Application Report
SLVA970E–March 2018–Revised August 2019

BQ79606A-Q1 Software Design Reference

Vince Toledo

ABSTRACT

This application note provides an outline for the basic communications between the BQ79606A-Q1 device and a host system. This includes communications for a single BQ79606A-Q1 device or a stack of BQ79606A-Q1 devices. Examples, such as auto-addressing and reverse-addressing, are included to provide the user with simple demonstrations of the basic communications of the device. The information is meant to provide an overview of the communications information outlined in the *BQ79606A-Q1 SafeTI™ Precision Monitor With Integrated Hardware Protector for Automotive Battery Pack Applications* data sheet.

The communications used in this document are presented in a series of hexadecimal byte values. The actual device communications are sent in standard UART (universal asynchronous receiver-transmitter) format.

Contents

1 Reading and Writing Registers 2
2 Shutdown and Wakeup Sequence 4
3 Auto-Addressing 4
4 Initializing Devices 8
5 Read Cell Voltages 10
6 Enable Cell Balancing 10
7 Reverse Device Addressing and Communications 11

List of Figures

List of Tables

1 Single Device Read Command Frame 2
2 Single Device Write Command Frame 2
3 Stack Read Command Frame 2
4 Stack Write Command Frame 2
5 Broadcast Read Command Frame 3
6 Broadcast Write Command Frame 3

Trademarks

1 Reading and Writing Registers

Reading and writing registers underlies nearly all basic communication with the BQ79606A-Q1. All read and write commands will be provided in hexadecimal format, in command frame order.

1.1 Command Frame Template Tables

Command frame format templates are provided in the following tables for single device read/write, stack read/write, and broadcast read/write. For bit-level detail on the command frames, see the "Command and Response Protocol Layer" section of the *BQ79606A-Q1 SafeTI™ Precision Monitor With Integrated Hardware Protector for Automotive Battery Pack Applications* data sheet.

Table 1. Single Device Read Command Frame

	DATA	COMMENTS
Initialization Byte	0x80	Always 0x80
Device ID Address	0x00	Device address 0 is addressed in this case.
Register Address	0x0215	Start with address 0x215.
Data	0x0B	Send 12 bytes worth of data back (register contents from 0x215 to 0x220). 128 bytes max
CRC	0xCB49	

Table 2. Single Device Write Command Frame

	DATA	COMMENTS
Initialization Byte	0x93	Writing four data bytes to a single device (0x90 for one bytewrite)
Device ID Address	0x00	Device address 0 is addressed in this case.
Register Address	0x0100	Start with address 0x100.
Data	0x02B778BC	Write four bytes to registers 0x100-0x103. 8 bytes max
CRC	0x9A8C	

Table 3. Stack Read Command Frame

	DATA	COMMENTS
Initialization Byte	0xA0	Always 0xA0
Device ID Address	--	No address byte is sent in stack read.
Register Address	0x0215	Start with address 0x215.
Data	0x02B778BC	Send 12 bytes worth of data back (register conents from 0x215 to 0x220) from each device in the stack. 128 bytes max per device
CRC	0xCCB3	

Table 4. Stack Write Command Frame

	DATA	COMMENTS
Initialization Bte	0xB3	Writing four bytes of the stack devices
Device ID Address	--	No address byte is sent in stack write.
Register Address	0x0100	Start with address 0x100.
Data	0x02B778BC	Write four bytes to registers 0x100-0x103 to all devices in stack. 8 bytes max per device

Table 4. Stack Write Command Frame (continued)

	DATA	COMMENTS
CRC	0x0A35	

Table 5. Broadcast Read Command Frame

	DATA	COMMENTS
Initialization Byte	0xC0	Always 0xC0
Device ID Address	--	No address byte is sent in broadcast mode.
Register Address	0x0215	Star with address 0x215.
Data	0x0B	Send 12 bytes worth of data back (register contents from 0x215 to 0x220). 128 bytes max per device
CRC	0xD2B3	

Table 6. Broadcast Write Command Frame

	DATA	COMMENTS
Initialization Bte	0xD3	Writing four bytes to the all devices
Device ID Address	--	No address byte is sent in broadcast mode.
Register Address	0x0100	Start with address 0x100.
Data	0x02B778BC	Write four bytes to registers 0x100-0x103 to all devices. 8 bytes max per device
CRC	0x336A	

1.2 ReadReg and WriteReg Functions

When using the BQ79606A-Q1 sample code, ReadReg and WriteReg act as the primary communication wrapper functions between the TMS570 LaunchPad and BQ79606A-Q1.

1.2.1 ReadReg

The basic structure for the ReadReg function is as follows:

```
#_of_Read_Bytes = ReadReg(Device_Address, Register_Address, Incoming_Data_Byte_Array,
#_Data_Bytes, ms_Before_Time_Out, Packet_Type)
```

Device_Address, #_Data_Bytes, and ms_Before_Time_Out are integers while Incoming_Data_Byte_Array and Register_Address are hex values (with the prefix "0x"). For example:

```
nRead = ReadReg(nDev_ID, 0x0207, bFrame, 12, 0, FRMWRT_SGL_R);
```

This line reads 12 bytes of data from register 0x0207 of the device nDev_ID and stores it in a local byte array (on the microcontroller) called bFrame. The packet type is a single device read.

1.2.2 WriteReg

The basic structure for the WriteReg function is as follows:

```
#_of_Sent_Bytes = WriteReg(Device_Address, Register_Address, Data, #_Data_Bytes, Packet_Type)
```

Device_Address and #_Data_Bytes are integers, while Register_Address and Data are hex values (with the prefix "0x"). For example:

```
nSent = WriteReg(nDev_ID, 0x0106, 0x01, 1, FRMWRT_SGL_NR);
```

This line writes to register 0x0106 of the device nDev_ID, with one byte of data. The data sent is 0x01. The type of packet is a single device write.

1.2.3 Packet Types Available in Sample Code

The following table provides the various packet types available for use in the ReadReg and WriteReg functions:

FRAME SIGNIFIER	PACKET TYPE
FRMWRT_SGL_NR	Single Device Write
FRMWRT_SGL_R	Single Device Read
FRMWRT_STK_NR	Stack Write
FRMWRT_STK_R	Stack Read
FRMWRT_ALL_NR	Broadcast Write
FRMWRT_ALL_R	Broadcast Read
FRMWRT_REV_ALL_NR	Broadcast Write Reverse Direction

2 Shutdown and Wakeup Sequence

Before communicating, the user must first ensure that all devices are in the correct power state. To do so, the wake ping is sent to the base device which propagates through the daisy-chain to the rest of the stack. All devices are brought to the same state. Clearing faults after this is recommended.

To send a wake pulse, assert wake (active low) on the WAKEUP pin, wait 250 µs, then de-assert wake.

This is done with the following in the LaunchPad sample code:

```
gioSetBit(gioPORTA, 0, 0); // assert wake (active low)
delayus(250); //250us to 300us
gioToggleBit(gioPORTA, 0); // deassert wake
```

From shutdown, ensure there is $t_{SU(WAKE)}$ = 7 ms of delay per board before beginning communications.

The broadcast shutdown command is called as follows:

```
D0 01 05 08 6B B2
```

- D0 = Broadcast write of 1 byte
- 0105 = Write to register address 0x105
- 08 = Data byte to write
- 6BB2 = CRC

This can be done in the LaunchPad sample code by broadcasting the following:

```
WriteReg(0, CONTROL1, 0x08, 1 FRMWRT_ALL_NR);
```

3 Auto-Addressing

Preparing the BQ79606A-Q1 device (or device stack) for communication requires wakeup (see previous section), auto-addressing (this section), and initialization of each device (next section).

3.1 TMS570 Code Setup

When using the TMS570 sample code, update the number of boards in the bq79606.h file (in the "include" folder). Update the following line with the appropriate number of BQ79606A-Q1 devices used in your application:

```
#define TOTALBOARDS 3
```

This line must be immediately after the "#include" lines in the bq79606.h file. This variable are used throughout the TMS570 sample code (in sys_main.c and bq79606.c).

3.2 Baudrate Setup

Note: If the host microcontroller and all BQ79606A-Q1 devices are already set to 1M baudrate, this section is not required (but can be useful in debugging communications issues).

Before beginning device communications, it is important to ensure proper baudrate is set for the host microcontroller and every device in the stack. To do this, first set the host baudrate to 250 k. For the TMS570, this is done by the following line (this baudrate change is handled by the CommReset() function):

```
sciSetBaudrate(scilinREG, 250000); //set microcontroller baudrate to 250k
```

Next, apply a COMM_RESET pulse to the base BQ79606A-Q1 device. This is done by sending a 500 µs low pulse to the 606-RX line. The base BQ79606A-Q1 device is now 250 k baudrate. All other devices are still at their default baudrate (1M). For the TMS570, this RX pulse is sent with the following command (this function handles all other requirements for changing the baudrate):

```
CommReset(); // send COMM_RESET pulse to base device
```

Now that the base device is guaranteed to be at the same rate as the microcontroller, you can now send a command to all of the BQ79606A-Q1 devices to change all baudrates. This can be whatever baudrate is desired. You can also enable all interfaces for the BQ79606A-Q1 during this write by writing two bytes instead of one. The first byte sets the baudrate, the second byte enables all interfaces:

- 0x303C (baudrate = 125000)
- 0x343C (baudrate = 250000)
- 0x383C (baudrate = 500000)
- 0x3C3C (baudrate = 1000000)

The command frame to set the baudrate to 1M and enable all interfaces for all BQ79606A-Q1 devices is as follows:

```
D1 00 20 3C 3C C8 C9 //set baud rate to 1000000, enable interfaces
```

- D1 = Broadcast write of two bytes
- 0020 = Write beginning with register address 0x0020 (COMM_CTRL and DAISY_CHAIN_CTRL registers)
- 3C3C = Data bytes to write (set baud rate to 1000000 and enable UART transmitter, NFAULT function, COML/H receivers, and COML/H transmitters)
- C8C9 = CRC

For the TMS570, this is done with the following line of code (within the CommReset() function):

```
WriteReg(0, COMM_CTRL, 0x3C3C, 2, FRMWRT_ALL_NR);
```

Lastly, set the baudrate of the microcontroller to the desired baudrate to continue communications (Note: If you do not set the baudrate of the microcontroller to the same baudrate that you just gave the BQ79606A-Q1 devices, communications fails). For the TMS570, the baudrate can be set by the following (handled within the CommReset() function):

```
sciSetBaudrate(scilinREG, 1000000); //set TMS570 baudrate to 1M
```

3.3 Auto-Addressing Sequence

To auto-address the devices, users must first dummy write to the ECC_TEST register to sync the DLL (delay-locked loop):

```
D0 01 1D 00 60 74
```

- D0 = Broadcast write of one byte
- 011D = Write to register address 0x11D
- 00 = Data byte to write
- 6074 = CRC

LaunchPad:

```
WriteReg(0,  ECC_TEST, 0x00, 1, FRMWRT_ALL_NR);
```

Next, make sure auto-address mode is set on all devices (and **NOT** GPIO address mode) by setting the CONFIG register (0x001):

```
D0 00 01 00 39 74
```

- D0 = Broadcast write of one byte
- 0001 = Write to register address 0x001
- 00 = Data byte to write
- 3974 = CRC

LaunchPad:

```
WriteReg(0, CONFIG, 0x00, 1, FRMWRT_ALL_NR);
```

Now, enable and enter auto-addressing mode on the devices by setting the CONTROL1 register (0x105):

```
D0 01 05 01 AB B4
```

- D0 = Broadcast write of one byte
- 0105 = Write to register address 0x105
- 01 = Data byte to write
- ABB4 = CRC

LaunchPad:

```
WriteReg(0, CONTROL1, 0x01, 1, FRMWRT_ALL_NR);
```

To individually distribute an address for each device, loop through the number of devices and give each device an address. Broadcast write consecutive addresses until all parts have been assigned a valid address.

Note: This is a rare instance in which a "Broadcast Write" command actually acts on each device INDIVIDUALLY, despite being a broadcast.

```
D0 01 04 00 6B E4
D0 01 04 01 AA 24
D0 01 04 02 EA 25
....
```

- D0 = Broadcast write of one byte
- 0104 = Write to register address 0x0104 (DEVADD_USR register)
- 00, 01, 02... = Data byte to write (assigns each device address)
- XXXX (last 2 bytes) = CRC

LaunchPad:

```
for (nCurrentBoard = 0; nCurrentBoard <= TOTALBOARDS - 1; nCurrentBoard++)
{
    WriteReg(nCurrentBoard, DEVADD_USR, nCurrentBoard, 1, FRMWRT_ALL_NR);
}
```

Now that each device has an address, the devices must be set to base, stack, and/or top of stack. There are two possible setups:

2 OR MORE DEVICES IN STACK:

If there are multiple devices in the stack, there must be a base device and a top of stack device. The easiest way to configure each device is to first set EVERY device as a stack device (0x02), then set the top of stack and base devices individually:

1. Broadcast write CONFIG=0x02 to all devices
2. Single device write CONFIG=0x00 to device 0x00 (base device)
3. Single device write CONFIG=0x00 to device TOTALBOARDS-1 (top of stack)

Provided is a table representation of four devices in a stack:

DEVICE	BASE	STACK	STACK	TOP
Write to CONFIG:	0x00	0x02	0x02	0x03

Additionally, here is a table representation of two devices in a stack:

DEVICE	BASE	TOP
Write to CONFIG:	0x00	0x03

To broadcast write CONFIG=0x02 to all devices, the following is sent:

```
D0 00 01 02 B8 B5
```

- D0 = Broadcast write of one byte
- 0001 = Write to register address 0x0001 (CONFIG register)
- 02 = Data byte to write
- B8 B5= CRC

LaunchPad:

```
WriteReg(0, CONFIG, 0x02, 1, FRMWRT_ALL_NR);
```

For the base, clear the CONFIG register:

```
90 00 00 01 00 E5 8D
```

- 90 = Single device write of one byte
- 00 = Device address (base device, bottom of stack)
- 0001 = Write to register address 0x0001 (CONFIG register)
- 00 = Data byte to write
- E58D = CRC

LaunchPad:

```
WriteReg(0, CONFIG, 0x00, 1, FRMWRT_SGL_NR);
```

For the top of the stack, set the TOP_STACK bit and the STACK_DEV bit in the CONFIG register:

```
90 0F 00 01 03 A6 98
```

- 90 = Single device write of one byte
- 0F = Device address (choose the address of the top device, in this case, the top device address is 15 [0x0F])
- 0001 = Write to register address 0x0001 (CONFIG register)
- 03 = Data byte to write (set as stack device AND top of stack)
- A698 = CRC

LaunchPad:

```
WriteReg(TOTALBOARDS-1, CONFIG, 0x03, 1, FRMWRT_SGL_NR);
```

1 DEVICE IN STACK:

If there is only one device, instead of the above commands, you can instead do one command to assign the device as both the base AND top of stack (set CONFIG to 0x01):

```
90 00 00 01 01 24 4D
```

- 90 = Single device write of one byte
- 00 = Device address (there is only one device)
- 0001 = Write to register address 0x0001 (CONFIG register)
- 01 = Data byte to write (set as base device AND top of stack)
- 244D = CRC

Finally, an additional dummy read of ECC_TEST must be done to finish synchronizing the DLL:

```
C0 01 1D 00 64 B4
```

- C0 = Broadcast read of 1 byte

- 011D = Read starting from register 0x11D (ECC_TEST)
- 00 = Read 1 byte of data from each device
- 64B4 = CRC

4 Initializing Devices

There are several device configuration settings that are useful (but not required) for basic operation, and must be set once the device has been auto-addressed.

4.1 Set Communications Timeout

Communications timeout can be set by manipulating the COMM_TO (0x23) register.

```
D0 00 23 56 A1 EA
```

- D0 = Broadcast write of one byte
- 0023 = Write to register address 0x23 (COMM_TO register)
- 56 = Data byte to write (10 minute short communication timeout, sleep mode on long communication timeout, long timeout length of 30 minutes)
- A1EA = CRC

Communications transmit delay can be set to zero by changing the TX_HOLD_OFF register.

```
D0 00 22 00 20 44
```

- D0 = Broadcast write of one byte
- 0022 = Write to register address 0x22 (TX_HOLD_OFF register)
- 00 = Data byte to write (0 transmit delay)
- 2044 = CRC

4.2 Masking Low Level Faults

This example shows how to mask all low level faults, which can be done by the following register manipulations:

```
D0 00 02 3F 79 94 //GPIO
D0 00 03 3F 78 04 //UV
D0 00 04 3F 7A 34 //OV
D0 00 05 3F 7B A4 //UT
D0 00 06 3F 7B 54 //OT
D0 00 07 07 7B 16 //all tone faults
D0 00 08 07 7E E6 //UART
D0 00 09 3F 7E A4 //UART
D0 00 0A 3F 7E 54 //UART
D0 00 0B 03 7F D5 //UART
D0 00 0C 3F 7D F4 //COMH
D0 00 0D 3F 7C 64 //COMH
D0 00 0E 3F 7C 94 //COMH
D0 00 0F 03 7D 15 //COMH
D0 00 10 3F 75 34 //COML
D0 00 11 3F 7A A4 //COML
D0 00 12 3F 74 54 //COML
D0 00 13 03 75 D5 //COML
D0 00 14 07 76 26 //OTP
D0 00 15 FF 76 34 //power rail
D0 00 16 7F 77 64 //SYS_FAULT 1
D0 00 17 FF 77 54 //SYS_FAULT 2
D0 00 18 7F 73 04 //SYS_FAULT 3
D0 00 19 03 73 75 //OVUV BIST
D0 00 1A FF 73 C4 //OTUT BIST
```

- D0 = Broadcast write of one byte
- 0002-001A = Write to register address 0x00##
- 3F, 7F, and so forth = Data byte to write

- XXXX (last two bytes) = CRC

4.3 Enable OVUV

Prepare cell OV/UV features by manipulating corresponding registers:

```
D0 00 29 3F 67 64 //enable OVUV for all 6 cell channels
D0 00 2A 53 67 B9 //set cell UV to 2.8 V
D0 00 2B 5B 67 EF //set cell OV to 4.3 V
```

Now that the settings are chosen, OVUV functions can be enabled. OVUV_EN=1 must be set in the CONTROL2 register:

```
D0 01 06 04 6B 47 // OVUV_EN=1
```

Set OVUV_EN=0 before changing settings.

4.4 GPIO - Absolute vs Ratiometric Voltage

Changing GPIO values to AUX voltage (absolute voltage instead of ratiometric) can be done by the following:

```
D0 00 28 3F 66 F4 //configure GPIO as AUX voltage (absolute voltage)
```

4.5 Miscellaneous ADC Settings

It may also be useful to modify the ADC delay of each device individually. The delay is programmable from 0 µs to 155 µs with 5 µs step size. This allows for synchronizing of ADC readings between stack devices. In this example, set the delay to 0 µs for all devices in the stack:

```
90 00 00 27 00 FF ED // modify device 0 delay
90 01 00 27 00 FE 11 // modify device 1 delay
...
```

The following modify values for delay and sample rate:

```
D0 00 26 08 23 42 //AUX sample rate 1 MHz, 128 decimation ratio
D0 00 24 23 62 3D //1 MHz sample rate, 64 decimation ratio, 19.7 Hz LPF
D0 00 25 02 A3 B5 //5 ms conversion interval if continuous conversion enabled
```

4.6 Enable TSREF

Enabling TSREF LDO output is useful for external temperature sensors. Ensure that there is a delay of approximately 2 ms to give enough settling time.

```
D0 01 06 10 6B 48
```

Example TMS570 code for this:

```
nSent = WriteReg(0, CONTROL2, 0x10, 1, FRMWRT_ALL_NR); // enable TSREF
delayms(2); // provides settling time for TSREF
```

4.7 Checking Device Status

In order to ensure that the status of each device is okay, reading the following registers for each device is helpful:

```
80 00 02 00 00 84 1E // read PARTID
80 00 02 04 00 86 DE // read DEV_STAT
80 00 02 06 00 87 BE // read FAULT_SUM
```

- 80 = Single device read of 1 byte
- 00 = Device 0
- 0200, 0204, 0206 = Register address 0x200, 0x204, 0x206
- 00 = Read back one byte of data
- 841E, 86DE, 87BE = CRC

5 Read Cell Voltages

5.1 General Setup - One-Shot and Continuous Conversions

For both one-shot and continuous ADC conversions, it is necessary to first choose which cells are enabled for the readings. For this example, all cells are enabled.

Enable CELL1-CELL6 in CELL_ADC_CTRL:

```
D0 01 09 3F 2F 64 // enables ADC for all 6 cell channels
[delay 5 ms] // ensure proper settling time for best accuracy
```

5.2 One-Shot ADC Conversions

To begin a single, one-shot cell ADC conversion, set the CELL_ADC_GO bit in the CONTROL2 register, wait 5 ms to ensure ADC accuracy, and finally read back the data:

```
D0 01 06 01 AB 44 // CELL_ADC_GO - convert all 6 cell channels for all devices
[delay 5 ms] // delay for ADC accuracy
C0 02 15 0B D2 B3 // will return 6 overhead bytes and 12 data bytes per device
// highest device address responds first
[delay 1 ms]
```

- D0 = Broadcast write of one byte
- 0106 = Write to register 0x106 (CONTROL2 register)
- 01 = Data byte to write
- AB44 = CRC
- C0 = Broadcast read of 12 bytes
- 0215 = Read starting from register 0x215 (VCELL1H)
- 0B = Read 12 bytes of data from each device
- D2B3 = CRC

5.3 Continuous ADC Conversions

8.3.4.2.1 Continuous ADC Conversions

Running continuous ADC conversions requires a similar setup to one-shot conversions, with two additional steps before running the conversion: enabling continuous cell conversions and choosing the ADC conversion interval. These are both located in the CELL_ADC_CONF2 register:

```
D0 00 25 0A A2 73 // enable continuous conversion with 5ms conversion interval
```

Set CELL_ADC_GO to 1 to begin ADC conversions:

```
D0 01 06 01 AB 44 // CELL_ADC_GO - convert all 6 cell channels for all devices
[delay 5 ms] // delay for ADC accuracy
```

Finally, the results can be read as before, and update continuously at the interval specified previously (5 ms for our example):

```
C0 02 15 0B D2 B3 // will return 6 overhead bytes and 12 data bytes per device
// highest device address responds first
[delay 1ms] //need to wait for the read bytes to complete sending
```

6 Enable Cell Balancing

6.1 Cell Balance Setup

Voltage thresholds, timers, and sequencing must all be programmed to set up balancing.

6.1.1 Sequencing, Overall Duty Cycle, Fault Reaction

Changing the duty cycle unit, cell balancing duty cycle, fault reaction, and sequencing of cell balancing (odds or evens first, or just odds/evens) can all be set with the CB_CONFIG register:

```
D0 01 0D FA ED F7 // 30 second duty cycle, continue on fault, odds then evens
```

6.1.2 Voltage Thresholds

Using the CB_DONE_THRESH register, the cell balancing "done" threshold can be enabled/disabled and set. The voltage threshold is programmable from 2.8 V to 4.3 V with 25 mV step size. Values set above 4.3 V are capped to 4.3 V.

Enabling the CBDONE voltage threshold overrides the OVUV function and pauses it.

In the example TMS570 code, the CBDONE comparator function is disabled. However the following provides an example of enabling the voltage thresholds using CBDONE:

```
D0 01 14 64 67 CF // enable CBDONE voltage thresholds, and set to 3.7 V threshold
```

6.1.3 Individual Cell Balancing Timers

Cell balancing time for each cell can be modified using the CB_CELL*_CTRL registers, and is programmable from 0 to 127 minutes. Setting 0 disables the balancing for that cell. These cannot be modified while CONTROL2[BAL_GO] = 1 is set, so cell balancing must be disabled to make changes to these registers, then restarted. To set a one minute balance timer for each cell:

```
D0 01 0E 01 AC 84 // cell1 - 1 minute balance timer
D0 01 0F 01 AD 14 // cell2 - 1 minute balance timer
D0 01 10 01 A5 24 // cell3 - 1 minute balance timer
D0 01 11 01 A4 B4 // cell4 - 1 minute balance timer
D0 01 12 01 A4 44 // cell5 - 1 minute balance timer
D0 01 13 01 A5 D4 // cell6 - 1 minute balance timer
```

6.2 Run Cell Balancing

Once all of the cell balancing settings have been configured as desired, use the following to run the cell balancing.

```
D0 01 06 30 6A 90 // set BAL_GO and ensure TSREF is enabled
[delay 100 us]
C0 02 04 00 9F 24 // check to see if DEV_STAT[CB_RUN] = 1
[delay 500 us]
C0 02 04 00 9F 24 // check to see if DEV_STAT[CB_DONE] = 1
[delay 500 us] // then check as often as wanted to see if CB_DONE is set]
```

7 Reverse Device Addressing and Communications

To reverse the communications and addressing of an already functioning stack, first disable the high-side RX and TX by writing DAISY_CHAIN_CTRL[COMHRX/TX_EN]=0, and DAISY_CHAIN_CTRL[COMLRX/TX_EN]=1:

```
90 00 00 21 30 FC 59 //swapping to low-side comms for base device
```

- 90 = Single device write of one byte
- 00 = Base device
- 0021 = Register address 0x0021 (DAISY_CHAIN_CTRL)
- 30 = Byte to write (swap to low-side comms from high-side)
- FC59 = CRC

Now enable DAISY_CHAIN_CTRL_EN in CONTROL2 on the base device, allowing COMH/L TX/RX to be controlled by the DAISY_CHAIN_CTRL register:

```
90 00 01 06 40 B7 8D //give control to DAISY_CHAIN_CTRL register
```

Next, reverse the direction of the base device's communications so that subsequent commands go to low side:

```
90 00 01 05 80 B7 2D // set DIR_SEL in CONTROL1 register
```

Now that communications for the base device is in the proper direction, send broadcast write to all devices to change their communications direction.

```
E0 01 05 80 64 D4 // broadcast to all devices to reverse direction
```

Clear the CONFIG register of all devices so that STACK_DEV and TOP_STACK are cleared for all devices:

```
D0 00 01 00 39 74 //clear CONFIG register
```

Now that the stack is fully reversed, auto-addressing can occur as normal. First, ensure that all devices are prepared for auto-addressing by using a broadcast write to the CONTROL1 register. Set ADD_WRITE_EN=1 and make sure that DIR_SEL=1 is still set in CONTROL1:

```
D0 01 05 81 AA 14 //prep auto-addressing with ADD_WRITE_EN=1 and make sure DIR_SEL is still 1
```

Assign each device a new address by broadcast writing to DEVADD_USR register, once for each device in the stack, just as mentioned in the Auto-Addressing section of this guide:

```
D0 01 04 00 6B E4 // program device 0 (new base device)
D0 01 04 01 AA 24 // program device 1
D0 01 04 02 EA 25 // program device 2
...
```

Do not forget to correctly set stack devices. For a stack with three devices total (top of stack will be different):

```
90 01 00 01 02 65B0 //set stack device with devID 1 as STACK_DEV
```

Do not forget to set the top of stack. For a stack with three devices total:

```
90 02 00 01 03 A4 34 //set device with devID 2 as TOP_STACK and STACK_DEV
```

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from D Revision (July 2019) to E Revision **Page**

- Updated the Data row for Table 1 through Table 6 2
- Updated FRMWRT_REV_ALL_NR in Section 1.2.3 table 4
- Updated Section 2 4
- Updated Section 3.2 4
- Updated Section 3.3 5
- Updated Section 4.5 9
- Edited code in Section 7 11

Changes from C Revision (April 2019) to D Revision **Page**

- Changed typo in Section 2 4

Changes from B Revision (October 2018) to C Revision	Page
• Changed instances of BQ79606/BQ79606-Q1 to BQ79606A/BQ79606A-Q1.	1

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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, 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 (www.ti.com/legal/termsofsale.html) 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.

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