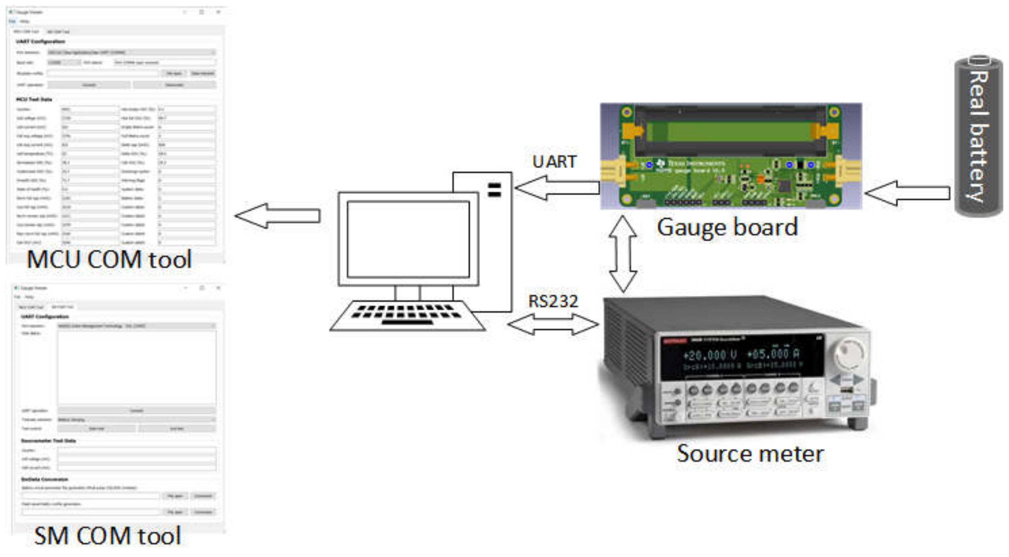
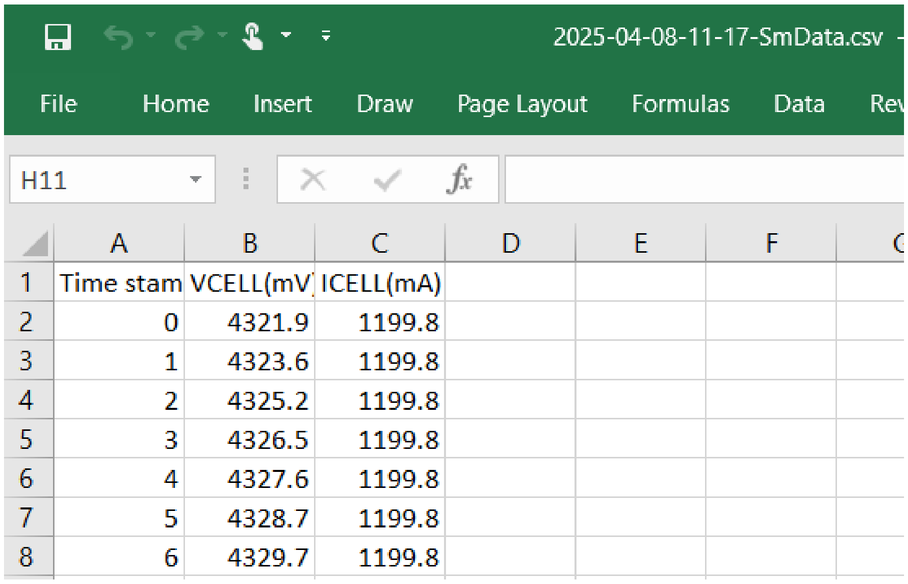


Figure 4-2. Hardware Structure to Get Battery Model

If you use your own test machine to do the test, you can construct the test data according to the SMData format and using SM COM tool later for battery model generation. Here is the SMData format. You need to input your tested Vcell and Icell data in Row B and Row C from Line 2. And then name the file with "-SmData.csv" at last.

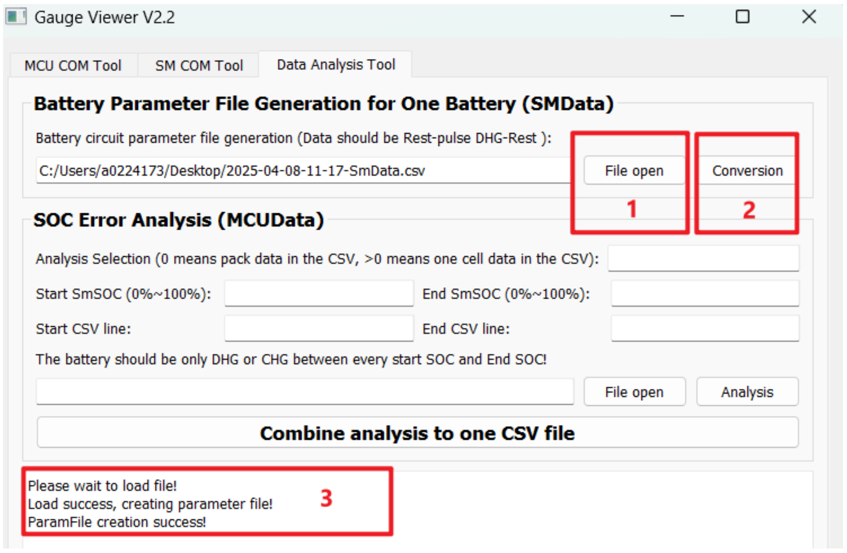


	A	B	C	D	E	F
1	Time stam	VCELL(mV	ICELL(mA)			
2	0	4321.9	1199.8			
3	1	4323.6	1199.8			
4	2	4325.2	1199.8			
5	3	4326.5	1199.8			
6	4	4327.6	1199.8			
7	5	4328.7	1199.8			
8	6	4329.7	1199.8			

Figure 4-3. SmData Type

4.2.2 Battery Model Generation

After the battery is running data in SMData format or the MCUData format (the name needs to follow the naming format), users can use *Battery Parameter File Generation for One Battery* to get the battery model (battery circuit file) in csv and text by following in the steps in Figure 4-4.


Gauge Viewer V2.2
MCU COM Tool
SM COM Tool
Data Analysis Tool
Battery Parameter File Generation for One Battery (SMData)
File open
Conversion
1
2
SOC Error Analysis (MCUData)
File open
Analysis
Combine analysis to one CSV file
3

Battery circuit parameter file generation (Data should be Rest-pulse DHG-Rest): C:/Users/a0224173/Desktop/2025-04-08-11-17-SmData.csv

Analysis Selection (0 means pack data in the CSV, >0 means one cell data in the CSV):

Start SmSOC (0%~100%): End SmSOC (0%~100%):

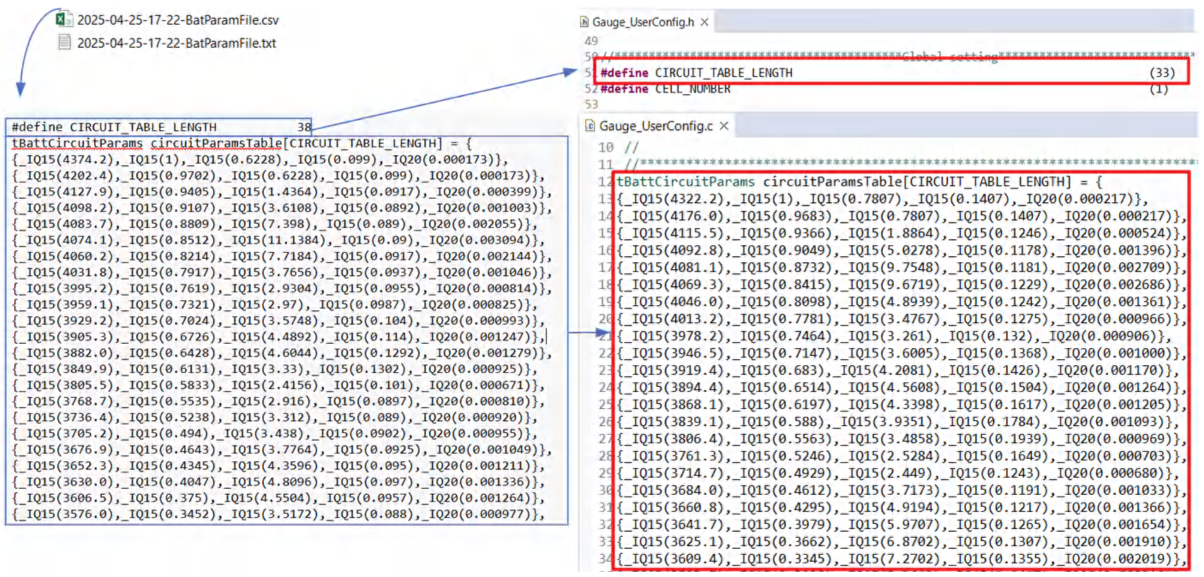
Start CSV line: End CSV line:

The battery should be only DHG or CHG between every start SOC and End SOC!

```
Please wait to load file!
Load success, creating parameter file!
ParamFile creation success!
```

Figure 4-4. Generate Parameter File From SMData

Copy the generated table in the text into Gauge_UserConfig.c, and the table length into Gauge_UserConfig.h. Then, finish the battery circuit table input.


2025-04-25-17-22-BatParamFile.csv
2025-04-25-17-22-BatParamFile.txt
Gauge_UserConfig.h
Gauge_UserConfig.c

```
#define CIRCUIT_TABLE_LENGTH                    38
tBattCircuitParams circuitParamsTable[CIRCUIT_TABLE_LENGTH] = {
{_IQ15(4374.2),_IQ15(1),_IQ15(0.6228),_IQ15(0.099),_IQ20(0.000173)},
{_IQ15(4202.4),_IQ15(0.9702),_IQ15(0.6228),_IQ15(0.099),_IQ20(0.000173)},
{_IQ15(4127.9),_IQ15(0.9405),_IQ15(1.4364),_IQ15(0.0917),_IQ20(0.000399)},
{_IQ15(4098.2),_IQ15(0.9107),_IQ15(3.6108),_IQ15(0.0892),_IQ20(0.001003)},
{_IQ15(4083.7),_IQ15(0.8809),_IQ15(7.398),_IQ15(0.089),_IQ20(0.002055)},
{_IQ15(4074.1),_IQ15(0.8512),_IQ15(11.1384),_IQ15(0.09),_IQ20(0.003094)},
{_IQ15(4060.2),_IQ15(0.8214),_IQ15(7.7184),_IQ15(0.0917),_IQ20(0.002144)},
{_IQ15(4031.8),_IQ15(0.7917),_IQ15(3.7656),_IQ15(0.0937),_IQ20(0.001046)},
{_IQ15(3995.2),_IQ15(0.7619),_IQ15(2.9304),_IQ15(0.0955),_IQ20(0.000814)},
{_IQ15(3959.1),_IQ15(0.7321),_IQ15(2.97),_IQ15(0.0987),_IQ20(0.000825)},
{_IQ15(3929.2),_IQ15(0.7024),_IQ15(3.5748),_IQ15(0.104),_IQ20(0.000993)},
{_IQ15(3905.3),_IQ15(0.6726),_IQ15(4.4892),_IQ15(0.114),_IQ20(0.001247)},
{_IQ15(3882.0),_IQ15(0.6428),_IQ15(4.6044),_IQ15(0.1292),_IQ20(0.001279)},
{_IQ15(3849.9),_IQ15(0.6131),_IQ15(3.33),_IQ15(0.1302),_IQ20(0.000925)},
{_IQ15(3805.5),_IQ15(0.5833),_IQ15(2.4156),_IQ15(0.101),_IQ20(0.000671)},
{_IQ15(3768.7),_IQ15(0.5535),_IQ15(2.916),_IQ15(0.0897),_IQ20(0.000810)},
{_IQ15(3736.4),_IQ15(0.5238),_IQ15(3.312),_IQ15(0.089),_IQ20(0.000920)},
{_IQ15(3705.2),_IQ15(0.494),_IQ15(3.438),_IQ15(0.0902),_IQ20(0.000955)},
{_IQ15(3676.9),_IQ15(0.4643),_IQ15(3.7764),_IQ15(0.0925),_IQ20(0.001049)},
{_IQ15(3652.3),_IQ15(0.4345),_IQ15(4.3596),_IQ15(0.095),_IQ20(0.001211)},
{_IQ15(3630.0),_IQ15(0.4047),_IQ15(4.8096),_IQ15(0.097),_IQ20(0.001336)},
{_IQ15(3606.5),_IQ15(0.375),_IQ15(4.5504),_IQ15(0.0957),_IQ20(0.001264)},
{_IQ15(3576.0),_IQ15(0.3452),_IQ15(3.5172),_IQ15(0.088),_IQ20(0.000977)},
```

```

//*****************************************Global setting*****************************
#define CIRCUIT_TABLE_LENGTH                                          (33)
#define CELL_NUMBER                                                   (1)

```

```
//
//*************************************************************************
tBattCircuitParams circuitParamsTable[CIRCUIT_TABLE_LENGTH] = {
{_IQ15(4322.2),_IQ15(1),_IQ15(0.7807),_IQ15(0.1407),_IQ20(0.000217)},
{_IQ15(4176.0),_IQ15(0.9683),_IQ15(0.7807),_IQ15(0.1407),_IQ20(0.000217)},
{_IQ15(4115.5),_IQ15(0.9366),_IQ15(1.8864),_IQ15(0.1246),_IQ20(0.000524)},
{_IQ15(4092.8),_IQ15(0.9049),_IQ15(5.0278),_IQ15(0.1178),_IQ20(0.001396)},
{_IQ15(4081.1),_IQ15(0.8732),_IQ15(9.7548),_IQ15(0.1181),_IQ20(0.002709)},
{_IQ15(4069.3),_IQ15(0.8415),_IQ15(9.6719),_IQ15(0.1229),_IQ20(0.002686)},
{_IQ15(4046.0),_IQ15(0.8098),_IQ15(4.8939),_IQ15(0.1242),_IQ20(0.001361)},
{_IQ15(4013.2),_IQ15(0.7781),_IQ15(3.4767),_IQ15(0.1275),_IQ20(0.000966)},
{_IQ15(3978.2),_IQ15(0.7464),_IQ15(3.261),_IQ15(0.132),_IQ20(0.000906)},
{_IQ15(3946.5),_IQ15(0.7147),_IQ15(3.6005),_IQ15(0.1368),_IQ20(0.001000)},
{_IQ15(3919.4),_IQ15(0.683),_IQ15(4.2081),_IQ15(0.1426),_IQ20(0.001170)},
{_IQ15(3894.4),_IQ15(0.6514),_IQ15(4.5608),_IQ15(0.1504),_IQ20(0.001264)},
{_IQ15(3868.1),_IQ15(0.6197),_IQ15(4.3398),_IQ15(0.1617),_IQ20(0.001205)},
{_IQ15(3839.1),_IQ15(0.588),_IQ15(3.9351),_IQ15(0.1784),_IQ20(0.001093)},
{_IQ15(3806.4),_IQ15(0.5563),_IQ15(3.4858),_IQ15(0.1939),_IQ20(0.000969)},
{_IQ15(3761.3),_IQ15(0.5246),_IQ15(2.5284),_IQ15(0.1649),_IQ20(0.000703)},
{_IQ15(3714.7),_IQ15(0.4929),_IQ15(2.449),_IQ15(0.1243),_IQ20(0.000680)},
{_IQ15(3684.0),_IQ15(0.4612),_IQ15(3.7173),_IQ15(0.1191),_IQ20(0.001033)},
{_IQ15(3660.8),_IQ15(0.4295),_IQ15(4.9194),_IQ15(0.1217),_IQ20(0.001366)},
{_IQ15(3641.7),_IQ15(0.3979),_IQ15(5.9707),_IQ15(0.1265),_IQ20(0.001654)},
{_IQ15(3625.1),_IQ15(0.3662),_IQ15(6.8702),_IQ15(0.1307),_IQ20(0.001910)},
{_IQ15(3609.4),_IQ15(0.3345),_IQ15(7.2702),_IQ15(0.1355),_IQ20(0.002019)},
```

Figure 4-5. Battery Circuit Table Input

The element of the battery circuit table (battery model) has five combinations:

- The first is OCV (mV).
- The second is SOC.
- The third is Rcell (Ω).
- The fourth is cap factor.
- The fifth is slope rate.

A brief introduction is given on how these parameters are generated. As shown in Figure 4-6, OCV equals to the final Vcell before discharging. SOC is obtained after the test with the Qmax at the same time using Equation 2. Rcell equals to the Ohmic resistance shown in Figure 4-6. The voltage change in one second is treated as the influence of Rcell and the value equals to dOcv(mV)/Current (mA). The cap factor equals to dSOC(%)/dOCV(mV)*Qmax(As) or dSOC(%)/dOCV(V)*3.6*Qmax(mAh). Slope rate equals to dSOC(Dec) / dOCV(mV). For the detailed parameters generation method, see the python source code shared in the development package of this document.

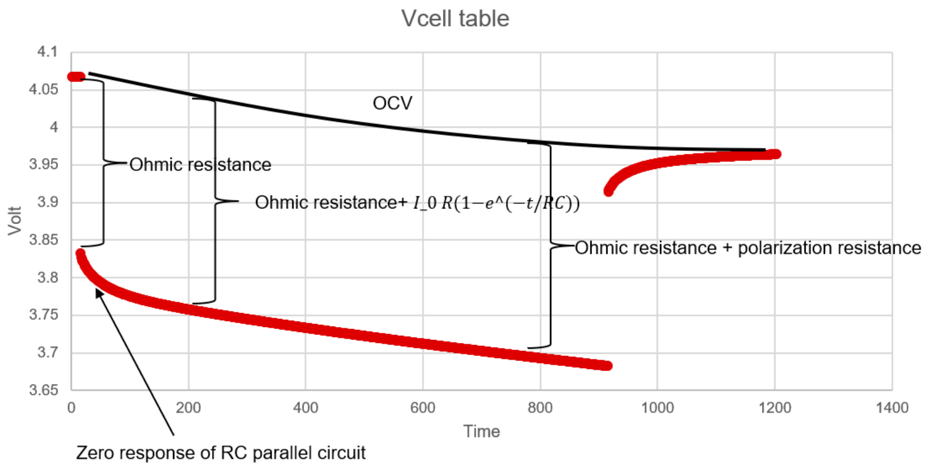
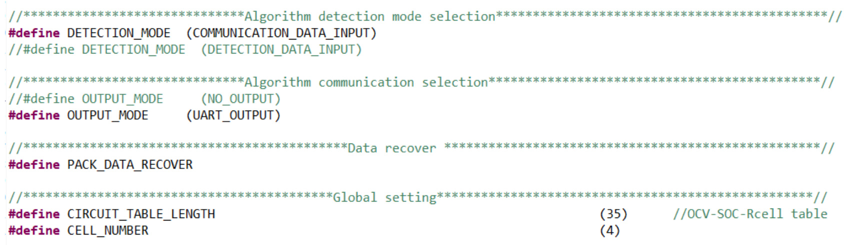


Figure 4-6. Pulse Discharge Example

4.3 Step 3: Input Customized Configuration

First, users need to make system changes based on the applications in *Gauge_UserConfig.h*. The commonly changed parameters are shown in Figure 4-7.



```
//*****************************Algorithm detection mode selection******************************************//
#define DETECTION_MODE  (COMMUNICATION_DATA_INPUT)
//#define DETECTION_MODE  (DETECTION_DATA_INPUT)

//*****************************Algorithm communication selection*****************************************//
//#define OUTPUT_MODE     (NO_OUTPUT)
#define OUTPUT_MODE     (UART_OUTPUT)

//******************************************Data recover **************************************************//
#define PACK_DATA_RECOVER

//*****************************************Global setting**************************************************//
#define CIRCUIT_TABLE_LENGTH                                                    (35)     //OCV-SOC-Rcell table
#define CELL_NUMBER                                                             (4)
```

Figure 4-7. Gaugge_UserConfig.h Setting

The explanation of these parameters is shown in Table 4-2.

Table 4-2. System Configuration Parameters

Parameters	Comment
DETECTION_MODE	Affects the algorithm data input source and described in Section 4.4.
OUTPUT_MODE	Controls whether to output tested data to GUI and described in Section 4.4.
PACK_DATA_RECOVER	If this parameter is defined, then the pack data recover function is enabled. This is reserved for the data saving requirement when MCU is shut down. For data saving and data loading functions, users need to do the development.
CELL_NUMBER	The cell numbers for the battery pack.
CIRCUIT_TABLE_LENGTH	circuitParamsTable length.

Second, users need to fulfill the data structure configuration in *Gauge_UserConfig.c*. A brief introduction to the used data structure in this gauge implementation is shown in Figure 4-8.

tGaugeApplication represents the battery pack. All the pack related results are saved in *tBattPackParams*. *tbattGlobalParams[]* represents every battery cell in the battery pack. All the algorithm data structures are all underneath.

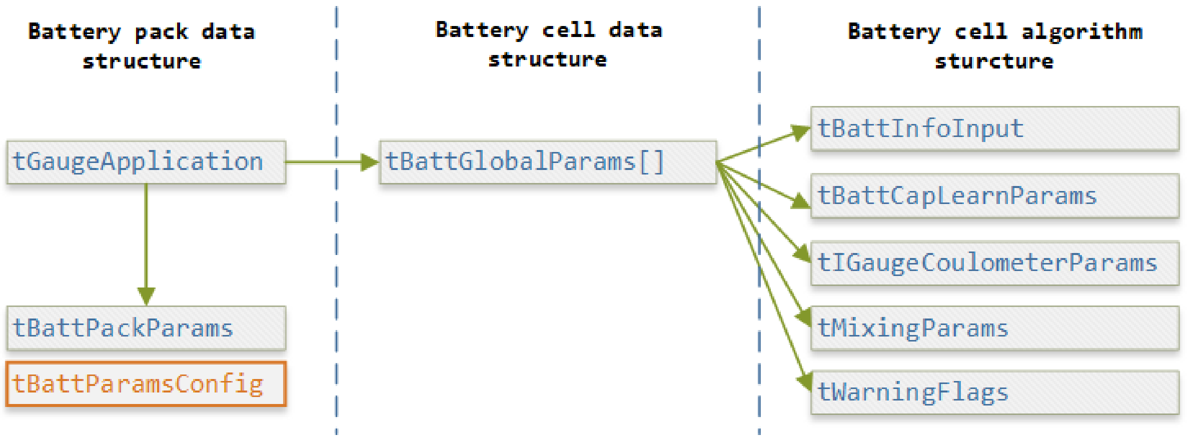
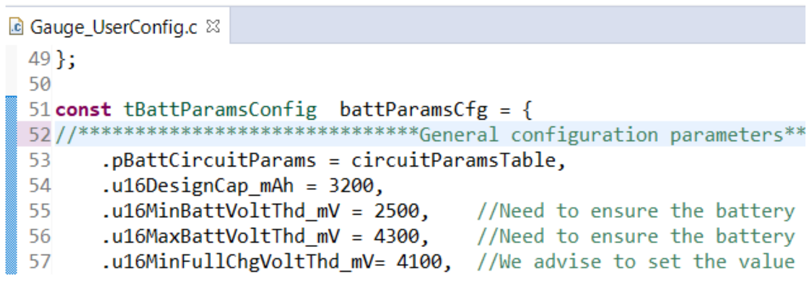


Figure 4-8. Data Structure

The most important data structure is *tBattParamsConfig* as shown in Figure 4-9. This contains all the battery parameter settings and algorithm settings.


Gauge_UserConfig.c

```
};

const tBattParamsConfig  battParamsCfg = {
//******************************General configuration parameters**
    .pBattCircuitParams = circuitParamsTable,
    .u16DesignCap_mAh = 3200,
    .u16MinBattVoltThd_mV = 2500,    //Need to ensure the battery
    .u16MaxBattVoltThd_mV = 4300,    //Need to ensure the battery
    .u16MinFullChgVoltThd_mV= 4100,  //We advise to set the value
```

Figure 4-9. tBattParamsConfig Structure

For easy evaluation, users only need to change the general configuration parameters. These parameters are divided into five parts. A short description is provided for all these related parameters.

Table 4-3. General Configuration Parameters

Parameters	Comment
battModelUse	This enables a thermal model to evaluate the core temperature iq15EvlCoreTemp_C, which is only used to update and search emptySoc and fullSoc. When a module is used, the suggested iq15Tcell_C temperature resolution is 0.1℃. With 1℃ resolution, the iq15EvlCoreTemp_C error is added with about ±1.5℃. When no module used, surface temperature is used as battery core temperature.
u16battWeight_g	Battery weight used for core temperature evaluation. For example, 18650: 49g; 21700: 60g; 26650: 96g.
u16DesignCap_mAh	Design capacity. The input the standard capacity of battery or the tested one through battery parameter generation test.
u16MinBattVoltThd_mV / u16MaxBattVoltThd_mV i16MaxChgCurtThd_mA / i16MinDhgCurtThd_mA i8MaxChgTempThd_C / i8MinChgTempThd_C i8MaxDhgTempThd_C / i8MinDhgTempThd_C	Battery Vcell, Icell and Tcell threshold. These are reserved to control warning flags when the battery situation is above these parameters. These do not influence the gauge performance.
u16MinFullChgVoltThd_mV u16MaxFullChgVoltThd_mVi16FullChgCurtThd_mA	Battery full related setting. The battery charge voltage is in this range. u16MinFullChgVoltThd_mV helps to judge when the battery is full. u16MaxFullChgVoltThd_mV is used as default Full OCV after MCU power on. When the current is below than i16FullChgCurtThd_mA and the voltage is above u16MinFullChgVoltThd_mV, treat the battery as full.
u16EmptyDhgVoltThd_mV	When the voltage reaches this value, battery as empty.
u8AvgBattParamsUpdateCount	The average data is obtained after the settled cycles.

Table 4-4. Mixing Algorithm Related Parameters

Parameters	Comment
u16ConvergeStartVolt_mV	When the run-time voltage is below than this parameter, the converging algorithm starts to work to compensate for the emptySoc.
i8TempThd_C[] i16CurtThd_mA[]	Used to find an excellent emptySoc and fullSoc in iq15AbsEmptySocMatrix[] and iq15AbsFullSocMatrix[] according to the current and temperature thresholds. For more details, refer to Section 2.3.4.
u16EmptyOcvMatrix[]	Used to calculate the related emptySoc for iq15AbsEmptySocMatrix[]. If all is 0, then u16EmptyDhgVoltThd_mV is used as the empty OCV to calculate a related emptySoc. Then, this is automatically learned after cycling. If users want to get better gauge performance in learning cycles, then users can test the OCVs in the related ranges and input into the matrix.
u16FullOcvMatrix[]	Used to calculate the relatedfulSoc for iq15AbsFullSocMatrix[]. If all is 0, then u16MaxFullChgVoltThd_mV is used as the full OCV to calculate a related fullSoc. Then, this is automatically learned after cycling. If users want to get better gauge performance in learning cycles, then users can test the OCVs in the related ranges and input into the matrix.

Table 4-5. Capacity Learn Algorithm Related Parameters

Parameters	Comment
i16UnloadCurtLowThd_mA i16UnloadCurtHighThd_mA	If the current is between this range, then the battery is treated as rest. Consider the noise of current detection. Otherwise, the rest detection has problems.
u8SOHCalcCycleThd	The battery discharge cycle threshold to do SOH calculation.
iq15DefaultSOH_DEC	Battery default SOH value.
u8NomFullCapIIRLevel	Used to control the IIR filter level for NomFullCap. freq_cut = Freq_sample/(2*pi*2^ui8Beta).

Table 4-6. VGauge Algorithm Related Parameters

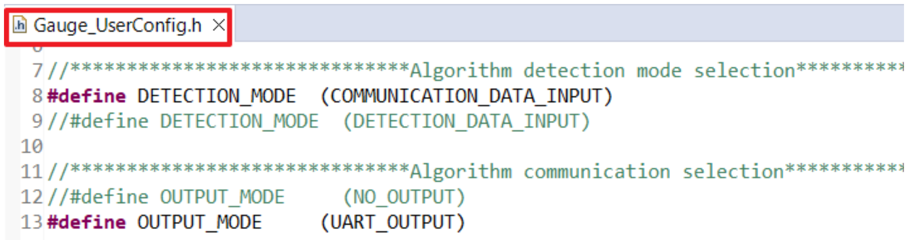
Parameters	Comment
u8CircuitTableLength	Circuit table length.
u8CircuitTableTestTemp_C iq15RcellNegTshift_R iq15RcellPosTshift_R	These parameters are used to evaluate the Rcell under different temperatures. This does not affect the performance too much. Users can keep them to be same.

Table 4-7. IGauge Algorithm Related Parameters

Parameters	Comment
i16AvgLeckageCurt_mA	Internal leakage current compensation. This means the evaluated current that can't be measured by shunt resistor, consumed by MCU or battery internal leakage current.
i16EqualizationCurt_mA	This means the equalization current, users need to handle the battery equalization by themselves.

4.4 Step 4: Evaluation

Figure 4-10 shows different evaluation modes used for different conditions selected in Gauge_UserConfig.h.



```
//*****************************Algorithm detection mode selection*********
#define DETECTION_MODE  (COMMUNICATION_DATA_INPUT)
//#define DETECTION_MODE  (DETECTION_DATA_INPUT)

//*****************************Algorithm communication selection**********
//#define OUTPUT_MODE     (NO_OUTPUT)
#define OUTPUT_MODE     (UART_OUTPUT)
```

Figure 4-10. Gauge Mode Setting

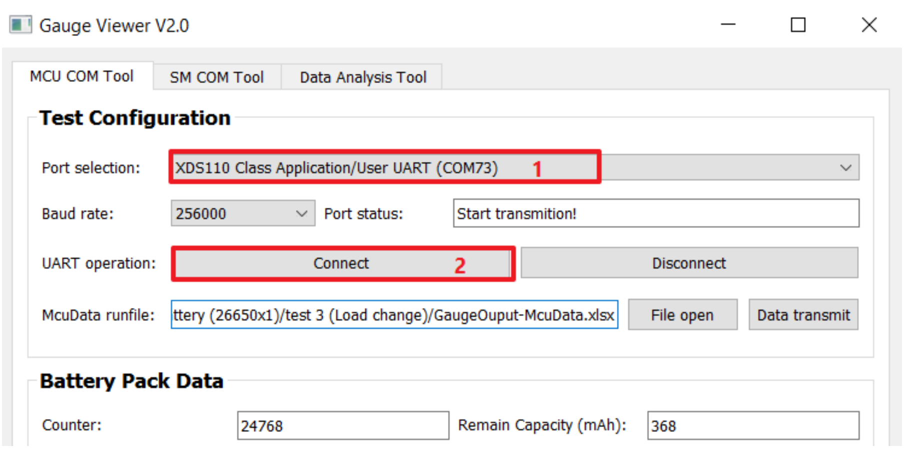
For different output modes, UART_OUTPUT means enabling data output through universal asynchronous receiver or transmitter (UART). Then, users can observe the battery running parameters on the GUI through USB to tool. NO_OUTPUT means terminating the UART data output.

The different detection modes are detailed in the following section. Detection data input mode is the common used mode. All the algorithm input is from real data tested by AFE. The input data of the communication data input mode comes from the GUI and the one cycle time limitation lies on the UART communication speed.

4.4.1 Detection Data Input Mode

In this mode, users need the MSPM0 Gauge board and a real battery for test. The detection data (Vcell, Icell, Tcell) comes from the real detected signals. The GUI can help to record the battery running data for further analysis.

For software setting, users need to download the gauge code to the LaunchPad after changing the detection mode to *DETECTION_DATA_INPUT* in *Gauge_UserConfig.h*.


Gauge Viewer V2.0

MCU COM Tool | SM COM Tool | Data Analysis Tool

Test Configuration

Port selection: XDS110 Class Application/User UART (COM73) 1

Baud rate: 256000 Port status: Start transmition!

UART operation: Connect 2 | Disconnect

McuData runfile: ttery (26650x1)/test 3 (Load change)/GaugeOuput-McuData.xlsx | File open | Data transmit

Battery Pack Data

Counter: 24768 Remain Capacity (mAh): 368

Figure 4-11. Detection Data Input Mode

For GUI control, users need to select the right port with the name XDS110 Class Application or User UART and click the connect button. If the MCU already works, then users see the test data at MCU Test Data block. After users click the disconnect button, the data is saved in the CSV type under the same address of GUI with the name *YYYY-MM-DD-HH-MM-McuData.csv*.

Note

If there is some high frequency noise on the Vcell, Tcell and Icell, then TI recommends users to add an additional IIR filter for these algorithm inputs.

4.4.2 Communication Data Input Mode

For this mode, the battery running data is input from the GUI. This enables users to run the real test case or evaluate the MSPM0 Gauge with only a LaunchPad. This method can remove the need of hardware, increase algorithm running frequency and have no limit to the length of battery running data.

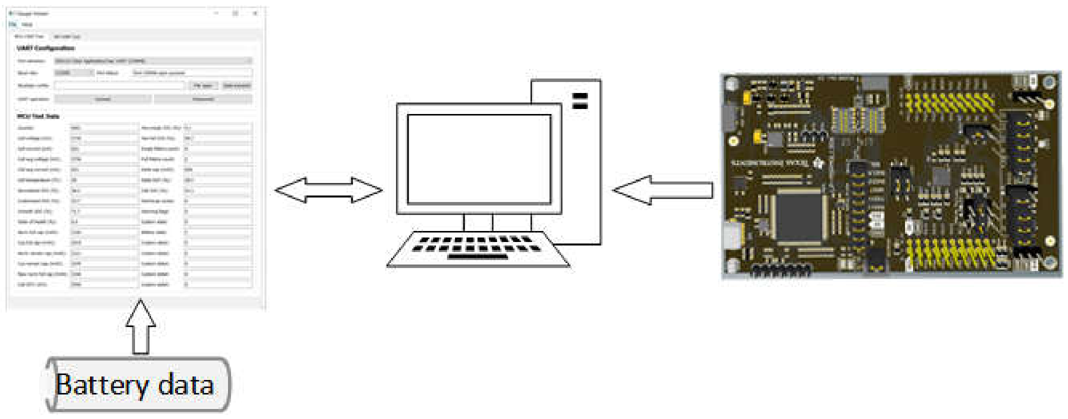
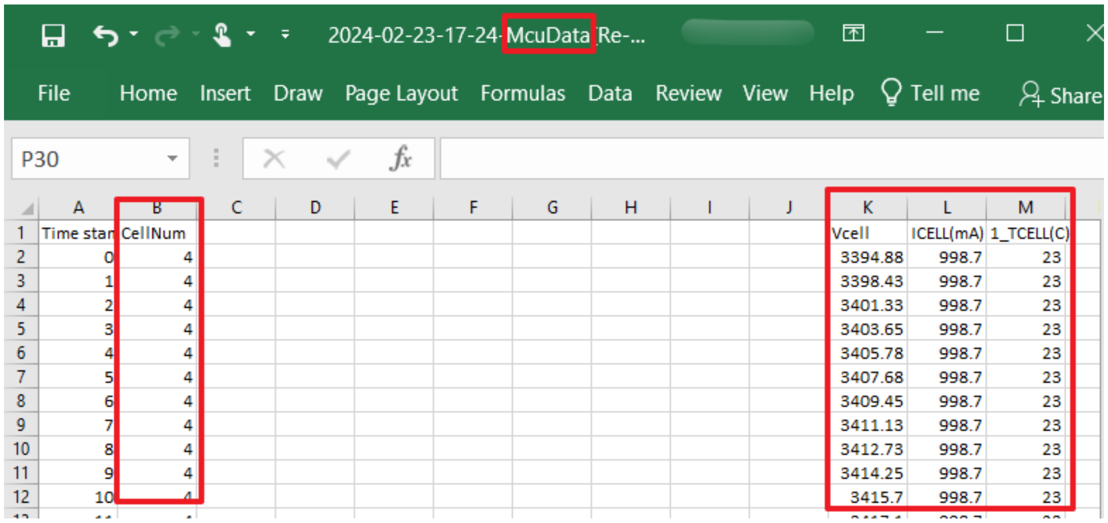


Figure 4-12. Communication Data Input Mode Structure

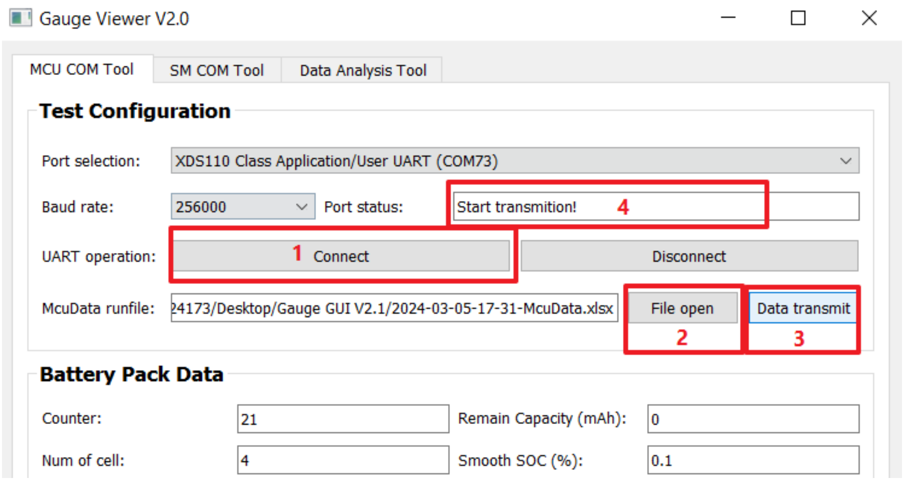
1. First, to realize this method, users only need a LaunchPad and do the right hardware setting.
2. Second, download the gauge code to the LaunchPad after changing the detection mode to *COMMUNICATION_DATA_INPUT* in *Gauge_UserConfig.h*.
3. Third, users need to have a MCUData file. An introduction is provided on how to transmit a test data into a recognized file by the GUI, especially for those who do not generate the test file from GUI. Users need to input the Cell num at column B. Then, input each Vcell(mV), Icell(mA) and Tcell(°C) of the battery into the same column as the same in McuData file. For this, users can generate a McuData file first and refer to that one to do the transmit on. Finally, name the file with *-McuData.csv*.


2024-02-23-17-24-McuData Re-...
File Home Insert Draw Page Layout Formulas Data Review View Help Tell me Share
P30

	A	B	K	L	M
1	Time stan	CellNum	Vcell	ICELL(mA)	1_TCELL(C)
2	0	4	3394.88	998.7	23
3	1	4	3398.43	998.7	23
4	2	4	3401.33	998.7	23
5	3	4	3403.65	998.7	23
6	4	4	3405.78	998.7	23
7	5	4	3407.68	998.7	23
8	6	4	3409.45	998.7	23
9	7	4	3411.13	998.7	23
10	8	4	3412.73	998.7	23
11	9	4	3414.25	998.7	23
12	10	4	3415.7	998.7	23

Figure 4-13. McuData Type

4. Fourth, connect the UART COM port following Figure 4-14 and load the MCUData runfile in MCU COM Tool by clicking the *File Open* button. After clicking the *Data transmit* button, wait until the port status changes to *Start transmit!*.



Test Configuration

Port selection: XDS110 Class Application/User UART (COM73)

Baud rate: 256000 Port status: Start transmition!

UART operation:

McuData runfile: 24173/Desktop/Gauge GUI V2.1/2024-03-05-17-31-McuData.xlsx

Battery Pack Data

Counter: 21 Remain Capacity (mAh): 0

Num of cell: 4 Smooth SOC (%): 0.1

Figure 4-14. Communication Data Input Steps

Users receive the battery running data from MCU shown in MCU Test Data block. After this finishes transmitting, the GUI automatically saves the received data under the GUI address.

4.5 Step 5: Gauge Performance Check

4.5.1 Learning Cycles

Before evaluating the gauge performance, users need to have a concept about the learning cycles of this algorithm. This algorithm has three parameters that are learned in the code running. The first is the NomFullCap and MaxNomFullCap. The second is the FullSoc and the third is the EmptySoc. All of these affect the SmoothSoc output performance and accuracy. In the following part, a battery test case is shown to show the learning capacity of this algorithm in Figure 4-11.

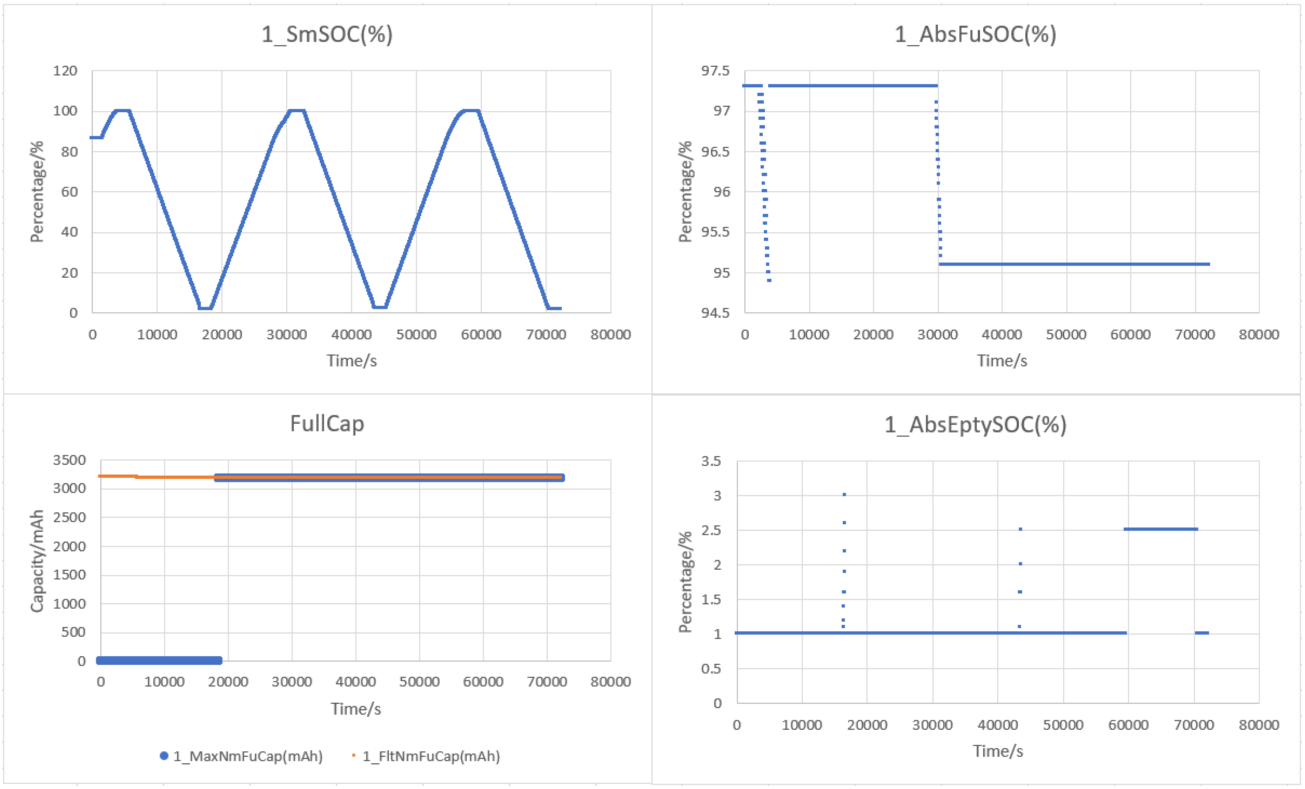


Figure 4-15. Battery Discharge and Charge for 3 Times

When the iGaugeDominationFlg is set, the MaxNomFullCap changes from 0 to a value. At this time, this means the gauge algorithm calculates the NomFullCap error and NomFullSoc error reduce to an acceptable range. Normally, this takes one full discharge or charge.

The FullSoc and EmptySoc and are related to the CusSoc and SmoothSoc. These adjust when the battery turns to full or empty. However, the learning result is not saved at the beginning, as the NomFullSoc error is large. In Figure 4-15, the FullSoc adjusts to 94.9% at about 4000s and readjusts again at 30000s. The EmptySoc adjusts at 16000s and readjusts again at 43000s. At 60000s, as the load is same, the EmptySoc loads the saved value from the EmptySocMatrix.

SmoothSoc can have large jumps when EmptySoc and FullSoc is learning, especially when MaxNomFullCap is not learned and NomAbsSoc has large errors in the learning cycles. The battery is doing a constant discharge from 100% to 0%. An example is shown in Figure 4-16. This can be common on LiFePO4 batteries. To reduce the SmoothSoc error, users can use PackRecordLoad function to give the algorithm a start NomAbsSoc value and set the EmptyOcvMatrix and FullOcvMatrix in the battParamsCfg. If users just want to avoid the large jump of the SmoothSoc, then an IIR filter with an excellent cut of frequency is suggested to be added on the SmoothSoc output.

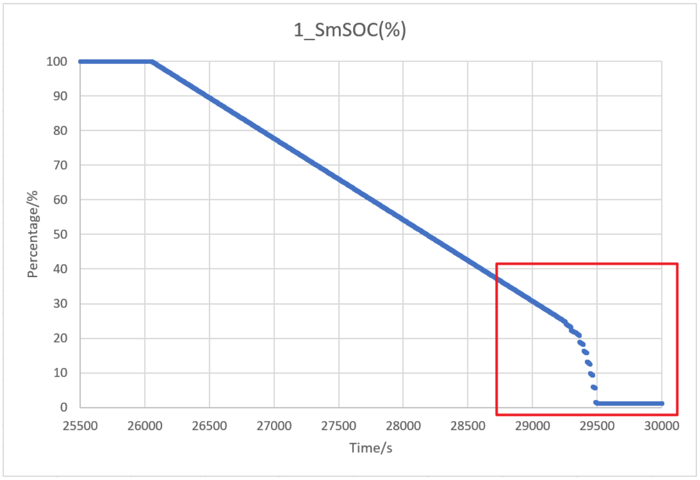


Figure 4-16. SmoothSoc Jump

4.5.2 SOC and SOH Accuracy Evaluation

If users want to evaluate SOC accuracy, then refer to the SmoothSoc instead of the NomSoc or CusSoc. The reason is that, from end user's side, the wanted SOC is a relative SOC, which can only change from 0% to 100% and no SOC jump is seen in the SOC output.

As SmoothSoc is a relative SOC, the accurate value cannot be obtained by searching the SOC-OCV table. The point when SmoothSoc is 100% and 0% is easily seen. To know the accurate SmoothSoc value other than 0% and 100%, users need to first construct the relationship between SmoothSoc and CusRemCap by charging the battery to 100% and discharge to 0%. Then, redo the test with the same load and same temperature. After that, users can use the CusRemCap to get the accurate SmoothSoc value. Refer to Table 4-8 for further description.

Table 4-8. Get Accurate SmoothSoc Value

Accurate Value of SmoothSoc	Condition
100%	When current drops to FullChgCurtThd and the voltage is between MinFullChgVoltThd and MaxFullChgVoltThd.
0%	When voltage drops to EmptyDhgVoltThd.
Other values	Get a SmoothSoc-CusRemCap table first and redo the test with the same condition to keeep the EmptySoc and FullSoc to be same. Then, users can use the CusRemCap to get the right smoothSoc value.

As shown in Section 4.5.1, at least after one discharge or charge, the MaxNomFullCap is obtained and the EmptySoc and FullSoc starts to learn and save into the matrix. To get a more accurate SmoothSoc, TI recommends users to run two battery cyles first. The first cycle is to learn the MaxNomFullCap. The second cycle is to learn and save the SmoothSoc and FullSoc. After that, the gauge algorithm works in the best performance.

The SOH equals to MaxNomFullCap / NomFullCap. In the code, add a 2% margin to SOH output to avoid being below 100% at the first battery cycle, due to NomFullCap fluctuation. If users want to check the SOH accuracy, then TI recommends to check the NomFullCap accuracy instead. To evaluate NomFullCap accuracy after multicycles, the suggested flow is:

1. Rest the battery for 1 hour, measure the cell voltage, and map the voltage as the COV to the SOC1, using the SOC-OCV Table.
2. After one charge or discharge cycle, rest for 1 hour, measure the OCV again, and map to the SOC2.
3. Calculate the NomFullCap: NomFullCap=Quse/(SOC2−SOC1).

5 MSPM0 Gauge Solutions

In this section, two different test cases are used to show the capability of MSPM0 Gauge L2:

- Able to switch from one cell detection to multiple cell detection easily
- Able to handle different battery type, especially for LiFePO4, which SOC-OCV table is much flat

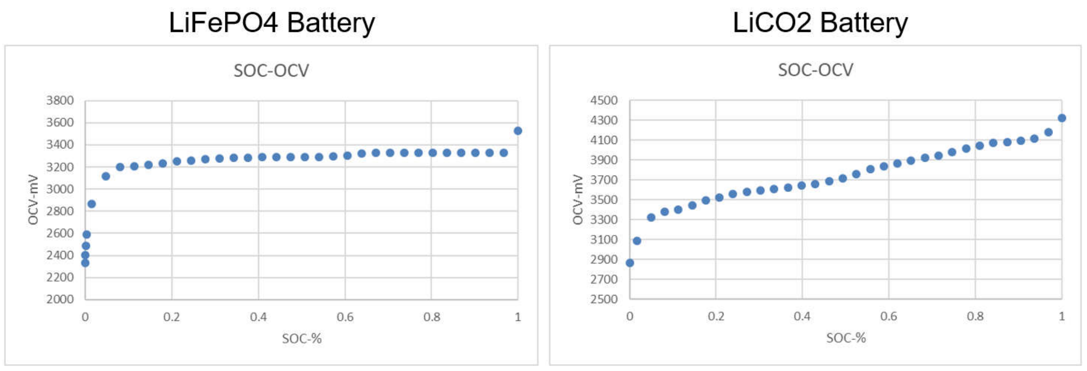


Figure 5-1. Battery SOC

MSPM0 hardware features are also shown, which is why this is excellent for BMS implementations, including internal high precision analog peripherals and numerous communication methods (CAN, UART, serial peripheral interface (SPI) , inter-integrated circuit (I2C)).

5.1 MSPM0L1306 and 1 LiCO2 Battery

Implementation features:

- Total implementation takes about 15K flash and 1.3K SRAM
- Current consumption without universal asynchronous receiver or transmitter (UART) communication (NO_OUTPUT mode) is about 9uA

Implementation advantage:

- A one-chip implementation with internal analog peripherals
- Able to self-calibrate current detection with approx. 1% error
- High performance gauge algorithm

5.1.1 Hardware Setup Introduction

The hardware board is typically made to evaluate the one-cell battery gauge implementation.

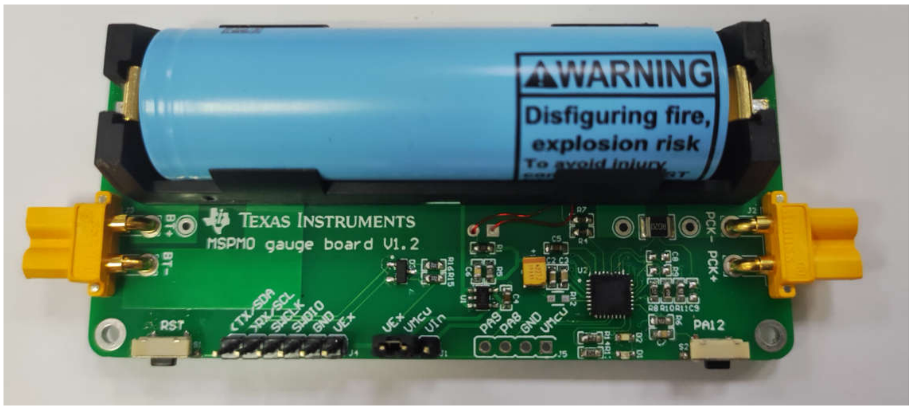


Figure 5-2. MSPM0 Gauge Hardware Board

Figure 5-3 shows the hardware high-level block diagram, showing all the used pins by this demo. The implementation tests the current at the analog-to-digital controller (ADC) channel 13, the temperature at ADC channel 5 and voltage at ADC channel 1.

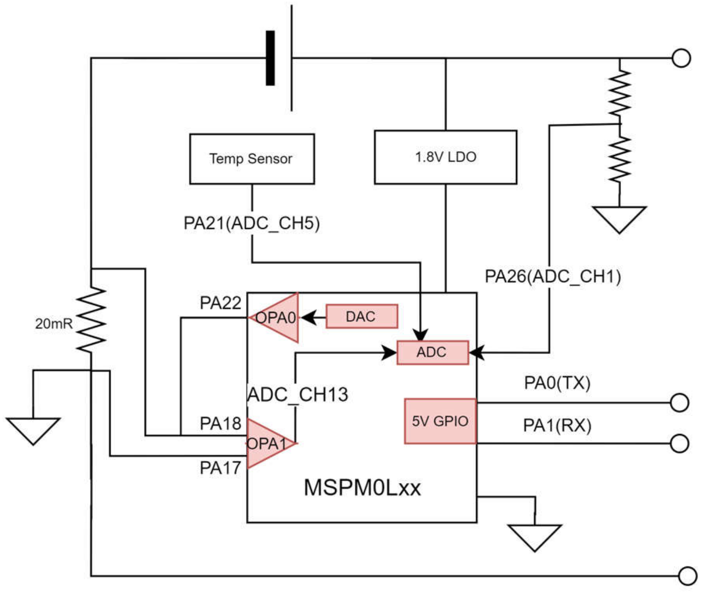


Figure 5-3. MSPM0 Gauge Board Block Diagram

With the internal OPA for current detection, the detection error at room temperature can reach ±0.25% at ±2A load. For more hardware introduction and performance, refer to the A Self-Calibratable Current Detection Solution Based on MSPM0 application note.

The gauge board instructions are shown in Figure 5-4. Pay attention to the MCU power switch supply jumper. For downloading, connect VMCU to VEx, then the MCU is supplied with 3.3V, which can make sure the voltage matches with the debugger. For evaluation, connect VMCU to VIn, then the MCU is supplied with 1.8V-LDO. This can make sure of the best analog performance. When the MCU is powered in around 500ms, the MCU calibrates the ADC+OPA for current detection. At this time, the current is 0. Otherwise, there is a constant current offset.

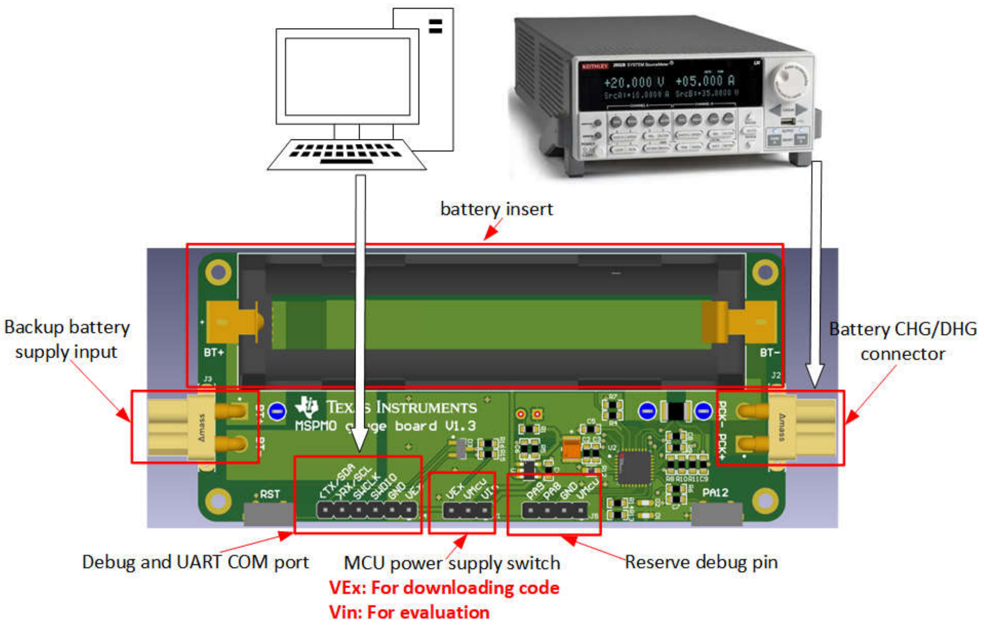


Figure 5-4. Gauge Board Instructions

If users are using the MSPM0L1306 LaunchPad in communication data input mode, then connect the UART pin as follows. This does not an exception to the software change.

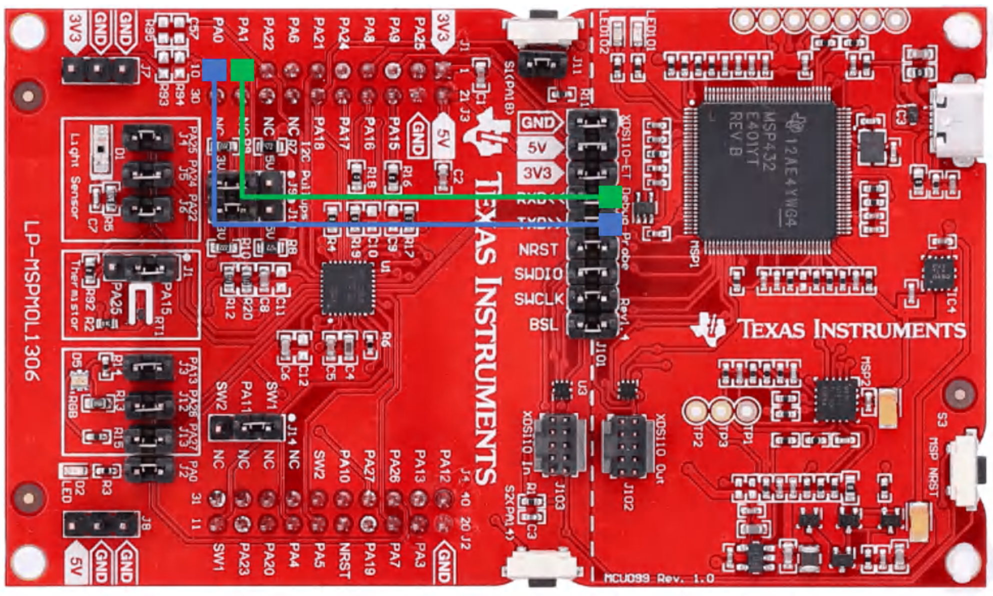


Figure 5-5. MSPM0L1306 Launchpad UART Connection

5.1.2 Software and Evaluation Introduction

Before starting software development and evaluation, TI recommend to refer to the MSPM0 Design Flow Guide user's guide to have a basic understanding about MSPM0 ecosystem. This helps users learn about MSPM0 development.

Figure 5-6 illustrates the software project. The project and files related to the gauge algorithm has five parts. Other files are the same for all the MSPM0 projects.

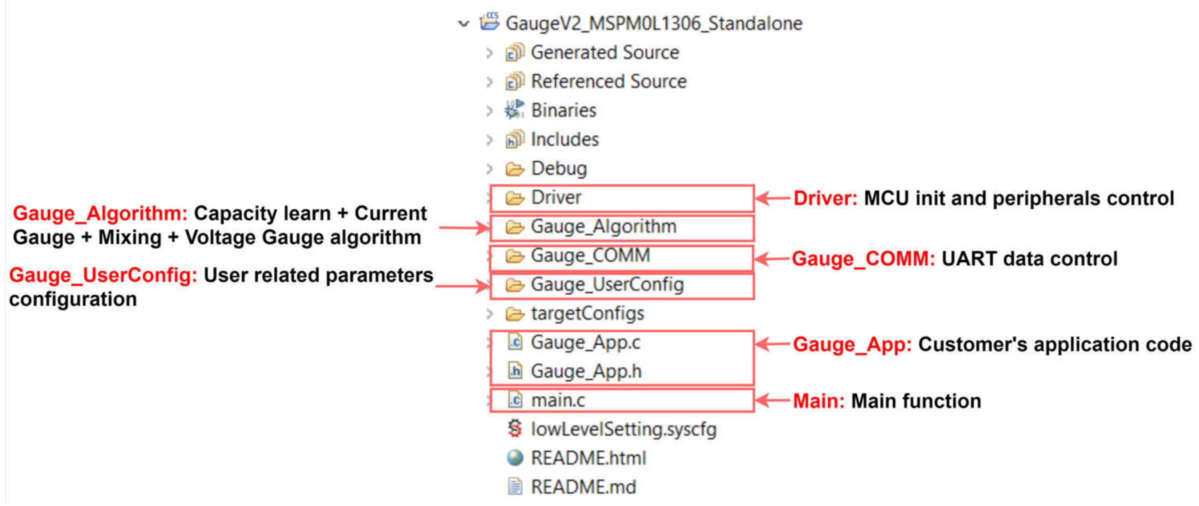

GaugeV2_MSPM0L1306_Standalone
Generated Source
Referenced Source
Binaries
Includes
Debug
Driver
Gauge_Algorithm
Gauge_COMM
Gauge_UserConfig
targetConfigs
Gauge_App.c
Gauge_App.h
main.c
lowLevelSetting.syscfg
README.html
README.md
Driver: MCU init and peripherals control
Gauge_Algorithm: Capacity learn + Current Gauge + Mixing + Voltage Gauge algorithm
Gauge_COMM: UART data control
Gauge_UserConfig: User related parameters configuration
Gauge_App: Customer's application code
Main: Main function

Figure 5-6. MSPM0 Gauge Software Project View

For Gauge_UserConfig part, find the description in Section 4.3. The Gauge_Algorithm part introduction is in Section 2. The Driver part includes all the MCU related control. This prepares Icell, Vcell and Tcell data into Gauge_Algorithm. The Gauge_COMM part handles all the UART protocol. The Gauge_App part includes the high-level gauge algorithm calling. This is the place for customers to customize functions. The Main part includes the highest system function code.

Remember to follow the steps Section 4 to update the configurations in Gauge_UserConfig folder:

- Generate battery models or use the default one
- Update the configuration in tBattParamsConfig structure
- Update battGlobalParams_xx, and battGlobalParamsArray according to your own battery cells
- Update the detection mode, communication mode, circuit table length and cell numbers

After users program the MSPM0 through XDS110, use the GUI to check and record the results.

5.1.3 Battery Test Cases

5.1.3.1 Performance Test

Here is the test based on a 3200mAh LiCO2 battery under 25°C. The u16MaxFullChgVoltThd setting is 4200mV. The EmptyDhgVoltThd setting is 3000mV.

Note

Make sure the battery is settled before the MCU is powered and the battery is in rest state before testing. Otherwise, the first SOC output is met with an error.

Here is the test pattern:

- Do pulse discharge and pulse charge with different load.
- Constant charge and discharge without rest for 4 cycles with different load.

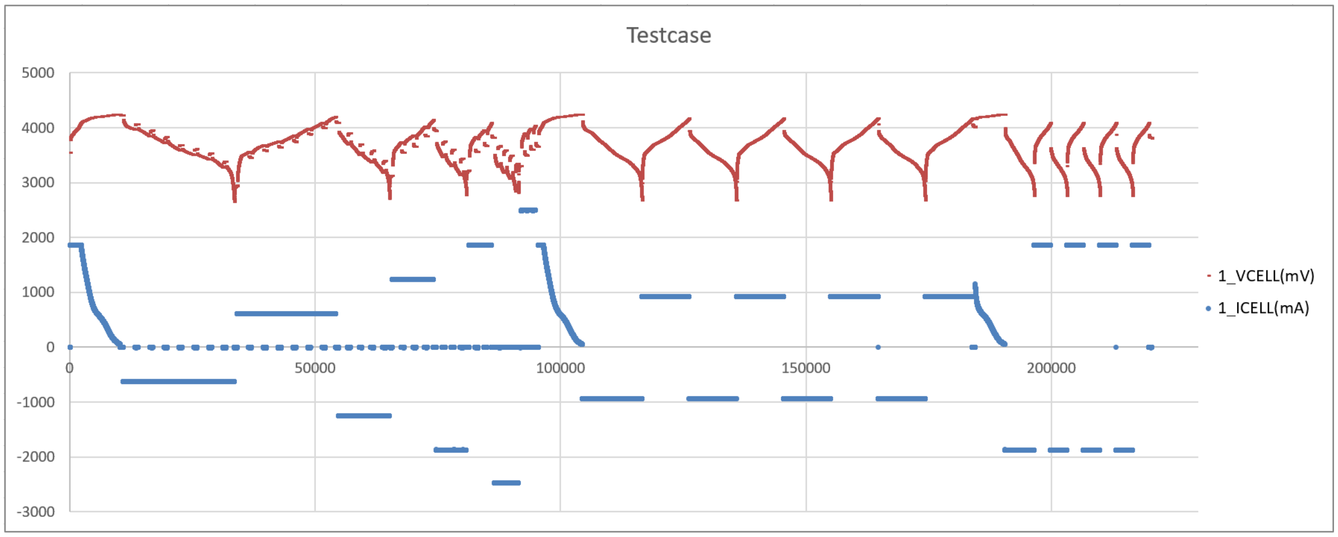


Figure 5-7. Battery Test Case

At the beginning, there is an obvious gap for NomSOC, CusSOC and SmoothSOC. See the test results in Figure 5-8. This is caused from first OCV calibration error.

The CusSOC has some gaps at the end of discharge, and this is because of EmptySOC. The SmoothSOC is flat and no jump at battery rest. All the data is controlled in the 0% to 100% range.

For different NomFullCap, the FltNomFullCap is updated after almost every rest. With the digital filter help, the NomFullCap gets more and more accurate. The MaxNomFullCap changes from 0 to a value, which means the output NomFullCap has an acceptable accuracy.

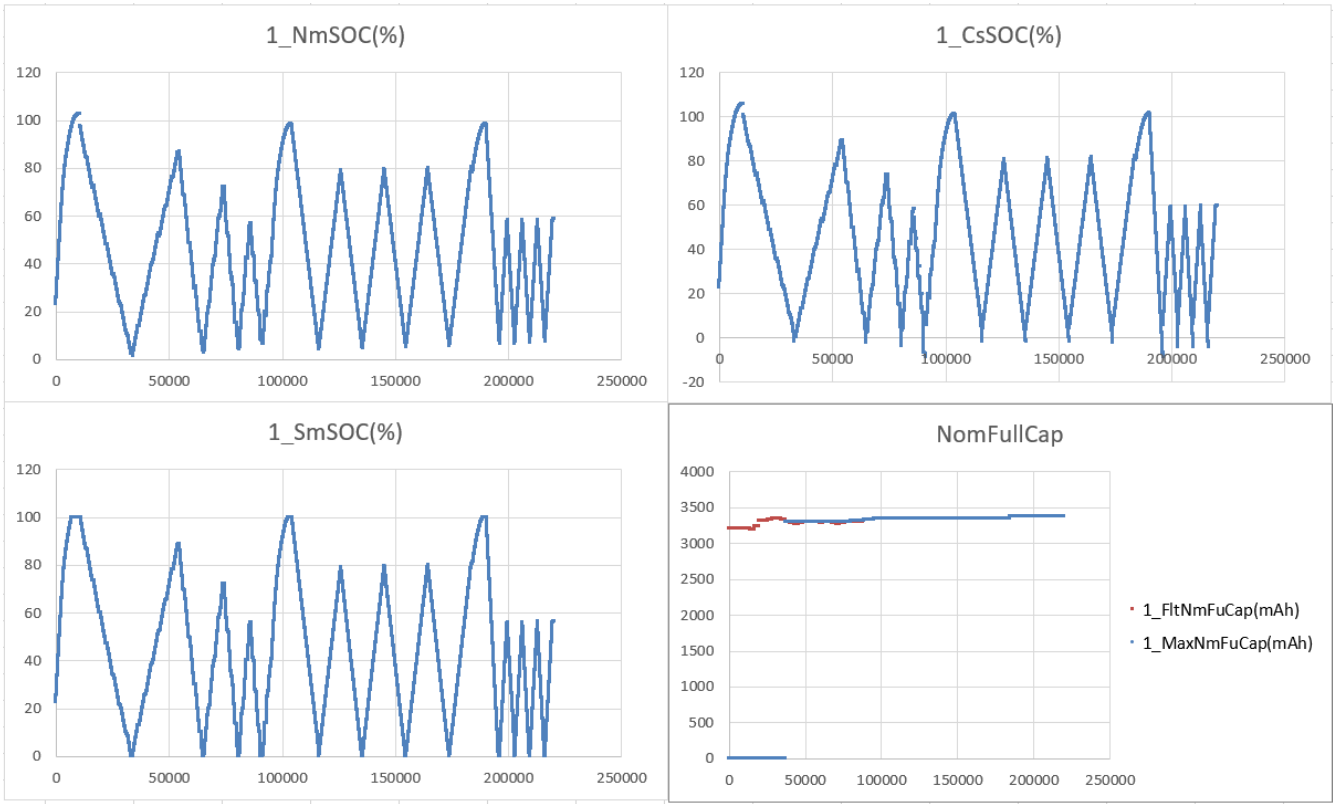
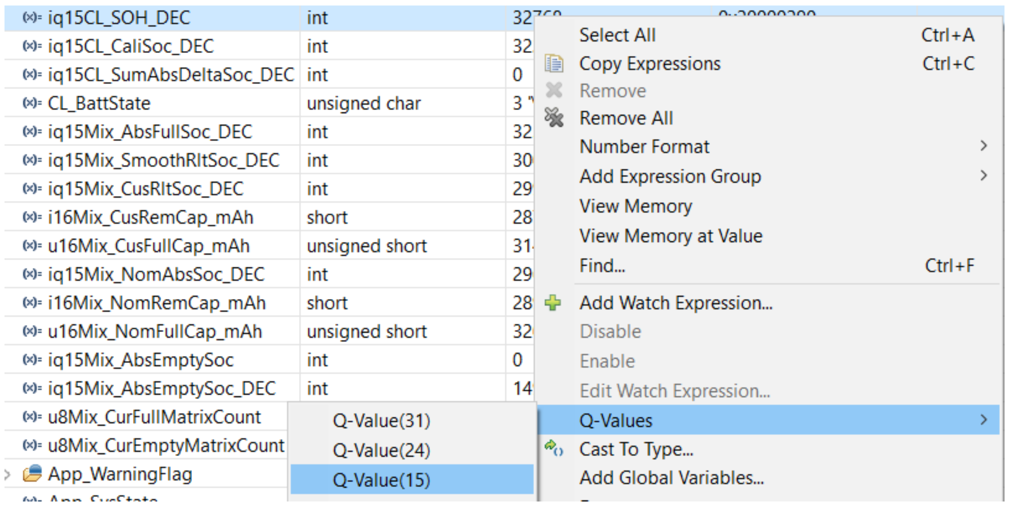


Figure 5-8. Battery Test Result

If users want to check more parameters under debug mode with Q format, then right-click the value and select the related Q-Value format.



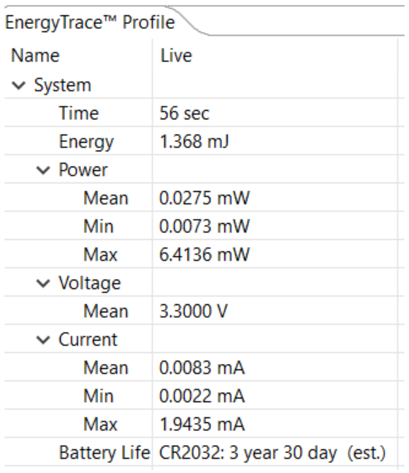
iq15CL_SOH_DEC	int	32760
iq15CL_CaliSoc_DEC	int	32
iq15CL_SumAbsDeltaSoc_DEC	int	0
CL_BattState	unsigned char	3
iq15Mix_AbsFullSoc_DEC	int	32
iq15Mix_SmoothRltSoc_DEC	int	30
iq15Mix_CusRltSoc_DEC	int	29
i16Mix_CusRemCap_mAh	short	28
u16Mix_CusFullCap_mAh	unsigned short	31
iq15Mix_NomAbsSoc_DEC	int	29
i16Mix_NomRemCap_mAh	short	28
u16Mix_NomFullCap_mAh	unsigned short	32
iq15Mix_AbsEmptySoc	int	0
iq15Mix_AbsEmptySoc_DEC	int	14
u8Mix_CurFullMatrixCount		
u8Mix_CurEmptyMatrixCount		
App_WarningFlag		

Figure 5-9. Read Q Values

5.1.3.2 Current Consumption Test

The functions of the MSPM0 gauge board focuses on evaluating the function and does not consider how to optimize the power consumption. The current tested based on the gauge board is a little high. To optimize, remove the tantalum capacitor, connect the temperature sensor to GPIO as the GND, and increase the voltage divider resistors.

Here is the current test result, under NO_OUTPUT mode, and removes the tantalum capacitor, the temperature sensor, and voltage divider resistors.



EnergyTrace™ Profile

Name	Live
System	
Time	56 sec
Energy	1.368 mJ
Power	
Mean	0.0275 mW
Min	0.0073 mW
Max	6.4136 mW
Voltage	
Mean	3.3000 V
Current	
Mean	0.0083 mA
Min	0.0022 mA
Max	1.9435 mA
Battery Life	CR2032: 3 year 30 day (est.)

Figure 5-10. Current Consumption

5.2 MSPM0G3507, BQ76952 and 4 LiFePO4 Batteries

Implementation features:

- Total implementation takes about 15K flash and 2.9K SRAM

Implementation advantage:

- High-performance MCU with CAN integrated
- A combination implementation for BMS from TI
- High-performance gauge algorithm

5.2.1 Hardware Setup Introduction

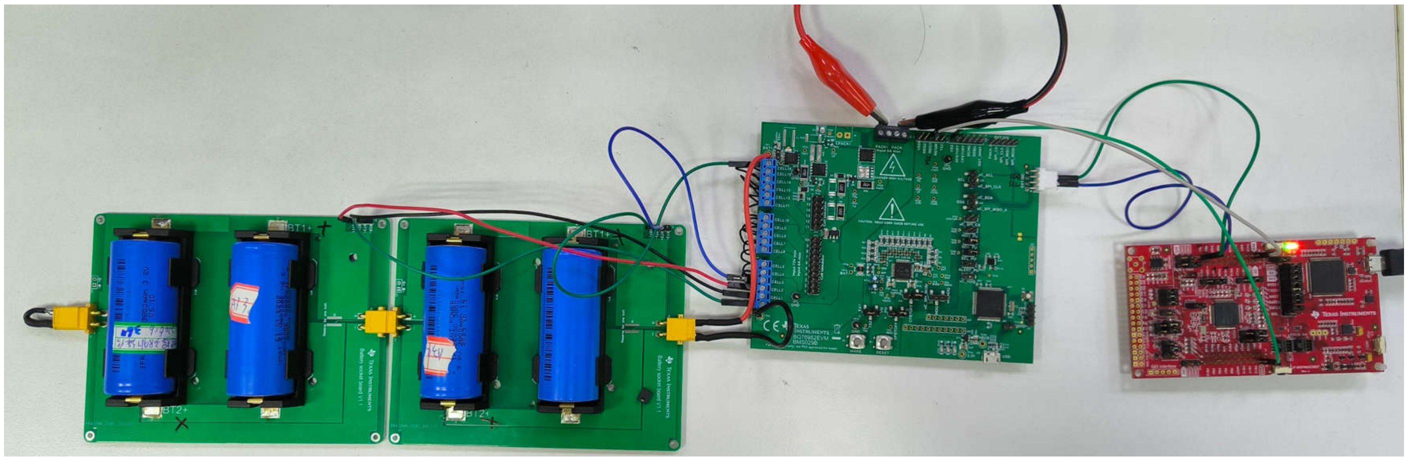


Figure 5-11. MSPM0G3507 LaunchPad and BQ76952EVM

The hardware board is built based on MSPM0G3507 LaunchPad and BQ76952EVM.

Refer to Figure 5-12 for the hardware connection. Connect power and I2C between MSPM0 LaunchPad (SDA: PB3, SCL: PB2)and BQ76952EVM (SDA: J17 PIN3, SCL:J17:PIN2). Remember to remove the cell simulation jumpers and add jumpers for I2C pullup.

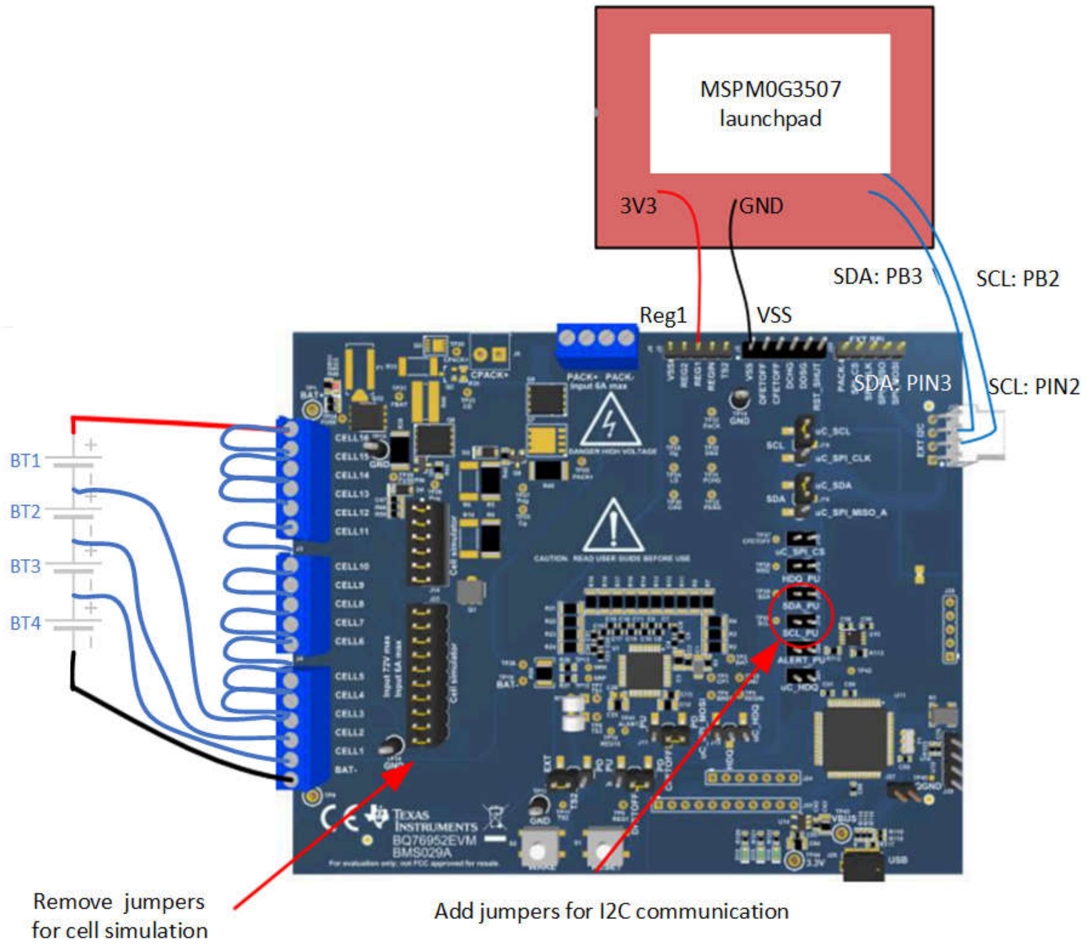


Figure 5-12. MSPM0G3507 and BQ75952EVM Block Diagram

If users want to use MSPM0G3507 LaunchPad in communication data input mode, then no other hardware change is needed compared with default MSPM0G3507 LaunchPad hardware set.

5.2.2 Software and Evaluation Introduction

Before starting software development and evaluation, TI recommends to refer to the MSPM0 Design Flow Guide user's guide to have a basic understanding about MSPM0 ecosystem. This helps users quickly learn about MSPM0 development.

Figure 5-13 shows the software project. The project and files related to the gauge algorithm has five parts. Other files are same for all the MSPM0 projects.

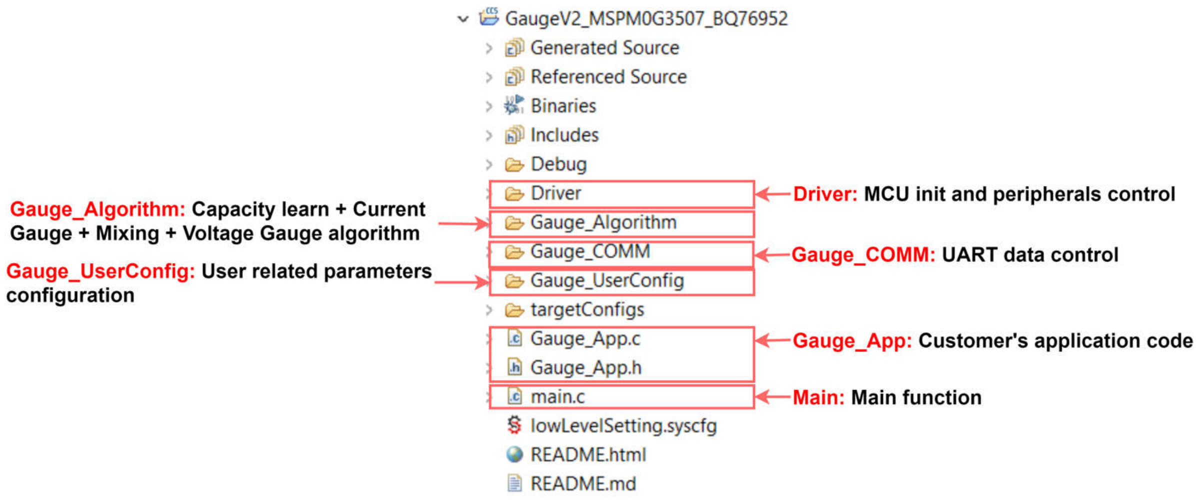

GaugeV2_MSPM0G3507_BQ76952
Generated Source
Referenced Source
Binaries
Includes
Debug
Driver
Gauge_Algorithm
Gauge_COMM
Gauge_UserConfig
targetConfigs
Gauge_App.c
Gauge_App.h
main.c
lowLevelSetting.syscfg
README.html
README.md
Gauge_Algorithm: Capacity learn + Current Gauge + Mixing + Voltage Gauge algorithm
Gauge_UserConfig: User related parameters configuration
Driver: MCU init and peripherals control
Gauge_COMM: UART data control
Gauge_App: Customer's application code
Main: Main function

Figure 5-13. MSPM0 Gauge Software Project View

For Gauge_UserConfig part, find the description in Section 4.3. The Gauge_Algorithm part introduction is in Section 2. The Driver part includes all the MCU related control. This prepares Icell, Vcell and Tcell data into Gauge_Algorithm. The Gauge_COMM part handles all the UART protocol. The Gauge_App part includes the high-level gauge algorithm calling. This is the place for customers to customize the functions. The Main part includes the highest system function code.

Follow the steps in Section 4 to update the configurations in Gauge_UserConfig folder:

- Generate battery models or use the default
- Update the configuration in tBattParamsConfig structure
- Update battGlobalParams_xx, and battGlobalParamsArray according to the battery cells. This implementation is four cells.
- Update the detection mode, communication mode, circuit table length and cell numbers

After programming the MSPM0 through XDS110, use the GUI to check and record the results.

5.2.3 Battery Test Cases

5.2.3.1 Performance Test 1 (Pulse Discharge)

Here is the test based on a 3800mAh LiFePO4 battery, under 25°C. u16MaxFullChgVoltThd setting is 3800mV. EmptyDhgVoltThd setting is 2300mV.

Note

Make sure the battery is settled before the MCU is powered and the battery is in rest state before testing, otherwise, the first SOC output is met with an error.

Here is the test pattern: do a pulse discharge 2 times. Figure 5-14 shows the condition of a battery Cell in the battery pack. Due to sourcemeter power limitations, only simple tests are run.

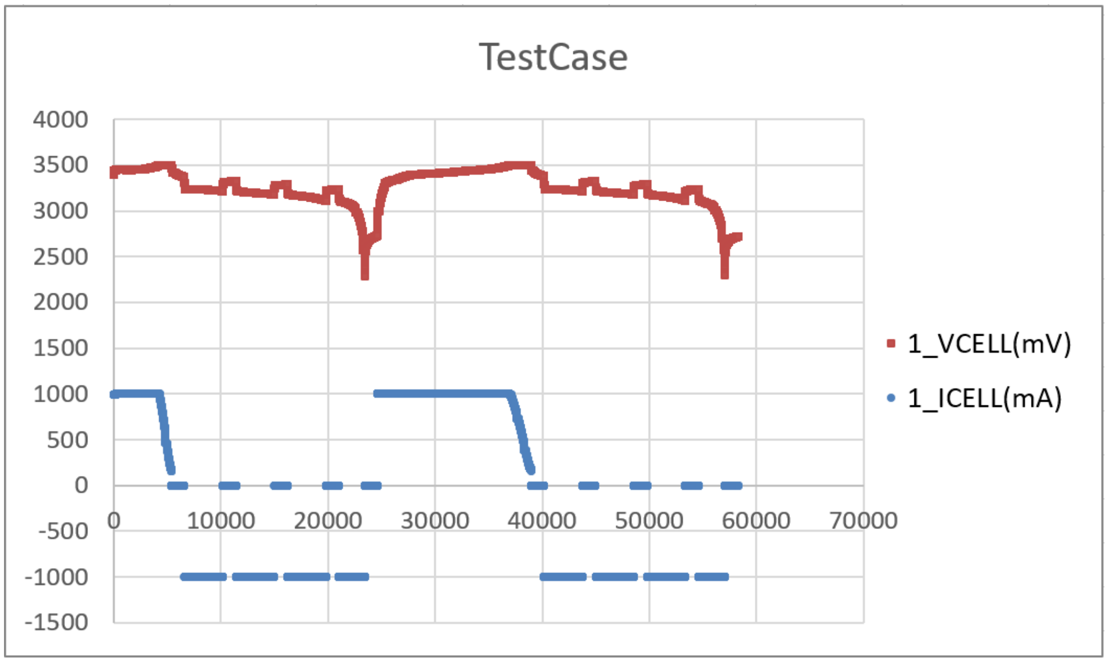


Figure 5-14. Battery Testcase

At the beginning, there are obvious gaps for NomSOC, CusSOC and SmoothSOC. See the test results in Figure 5-14. This is caused from the first OCV calibration error. The reason why this is so high is that the calibration point is under the LiFePO4 SOC-OCV flat area.

Here are the results for one cell. With the integrated digital filter, the NomSOC changes follow the real condition even in first discharge cycle. However, as the current load is only 1A, users cannot see much influence of EmptySOC in CusSOC.

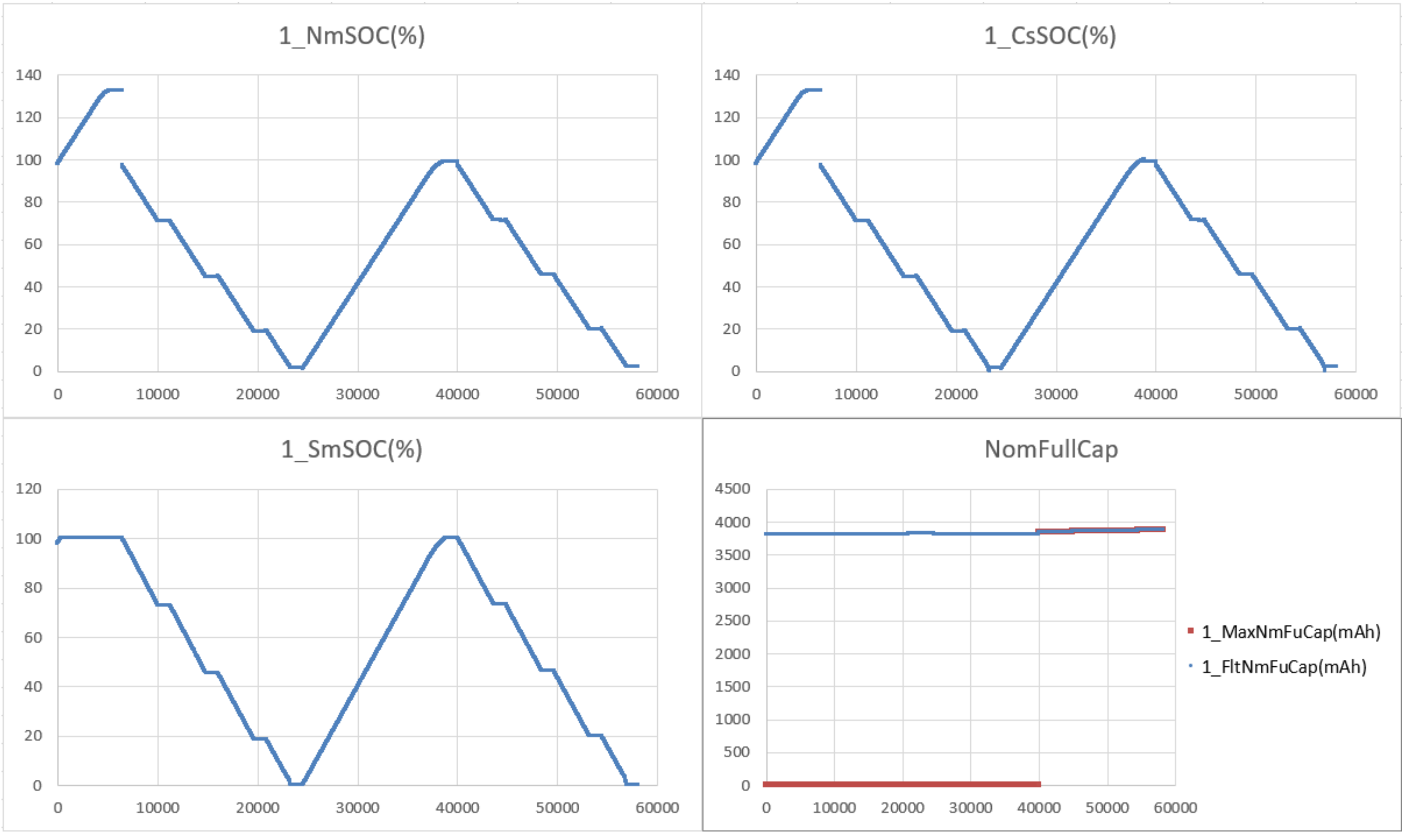


Figure 5-15. Battery Cell Test Result

Here is the result for the battery pack,PackSOC, which follows the minimum SmoothSOC of all the Cells in the battery pack. PackFullCap is the combination of all the CusFullCap values, which are influenced by EmptySOC and FullSOC. That is why the data jumps in the PackFullCap chart. This is the same for PackRemCap, which is the combination of all the CusRemCap.

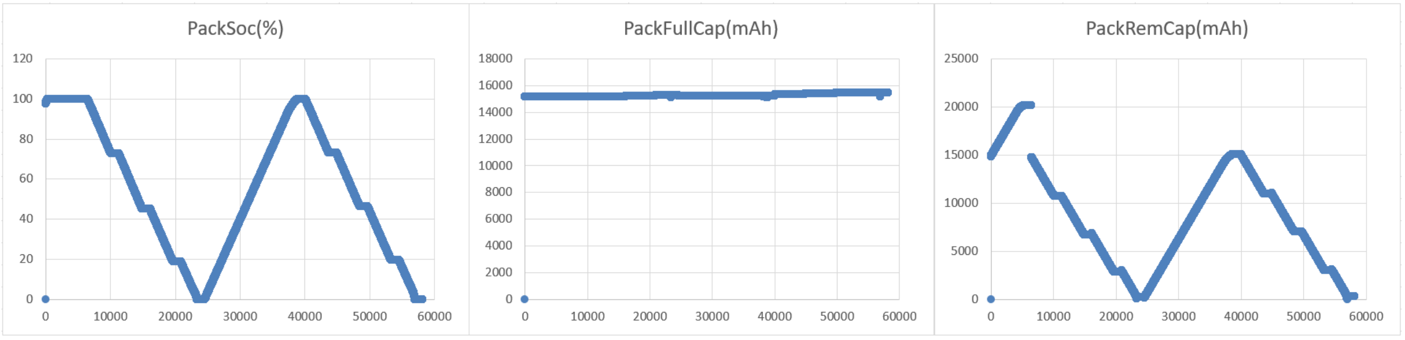


Figure 5-16. Battery Pack Test Result

5.2.3.2 Performance Test 2 (Load Change)

Here is the test based on a 3800mAh LiFePO4 battery, under 25°C. The u16MaxFullChgVoltThd setting is 3800mV. The EmptyDhgVoltThd setting is 2300mV.

Note

Make sure the battery is settled before the MCU is powered and the battery is in rest state before testing, otherwise, the first SOC output is met with an error.

Here is the test pattern: do constant discharge 2 times and then change the load. The Figure 5-17 shows the condition of a battery cell in the battery pack. Due to sourcemeter power limitation, only simple tests are run.

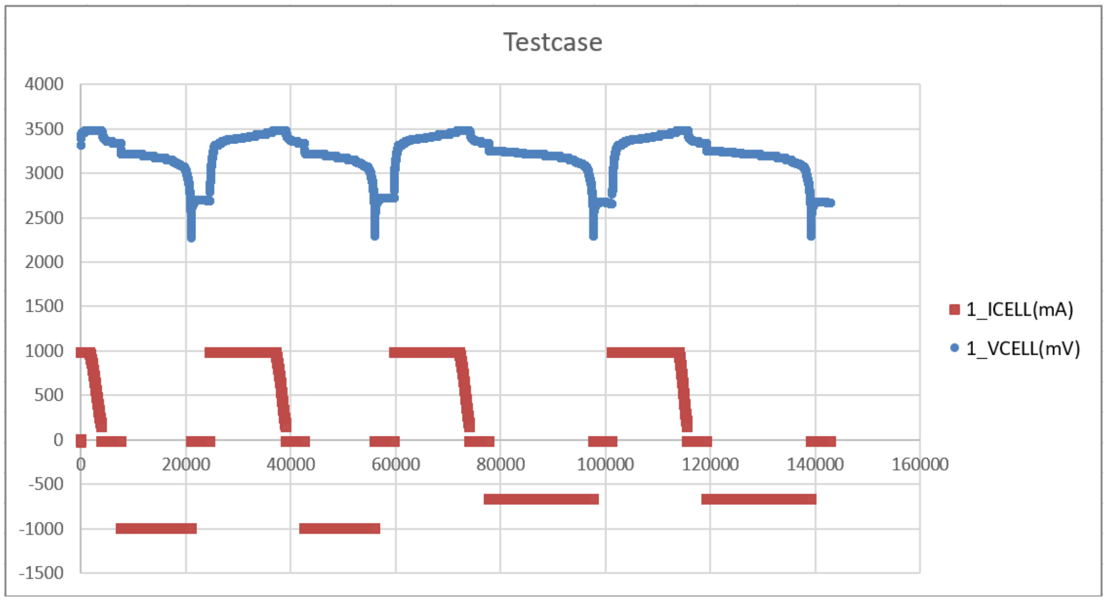


Figure 5-17. Battery Test Case

See the test result in Figure 5-17. At beginning, there is a obvious gap for NomSOC, CusSOC and SmoothSOC. This is caused from the first OCV calibration error.

Due to residual learn algorithm, see that the SmoothSOC can perform between 0% and 100% when the voltage is reaching the end of discharge voltage (2300mV). Remember at the same time, the EmptySOC needs learning cycles, which means if users do not input iq15AbsEmptySocMatrixInput, then the SmoothSOC error is large when the battery reaches the end of the first discharge voltage.

For different NomFullCap, the FltNomFullCap is updated after almost every rest. With the digital filter help, the NomFullCap gets more and more accurate. After the MaxNomFullCap changes from 0 to a value, this means the output NomFullCap is with an acceptable accuracy.

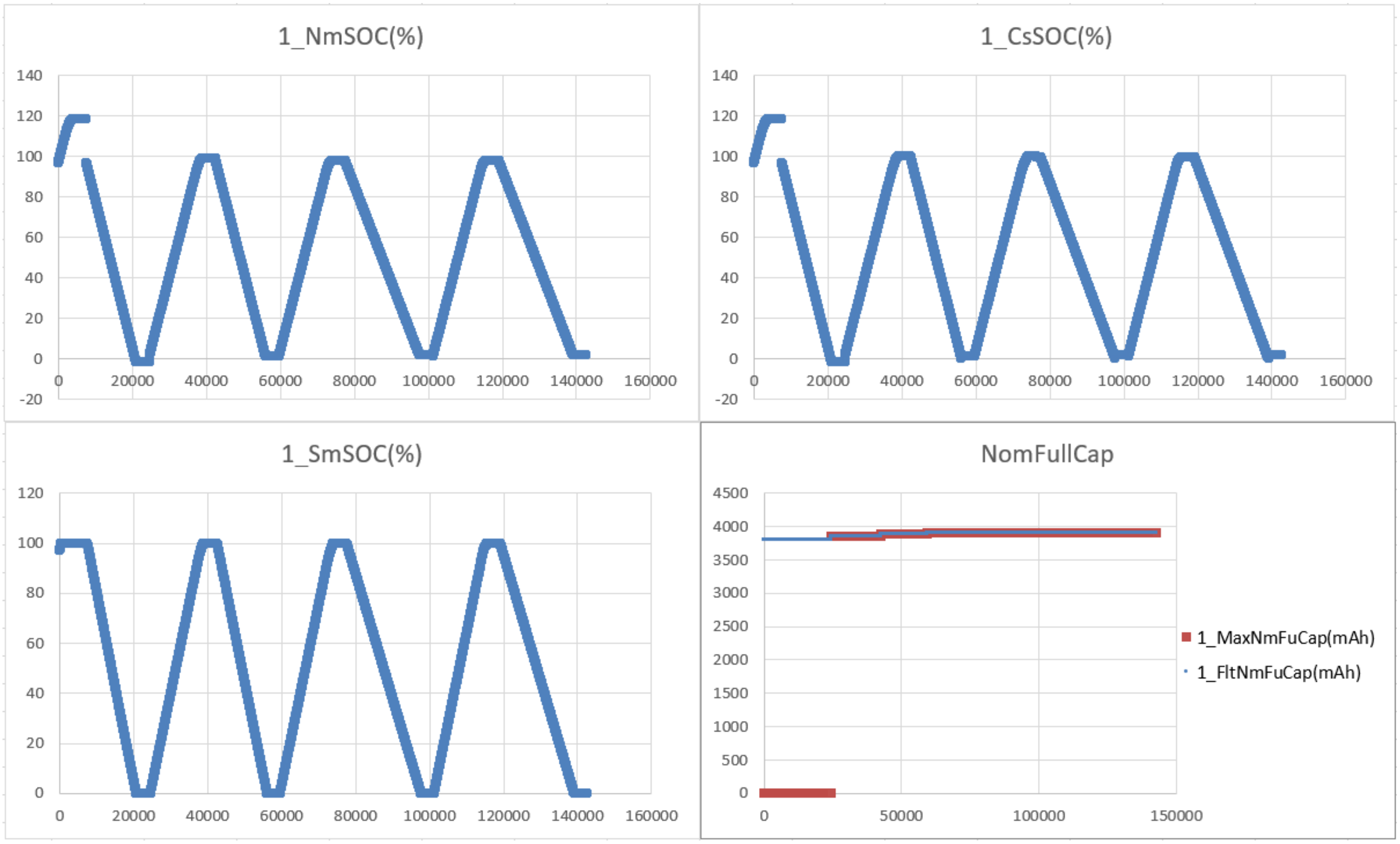


Figure 5-18. Battery Test Result

Figure 5-19 the result for battery pack.

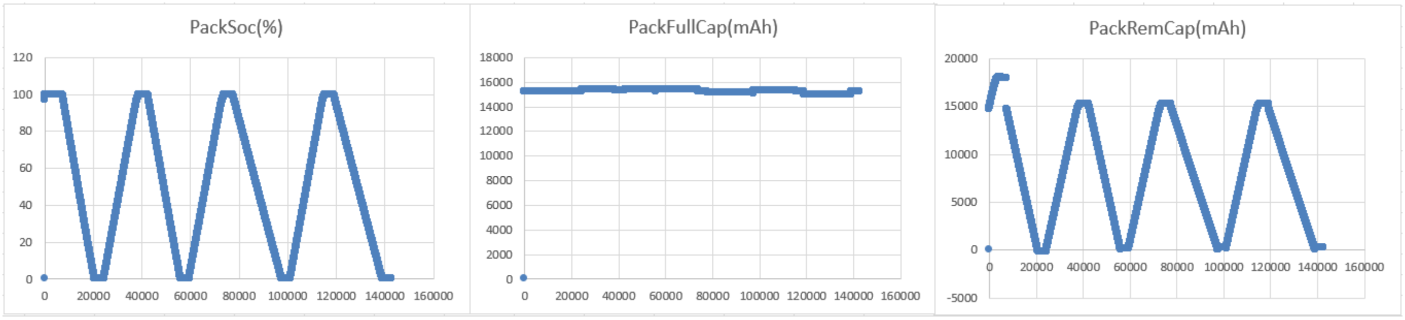


Figure 5-19. Battery Test Result

5.3 MSPM0L1306 and BQ76905

This implementation is used to show the setup between MSPM0L1306 and BQ76905 with a setup of 5 cells to accelerate the evaluation speed on typical BQ devices. Unlike with BQ76952, BQ76905 does not have the passive equalization function, which needs to be implemented by users themselves.

Here is the hardware connection:

Table 5-1. Hardware Connection

	LP-MSPM0L1306	BQ76905 EVM
GND	J7-1	J4-1
SCL	PA1	J4-2
SDA	PA0	J4-3

Here is the real setup picture. Remember to short CELL5 to CELL4 and CELL3 to CELL2 cell input terminals before doing the evaluation.

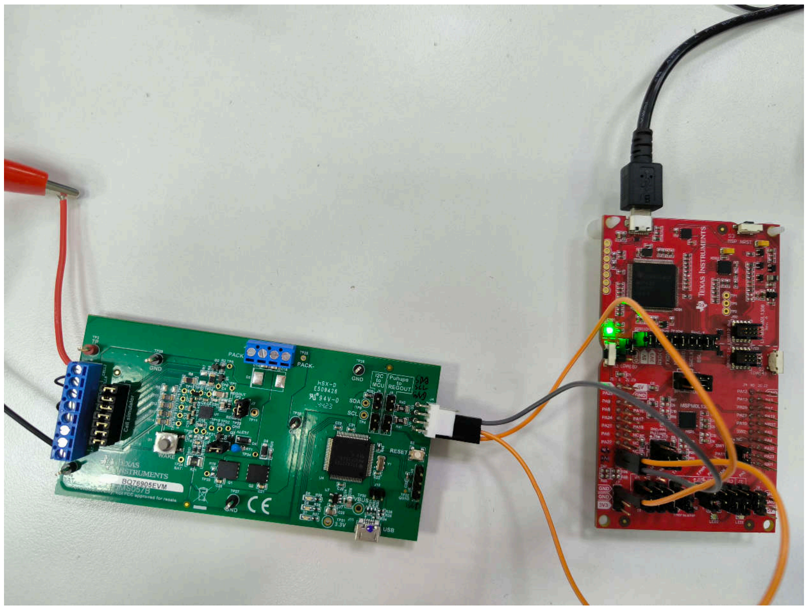


Figure 5-20. Hardware Setup

6 Summary

This document outlines a Level 2 BMS gauge implementation utilizing MSPM0 microcontroller. The solution achieves long-term State-of-Charge (SOC) accuracy (±1.5% under typical conditions) by integrating a dual-source data fusion algorithm. Users can follow this document to finish the battery parameterization, algorithm evaluation and performance analysis. With the limited parameter adjustment, this solution can be fit into any kinds of Li-ion batteries and meet users' features and accuracy requirement.

7 References

- Texas Instruments: *MSPM0 Gauge L1 Solution Guide*,
- Texas Instruments: *A Self-Calibratable Current Detection Solution Based on MSPM0*
- Texas Instruments: *MSPM0 MCUs Development Guide*

8 Revision History

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

Changes from Revision A (June 2025) to Revision B (June 2025) **Page**

- Updated development package link in the Abstract of this document..1
- Updated development package link in Section 1..3