Application Report

BQ769x2 Software Development Guide

Matt Sunna

ABSTRACT

This application report provides examples of communication packets and sequences for the BQ769x2 device family of battery monitors (which includes the BQ76952, BQ76942, and BQ769142). Examples include bit-transaction details of direct commands, subcommands, and reads and writes to RAM registers. Examples include instructions for using the *BQStudio Command Sequence* panel to perform these read and write transactions. Simple code examples are also provided. Use this document along with the device-specific Technical Reference Manual and data sheet. These documents are listed in Section 7. BQSTUDIO software is also used for many examples and offers a convenient way to view all of the device registers. For the BQ769x2 device family, version 1.3.102 or above of BQStudio is required.

The BQ769x2 device family integrates three different communication interfaces - I^{2}C, SPI, and single-wire HDQ. The I^{2}C and SPI interfaces include an optional CRC check. For the full list of options, see the device-specific data sheet. This document covers many examples using the I^{2}C interface and then covers some of the same examples using SPI with CRC.

Table of Contents

List of Figures

List of Tables

Trademarks

All trademarks are the property of their respective owners.

1 Direct Commands

A complete list of direct commands can be found in the device-specific Technical Reference Manual. The format for a direct command is shown in the following examples.

1.1 Alarm Enable - 0x66

Table 1-1 shows the Alarm Enable command that uses command 0x66. By default, the register setting for Alarm Enable is set to 0xF800. In the example, the setting is changed to 0xF082. The data is in little endian format. The device address for the BQ769x2 is 0x10 (8-bits) where the LSB is the R/W bit. A direct command follows the format *I2C_Write(I2C_ADDR, Command, DataBlock)*, so for this example the command would be *I2C_Write(0x10, 0x66, [0x82, 0xF0])*.

Table 1-1. Alarm Enable Command Description

Command	Name	Units	Type	Description
0x66	Alarm Enable	Hex	H2	Mask for Alarm Status(). Can be written to change during operation to change which alarm sources are enabled.

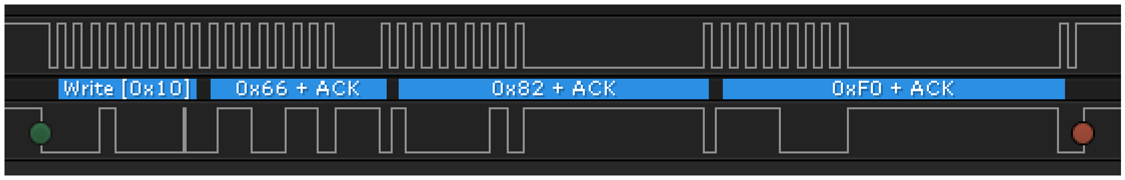


Figure 1-1. Captured I2C Waveform for Setting Alarm Enable to 0xF082

1.2 Cell 1 Voltage - 0x14

Table 1-2 shows how to read the voltage for Cell 1. The Cell 1 Voltage command is 0x14 and is a read only command. The Cell 1 voltage is read by writing the I2C command 0x14 followed by a 2-byte read. The data is returned in little endian format. In the following example, the 16-bit Cell 1 voltage read 0x0E74, which corresponds to 3700 mV.

Table 1-2. Cell 1 Voltage Command Description

Command	Name	Units	Type	Description
0x14	Cell 1 Voltage	mV	I2	16-bit voltage on cell 1

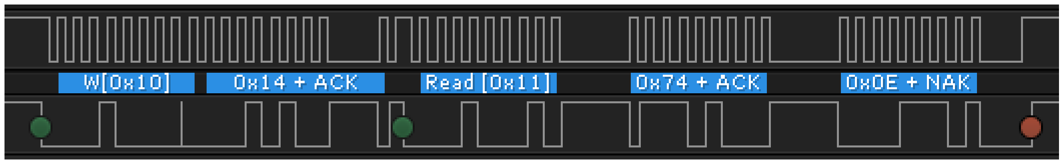


Figure 1-2. Captured I2C Waveform for Cell 1 Voltage Reading

1.3 Internal Temperature - 0x68

Table 1-3 shows how to read the internal temperature sensor. The units for the 16-bit temperature sensor reading are in 0.1 K. In the following example, the reading of 0x0BA6 represents a decimal value of 2982, which is 298.2 K. This converts to about 25°C.

Table 1-3. Internal Temperature Command Description

Command	Name	Units	Type	Description
0x68	Int Temperature	0.1 K	I2	This is the most recent measured internal die temperature.

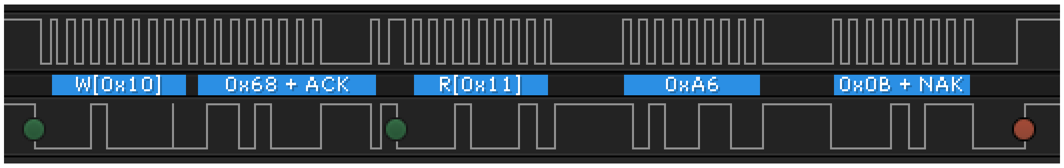


Figure 1-3. Captured I2C Waveform for Internal Temperature Reading

1.4 CC2 Current - 0x3A

Table 1-4 shows how to read the 16-bit current measurement from CC2. The current reading in the following example shows 7 mA.

Table 1-4. CC2 Current Command Description

Command	Name	Units	Type	Description
0x3A	CC2 Current	userA	I2	16-bit CC2 current

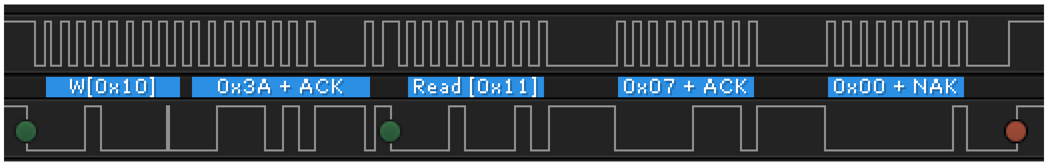
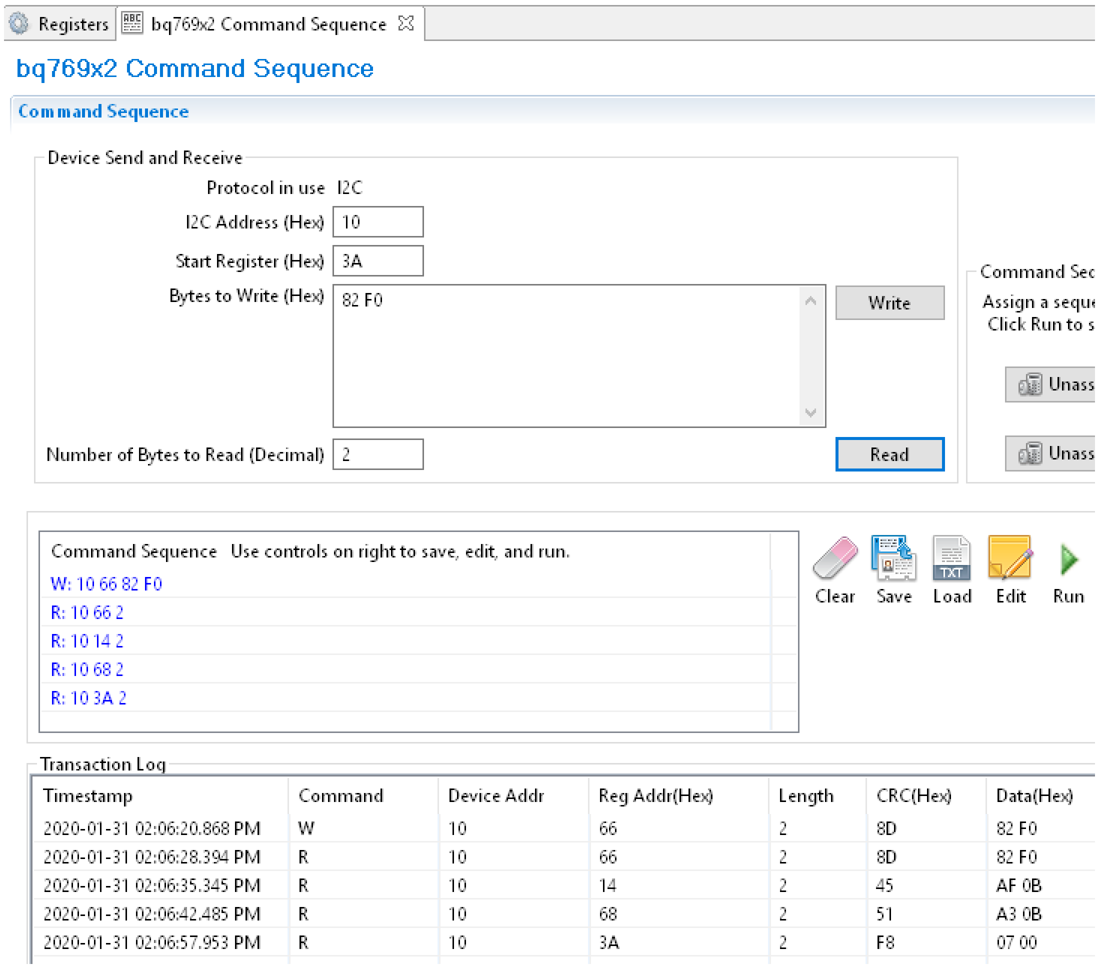


Figure 1-4. Captured I2C Waveform for CC2 Current Reading

The Command Sequence module in the BQStudio software enables you to try commands. This tool can also be used to create and save command sequences. The *Transaction Log* in this example shows all of the commands that have been covered so far.



Registers | bq769x2 Command Sequence

bq769x2 Command Sequence

Command Sequence

Device Send and Receive

Protocol in use I2C

I2C Address (Hex) 10

Start Register (Hex) 3A

Bytes to Write (Hex) 82 F0

Number of Bytes to Read (Decimal) 2

Command Sequence Use controls on right to save, edit, and run.

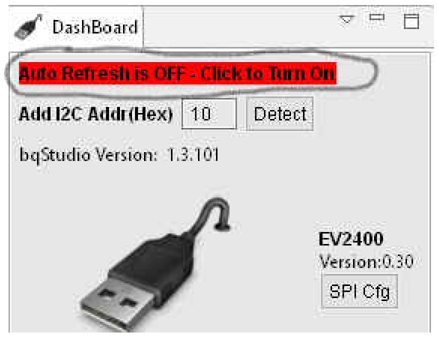
W: 10 66 82 F0
R: 10 66 2
R: 10 14 2
R: 10 68 2
R: 10 3A 2

Transaction Log

Timestamp	Command	Device Addr	Reg Addr(Hex)	Length	CRC(Hex)	Data(Hex)
2020-01-31 02:06:20.868 PM	W	10	66	2	8D	82 F0
2020-01-31 02:06:28.394 PM	R	10	66	2	8D	82 F0
2020-01-31 02:06:35.345 PM	R	10	14	2	45	AF 0B
2020-01-31 02:06:42.485 PM	R	10	68	2	51	A3 0B
2020-01-31 02:06:57.953 PM	R	10	3A	2	F8	07 00

Figure 1-5. BQStudio Example Showing Execution of Multiple Direct Commands

BQStudio has an *Auto Refresh* on the *Dashboard* which periodically reads the registers of the device to refresh the measurements displayed. When using the *Command Sequence* module, it is recommended to disable *Auto Refresh* by clicking on the green banner. The banner will turn red to indicate *Auto Refresh* is disabled (see Figure 1-6).



DashBoard

Auto Refresh is OFF - Click to Turn On

Add I2C Addr(Hex) 10

bqStudio Version: 1.3.101

EV2400
Version:0.30

Figure 1-6. Auto Refresh Disabled

2 Subcommands

Subcommands use a different format from direct commands and are accessed indirectly using the 7-bit command address space. They also provide the capability for block transfers. To issue a subcommand, the command address is written to 0x3E/0x3F. If data is to be read back, it will be populated in the 32-byte transfer buffer which uses addresses 0x40 - 0x5F. Multiple examples follow.

The timing required for the device to fetch data depends on the specific subcommand and any other processing underway within the device, so it will vary during operation. The approximate times for each subcommand are shown in the Technical Reference Manual. There are two approaches for addressing this timing when reading data from a subcommand:

The simplest approach is to use a 2 ms wait time after writing to 0x3E/0x3F before reading the result from the transfer buffer.

A second approach is described in Chapter 3 of the Technical Reference Manual. This approach is to read from 0x3E/0x3F until the subcommand has completed operation. If the value returned is 0xFF, this indicates the subcommand has not completed operation yet. When the subcommand has completed, the value returned will match the command that was written. This response only applies to subcommands that return data to be read back.

Certain subcommands write data to a register and must be followed by a write to 0x60/0x61 with the checksum and length. This only applies to the *FET_Control()*, *REG12_Control()*, *CB_Active_Cells()*, and *CB_SET_LVL()* subcommands. Examples for calculating checksum and length are provided in the next section since this is also required when writing to RAM registers.

2.1 DEVICE_NUMBER - 0x0001

The device number can be read by first writing the subcommand number 0x0001 (little endian) to the command address 0x3E. This is followed by reading from the data buffer at address 0x40. In this example, the device number returned is 0x7694 (which represents BQ76942).

Table 2-1. DEVICE_NUMBER Subcommand Description

Command	Name	Data	Units	Type	Description
0x0001	DEVICE_NUMBER	Device Number	Hex	U2	Reports the device number that identifies the product. The data is returned in little-endian format

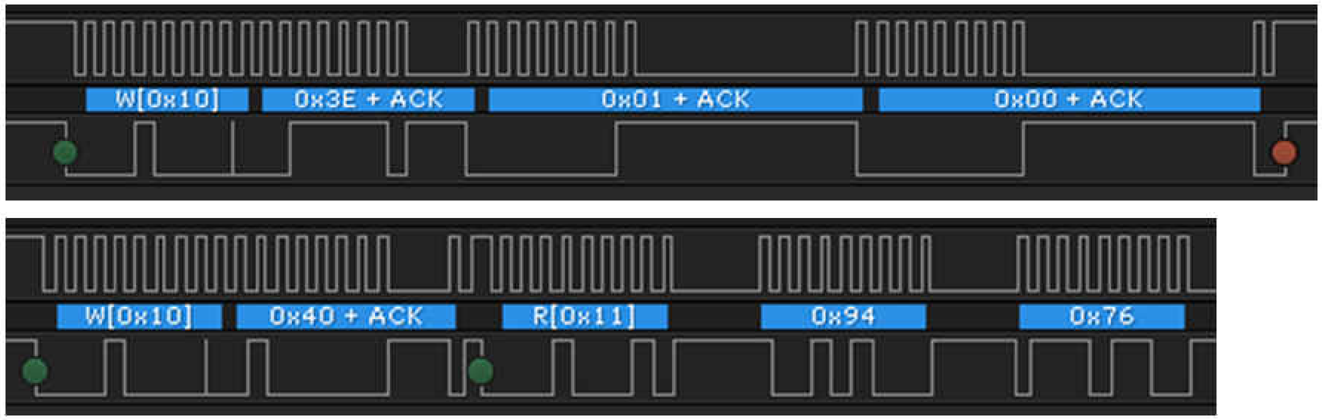


Figure 2-1. Captured I2C Waveform for DEVICE_NUMBER Subcommand

2.2 MANUFACTURING STATUS - 0x0057

The MANUFACTURING STATUS subcommand reads two bytes from the Manufacturing Status register. First, the command 0x0057 is written to 0x3E followed by a read of two bytes from 0x40.

Table 2-2. MANUFACTURING STATUS Subcommand Description

Command	Name	Data	Units	Type	Description
0x0057	MANUFACTURING STATUS	Manufacturing Status	Hex	H2	Provides flags for use during manufacturing.

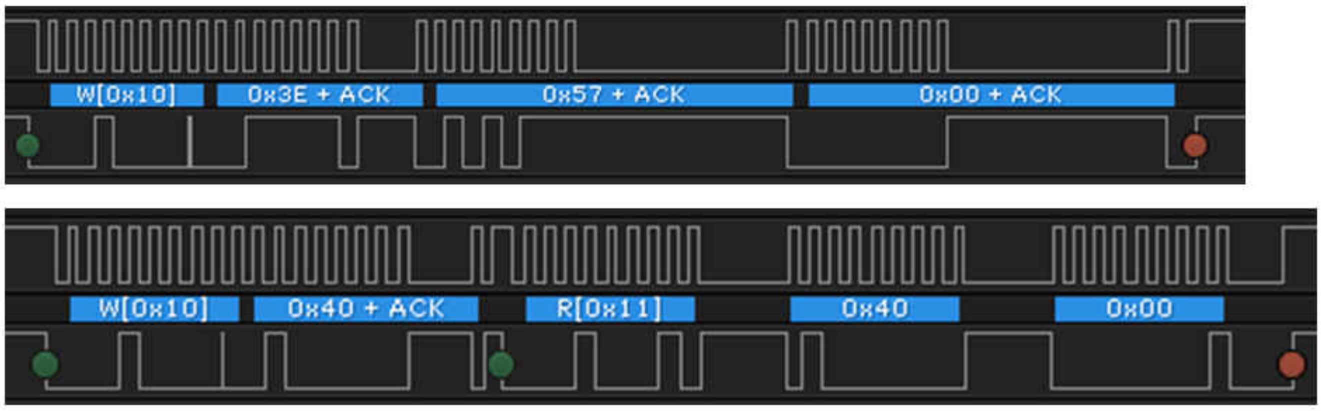


Figure 2-2. Captured I2C Waveform for MANUFACTURING_STATUS Subcommand

2.3 FET_ENABLE - 0x0022

Some subcommands do not require a data read from the data buffer since they only provide an instruction. The FET_ENABLE subcommand is one example. This command is issued by writing 0x0022 to 0x3E.

Table 2-3. FET_ENABLE Subcommand Description

Command	Name	Description
0x0022	FET_ENABLE	Toggle FET_EN in Manufacturing Status. FET_EN = 0 means FET Test Mode. FET_EN = 1 means Firmware FET Control.

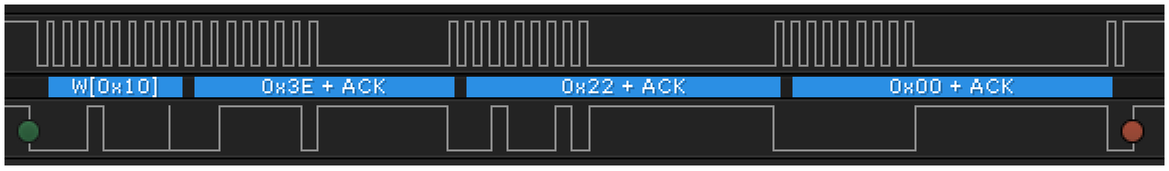


Figure 2-3. Captured I2C Waveform for FET_ENABLE Subcommand

2.4 RESET - 0x0012

The RESET subcommand performs a reset on the device and returns RAM register settings back to default (or OTP programmed) values. This command is issued by writing 0x0012 to 0x3E.

Table 2-4. RESET Subcommand Description

Command	Name	Description
0x0012	RESET	Resets the device.

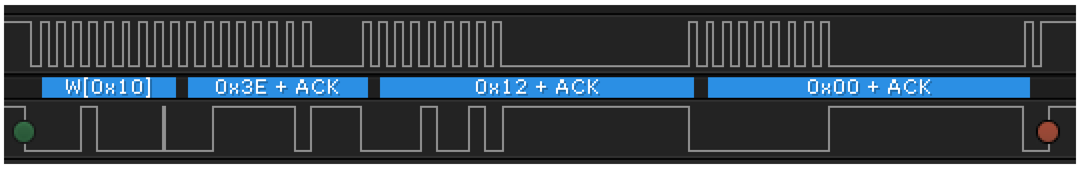
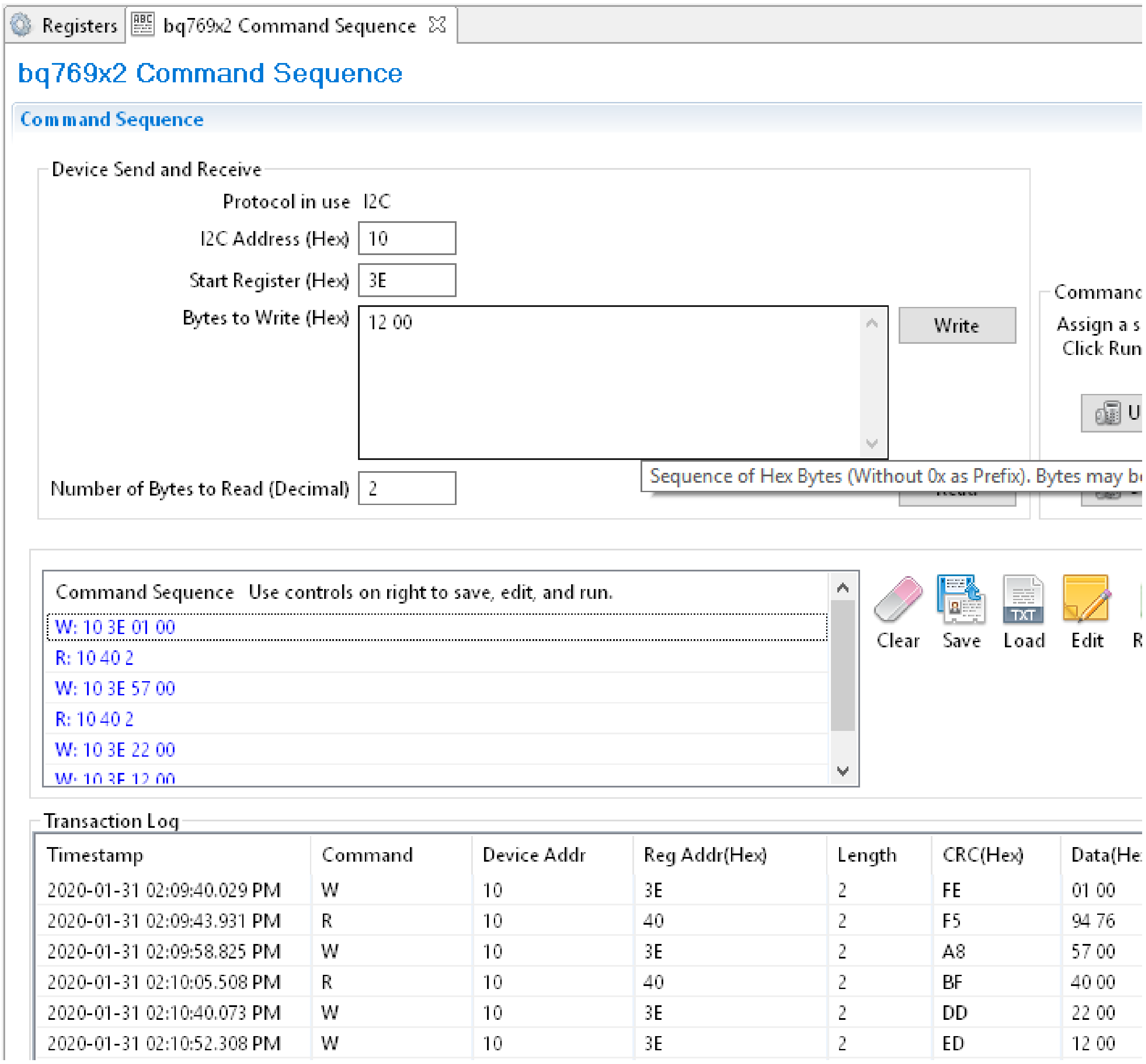


Figure 2-4. Captured I2C Waveform of RESET Subcommand

The *Transaction Log* in this example shows all of the commands that have been covered for executing Subcommands.



Protocol in use I2C
I2C Address (Hex) 10
Start Register (Hex) 3E
Bytes to Write (Hex) 12 00
Number of Bytes to Read (Decimal) 2
Sequence of Hex Bytes (Without 0x as Prefix). Bytes may b

Command Sequence Use controls on right to save, edit, and run.

```
W: 10 3E 01 00
R: 10 40 2
W: 10 3E 57 00
R: 10 40 2
W: 10 3E 22 00
W: 10 3E 12 00
```

Transaction Log

Timestamp	Command	Device Addr	Reg Addr(Hex)	Length	CRC(Hex)	Data(He
2020-01-31 02:09:40.029 PM	W	10	3E	2	FE	01 00
2020-01-31 02:09:43.931 PM	R	10	40	2	F5	94 76
2020-01-31 02:09:58.825 PM	W	10	3E	2	A8	57 00
2020-01-31 02:10:05.508 PM	R	10	40	2	BF	40 00
2020-01-31 02:10:40.073 PM	W	10	3E	2	DD	22 00
2020-01-31 02:10:52.308 PM	W	10	3E	2	ED	12 00

Figure 2-5. BQStudio Example Showing Execution of Multiple Subcommands

3 Reading and Writing RAM Registers

A full view of registers in RAM can be found in the device-specific Technical Reference Manual and also in the Data Memory screen of BQStudio. To enable viewing the RAM register addresses in BQStudio, go to the Window->Preferences menu and select 'Show Advanced Views'. Reading from a RAM register is accomplished by writing the register address to 0x3E and then reading from the data buffer starting at 0x40. Writing to a RAM register starts with writing the register address to 0x3E followed by the data, followed by a write to 0x60/0x61 with the checksum and length. The checksum and length calculation is described more in the device-specific data sheet, but is illustrated in the following examples.

Note

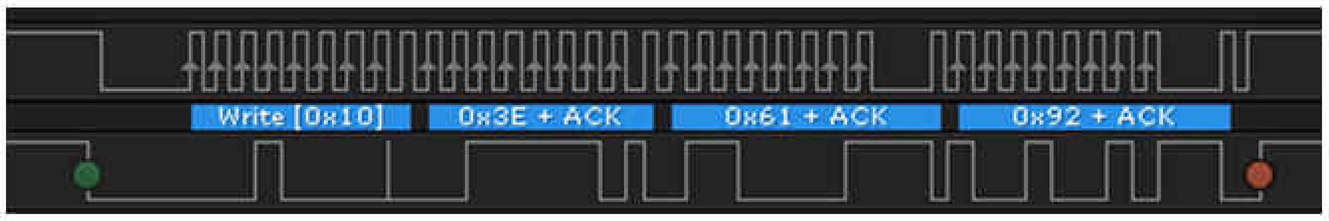
When writing to RAM registers, it is highly recommended to first enter CONFIG_UPDATE mode and then perform the command to exit CONFIG_UPDATE mode once complete. This ensures stable operation while settings are being modified.

3.1 Read 'Enabled Protections A'

The default settings for the BQ769x2 devices have COV (over-voltage) and SCD (short-circuit) protections enable. This is verified in the following by reading from the ***Enabled Protections A*** register where the value returned is 0x88 from the RAM address 0x9261.

Table 3-1. Enabled Protections A Description

Class	Subclass	Name	Type	Min	Max	Default	Unit
Settings	Protection	Enabled Protections A	U1	0x00	0xFF	0x88	Hex



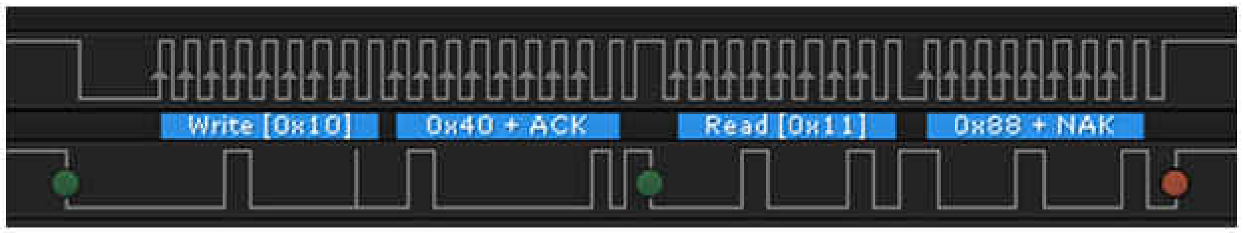


Figure 3-1. Captured I2C Waveform for Reading 'Enabled Protections A' Register

3.2 Enter CONFIG_UPDATE Mode

Before writing RAM registers, it is recommended to enter CONFIG_UPDATE mode to prevent settings from taking effect until all changes are made. SET_CFGUPDATE follows the Subcommand format.

Table 3-2. SET_CFGUPDATE and EXIT_CFGUPDATE Descriptions

Command	Name	Description
0x0090	SET_CFGUPDATE	Enters CONFIG_UPDATE mode.
0x0092	EXIT_CFGUPDATE	Exits CONFIG_UPDATE mode. This also clears the Battery Status()[POR] and Battery Status()[WD] bits.

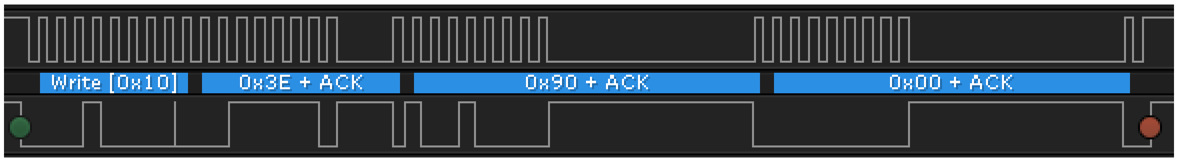
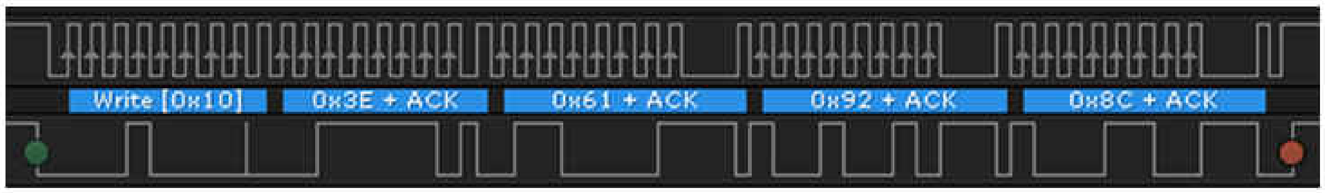


Figure 3-2. Captured I2C Waveform for SET_CFGUPATE

3.3 Write 'Enabled Protections A'

In this example, the CUV (undervoltage) protection feature is enabled along with the default protections. This requires writing 0x8C to RAM address 0x9261. The checksum is calculated on the address and data (0x61, 0x92, 0x8C) and is the complement of the sum of these bytes. The length also includes the two bytes for device address and command address for a total length of 5.



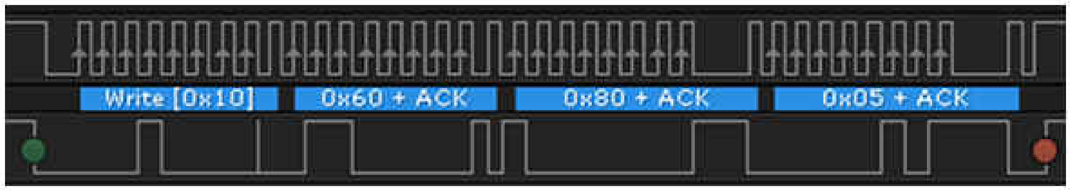


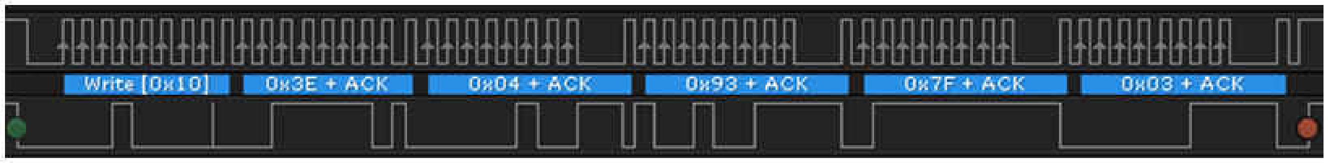
Figure 3-3. Captured Waveform for Writing to Enabled Protections A

3.4 Write 'VCell Mode'

Next, write to the ***VCell Mode*** register to configure the device BQ76942 for 9 cells. The following example writes 0x037F to 0x9304 and then writes the new checksum and length to 0x60/0x61.

Table 3-3. VCell Mode Description

Class	Subclass	Name	Type	Min	Max	Default	Unit
Settings	Configuration	VCell Mode	H2	0x0000	0xFFFF	0x0000	Hex



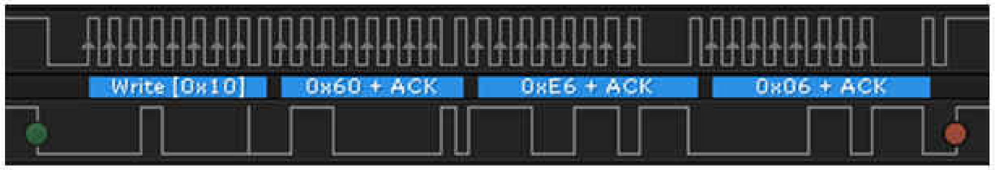


Figure 3-4. Captured I2C Waveform for Writing VCell Mode

3.5 Exit CONFIG_UPDATE Mode

After writing RAM registers, exit CONFIG_UPDATE mode at which point the new settings will take effect.

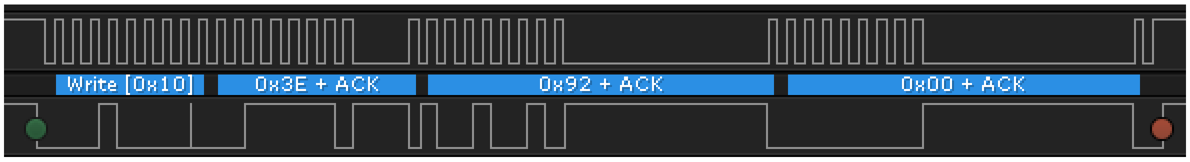
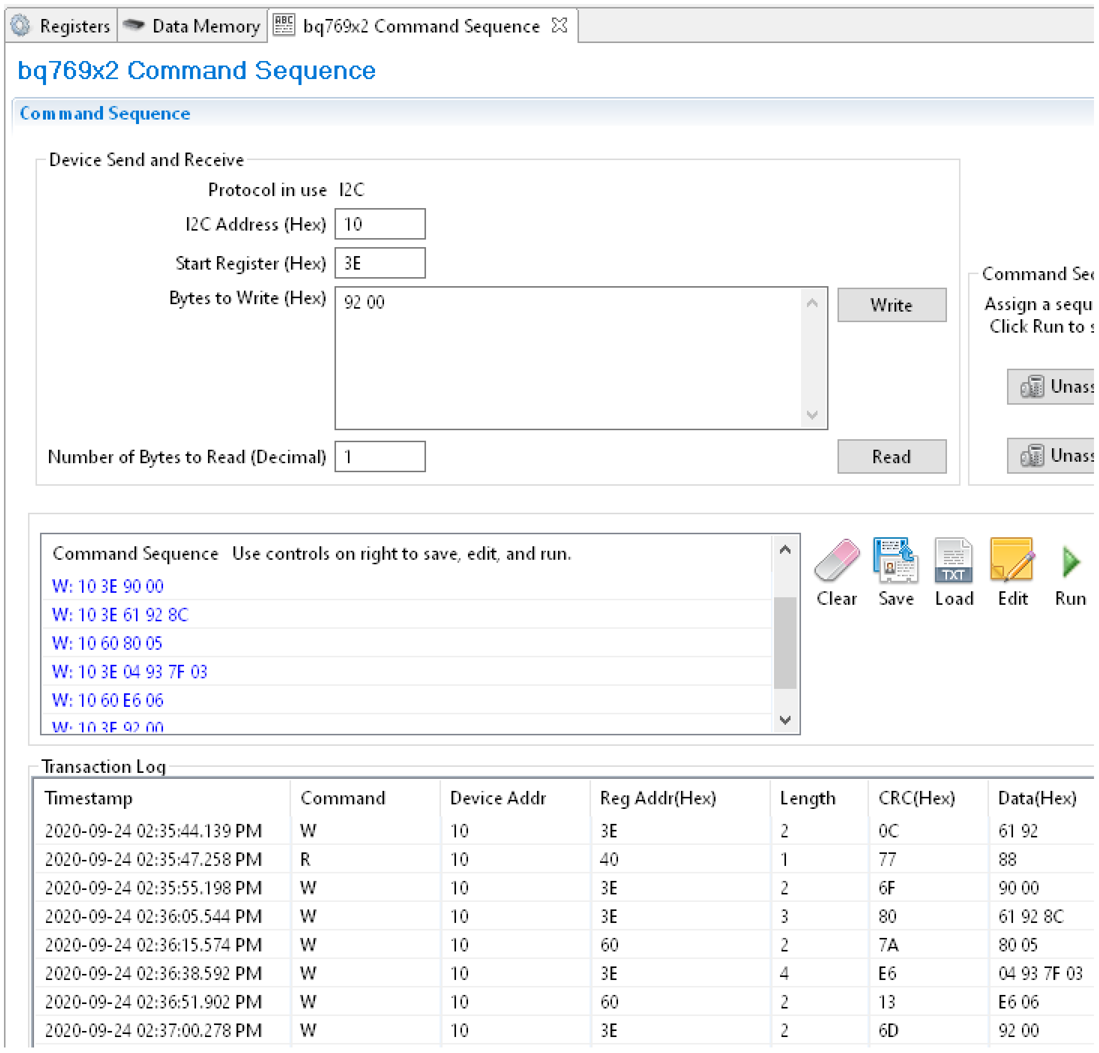


Figure 3-5. Captured I2C Waveform for EXIT_CFGUPDATE

The *Transaction Log* in this example shows all of the commands that have been covered for reading and writing to RAM registers.



Protocol in use I2C
I2C Address (Hex) 10
Start Register (Hex) 3E
Bytes to Write (Hex) 92 00
Number of Bytes to Read (Decimal) 1

Command Sequence Use controls on right to save, edit, and run.

```
W: 10 3E 90 00
W: 10 3E 61 92 8C
W: 10 60 80 05
W: 10 3E 04 93 7F 03
W: 10 60 E6 06
W: 10 3E 92 00
```

Transaction Log

Timestamp	Command	Device Addr	Reg Addr(Hex)	Length	CRC(Hex)	Data(Hex)
2020-09-24 02:35:44.139 PM	W	10	3E	2	0C	61 92
2020-09-24 02:35:47.258 PM	R	10	40	1	77	88
2020-09-24 02:35:55.198 PM	W	10	3E	2	6F	90 00
2020-09-24 02:36:05.544 PM	W	10	3E	3	80	61 92 8C
2020-09-24 02:36:15.574 PM	W	10	60	2	7A	80 05
2020-09-24 02:36:38.592 PM	W	10	3E	4	E6	04 93 7F 03
2020-09-24 02:36:51.902 PM	W	10	60	2	13	E6 06
2020-09-24 02:37:00.278 PM	W	10	3E	2	6D	92 00

Figure 3-6. BQStudio Example Showing Execution of RAM Register Reads and Writes

4 I2C With CRC

The I2C interface on the BQ769x2 family includes an optional CRC check. The CRC feature can be enabled in the ***Settings:Configuration:Comm Type*** register. If this register is changed while using BQStudio, the *SWAP_COMM_MODE()* subcommand should be executed and then BQStudio should be restarted so that it can detect the new communication mode. Two examples follow of I2C waveform captures with the CRC check enabled.

The CRC for the first data byte is computed on all of the bytes after the I2C start up to and including the first data byte. For every data byte after the first byte, the CRC byte is computed for only that byte. In Figure 4-1, using the *FET_ENABLE* subcommand, the CRC for the first byte is computed for [0x10 0x3E 0x22] - the resulting CRC is 0x63. The CRC for the second byte [0x00] is 0x00.

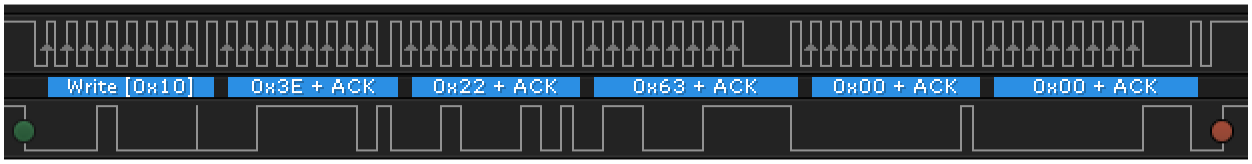


Figure 4-1. Captured I2C Waveform for FET_ENABLE Subcommand With CRC

Figure 4-2, using the *VCell 1* command, the CRC for the first byte is computed for [0x10 0x14 0x11 0x68] - the resulting CRC is 0x33. The CRC for the second byte [0x0B] is 0x31.

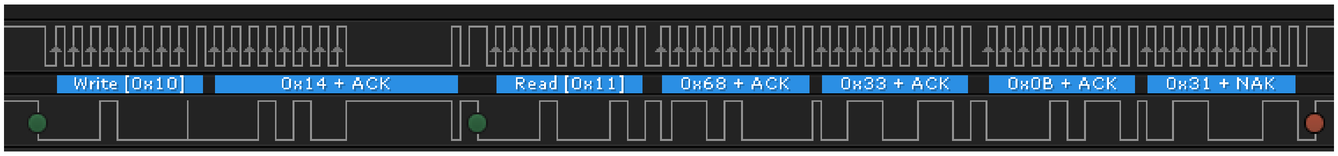


Figure 4-2. Captured I2C Waveform for VCell 1 Command With CRC

5 SPI With CRC Examples

The SPI interface on the BQ769x2 family can be enabled in the ***Settings:Configuration:Comm Type*** register. When changing to SPI mode, the default SPI output logic voltage level is 1.8V since it is referencing the internal voltage regulator of the device. To change the logic level, the REG1 LDO should be enabled and programmed to the desired voltage level and the ***SPI Configuration*** register should be programmed to 0x60 to enable the MISO_REG1 bit. Next, the *SWAP_COMM_MODE()* subcommand should be executed. If using BQStudio, then BQStudio should be restarted so that it can detect the new communication mode.

Some versions of the device will be available that are pre-configured to SPI mode.For information on the different part numbers available, see the device-specific data sheet.

The following examples cover some of the same commands covered in the I2C examples. Some important notes on the SPI interface protocol (with CRC):

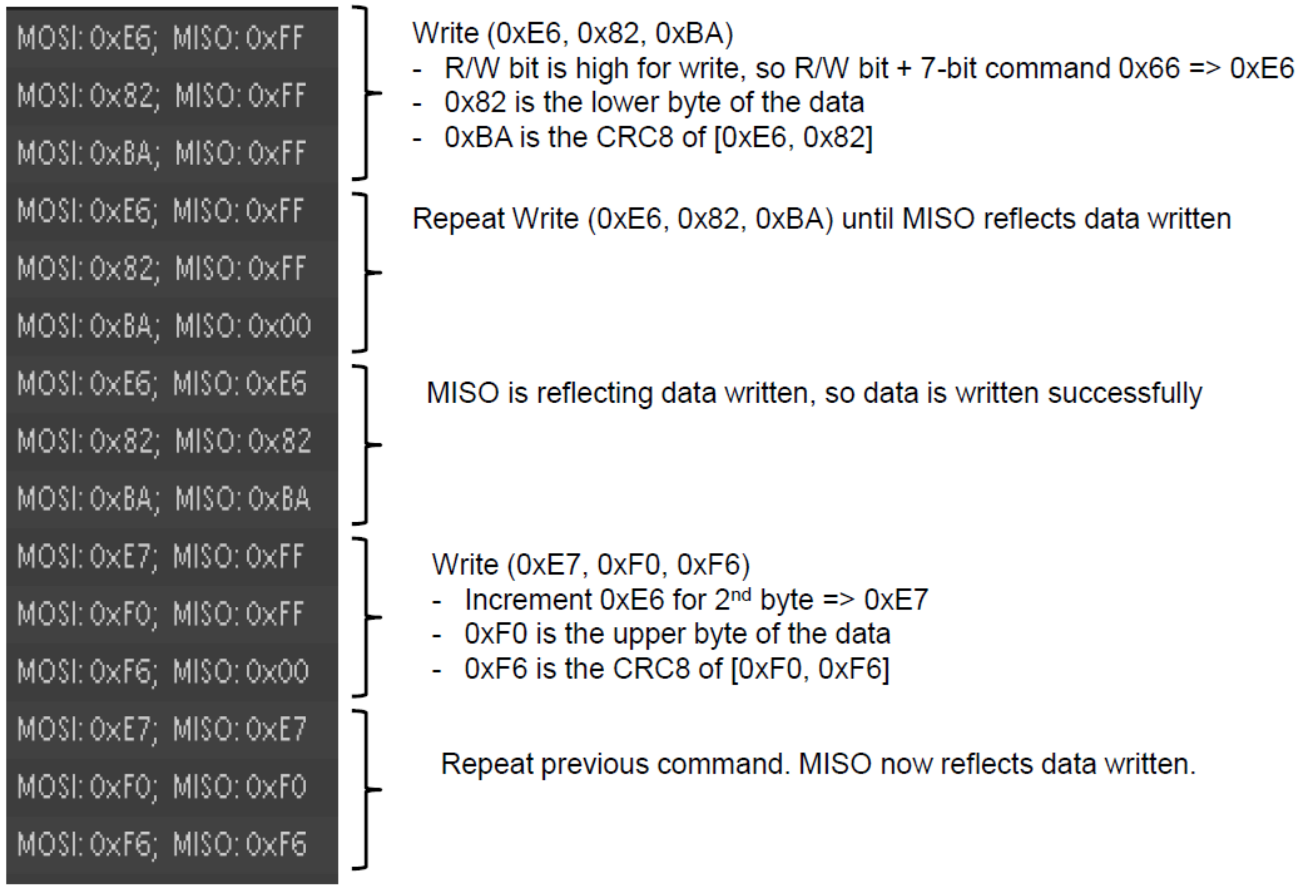
- SPI_CS is active low.
- The first SPI packet is 8 bits. The first bit is the R/W bit followed by a 7-bit address.
- The second packet is 8-bit data.
- The third packet is the 8-bit CRC calculated over the first and second bytes.

All of the examples include multiple writes for each transaction. This method is used by the EV2400 and BQStudio to verify that the commands have been successfully written. This is because some transactions are ignored by the device if the internal oscillator is not running (if the device is in SLEEP mode) or if the processor is busy. Once the data on the MISO pin (which should reflect the data previously written on MOSI) shows the correct data, then it is confirmed the packets have been written successfully. A more detailed description of the SPI interface is available in the device-specific Technical Reference Manual.

There are a couple of differences to be aware of when using the BQ769x2 family in SPI mode versus I2C mode. While I2C mode supports block writes and reads, SPI mode supports only single byte transactions. While I2C mode supports clock stretching for direct commands, SPI mode does not have this feature so it is important to be aware of the direct command timing in addition to subcommand timing.

5.1 Direct Command Example: Alarm Enable - 0x66

Figure 5-1 sets the Alarm Enable register to a value of 0xF082.



```
MOSI: 0xE6;  MISO: 0xFF
MOSI: 0x82;  MISO: 0xFF
MOSI: 0xBA;  MISO: 0xFF
MOSI: 0xE6;  MISO: 0xFF
MOSI: 0x82;  MISO: 0xFF
MOSI: 0xBA;  MISO: 0x00
MOSI: 0xE6;  MISO: 0xE6
MOSI: 0x82;  MISO: 0x82
MOSI: 0xBA;  MISO: 0xBA
MOSI: 0xE7;  MISO: 0xFF
MOSI: 0xF0;  MISO: 0xFF
MOSI: 0xF6;  MISO: 0x00
MOSI: 0xE7;  MISO: 0xE7
MOSI: 0xF0;  MISO: 0xF0
MOSI: 0xF6;  MISO: 0xF6
```

Write (0xE6, 0x82, 0xBA)
- R/W bit is high for write, so R/W bit + 7-bit command 0x66 => 0xE6
- 0x82 is the lower byte of the data
- 0xBA is the CRC8 of [0xE6, 0x82]

Repeat Write (0xE6, 0x82, 0xBA) until MISO reflects data written

MISO is reflecting data written, so data is written successfully

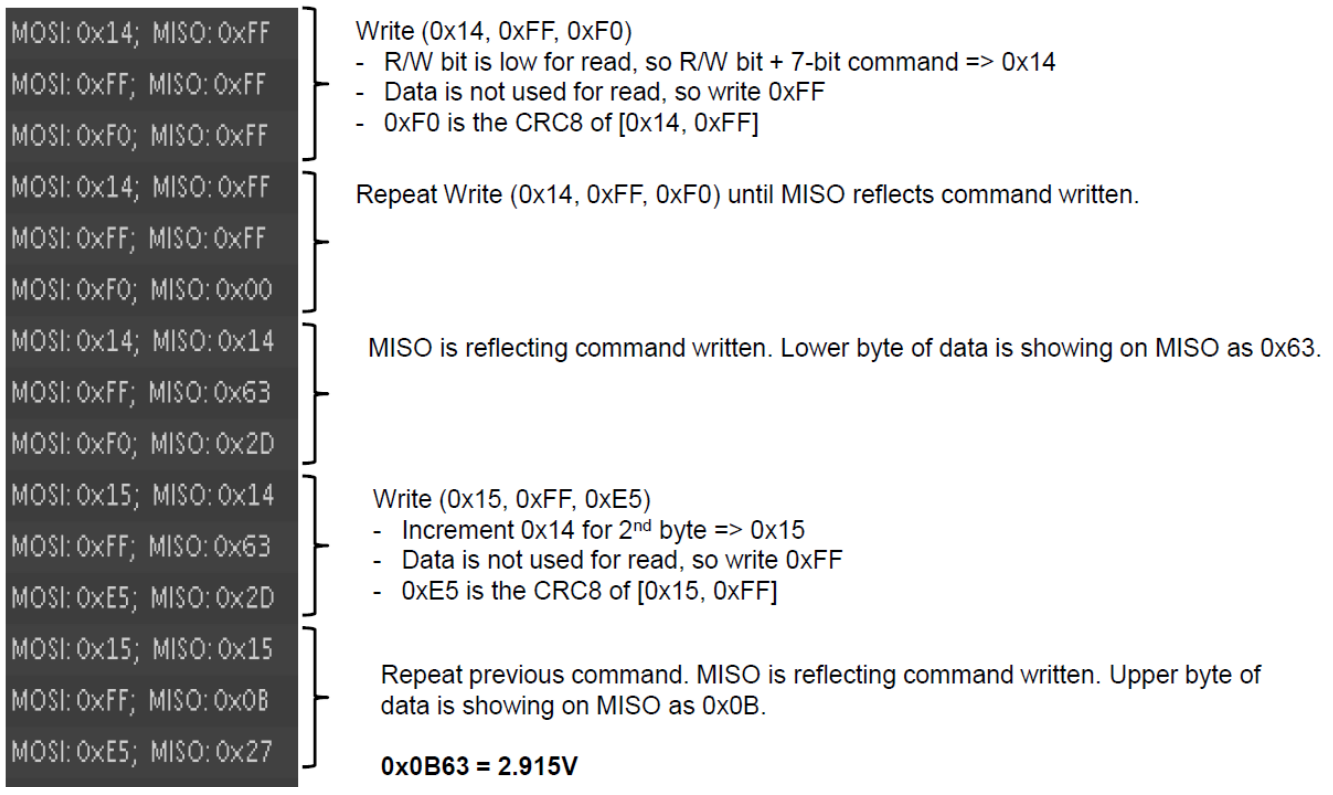
Write (0xE7, 0xF0, 0xF6)
- Increment 0xE6 for 2nd byte => 0xE7
- 0xF0 is the upper byte of the data
- 0xF6 is the CRC8 of [0xF0, 0xF6]

Repeat previous command. MISO now reflects data written.

Figure 5-1. Alarm Enable Direct Command

5.2 Direct Command Example: Cell 1 Voltage - 0x14

Figure 5-2 shows how to read the voltage for Cell 1.



```
MOSI: 0x14; MISO: 0xFF
MOSI: 0xFF; MISO: 0xFF
MOSI: 0xF0; MISO: 0xFF
MOSI: 0x14; MISO: 0xFF
MOSI: 0xFF; MISO: 0xFF
MOSI: 0xF0; MISO: 0x00
MOSI: 0x14; MISO: 0x14
MOSI: 0xFF; MISO: 0x63
MOSI: 0xF0; MISO: 0x2D
MOSI: 0x15; MISO: 0x14
MOSI: 0xFF; MISO: 0x63
MOSI: 0xE5; MISO: 0x2D
MOSI: 0x15; MISO: 0x15
MOSI: 0xFF; MISO: 0x0B
MOSI: 0xE5; MISO: 0x27
```

Write (0x14, 0xFF, 0xF0)
- R/W bit is low for read, so R/W bit + 7-bit command => 0x14
- Data is not used for read, so write 0xFF
- 0xF0 is the CRC8 of [0x14, 0xFF]

Repeat Write (0x14, 0xFF, 0xF0) until MISO reflects command written.

MISO is reflecting command written. Lower byte of data is showing on MISO as 0x63.

Write (0x15, 0xFF, 0xE5)
- Increment 0x14 for 2nd byte => 0x15
- Data is not used for read, so write 0xFF
- 0xE5 is the CRC8 of [0x15, 0xFF]

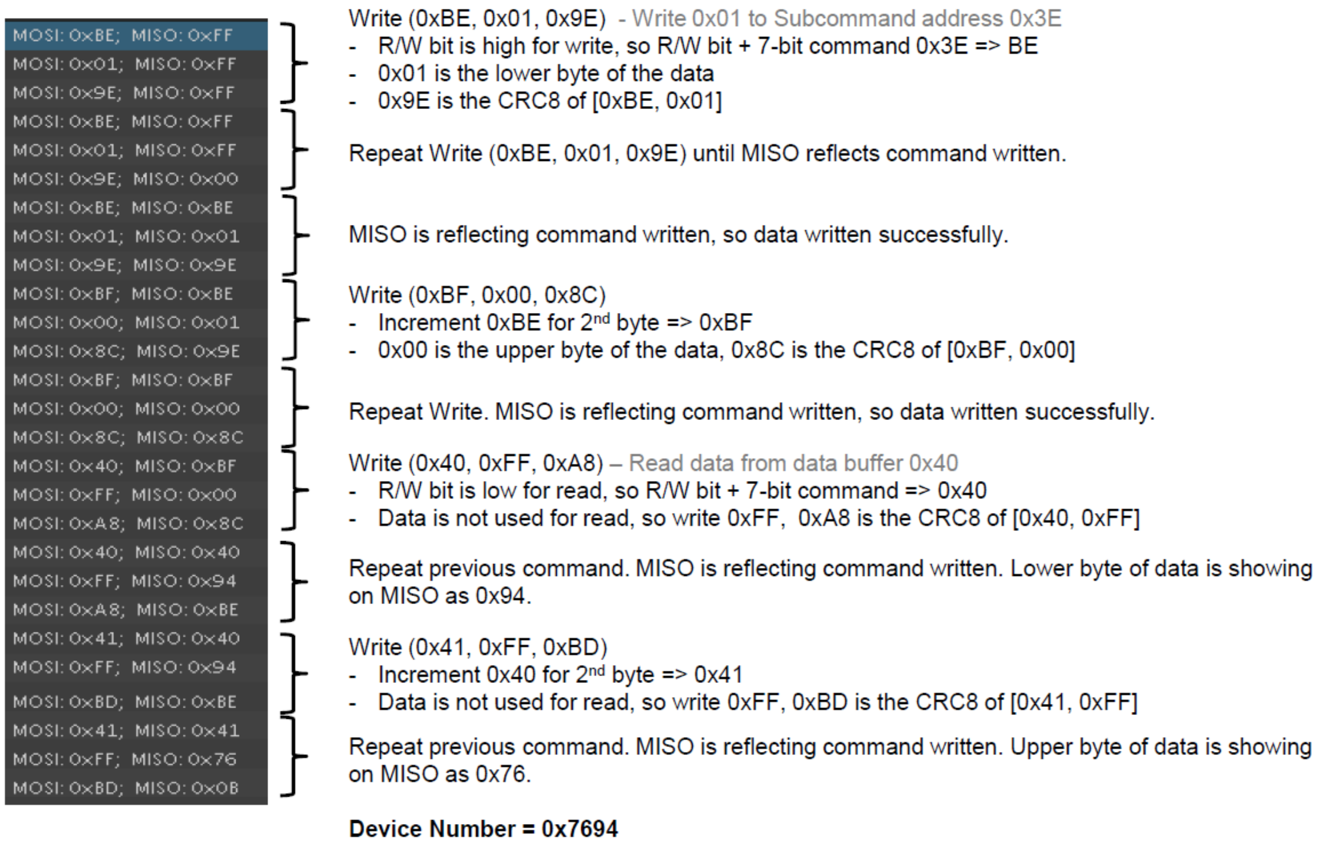
Repeat previous command. MISO is reflecting command written. Upper byte of data is showing on MISO as 0x0B.

0x0B63 = 2.915V

Figure 5-2. Cell1 Voltage Read Direct Command

5.3 Subcommand Example: Device Number - 0x0001

Figure 5-3 shows how to read the Device Number from Subcommand 0x0001.



```
MOSI: 0xBE;  MISO: 0xFF
MOSI: 0x01;  MISO: 0xFF
MOSI: 0x9E;  MISO: 0xFF
MOSI: 0xBE;  MISO: 0xFF
MOSI: 0x01;  MISO: 0xFF
MOSI: 0x9E;  MISO: 0x00
MOSI: 0xBE;  MISO: 0xBE
MOSI: 0x01;  MISO: 0x01
MOSI: 0x9E;  MISO: 0x9E
MOSI: 0xBF;  MISO: 0xBE
MOSI: 0x00;  MISO: 0x01
MOSI: 0x8C;  MISO: 0x9E
MOSI: 0xBF;  MISO: 0xBF
MOSI: 0x00;  MISO: 0x00
MOSI: 0x8C;  MISO: 0x8C
MOSI: 0x40;  MISO: 0xBF
MOSI: 0xFF;  MISO: 0x00
MOSI: 0xA8;  MISO: 0x8C
MOSI: 0x40;  MISO: 0x40
MOSI: 0xFF;  MISO: 0x94
MOSI: 0xA8;  MISO: 0xBE
MOSI: 0x41;  MISO: 0x40
MOSI: 0xFF;  MISO: 0x94
MOSI: 0xBD;  MISO: 0xBE
MOSI: 0x41;  MISO: 0x41
MOSI: 0xFF;  MISO: 0x76
MOSI: 0xBD;  MISO: 0x0B
```

Write (0xBE, 0x01, 0x9E) - Write 0x01 to Subcommand address 0x3E
- R/W bit is high for write, so R/W bit + 7-bit command 0x3E => BE
- 0x01 is the lower byte of the data
- 0x9E is the CRC8 of [0xBE, 0x01]

Repeat Write (0xBE, 0x01, 0x9E) until MISO reflects command written.

MISO is reflecting command written, so data written successfully.

Write (0xBF, 0x00, 0x8C)
- Increment 0xBE for 2nd byte => 0xBF
- 0x00 is the upper byte of the data, 0x8C is the CRC8 of [0xBF, 0x00]

Repeat Write. MISO is reflecting command written, so data written successfully.

Write (0x40, 0xFF, 0xA8) – Read data from data buffer 0x40
- R/W bit is low for read, so R/W bit + 7-bit command => 0x40
- Data is not used for read, so write 0xFF, 0xA8 is the CRC8 of [0x40, 0xFF]

Repeat previous command. MISO is reflecting command written. Lower byte of data is showing on MISO as 0x94.

Write (0x41, 0xFF, 0xBD)
- Increment 0x40 for 2nd byte => 0x41
- Data is not used for read, so write 0xFF, 0xBD is the CRC8 of [0x41, 0xFF]

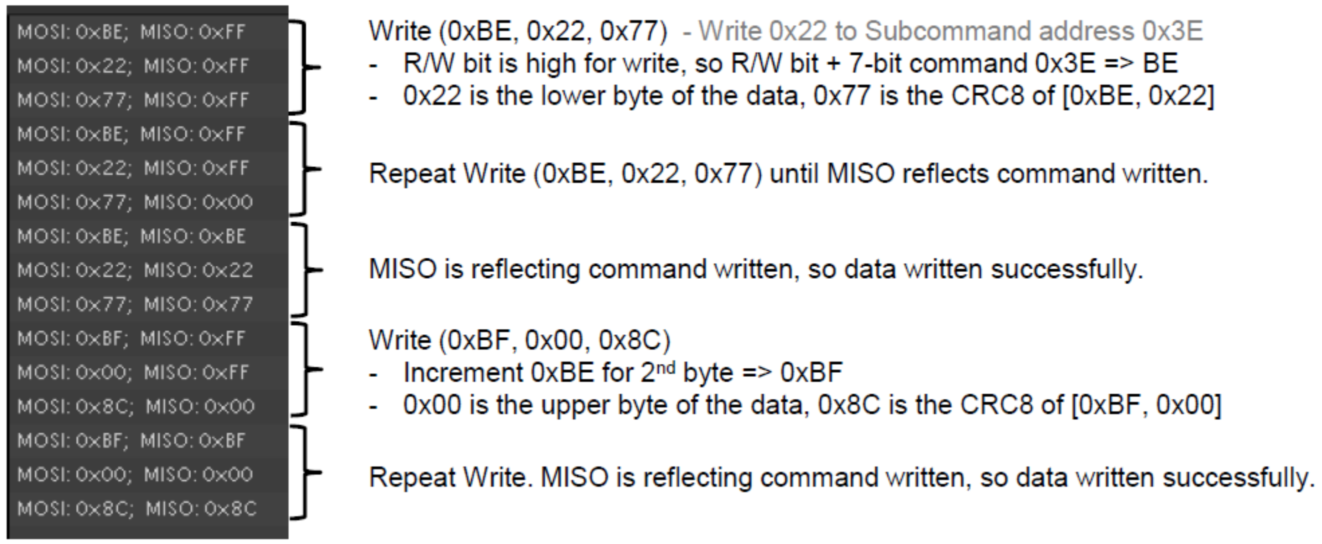
Repeat previous command. MISO is reflecting command written. Upper byte of data is showing on MISO as 0x76.

Device Number = 0x7694

Figure 5-3. Device Number Subcommand

5.4 Subcommand Example: FET_ENABLE - 0x0022

Figure 5-4 shows how to write the FET_ENABLE Subcommand 0x0022.



```
MOSI: 0xBE;  MISO: 0xFF
MOSI: 0x22;  MISO: 0xFF
MOSI: 0x77;  MISO: 0xFF
MOSI: 0xBE;  MISO: 0xFF
MOSI: 0x22;  MISO: 0xFF
MOSI: 0x77;  MISO: 0x00
MOSI: 0xBE;  MISO: 0xBE
MOSI: 0x22;  MISO: 0x22
MOSI: 0x77;  MISO: 0x77
MOSI: 0xBF;  MISO: 0xFF
MOSI: 0x00;  MISO: 0xFF
MOSI: 0x8C;  MISO: 0x00
MOSI: 0xBF;  MISO: 0xBF
MOSI: 0x00;  MISO: 0x00
MOSI: 0x8C;  MISO: 0x8C
```

Write (0xBE, 0x22, 0x77) - Write 0x22 to Subcommand address 0x3E
- R/W bit is high for write, so R/W bit + 7-bit command 0x3E => BE
- 0x22 is the lower byte of the data, 0x77 is the CRC8 of [0xBE, 0x22]

Repeat Write (0xBE, 0x22, 0x77) until MISO reflects command written.

MISO is reflecting command written, so data written successfully.

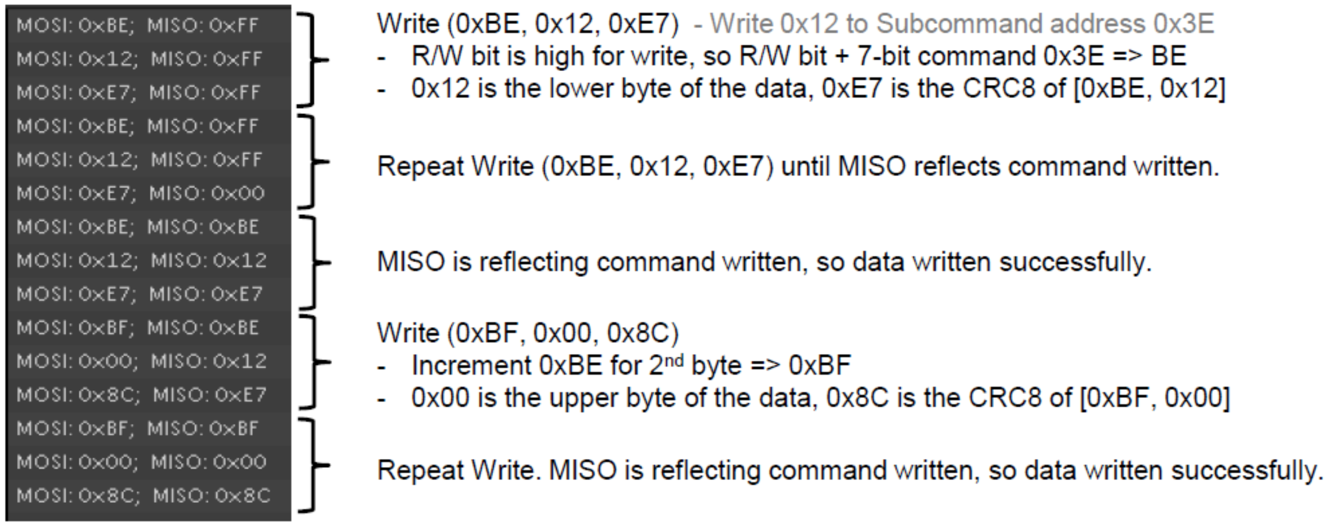
Write (0xBF, 0x00, 0x8C)
- Increment 0xBE for 2nd byte => 0xBF
- 0x00 is the upper byte of the data, 0x8C is the CRC8 of [0xBF, 0x00]

Repeat Write. MISO is reflecting command written, so data written successfully.

Figure 5-4. FET_ENABLE Subcommand

5.5 Subcommand Example: RESET - 0x0012

Figure 5-5 shows how to write the RESET Subcommand 0x0012.



```
MOSI: 0xBE;  MISO: 0xFF
MOSI: 0x12;  MISO: 0xFF
MOSI: 0xE7;  MISO: 0xFF
MOSI: 0xBE;  MISO: 0xFF
MOSI: 0x12;  MISO: 0xFF
MOSI: 0xE7;  MISO: 0x00
MOSI: 0xBE;  MISO: 0xBE
MOSI: 0x12;  MISO: 0x12
MOSI: 0xE7;  MISO: 0xE7
MOSI: 0xBF;  MISO: 0xBE
MOSI: 0x00;  MISO: 0x12
MOSI: 0x8C;  MISO: 0xE7
MOSI: 0xBF;  MISO: 0xBF
MOSI: 0x00;  MISO: 0x00
MOSI: 0x8C;  MISO: 0x8C
```

Write (0xBE, 0x12, 0xE7) - Write 0x12 to Subcommand address 0x3E
- R/W bit is high for write, so R/W bit + 7-bit command 0x3E => BE
- 0x12 is the lower byte of the data, 0xE7 is the CRC8 of [0xBE, 0x12]

Repeat Write (0xBE, 0x12, 0xE7) until MISO reflects command written.

MISO is reflecting command written, so data written successfully.

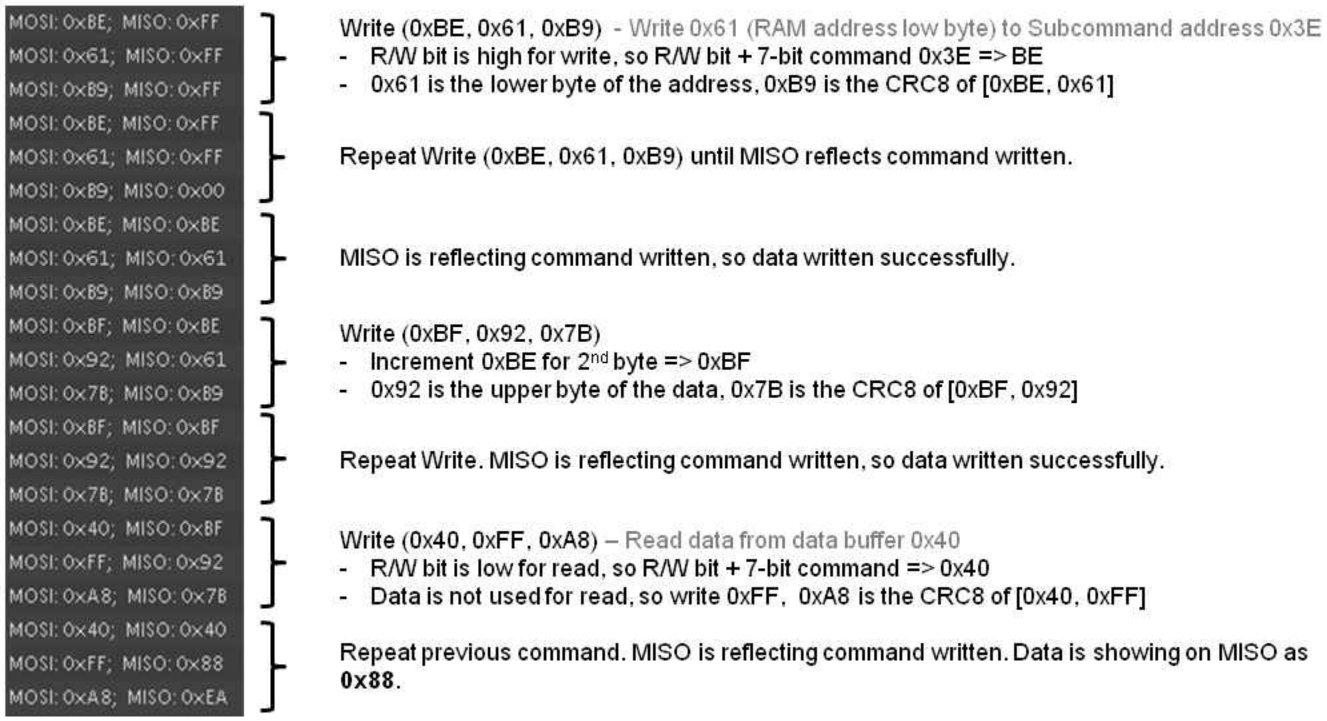
Write (0xBF, 0x00, 0x8C)
- Increment 0xBE for 2nd byte => 0xBF
- 0x00 is the upper byte of the data, 0x8C is the CRC8 of [0xBF, 0x00]

Repeat Write. MISO is reflecting command written, so data written successfully.

Figure 5-5. RESET Subcommand

5.6 RAM Register Read Example: Enabled Protections A

Figure 5-6 shows how to read RAM register ***Enabled Protections A***. The address is 0x9261.



```
MOSI: 0xBE;  MISO: 0xFF
MOSI: 0x61;  MISO: 0xFF
MOSI: 0xB9;  MISO: 0xFF
MOSI: 0xBE;  MISO: 0xFF
MOSI: 0x61;  MISO: 0xFF
MOSI: 0xB9;  MISO: 0x00
MOSI: 0xBE;  MISO: 0xBE
MOSI: 0x61;  MISO: 0x61
MOSI: 0xB9;  MISO: 0xB9
MOSI: 0xBF;  MISO: 0xBE
MOSI: 0x92;  MISO: 0x61
MOSI: 0x7B;  MISO: 0xB9
MOSI: 0xBF;  MISO: 0xBF
MOSI: 0x92;  MISO: 0x92
MOSI: 0x7B;  MISO: 0x7B
MOSI: 0x40;  MISO: 0xBF
MOSI: 0xFF;  MISO: 0x92
MOSI: 0xA8;  MISO: 0x7B
MOSI: 0x40;  MISO: 0x40
MOSI: 0xFF;  MISO: 0x88
MOSI: 0xA8;  MISO: 0xEA
```

Write (0xBE, 0x61, 0xB9) - Write 0x61 (RAM address low byte) to Subcommand address 0x3E
- R/W bit is high for write, so R/W bit + 7-bit command 0x3E => BE
- 0x61 is the lower byte of the address, 0xB9 is the CRC8 of [0xBE, 0x61]

Repeat Write (0xBE, 0x61, 0xB9) until MISO reflects command written.

MISO is reflecting command written, so data written successfully.

Write (0xBF, 0x92, 0x7B)
- Increment 0xBE for 2nd byte => 0xBF
- 0x92 is the upper byte of the data, 0x7B is the CRC8 of [0xBF, 0x92]

Repeat Write. MISO is reflecting command written, so data written successfully.

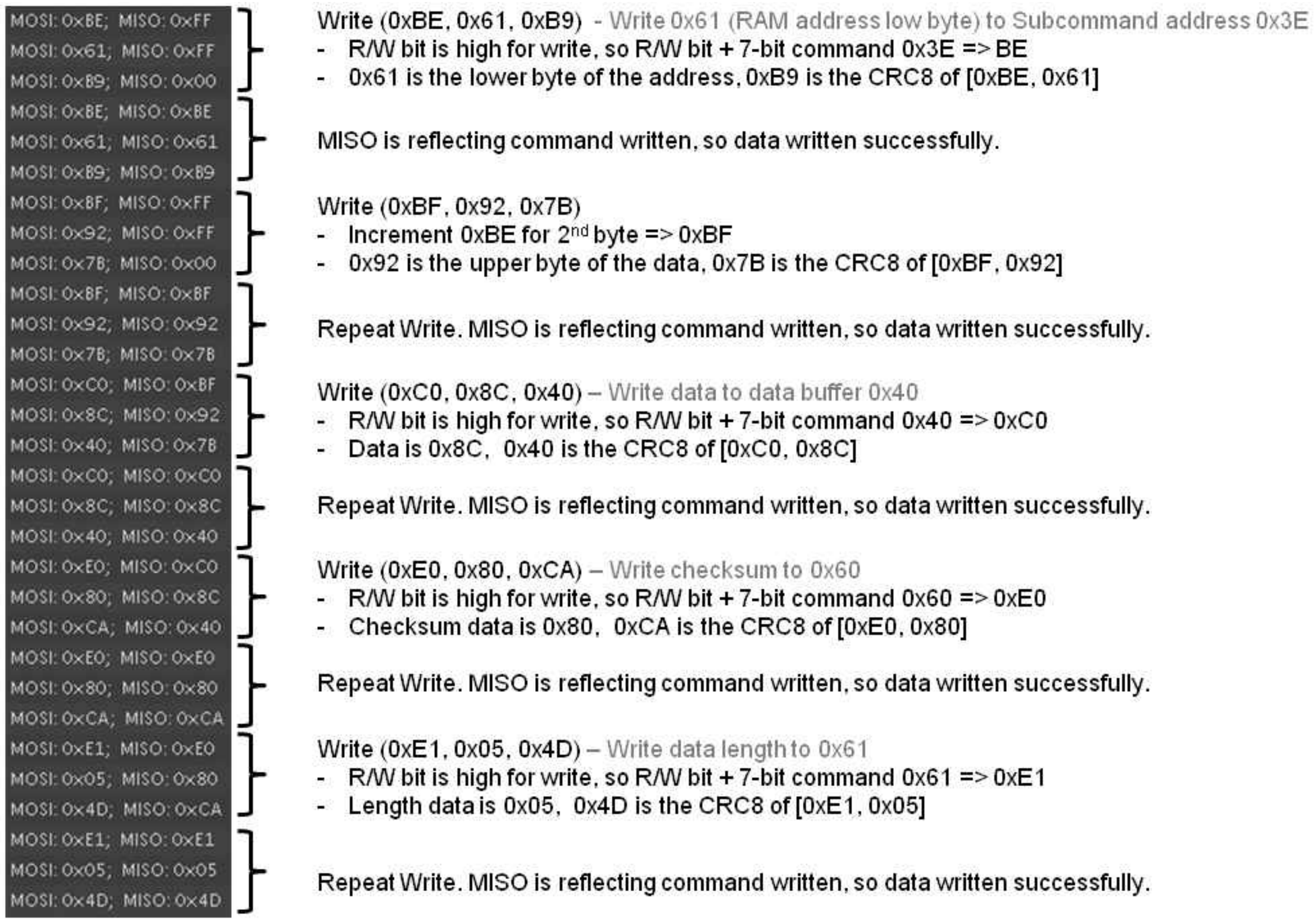
Write (0x40, 0xFF, 0xA8) – Read data from data buffer 0x40
- R/W bit is low for read, so R/W bit + 7-bit command => 0x40
- Data is not used for read, so write 0xFF, 0xA8 is the CRC8 of [0x40, 0xFF]

Repeat previous command. MISO is reflecting command written. Data is showing on MISO as **0x88**.

Figure 5-6. Read Enabled Protections A

5.7 RAM Register Write Example: Enabled Protections A

Figure 5-7 shows how to write RAM register ***Enabled Protections A*** with a value of 0x8C.



```
MOSI: 0xBE; MISO: 0xFF
MOSI: 0x61; MISO: 0xFF
MOSI: 0xB9; MISO: 0x00
```

Write (0xBE, 0x61, 0xB9) - Write 0x61 (RAM address low byte) to Subcommand address 0x3E
- R/W bit is high for write, so R/W bit + 7-bit command 0x3E => BE
- 0x61 is the lower byte of the address, 0xB9 is the CRC8 of [0xBE, 0x61]

```
MOSI: 0xBE; MISO: 0xBE
MOSI: 0x61; MISO: 0x61
MOSI: 0xB9; MISO: 0xB9
```

MISO is reflecting command written, so data written successfully.

```
MOSI: 0xBF; MISO: 0xFF
MOSI: 0x92; MISO: 0xFF
MOSI: 0x7B; MISO: 0x00
```

Write (0xBF, 0x92, 0x7B)
- Increment 0xBE for 2nd byte => 0xBF
- 0x92 is the upper byte of the data, 0x7B is the CRC8 of [0xBF, 0x92]

```
MOSI: 0xBF; MISO: 0xBF
MOSI: 0x92; MISO: 0x92
MOSI: 0x7B; MISO: 0x7B
```

Repeat Write. MISO is reflecting command written, so data written successfully.

```
MOSI: 0xC0; MISO: 0xBF
MOSI: 0x8C; MISO: 0x92
MOSI: 0x40; MISO: 0x7B
```

Write (0xC0, 0x8C, 0x40) – Write data to data buffer 0x40
- R/W bit is high for write, so R/W bit + 7-bit command 0x40 => 0xC0
- Data is 0x8C, 0x40 is the CRC8 of [0xC0, 0x8C]

```
MOSI: 0xC0; MISO: 0xC0
MOSI: 0x8C; MISO: 0x8C
MOSI: 0x40; MISO: 0x40
```

Repeat Write. MISO is reflecting command written, so data written successfully.

```
MOSI: 0xE0; MISO: 0xC0
MOSI: 0x80; MISO: 0x8C
MOSI: 0xCA; MISO: 0x40
```

Write (0xE0, 0x80, 0xCA) – Write checksum to 0x60
- R/W bit is high for write, so R/W bit + 7-bit command 0x60 => 0xE0
- Checksum data is 0x80, 0xCA is the CRC8 of [0xE0, 0x80]

```
MOSI: 0xE0; MISO: 0xE0
MOSI: 0x80; MISO: 0x80
MOSI: 0xCA; MISO: 0xCA
```

Repeat Write. MISO is reflecting command written, so data written successfully.

```
MOSI: 0xE1; MISO: 0xE0
MOSI: 0x05; MISO: 0x80
MOSI: 0x4D; MISO: 0xCA
```

Write (0xE1, 0x05, 0x4D) – Write data length to 0x61
- R/W bit is high for write, so R/W bit + 7-bit command 0x61 => 0xE1
- Length data is 0x05, 0x4D is the CRC8 of [0xE1, 0x05]

```
MOSI: 0xE1; MISO: 0xE1
MOSI: 0x05; MISO: 0x05
MOSI: 0x4D; MISO: 0x4D
```

Repeat Write. MISO is reflecting command written, so data written successfully.

Figure 5-7. Write Enabled Protections A

6 Simple Code Examples

The following example code is written in Python and designed to communicate to the BQ769x2 device from a PC through an EV2400 module or through the USB connector on the BQ76942 or BQ76952 Evaluation Module. The code shows the creation of simple I2C Read and Write functions, a DataRAM_Read function, (which can also be used to execute subcommands since these follow the same format), and a DataRAM_Write function that shows the calculation of checksum and length. The main section of the code goes through all of the examples covered in the first three sections of this document.

This simple code example is intended to illustrate the basic command structure for I2C commands. Microcontroller code examples are also available for I2C and SPI. The link to the microcontroller code is provided in Section 7.

```
'''
/* BQ769x2 example Program demonstrates examples for direct commands, subcommands, and writing /
reading from device RAM.
'''
import pywinusb
import bqcomm
import sys
import time
from time import sleep
import sets
I2C_ADDR = 0x10   # BQ769x2 slave address
numCells = 10     # Set to 10 for BQ76942
####################################################
## Check to see if EV2400 is connected
####################################################
try:
    a = bqcomm.Adapter()  # This will use the first found Aardvark or EV2400
except:
    print "No EV2400 Available"
    sys.exit(1)
######################################################
## Define some command functions
######################################################
def I2C_Read(device_addr, reg_addr, length):
    '''
    Uses global I2C address and returns value read
    '''
    try:
        value = a.i2c_read_block(device_addr, reg_addr, length)
    except:
        print "Nack received"
        return
    return value
def I2C_Write(device_addr, reg_addr, block):
    '''
    Uses global I2C address
    '''
    try:
        a.i2c_write_block(device_addr, reg_addr, block)
    except:
        print "Nack received"
    return
def DataRAM_Read(addr, length):
    '''
    Write address location to 0x3E and read back from 0x40
    Used to read configuration registers and for subcommands
    '''
    addressBlock = [(addr%256), (addr/256)]
    I2C_Write(I2C_ADDR, 0x3E, addressBlock)
    value = I2C_Read(I2C_ADDR, 0x40,length)
    return value
def DataRAM_Write(addr, block):
    '''
    Write address location to 0x3E and Checksum,length to 0x60
    Used to write configuration registers
    '''
    addressBlock = [(addr%256), (addr/256)]
    wholeBlock = addressBlock + block
    I2C_Write(I2C_ADDR, 0x3E, wholeBlock)              # Write Data Block
    # Write Data Checksum and length to 0x60, required for RAM writes
    I2C_Write(I2C_ADDR, 0x60, [~sum(wholeBlock) & 0xff, len(wholeBlock)+2])
    return
```

```
def crc8(b,key):
    crc = 0
    ptr = 0
    for j in range(len(b),0,-1):
        for k in range(8):
            i = 128 / (2**k)
            if ((crc & 0x80) != 0):
                crc = crc * 2
                crc = crc ^ key
            else:
                crc = crc * 2
            if ((b[ptr] & i) != 0):
                crc = crc ^ key
        ptr = ptr + 1
    return crc
##########################################
#  Start of Main Script
##########################################
################ Direct Command Examples ################
#Write Alarm Enable to 0xF082
I2C_Write(I2C_ADDR, 0x66, [0x82, 0xF0])
#Read Voltage on Cell #1
result = I2C_Read(I2C_ADDR, 0x14, 2)
print "Cell 1 = ", (result[1]*256 + result[0]), " mV"
#Read Internal Temperature
result = I2C_Read(I2C_ADDR, 0x68, 2)
print "Internal Temp = ", ((result[1]*256 + result[0])/10 - 273.15), "degrees C"
#Read CC2 Current Measurement
result = I2C_Read(I2C_ADDR, 0x3A, 2)
print "CC2 = ", (result[1]*256 + result[0]), " mA"
################ Subcommand Examples ################
#Read Device Number
b = DataRAM_Read(0x0001,6)
print "Device_Number = " '{0:04X}'.format(b[1]*256+b[0])
#Read Manufacturing Status
b = DataRAM_Read(0x0057,2)
print "Manufacturing Status = " '{0:04X}'.format(b[0]+256*b[1])
## Command-only Subcomands ##
#FET_ENABLE
I2C_Write(I2C_ADDR, 0x3E, [0x22, 0x00])
#RESET - returns device to default settings
I2C_Write(I2C_ADDR, 0x3E, [0x12, 0x00])
sleep(1)
############ Reading and Writing to RAM Registers ##########
# Read 'Enabled Protections A' RAM register 0x9261
b = DataRAM_Read(0x9261,1)
print "Enabled Protections A = 0x" '{0:02X}'.format(b[0])
#Set CONFIG_UPDATE Mode (RAM registers should be written while in
#CONFIG_UPDATE mode and will take effect after exiting CONFIG_UPDATE mode
I2C_Write(I2C_ADDR, 0x3E, [0x90, 0x00])
#Write to 'Enabled Protections A' RAM register to enable CUV protection
DataRAM_Write(0x9261, [0x8C])
#Write to 'VCell Mode' RAM register to configure for a 9-cell battery
DataRAM_Write(0x9304, [0x03, 0x7f])
#Exit CONFIG_UPDATE Mode
I2C_Write(I2C_ADDR, 0x3E, [0x92, 0x00])
# CRC8 Example Calculation
TestValue = [0x10, 0x14, 0x11, 0x68]
crcKey = 0x107
check = 0xff & crc8(TestValue,crcKey)
print "crc8 check = 0x" '{0:02X}'.format(check)
```

The output from running the example Python script on a BQ76942 Evaluation Module follows.

```
Cell 1 =  3700  mV
Internal Temp =  25.05 degrees C
CC2 = 7  mA
Device_Number = 7694
Manufacturing Status = 0040
Enabled Protections A = 0x88
crc8 check = 0x33
```

7 References

- Texas Instruments: *BQ76952 3S-16S Battery Monitor and Protector Data Sheet*
- Texas Instruments: *BQ76942 3S-160S Battery Monitor and Protector Data Sheet*
- Texas Instruments: *BQ76952 Technical Reference Manual*
- Texas Instruments: *BQ76942 Technical Reference Manual*

8 Revision History

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

Changes from Revision A (October 2020) to Revision B (August 2021) **Page**

- Updated the numbering format for tables, figures and cross-references throughout the document....................3
- Updates were made in Section 2...6
- Updates were made in Section 5...13
- Changes were made in Section 6..19