Application Note

BQ79600-Q1 软件设计参考

Leslie Marquez Arroyo

摘要

本应用手册概述了主机系统与 BQ79600-Q1 桥接器件（连接 BQ79616-Q1 堆叠式电池监测器件）之间的基本通信信息，包括使用 SPI 接口或 UART 接口与主机进行的通信。本文提供了自动寻址和反向寻址等示例，为用户简单演示了器件的基本通信。本文中的信息是对 *BQ79600-Q1 具备自动主机唤醒功能且符合功能安全标准的汽车类 SPI/UART 通信接口* 数据表中通信信息的概述。

本文档中使用的通信示例通过一系列十六进制字节值来表示。实际的器件通信使用标准 UART（通用异步接收器/发送器）格式或 SPI（串行外设接口）格式发送。

内容

商标

1 命令帧

使用命令帧读写寄存器是与 BQ79600-Q1 进行几乎所有基本通信的基础。BQ79600-Q1 上的命令帧与 BQ79616-Q1 上的命令帧具有相同的结构。所有读写命令全部按照命令帧顺序以十六进制格式提供。

1.1 结构

1.1.1 初始化字节

类型	值
单器件读	0x80
单器件写	0x90
栈读	0xA0
栈写	0xB0
广播读	0xC0
广播写	0xD0
广播写反向	0xE0

1.1.2 器件 ID 地址

器件 ID 地址仅用于单个器件读取/写入命令。一个字节用于表示器件 ID 地址，例如：0x02 表示器件地址 0x02。

1.1.3 寄存器地址

用两个字节来表示寄存器地址，例如：0x0306 表示读取/写入寄存器地址 0x306。

1.1.4 数据

对于读命令，使用一个字节。该数字表示要读取的字节数 - 1，最多请求 128 个字节，例如：0x00 表示读取一个字节的数据。

对于写命令，这表示要写入的数据字节，最多可发送 8 个字节的数据，例如：0xA500 表示写入两个字节的数据。

1.1.5 CRC

两个字节用于 CRC，使用 CRC-16-IBM 生成多项式计算得出。

1.2 命令帧模板表

下面提供了单器件读/写、栈读/写和广播读/写的命令帧框架格式模板。有关命令帧位级的详细信息，请参阅 *BQ79600-Q1 具备自动主机唤醒功能且符合功能安全标准的汽车类 SPI/UART 通信接口* 数据表的“数据通信协议”部分。

表 1-1. 单器件读命令帧

	数据	说明
初始化字节	0x80	始终为 0x80
器件 ID 地址	0x01	本例中进行寻址的是器件地址 0x01
寄存器地址	0x0215	从地址 0x215 开始
数据	0x0B	发回 12 个字节的数据（寄存 0x215 至 0x220 的内容）
CRC	0xCAB5	

表 1-2. 单器件写命令帧

	数据	说明
初始化字节	0x93	向单一器件写入 4 个数据字节（0x90 用于 1 个字节的数据）
器件 ID 地址	0x01	本例中进行寻址的是器件地址 0x01
寄存器地址	0x0100	从地址 0x100 开始
数据	0x02B778BC	向寄存器 0x100-0x103 写入 4 个字节
CRC	0x8A4C	

表 1-3. 栈读命令帧

	数据	说明
初始化字节	0xA0	始终为 0xA0
器件 ID 地址	--	栈读过程中不发送地址字节
寄存器地址	0x0215	从地址 0x215 开始
数据	0x0B	从栈中的每个器件发回 12 个字节的数据（0x215 至 0x220 的寄存器内容）
CRC	0xCCB3	

表 1-4. 栈写命令帧

	数据	说明
初始化字节	0xB3	向栈器件写入 4 个字节
器件 ID 地址	--	栈写过程中不发送地址字节
寄存器地址	0x0100	从地址 0x100 开始
数据	0x02B778BC	依次向寄存器 0x100-0x103 和栈中的所有器件写入 4 个字节
CRC	0x0A35	

表 1-5. 广播读命令帧

	数据	说明
初始化字节	0xC0	始终为 0xC0
器件 ID 地址	--	广播模式下不发送地址字节
寄存器地址	0x0215	从地址 0x215 开始
数据	0x0B	发回 12 个字节的数据（0x215 至 0x220 的寄存器内容）。如果寄存器地址不是 BQ79600-Q1 上的有效寄存器地址，那么该器件会在响应帧中原本应该是响应内容的位置附加零。
CRC	0xD2B3	

表 1-6. 广播写命令帧

	数据	说明
初始化字节	0xD3	向所有器件写入 4 个字节
器件 ID 地址	--	广播模式下不发送地址字节
寄存器地址	0x0100	从地址 0x100 开始
数据	0x02B778BC	依次向寄存器 0x100-0x103 和所有器件写入 4 个字节
CRC	0x6A33	

1.3 读取寄存器和写入寄存器函数

使用 BQ79600 UART 示例代码时，使用 *ReadReg* 函数生成读取命令并接收响应，而 *WriteReg* 函数用于生成写入命令。使用 BQ79600 SPI 示例代码时，这些函数的名称分别为 *SpiReadReg* 和 *SpiWriteReg*。这些函数充当 TMS570 LaunchPad 和 BQ79600-Q1 之间的主要通信包装器函数。CRC 由这些函数自动计算并附加。

1.3.1 ReadReg/SpiReadReg

ReadReg 和 *SpiReadReg* 函数的基本结构如下：

```
UART sample code:
#_of_Read_Bytes = ReadReg(Device_Address, Register_Address, Incoming_Data_Byte_Array, #_Data_Bytes,
ms_Before_Time_Out, Packet_Type)
SPI sample code:
#_of_Read_Bytes = SpiReadReg(Device_Address, Register_Address, Incoming_Data_Byte_Array,
#_Data_Bytes, ms_Before_Time_Out, Packet_Type)
```

Device_Address、*#_Data_Bytes*、*ms_Before_Time_Out* 和 *Packet_Type* 是整数。*Register_Address* 是十六进制值（带有前缀“0x”）。*Incoming_Data_Byte_Array* 是 UART 示例代码中的 1 字节十六进制值数组和 SPI 示例代码中 2 字节十六进制值数组。

Device_Address 在广播和栈读命令中会被忽略。

例如：

```
UART sample code:
nRead = ReadReg(nDev_ID, 0x0306, bFrame, 12, 0, FRMWRT_SGL_R);
SPI sample code:
nRead = SpiReadReg(nDev_ID, 0x0306, bFrame, 12, 0, FRMWRT_SGL_R);
```

此行会从器件 *nDev_ID* 的寄存器 0x0306 读取 12 个字节的数据，并将其存储在名为 *bFrame* 的本地字节数组中（在微控制器上）。数据包类型为单器件读。

1.3.2 WriteReg/SPIWriteReg

WriteReg 和 *SpiWriteReg* 函数的基本结构如下：

```
UART sample code:
#_of_Sent_Bytes = WriteReg(Device_Address, Register_Address, Data, #_Data_Bytes, Packet_Type)
SPI sample code:
#_of_Sent_Bytes = SpiWriteReg(Device_Address, Register_Address, Data, #_Data_Bytes, Packet_Type)
```

Device_Address、*#_Data_Bytes* 和 *Packet_Type* 是整数，而 *Register_Address* 和 *Data* 是十六进制值（带前缀“0x”）。*Device_Address* 在广播和栈写操作中会被忽略。

例如：

```
UART sample code:
nSent = WriteReg(nDev_ID, 0x0306, 0x01, 1, FRMWRT_SGL_NR);
SPI sample code:
nSent = SpiWriteReg(nDev_ID, 0x0306, 0x01, 1, FRMWRT_SGL_NR);
```

此行会将 1 字节数据写入器件 *nDev_ID* 的寄存器 0x0306。发送的数据为 0x01。数据包的类型为单器件写。

1.3.3 示例代码中可用的数据包类型

下表提供了可用于 *ReadReg*/*SpiReadReg* 和 *WriteReg*/*SpiWriteReg* 函数的各种数据包类型：

帧能指	数据包类型
FRMWRT_SGL_W	单器件写
FRMWRT_SGL_R	单器件读
FRMWRT_STK_W	栈写
FRMWRT_STK_R	栈读
FRMWRT_ALL_W	广播写
FRMWRT_ALL_R	广播读

2 快速入门指南

若要快速开始测量，只需阅读本指南的“唤醒序列”、“自动寻址”和“读取电池电压”部分。

2.1 唤醒序列

使用 UART 时，微控制器通过将线路拉至低电平并持续 2.75ms 将唤醒 ping 应用于 BQ79600-Q1 器件的 MOSI/RX 引脚。

使用 SPI 时，微控制器必须将 nCS 线路拉低并等待 2µs，将 MOSI/RX 线路拉低并持续 2.75ms，然后将其拉回高电平并等待 2µs，最后将 nCS 引脚再次拉至高电平。

若要唤醒器件：

1. 发送唤醒 ping（如上所述）。
2. 等待至少 3.5ms。
3. 向 BQ79600-Q1 发送单个器件写入以设置 CONTROL1[SEND_WAKE]=1，这将唤醒所有堆叠器件。

```
90 00 03 09 20 13 95       //Step 3 (wake up stacked devices)
```

4. 等待适当的时间以允许所有器件接收 WAKE 音调并进入 ACTIVE 模式。要计算总等待时间，请将 WAKE 音调持续时间（大约 1.6ms）加上进入 ACTIVE 模式的时间（大约 10ms），然后将结果乘以堆叠的 BQ7961X-Q1 器件数量。

备注

如果 BQ79600-Q1 器件通过 SHUTDOWN ping 关闭，则 COMH RX 和 COML RX 将在下次唤醒时被禁用。在这种情况下，在步骤 1 中，主机需要首先发送 WAKE ping，等待至少 3.5ms，然后发送第二个 WAKE ping。COMH RX 和 COML RX 将在第二个 WAKE ping 后启用。然后继续执行步骤 2 至 4。

2.2 自动寻址

下面是标准方向自动寻址过程。

2.2.1 步骤

1. 虚拟栈将寄存器 OTP_ECC_DATAIN1 写入 OTP_ECC_DATAIN8 = 0x00 以同步 DLL（延迟锁相环）。共有 8 个栈写入命令。
2. 广播写入以启用自动寻址模式 (CONTROL1 = 0x01)。
3. 广播连续写入 DIR0_ADDR = 0、1、2、3（寄存器地址 0x306）。
4. 广播写入，用以先将所有器件设置为栈器件 (COMM_CTRL=0x02)。
5. 可向栈中最高器件写入内容的单个器件，用以将其配置为栈和栈顶 (COMM_CTRL=0x03)。
6. 虚拟栈读取寄存器 OTP_ECC_DATAIN1 至 OTP_ECC_DATAIN8 以同步 DLL。共有 8 个栈读命令。

2.2.2 由 3 个器件组成的栈的示例命令

```
B0 03 43 00 E7 D4       //Step 1 (dummy write OTP_ECC_DATAIN1 to sync DLL)
B0 03 44 00 E5 E4       //Step 1 (dummy write OTP_ECC_DATAIN2 to sync DLL)
B0 03 45 00 E4 74       //Step 1 (dummy write OTP_ECC_DATAIN3 to sync DLL)
B0 03 46 00 E4 84       //Step 1 (dummy write OTP_ECC_DATAIN4 to sync DLL)
B0 03 47 00 E5 14       //Step 1 (dummy write OTP_ECC_DATAIN5 to sync DLL)
B0 03 48 00 E0 E4       //Step 1 (dummy write OTP_ECC_DATAIN6 to sync DLL)
B0 03 49 00 E1 74       //Step 1 (dummy write OTP_ECC_DATAIN7 to sync DLL)
B0 03 4A 00 E1 84       //Step 1 (dummy write OTP_ECC_DATAIN8 to sync DLL)
D0 03 09 01 0F 74       //Step 2 (enable auto-addressing mode)
D0 03 06 00 CB 44       //Step 3 (set bridge device address DIR0_ADDR = 0)
D0 03 06 01 0A 84       //Step 3 (set stack 1 device address DIR0_ADDR = 1)
D0 03 06 02 4A 85       //Step 3 (set stack 2 device address DIR0_ADDR = 2)
D0 03 06 03 8B 45       //Step 3 (set stack 3 device address DIR0_ADDR = 3)
D0 03 08 02 4E E5       //Step 4 (set all stacked devices as stack)
90 03 03 08 03 53 98    //Step 5 (set stack 3 as both stack and top of stack)
A0 03 43 00 E3 14       //Step 6 (dummy read OTP_ECC_DATAIN1 to sync DLL)
A0 03 44 00 E1 24       //Step 6 (dummy read OTP_ECC_DATAIN2 to sync DLL)
A0 03 45 00 E0 B4       //Step 6 (dummy read OTP_ECC_DATAIN3 to sync DLL)
A0 03 46 00 E0 44       //Step 6 (dummy read OTP_ECC_DATAIN4 to sync DLL)
A0 03 47 00 E1 D4       //Step 6 (dummy read OTP_ECC_DATAIN5 to sync DLL)
A0 03 48 00 E4 24       //Step 6 (dummy read OTP_ECC_DATAIN6 to sync DLL)
A0 03 49 00 E5 B4       //Step 6 (dummy read OTP_ECC_DATAIN7 to sync DLL)
A0 03 4A 00 E5 44       //Step 6 (dummy read OTP_ECC_DATAIN8 to sync DLL)
```

第一个栈写命令帧 (B0 03 43 00 E7 D4) 说明：

- B0 = 栈写入一个字节
- 0343 = 寄存器地址
- 00 = 写入值 0x00
- E7D4 = CRC

第一个广播写命令帧 (D0 03 09 01 0F 74) 说明：

- D0 = 广播写入一个字节
- 0309 = 寄存器地址
- 01 = 写入值 0x01
- 0F74 = CRC

第一个单器件写命令帧 (90 03 03 08 03 53 98) 说明：

- 90 = 单器件写入一个字节
- 03 = 器件地址
- 0308 = 寄存器地址
- 03 = 写入值 0x03
- 5398 = CRC

第一个栈读命令帧 (A0 03 43 00 E3 14) 说明：

- A0 = 栈读
- 0343 = 寄存器地址
- 00 = 读取一个字节的数据
- E314 = CRC

2.3 读取电芯电压

2.3.1 步骤

1. 将所有已用电芯设置为活动状态。例如，对于 16 节电芯，ACTIVE_CELL=0x0A。
2. 设置所需的运行模式，然后启动 ADC。例如，对于连续运行，ADC_CTRL1=0x06。
3. 等待所需的循环时间（每个循环 192 µs，加上写入 ADC_CTRL1 寄存器产生的任何重新计时延迟）。
4. 循环读取适当的电芯测量寄存器。例如，VCELL16_HI 至 VCELL1_LO。

2.3.2 由 3 个器件组成的栈的示例命令

```
B0 00 03 0A A6 13                    //Step 1 (16 active cells)
B0 03 0D 06 52 76                    //Step 2 (set continuous run and start ADC)
delay [192us + (5us x TOTALBOARDS)] //Step 3 (delay)
A0 05 68 1F 5C 2D                    //Step 4 (read ADC measurements)
```

2.3.3 转换为电压

将 16 位 ADC 值转换为实际电压：

1. 将 16 位值从二进制补码格式转换为 16 位十进制值。
2. 乘以 ADC 分辨率 (190.73μV/LSB)。

2.4 反向寻址

此示例提供有关对整个菊花链进行反向寻址的详细信息。请注意，当双向寻址完成后，主机能够在更换方向时跳过自动寻址。

2.4.1 步骤

1. 向 BQ79600-Q1 发送单个器件写入以设置 CONTROL1[DIR_SEL]=1 (CONTROL1=0x80)。
2. 向 BQ79600-Q1 发送单个器件写入以设置 CONTROL1[SEND_WAKE]=1，而不覆盖 DIR_SEL 位 (CONTROL1=0xA0)。
3. 虚拟栈将寄存器 OTP_ECC_DATAIN1 写入 OTP_ECC_DATAIN8 = 0x00 以同步 DLL（延迟锁相环）。共有 8 个栈写入命令。
4. 发送**广播写反向**命令以更改堆叠器件上的方向 (CONTROL1=0x80)。这种命令类型应仅用于用户更改菊花链通信方向这一场景。请勿将其用于其他命令。
5. 广播写入，用以将所有器件设置为栈器件 (COMM_CTRL=0x02)。如果之前在北方向设置了 TOP_STACK 位，则会清除该位。
6. 广播写入，用以启用自动寻址模式 (CONTROL1 = 0x81)。
7. 广播连续写入 DIR1_ADDR = 0、1、2、3（寄存器地址 0x307）。
8. 广播写入，用以先将所有器件设置为栈器件 (COMM_CTRL=0x02)。
9. 可向顶部器件写入内容的单个器件，用以将其配置为栈和栈顶 (COMM_CTRL=0x03)。
10. 虚拟栈读取寄存器 OTP_ECC_DATAIN1 至 OTP_ECC_DATAIN8 以同步 DLL。共有 8 个栈读命令。

2.4.2 由三个器件组成的栈的示例命令

```
90 00 03 09 80 13 ED   //Step 1 (change BQ79600-Q1 direction)
90 00 03 09 A0 12 35   //Step 2 (wake up stack devices)
B0 03 43 00 E7 D4      //Step 3 (dummy write OTP_ECC_DATAIN1 to sync DLL)
B0 03 44 00 E5 E4      //Step 3 (dummy write OTP_ECC_DATAIN2 to sync DLL)
B0 03 45 00 E4 74      //Step 3 (dummy write OTP_ECC_DATAIN3 to sync DLL)
B0 03 46 00 E4 84      //Step 3 (dummy write OTP_ECC_DATAIN4 to sync DLL)
B0 03 47 00 E5 14      //Step 3 (dummy write OTP_ECC_DATAIN5 to sync DLL)
B0 03 48 00 E0 E4      //Step 3 (dummy write OTP_ECC_DATAIN6 to sync DLL)
B0 03 49 00 E1 74      //Step 3 (dummy write OTP_ECC_DATAIN7 to sync DLL)
B0 03 4A 00 E1 84      //Step 3 (dummy write OTP_ECC_DATAIN8 to sync DLL)
E0 03 09 80 C0 14      //Step 4 (broadcast write reverse command)
D0 03 08 02 4E E5      //Step 5 (set all devices as stack)
D0 03 09 81 0E D4      //Step 6 (enable auto-addressing mode)
D0 03 07 00 CA D4      //Step 7 (set bridge device address DIR1_ADDR = 0)
D0 03 07 01 0B 14      //Step 7 (set stack 1 device address DIR1_ADDR = 1)
D0 03 07 02 4B 15      //Step 7 (set stack 2 device address DIR1_ADDR = 2)
D0 03 07 03 8A D5      //Step 7 (set stack 3 device address DIR1_ADDR = 3)
D0 03 08 02 4E E5      //Step 8 (set all devices as stack)
90 03 03 08 03 53 98   //Step 9 (set stack 3 as both stack and top of stack)
A0 03 43 00 E3 14      //Step 10 (dummy read OTP_ECC_DATAIN1 to sync DLL)
A0 03 44 00 E1 24      //Step 10 (dummy read OTP_ECC_DATAIN2 to sync DLL)
A0 03 45 00 E0 B4      //Step 10 (dummy read OTP_ECC_DATAIN3 to sync DLL)
A0 03 46 00 E0 44      //Step 10 (dummy read OTP_ECC_DATAIN4 to sync DLL)
A0 03 47 00 E1 D4      //Step 10 (dummy read OTP_ECC_DATAIN5 to sync DLL)
A0 03 48 00 E4 24      //Step 10 (dummy read OTP_ECC_DATAIN6 to sync DLL)
A0 03 49 00 E5 B4      //Step 10 (dummy read OTP_ECC_DATAIN7 to sync DLL)
A0 03 4A 00 E5 44      //Step 10 (dummy read OTP_ECC_DATAIN8 to sync DLL)
```

3 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision * (August 2020) to Revision A (October 2023)	Page
• 更新了整个文档中的表格、图和交叉参考的编号格式	1
• 第一版。	2

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