AND9966/D

Battery Fuel Gauging LSI [Smart LiB Gauge] for 1-Cell Lithium-ion/Polymer with LC709204F/LC709205F

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE

LC709204F and LC709205F are Fuel Gauges for 1−Cell Lithium−ion/Polymer batteries. They are parts 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 help the users to select an appropriate Fuel Gauge that fits best for their applications. It will further explains on 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 LSI communicates with the host side application processors.

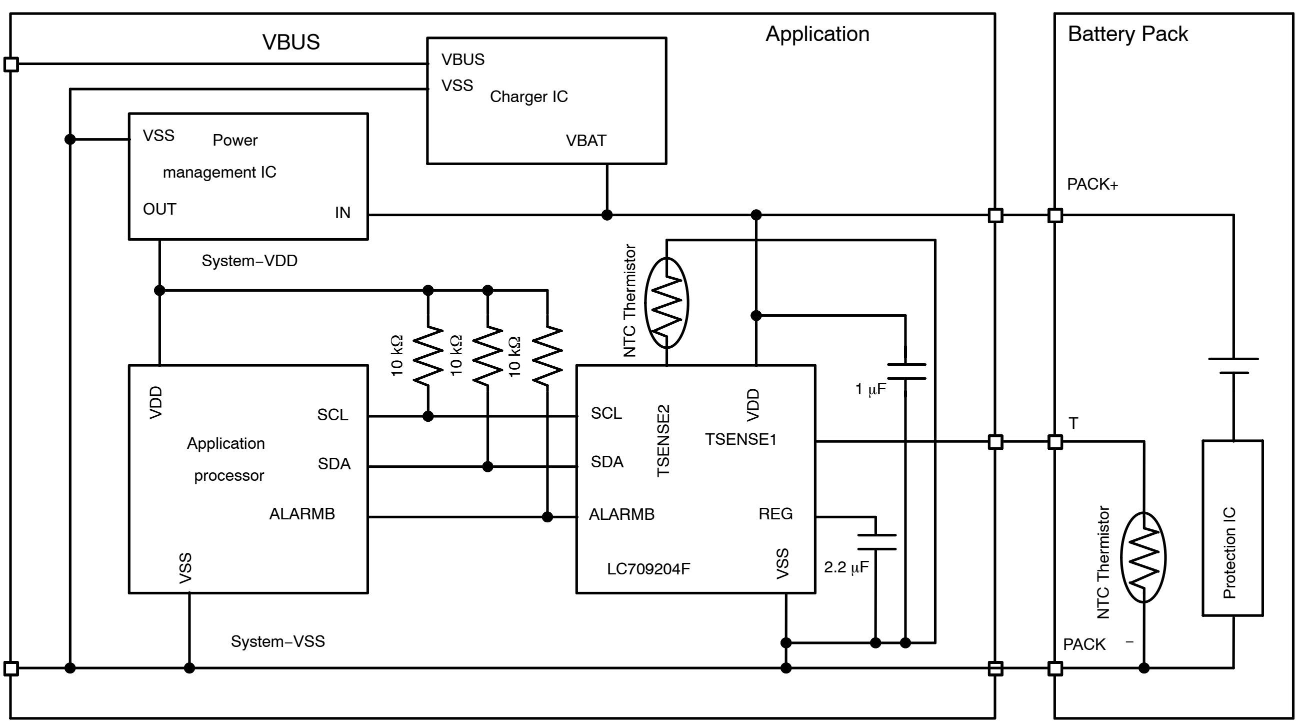


Figure 1. An Example of an Application Schematic using LC709204F

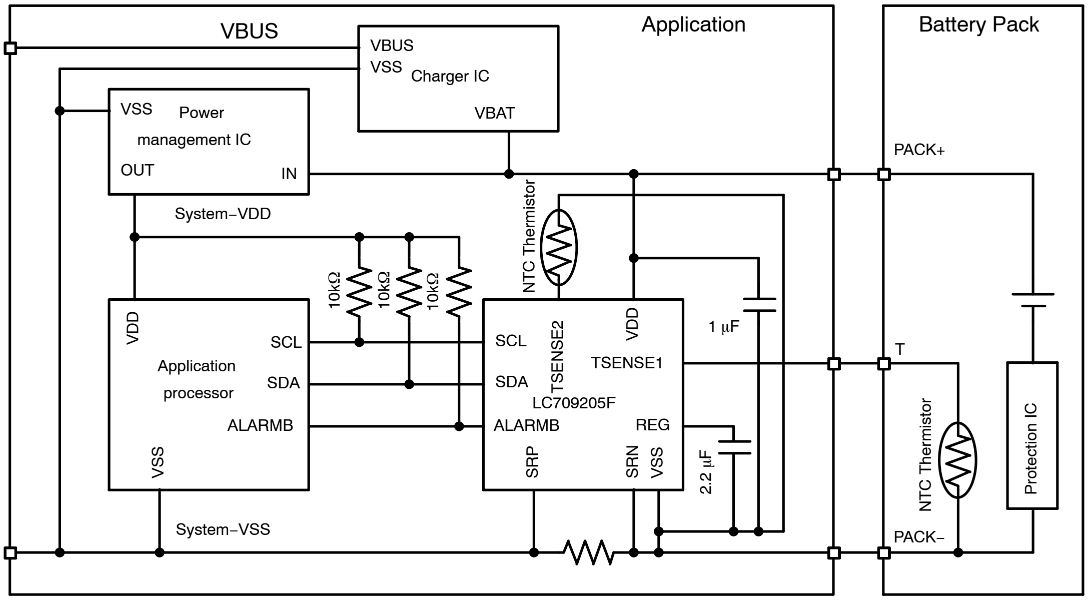


Figure 2. An Example of an Application Schematic using LC709205F

LC709205F Vs LC709204F

Figure 1 and 2 shows the application circuit diagram for LC709204F and LC709205F respectively. LC709205F has an additional function that can measure the battery current and capacity using an external sense resistor. Their main difference is the presence or absence of an external sense resistor on application. Both of them uses the HG−CVR2 algorithm to measure the RSOC and have same accuracy for RSOC measurement. There are two thermistor sense inputs TSENSE1 and TSENSE2 for both the device. Table 1 shows the registers for the extended functionality added by LC709205F over LC709204F. LC709205F is recommended if those functions are required on user's application.

Evaluation board

Evaluation board and GUI tools are available for evaluation of both the Fuel Gauges and Battery. By using these tools, Fuel Gauge and Batteries can be evaluated before creating an application board. It is also possible to connect these boards to an existing application board to evaluate them. Please refer to the documents given in Table 2 for further details about these evaluation boards.

Table 1. LC709205F ADD−ON FUNCTIONS

Command code	Register Name
0x10	FullChargeCapacity
0x18	DesignCapacity
0x22	Alarm Over Charging Current
0x23	Alarm Over Discharging Current
0x2E	Maximum Cell Current
0x2F	Minimum Cell Current
0x31	Sense resistance
0x33	Dynamic Cell current
0x34	Average Cell current
0x35	RemainingCapacity

Table 2. EVALUATION BOARDS AND DOCUMENTS FOR THE TARGET DEVICE AND BATTERY

Evaluation Board	Target Device	Battery Type	Related Documents
LC709204FXE−01−GEVB	LC709204FXE−01TBG	01, 04, 05, 06, 07	LC709204FXE−01−GEVB_Test Procedure.pdf LC709204FXE−01−GEVB_SCHEMATIC.pdf LC709204FXE−01−GEVB_GERBER.zip LC709204FXE−01−GEVB_BOM.pdf
LC709205FXE−01−GEVB	LC709205FXE−01TBG	01, 04, 05, 06, 07	LC709205FXE−01−GEVB_Test Procedure.pdf LC709205FXE−01−GEVB_SCHEMATIC.pdf LC709205FXE−01−GEVB_GERBER.zip LC709205FXE−01−GEVB_BOM.pdf

NOTE: https://www.onsemi.jp/products/power−management/battery−management/battery−fuel−gauges.

Parameter Initialization

In order to start the RSOC measurement with this LSI, it is necessary to initialize some basic parameters in advance. Table 3 shows the parameters and the corresponding registers name with their command code to set them individually. The parameters specified as Mandatory in Table 3 are the basic parameters required to measure the RSOC. Optional parameters can be initialized if the user's application requires given functionality. The detail method on how to set the required parameters is given below.

Battery profile (0x12)

Both the LSI are installed with five types of Battery profiles. Users must select an appropriate profile for their applications based on type of battery used. Please check the battery nominal voltage and charging voltage as shown in Table 4 for a better profile selection. Select the Battery type that matches either of them as given in the Table. To set the Battery type to be used to LSI, write the value specified in Table 4 to Change of The Parameter register (0x12). For example write 0x01 to Change of The Parameter to select the Battery Type−04.

APA value to improve RSOC accuracy

APA values are parameter to fit installed battery profiles in a target battery characteristics. Appropriate APA value for the target battery will improve RSOC accuracy. The APA values are set in APA register (0x0B).

Typical APA values can be taken from the design capacity of the battery in table 5. The table shows relations of typical APA value and the design capacity. Use capacity per 1−cell for the table if some batteries are connected in parallel. Calculate APA values using linear supplement if there is not a requested design capacity in the table. See eq. 1 to calculate APA value manually. An example for 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 e.g in case of 1500 mAh Battery Type−01.

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

The upper 8−bits and the lower 8−bits of APA register are for charging and discharging adjustment parameters each. See table 6 for bit configuration. Table 5 shows both the bits with similar value. For example: set your value in APA register as 0x0D0D for 0x0D APA value.

But RSOC accuracy can be improved by setting different APA values for each bits depending on the target battery characteristics.

Please contact ON Semiconductor if you don't satisfy the RSOC accuracy. The deeper adjustment of APA for charging and discharging will increase the calculation accuracy.

Table 3. PARAMETER VS REGISTER

Command Code	Register Name	Parameter	Mandatory or Option	Unit	IC Type
0x06	TSENSE1 Thermistor B	B−constant of a TSENSE1 thermistor	Mandatory	K	LC709204F LC709205F
0x0B	APA	Adjustment parameter for RSOC measurement	Mandatory	−	
0x0C	APT	Delay time to temperature sampling	Option	−	
0x0E	TSENSE2 Thermistor B	B−constant of a TSENSE2 thermistor	Option	K	
0x12	Change Of The Parameter	Battery profile	Mandatory	−	
0x18	DesignCapacity	Design capacity of a target battery	Mandatory	0.1 mAh	LC709205F
0x1C	Termination Current Rate	Termination current rate at the end of charging	Option	0.01C	LC709204F LC709205F
0x1D	Empty Cell Voltage	Empty Cell Voltage	Option	mV	
0x31	Sense Resistance	Resistance of a sense resistor	Mandatory	0.1 mΩ	LC709205F

Table 4. BATTERY PROFILE VS REGISTER

IC Type	Battery Type	Nominal / Rated Voltage	Charging Voltage	Number of The Parameter (0x1A)	Change of The Parameter (0x12)
LC709204F LC709205F	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

Table 5. TYPICAL APA VALUE FOR CHARGING AND DISCHARGING ADJUSTMENT

Design Capacity / Cell (Note 1)	APA[15:8], APA[7:0]				Design Capacity / Cell (Note 1)	APA[15:8], APA[7:0]	
	Type−01	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	–				

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

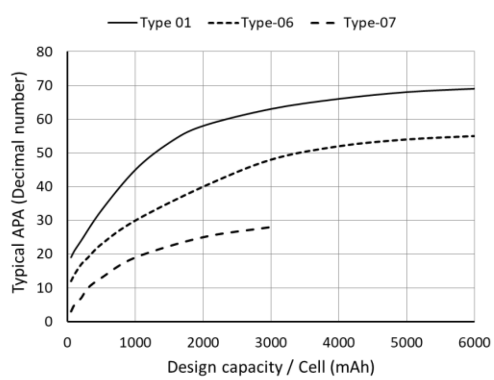


Figure 3. Typical APA of Type−01/06/07

Table 6. 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 (0x06/0x0E)

This section explains how to find an appropriate B constant value to set in the Thermistor B register (0x06 and 0x0E). The types of thermistor that this LSI can support is 10 kΩ NTC thermistor. It is possible to measure two NTC thermistors while setting different B constant value for each in LSI register. Cell temperature (TSENSE1) is an essential parameter used for the battery measurement. It is required to set an appropriate value to TSENSE1 Thermistor B (0x06) unless the application processor provides the battery temperature directly to this LSI (using I^2C mode). Application processor can use TSENSE2 (Ambient temperature) parameter for other purposes.

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

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

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 7 shows an example for the relationship between 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.

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

A sample plots using eq. 4 is shown in Figure 4. 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 4, B constant = 3400 K will give higher RSOC accuracy for the given range of temperature.

Another example is shown in Table 8. If only the temperature range and B constant is 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 7. 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

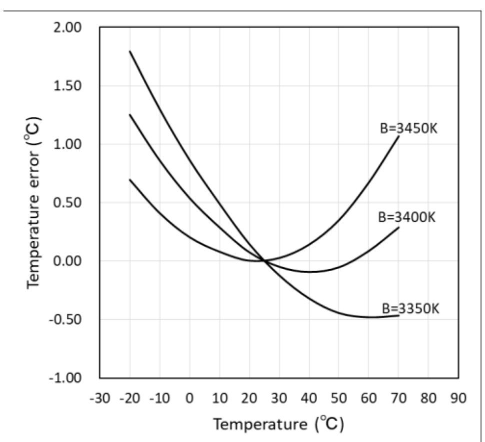


Figure 4. An Example of Temperature Error which is Calculated from a Thermistor Resistance

Table 8. 10 kΩ NTC THERMISTOR EXAMPLE (2)

R0 or R25	B constant		
	25−50°C	25−80°C	25−100°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 TSENSE1 and TSENSE2 pins while measuring temperature with a NTC thermistor. This LSI optimizes the temperature measurement interval automatically based on the battery current flow. The measurement interval ranges between a few seconds to a minute. Two 10 kΩ pull−up resistors are integrated with TSENSE1 and TSENSE2 in LSI as shown in Figure 5. These resistors are connected to the REG supply only during the temperature measurement. Pins remain in a high− impedance state except while measuring the temperature. Figure 6 shows an example of TSENSE1 and TSENSE2 waveforms during the temperature measurement. When the voltage on TSENSE1 and

TSENSE2 gets stabilized while thermistors are connected to the pins, the voltage on these pins is measured in turn for finding the target temperature. The pull−up resistors automatically gets disconnected from REG power supply after a successful temperature measurement.

The APT delay is shown in Figure 6, it shows the time delay until voltage measurement starts when REG power is supplied to the thermistors. The APT register shown in Table 3 is used to change the APT delay. APT delay is calculated using eq. 5 based on the value set in the APT register.

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

To improve the accuracy of temperature measurement, the voltage on TSENSE1 and TSENSE2 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 of the applications, initially defined APT delay time on LSI is sufficient for the voltage stability. However, for the battery pack examples like shown in Figure 7 need to consider the APT delay. The capacitive element is placed in parallel with the thermistor in given example. It is assumed that it will take longer time for the voltage of TSENSE1 and TSENSE2 to stabilize. It also takes 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.

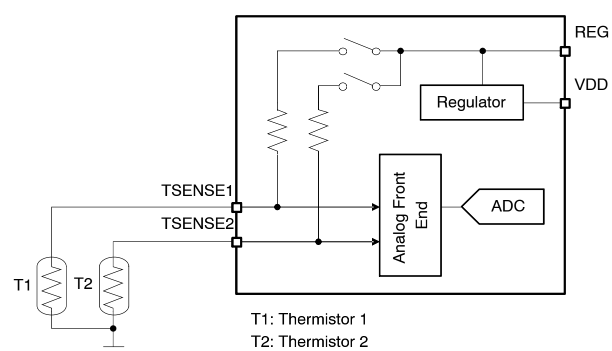


Figure 5. TSENSE1/TSENSE2 Port Block Diagram

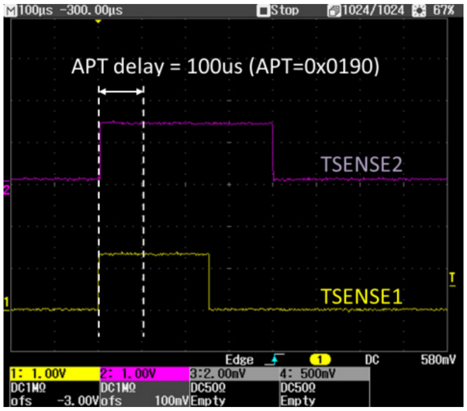


Figure 6. TSENSE1/TSENSE2 pulse at 25°C (APT = 0x0190)

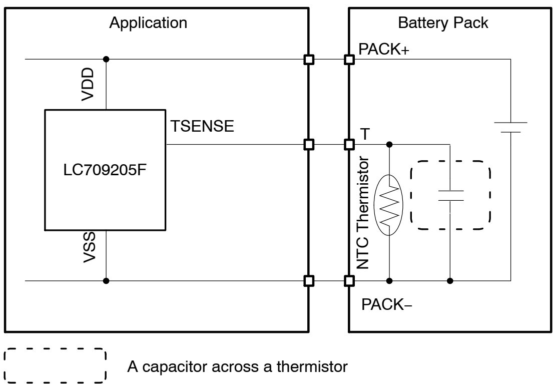


Figure 7. An Example of a Capacitor Across the Thermistor

Design Capacity (0x18)

Design capacity or Typical capacity or Nominal capacity can be found on the battery datasheet.

Termination Current Rate (0x1C)

Termination current rate is manually calculated by dividing Termination current with Design capacity. Termination current is given on the datasheet of the battery used. Further, Termination current rate can be calculated using higher values than that of datasheet defined value for higher safety. For example if termination current on datasheet is 0.02C, users can select 0.02C or higher values while calculating Termination current rate.

Both the LSI only support battery types with 0.02C or higher termination current values. Battery types that are lower than 0.02C must calculate Termination current rate using 0.02C or higher values.

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 RSOC rescaling section.

Sense Resistance (0x31)

Typical resistance of a sense resistor can be found on the datasheet. All of the sense resistance, the maximum load and charge current must satisfy eq. 6.

$$-24\ \text{mV} < \text{IMAX} \times \text{RSENSE} < 24\ \text{mV} \qquad \text{(eq. 6)}$$

Table 9 shows an example of sense resistance.

Table 9. EXAMPLES OF SENSE RESISTANCE

RSENSE	0x31 Register	Current resolution	Current range
2 mΩ	0x0014	1500 μA	±12 A
5 mΩ	0x0032	600 μA	±4.8 A
10 mΩ	0x0064	300 μA	±2.4 A
20 mΩ	0x00C8	150 μA	±1.2 A
50 mΩ	0x01F4	60 μA	±0.48 A
100 mΩ	0x03E8	30 μA	±0.24 A
200 mΩ	0x07D0	15 μA	±0.12 A

FUNCTIONAL DESCRIPTION

Get Initial RSOC after Power−on Reset

This LSI starts the initialization sequence automatically when the power−on reset is released after a battery pack insertion. Please refer to the LSI data sheet for the duration of the initialization sequence. During the initialization sequence, LSI acquires Cell voltage for the RSOC initialization. Initial RSOC is obtained using Open Circuit voltage (OCV) of the battery. OCV is the measurement of battery voltage at no−load condition. The LSI has a built−in OCV look−up table. Measured cell voltage is applied to the table to get new Initial RSOC. After the completion of initialization sequence, acquired initial RSOC is set in the RSOC (0x0D) and ITE (0x0F) registers.

Obtaining an Initial RSOC using Before RSOC

A battery or charger may supply the power to the VDD of LSI. If the RSOC value after the completion of initialization sequence is not as expected, it is assumed that the battery was charged or discharged during that period. If the charger was not operating during the initialization sequence, Before RSOC command can be used to get a more accurate RSOC.

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 command can initialize RSOC using the other 2nd to 4th sampled voltage. To select the appropriate voltage for initialization, write DATA as shown in Table 10 to 0x04 register. After writing 0x04, RSOC (0x0D) can be read again for the expected RSOC value. Select the highest RSOC value from the sampled RSOCs DATA. This is because the highest RSOC is obtained using the highest voltage and that voltage is closer to the OCV.

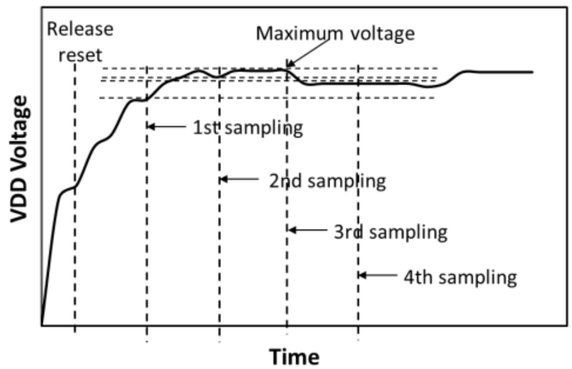


Figure 8. Sampling Order for Before RSOC Command

Table 10. 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, Lib−protection IC disconnects the battery when an overvoltage or overcurrent is detected. The power supply to the LSI is also stopped at that time. General Lib−protection IC reconnects the battery when it detects a voltage supply from the charger. In such cases, charger starts to supply power to this LSI first. Therefore, the voltage acquired by the LSI 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 using the charge voltage. Following two functions are explained 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 of 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 get RSOC 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 LSI to Operational mode and set Current Direction (0x0A) register to Auto mode.

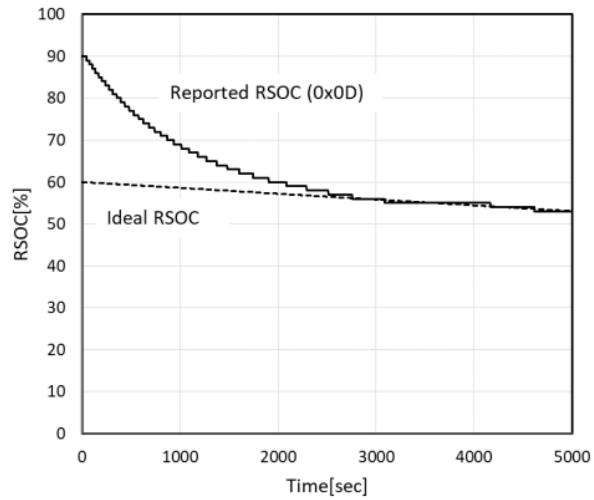


Figure 9. An Example of RSOC Automatic Convergence with 0.05C Load Current. RSOC: 90% to 60%

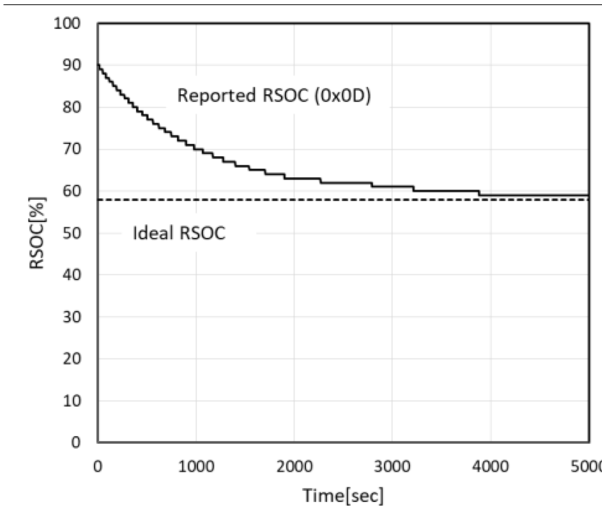


Figure 10. An Example of RSOC Automatic Convergence without Load Current. RSOC: 90% to 58%

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).

In order to select an appropriate profile for your applications, write the value for your profile in Change of The Parameter register. RSOC initialization is performed again as soon as the profile is changed. For the initialization sequence, OCV look−up table of the selected profile and the 1st sampled cell voltage is used.

Use 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 corresponding to TSENSE1 or TSENSE2 to 1 to measure the temperature with the attached thermistor. The bit selection details is shown in Table 11. Battery temperature information is an essential parameter for the RSOC measurement. If the thermistor in the battery pack is connected to another device, LSI cannot measure the battery temperature using the thermistor. In that case, set TSENSE1 to I^2C mode. This LSI cannot update the Cell temperature in I^2C mode. Application processor must write the battery temperature to Cell temperature (0x08). For the high precision RSOC measurement, it is recommended to update the cell temperature every time when the temperature changes more than 1°C. Temperature update is not required when the LSI is in Sleep mode.

Table 11. STATUS BIT

Register name	Status BIT	Set value in Status Bit	
		0	1
Cell temperature (TSENSE1)	BIT0	I^2C Mode	Thermistor Mode
Ambient Temperature (TSENSE2)	BIT1	Disable	Thermistor Mode

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

Alarm Functions

By using the alarm functions, application processor can quickly detect a condition exceeding a preset threshold. Table 12 shows the registers for setting alarm thresholds and the monitored registers by the alarm function. The alarm function is disabled if the threshold register have default value. When an alarm condition occurs, this LSI 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.

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

Log Functions

Table 13 shows the list of log register and the monitored register by the log function. These log functions start counting from the initial value and detects maximum and minimum log values after the initialization sequence of LSI. The log function is only effective in Operational mode. All the log registers are Read/write enabled except CycleCount (0x17).

If these registers are written with 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.

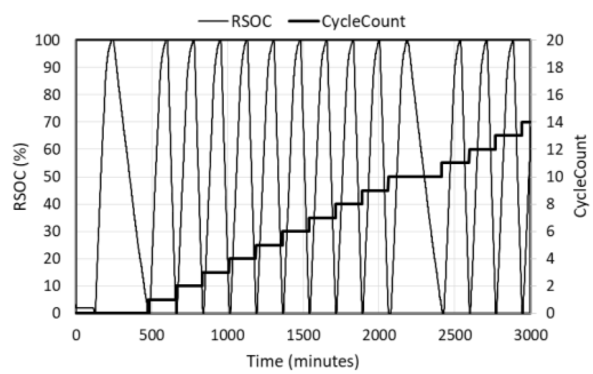


Figure 11. CycleCount (0x17) Report Example

Table 12. ALARM FUNCTIONS

Threshold Register	Monitored Register	BIT of Battery Status (0x19)	Unit	IC Type
Alarm High Cell Voltage (0x1F)	Cell Voltage (0x09)	15	mV	LC709204F LC709205F
Alarm Over Charging current (0x22)	Dynamic Cell current (0x33)	13	μV	LC709205F
Alarm High Temperature (0x21) (Note 2)	Cell Temperature (TSENSE1) (0x08)	12	0.1 K	LC709204F LC709205F
Alarm Low Cell Voltage (0x14)	Cell Voltage (0x09)	11	mV	
Alarm Over Discharging current (0x23)	Dynamic Cell current (0x33)	10	μV	LC709205F
Alarm Low RSOC (0x13)	RSOC (0x0D)	9	%	LC709204F LC709205F
Alarm Low Temperature (0x20) (Note 2)	Cell Temperature (TSENSE1) (0x08)	8	0.1 K	

2. These alarms are enable when TSENSE1 is Thermistor mode.

Table 13. LOG FUNCTIONS

Log register	Monitored register	Unit	Initial value	IC Type
CycleCount (0x17)	RSOC (0x0D)	count	0x0000	LC709204F LC709205F
TotalRuntime (0x25,0x24)	N/A	minutes	0x0000	
Accumulated Temperature (0x27,0x26)	Cell Temperature (TSENSE1) (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 (TSENSE1) (0x2C) (Note 3)	Cell Temperature (TSENSE1) (0x08)	0.1 K	0x0980	
Minimum Cell temperature (TSENSE1) (0x2D) (Note 3)	Cell Temperature (TSENSE1) (0x08)	0.1K	0x0DCC	
Maximum Cell Current (0x2E)	Dynamic Cell current (0x33)	μV	0x0000	LC709205F
Minimum Cell Current (0x2F)	Dynamic Cell current (0x33)	μV	0x0000	

3. These logs are updated when TSENSE1 is Thermistor mode.

Detection of Battery Status

This LSI detects whether the battery is charged or discharged and outputs that status to the Discharging Bit (Bit 6 of BatteryStatus). Table 14 shows the relationship of Discharging bit with the Battery status. Figure 12 shows an example of Discharging Bit measurement when the battery is charging, discharging, and at no load condition.

Table 14. DISCHARGING BIT (BIT 6 OF BATTERY STATUS REGISTER)

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

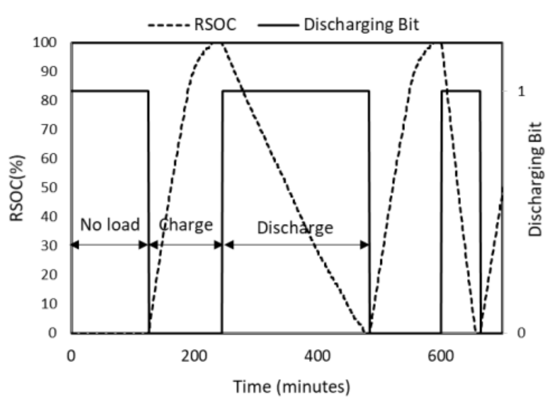


Figure 12. Discharging Bit and RSOC during Charge and Discharge Cycle

Detection of System reset

This LSI is directly powered from the battery. If the following situation occurs power supply is stopped and this LSI is reset.

- The battery is removed
- The battery voltage falls below the reset release voltage of this LSI due to excessive load current
- Lib protection IC disconnects the battery

In order to continue the battery measurement smoothly, it is highly recommended to perform the control operation as shown in Figure 13. This operation will be valid for those applications that are expected to experience the above situations.

Status bit 7 of BatteryStatus (0x19) or INITILIZED is automatically set to 1 after a power−on reset. If the application processor had set this bit to 0 immediately after the last power−on reset, the processor can detect the LSI reset operation by reading this bit again. If INITILAIZED is 1 then execute the Starting flow again.

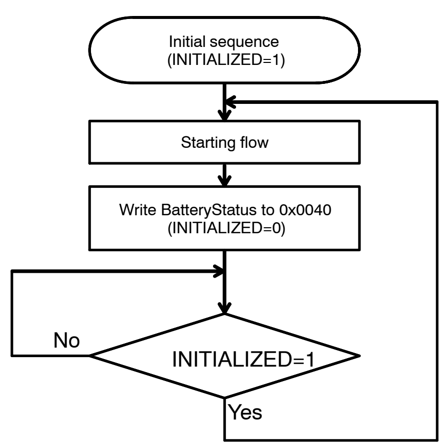


Figure 13. Flow to Restart the Gauge after Excessive Voltage Drop

How to Estimate Time to Empty

This section describes how LSI estimates the battery remaining time. Time to Empty register (0x03) provides estimated remaining time until RSOC reach 0%. This LSI automatically learns an average time that is required for 10% RSOC decrease during each discharging operations. Time to Empty is calculated by using the learned decreased rate before RSOC reach 0%. See Figure 14 for details. If RSOC increases after a charging operation, previously learned decrease rate before charging is used to predict Time to Empty.

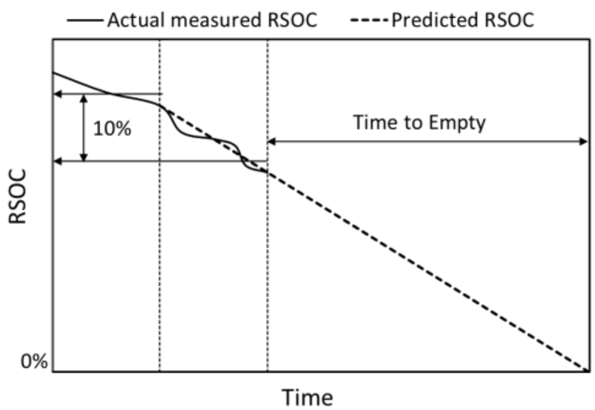


Figure 14. How to Estimate Time to Empty

How to Estimate Time to Full

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

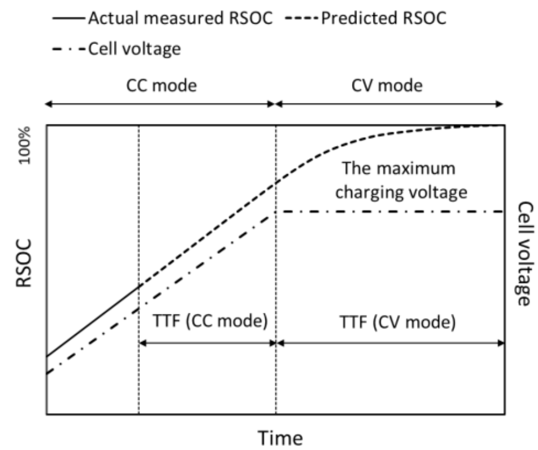


Figure 15. How to Estimate Time to Full

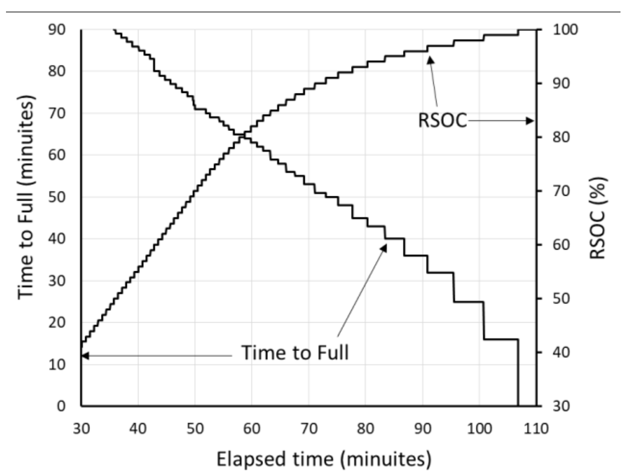


Figure 16. Time to Full (0x05) Report Example under CC−CV Charging

IDD Reporting Functions

LC709205F has two IDD reporting functions i.e. Dynamic Cell Current (0x33) and Average Cell current (0x34). The value of these registers is signed int. The positive sign indicates battery charging and the negative sign indicates battery discharging operation. The unit is given in μV. It can be converted to current on dividing by sense resistance.

Dynamic Cell Current contains the latest measured current. The current is used to detect Alarm for over charging and discharging current. Average Cell current contains average current. This current will help to evaluate an average consumption current of an application and battery charging current. Figure 17 and 18 show the result of Dynamic Cell Current and Average Cell current measurement using a smart phone. Some apps operate on the Smart phone during the measurement. In the figures Dynamic Cell Current measures fluctuations in consumption current of the Smart phone. On the one hand Average Cell current measures current that the fluctuations are reduced.

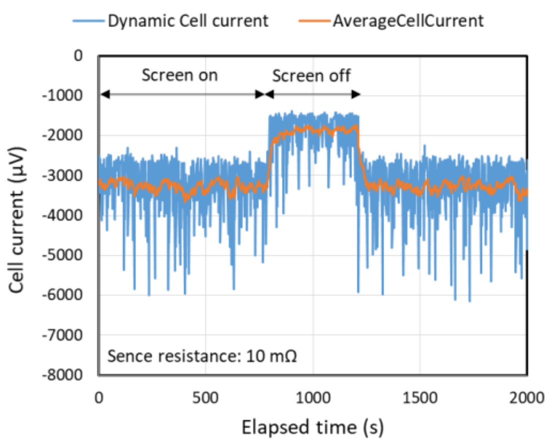


Figure 17. Example of Discharging Current Measurement with a Smart Phone

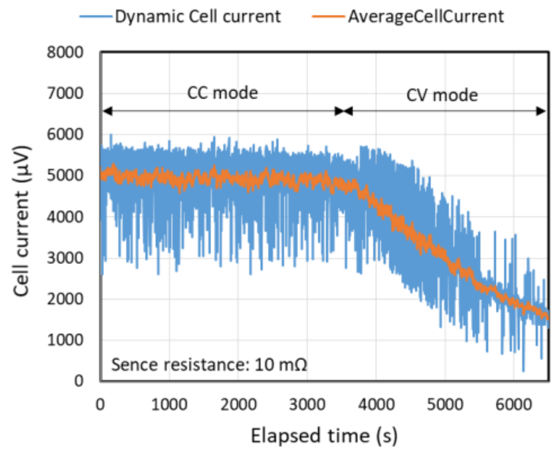


Figure 18. Example of Charging Current Measurement with a Smart Phone

I^2C Communication Protocol

This section describes I^2C protocol and the actual waveform. Refer to the datasheet about the characteristics.

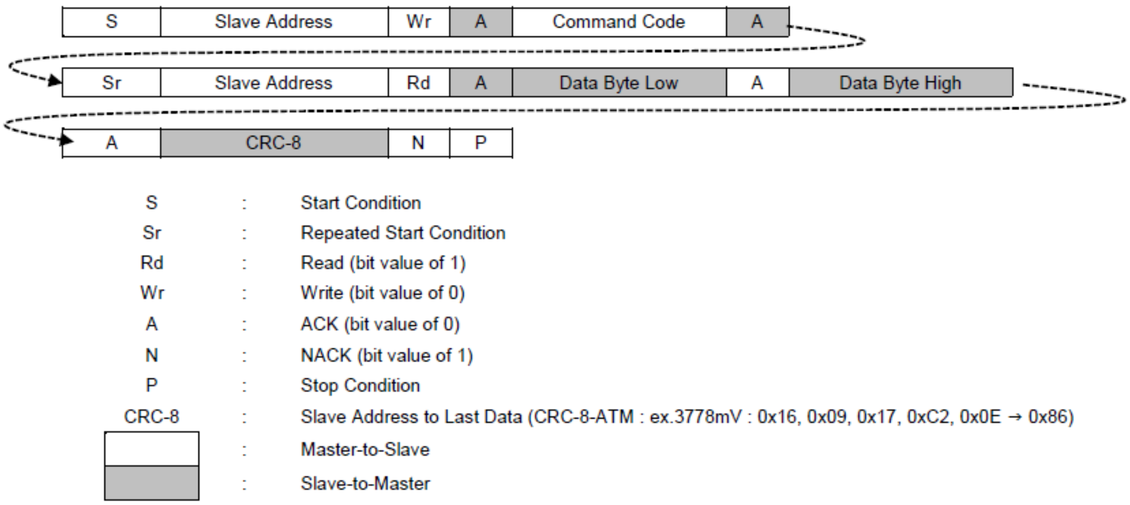


Figure 19. Read Word Protocol

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 20. Overview of Read Waveform

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

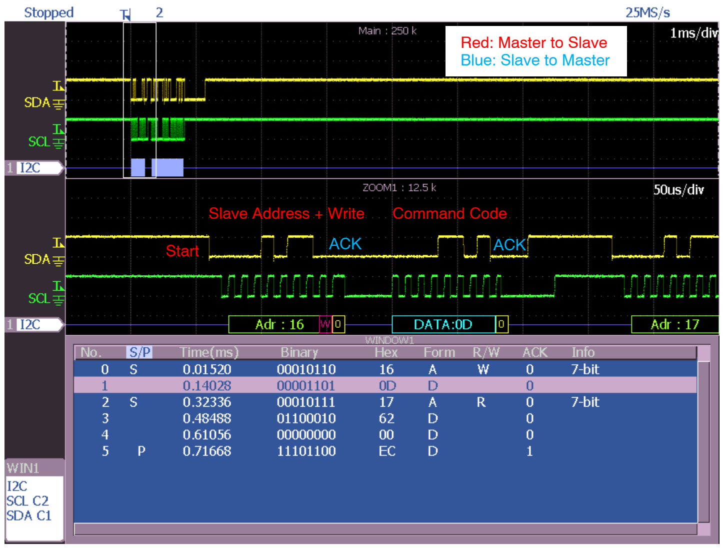
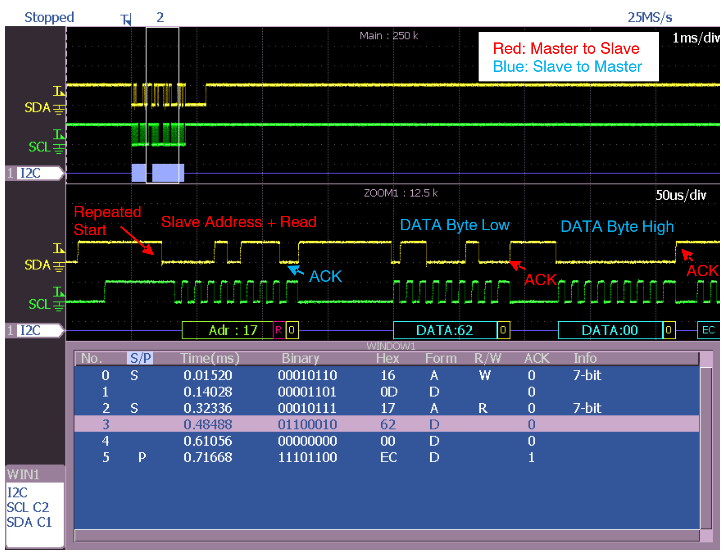


Figure 21. Read Waveform (1)

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 22. Read Waveform (2)

3. CRC−8: 0xEC

Figure 23. Read Waveform (3)

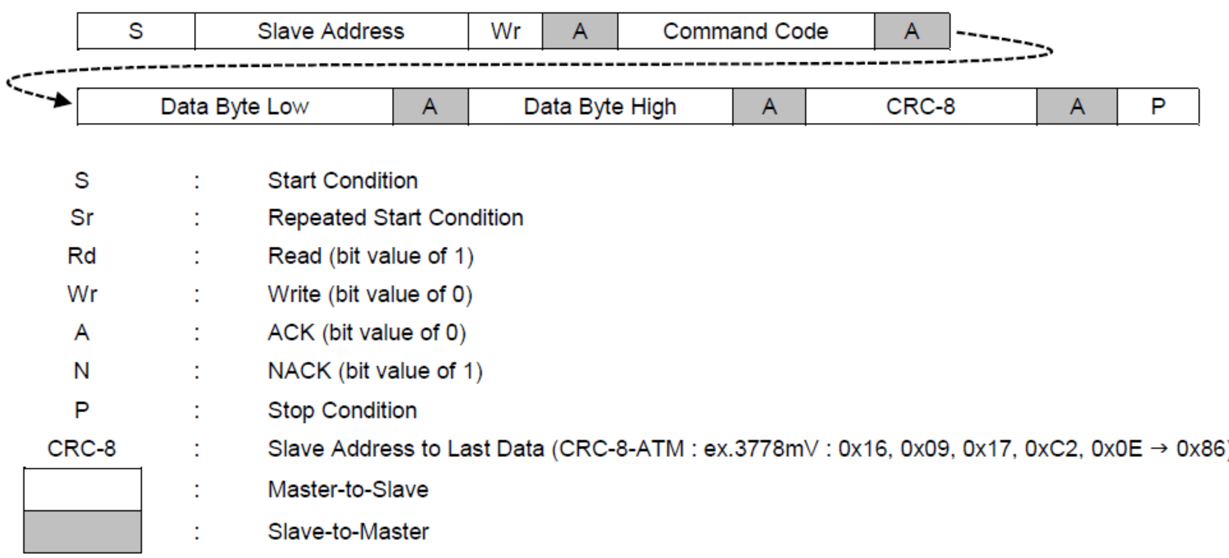


Figure 24. Write Word Protocol

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)

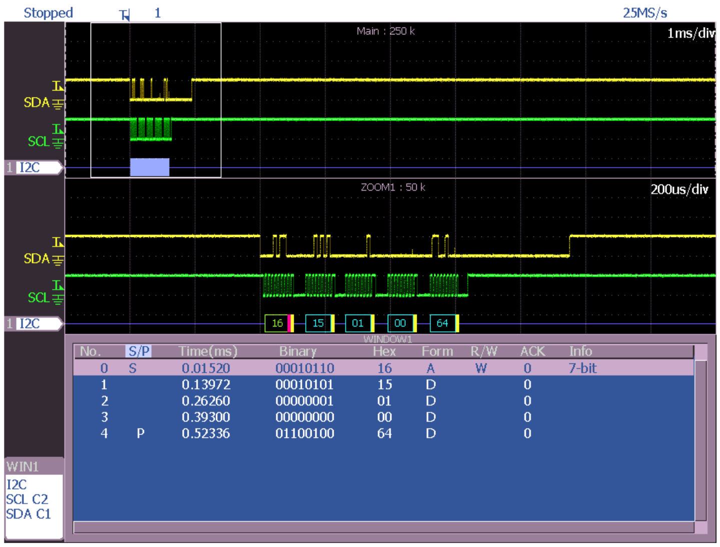


Figure 25. Overview of Write Waveform

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

Command Code: 0x15

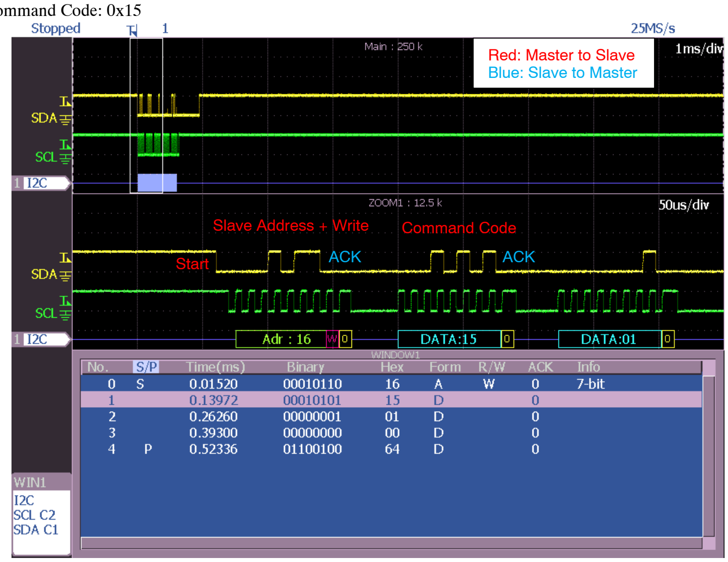


Figure 26. Write Waveform (1)

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

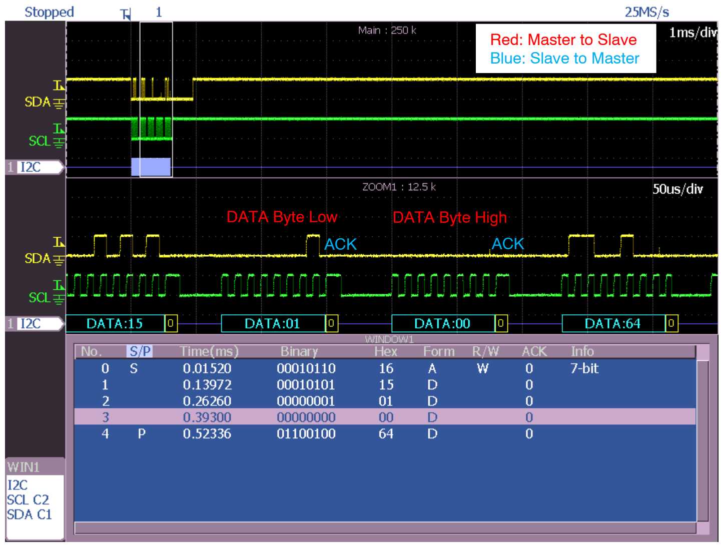


Figure 27. Write Waveform (2)

3. CRC−8: 0x64

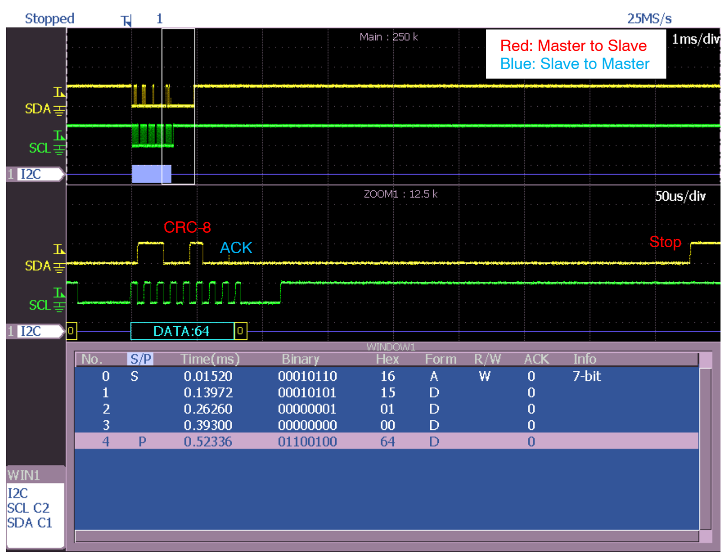


Figure 28. Write Waveform (3)

STARTING FLOW AND SAMPLE CODE

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

Sample code

- CRC−8 calculation
- LC709204F Starting flow with Thermistor mode
- LC709204F Starting flow with I^{2}C mode
- LC709205F Starting flow with Thermistor mode
- LC709205F 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 LSI starts initial sequence automatically after reset release with power−on reset. I2C communication is enabled after the sequence. Then set registers to start gauging according to 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
}
```

LC709204F Starting Flow with Thermistor Mode

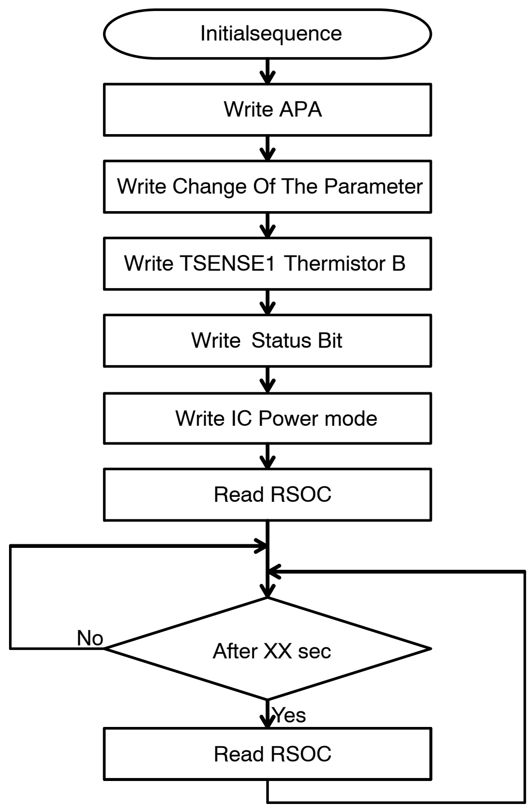


Figure 29. LC709204F Starting Flow with Thermistor Mode

```
/**
 *==============================================================================
 *  Sample of Application Processor(LC709204F / Thermistor mode)
 *==============================================================================
 */

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
}

int main( void )
{
    unsigned short      u2RSOC;

    /*
        Battery connection
            ↓
        LC709204F 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 : TSENSE1 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
            }
        }
    }

}
```

LC709204F Starting Flow with I^{2}C Mode

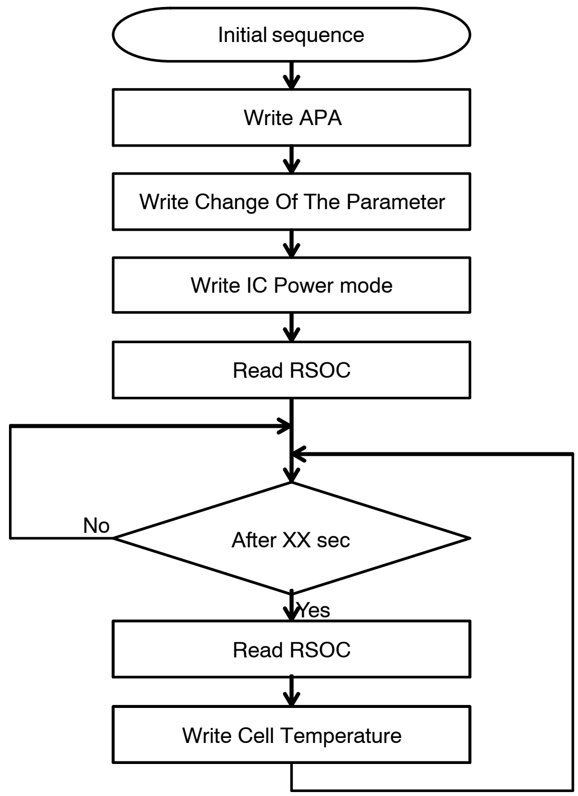


Figure 30. LC709204F Starting Flow with I2C Mode

```
/**
 *==============================================================================
 *  Sample of Application Processor(LC709204F / I2C mode)
 *==============================================================================
 */

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
}

int main( void )
{
    unsigned short      u2RSOC;

    /*
        Battery connection
            ↓
        LC709204F 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. 25℃ → 25 * 10 + 2732 → 0x0BA6)
        }else{
            // SmartPhone Power OFF
            while( SmartPhone_PowerOff ){
                // AP Low Power Mode
            }
        }
    }

}
```

LC709205F Starting Flow with Thermistor Mode

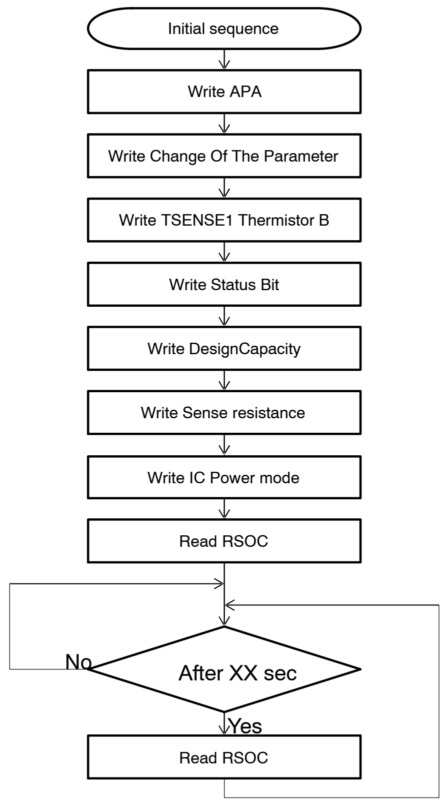


Figure 31. LC709205F Starting Flow with Thermistor Mode

```
/**
 *================================================================================
 *  Sample of Application Processor(LC709205F / Thermistor mode)
 *================================================================================
 */

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
}

int main( void )
{
    unsigned short      u2RSOC;

    /*
        Battery connection
            ↓
        LC709205F 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 : TSENSE1 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( 0x18 , 0x2710 );          // Slave Function : Design Capacity
                                             // Command : 0x18
                                             // Data : 0x2710 (ex. Design Capacity = 1000mAh)

    i2c_WriteWord( 0x31 , 0x0064 );          // Slave Function : Sense Resistance
                                             // Command : 0x31
                                             // Data : 0x0064 (ex. Sense Resistance = 10mohm)

    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
            }
        }
    }

}
```

LC709205F Starting Flow with I^2C Mode

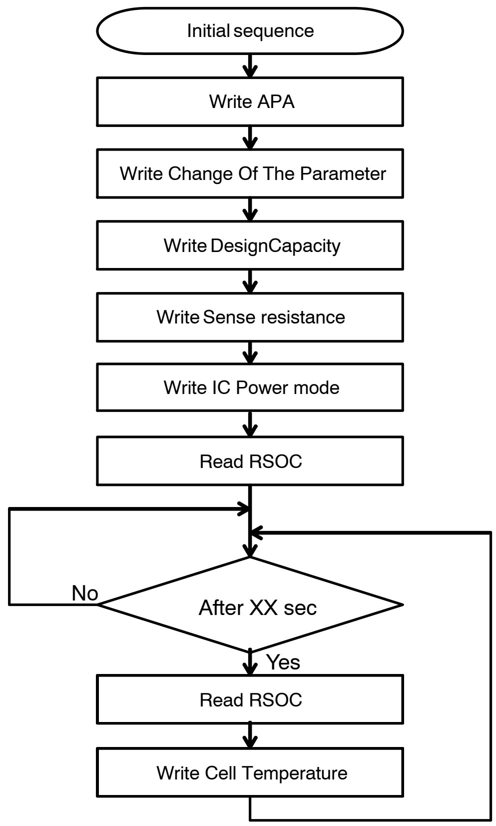


Figure 32. LC709205F Starting Flow with I2C Mode

```
/**
 *==============================================================================
 *  Sample of Application Processor(LC709205F / I2C mode)
 *==============================================================================
 */

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
}

int main( void )
{
    unsigned short      u2RSOC;

    /*
        Battery connection
            ↓
        LC709205F 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( 0x18 , 0x2710 );         // Slave Function : Design Capacity
                                            // Command : 0x18
                                            // Data : 0x2710 (ex. Design Capacity = 1000mAh)

    i2c_WriteWord( 0x31 , 0x0064 );         // Slave Function : Sense Resistance
                                            // Command : 0x31
                                            // Data : 0x0064 (ex. Sense Resistance = 10mohm)

    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. 25℃ → 25 * 10 + 2732 → 0x0BA6)
        }else{
            // SmartPhone Power OFF
            while( SmartPhone_PowerOff ){
                // AP Low Power Mode
            }
        }
    }

}
```

User ID Writing Protocol

User ID (0x36, 0x37) provides 32−bits programmable registers in LSI built−in NVM. These resisters can be used for various purposes if required. User must program the built−in NVM preliminarily to use this function. A master device can program it using I^2C commands. See Figure 33 for block diagram. The I^2C communication protocol is explained below.

Conditions for ID Writing

Following operating conditions must be satisfied during programming User ID.

Allowable Operating Conditions during ID 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. Then the master device should control to prevent multiple ID programming. See Figure 34. Read User ID register before programming. Start programming if the read data is not same as the target data.

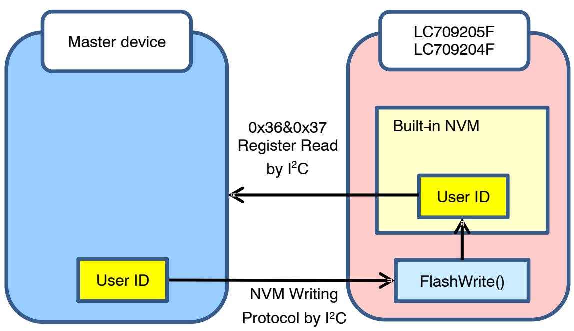


Figure 33. Block Diagram about User ID Writing

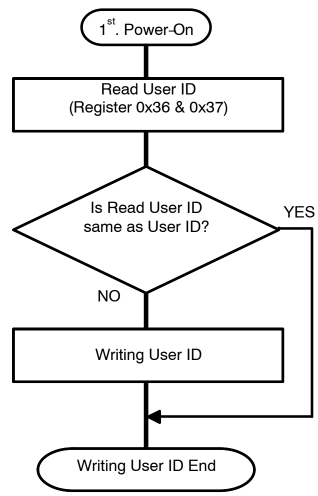


Figure 34. Flow to Prevent Multiple ID Writing

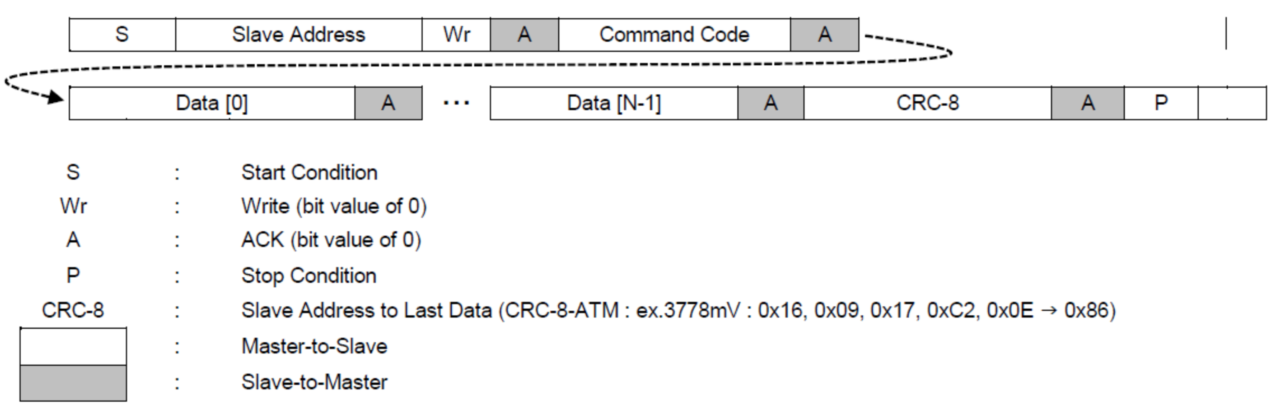


Figure 35. Write N−bytes Data Protocol for ID Writing

Table 15. 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 byte (0x55AA)
0x01	Enter Write mode	W	2 byte (0x55AA)
0x02	Set data	W	130 byte (instruction[2], UID[4], data[124])
0x03	Set key1	W	2 byte (0x55AA)
0x04	Set key2	W	2 byte (0x00A0)
0x05	Write exe / Verify exe	W	2 byte (0x55AA)
0x06	Start verify	W	4 byte (instruction[4])
0x07	Read verify result	R	2 byte (result[2])
0x08	Reset Write mode	W	2 byte (0x55AA)

4. 0x03 to 0x08 commands are enable in Write mode.

Outline of User ID Writing Flow

Following process is required for a successful write operation of User ID registers. Flow diagram is shown in Figure 36.

- Change power mode
  This process changes power mode to Test mode.
- Enter write mode
  This process changes executing routine to "NVM Write Routine" from "Normal Routine"
- Wait 300 ms
  Waiting 300 ms is needed to change executing routine.
- Data transfer #1
  This process sends 128 bytes data include 32bits User ID with instruction of write to User ID of NVM
- Start Verify
  This process sends notification of start of verify with instruction of verify to NVM.
- Data transfer #2
  This process sends 128 bytes data include 32 bits User ID with instruction of verify to User ID of NVM
- Read result
  This process receives result of verify
- Reset Write mode
  This process does reset LC709205F
- Wait 1.5 s
  Wait for 1.5 s if you operates the registers of LC709205F continuously without power−off

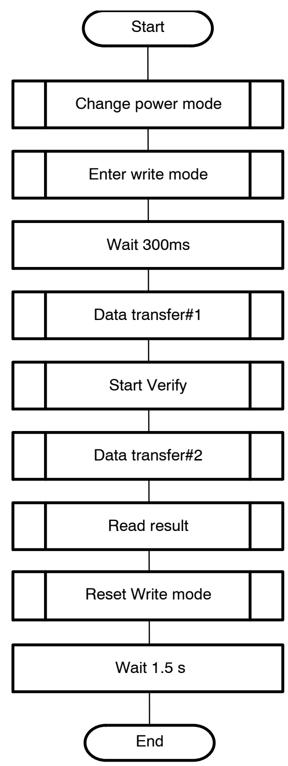


Figure 36. Outline of User ID Writing Flow

Change Power Mode

- This command is Write Command
- This command changes power mode to Test mode

Table 16.

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

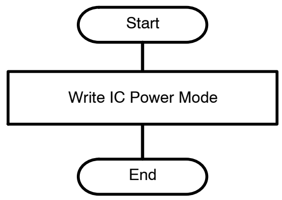


Figure 37. Change Power Mode

Enter Write Mode

- These commands are Write Command
- This command changes executing routine to “NVM Write Routine” from “Normal Routine”

Table 17.

	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

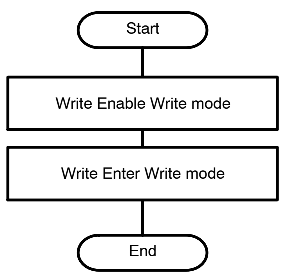


Figure 38. Enter Write Mode

Data Transfer#1

- These commands are Write Command for User ID Writing Protocol
- This command writes 128 bytes data to NVM
- Waiting 40 ms is needed to wait end of write to NVM
- “Set data” command’s CRC8 is changed by Data[2~5]
- Create UID data from 32bits User ID using the following formula
  - Lower 16bits UID = Lower 16bits User ID – 0x55AA
  - Upper 16bits UID = Upper 16bits User ID – 0x55AA

  ex.) In the case of 32bits User ID 0x12345678
  - Lower 16bits UID = 0x5678 – 0x55AA = 0x00CE … UID[0] = 0xCE, UID[1] = 0x00
  - Upper 16bits UID = 0x1234 – 0x55AA = 0xBC8A … UID[2] = 0x8A, UID[3] = 0xBC
- Data[0] = Fixed value 0x80
- Data[1] = Fixed value 0x81
- Data[2] = UID[0] data
- Data[3] = UID[1] data
- Data[4] = UID[2] data
- Data[5] = UID[3] data

Table 18.

	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

Table 19.

	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

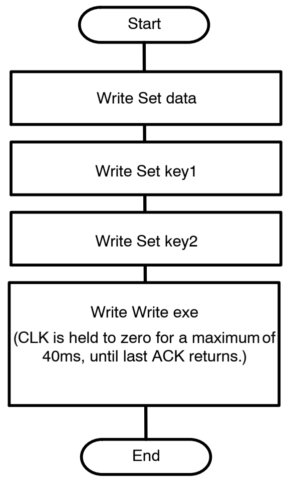


Figure 39. Data Transfer#1

Start Verify

- This command is Write Command for User ID Writing Protocol

Table 20.

	Slave Address (W)	Command Code	Data[0]	Data[1]	Data[2]	Data[3]	CRC−8
Start verify	0x16	0x06	0x80	0x81	0x00	0x82	0xF5

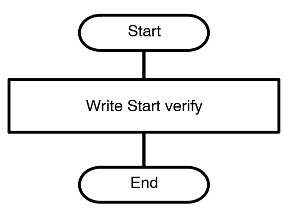


Figure 40. Start Verify

Data Transfer#2

- These commands are Write Command for User ID Writing Protocol
- This command writes 128 bytes data to NVM
- Waiting 40 ms is not needed to verify data
- "Set data" command's CRC8 is changed by Data[2~5]
- Create UID data from 32bits User ID using the following formula
  - Lower 16bits UID = Lower 16bits User ID – 0x55AA
  - Upper 16bits UID = Upper 16bits User ID – 0x55AA

    ex.) In the case of 32bits User ID 0x12345678
    - Lower 16bits UID = 0x5678 – 0x55AA = 0x00CE … UID[0] = 0xCE, UID[1] = 0x00
    - Upper 16bits UID = 0x1234 – 0x55AA = 0xBC8A … UID[2] = 0x8A, UID[3] = 0xBC
- Data[0] = Fixed value 0x80
- Data[1] = Fixed value 0x81
- Data[2] = UID[0] data
- Data[3] = UID[1] data
- Data[4] = UID[2] data
- Data[5] = UID[3] data

Table 21.

	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

Table 22.

	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
Verify exe	0x16	0x05	0xAA	0x55	0xE5

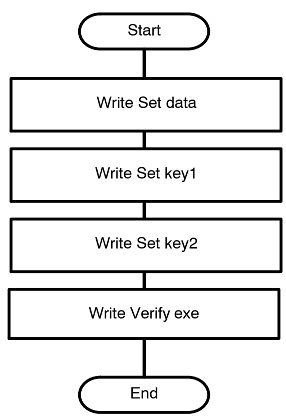


Figure 41. Data Transfer#2

Read Result

- This command is Read Command for User ID Writing Protocol
- Can read result of verification, by this command
- If verification was success, result is set to 0x0001
- If verification was failure, result is set to 0x0000
- This command's CRC8 is changed by Data[0~1]

Table 23.

	Slave Address (W)	Command Code	IC address (R)	Data[0]	Data[1]	CRC−8
Read verify result	0x16	0x07	0x17	Result Low8bit	Result High8bit	0xXX

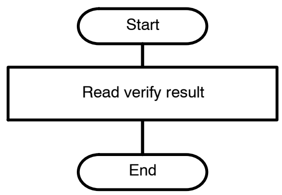


Figure 42. Read Result

Reset Write Mode

- This command is Write Command for User ID Writing Protocol
- If this command was executed, LC709205F is reset. Then all the registers are initialized. The initialized values are the same as them after power on reset

Table 24.

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

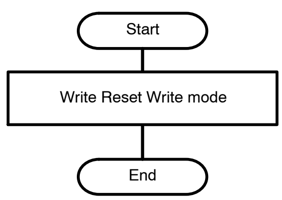


Figure 43. Reset Write Mode

Retry by Error

- I^2C Error at Change power mode
  Retry from (A) point. Resetting LC709205F is recommended when it becomes an error here
- I^2C Error at after Change power mode
  Retry from (B) point
- Result code is NG
  Retry from (B) point

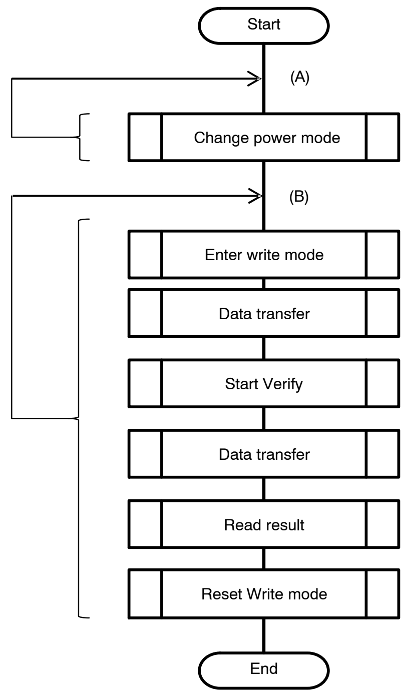


Figure 44. Retry by Error

This sample code writes User ID to 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_L = 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
            ↓
        LC709205F 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 to 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] = 0x00;
            u1Data[3] = 0x82;
            i2c_WriteData( 0x06 , u1Data , 4 );      // Slave Function : Start verify
                                                     // Command : 0x06
                                                     // Data[0] : 0x80 , Data[1] : 0x81 ,
                                                     // Data[2] : 0x00 , Data[3] : 0x82

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

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

            if( u2Result == 0x00001 ){
                // User ID writing success
                i2c_WriteWord( 0x08 , 0x55AA );          // Slave Function : Reset write mode
                                                         // Command : 0x08
                                                         // Data : 0x55AA

                break;
            }else{
                // User ID writing failure
            }
        }
    }
}
```

ON Semiconductor is licensed by the Philips Corporation to carry the I2C bus protocol.

PUBLICATION ORDERING INFORMATION

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

ON Semiconductor 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