User's Guide

BQ7690x Software Development Guide

Andria McIntyre

ABSTRACT

This application note provides examples of communication packets and sequences for the BQ7690x device family of battery monitors (which includes the BQ76905 and BQ76907). Examples include bit-transaction details of direct commands, sub-commands, 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. BQSTUDIO software is also used for many examples and offers a convenient way to view all of the device registers. For the BQ7690x device family, version 1.3.115 or above of BQStudio is required.

The BQ7690x device family integrates the I^2C communication interfaces. The I^2C interface includes an optional CRC check. For the full list of options, see the device-specific data sheet. This document covers many examples using the I^2C interface.

Table of Contents

Trademarks

1 Direct Commands

A complete list of direct commands can be found in the 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 0xC200. In Figure 1-1, the setting is changed to 0x0060. The data is in little endian format. The device address for the BQ7690x 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 can be *I2C_Write(0x10, 0x66, [0x60, 0x00])*.

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.

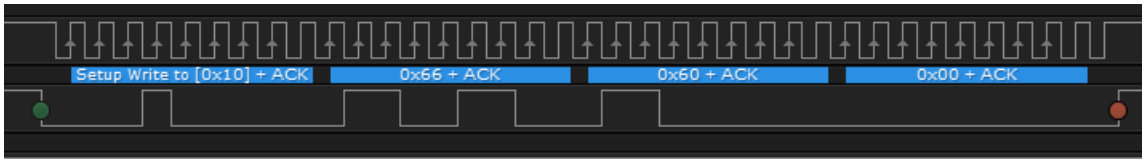


Figure 1-1. Captured I2C Waveform for Setting Alarm Enable to 0x0060

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 Figure 1-2, the 16-bit Cell 1 voltage reads 0x0C0C, which corresponds to 3,084 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

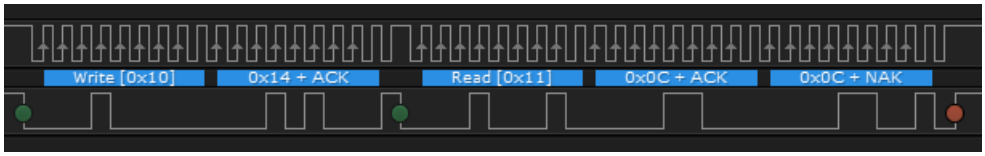


Figure 1-2. Captured I2C Waveform for Cell 1 Voltage Reading

1.3 Internal Temperature - 0x28

Table 1-3 shows how to read the internal temperature sensor. The units for the 16-bit temperature sensor reading are in 0.1 °C. In Figure 1-3, the reading of 0x001D represents a decimal value of 29, which is 29°C.

Table 1-3. Internal Temperature Command Description

Command	Name	Units	Type	Description
0x28	Int Temperature	0.1 °C	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. In Figure 1-4, the current reading of 0x022D represents a decimal value of 557, which is 557 mA.

Table 1-4. CC2 Current Command Description

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

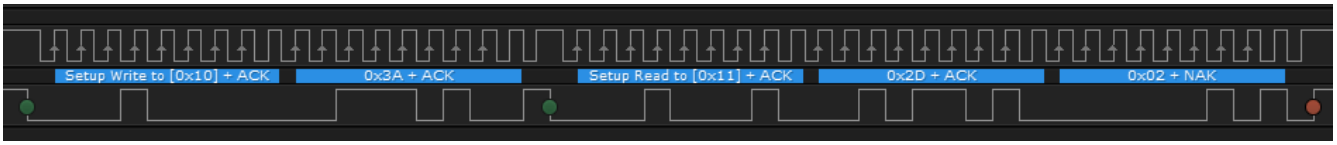


Figure 1-4. Captured I2C Waveform for CC2 Current Reading

1.5 Direct Command Summary

The Command Sequence module in the BQStudio software enables you to write a custom command frame. This tool can also be used to create and save command sequences. The *Transaction Log* in Figure 1-5 shows all of the commands that have been covered so far.

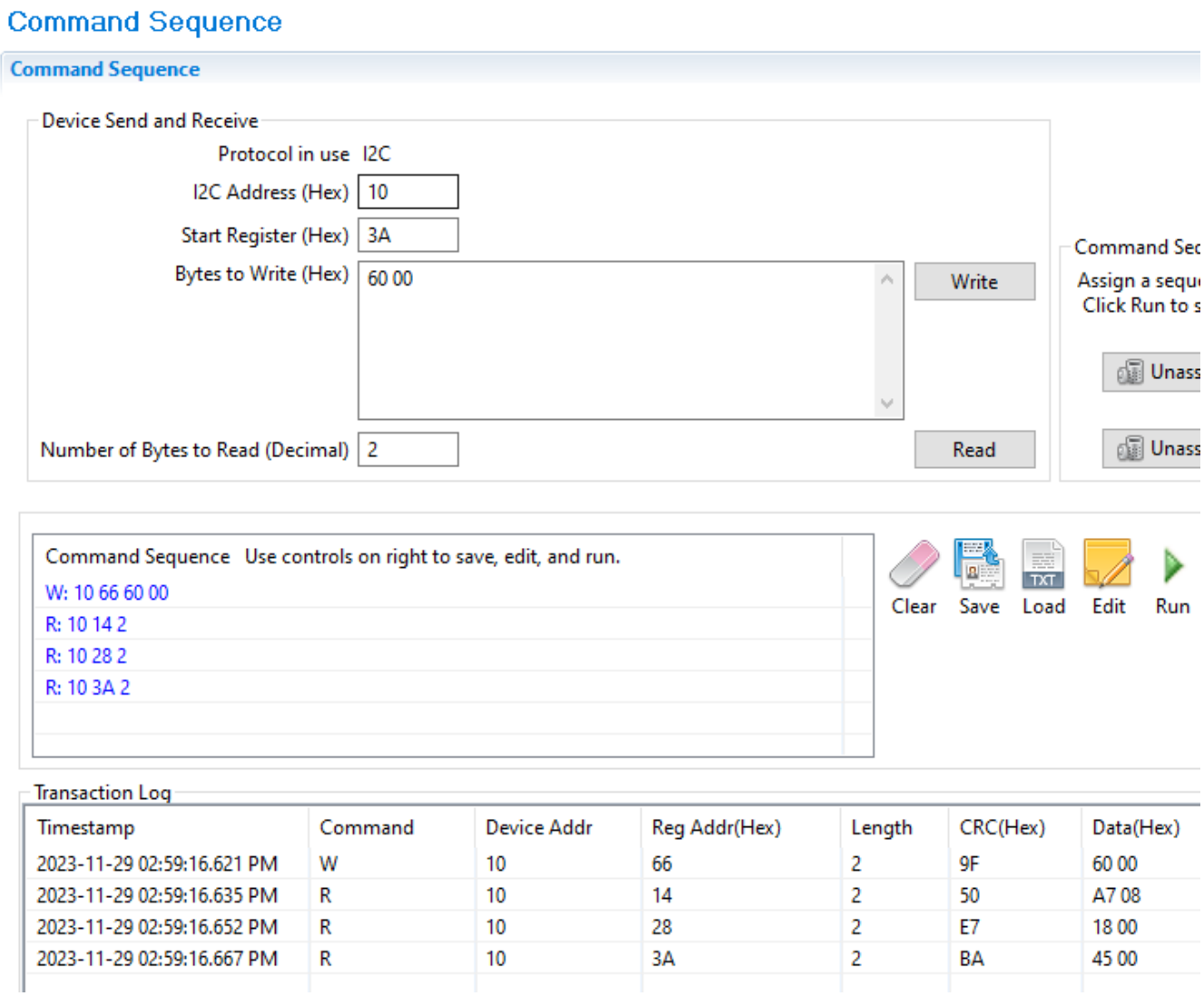


Figure 1-5. BQStudio Showing Execution of Multiple Direct Commands

1.5.1 Disabling Auto Refresh

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

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.

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 CB_ACTIVE_CELLS, PROG_TIMER, PROT_RECOVERY, and SECURITY_KEYS 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

Table 2-1 shows how to read the BQ7690x device number. 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 Figure 2-1, the device number returned is 0x7605 (which represents BQ76905).

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

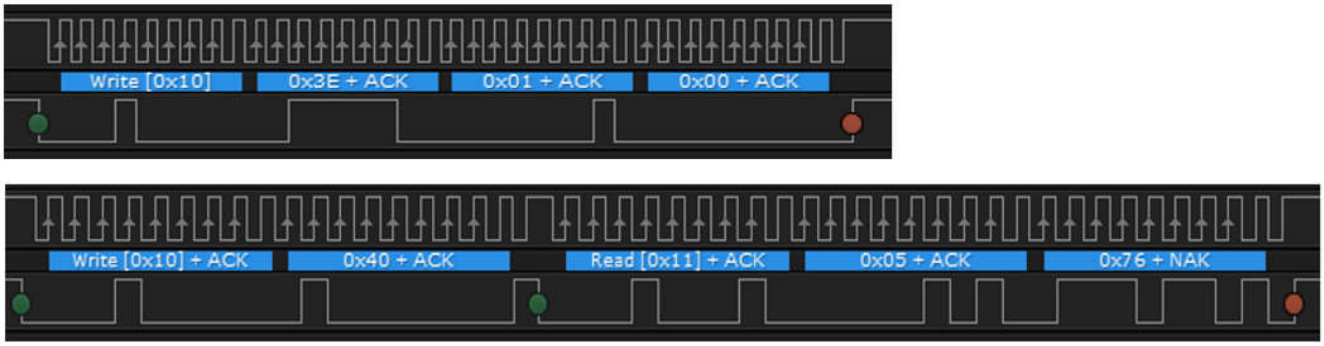


Figure 2-1. Captured I2C Waveform for DEVICE_NUMBER Subcommand

2.2 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 shown in Table 2-2 is one example. In Figure 2-2, this command is issued by writing 0x0022 to 0x3E.

Table 2-2. 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 2-2. Captured I2C Waveform for FET_ENABLE Subcommand

2.3 RESET - 0x0012

The RESET subcommand shown in Table 2-3 performs a reset on the device and returns RAM register settings back to default values. In Figure 2-3, this command is issued by writing 0x0012 to 0x3E.

Table 2-3. RESET Subcommand Description

Command	Name	Description
0x0012	RESET	Resets the device.

Figure 2-3. Captured I2C Waveform of RESET Subcommand

2.4 CB_ACTIVE_CELLS - 0x0083

The CB_ACTIVE_CELLS subcommand shown in Table 2-4 is an example of a subcommand that can read or write cell balancing data to a register. When written, balancing starts on the specified cells. In Figure 2-4, cell balancing is performed on cell 1 by writing the 0x0083 command and 0x02 data to 0x3E followed by a write to 0x60/0x61 with the checksum and length. When writing data with subcommands, the checksum and length are necessary for the data to be accepted. The checksum is calculated on the address and data (0x83, 0x00, 0x02) and is the complement of the sum of these bytes. In this case, the checksum is 0x7A. The length includes the two bytes for device address and command address for a total length of 0x05.

Note

The total length must be in hexadecimal format. For example, a block of 10 registers can have a length of 0x0A, not 0x10.

Table 2-4. CB_ACTIVE_CELLS Subcommand Description

Command	Name	Description
0x0083	CB_ACTIVE_CELLS	Cell balancing active cells: When written, starts balancing on the specified cells.

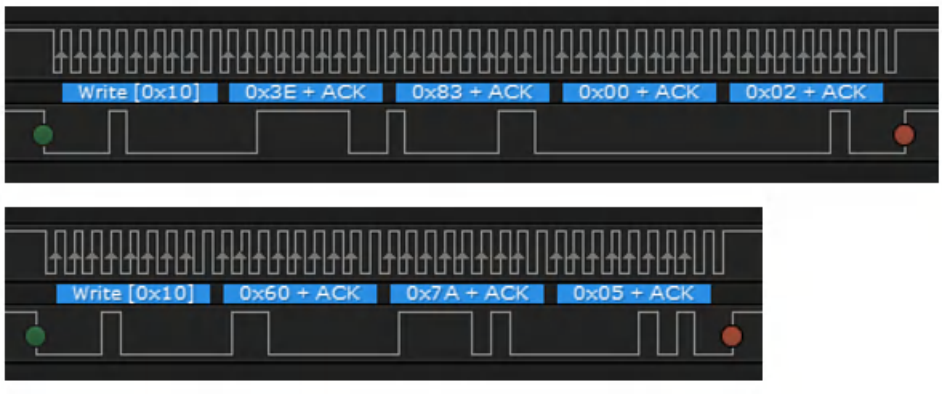


Figure 2-4. Captured I2C Waveform for CB_ACTIVE_CELLS Subcommand

2.5 Subcommand Summary

The *Transaction Log* in Figure 2-5 shows all of the commands that have been covered for executing Subcommands.

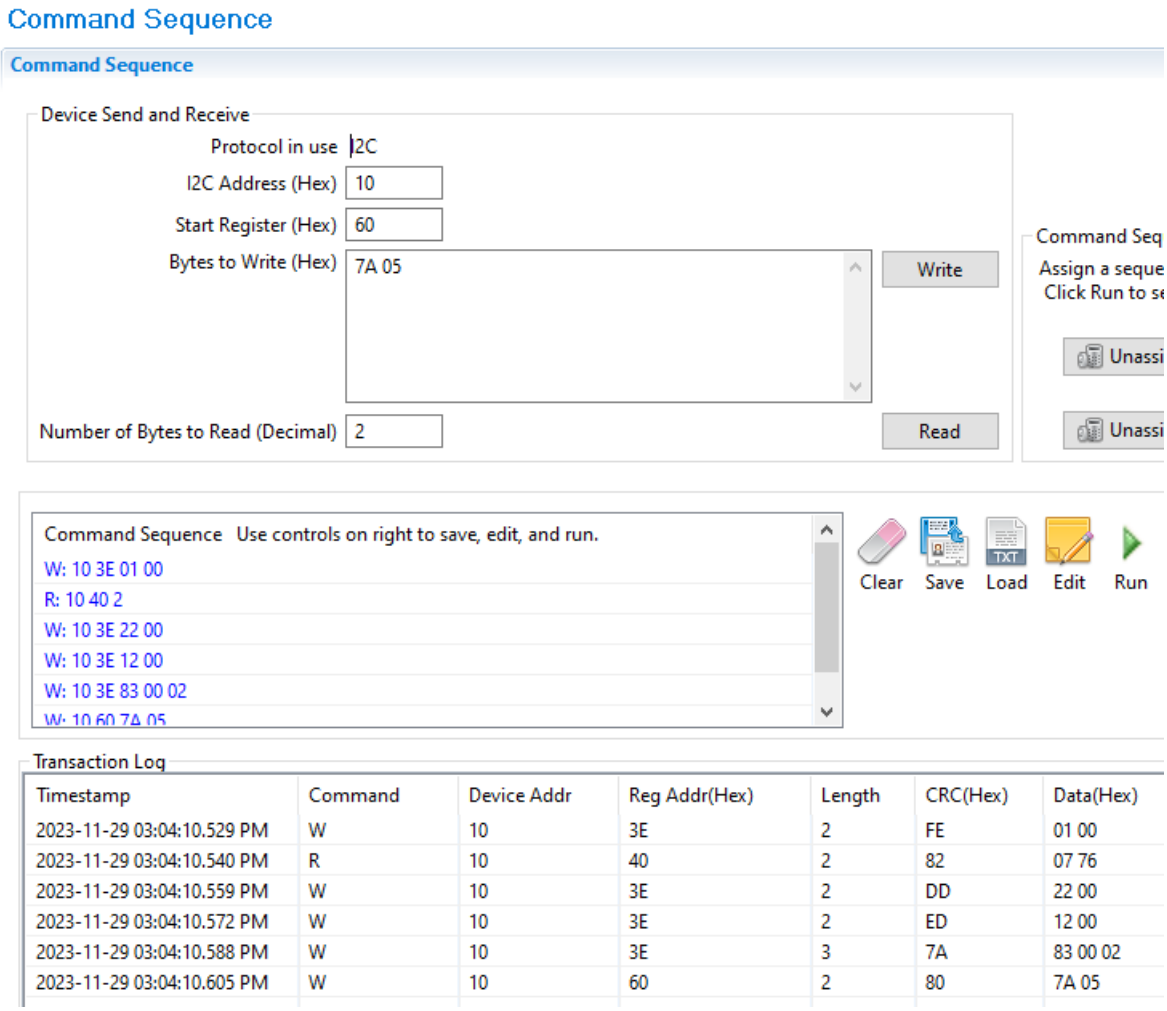


Figure 2-5. BQStudio Example Showing Execution of Multiple Subcommands

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 data sheet, but is illustrated in the following examples.

Note

When writing to RAM registers, the recommendation is to first enter CONFIG_UPDATE mode and then perform the command to exit CONFIG_UPDATE mode once complete. This makes sure there is stable operation while settings are being modified.

3.1 Read Enabled Protections A

The default settings for the BQ7690x devices have COV (over-voltage), SCD (short-circuit), and REGOUT (LDO output) protections enabled. This is verified in the following by reading from the ***Enabled Protections A*** register shown in Table 3-1. In Figure 3-1, the value returned from the RAM address 0x9024 is 0xA1.

Table 3-1. Enabled Protections A Description

Class	Subclass	Name	Type	Min	Max	Default	Unit
Settings	Protection	Enabled Protections A	U1	0x00	0xFF	0xA1	Hex

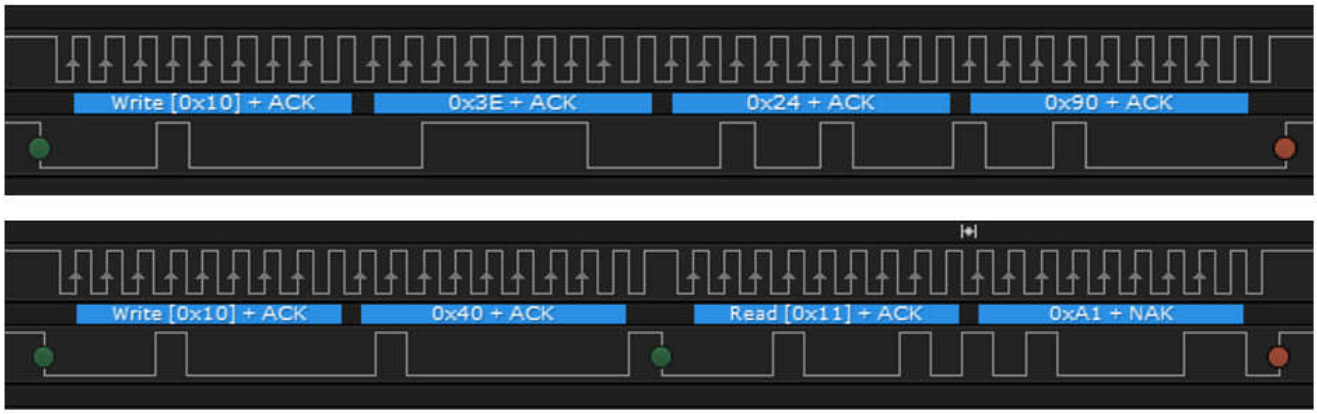


Figure 3-1. Captured I2C Waveform for Reading Enabled Protections A Register

3.2 Enter CONFIG_UPDATE Mode

Before writing RAM registers, the recommendation is to enter CONFIG_UPDATE mode to prevent settings from taking effect until all changes are made. See Table 3-2. SET_CFGUPDATE and EXIT_CFGUPDATE both follow the Subcommand format. In Figure 3-2, the SET_CFGUPDATE Subcommand is given by writing 0x0090 to 0x3E.

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 0xE1 to RAM address 0x9024, shown in Figure 3-3. The checksum is calculated on the address and data (0x24, 0x90, 0xE1) and is the complement of the sum of these bytes. In this case, the checksum is 0x6A. The length includes the two bytes for device address and command address for a total length of 0x05.

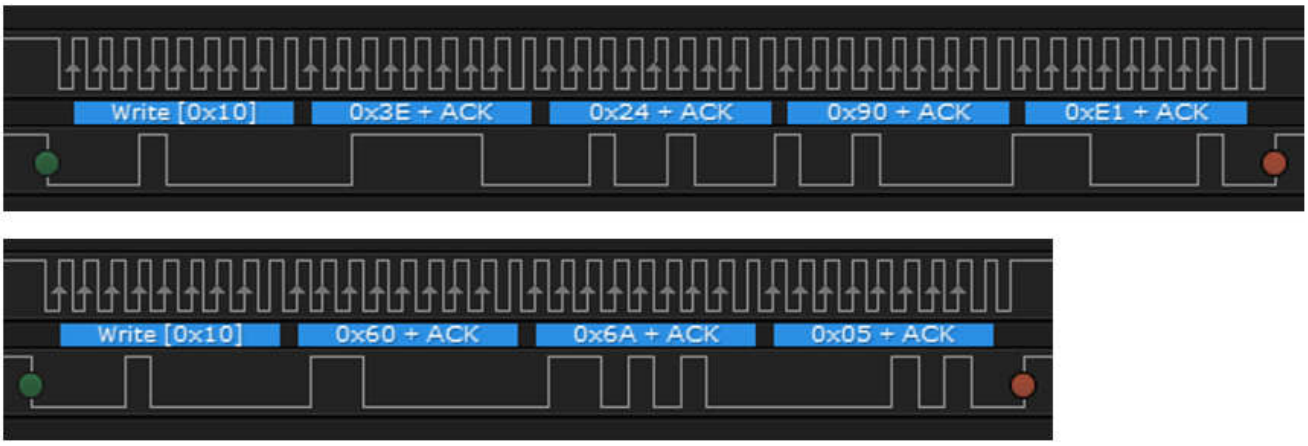


Figure 3-3. Captured Waveform for Writing to Enabled Protections A

3.4 Write VCell Mode

Next, write to the ***VCell Mode*** register shown in Table 3-3 to configure the device BQ76905 for 4 cells. The following example writes 0x04 to 0x901B and then writes the new checksum and length to 0x60/0x61 as seen in Figure 3-4.

Table 3-3. VCell Mode Description

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

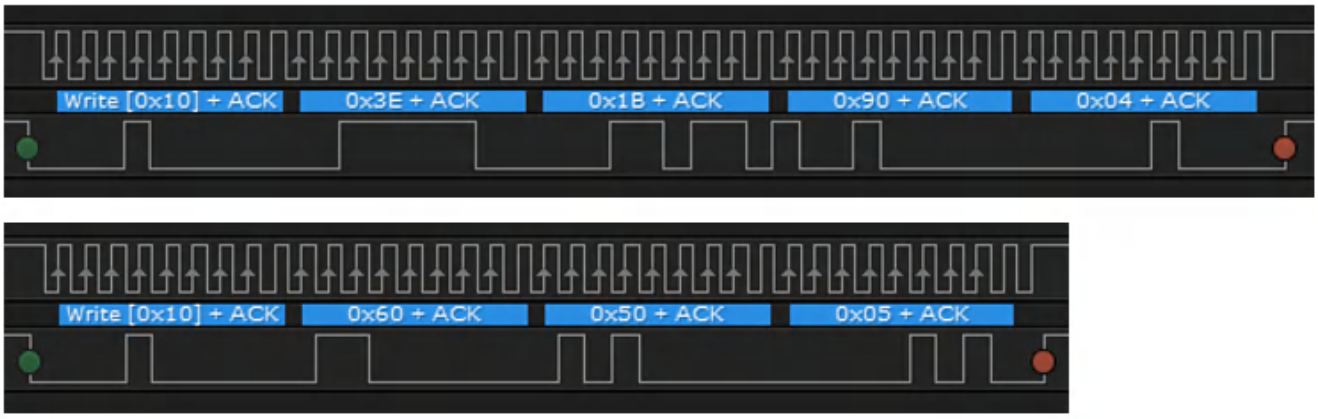


Figure 3-4. Captured I2C Waveform for Writing VCell Mode

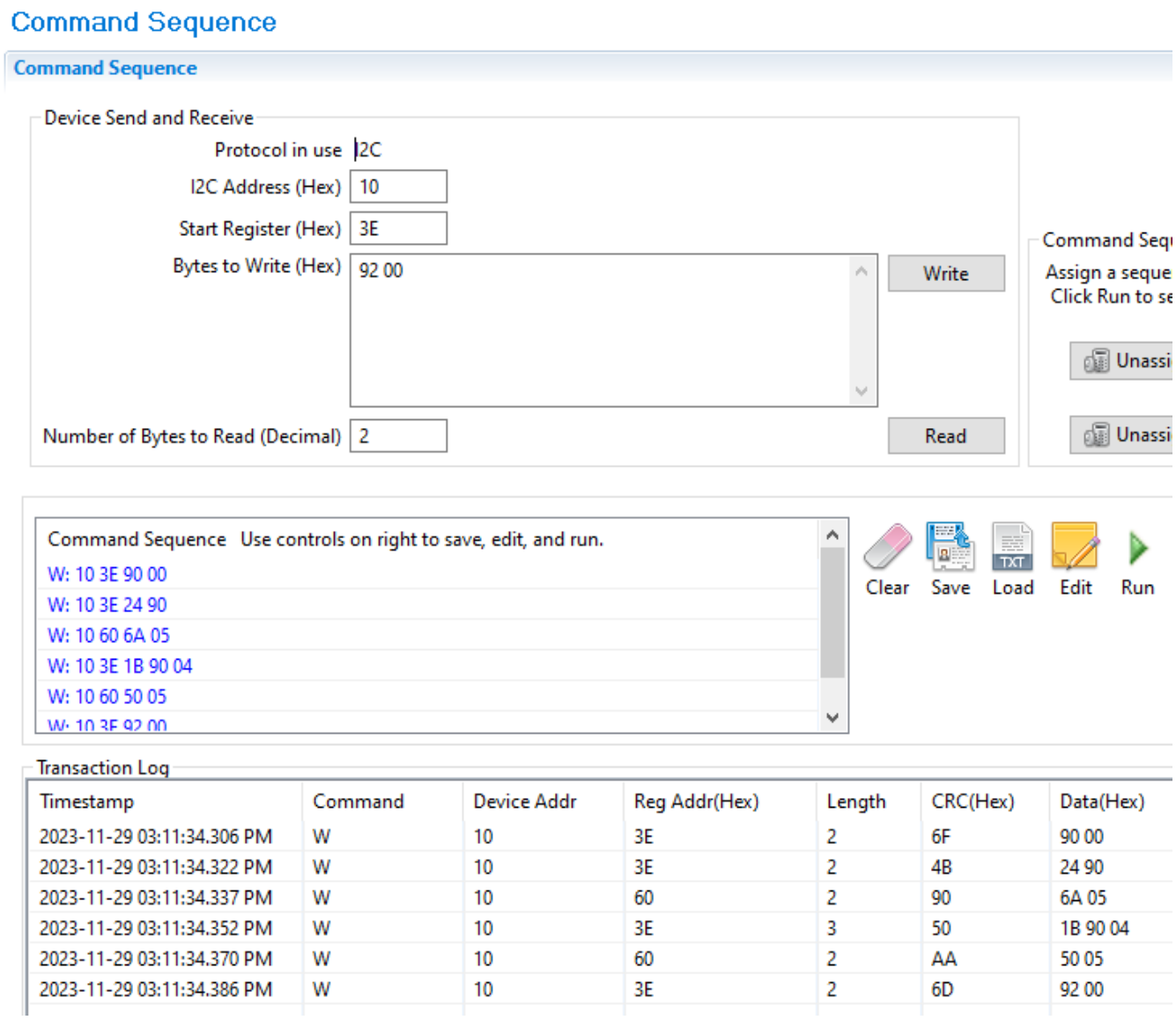
3.5 Exit CONFIG_UPDATE Mode

After writing RAM registers, exit CONFIG_UPDATE mode using the EXIT_CFGUPDATE Subcommand. See Figure 3-5. At this point, the new settings can take effect.

Figure 3-5. Captured I2C Waveform for EXIT_CFGUPDATE

3.6 Reading and Writing RAM Registers Summary

The *Transaction Log* in Figure 3-6 shows all the commands that have been covered for reading and writing to RAM registers.



Timestamp	Command	Device Addr	Reg Addr(Hex)	Length	CRC(Hex)	Data(Hex)
2023-11-29 03:11:34.306 PM	W	10	3E	2	6F	90 00
2023-11-29 03:11:34.322 PM	W	10	3E	2	4B	24 90
2023-11-29 03:11:34.337 PM	W	10	60	2	90	6A 05
2023-11-29 03:11:34.352 PM	W	10	3E	3	50	1B 90 04
2023-11-29 03:11:34.370 PM	W	10	60	2	AA	50 05
2023-11-29 03:11:34.386 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 BQ7690x family includes an optional CRC check. The CRC feature can be enabled in the ***Settings:Configuration:I2C_Config[CRC]*** register. If this register is changed while using BQStudio, then BQStudio can 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

The *VCell 1* command is used in Figure 4-2, and the CRC for the first byte is computed for [0x10 0x14 0x11 0xC5]. The resulting CRC is 0x79. The CRC for the second byte [0x0B] is 0x31.

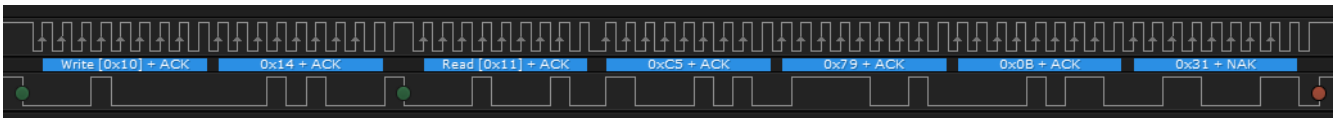


Figure 4-2. Captured I2C Waveform for VCell 1 Command With CRC

5 Simple Code Examples

The following example code is written in Python and designed to communicate to the BQ7690x device from a PC through an EV2400 module or through the USB connector on the BQ76905 or BQ76907 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.

```
'''
/* BQ7690x example Program demonstrates examples for direct commands, subcommands, and writing /
reading from device RAM.
'''
I2C_ADDR = 0x10  # BQ7690x default responder address
numCells = 5    # Set to 7 for BQ76907
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 Octane
    '''
    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 ReadCellVoltage(cell):
    '''
    Reads a specific cell voltage
    '''
    cmd_addr = 0x14 + (cell * 2)
    result = I2C_Read(I2C_ADDR, cmd_addr, 2)
    print ("Cell", cell, " = ", (result[1]*256 + result[0]), " mV")
    return
def ReadAllCellVoltages():
    '''
    Reads all cell voltages, Stack voltage, PACK voltage, and LD voltage
    '''
    cmd_addr = 0x12
    for i in range(0,numCells):
        cmd_addr += 2
        result = I2C_Read(I2C_ADDR, cmd_addr,2)
        print ("Cell", (i+1), " = ", (result[1]*256 + result[0]), " mV")
    result = I2C_Read(I2C_ADDR, 0x26,2)
    print ("Stack Voltage = ", (result[1]*256 + result[0]), " mV")
    result = I2C_Read(I2C_ADDR, 0x22,2)
    print "REG18 Voltage = ", (result[1]*256 + result[0]), " ADC Counts"
    result = I2C_Read(I2C_ADDR, 0x24,2)
    print ("VSS Voltage = ", (result[1]*256 + result[0]), " ADC Counts")
    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 0x0060 - FULLSCAN, ADSCAN
I2C_Write(I2C_ADDR, 0x66, [0x60, 0x00])
#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, 0x28, 2)
print ("Internal Temp = ", ((result[1]*256 + result[0])), "degrees C")
#Read CC2 Current Measurement
result = I2C_Read(I2C_ADDR, 0x3A, 2)
print ("CC2", (result[1]*256 + result[0]), " mA")
################ Subcommand Examples ################
## Command-only Subcomands ##
#Read Device Number
b = DataRAM_Read(0x0001,6)
print ("Device_Number = " '{0:04X}'.format(b[1]*256+b[0]))
#FET_ENABLE
I2C_Write(I2C_ADDR, 0x3E, [0x22, 0x00])
#Cell Balance Write Command - starts balancing on specified cells when written
DataRAM_Write(0x0083, [0x02])
#Cell Balance Read Command - read which cells are being balanced
b = DataRAM_Read(0x0083,1)
print ("Cell Balancing = 0x" '{0:02X}'.format(b[0]))
#RESET - returns device to default settings
I2C_Write(I2C_ADDR, 0x3E, [0x12, 0x00])
sleep(2)
# Read 'Enabled Protections A' RAM register 0x9024
b = DataRAM_Read(0x9024,1)
print ("Enabled Protections A = 0x" '{0:02X}'.format(b[0]))
#Set CONFIG_UPDATE Mode (RAM registers can 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(0x9024, [0xE1])
#Write to 'VCell Mode' RAM register to configure for a 9-cell battery
DataRAM_Write(0x901B, [0x04])
#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)
ReadAllCellVoltages()
```

The output from running the example Python script on a BQ76905 Evaluation Module follows.

```
('Cell 1 = ', 3089, ' mV')
('Internal Temp = ', 29, 'degrees C')
('CC2', 45, ' mA')
Device_Number = 7605
Cell Balancing = 0x02
Enabled Protections A = 0xA1
crc8 check = 0x33
('Cell', 1, ' = ', 3089, ' mV')
('Cell', 2, ' = ', 3082, ' mV')
('Cell', 3, ' = ', 3093, ' mV')
('Cell', 4, ' = ', 3089, ' mV')
('Cell', 5, ' = ', 3089, ' mV')
('Stack Voltage = ', 15471, ' mV')
REG18 Voltage =  19153  ADC Counts
('VSS Voltage = ', 3, ' ADC Counts')
```

6 References

- Texas Instruments, *2S-7S BQ76907 Battery Monitor and Protector*, data sheet.
- Texas Instruments, *2S-5S BQ76905 Battery Monitor and Protector*, data sheet.
- Texas Instruments, *BQ76907 Technical Reference Manual*.
- Texas Instruments, *BQ76905 Technical Reference Manual*.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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, regulatory 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 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.

TI objects to and rejects any additional or different terms you may have proposed.

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