Application Note

How to Charge a Smart Battery Using MCU in Between to Translate SMBus or I²C

Jibin Biju

ABSTRACT

When creating a Battery Management System (BMS), and the charger and gauge uses different communication protocols, such as Inter-Integrated Circuit (I²C) and System Management Bus (SMBus), the microcontroller unit (MCU) needs to interface between chargers and gauges, or needs a system that has different communication protocols for each device. This application note illustrates the use of MCU for a Smart Battery System (SBS) where the MCU polls the SMBus gauge, translates the received data to be compatible with the I²C charger (based on I²C specifications), and transmits to the I²C charger.

Table of Contents

List of Figures

List of Tables

Trademarks

1 Introduction

The SBS specifications are a set of standards that allow the system host to interface between Smart Battery Chargers, Smart Batteries, and other SMBus devices through the SMBus interface. The SBS has a set of features such as error detection, error signaling, and pre-defined commands from the System Management Specifications. These pre-defined commands are standardized functions that either read or write a 2-byte word with a minimum accuracy, range, granularity, and an invalid detection indication. These features and other specifications have to be met to be SBS compliant.

For this application note, the SBS specifications of interest are the charging voltage and current registers. The gauge is required to transmit *ChargingVoltage()* and *ChargingCurrent()* to the 0x15 and 0x14 commands respectively. The SBS Specification states the *ChargingVoltage()* data range is from 0 to 65,534 mV in voltage regulation while there is a good power supply attached. The *ChargingCurrent()* has the data range from 0 to 65,534 mA. For more information on exceptions, refer to the *ChargingCurrent()* (0x14) section and *ChargingVoltage()* (0x15) section of the SBS Specification.

2 The Role of MCU in Smart Battery System

The MCU allows SBS capabilities to be expanded where the MCU can configure advanced features of chargers. The MCU can translate between different communication protocols of components, allowing different SBS configurations to work regardless of communication compatibilities of individual devices. The MCU can also allow the user to implement charging algorithms beyond what is offered by chargers and smart battery packs. For most BMS systems, having an MCU act as the controller provides ease of use, more flexibility, more control, and facilitate troubleshooting.

The MCU for Smart Battery BMS system receives information from the gauge and transmits that data to the charger, or performs data modifications based on system need before transmission to the charger. In SBS, there is a broadcast mode where the gauge can transmit data to the charger without a host. In broadcast mode, the gauge transmits *ChargingVoltage()*, *CharingCurrent()*, and *AlarmWarning()* to the charger, but this is not always an option if there is a difference in communication protocols between charger and gauge. As a result, translation of the gauge data based on the communication protocol and charger properties is necessary. For this application note, the MCU polls the gauge to read *ChargingVoltage()* and *ChargingCurrent()*. Before translating from SMBus to I^2C, translation of gauge data is then performed by the MCU. Finally, the MCU transmits data to program the charger by I^2C.

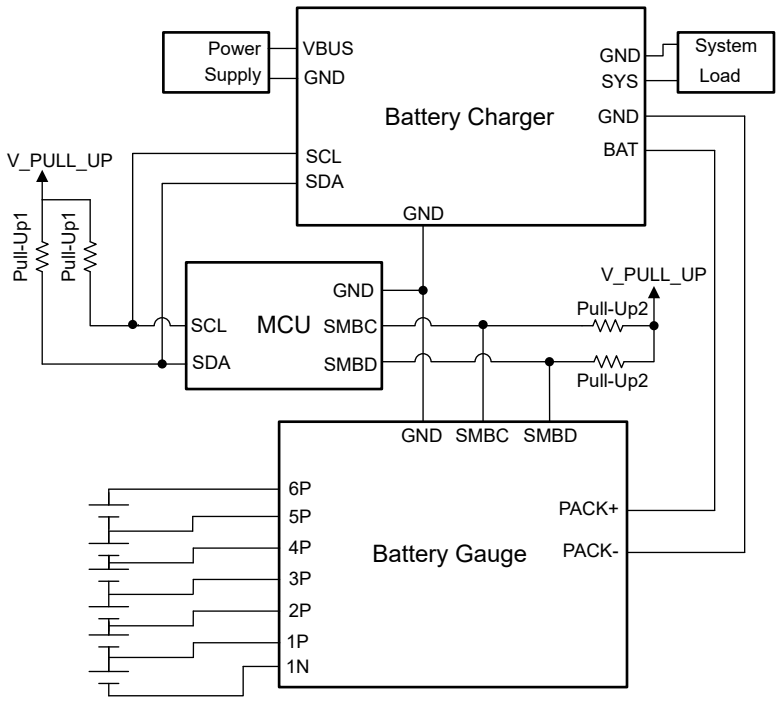

Figure 2-1. Conventional Battery Management System Configuration

3 Application Example Using BQ25750, BQ40z80, and MSPM0L1306

The gauge and charger are set up as per the data sheets. Gauge parameters such as charging currents, charging voltages, and temperatures ranges have been changed. The only charger parameter changed was feedback voltage (based on the MCU algorithm).

3.1 Gauge Setup

The gauge was set up such that using advanced charging algorithm and that the charging profile is similar to Figure 3-1.

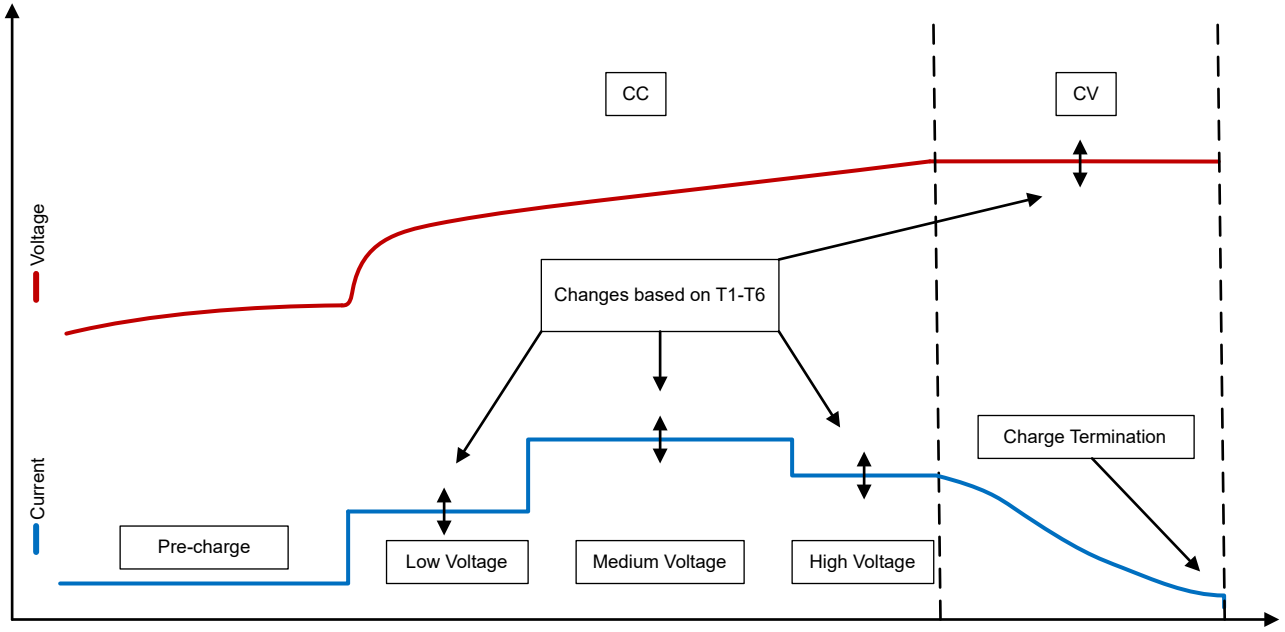

Figure 3-1. Charging Profile Using Advanced Charging Algorithm

The thresholds for each parameter within constant current (CC) mode, constant voltage (CV), and pre-charge can be modified following The Advanced Charge Algorithm Application Report. The parameters for this example were chosen for the sole purpose of clear transitions in the charging profile. The gauge is connected to a 6S4P Li-on battery.

Although BQ40z80 offers much more options, these are the simplified settings relevant for the application note and validate the charging profile. These settings are for each cell in the pack; there is another setting for the number of cells in series is programmed.

Table 3-1. Simplified Gauge Settings

Standard High Temp Charging	Value	Units
Max charge voltage	4100	mV
Low voltage threshold	2900	mV
Medium voltage threshold	3600	mV
High voltage threshold	3900	mV
Current at low voltage	1750	mA
Current at medium voltage	2250	mA
Current at high voltage	2100	mA
Termination current	300	mA

3.2 Charger Setup

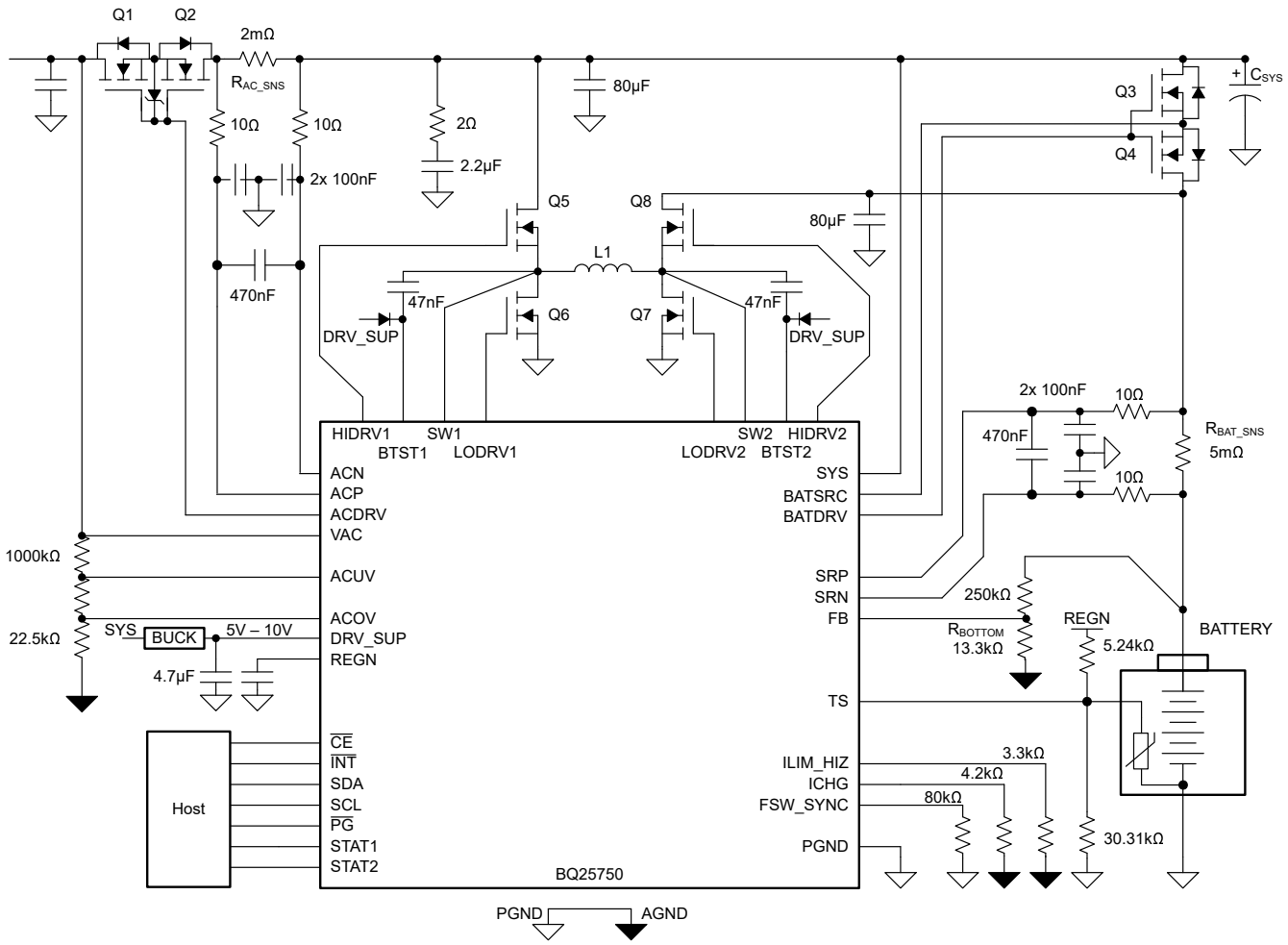

Figure 3-2. Typical Application Block Diagram of the BQ25750

TI provides a wide portfolio of chargers which are I^{2}C only, and SMBus only. Table 3-2 is the typical application of using the BQ25750 which is an I^{2}C based charger. For this application note, the only configuration changed from the above-mentioned block diagram is the bottom resistor (R_{BOTTOM}) of the voltage divider that determines the feedback voltage. The feedback voltage is then scaled to determine the desired battery regulation voltage. The resistor divider must be chosen such that the divider covers the different ranges of charging voltage requested from the gauge. In BQ25750, the resistor divider first sets the desired target battery voltage and then feedback voltage can be changed (through register 0x00).

For this example, the range of charging voltages from the gauge is 6×Voltage at different temperature ranges per cell. Depending on the temperature ranges, the low voltage, medium voltage, high voltage, and max charge voltage changes (set by user). Using the value in Table 3-1, users can determine the highest requested charging voltage is 6×4.1 V. After determining the range of charging voltage of the gauge, 16.7kΩ resistor is chosen for R_{BOTTOM} to cover 23.97V to 24.96V (3.995/cell to 4.16/cell) following the BQ25756 Design Calculator. If the range requested by the gauge is below the minimum value, then set to the lowest feedback voltage by the MCU. For more information on the BQ25750 or other charger setup and configuration, refer to the data sheet and user guide.

The charging current must be written to the Charge_Current_Limit 16 bit register at 0x02 address. The register has a range from 400mA-20000mA, the bitstep of 50mA, and only 9 bits out of the 16 bits are read by the charger to represent the charging current (only bits 10:2, the rest of the bits are reserved). See the Charge Charge_Current_Limit register section of the BQ25750: Standalone/I2C Controlled, 1- to 14-Cell Bidirectional

Buck-Boost Battery Charge Controller with Direct Power Path Control data sheet for more information on the details and refer to Section 3.5 for the exact formatting of the charging current.

Table 3-2. Charging Current Register Settings in BQ25750

Bit	Field	Type	Reset	Notes	Description
15:11	RESERVED	R	0x0		Reserved
10:2	ICHG_REG	R/W	0x190	Reset by: REG_RESET WATCHDOG	Fast Charge Current Regulation Limit with 5mΩ RBAT_SNS: Actual charge current is the lower of ICHG_REG and ICHG pin POR: 20000mA (190h) Range: 400mA-20000mA (8h-190h) Clamped Low Clamped High Bit Step: 50mA

The charging voltage must be written to the Charge_Voltage_Limit 16 bit register at 0x00 address. The register has a range from 1504mV-1566mV, the offset of 1504mV, the bit step of 2mV, and the only the last 5 bits are read by the charger to represent the charging voltage. See the Charge Charge_Voltage_Limit register section of the BQ25750: Standalone/I2C Controlled, 1- to 14-Cell Bidirectional Buck-Boost Battery Charge Controller with Direct Power Path Control data sheet for more information on the details and refer to Section 3.5 for the exact formatting of the charging voltage.

Table 3-3. Charging Voltage Register Settings in BQ25750

Bit	Field	Type	Reset	Notes	Description
15:5	RESERVED	R	0x0		Reserved
4:2	VFB_REG	R/W	0x10	Reset by: REG_RESET	FB voltage regulation limit: POR: 1536mV (10h) Range: 1504mV-1566mV (0h-1Fh) Bit Step: 2mV Offset: 1054mV

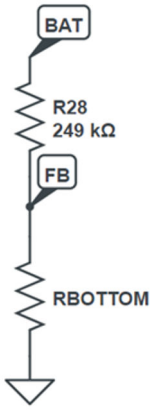

Figure 3-3. Resistor Divider Setting Feedback Voltage

Note

The charge voltage written to the register is the feedback voltage. The battery regulation voltage is the set by feedback voltage divided by the resistor divider.

$$V_{BAT} = FB \times \frac{R_{BOTTOM} + R_{28}}{R_{BOTTOM}} \quad (1)$$

3.3 MCU Setup

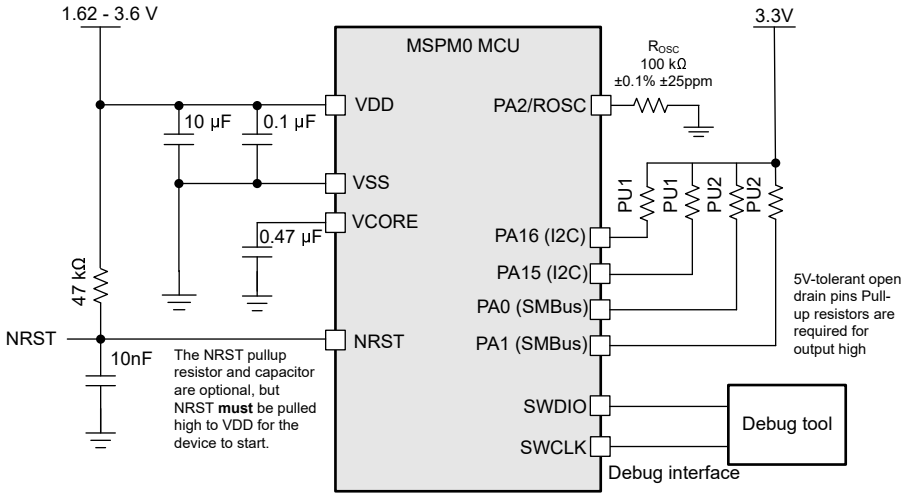

Figure 3-4. Typical Application block diagram of the MSPM0L1306

The block diagram shown above is the typical application of using the MSPM0L1306. For this application note, PA0, PA1, PA15 and P16 were used as I^{2}C and SMBus lines. Those ports have to be configured either through sysconfig file provided by TI or manually defining before compile time. The clock speeds for both were 100kHz.

To verify normal operation of MCU, refer to the MSPM0L1306 LaunchPad Development Kit (LP-MSPM0L1306) user's guide. TI has an extensive device driver library with multiple examples and use cases for different communication protocols. TI also provides an integrated development environment (IDE) to develop and debug code for the MCU, available through the cloud and on desktops. Applications using TI's MSP MCU can leverage SMBus library and I^{2}C library to interface with multiple devices with only the device addresses.

Once data can be transmitted and received by the MCU through TI's function calls, the MCU has to translate such that the data can be sent through different communication protocols. Code executed on the MCU has to be mindful of the endianness of the data, and scaling of data based on the device properties (reserved bits in registers, bit step, and range). The functionality of the MCU can also be scaled to more than translating between communication protocols with the help of TI's extensive library and products.

3.4 Communication Protocol

Using the register bit definitions outlined in the data sheet of the charger, the charge current and voltage writes can be verified using a logic analyzer or oscilloscope. For the BQ40Z80 and other SMBus devices, the communications in broadcast mode can be Packet Error Checking (PEC) enabled with the SBS configuration [CPE] bit. If both SBS configuration [HPE] and SBS configuration [CPE] are disabled, then the gauge does not transmit a PEC byte during any communication. The current and voltage are transmitted from the gauge in little endian, so the format of transmission is the following when the SBS configuration [CPE] bit is set:

Target address (write) -> SMBus Command-> Least significant byte -> Most significant byte -> PEC Byte.

For the I^{2}C-based system, the package structure is the same, except the PEC byte is not used for I^{2}C-based systems as there is no PEC enable options for the BQ25750 and no command (rather a register address).

Target address (write) -> Register Address-> Least significant byte -> Most significant byte.

I^{2}C requires a register address whereas SMBus requires an SMBus command that implicitly addresses the correct registers. For multiple byte reads and writes, the SMBus requires the byte count to be transmitted (from target for reads and from host for writes) while I^{2}C only requires the register address and data is transmitted or received (dependent on the read or write bit) until the stop condition. If the device has an 8-bit address and uses TI functions to read and write in I^{2}C or SMbus, then a right shift by one needs to be performed. For more differences such as the clock speeds, data hold times, and DC specifications, refer to the SMBus Compatibility with an I2C Device Application Report.

3.5 MCU Code Example

The code for the MCU to poll the gauge for *ChargingVoltage()* is shown below.

```
while(1)
    {
        //Checking for the Charging voltage from the Gauge
        ret = SMBus_controllerReadByteWord(&sSMBController,// SMB struct
                                    TARGET_ADDRESS,     // Target Addr
                                    0x15,//SMB Command for charging voltage
                                    voltage,            // ResponsePtr
                                    2);                 // 2 bytes expected
        RXExpected = true;      // Response expected

         if(ret == SMBUS_RET_OK) //error checking
                {
                    // If the command was sent to target, wait for completion
                   if(SMBus_controllerWaitUntilDone (&sSMBController,
                                                DEMO_TIMEOUT) != SMBUS_RET_OK)
                    {
                        ret = DEMO_TIMEOUT_ERROR;
                //Timeout detected in App but not by SCL line, restart interface
                        SMBus_controllerReset(&sSMBController);
                    }
                    else
                    {
                        ret = sSMBState;
                    }

                    // If we are waiting for a response, check if we got it
                    if((ret == SMBus_State_OK) && (RXExpected == true))
                    {
                        // Get the length of payload for response
                      RespLen = SMBus_getRxPayloadAvailable(&sSMBController);
                        if(RespLen >= 1)
                        {
                         ret = DEMO_NO_ERROR_DATA_AVAIL; // Data Received OK
                        }
                        else
                        {
                       ret = DEMO_RX_ERROR;             // RX Data size Error
                        }
                    }
     }

      sSMBController.status.u8byte = 0x00; // repeated for ChargingCurrent()
```

Figure 3-5. Code Example For Polling ChargingVoltage()

From TI's SMBus Library, a predefined function is called to communicate with the gauge through SMBus. The target address is the device address, and the result of the request is stored in the voltage array. The same sequence is repeated for the *ChargingCurrent()* where the SMBus command is 0x14.

```
//Checking for the Charging current from the Gauge
ret = SMBus_controllerReadByteWord(&sSMBController,// SMB struct
                                   TARGET_ADDRESS,     // Target Addr
                                   0x14,//SMB Command for charging current
                                   voltage,            // ResponsePtr
                                   2);                 // 2 bytes expected
```

Figure 3-6. SMBus function call for ChargingCurrent()

The code to translate *ChargingCurrent ()* is such to be compatible with the charger.

```
//the ChargingCurrent() data is received in little endian
Chr_current [0] = (current[1]<<8) | current[0];
        if( Chr_current [0] < 400 ){

            //send 400mA to charger using I2C
            c =  400 ;  //default is 400mA, when charging current is < 400mA
            uint32_t mask = 0x07FC;
            c = (c<<2) & mask; //ensures correct formatting

}

       else {

            //Convert charge current to 50 mA
            c =  Chr_current[0] / 50 ;  //the bit step is 50mA
            uint32_t mask = 0x07FC;
            c = (c<<2) & mask; //ensures correct formatting
}
```

Figure 3-7. Converting ChargingCurrent() Before Transmission To Charger

From TI's I²C Library, a predefined function is called to communicate with the charger through I²C. The endianness of the data must be considered during transmission. The I2C_TARGET_ADDRESS is the charger address shown below.

```
 //send c to charger using I2C, send it in little endian
/* Data sent to the Target */
uint8_t gTxPacket[I2C_TX_PACKET_SIZE] = {
0x02, (c & 0xFF), ((c>>8) & 0xFF)};//the charge current register address
is 0x02
    /*
* Fill FIFO with data. This example will send a MAX of 8 bytes since it
* doesn't handle the case where FIFO is full
*/
DL_I2C_fillControllerTXFIFO(I2C_INST, &gTxPacket[0], I2C_TX_PACKET_SIZE);

/* Wait for I2C to be Idle */
 while (!(DL_I2C_getControllerStatus(I2C_INST) &
 DL_I2C_CONTROLLER_STATUS_IDLE));

/* Send the packet to the controller.
* This function will send Start + Stop automatically.
*/
DL_I2C_startControllerTransfer(I2C_INST, I2C_TARGET_ADDRESS,
    DL_I2C_CONTROLLER_DIRECTION_TX, I2C_TX_PACKET_SIZE);

/* Poll until the Controller writes all bytes */
while (DL_I2C_getControllerStatus(I2C_INST) &
    DL_I2C_CONTROLLER_STATUS_BUSY_BUS)
    ;

/* Trap if there was an error */
if (DL_I2C_getControllerStatus(I2C_INST) &
    DL_I2C_CONTROLLER_STATUS_ERROR) {
    /* LED will remain high if there is an error */
    __BKPT(0);
}

/* Wait for I2C to be Idle */
while (!(
DL_I2C_getControllerStatus(I2C_INST) &
DL_I2C_CONTROLLER_STATUS_IDLE));

/* Add delay between transfers */
delay_cycles(1000);
```

Figure 3-8. Transmission of Data To The Charger Through I²C

The code to translate *ChargingVoltage()* is such to be compatible with the charger.

```
//the ChargingVoltage() data is received in little endian
Chr_voltage [0] = (voltage[1]<<8) | voltage[0];
//From the application note
float R_div = 0.06285284;
uint32_t FB = ((float) Chr_voltage[0]) * R_div;
if (FB<1504) {
            //set FB to min
            FB = 1504;
            //send FB to charger using I2C
            /* Data sent to the Target */
            uint8_t gTxPacket[I2C_TX_PACKET_SIZE] = {
                    0x00,  (FB & 0xFF),((FB>>8) & 0xFF)}; //the charge voltage register address is 0x0

}
        else {
            uint32_t v = FB;
            v -= 1504; // this is the offset
            v /=2; //the bit step is 2mV
            uint32_t mask = 0x001F;
            v =  v & mask;
            //send v to charger using I2C
            /* Data sent to the Target */
            uint8_t gTxPacket[I2C_TX_PACKET_SIZE] = {
                    0x00, (v & 0xFF), ((v>>8) & 0xFF)}; //the charge voltage register address is 0x0
}
```

Figure 3-9. Converting ChargingVoltage() Before Transmission To Charger

Then, the same TI function shown in Figure 3-8 is called to communicate with the charger.

3.6 Data Collected

All the data collected was using the BQ40Z80 device, which follows the SBS guidelines, BQ25750 device with the I²C protocol, and MSPM0L1306 in between to translate.

The MCU polling the gauge is shown below.

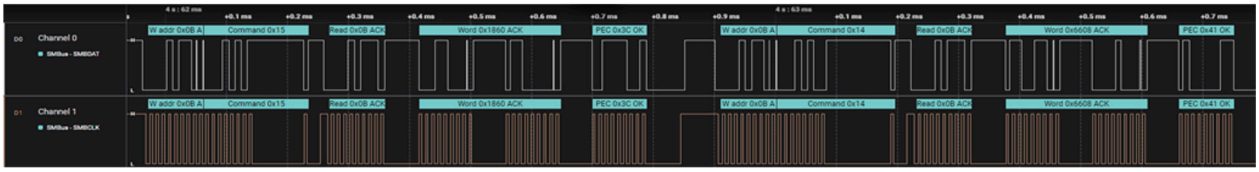

Figure 3-10. Transmission of Gauge Data to MCU through SMBus

The SMBus structure observed follows the structure in Section 3.4. Also note that the data received is little endian. Charging Voltage: 0x6018 = 24600mV and Charging Current: 0x0866 = 2150mA. Note: This is before the gauge settings were changed to what is shown in Figure 3-2.

The MCU sending *ChargingVoltage()* and *ChargingCurrent()* to the charger is shown below.

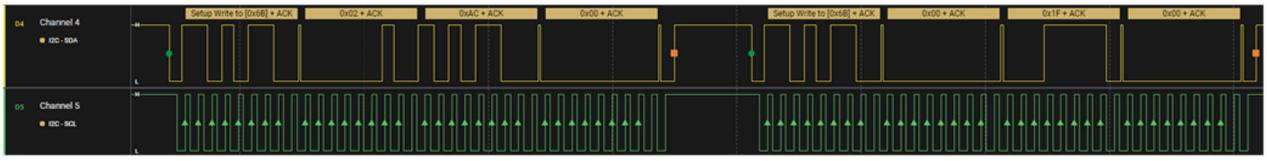

Figure 3-11. Transmission of Charger Data from MCU through I²C

The I²C structure observed follows the structure in Section 3.4. Also note that the data transmitted is little endian. 24600mV *ChargingVoltage()* is scaled to 31 = 0x001F for the charger FB regulation target (following the logic in MCU Code Example) and charging current: 2150mA is scaled to 172 = 0x00AC. The address registers for the voltage and current are 0x00 and 0x02, respectively.

The complete charging profile can be seen below.

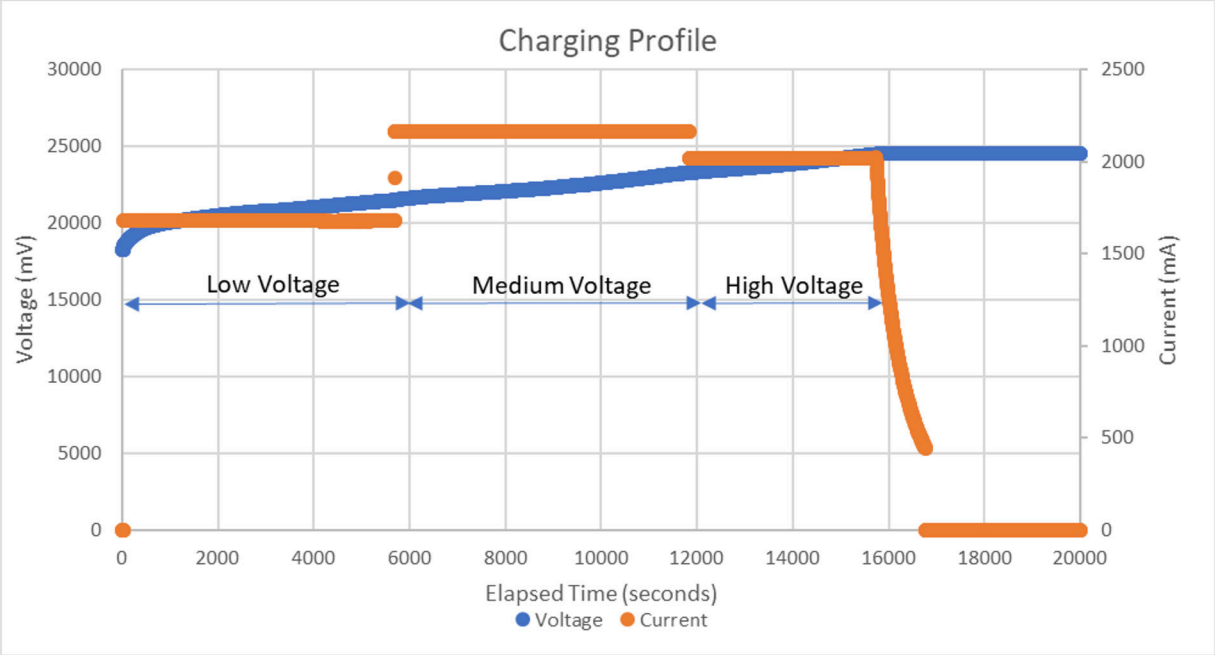

Figure 3-12. Charging Profile Using the MCU

Figure 3-12 illustrates the test result using test set up discussed in the previous section. The result follows the gauge setting detailed in Table 3-1 and charge profile in Figure 3-1. Charge current is first regulated to the *charging current at low voltage* setting. Once the battery voltage reaches the *Medium Voltage Threshold* setting, charge current is increased to *charging current at medium voltage*. Then, current drops to *charging current at high voltage* once the battery voltage is increased to *High Voltage Threshold*. Then, the charger enters constant voltage mode once the battery reaches the *Max Charge Voltage*. In constant voltage mode, the current tapers until the current reaches *Termination current*. The termination in this application is handled by the charger because, once the charger reaches termination current, the charger disables the charge. The termination can by handled by the gauge and MCU if needed, which can be done by disabling the enable termination bit in BQ25750.

Note

The PFM was disabled according data sheet of BQ25750 since the termination current was below 2A and the watchdog was disabled as the PFM was not needed.

4 Summary

The MCU provides flexibility in SBS and allows the user to make the SBS as complex or simple as needed. The MCU allows for flexibility of part selections that others are incompatible together due to differing communication protocols. This application note illustrates the setup of SBS where the MCU polls charging current and charging voltage from the gauge and transmits to the charger, but more alerts and functionality can be added based on user needs. Rather than polling, interrupts must be implemented for efficiency and settings need to be chosen carefully so that the parts work in unison.

5 References

1. SBS Smart Battery System, Smart Battery Charger Specification
2. Texas Instruments, Advanced Charge Algorithm Application Report
3. Texas Instruments, BQ25756 Design Calculator
4. Texas Instruments, BQ25750: Standalone/I2C Controlled, 1- to 14-Cell Bidirectional Buck-Boost Battery Charge Controller with Direct Power Path Control Data Sheet
5. Texas Instruments, MSPM0L1306 LaunchPad Development Kit (LP-MSPM0L1306) User's Guide
6. Texas Instruments, SMBus Compatibility with an I^2C Device Application Report

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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