Battery Fuel Gauge [Smart LiB Gauge] for 1-Cell Lithium-ion/Polymer with LC709209F

AND90162/D

LC709209F is a Fuel Gauge for 1−Cell Lithium−ion/Polymer batteries. It is a part of our *Smart LiB Gauge* family of Fuel Gauges which measure the battery RSOC (Relative State Of Charge) using its unique algorithm called *HG−CVR2*. The *HG−CVR2* algorithm provides accurate RSOC information even under unstable conditions (e.g. changes of battery; temperature, loading, aging and self−discharge).

This application note will explain how to initialize various parameters for the selected battery to start a higher accuracy gauging. Users can set various registers based on their application requirement using the notes, guidelines and examples given in this note. Sample program codes explained at the end of the note will provide various guideline on how this device communicates with the host side application processors.

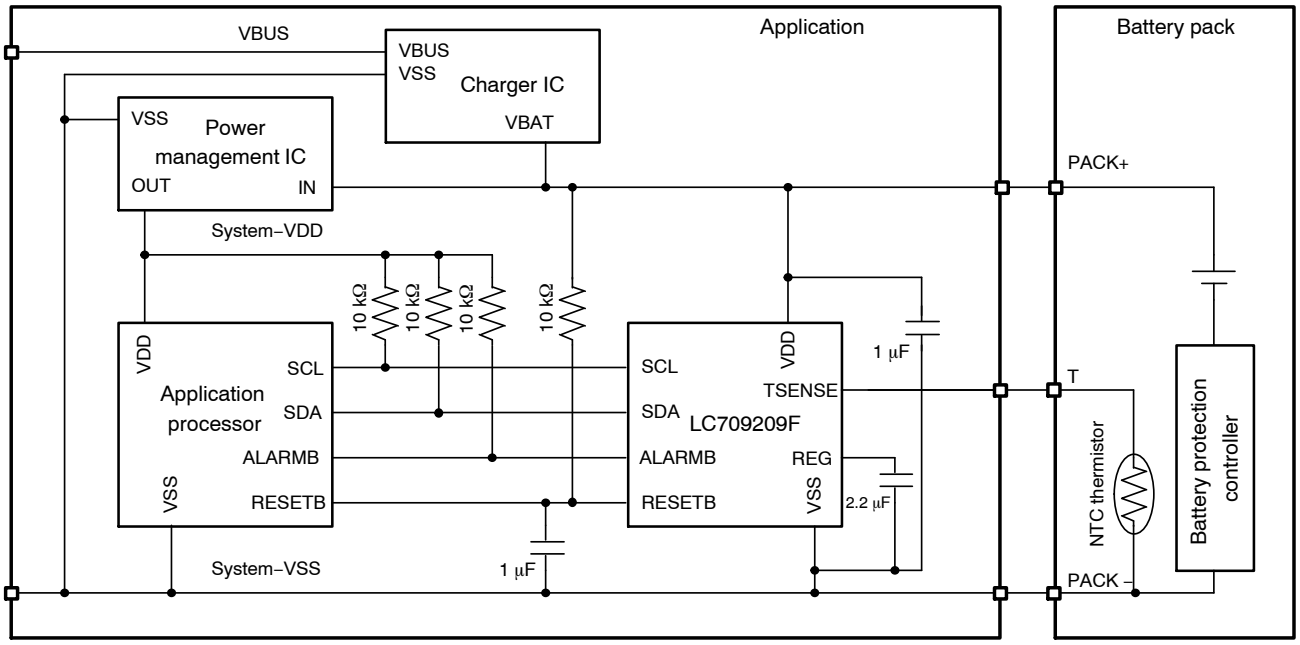


Figure 1. An Example of an Application Schematic using LC709209F

Application Circuit Diagram

Figure 1 shows the application circuit diagram for LC709209F. It is recommended that a TSENSE thermistor is placed inside the battery pack or in contact with the battery to measure the battery temperature. The application processor can control the fuel gauge with I^2C communication, receive an alarm and reset it directly using the RESETB pin. The fuel gauge can also reset itself with the built−in reset circuits, so it is possible to disconnect the RESETB pin from the processor. In that case, connect the RESETB pin to the battery's positive pin without a resistor.

Evaluation Tools

LC709209FXE−01−GEVB Evaluation Board

An evaluation board with a GUI controller is available to evaluate LC709209F. The board features a USB−IF microcontroller and LC709209F. If a battery is connected to the board and then connected to a PC using a USB cable, users can read and write any registers of the LC709209F, and log the cell temperature, cell voltage, RSOC and some other register values using the GUI. The logged data can be saved as a text file. Please refer to the documents given in Table 1 for further details about the board.

STR−SMARTLIBGAUGE−GEVK Smart LiB Gauge Automatic Support Tool (EOL)

The Smart LiB Gauge Automatic Support Tool automatically evaluates the battery parameters for LC709209F. The battery parameters for the device are explained as the adjustment parameter (APA) and battery profile in the next section. The optimized battery parameters for a target battery will improve the RSOC accuracy. For the evaluation, the tool discharges a target battery using the on−board programmable load and measures the cell voltage and temperature.

The tool works in the Strata Developer Studio™. Please refer to the documents in the Strata Developer Studio for further details about the tool.

Table 1. EVALUATION BOARDS AND DOCUMENTS FOR THE TARGET DEVICE AND BATTERY

Evaluation Board	Target Device	Battery Type	Related Documents
LC709209FXE−01−GEVB	LC709209FXE−01TBG	01, 04, 05, 06, 07	LC709209FXE−01−GEVB_Test Procedure.pdf LC709209FXE−01−GEVB_SCHEMATIC.pdf LC709209FXE−01−GEVB_GERBER.zip LC709209FXE−01−GEVB_BOM.pdf
STR−SMARTLIBGAUGE−GEVK (EOL)			STR−SMARTLIBGAUGE−GEVK_USER_GUIDE.PDF STR−SMARTLIBGAUGE−GEVK_SCHEMATIC.PDF STR−SMARTLIBGAUGE−GEVK_BOM_ROHS.PDF (Note 1)

1. There are other related documents in the Strata Developer Studio.
   Link to related documents: https://www.onsemi.com/products/power−management/battery−management/battery−fuel−gauges/LC709209F

Parameter Initialization

In order to start the RSOC measurement with this device, you must initialize some basic parameters in advance. Table 2 shows the parameters and the corresponding register names with the command code to set them individually. The parameters specified as Mandatory in Table 2 are the basic parameters required to measure the RSOC. Optional parameters can be initialized if the user's application requires the given functionality. The detailed method on how to set the required parameters is given below.

Battery Profile (0x12)

The device is installed with five types of Battery profiles. Users must select an appropriate profile for their applications based on the type of battery used. Please check the battery nominal voltage and charging voltage against Table 3 and select the Battery Type where either of them matches. To set the Battery Type to be used, write the value specified in Table 3 to Change of The Parameter register (0x12) of the Fuel Gauge. For example write 0x01 to Change of The Parameter to select the Battery Type−04.

Users can also select the suitable battery profile by using the Smart LiB Gauge Automatic Support Tool to calculate the optimized parameters. Please refer to the user guides in the Strata Developer Studio for the details.

Table 2. PARAMETER VS REGISTER

Command Code	Register Name	Parameter	Mandatory or Optional	Unit
0x06	TSENSE Thermistor B	B−constant of a TSENSE thermistor	Mandatory	K
0x0B	APA	Adjustment parameter for RSOC measurement	Mandatory	−
0x0C	APT	Delay time to temperature sampling	Optional	−
0x12	Change Of The Parameter	Battery profile	Mandatory	−
0x1C	Termination Current Rate	Termination current rate at the end of charging	Optional	0.01C
0x1D	Empty Cell Voltage	Empty Cell Voltage	Optional	mV

Table 3. BATTERY PROFILE VS REGISTER

IC Type	Battery Type	Nominal / Rated Voltage	Charging Voltage	Number of The Parameter (0x1A)	Change of The Parameter (0x12)
LC709209F	01	3.7 V	4.2 V	0x1001	0x00
	04	UR18650ZY (Panasonic)			0x01
	05	ICR18650−26H (SAMSUNG)			0x02
	06	3.8 V	4.35 V		0x03
	07	3.85V	4.4V		0x04

APA Value to Improve RSOC Accuracy

APA values are parameter to fit a pre−installed battery profile into target battery characteristics. They are set in APA register (0x0B). Appropriate APA values for the target battery will improve RSOC accuracy. Users can select either of the two following approaches to obtain the APA value.

- Design capacity to typical APA conversion table
- Smart LiB Gauge Automatic Support Tool

The Design capacity to typical APA conversion table is Table 4. Typical APA values can be taken from the design capacity of the cell in the table. Use capacity per 1−cell of the table if some batteries are connected in parallel. Calculate APA values using linear supplement if your required design capacity is not shown in the table. See eq. 1 for how to calculate the APA value manually. An example for a 1500 mAh battery with corresponding DEC value for their HEX is also shown.

$$\text{APA value} = \text{Lower\_APA} + (\text{Upper\_APA} - \text{Lower\_APA}) \times \frac{\text{Capacity} - \text{Lower\_Cap.}}{\text{Upper\_Cap.} - \text{Lower\_Cap.}} \quad \text{(eq. 1)}$$

Calculation example for a 1500 mAh Battery Type−01.

$$\text{APA value} = 45 : 0x2D + (58 : 0x3A - 45 : 0x2D) \times (1500 - 1000)/(2000 - 1000) \approx 52 : 0x34 \quad \text{(eq. 2)}$$

The upper 8 bits and the lower 8 bits of the APA register correspond to the charging and discharging adjustment parameters respectively. See Table 5 for the bit configuration. Table 4 shows the case where both the upper and lower bits have the same value. For example, set the value in the APA register to 0x0D0D for an APA value of 0x0D.

The Smart LiB Gauge Automatic Support Tool automatically evaluates the optimum APA by measuring the target battery. The evaluated APA will improve the RSOC accuracy than the APA from the conversion table. Please refer to the user guides in the Strata Developer Studio to evaluate APA with the tool.

Table 4. DESIGN CAPACITY TO TYPICAL APA CONVERSION TABLE

Design Capacity / Cell (Note 2)	APA[15:8], APA[7:0]				Design Capacity / Cell (Note 2)	APA[15:8], APA[7:0]	
	Type−A01	Type−06	Type−07			Type−04	Type−05
50 mAh	0x13, 0x13	0x0C, 0x0C	0x03, 0x03		2600 mAh	0x10, 0x10	0x06, 0x06
100 mAh	0x15, 0x15	0x0E, 0x0E	0x05, 0x05				
200 mAh	0x18, 0x18	0x11, 0x11	0x07, 0x07				
500 mAh	0x21, 0x21	0x17, 0x17	0x0D, 0x0D				
1000 mAh	0x2D, 0x2D	0x1E, 0x1E	0x13, 0x13				
2000 mAh	0x3A, 0x3A	0x28, 0x28	0x19, 0x19				
3000 mAh	0x3F, 0x3F	0x30, 0x30	0x1C, 0x1C				
4000 mAh	0x42, 0x42	0x34, 0x34	–				
5000 mAh	0x44, 0x44	0x36, 0x36	–				
6000 mAh	0x45, 0x45	0x37, 0x37	–				

2. Use capacity per 1−cell if some batteries are connected in parallel.

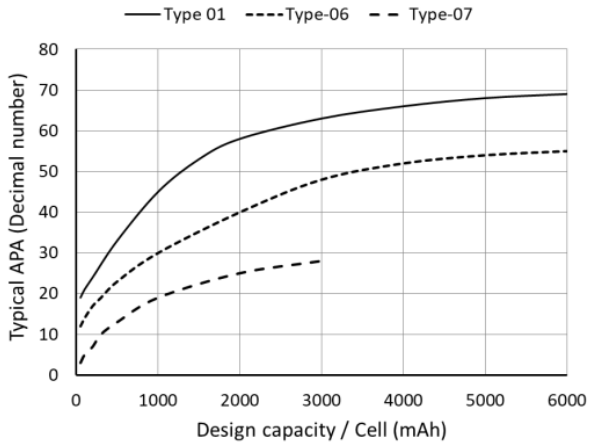


Figure 2. Typical APA of Type−01/06/07

Table 5. BIT CONFIGURATION OF APA REGISTER (0X0B)

Bit	Function
APA[15:8]	APA value for charging adjustment
APA[7:0]	APA value for discharging adjustment

B Constant of NTC Thermistor (0x06)

This device can support 10 kΩ NTC thermistor, and this section explains how to find the appropriate B constant value to set in the Thermistor B register (0x06). Cell temperature (TSENSE) is an essential parameter used for the battery measurement. You must set an appropriate value in the TSENSE Thermistor B register (0x06) unless the application processor provides the battery temperature directly to this device (using I^2C mode).

The device calculates temperature assuming that the resistance value of the thermistor follows eq. 3.

$$R = R_0 \times \exp B(1/T - 1/T_0) \quad \text{(eq. 3)}$$

R: Thermistor resistance in T (K)
R_0: 10 kΩ
B: B constant (K)
T: Temperature (K)
T_0: 298.2 K (25°C)

Table 6 shows an example for the relationship between the resistance and temperature of an available 10 kΩ thermistor. If similar values are given in the data sheet for the thermistor used, please substitute the thermistor resistance at each temperature into eq. 4 to calculate temperature.

$$T = 1 / [1/T0 + 1/B \times \ln(R/R0)] \quad \text{(eq. 4)}$$

Sample plots using eq. 4 are shown in Figure 3. The horizontal axis shows the actual temperature and the vertical axis shows the difference between the temperatures calculated from the resistance value of a thermistor (eq. 4) with the actual temperature. Three B constant values are used to calculate the vertical axis. Select a B constant value that minimizes the absolute value of the vertical axis in the temperature range where RSOC accuracy is required. In Figure 3, B constant = 3400 K will give higher RSOC accuracy for the given range of temperature.

Another example is shown in Table 7. If only the temperature range and B constant are specified in the thermistor datasheet, select a B constant value that fits with the user's application temperature range so that higher RSOC accuracy can be obtained.

Table 6. 10 kΩ NTC THERMISTOR EXAMPLE (1)

Temperature	Resistance	Temperature	Resistance
−20°C	71 kΩ	30°C	8.3 kΩ
−10°C	44 kΩ	40°C	5.8 kΩ
0°C	28 kΩ	50°C	4.1 kΩ
10°C	18 kΩ	60°C	3.0 kΩ
20°C	12 kΩ	70°C	2.2 kΩ

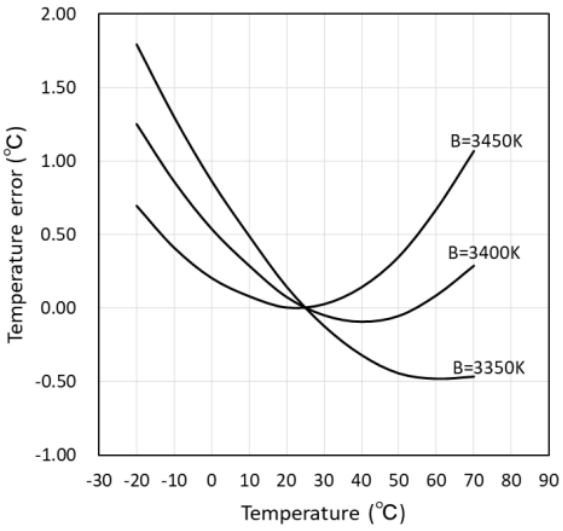


Figure 3. An Example of Temperature Error which is Calculated from a Thermistor Resistance

Table 7. 10 kΩ NTC THERMISTOR EXAMPLE (2)

R0 or R25	B Constant		
	25°C to 50°C	25°C to 80°C	25°C to 100°C
10.0 kΩ	3435 K	3474 K	3595 K

Thermistor Measurement Delay (0x0C)

This section explains about the APT (Adjustment Pack Thermistor) delay and the behavior of TSENSE pin while measuring temperature with an NTC thermistor. This device optimizes the temperature measurement interval automatically based on the battery current flow. The measurement interval ranges between a few seconds to a minute. 10 kΩ pull−up resistor is integrated with TSENSE in the device as shown in Figure 4. This resistor is connected to the REG supply only during temperature measurement. The pin remains in a high impedance state except while measuring the temperature. Figure 5 shows an example of TSENSE waveform during temperature measurement. When the voltage on TSENSE gets stabilized while the thermistor is connected to the pin, the voltage on the pin is measured for finding the target temperature. The pull−up resistor automatically gets disconnected from REG power supply after a successful temperature measurement.

The APT delay shown in Figure 5 is the time delay between when REG power is supplied to the thermistors and when voltage measurement begins. The APT register shown in Table 2 is used to change the APT delay. APT delay is calculated using eq. 5 based on the value set in the APT register.

$$\text{APT delay} = 0.167\ \mu\text{s} \times (200 + \text{APT}) \qquad \text{(eq. 5)}$$

To improve the accuracy of temperature measurement, the voltage on TSENSE must be stabilized before the measurement starts. The APT delay parameter provides a delay time for the system to wait for voltage stability. For most applications, the pre−defined APT delay time in the device is sufficient for voltage stability. However, in the case of a battery pack example shown in Figure 6, the APT delay must be considered. The capacitive element is placed in parallel with the thermistor in the given example. It is assumed that it will take a longer time for the voltages of TSENSE to stabilize. It also takes a longer time to stabilize at lower temperatures as thermistor resistance increases when temperature decreases. Therefore, APT delay should be considered according to the thermistor resistance at low temperature.

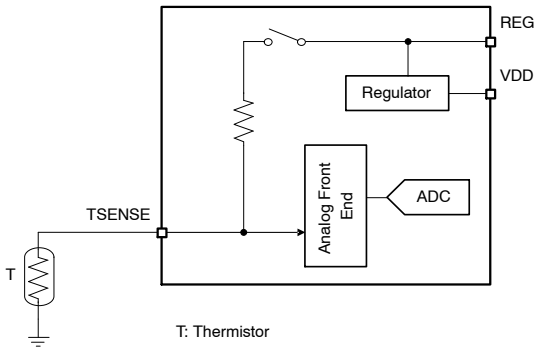


Figure 4. TSENSE Port Block Diagram

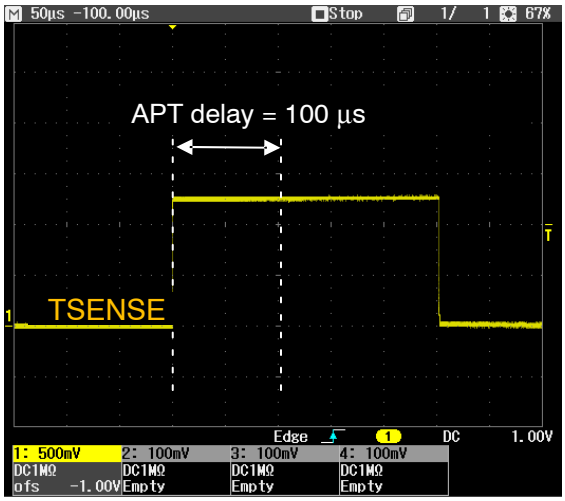


Figure 5. TSENSE Pulse at 25°C (APT = 0x0190)

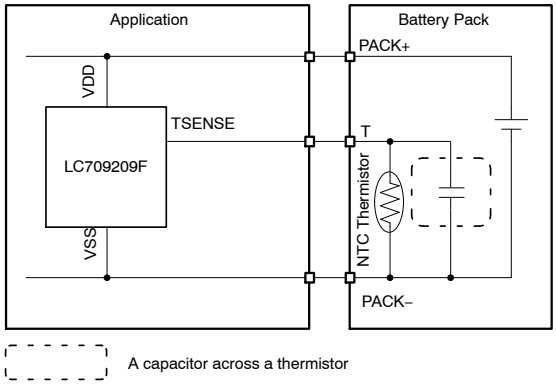


Figure 6. An Example of a Capacitor Across the Thermistor

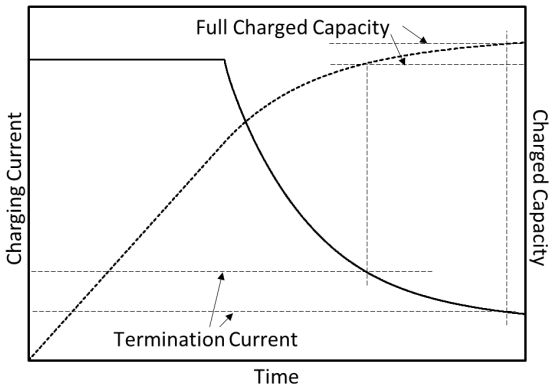


Figure 7. Termination Current and Full Charge Capacity

Design Capacity (0x18)

Design capacity or Typical capacity or Nominal capacity can be found in the battery's datasheet.

Termination Current Rate (0x1C)

This termination current rate is used to adjust RSOC reporting so that 100% is reported at the end of the charging period, or even before the charger finishes charging. There are several factors that can hinder RSOC from reaching 100% (Full Charge Status), for example when the battery charger varies the termination current. In general, reaching the termination current is the condition to stop charging for Constant Voltage (CV) lithium−ion batteries. In the CV mode a lithium−ion charger decreases the charging current continually as charging progresses, and stop charging when the termination current is reached. Therefore an increased termination current value will result in a reduction of the Full Charge Capacity (FCC) at the end of the charging process as shown in Figure 7. This register value adjusts the RSOC reporting so that 100% is reported for such a reduced FCC.

Termination current is set for the charger or given in the datasheet of the battery. Users should apply the maximum current in them for this register. It can prevent RSOC reporting from being less than 100% at the end of charge.

The termination current rate set in this register is calculated by dividing the termination current with design capacity. The unit of the calculation result is C. (i.e. when the design capacity is 3000 mAh and the termination current is 150 mA, the termination current rate is 0.05C. In that case the register value is 0x05, since the unit of the register is 0.01C.) The Fuel Gauge only supports 0.02C or higher termination current rate. Set values of 0.02C or higher for lower termination current rates.

Empty Cell Voltage (0x1D)

The lowest cell voltage that the application requests. The lower side of RSOC (0x0D) is adjusted by this value. Refer to the RSOC rescaling section.

FUNCTIONAL DESCRIPTION

Get Initial RSOC after Power−on Reset

This device starts the initialization sequence automatically after both of the power−on reset and the RESETB pin are released. Please refer to the Fuel Gauge datasheet for the duration of the initialization sequence. During the initialization sequence, the device acquires the Cell voltage for the RSOC initialization. The initial RSOC is obtained using the Open Circuit Voltage (OCV) of the battery, which is the measurement of the battery voltage when no load is applied. The device has a built−in OCV look−up table. The measured cell voltage is translated using the table to obtain the new Initial RSOC. After the completion of the initialization sequence, the acquired initial RSOC is set in the RSOC (0x0D) and the ITE (0x0F) registers.

Obtaining an Initial RSOC using Before RSOC

A battery or charger may supply the power to the VDD terminal of the device. If the RSOC value after the completion of the initialization sequence is not as expected, it is assumed that the battery was charged or discharged during that period. If the battery is not charged, the maximum voltage is suitable for more accurate initial RSOC. Because the maximum voltage is closer to the OCV. Try all "Before RSOC" commands and read RSOC (0x0D) to search the maximum voltage. The higher RSOC after the commands is caused by the higher voltage. Voltage sampling is performed four times during the initialization sequence as shown in Figure 8. 1st sampled Cell voltage is referenced to get the Initial RSOC. Before RSOC commands can initialize RSOC using the other 2nd to 4th sampled voltage. Table 8 shows the Before RSOC commands to initialize RSOC using each sampled voltage.

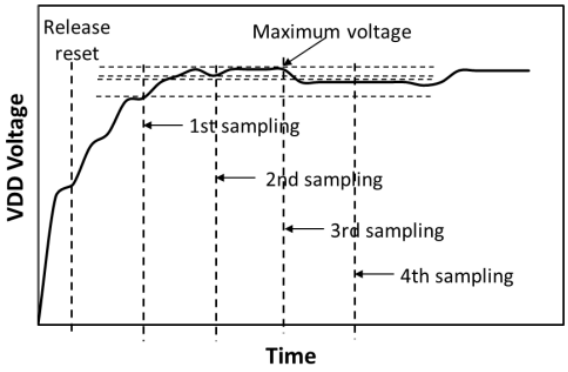


Figure 8. Sampling Order for Before RSOC Command

Table 8. BEFORE RSOC COMMAND

Command Code	DATA	Sampling Order of Battery Voltage for RSOC Initialization
0x04	0xAA55	1st sampling
	0xAA56	2nd sampling
	0xAA57	3rd sampling
	0xAA58	4th sampling

Power−on Using Charger

In general, the battery protection controller as shown in Figure 1 disconnects the battery when an overvoltage or overcurrent is detected. The power supply to the Fuel Gauge is also stopped at that time. In general, the battery protection controller reconnects the battery when it detects a voltage supply from the charger. In such cases, the charger starts to supply power to the Fuel Gauge first. Therefore, the voltage acquired by the device in the initialization sequence is the charging voltage of the charger. Depending on the charging voltage, a higher RSOC is obtained. Therefore, accurate initial RSOC cannot be obtained during charging. After the charger has stopped, the following two functions can be used to fix this problem.

- Initial RSOC Command (0x07)
- Automatic Convergence of the Error

Initial RSOC Command initializes the RSOC using the Cell Voltage obtained after writing the command. At this time, application is running for I^2C communication and so on, so the battery is not completely unloaded. However, if the load is 0.025C or less (i.e. less than 75 mA for 3000 mAh design capacity battery), this command can return an RSOC reading that is close to the actual value.

Automatic Convergence of the Error is a function that automatically corrects RSOC errors. This feature corrects 30% errors in around one hour regardless of the load connected. Figures 9 and 10 are examples of modifications made using this feature. This function can also fix the case of lower RSOC problem during the battery discharging conditions. To enable Automatic Convergence of the Error function, set this device to Operational mode and set the Current Direction (0x0A) register to Auto mode.

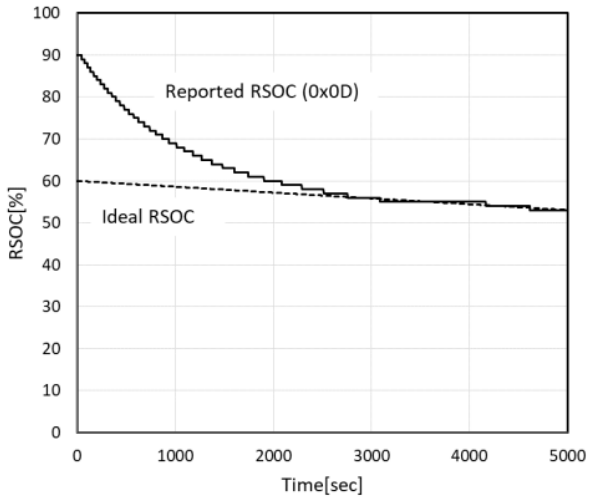


Figure 9. An Example of RSOC Automatic Convergence with 0.05C Load Current. RSOC: 90% to 60%

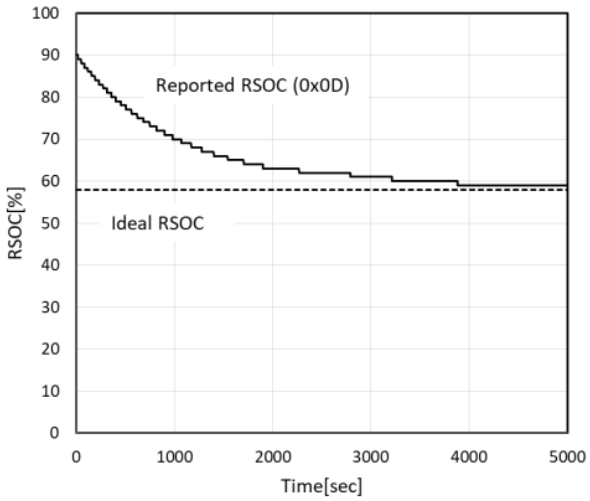


Figure 10. An Example of RSOC Automatic Convergence without Load Current. RSOC: 90% to 58%

Selection and Initialization of Profile

The OCV look−up table for obtaining initial RSOC is different for each Battery profile. The initial RSOC is obtained using the Battery profile specified by the initial value of Change of The Parameter (0x12), which is stored in the built−in Non Volatile Memory (NVM) of the device. If an initial value to select an appropriate profile has already been programmed in the NVM, you can omit the I^2C command described below. Refer to "Built−in NVM Writing Protocol" section for instructions on how to program the NVM.

If the appropriate profile is not programmed in the NVM, you must write it into the Change of the Parameter register using I^2C command. The device also automatically initializes the RSOC, when it receives the write. For the

initialization, the OCV look−up table of the selected profile and the first sampled cell voltage is used.

Use the above−mentioned functions (i.e. Before RSOC command, Initial RSOC command, and Automatic Convergence of the Error) to correct the initial RSOC after selecting an appropriate Profile for your applications.

Temperature Measurement

The Status Bit (0x16) controls temperature measurement with the thermistor. Set the bit 0 to 1 to measure the temperature with the attached thermistor. The bit selection details are shown in Table 9. Battery temperature information is an essential parameter for the RSOC measurement. If the thermistor in the battery pack is connected to another device, the Fuel Gauge cannot measure the battery temperature using the thermistor. In that case, set TSENSE1 to I^2C mode. Please note that the device cannot update the Cell temperature in I^2C mode. The application processor must write the battery temperature to Cell temperature (0x08). For high−precision RSOC measurement, it is recommended to update the cell temperature every time the temperature changes by more than 1°C. Temperature update is not required when the device is in Sleep mode.

Table 9. STATUS BIT

Register Name	Status BIT	Set Value in Status Bit	
		0	1
Cell Temperature (TSENSE)	BIT0	I^2C Mode	Thermistor Mode

- Thermistor mode: The device measures thermistors directly
- I^2C mode: The device receives temperature information via I^2C

Alarm Functions

By using the alarm functions, the application processor can quickly detect a condition exceeding a preset threshold. Table 10 shows the registers for setting alarm thresholds and the corresponding registers monitored by the alarm function. The alarm function is disabled if the threshold register contains its default value. When an alarm condition occurs, this device outputs Low to ALARMB to notify the application processor. The processor can determine the exact cause of the alarm by reading the Alarm bit in BatteryStatus (0x19). The ALARMB low output is cleared if the alarm condition is released. However, once the BatteryStatus Alarm bit is set, it will not reset itself on releasing the alarm condition. The reset must be performed by the processor.

There is a notice about the low output delay to ALARMB when an alarm condition occurs. When the battery is in neither charging nor discharging state, the output delays by up to 60 seconds in order to reduce the current consumption.

The alarm function is only valid in Operational mode. In Sleep mode, ALARMB output is canceled regardless of the alarm status.

Log Functions

Table 11 shows the list of log registers and their corresponding monitored registers. These log functions start counting from the initial value and detect maximum and minimum log values after the initialization sequence of the device. The log functions are only effective in Operational mode. All the log registers are Read/Write enabled except CycleCount (0x17).

If these registers are written with the user's value, counting and detection operation will start from the defined value. Figure 11 shows an example of cycle count measurement. When RSOC reduction reaches 100%, CycleCount is incremented by +1 count. The battery does not need to be in a full charge or empty charge state to continue the cycle count.

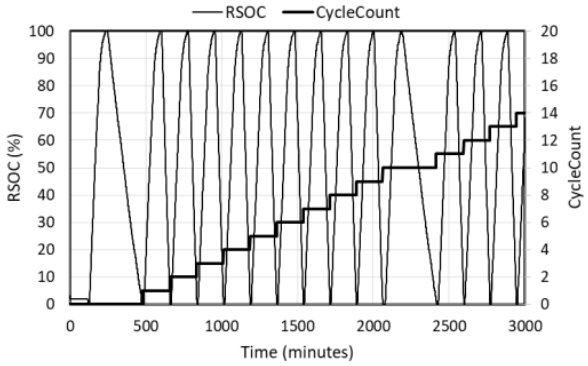


Figure 11. CycleCount (0x17) Report Example

Table 10. ALARM FUNCTIONS

Threshold Register	Monitored Register	BIT of Battery Status (0x19)	Unit
Alarm High Cell Voltage (0x1F)	Cell Voltage (0x09)	15	mV
Alarm High Temperature (0x21) (Note 3)	Cell Temperature (TSENSE) (0x08)	12	0.1 K
Alarm Low Cell Voltage (0x14)	Cell Voltage (0x09)	11	mV
Alarm Low RSOC (0x13)	RSOC (0x0D)	9	%
Alarm Low Temperature (0x20) (Note 3)	Cell Temperature (TSENSE) (0x08)	8	0.1 K

3. These alarms are enabled when TSENSE is set to Thermistor mode.

Table 11. LOG FUNCTIONS

Log Register	Monitored Register	Unit	Initial Value
CycleCount (0x17)	RSOC (0x0D)	count	0x0000
TotalRuntime (0x25,0x24)	N/A	minutes	0x0000
Accumulated Temperature (0x27,0x26)	Cell Temperature (TSENSE) (0x08)	2 K × minutes	0x0000
Accumulated RSOC (0x29,0x28)	RSOC (0x0D)	% × minutes	0x0000
Maximum Cell Voltage (0x2A)	Cell Voltage (0x09)	mV	0x0000
Minimum Cell Voltage (0x2B)	Cell Voltage (0x09)	mV	0x1388
Maximum Cell temperature (TSENSE) (0x2C) (Note 4)	Cell Temperature (TSENSE) (0x08)	0.1 K	0x0980
Minimum Cell temperature (TSENSE) (0x2D) (Note 4)	Cell Temperature (TSENSE) (0x08)	0.1 K	0x0DCC

4. These logs are updated when TSENSE is Thermistor mode.

Detection of Battery Status

This device detects whether the battery is charged or discharged and outputs that status to the Discharging Bit (Bit 6 of BatteryStatus). Table 12 shows the relationship between the Discharging bit and the Battery status. Figure 12 shows an example of Discharging Bit measurement when the battery is charging, discharging, and at no load condition. If Current Direction(0x0A) is set 0x0000(Auto mode), Battery Status shows result of Auto mode.

Table 12. DISCHARGING BIT (BIT 6 OF BATTERY STATUS REGISTER)

Discharging Bit	Battery Status
0	Charge
1	Discharge or No load current

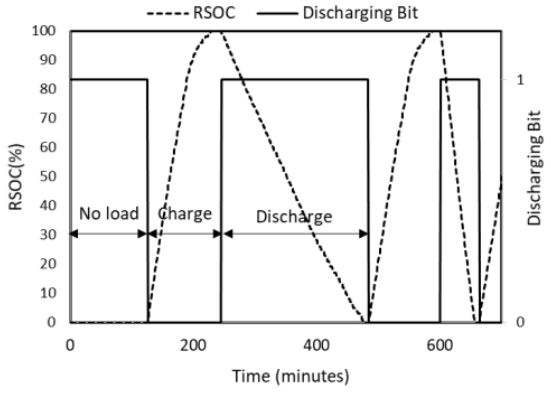


Figure 12. Discharging Bit and RSOC during Charge and Discharge Cycle

Detection of System Reset

This device will be reset and stop battery measurement under the following conditions:

- The battery is removed
- The battery voltage falls below the reset release voltage of this device due to excessive load current
- The battery protection controller disconnects the battery
- The RESETB pin detects the low level

If appropriate initial values of the registers are written into the built−in NVM, the device can start measuring the battery immediately after these conditions are removed.

Alternatively, the application processor can also start the battery measurement by executing the starting flow to set the initial values of the registers. In this case, the processor can use the INITIALIZED bit of BatteryStatus (0x19) to trigger the starting flow, as shown in Figure 13. The INITIALIZED bit is automatically set to 1 after a power−on reset. If the processor had set this bit to 0 immediately after the last power−on reset, the processor can detect the device reset operation by reading this bit.

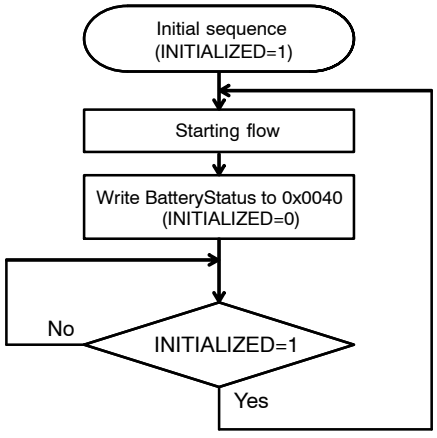


Figure 13. Flow to Restart the Gauge after Excessive Voltage Drop

How to Estimate Time to Empty

This section describes how the Fuel Gauge estimates the remaining battery time. The Time to Empty register (0x03) provides the estimated remaining time until RSOC reaches 0%. This device automatically learns an average time that is required for continuous 10% RSOC decrease during each discharge operation. Time to Empty is calculated by using the learned decreased rate before RSOC reaches 0%. See Figure 14 for details. If RSOC increases after a charge operation, the previously−learned decrease rate before charging is used to predict Time to Empty. Time to Empty set 0xFFFF to get continuous 10% RSOC decrease.

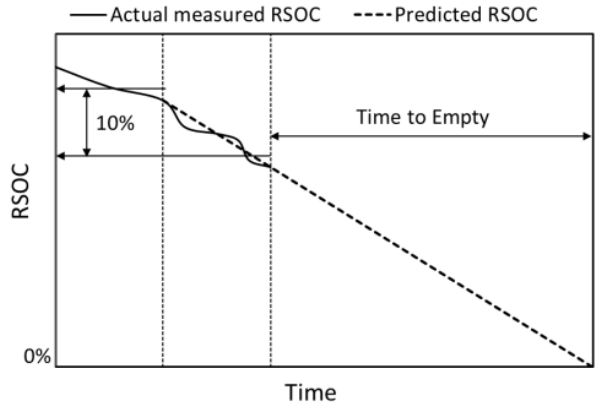


Figure 14. How to Estimate Time to Empty

Time to Empty changes dynamically if load current changes. If stable Time to Empty is needed, Host processor can calculate below formula.

Time to Empty = 60 / "Average active discharge current (C rate)" / 100 x RSOC (minutes)

"Average active discharge current" is the average current which is used to estimate time until empty. Its unit should be C rate. Example:

Average active discharge current (C rate)=0.2C

RSOC=80%

Time to Empty(minute) = 60/0.2/100 x 80=240

How to Estimate Time to Full

This section describes how the Fuel Gauge estimates the full time. The Time to Full register (0x05) provides the estimated remaining time until RSOC becomes 100%. During constant current charging, this device continues learning the RSOC increase rate. The time until Cell voltage reaches the maximum charging voltage (predefined) is calculated using the learned rate. During constant voltage charging the charging current decreases to the termination current. Therefore, this LSI estimates that the charging time for each 1% RSOC gradually gets longer. See Figure 16. The Time to Full (TTF) register outputs the total time for both modes. Refer to Figure 15.

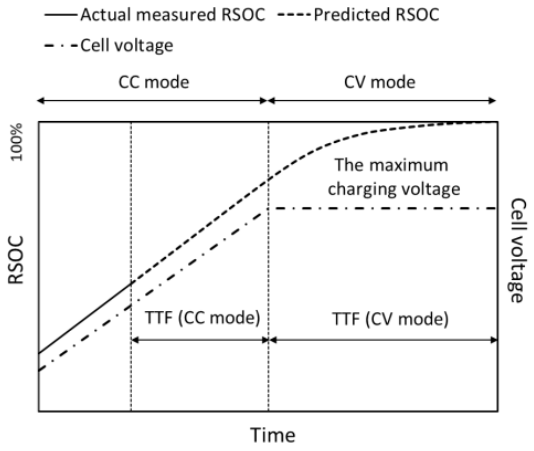


Figure 15. How to Estimate Time to Full

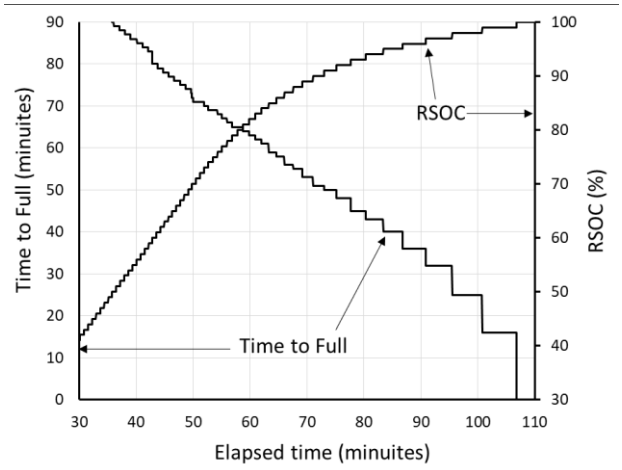
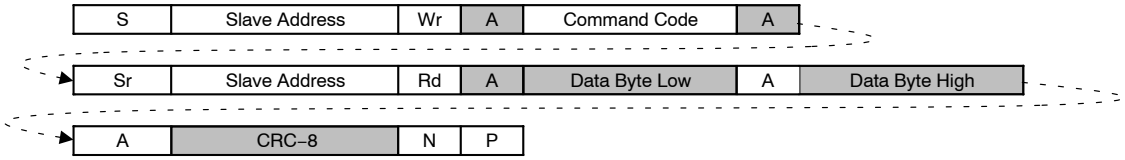


Figure 16. Time to Full (0x05) Report Example under CC−CV Charging

I²C Communication Protocol

This section describes I²C protocol and the actual waveform. Refer to the datasheet about the characteristics.



S	:	Start Condition
Sr	:	Repeated Start Condition
Rd	:	Read (bit value of 1)
Wr	:	Write (bit value of 0)
A	:	ACK (bit value of 0)
N	:	NACK (bit value of 1)
P	:	Stop Condition
CRC−8	:	Slave Address to Last Data (CRC−8−ATM : ex. 3778 mV : 0x16, 0x09, 0x17, 0xC2, 0x0E → 0x86)
	:	Master−to−Slave
	:	Slave−to−Master

Figure 17. Read Word Protocol

Read Waveform

Example: Read RSOC. RSOC = 98%.
I2C_ReadWord(0x0D);
Slave Address + Write: 0x16 (1)
Command Code: 0x0D
Slave Address + Read: 0x17 (2)
Data Byte Low: 0x62 (RSOC = 98%)
Data Byte High: 0x00
CRC−8: 0xEC (3)

Figure 18. Overview of Read Waveform

1. Slave Address + Write: 0x16
   Command Code: 0x0D

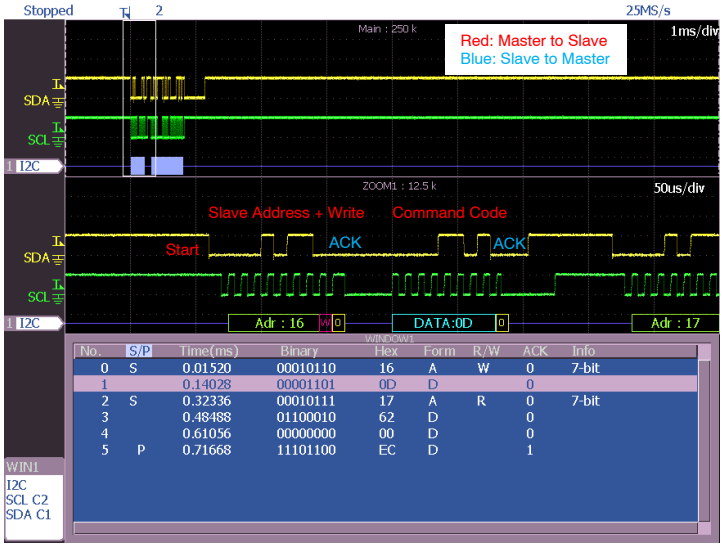
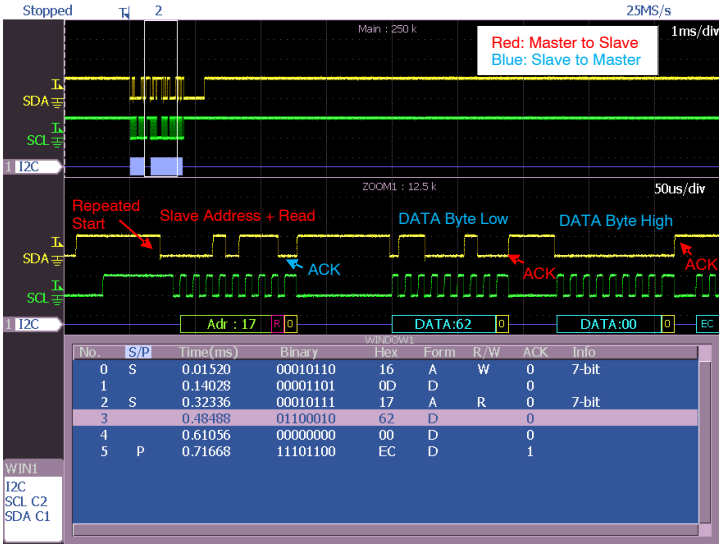


Figure 19. Read Waveform (1)

2. Slave Address + Read: 0x17
   Data Byte Low: 0x62 (RSOC = 98%)
   Data Byte High: 0x00

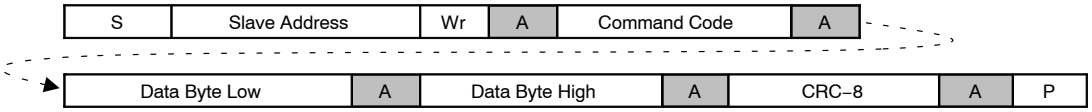


NOTE: The read data becomes 0xFFFF if Repeated Start Condition is not done.

Figure 20. Read Waveform (2)

3. CRC−8: 0xEC

Figure 21. Read Waveform (3)



S	:	Start Condition
Sr	:	Repeated Start Condition
Rd	:	Read (bit value of 1)
Wr	:	Write (bit value of 0)
A	:	ACK (bit value of 0)
N	:	NACK (bit value of 1)
P	:	Stop Condition
CRC−8	:	Slave Address to Last Data (CRC−8−ATM : ex. 3778 mV : 0x16, 0x09, 0x17, 0xC2, 0x0E → 0x86)
	:	Master−to−Slave
	:	Slave−to−Master

Figure 22. Write Word Protocol

Write Waveform

Example: Set IC Power Mode to Operational mode.
I2C_WriteWord (0x15 , 0x0001);
Slave Address + Write: 0x16 (1)
Command Code: 0x15
Data Byte Low: 0x01 (2)
Data Byte High: 0x00
CRC−8: 0x64 (3)

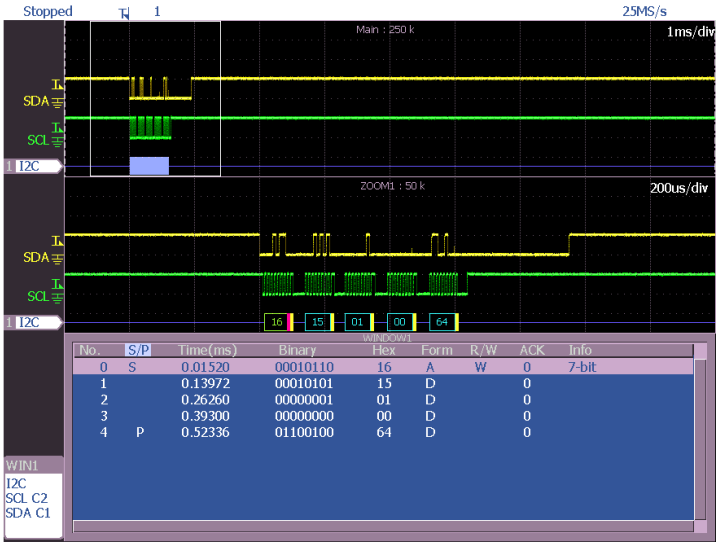


Figure 23. Overview of Write Waveform

1. Slave Address + Write: 0x16
   Command Code: 0x15

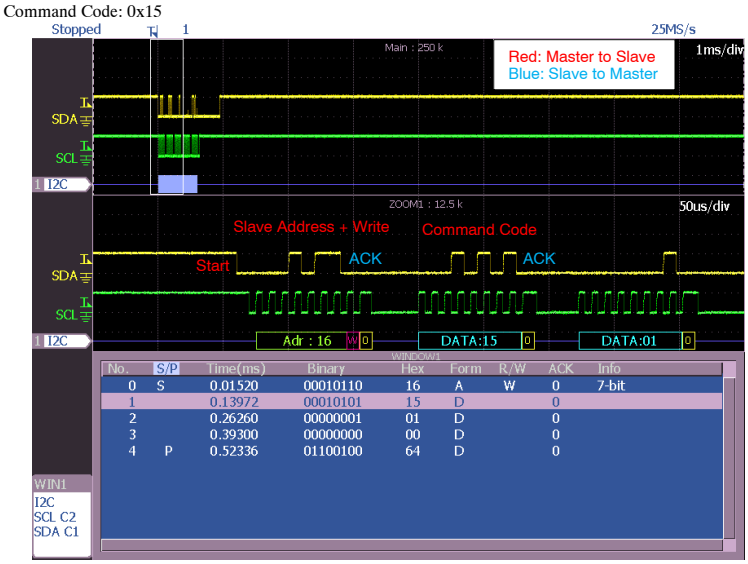


Figure 24. Write Waveform (1)

2. DATA Byte Low: 0x01
   DATA Byte High: 0x00

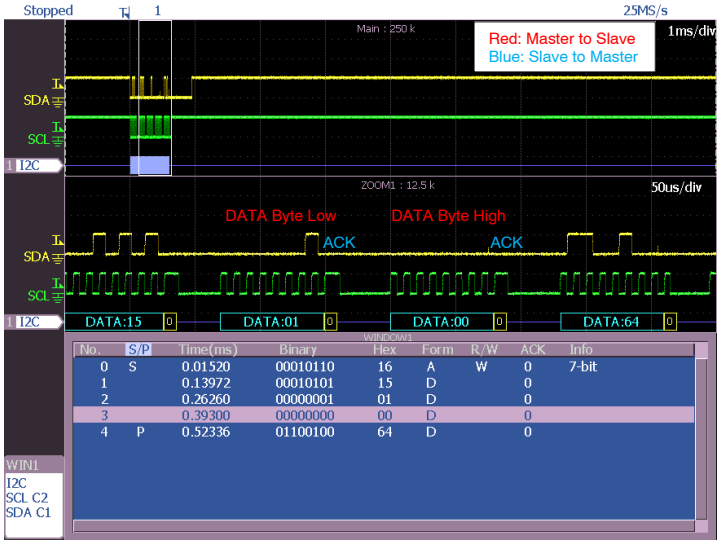


Figure 25. Write Waveform (2)

3. CRC−8: 0x64

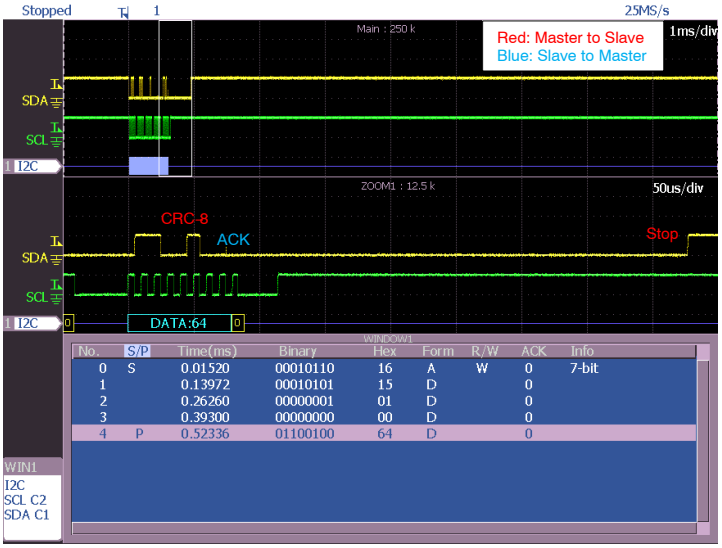


Figure 26. Write Waveform (3)

STARTING FLOW AND SAMPLE CODE

This section shows starting flow and the sample codes to startup the gauge. The sample codes set only the Mandatory registers.

Sample code

- CRC−8 calculation
- LC709209F Starting flow with Thermistor mode
- LC709209F Starting flow with I^{2}C mode

CRC−8 calculation

This code calculates CRC−8 to use in I^{2}C communication.

```
/**
 *================================================================================
 *  Calculate of CRC-8 by C-Language
 *================================================================================
 */

#define dPOLYNOMIAL8            0x8380

/*
 *================================================================================
 * Input data   : previous data of CRC-8 , calculate data
 * Output data  : CRC-8 data after calculate
 * Function : CRC-8 calculate
 *================================================================================
 */
static unsigned char u1_CRC_8_u1u1( unsigned char u1ArgBeforeData , unsigned char u1ArgAfterData)
{
    unsigned char   u1TmpLooper = 0;
    unsigned char   u1TmpOutData = 0;
    unsigned short  u2TmpValue = 0;

    u2TmpValue = (unsigned short)(u1ArgBeforeData ^ u1ArgAfterData);
    u2TmpValue <<= 8;

    for( u1TmpLooper = 0 ; u1TmpLooper < 8 ; u1TmpLooper++ ){
        if( u2TmpValue & 0x8000 ){
            u2TmpValue ^= dPOLYNOMIAL8;
        }
        u2TmpValue <<= 1;
    }

    u1TmpOutData = (unsigned char)(u2TmpValue >> 8);

    return( u1TmpOutData );
}

int main( void )
{
    static unsigned char    u1Calc = 0;
    static unsigned char    u1CRC8 = 0;

    // Write Word Protocol
    u1Calc = u1_CRC_8_u1u1( 0x00 , 0x16 );      // Address
    u1Calc = u1_CRC_8_u1u1( u1Calc , 0x07 );    // Command
    u1Calc = u1_CRC_8_u1u1( u1Calc , 0x55 );    // Data
    u1CRC8 = u1_CRC_8_u1u1( u1Calc , 0xAA );    // Data

    // Read Word Protocol
    u1Calc = u1_CRC_8_u1u1( 0x00 , 0x16 );      // Address
    u1Calc = u1_CRC_8_u1u1( u1Calc , 0x0D );    // Command
    u1Calc = u1_CRC_8_u1u1( u1Calc , 0x17 );    // Address
    u1Calc = u1_CRC_8_u1u1( u1Calc , 0x20 );    // Data
    u1CRC8 = u1_CRC_8_u1u1( u1Calc , 0x00 );    // Data

    return( 0 );            //
}
```

Starting Flow

This device starts the initial sequence automatically after release of the power−on reset. I^2C communication is enabled after the sequence. Then set registers to start gauging according to the following sample codes.

Write and Read Register (Common)

```
void i2c_WriteWord( unsigned char u1ArgCommand , unsigned short u2ArgData )
{
    // H/W of I2C for Application Processor
}

unsigned short i2c_ReadWord( unsigned char u1ArgCommand )
{
    // H/W of I2C for Application Processor
}
```

LC709209F Starting Flow with Thermistor Mode

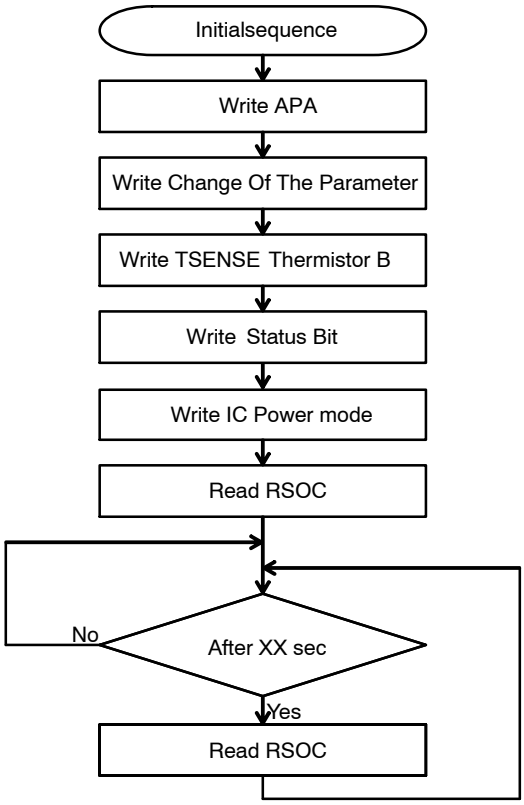


Figure 27. LC709209F Starting Flow with Thermistor Mode

```
/**
 *============================================================================
 *  Sample of Application Processor(LC709209F / temperature read via Thermistor)
 *============================================================================
 */

void i2c_WriteWord( unsigned char u1ArgCommand , unsigned short u2ArgData )
{
    // Implementation of I2C for Application Processor
}

unsigned short i2c_ReadWord( unsigned char u1ArgCommand )
{
    // Implementation of I2C for Application Processor
}

int main( void )
{
    unsigned short      u2RSOC;

    /*
        Battery connection
        LC709209F Power ON
        AP(Application Processor) Power On
    */

    // Initialization process from Application Processor
    i2c_WriteWord( 0x0B , 0x3534 );         // Slave Function : APA(Adjustment Pack Application)
                                            // Command : 0x0B
                                            // Data : 0x3534 (ex. APA = 0x3534)

    i2c_WriteWord( 0x12 , 0x0000 );         // Slave Function : Change Of The Parameter
                                            // Command : 0x12
                                            // Data : 0x0000 (ex. Battery profile = 0x0000)

    i2c_WriteWord( 0x06 , 0x0D34 );         // Slave Function : TSENSE Thermistor B
                                            // Command : 0x06
                                            // Data : 0x0D34 (ex. B = 3380)

    i2c_WriteWord( 0x16 , 0x0001 );         // Slave Function : Status Bit
                                            // Command : 0x16
                                            // Data : 0x0001 (Thermistor Mode)

    i2c_WriteWord( 0x15 , 0x0001 );         // Slave Function : IC Power Mode
                                            // Command : 0x15
                                            // Data : 0x0001 (Operational Mode)

    u2RSOC = i2c_ReadWord( 0x0D );          // Slave Function : RSOC
                                            // Command : 0x0D

    // Control from Application Processor
    while( 1 ){

        wait_XXs();                         // wait XX s
                                            // EX 10s

        if( SmartPhone_PowerOn ){
            // SmartPhone Power ON
            u2RSOC = i2c_ReadWord( 0x0D );  // Slave Function : RSOC
                                            // Command : 0x0D
        }else{
            // SmartPhone Power OFF
            while( SmartPhone_PowerOff ){
                // AP Low Power Mode
            }
        }
    }

}
```

LC709209F Starting Flow with I²C Mode

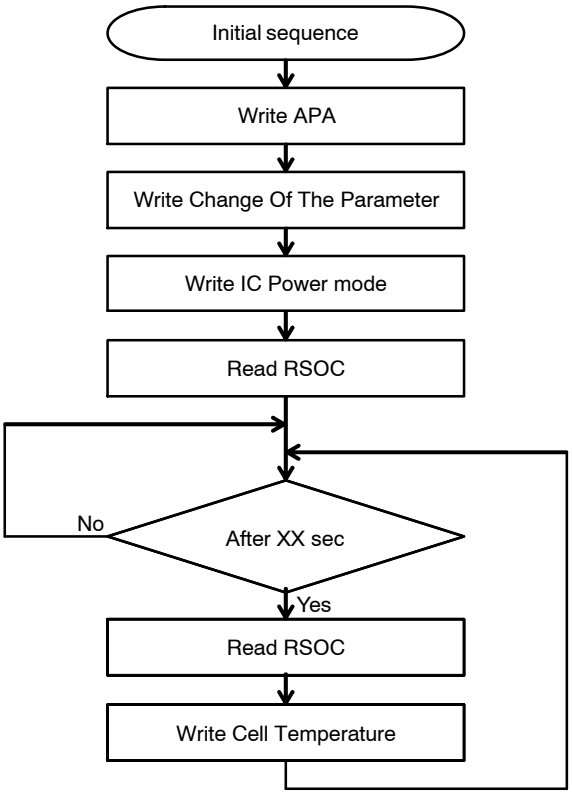


Figure 28. LC709209F Starting Flow with I²C Mode

```
/**
 *============================================================================
 *  Sample of Application Processor(LC709209F / temperature input to LSI Via I2C)
 *============================================================================
 */

void i2c_WriteWord( unsigned char u1ArgCommand , unsigned short u2ArgData )
{
    // Implementation of I2C for Application Processor
}
unsigned short i2c_ReadWord( unsigned char u1ArgCommand )
{
    // Implementation of I2C for Application Processor
}

int main( void )
{
    unsigned short      u2RSOC;

    /*
        Battery connection
        LC709209F Power ON
        AP(Application Processor) Power On
    */

    // Initialization process from Application Processor
    i2c_WriteWord( 0x0B , 0x3534 );         // Slave Function : APA(Adjustment Pack Application)
                                            // Command : 0x0B
                                            // Data : 0x3534 (ex. APA = 0x3534)

    i2c_WriteWord( 0x12 , 0x0000 );         // Slave Function : Change Of The Parameter
                                            // Command : 0x12
                                            // Data : 0x0000 (ex. Battery profile = 0x0000)

    i2c_WriteWord( 0x15 , 0x0001 );         // Slave Function : IC Power Mode
                                            // Command : 0x15
                                            // Data : 0x0001 (Operational Mode)

    u2RSOC = i2c_ReadWord( 0x0D );          // Slave Function : RSOC
                                            // Command : 0x0D

    // Control from Application Processor
    while( 1 ){

        wait_XXs();                         // wait XX s
                                            // EX 10s

        if( SmartPhone_PowerOn ){
            // SmartPhone Power ON
            u2RSOC = i2c_ReadWord( 0x0D );  // Slave Function : RSOC
                                            // Command : 0x0D

            i2c_WriteWord( 0x08 , 0x0BA6 ); // Slave Function : Cell Temperature
                                            // Command : 0x08
                                            // Data : 0x0BA6 (ex. 25C to 25*10 + 2732 to 0x0BA6)
        }else{
            // SmartPhone Power OFF
            while( SmartPhone_PowerOff ){
                // AP Low Power Mode
            }
        }
    }

}
```

Built−in NVM Writing Protocol

The following sections describe how to write user ID and initial setting data into the registers of the devices's built−in NVM. I2C commands of the device can control all of the writes. Therefore, a master device connected to the device by I^2C as shown in Figure 29 can control the write.

Two sample codes at the end of this note will help you reduce the time to implement your program and ensure your understanding.

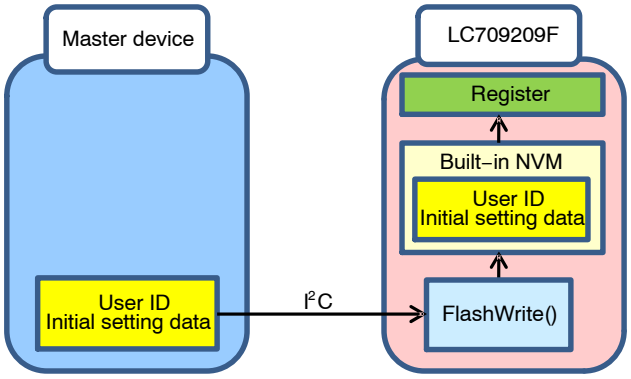


Figure 29. Block Diagram about User ID Writing

User ID

User ID (0x36, 0x37) provides 32−bit programmable registers stored in the built−in NVM. You can use that value for any purpose, for example individual identification of battery pack.

Initial Setting Data of Registers

The initial values of registers shown in Table 13 can be written into the built−in NVM. These provide basic information for battery gauging. If these has been programmed once, they are loaded automatically during every initial sequence after reset or power on. In that case, the device can start gauging without control of a master device.

Table 13. REGISTERS WHOSE INITIAL VALUE IS STORED IN THE BUILT−IN NVM

Command Code	Register Name	ISD No.
0x0B	APA	#1
0x0C	APT	#2
0x06	TSENSE Thermistor B	#3
0x16	Status Bit	#4
0x1C	Termination Current Rate	#5
0x1D	Empty Cell Voltage	#6
0x1E	ITE Offset	#7
0x13	Alarm Low RSOC	#8
0x14	Alarm Low Cell Voltage	#9
0x1F	Alarm High Cell Voltage	#10
0x20	Alarm Low Temperature	#11
0x21	Alarm High Temperature	#12
0x15	IC Power Mode	#13
0x12	Change of the Parameter	#14

Conditions for ID Writing

The following operating conditions must be satisfied during programming of the buit−in NVM.

Allowable operating conditions during NWM writing

- Supply voltage: 3.0 V to 5.0 V
- Ambient temperature: 10°C to 55°C

The re−writing cycle is confined to 100 cycles, and should be controlled by a master device in order to prevent multiple programming. Figure 30 shows how a master device confirms User ID or initial value of a target register before the writing. If the read data is not same as the target data, start writing.

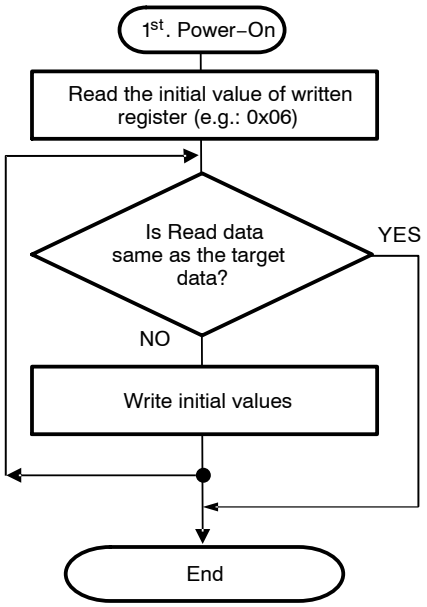


Figure 30. Flow to Prevent Multiple ID Writing

Outline of User ID Writing Flow

Figure 31 shows the flows for writing user ID and initial setting data into the built−in NVM. These flows consist of mode setting, programming, verification and mode release. In each step of this flow a master device transmits commands shown in Table 14. This table indicates that the write commands require some kind of data size. The write n−byte data protocol is shown in Figure 32. See Figure 17 for the read data protocol.

The commands in the data transfer and start verify processes are different for user ID and initial setting data. The details of each process are explained in the following sections.

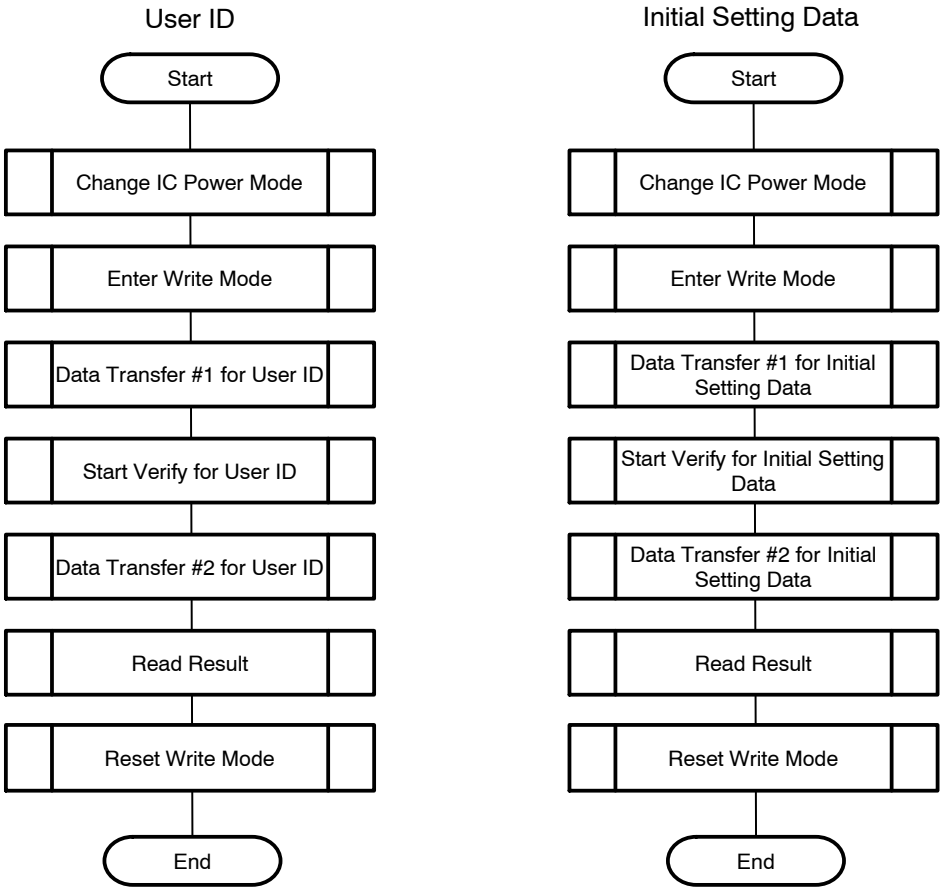


Figure 31. Outline of Writing Flow

Table 14. COMMAND LIST FOR USER ID WRITING PROTOCOL

2 bytes of all contents are little endian.

Ex1: 0x55AA → data [0] = 0xAA, data [1] = 0x55

Ex2: instruction [2] = 0x8180 → instruction [0] = 0x80, instruction [1] = 0x81

Command Code	Function	R/W	Data Size	Contents
0x00	Enable Write Mode	W	2	0x55AA
0x01	Enter Write Mode	W	2	0x55AA
0x02	Set Data	W	130	User ID: Data[0] to Data[1] 0x8180 Data[2] to Data[5] User ID data Data[6] to Data[129] 0x00 Initial setting data: Data[0] to Data[1] 0xA000 Data[2] to Data[29] Initial setting data Data[30] to Data[129] 0xAA56
0x03	Set Key1	W	2	0x55AA
0x04	Set Key2	W	2	0x00A0
0x05	Write Exe	W	2	0x55AA
0x06	Start Verify	W	4	User ID: 0x81808180 Initial setting data: 0xA000A000
0x07	Verify Result	R	2	Result of verification Success: 0x0001 Failure: 0x000
0x08	Reset Write Mode	W	2	0x55AA

NOTE: Commands 0x03 to 0x08 are enabled in Write mode.

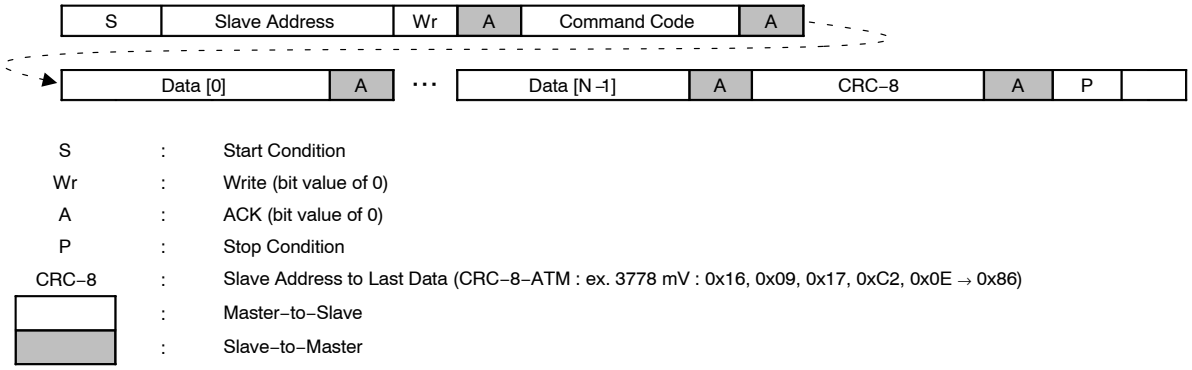


Figure 32. Write N−bytes Data Protocol

Change Power Mode

This process sets the device into a special mode to program the built−in NVM.

Table 15. IC POWER MODE COMMAND

Command Name	Slave Address (W)	Command Code	Data[0]	Data[1]	CRC−8
IC Power Mode	0x16	0x15	0x00	0x00	0x71

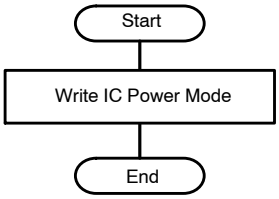


Figure 33. Change Power Mode

Enter Write Mode

This process sets the device into a special mode to program the built−in NVM. After these commands have been input, wait for 300 ms to transition into the mode.

Table 16. WRITE MODE COMMAND

Command Name	Slave Address (W)	Command Code	Data[0]	Data[1]	CRC−8
Enable Write Mode	0x16	0x00	0xAA	0x55	0x25
Enter Write Mode	0x16	0x01	0xAA	0x55	0x4E

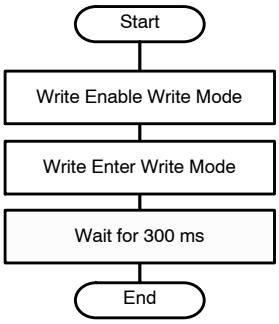


Figure 34. Enter Write Mode

Data Transfer #1 for User ID

This process programs User ID data in the built−in NVM. The Set Data command sends the data into the device. Then the Set Key1, Set Key2 and Write Exe commands execute programming into the built−in NVM. It takes up to 40 ms for programming to be completed. During this period the device is busy and cannot respond to any I^2C commands. It acknowledges the period to a master device using clock stretching.

Table 17. DATA TRANSFER COMMAND FOR USER ID (130 BYTES DATA)

Command Name	Slave Address(W)	Command Code	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data [6~129]	CRC−8
Set data	0x16	0x02	0x80	0x81	UID[0]	UID[1]	UID[2]	UID[3]	0x00	0xXX

NOTE: Set User ID data modified according to following formula in Data[5:2]. The Data is calculated as Two's complement, if it is a negative number. CRC−8 is calculated with the string composed of the data from slave address to Data[129].
UID[1:0] = Lower 16 bits of User ID − 0x55AA UID[3:2] = Upper 16 bits of User ID − 0x55AA
e.g.: In the case of 32 bits of User ID 0x12345678
UID[1:0] = 0x5678 − 0x55AA = 0x00CE … UID[0] = 0xCE, UID[1] = 0x00
UID[3:2] = 0x1234 − 0x55AA = 0xBC8A … UID[2] = 0x8A, UID[3] = 0xBC

Table 18. DATA TRANSFER COMMAND (2 BYTES DATA)

Command Name	Slave Address (W)	Command Code	Data[0]	Data[1]	CRC−8
Set Key1	0x16	0x03	0xAA	0x55	0x98
Set Key2	0x16	0x04	0xA0	0x00	0xA0
Write Exe	0x16	0x05	0xAA	0x55	0xE5

Data Transfer #1 for Initial Setting Data

This process programs initial setting data into the built−in NVM. As shown in Table 19, the data transferred by the Set Data command is different from user ID. The initial values of the registers numbered with the ISD# in Table 13 are transferred to the device using Data[2] to Data[29] of the Set Data command.

Data[30] to Data[129] of the command are filled with repeated and fixed data: 0x56 and 0xAA. Set Key1, Set Key 2 and Write Exe commands and programming time are the same as user ID.

Table 19. DATA TRANSFER COMMAND FOR INITIAL SETTING DATA (130 BYTES DATA)

Command Name	Slave Address(W)	Command Code	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6] to Data[27]
Set data	0x16	0x02	0x00	0xA0	ISD#1[0]	ISD#1[1]	ISD#2[0]	ISD#2[1]	ISD#3 to ISD#13
			Data[28]	**Data[29]**	**Data[30]**	**Data[31]**	**Data[32] to Data[129]**		**CRC−8**
			ISD#14[0]	ISD#14[1]	0x56	0xAA	0x56, 0xAA		0xXX

NOTE: Set the initial values modified according to following formula in Data [29:2]. The data is calculated as two's complement, if it is a negative number. CRC−8 is calculated with the string composed of the data from slave address to Data[129].
ISD#X [1:0] = "16−bit Initial value" − 0x55AA
E.g.: when the initial value of APA (0x0B) register is 0x5678,
ISD#1 [1:0] = 0x5678 − 0x55AA = 0x00CE ... ISD#1[0] = 0xCE, ISD#1[1] = 0x00

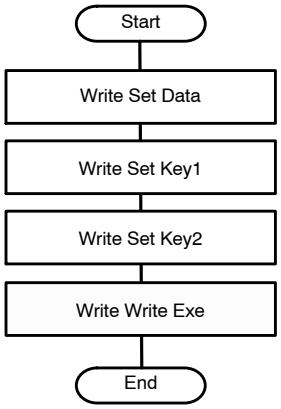


Figure 35. Data Transfer #1 and #2

Start Verify for User ID

This process sets the device into a special mode to verify the user ID data written in the built−in NVM data.

Table 20. START VERIFY COMMAND FOR USER ID

Command Name	Slave Address (W)	Command Code	Data[0]	Data[1]	Data[2]	Data[3]	CRC−8
Start Verify	0x16	0x06	0x80	0x81	0x80	0x81	0x4A

Start Verify for Initial Setting Data

This process sets the device into a special mode to verify the initial setting data written in the built−in NVM data.

Table 21. START VERIFY COMMAND FOR INITIAL SETTING DATA

Command Name	Slave Address (W)	Command Code	Data[0]	Data[1]	Data[2]	Data[3]	CRC−8
Start Verify	0x16	0x06	0x00	0xA0	0x00	0xA0	0x02

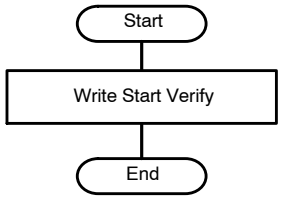


Figure 36. Start Verify

Data Transfer #2

This process verifies the programmed built−in NVM data with retransmit data. The commands are the same as Data transfer #1. But programming into built−in NVM is not executed because the device is in a special mode to verify. Therefore a master device does not have to wait for 40 ms.

Read Result

This process reads the result of Data transfer #2 verification. If the verification was successful, the read data is 0x0001. On the other hand, if the verification failed, the read data is 0x0000, and a master device can retry the write operation by following the steps in the Retry by Error section.

Table 22. READ VERIFY RESULT COMMAND

Command Name	Slave Address (W)	Command Code	IC Address (R)	Data[0]	Data[1]	CRC−8
Verify Result	0x16	0x07	0x17	Lower 8−bit of result	Upper 8−bit of result	0xXX

NOTE: CRC−8 is calculated with the string composed of the data from the first slave address to Data [1].

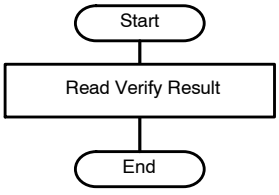


Figure 37. Read Result

Reset Write Mode

This process releases the device from all special modes and initializes all registers. Wait for 1.5 s to allow the initialization process to complete before you continue to access the device. Or you can turn off the power to the device without waiting.

Table 23. RESET WRITE MODE COMMAND

Command Name	Slave Address (W)	Command Code	Data[0]	Data[1]	CRC−8
Reset Write Mode	0x16	0x08	0xAA	0x55	0x74

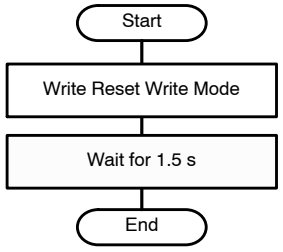


Figure 38. Reset Write Mode

Retry by Error

When an I^2C communication error or verification error occurs, the first step of the re−write process is limited as shown in Figure 39. If the error is detected during the Change IC Power Mode process, a master can restart the process to change IC Power Mode. But if the error was detected at any stage between Enter Write Mode and Reset Write Mode, a master must restart the sequence from the Enter Write Mode step.

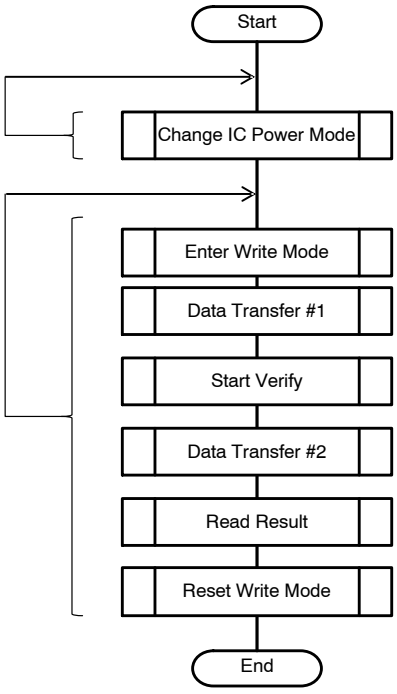


Figure 39. Retry by Error

This sample code writes user ID data into the built−in NVM. It includes the flow to prevent multiple ID writing.

```
/**
 *============================================================================
 *  Sample of Application Processor(User ID writing)
 *============================================================================
 */

#define USERID_L        (0x5678)            // Definition of lower 16bits of User ID
#define USERID_H        (0x1234)            // Definition of upper 16bits of User ID

void i2c_WriteWord( unsigned char u1ArgCommand , unsigned short u2ArgData )
{
    // Implementation of I2C for Application Processor
}

unsigned short i2c_ReadWord( unsigned char u1ArgCommand )
{
    // Implementation of I2C for Application Processor
}

void i2c_WriteData( unsigned char u1ArgCommand , unsigned char *u1ArgData, unsigned short u2ArgSz )
{
    // Implementation of I2C for Application Processor
}

void i2c_DataTransfer( void )
{
    unsigned char       u1Data[130];
    unsigned short      u2UID_L;
    unsigned short      u2UID_H;
    unsigned short      n;

    u2UID_L = USERID_L - 0x55AA;
    u2UID_H = USERID_H - 0x55AA;

    u1Data[0] = 0x80;
    u1Data[1] = 0x81;
    u1Data[2] = (u2UID_L & 0x00FF);
    u1Data[3] = (u2UID_L & 0xFF00) >> 8;
    u1Data[4] = (u2UID_H & 0x00FF);
    u1Data[5] = (u2UID_H & 0xFF00) >> 8;
    for (n=6; n<130; n++){
        u1Data[n] = 0;
    }
    i2c_WriteData( 0x02 , u1Data , 130 );   // Slave Function : Set data
                                            // Command : 0x02
                                            // Data[0] : 0x80 , Data[1] : 0x81 ,
                                            // Data[2] : UID0 , Data[3] : UID1 ,
                                            // Data[4] : UID2 , Data[5] : UID3 ,
                                            // Data[6] ... Data[129] : 0x00

    i2c_WriteWord( 0x03 , 0x55AA );         // Slave Function : Set key1
                                            // Command : 0x03
                                            // Data : 0x55AA

    i2c_WriteWord( 0x04 , 0x00A0 );         // Slave Function : Set key2
                                            // Command : 0x04
                                            // Data : 0x00A0

    i2c_WriteWord( 0x05 , 0x55AA );         // Slave Function : Write/Verify exe
                                            // Command : 0x05
                                            // Data : 0x55AA
}

int main( void )
{
    unsigned short      u2Result;
    unsigned char       u1Data[4];
    unsigned short      u2UserID_L;
    unsigned short      u2UserID_H;

    /*
        Battery connection
        LC709209F Power ON
        AP(Application Processor) Power On
    */
    u2UserID_L = i2c_ReadWord( 0x36 );      // Slave Function : User ID Lower 16bits
                                            // Command : 0x36

    u2UserID_H = i2c_ReadWord( 0x37 );      // Slave Function : User ID Upper 16bits
                                            // Command : 0x37
```

This sample code writes user ID data into the built−in NVM. It includes the flow to prevent multiple ID writing. (continued)

```
    if( (u2UserID_L != USERID_L) || (u2UserID_H != USERID_H) ) {

        // User ID writing is done only once after the first power on.

        // User ID Writing process from Application Processor
        i2c_WriteWord( 0x15 , 0x0000 );          // Slave Function : Change power mode
                                                 // Command : 0x15
                                                 // Data : 0x0000 (Test Mode)

        while( 1 ){

            i2c_WriteWord( 0x00 , 0x55AA );          // Slave Function : Enable write mode
                                                     // Command : 0x00
                                                     // Data : 0x55AA

            i2c_WriteWord( 0x01 , 0x55AA );          // Slave Function : Enter write mode
                                                     // Command : 0x01
                                                     // Data : 0x55AA

            wait_300ms();                            // wait 300 msec

            i2c_DataTransfer();                      // Data Transfer#1 for User ID

            u1Data[0] = 0x80;
            u1Data[1] = 0x81;
            u1Data[2] = 0x80;
            u1Data[3] = 0x81;
            i2c_WriteData( 0x06 , u1Data , 4 );      // Slave Function : Start verify
                                                     // Command : 0x06
                                                     // Data[0] : 0x80 , Data[1] : 0x81 ,
                                                     // Data[2] : 0x80 , Data[3] : 0x81

            i2c_DataTransfer();                      // Data Transfer#2 for User ID

            u2Result = i2c_ReadWord( 0x07 );         // Slave Function : Read result
                                                     // Command : 0x07

            if( u2Result == 0x0001 ){
                // User ID writing success
                i2c_WriteWord( 0x08 , 0x55AA );       // Slave Function : Reset write mode
                                                      // Command : 0x08
                                                      // Data : 0x55AA
                break;
            }else{
                // User ID writing failure
            }
        }
    }
}
```

This sample code writes initial setting data into the built−in NVM. It includes the flow to prevent multiple ID writing.

```
/**
 *============================================================================
 *      Sample of Application Processor(Initial Setting Data writing)
 *============================================================================
 */

#define INIT_CONST_B    (0x0FA0)                // Definition of Initial value of TSENSE Thermistor B(4000)

void i2c_WriteWord( unsigned char u1ArgCommand , unsigned short u2ArgData )
{
        // Implementation of I2C for Application Processor
}

unsigned short i2c_ReadWord( unsigned char u1ArgCommand )
{
        // Implementation of I2C for Application Processor
}

void i2c_WriteData( unsigned char u1ArgCommand , unsigned char *u1ArgData, unsigned short u2ArgSz )
{
        // Implementation of I2C for Application Processor
}

void i2c_DataTransfer( void )
{
        const unsigned char InitialSettingData[128] =
        {
                // Set the lower 16 bits of the result of subtracting 0x55aa from the value.
                0x56, 0xaa, // APA (Hex) = 0x0000
                0x74, 0xaa, // APT = 30
                0xf6, 0xb9, // TSENSE Thermistor B = 4000
                0x57, 0xaa, // Status bit (Hex) = 0x0001
                0x58, 0xaa, // Termination current rate = 2
                0x56, 0xaa, // Empty cell voltage  0
                0x56, 0xaa, // ITE offset = 0
                0x56, 0xaa, // Alarm Low RSOC = 0
                0x56, 0xaa, // Alarm Low Cell Voltage = 0
                0x56, 0xaa, // Alarm High Cell Voltage = 0
                0x56, 0xaa, // Alarm Low Temperature = 0
                0x56, 0xaa, // Alarm High Temperature = 0
                0x57, 0xaa, // IC power mode (Hex) = 0x0001
                0x56, 0xaa, // Change Of The Parameter = 0
                // The following data are fixed values.
                0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
                0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa, 0x56, 0xaa,
        };

        unsigned char           u1Data[130];
        unsigned short          n;

        u1Data[0] = 0x00;
        u1Data[1] = 0xA0;
        for (n=0; n<128; n++){
                u1Data[n+2] = InitialSettingData[n];
        }
        i2c_WriteData( 0x02 , u1Data , 130 );   // Slave Function : Set data
                                                // Command : 0x02
                                                // Data[0] : 0x00 , Data[1] : 0xA0 ,
                                                // Data[2] ... Data[129] : Initial Setting Data
```

```
        i2c_WriteWord( 0x03 , 0x55AA );         // Slave Function : Set key1
                                                // Command : 0x03
                                                        // Data : 0x55AA

        i2c_WriteWord( 0x04 , 0x00A0 );                 // Slave Function : Set key2
                                                        // Command : 0x04
                                                        // Data : 0x00A0

        i2c_WriteWord( 0x05 , 0x55AA );                 // Slave Function : Write/Verify exe
                                                        // Command : 0x05
                                                        // Data : 0x55AA
}

int main( void )
{
        unsigned short          u2Result;
        unsigned char           u1Data[4];
        unsigned short          u2ConstB;
        /*
                Battery connection
                LC709209F Power ON
                AP(Application Processor) Power On
        */
        // Read Initial Value change register ex. TSENSE Thermister B
        u2ConstB = i2c_ReadWord( 0x06 );        // Slave Function : TSENSE Themister B
                                                // Command : 0x06

        if( u2ConstB != INIT_CONST_B ) {

                // Initial Setting Data writing is done only once after the first power on.

                // Initial Setting Data Writing process from Application Processor
                i2c_WriteWord( 0x15 , 0x0000 );                 // Slave Function : Change power mode
                                                                // Command : 0x15
                                                                // Data : 0x0000 (Test Mode)

                while( 1 ){
                        i2c_WriteWord( 0x00 , 0x55AA );         // Slave Function : Enable write mode
                                                                // Command : 0x00
                                                                // Data : 0x55AA

                        i2c_WriteWord( 0x01 , 0x55AA );         // Slave Function : Enter write mode
                                                                // Command : 0x01
                                                                // Data : 0x55AA

                        wait_300ms();                           // wait 300 msec

                        i2c_DataTransfer();                     // Data Transfer#1 for Initial Setting Data

                        u1Data[0] = 0x00;
                        u1Data[1] = 0xA0;
                        u1Data[2] = 0x00;
                        u1Data[3] = 0xA0;
                        i2c_WriteData( 0x06 , u1Data , 4 );     // Slave Function : Start verify
                                                                // Command : 0x06
                                                                // Data[0] : 0x00 , Data[1] : 0xA0 ,
                                                                // Data[2] : 0x00 , Data[3] : 0xA0

                        i2c_DataTransfer();                     // Data Transfer#2 for Initial Setting Data

                        u2Result = i2c_ReadWord( 0x07 );        // Slave Function : Read result
                                                                // Command : 0x07

                        if( u2Result == 0x0001 ){
                                // Initial Setting Data writing success
                                i2c_WriteWord( 0x08 , 0x55AA );  // Slave Function : Reset write mode
                                                                // Command : 0x08
                                                                // Data : 0x55AA
                                break;
                        }else{
                                // Initial Setting Data writing failure
                        }
                }
        }
}
```

onsemi is licensed by the Philips Corporation to carry the I^2C bus protocol.
Strata is trademark of Semiconductor Components Industries, LLC (SCILLC) or its subsidiaries in the United States and/or other countries.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Email Requests to: orderlit@onsemi.com

onsemi Website: www.onsemi.com

TECHNICAL SUPPORT
North American Technical Support:
Voice Mail: 1 800−282−9855 Toll Free USA/Canada
Phone: 011 421 33 790 2910

Europe, Middle East and Africa Technical Support:
Phone: 00421 33 790 2910
For additional information, please contact your local Sales Representative

◊