Application Note

如何使用 MCU 转换 SMBus 或 I²C 信号来为智能电池充电

Jibin Biju

摘要

在创建电池管理系统 (BMS) 时，充电器和电量监测计使用不同的通信协议（例如内部集成电路 (I²C) 和系统管理总线 (SMBus)），因此，需要在充电器和电量监测计之间连接微控制器单元 (MCU)，或者每个器件都需要一个采用不同通信协议的系统。本应用手册说明了在智能电池系统 (SBS) 中使用 MCU 的方法，在此类应用中，MCU 会轮询 SMBus 电量监测计，转换接收到的数据以使其与 I²C 充电器兼容（基于 I²C 规范），并发送到 I²C 充电器。

内容

插图清单

表格清单

商标

1 引言

SBS 规范设立了一组标准，旨在允许系统主机通过 SMBus 接口连接智能电池充电器、智能电池和其他 SMBus 器件。SBS 具有一组功能，例如错误检测、错误信令和系统管理规范中的预定义命令。这些预定义的命令是标准化函数，用于读取或写入一个 2 字节字，具有最低精度、范围、粒度和无效检测指示。必须满足这些特性和其他规范才算符合 SBS。

对于本应用手册，相关的 SBS 规范是充电电压和充电电流寄存器。电量监测计需要分别向 0x15 和 0x14 命令发送 *ChargingVoltage()* 和 *ChargingCurrent()*。SBS 规范规定，在电压调节中，*ChargingVoltage()* 数据范围为 0mV 至 65.534mV，同时连接了良好的电源。*ChargingCurrent()* 的数据范围为 0mA 至 65.534mA。有关例外的更多信息，请参阅 SBS 规范的 *ChargingCurrent()* (0x14) 部分和 *ChargingVoltage()* (0x15) 部分。

2 MCU 在智能电池系统中的作用

MCU 可以配置充电器的高级功能，从而扩展 SBS 功能。MCU 可以在元件的不同通信协议之间转换，因此不管各个器件的通信兼容性如何，不同的 SBS 配置都可以正常工作。借助 MCU，用户还可以实现充电器和智能电池包固有充电算法之外的充电算法。对于大多数 BMS 系统，将 MCU 用作控制器可提高易用性、灵活性和控制能力，并有助于故障排除。

适用于智能电池 BMS 系统的 MCU 从电量监测计接收信息并将该数据传输到充电器，或在传输到充电器之前根据系统需要执行数据修改。在 SBS 中有一个广播模式，在该模式下，电量监测计可以在没有主机的情况下向充电器传输数据。在广播模式下，电量监测计向充电器发送 *ChargingVoltage()*、*CharingCurrent()* 和 *AlarmWarning()*，但如果充电器和电量监测计之间的通信协议存在差异，那么这种方法并不可行。因此，需要根据通信协议和充电器属性来转换电量监测计数据。在本应用手册中，MCU 轮询电量检测计以读取 *ChargingVoltage()* 和 *ChargingCurrent()*。在从 SMBus 转换为 I²C 之前，由 MCU 执行电量监测计数据的转换。最后，MCU 传输数据并通过 I²C 对充电器进行编程。

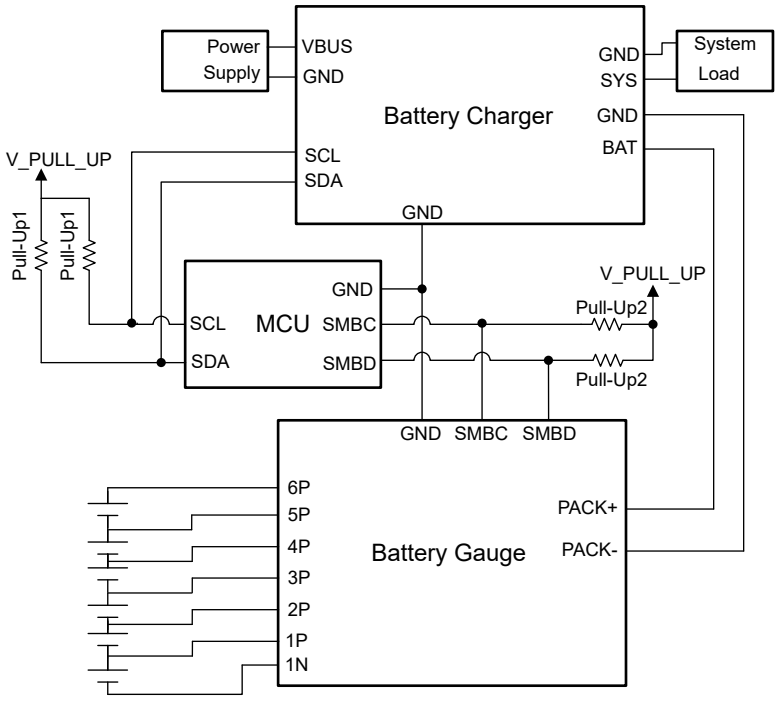

图 2-1. 传统电池管理系统配置

3 使用 BQ25750、BQ40z80 和 MSPM0L1306 的应用示例

根据数据表来设置电量监测计和充电器。更改了充电电流、充电电压和温度范围等电量监测计参数。唯一更改的充电器参数是反馈电压（基于 MCU 算法）。

3.1 电量监测计设置

电量监测计设置为使用先进的充电算法且充电曲线类似于图 3-1。

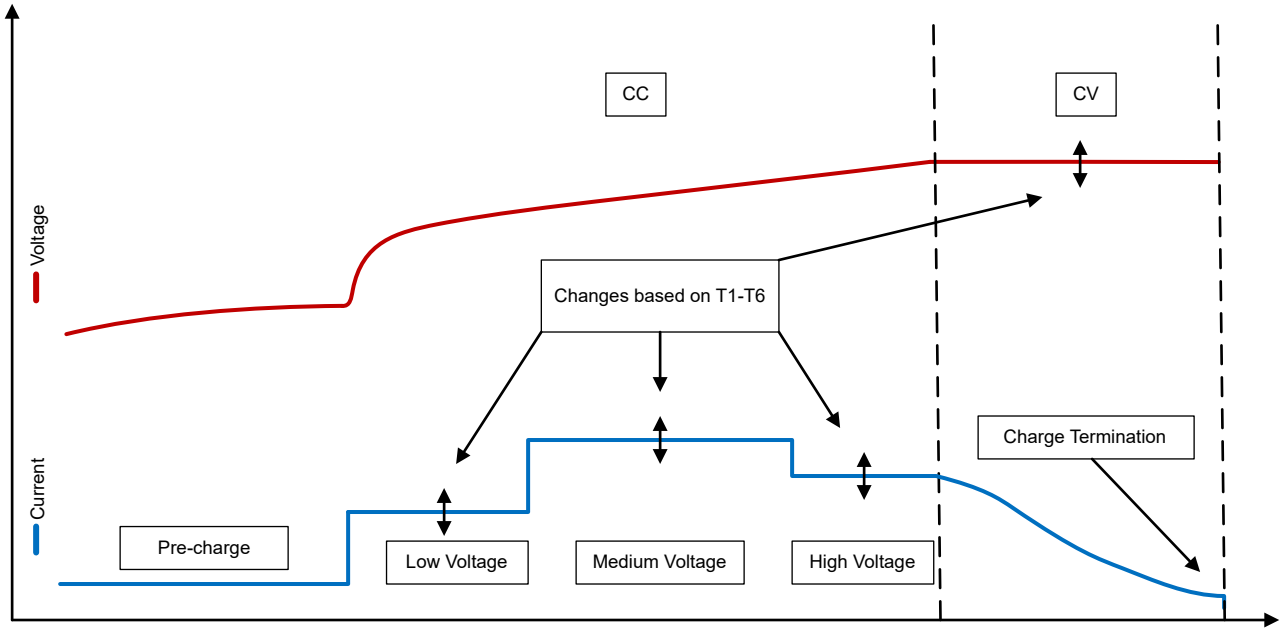

图 3-1. 使用高级充电算法的充电曲线

可以根据高级充电算法应用报告来修改恒流 (CC) 模式、恒压 (CV) 模式和预充电模式下每个参数的阈值。为该示例选择这些参数的目的是为了使充电曲线中的转变更加清晰。电量监测计连接到 6S4P 锂离子电池。

尽管 BQ40z80 提供了更多选项，但这些选项都是与应用手册相关的简化设置，并用于验证充电曲线。这些设置适用于电池包中的每个电芯；对于所编程的串联电芯数量还有另一个设置。

表 3-1. 简化的电量监测计设置

标准高温充电	值	单位
最大充电电压	4100	mV
低压阈值	2900	mV
中压阈值	3600	mV
高压阈值	3900	mV
低压下的电流	1750	mA
中压下的电流	2250	mA
高压下的电流	2100	mA
终止电流	300	mA

3.2 充电器设置

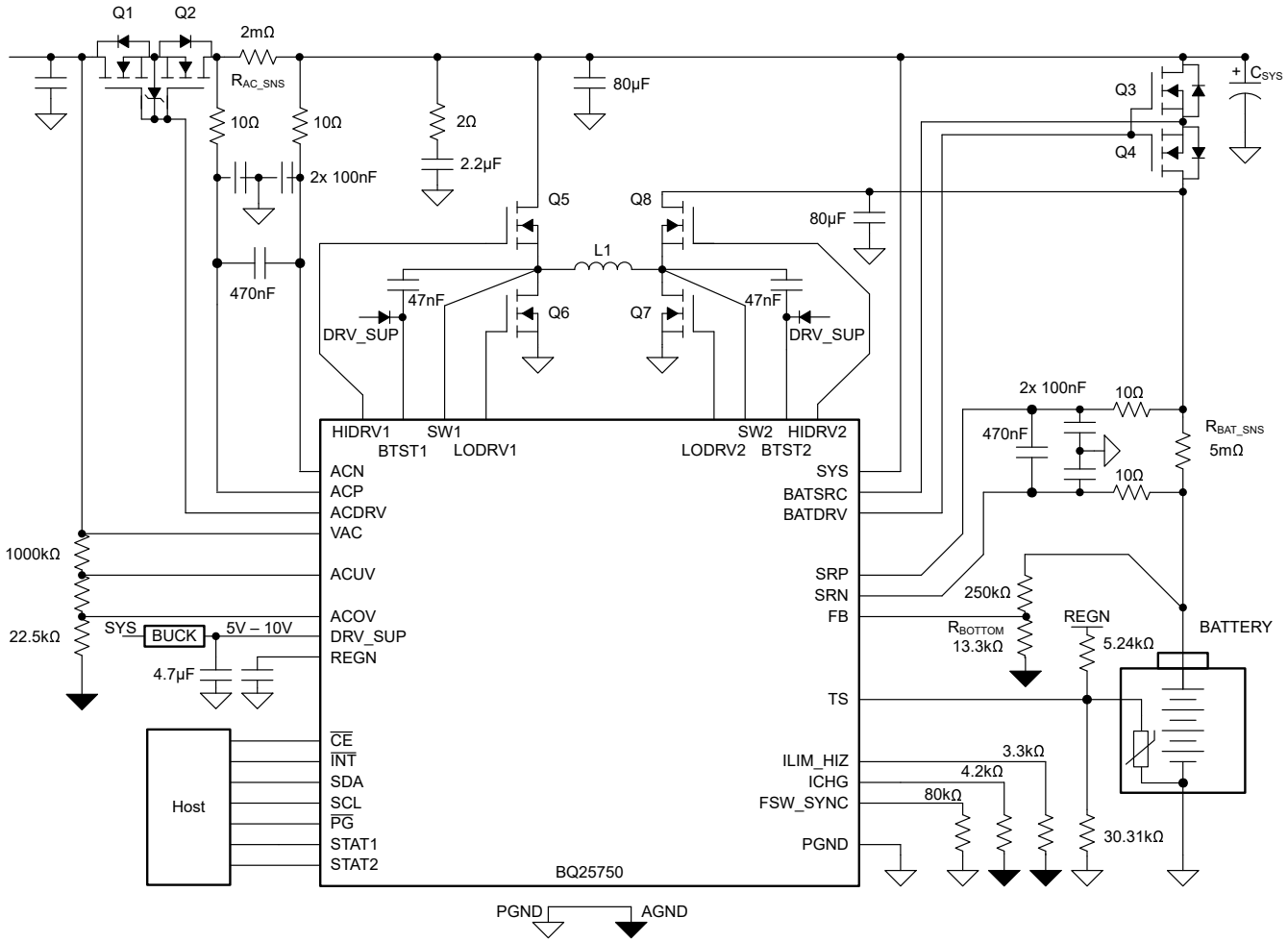

图 3-2. BQ25750 的典型应用方框图

TI 提供品类齐全的充电器产品系列，这些充电器仅支持 I²C 或仅支持 SMBus。表 3-2 是使用 BQ25750（这是基于 I²C 的充电器）的典型应用。对于本应用手册，与上述方框图相比唯一更改的配置是分压器的底部电阻器 (R_{Bottom})，该电阻器用于确定反馈电压。然后对反馈电压进行调节以确定所需的电池稳压电压。选择的电阻分压器必须使分压器涵盖电量监测计所要求的不同充电电压范围。在 BQ25750 中，电阻分压器首先设置所需的目标电池电压，然后可以更改反馈电压（通过寄存器 0x00）。

对于此示例，电量监测计的充电电压范围是每个电芯在不同温度下电压范围的 6 倍。根据具体的温度范围，低压、中压、高压和最大充电电压会发生变化（由用户设置）。根据表 3-1 中的值，用户可以确定所需的最大充电电压为 6×4.1V。在确定电量监测计的充电电压范围后，按照 BQ25756 设计计算器，为 R_{Bottom} 选择 16.7kΩ 电阻器来涵盖 23.97V 至 24.96V 的电压（3.995/节至 4.16/节）。如果电量监测计要求的范围低于最小值，则 MCU 会设置为最低反馈电压。有关 BQ25750 或其他充电器设置和配置的更多信息，请参阅数据表和用户指南。

充电电流必须写入 0x02 地址的 Charge_Current_Limit 16 位寄存器。该寄存器的范围为 400mA 至 20000mA，位步长为 50mA，而充电器仅读取 16 位中的 9 位来表示充电电流（只有位 10:2，其余位保留）。有关详细信息，请参阅 BQ25750：具有直接电源路径控制功能的独立/I2C 控制、1 至 14 节电芯双向降压/升压电池充电控制器数据表的 Charge_Current_Limit 寄存器部分，有关充电电流的精确设置，请参阅节 3.5。

表 3-2. BQ25750 中的充电电流寄存器设置

位	字段	类型	复位	注释	说明
15:11	保留	R	0x0		保留

表 3-2. BQ25750 中的充电电流寄存器设置（续）

位	字段	类型	复位	注释	说明
10:2	ICHG_REG	R/W	0x190	复位方式： REG_RESET 看门狗	使用 5mΩ 时的快速充电电流调节限制 RBAT_SNS： 实际充电电流是 ICHG_REG 和 ICHG 引脚中的较低者 POR：20000mA (190h) 范围: 400mA 至 20000mA (8h-190h) 钳位至低电平 钳位至高电平 位步长：50mA

充电电压必须写入 0x00 地址的 Charge_Voltage_Limit 16 位寄存器。该寄存器的范围为 1504mV 至 1566mV，失调电压为 1504mV，位步长为 2mV，充电器仅读取最后 5 位来表示充电电压。有关详细信息，请参阅 BQ25750：具有直接电源路径控制功能的独立/I2C 控制、1 至 14 节电芯双向降压/升压电池充电控制器数据表的 Charge_Voltage_Limit 寄存器部分，有关充电电压的精确设置，请参阅节 3.5。

表 3-3. BQ25750 中的充电电压寄存器设置

位	字段	类型	复位	注释	说明
15:5	保留	R	0x0		保留
4:2	VFB_REG	R/W	0x10	复位方式： REG_RESET	FB 电压调节限制： POR：1536mV (10h) 范围: 1504mV 至 1566mV (0h-1Fh) 位步长：2mV 失调电压：1054mV

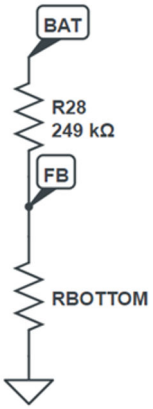

图 3-3. 电阻分压器设置反馈电压

备注

写入寄存器的充电电压是反馈电压。通过反馈电压除以电阻分压器来设置电池稳压电压。

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

3.3 MCU 设置

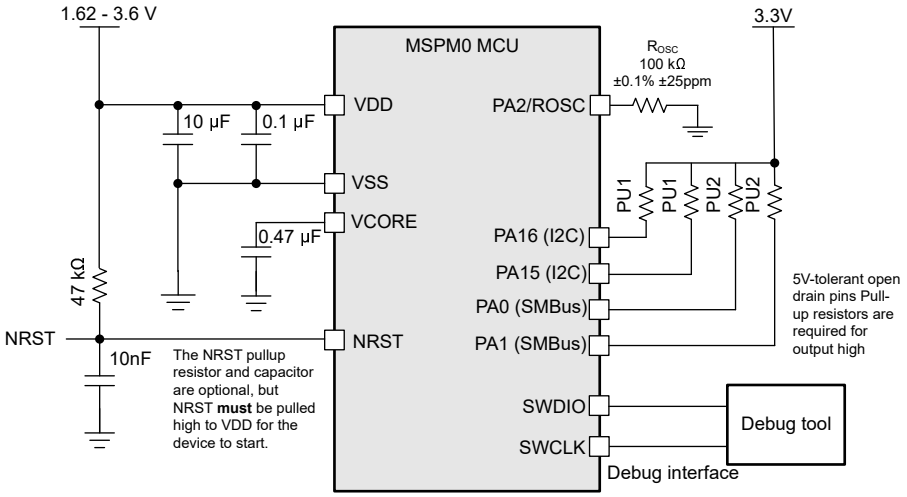

图 3-4. MSPM0L1306 的典型应用方框图

上面显示的方框图是使用 MSPM0L1306 的典型应用。在本应用手册中，PA0、PA1、PA15 和 P16 用作 I²C 和 SMBus 线路。这些端口必须通过 TI 提供的 sysconfig 文件进行配置，或者在编译前进行手动定义。两者的时钟速度均为 100kHz。

要验证 MCU 是否正常运行，请参阅 MSPM0L1306 LaunchPad 开发套件 (LP-MSPM0L1306) 用户指南。TI 拥有丰富的器件驱动程序库，其中包含不同通信协议的多个示例和用例。TI 还提供一个集成开发环境 (IDE)，用于为 MCU 开发和调试代码，可在云端或本地环境中使用。使用 TI 的 MSP MCU 的应用可利用 SMBus 库和 I²C 库与多个器件连接，只需要知道器件地址即可。

在 MCU 可以通过 TI 的函数调用发送和接收数据之后，MCU 就必须进行转换，以便可以通过不同的通信协议发送数据。在 MCU 上执行的代码必须注意数据的字节序，并根据器件属性（寄存器中的保留位、位步长和范围）对数据进行缩放。借助 TI 丰富的库和产品，还可以进一步扩展 MCU 的功能，而不仅仅用于在通信协议之间进行转换。

3.4 通信协议

按照充电器数据表中概述的寄存器位定义，可以使用逻辑分析仪或示波器来验证充电电流和电压写入。对于 BQ40Z80 和其他 SMBus 器件，广播模式下的通信可以通过 SBS 配置 [CPE] 位来启用数据包错误检查 (PEC)。如果 SBS 配置 [HPE] 和 SBS 配置 [CPE] 均禁用，那么电量监测计在任何通信期间都不会发送 PEC 字节。电量监测计以小端字节序格式传输电流和电压，因此在设置 SBS 配置 [CPE] 位时，传输格式如下：

目标地址（写入）-> SMBus 命令 -> 最低有效字节 -> 最高有效字节 -> PEC 字节。

对于基于 I²C 的系统，封装结构相同，只是因为 BQ25750 没有启用 PEC 的选项，也没有相关命令（而是通过寄存器地址来控制），所以基于 I²C 的系统不使用 PEC 字节。

目标地址（写入）-> 寄存器地址 -> 最低有效字节 -> 最高有效字节。

I²C 需要一个寄存器地址，而 SMBus 需要一个对正确寄存器进行隐式寻址的 SMBus 命令。对于多字节读取和写入，SMBus 需要发送字节计数（从目标进行读取，从主机进行写入），而 I²C 只需要发送或接收（取决于读取或写入位）寄存器地址和数据，直到达到停止条件。如果器件具有一个 8 位地址并使用 TI 函数在 I²C 或 SMBus 中进行读写，则需要右移一位。有关诸如时钟速度、数据保持时间和直流规格等更多差异信息，请参阅 SMBus 与 I2C 器件的兼容性应用报告。

3.5 MCU 代码示例

MCU 向电量监测计轮询 *ChargingVoltage()* 的代码如下所示。

```
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()
```

图 3-5. 轮询 ChargingVoltage() 的代码示例

从 TI 的 SMBus 库中调用了预定义函数，以便通过 SMBus 与电量监测计进行通信。目标地址是器件地址，请求的结果存储在电压数组中。对 *ChargingCurrent()* 重复相同的序列，其中 SMBus 命令为 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
```

图 3-6. ChargingCurrent() 的 SMBus 函数调用

为了与充电器兼容，转换 *ChargingCurrent()* 的代码如下。

```
//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
}
```

图 3-7. 在传输到充电器之前转换 ChargingCurrent()

从 TI 的 I²C 库中调用了预定义函数，以便通过 I²C 与充电器进行通信。在传输期间必须考虑数据的字节序。I2C_TARGET_ADDRESS 是充电器地址，如下所示。

```
 //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);
```

图 3-8. 通过 I²C 将数据传输到充电器

为了与充电器兼容，转换 *ChargingVoltage()* 的代码如下。

```
//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
}
```

图 3-9. 在传输到充电器之前转换 ChargingVoltage()

然后，调用图 3-8 中所示的同一 TI 函数以与充电器通信。

3.6 收集的数据

所有收集的数据均使用 BQ40Z80 器件（遵循 SBS 指引）、采用 I²C 协议的 BQ25750 器件和中间的 MSPM0L1306 进行转换。

MCU 轮询电量监测计如下所示。

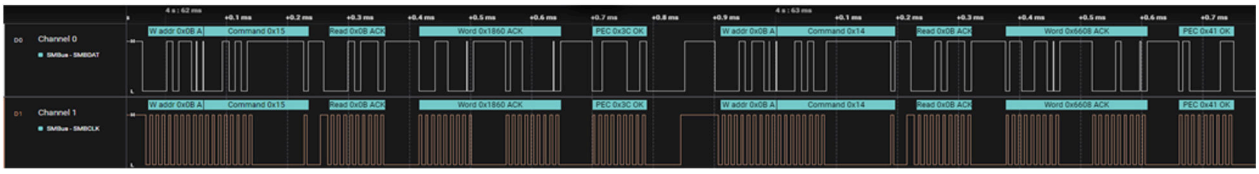

图 3-10. 通过 SMBus 将电量监测计数据传输到 MCU

观察到的 SMBus 结构遵循节 3.4 中的结构。另请注意，接收到的数据采用小端字节序格式。充电电压：0x6018 = 24600mV，充电电流：0x0866 = 2150mA。注意：这是在电量监测计设置更改为图 3-2 中所示值之前的情况。

下面显示了 MCU 向充电器发送 *ChargingVoltage()* 和 *ChargingCurrent()*。

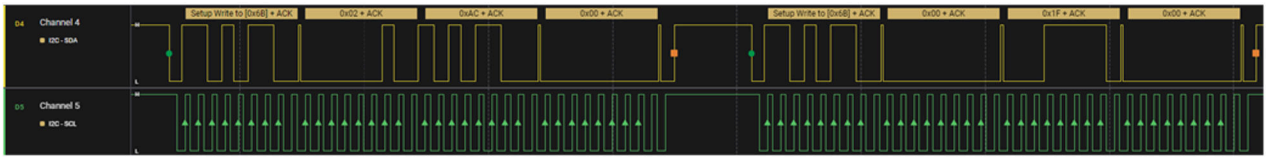

图 3-11. 通过 I²C 从 MCU 传输充电器数据

观察到的 I²C 结构遵循节 3.4 中的结构。另请注意，发送的数据采用小端字节序格式。对于充电器 FB 调节目标，24600mV *ChargingVoltage()* 调整为 31 = 0x001F（遵循 MCU 代码示例 中的逻辑），充电电流：2150mA 调整为 172 = 0x00AC。电压和电流的地址寄存器分别为 0x00 和 0x02。

完整的充电曲线如下所示。

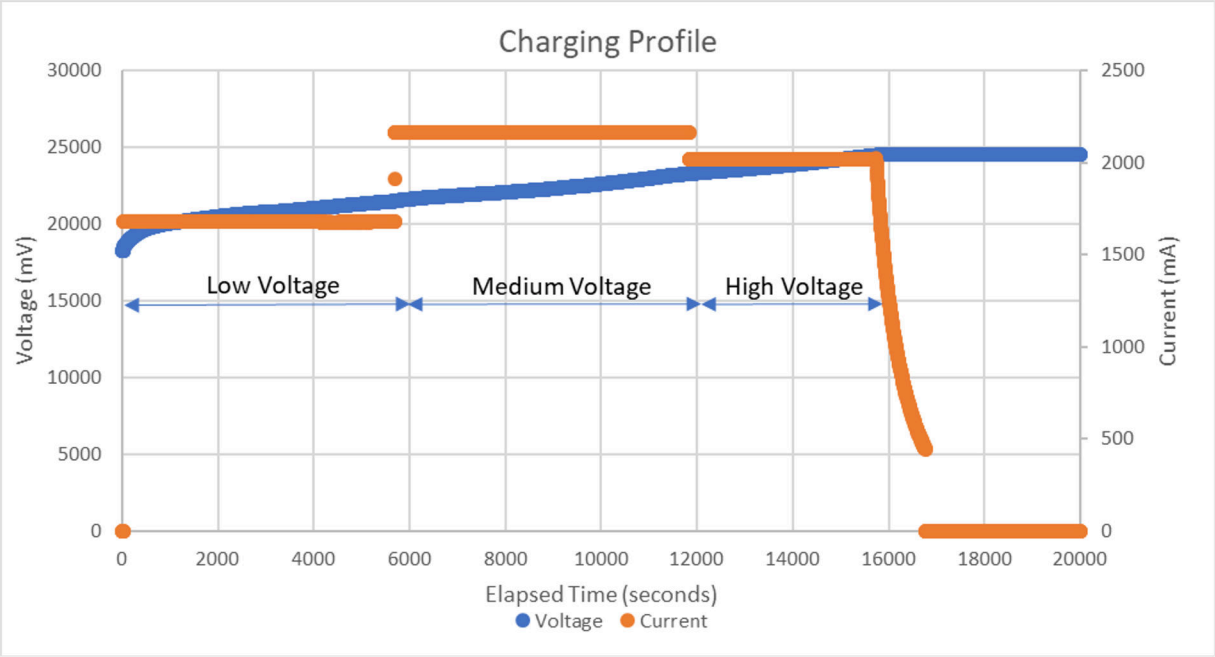

图 3-12. 使用 MCU 的充电曲线

图 3-12 展示了使用上一节讨论的测试设置获得的测试结果。根据表 3-1 中详细介绍的电量监测计设置和图 3-1 中的充电曲线来确定结果。先根据*低压下的充电电流* 设置来调节充电电流。在电池电压达到*中压阈值* 设置后，充电电流将增大到*中压下的充电电流*。然后，在电池电压增大到*高压阈值* 后，电流会下降至*高压下的充电电流*。然后，在电池达到*最大充电电压* 后，充电器进入恒压模式。在恒压模式下，电流逐渐减小，直至电流达到*终止电流*。因为在充电器达到终止电流后，充电器就会禁用充电，所以此应用中的终止由充电器处理。如果需要，可以通过电量监测计和 MCU 来处理终止，这可以通过禁用 BQ25750 中的启用终止位来完成。

备注

根据 BQ25750 的数据表，因为终止电流低于 2A，所以禁用了 PFM，并且由于不需要 PFM，也禁用了看门狗。

4 总结

MCU 在 SBS 中实现了灵活性，让用户可以根据需要调整 SBS 配置。MCU 支持灵活选择器件，从而可以与那些由于通信协议不同而不兼容的其他器件一起使用。此应用手册说明了 SBS 的设置，其中，MCU 会轮询电量监测计中的充电电流和充电电压，并将该数据传输到充电器，也可以根据用户需求添加更多警报和功能。为了提高效率，应使用中断而不是轮询，此外，还需要仔细选择设置，以便各器件能够协调一致地工作。

5 参考资料

1. SBS 智能电池系统，智能电池充电器规范
2. 德州仪器 (TI)，高级充电算法应用报告
3. 德州仪器 (TI)，BQ25756 设计计算器
4. 德州仪器 (TI)，BQ25750：具有直接电源路径控制功能的独立/I2C 控制型、1 至 14 节电池、双向降压/升压电池充电控制器数据表
5. 德州仪器 (TI)，MSPM0L1306 LaunchPad 开发套件 (LP-MSPM0L1306) 用户指南
6. 德州仪器 (TI)，SMBus 与 I²C 器件的兼容性应用报告

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265