Application Report
SLUAA11–February 2020

BQ76942, BQ76952 Software Development Guide

Matt Sunna

ABSTRACT

This application report provides examples of communication packets and sequences for the BQ769x2 device family of battery monitors. 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 *BQ76942 High Accuracy 3-s to 10-s High Accuracy Battery Monitor and Protector for Li-Ion, Li- Polymer, and LiFePO4 Battery Packs* or *BQ76952 3-s to 16-s High Accuracy Battery Monitor and Protector for Li-Ion, Li-Polymer, and LiFePO4 Battery Packs* data sheets. 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 1.3.97 or above of BQStudio is required.

The BQ769x2 device family integrates three different communication interfaces - I^2C, SPI, and single-wire HDQ. The I^2C and SPI interfaces include an optional CRC check. See the device data sheet for the full list of options. The examples in this document use the I^2C interface.

Contents

List of Figures

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 data sheet. The format for a direct command is shown in the following examples.

Alarm Enable - 0x66

The following example shows the Alarm Enable command which 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])*.

0x66	Alarm Enable	Hex	H2	Sealed: R/W Unsealed: R/W Full Access: R/W	Mask for *Alarm Status()*. Can be written to change during operation to change which alarm sources are enabled. The default value of this parameter is set by ***Settings:Alarm:Default Alarm Mask***. Bit descriptions can be found in *Alarm Enable(): 0x66* .

Figure 1. Alarm Enable Data Sheet Description

Figure 2. Captured I2C Waveform for Setting Alarm Enable to 0xF082

Cell 1 Voltage - 0x14

This example 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.

0x14	Cell 1 Voltage	mV	I2	Sealed: R Unsealed: R Full Access: R	16-bit voltage on cell 1

Figure 3. Cell 1 Voltage Data Sheet Description

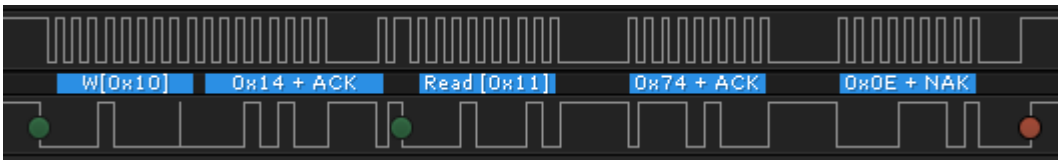

Figure 4. Captured I2C Waveform for Cell 1 Voltage Reading

Internal Temperature - 0x68

This example 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.

0x68	Int Temperature	0.1K	I2	Sealed: R Unsealed: R Full Access: R	This is the most recent measured internal die temperature.

Figure 5. Internal Temperature Data Sheet Description

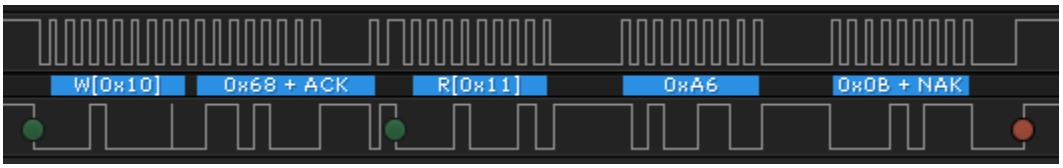

Figure 6. Captured I2C Waveform for Internal Temperature Reading

CC2 Current

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

0x3A	CC2 Current	user A	I2	Sealed: R Unsealed: R Full Access: R	16-bit CC2 current

Figure 7. CC2 Current Data Sheet Description

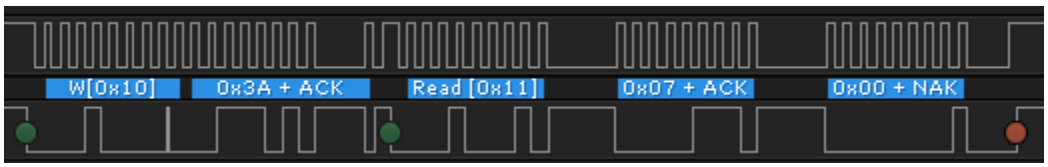

Figure 8. Captured I2C Waveform for CC2 Current Reading

The Command Sequence module in the BQStudio software enables the user 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.

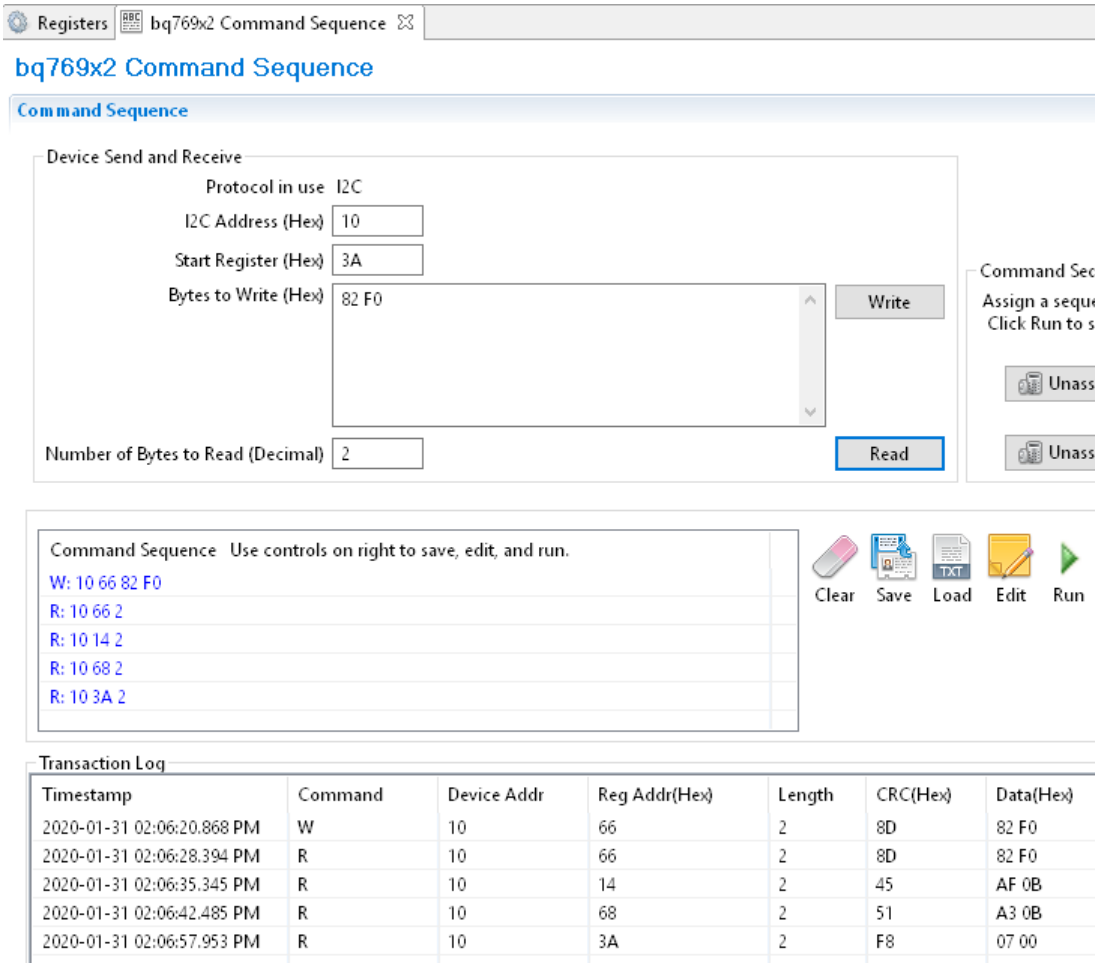

Figure 9. BQStudio Example Showing Execution of Multiple Direct Commands

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 10).

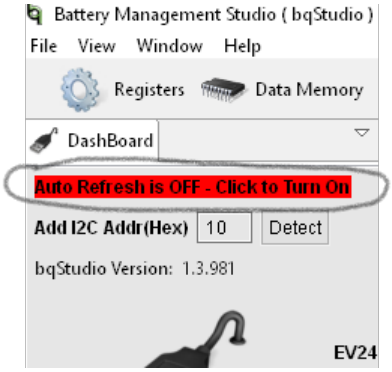

Figure 10. Auto Refresh Disabled

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.

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).

0x0001	DEVICE_NUMBER	Sealed: R Unsealed: R Full Access: R	0	Device Number	Hex	U2	Reports the device number that identifies the product. The data is returned in little-endian format.

Figure 11. DEVICE_NUMBER Data Sheet Description

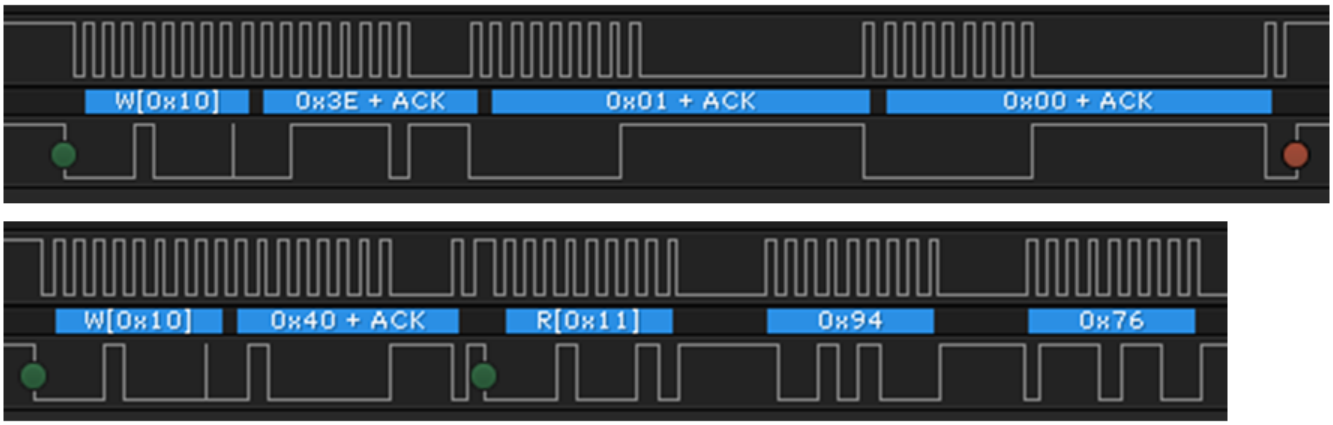

Figure 12. Captured I2C Waveform for DEVICE_NUMBER Subcommand

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.

0x0057	MANUFACTURING STATUS	Sealed: R Unsealed: R Full Access: R	0	Manufacturing Status	Hex	H2	Provides flags for use during manufacturing. Bit descriptions can be found in *0x0057 MANUFACTURINGSTATUS[0–1]: Manufacturing Status()* .

Figure 13. MANUFACTURING_STATUS Data Sheet Description

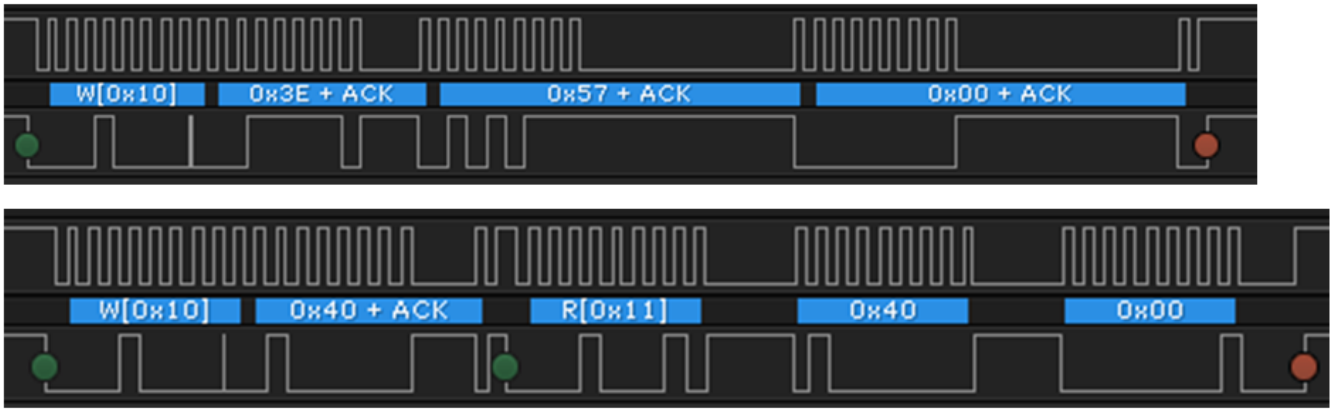

Figure 14. Captured I2C Waveform for MANUFACTURING_STATUS Subcommand

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.

0x0022	FET_ENABLE	Sealed: — Unsealed: W Full Access: W	Toggle FET_EN in Manufacturing Status. FET_EN = 0 means FET Test mode. FET_EN = 1 means Firmware FET Control.

Figure 15. FET_ENABLE Data Sheet Description

Figure 16. Captured I2C Waveform for FET_ENABLE Subcommand

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.

0x0012	RESET	Sealed: — Unsealed: W Full Access: W	Resets the device.

Figure 17. RESET Subcommand Data Sheet Description

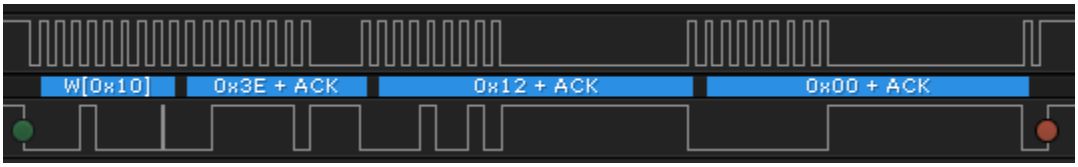

Figure 18. Captured I2C Waveform of RESET Subcommand

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

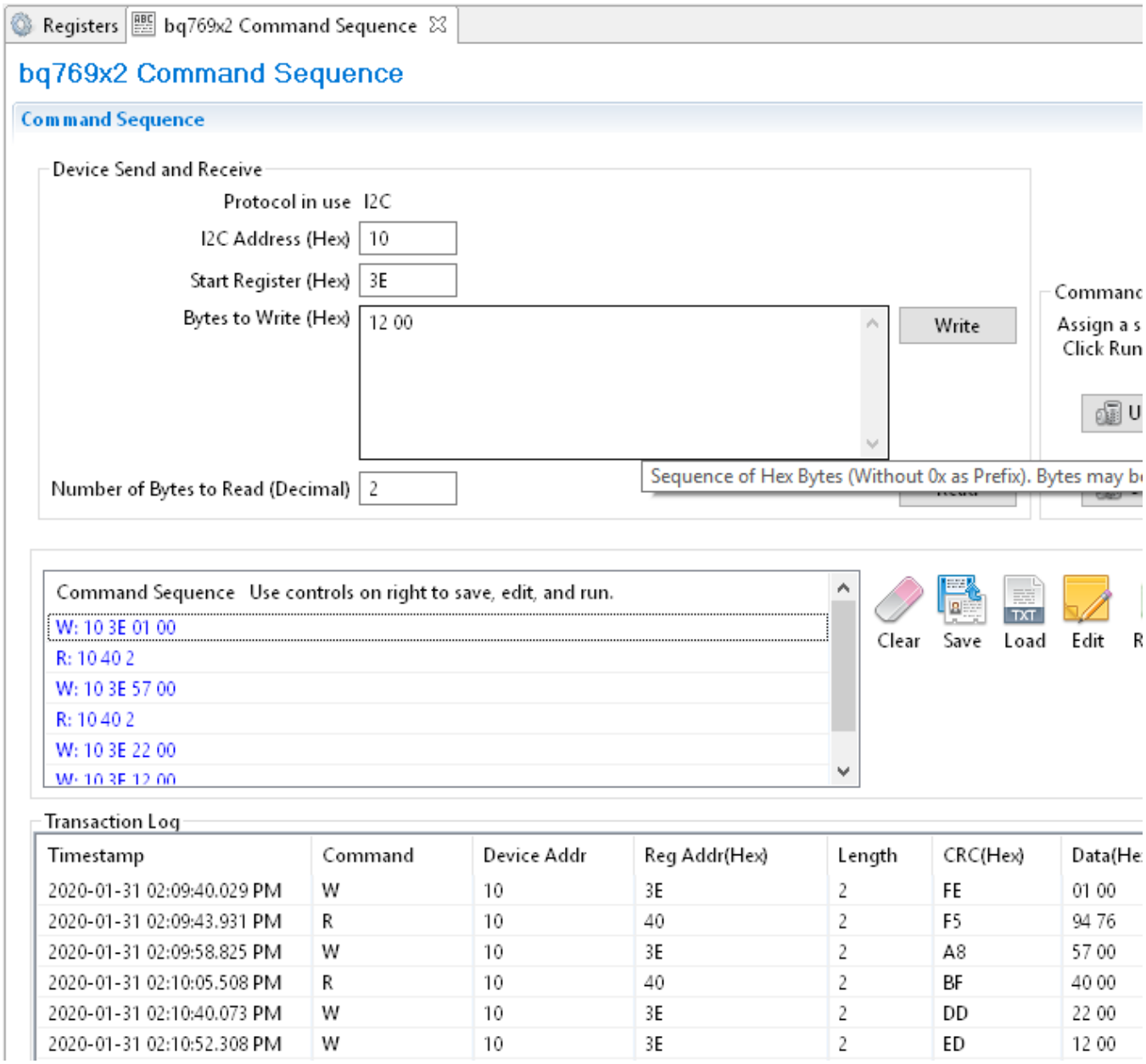

Figure 19. BQStudio Example Showing Execution of Multiple Subcommands

3 Reading and Writing RAM Registers

A full view of registers in RAM can be found in the device data sheet and also in the Data Memory screen of BQStudio. 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 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.

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 0x9243.

Class	Subclass	Address	Name	Type	Min Value	Max Value	Default	Units
Settings	Protection	0x9243	Enabled Protections A	U1	0x00	0xFF	0x88	Hex

Figure 20. Enabled Protections A Data Sheet Description

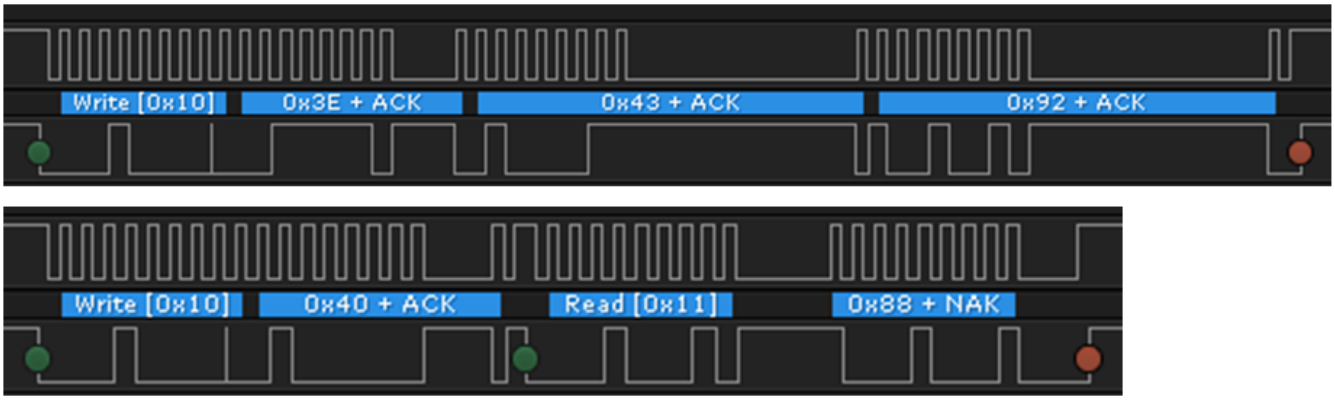

Figure 21. Captured I2C Waveform for Reading 'Enabled Protections A' Register

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.

0x0090	SET_CFGUPDATE	Sealed: — Unsealed: W Full Access: W	Enters CONFIG_UPDATE mode.
0x0092	EXIT_CFGUPDATE	Sealed: W Unsealed: W Full Access: W	Exits CONFIG_UPDATE mode. This also clears the *Battery Status()[POR]* and *Battery Status()[WD]* bits.

Figure 22. SET_CFGUPDATE and EXIT_CFGUPDATE Data Sheet Description

Figure 23. Captured I2C Waveform for SET_CFGUPATE

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 0x9243. The checksum is calculated on the address and data (0x43, 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.

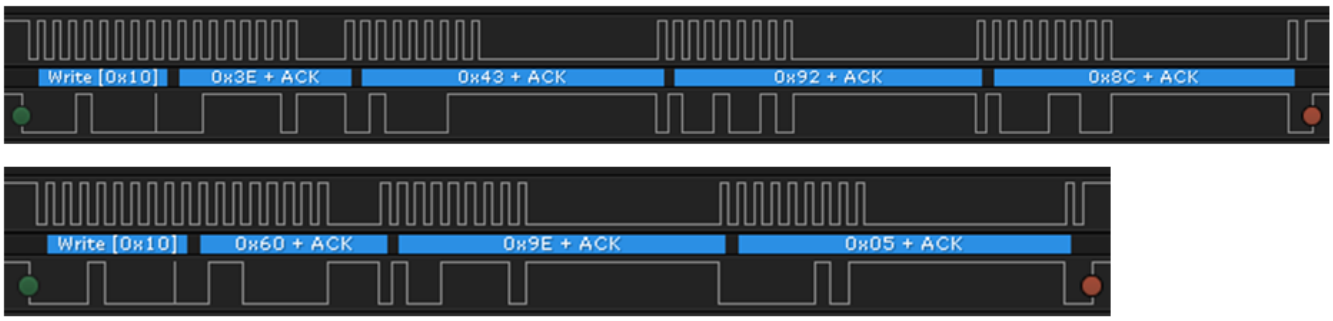

Figure 24. Captured Waveform for Writing to Enabled Protections A

Write 'VCell Mode'

Next, write to the ***VCell Mode*** register to configure the device for 9 cells. The following example writes 0x037F to 0x92EA and then writes the new checksum and length to 0x60/0x61.

Class	Subclass	Address	Name	Type	Min Value	Max Value	Default	Units
Settings	Configuration	0x92EA	Vcell Mode	H2	0x0000	0x03FF	0x001F	Hex

Figure 25. VCell Mode Data Sheet Description

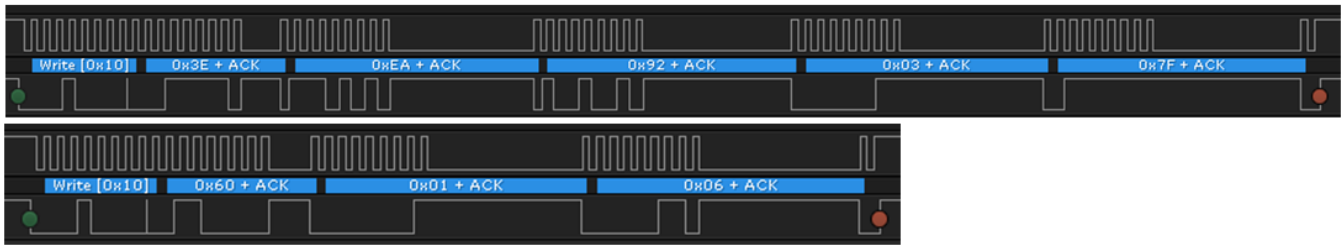

Figure 26. Captured I2C Waveform for Writing VCell Mode

Exit CONFIG_UPDATE Mode

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

Figure 27. 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.

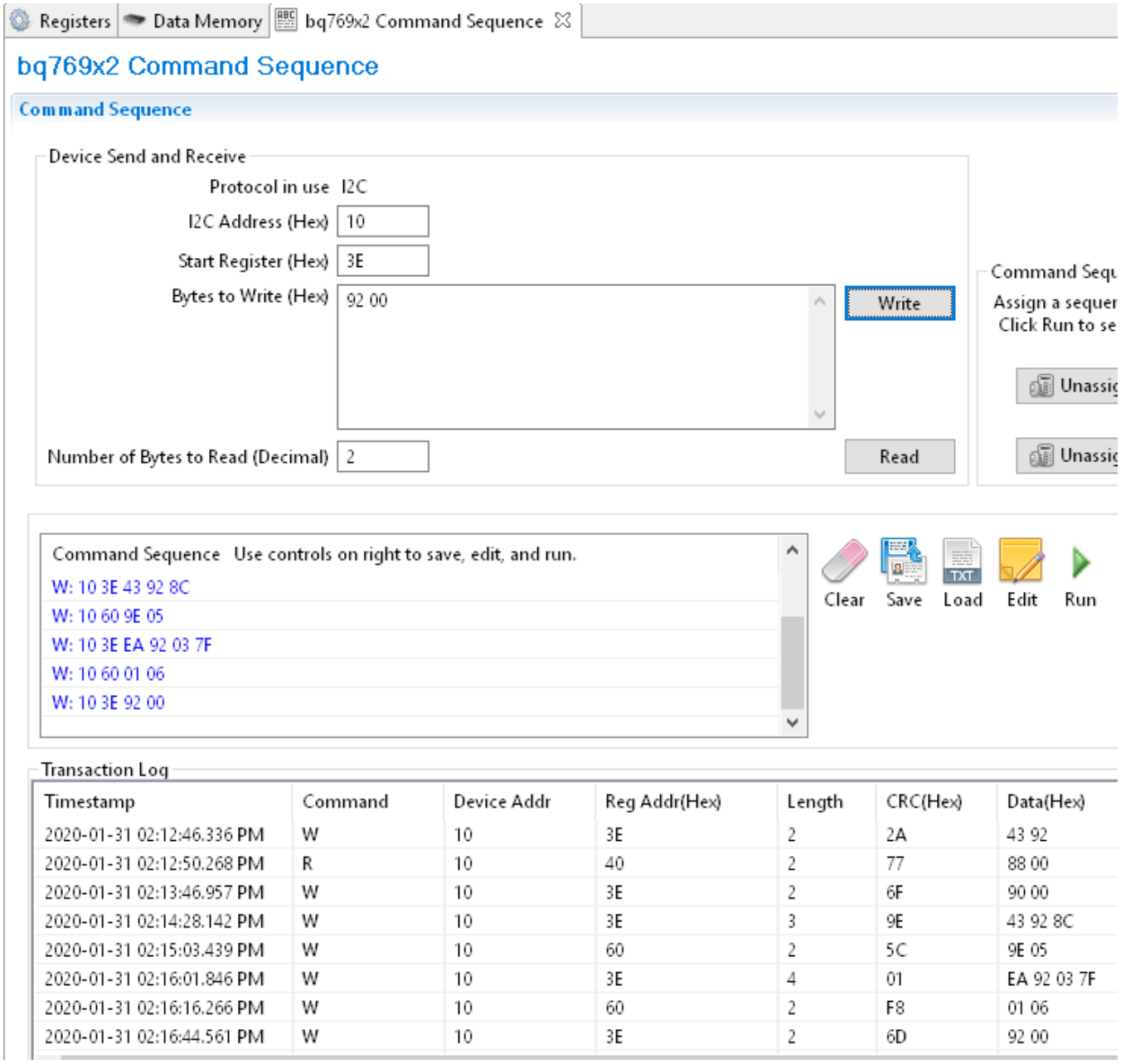

Figure 28. BQStudio Example Showing Execution of RAM Register Reads and Writes

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.

Figure 29. Captured I2C Waveform for FET_ENABLE Subcommand with CRC

Figure 30. Captured I2C Waveform for VCell 1 Command with CRC

5 Simple Code Example

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.

```
'''
/* BQ76942 / BQ76952 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 or Aardvark is connected
#####################################################
try:
    a = bqcomm.Adapter()  # This will use the first found Aardvark or EV2400
except:
    print "No EV2400 Available"
    sys.exit(1)

# How to Write
# a.i2c_write_block(0xaa, 0x3e, [0x0d, 0x00])
# How to Read
# a.i2c_read_block(0xaa, 0x40, 0x36)


#####################################################
## 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 dataflssh 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 dataflssh
        Add 2 to length for Rev A0 of Maximo2
        '''
        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 0x9243
b = DataRAM_Read(0x9243,1)
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(0x9243, [0x8C])

#Write to 'VCell Mode' RAM register to configure for a 9-cell battery
DataRAM_Write(0x92EA, [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
```

6 References

- Texas Instruments, *BQ76942 High Accuracy 3-s to 10-s High Accuracy Battery Monitor and Protector for Li-Ion, Li- Polymer, and LiFePO4 Battery Packs Data Sheet*
- Texas Instruments, *BQ76952 3-s to 16-s High Accuracy Battery Monitor and Protector for Li-Ion, Li-Polymer, and LiFePO4 Battery Packs Data Sheet*
- Texas Instruments, *BQ76942 Evaluation Module User's Guide*
- Texas Instruments, *BQ76952 Evaluation Module User's Guide*

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