

**ProDAQ**

**User Manual**

**ProDAQ 3416**  
**16-Channel 24-bit Sigma-Delta**  
**ADC Function Card**

**PUBLICATION NUMBER: 3416-XX-UM-1010**

**Copyright, © 2014, Bustec Production, Ltd.**



**Bustec Production, Ltd.**  
**Bustec House, Shannon Business Park, Shannon, Co. Clare, Ireland**  
**Tel: +353 (0) 61 707100, FAX: +353 (0) 61 707106**

## **PROPRIETARY NOTICE**

**This document and the technical data herein disclosed, are proprietary to Bustec Production Ltd., and shall not, without express written permission of Bustec Production Ltd, be used, in whole or in part to solicit quotations from a competitive source or used for manufacture by anyone other than Bustec Production Ltd. The information herein has been developed at private expense, and may only be used for operation and maintenance reference purposes or for purposes of engineering evaluation and incorporation into technical specifications and other documents, which specify procurement of products from Bustec Production Ltd. This document is subject to change without further notification. Bustec Production Ltd. reserves the right to change both the hardware and software described herein.**

## Table of Contents

<b>1. INTRODUCTION</b> .....	<b>9</b>
<b>2. INSTALLATION</b> .....	<b>11</b>
2.1. Unpacking and Inspection .....	11
2.2. Reshipment Instructions .....	11
2.3. ProDAQ VXIbus Module Installation.....	12
2.3.1. <i>Preparing the ProDAQ Module</i> .....	12
2.3.2. <i>Installing a ProDAQ Function Card</i> .....	13
2.3.3. <i>Removing a ProDAQ Function Card</i> .....	15
2.4. ProDAQ LXI Function Card Carrier Installation .....	15
2.4.1. <i>Opening the ProDAQ 6100 Enclosure</i> .....	15
2.4.2. <i>Installing a ProDAQ Function Card</i> .....	17
2.4.3. <i>Removing a ProDAQ Function Card</i> .....	19
2.4.4. <i>Closing the ProDAQ 6100 Enclosure</i> .....	20
<b>3. THEORY OF OPERATION</b> .....	<b>23</b>
3.1. Block Diagram .....	23
3.2. Analog Front-End Circuitry .....	23
3.3. TEDS Reader Interface .....	24
3.4. I <sup>2</sup> C Master Interface.....	24
3.5. Data Acquisition.....	25
3.6. Sampling Settings.....	26
3.7. Multiple Cards Synchronization .....	26
<b>4. SPECIFICATIONS</b> .....	<b>27</b>
4.1. Input Characteristics.....	27
4.2. Sampling .....	28
4.3. Triggering .....	28
4.4. Synchronization .....	28
4.5. Environmental Specifications .....	28
<b>5. THE VXIPLUG&amp;PLAY DRIVER</b> .....	<b>29</b>
5.1. Installation .....	29
5.2. The Soft Front Panel .....	29
5.2.1. <i>“Waveforms” Tab</i> .....	31
5.2.2. <i>“Channels” Tab</i> .....	31
5.2.3. <i>“Acquisition” Tab</i> .....	32
<b>6. PROGRAMMING THE PRODAQ 3416</b> .....	<b>33</b>
6.1. VXIplug&play Driver Organization .....	33
6.2. Connecting to the Function Card.....	34
6.3. Hardware Configuration.....	35
6.4. Single-Card Acquisition .....	35
6.4.1. <i>Single-shot Acquisition</i> .....	35
6.4.2. <i>Continuous Acquisition</i> .....	37

6.5. Calibration .....	39
<b>APPENDIX A: FRONT-PANEL CONNECTOR.....</b>	<b>41</b>
<b>APPENDIX B: REGISTER DESCRIPTION.....</b>	<b>43</b>
B.1 FCID (0x0) – Function Card ID Register.....	45
B.2 FCVER (0x1) – Function Card Version Register .....	45
B.3 FCCSR (0x2) – Function Card Control and Status Register .....	45
B.4 MODE1 (0x3) – Mode 1 Register .....	47
B.5 MODE2 (0x4) – Mode 2 Register .....	48
B.6 OTRI (0x5) – Output Trigger Configuration Register .....	49
B.7 ITRI (0x6) – Input Trigger Status Register.....	50
B.8 DDS_WX (0x8) – DDS Control Register .....	52
B.9 OCOEFL (0x9) – Offset Coefficient Write Low Register.....	52
B.10 OCOEFH (0xA) – Offset Coefficient Write High Register.....	52
B.11 GCOEFL (0xB) – Gain Coefficient Write Low Register .....	53
B.12 GCOEFH (0xC) – Gain Coefficient Write High Register .....	53
B.13 I2C_CTRL (0xE) – I2C Control Register .....	54
B.14 TEDS_ACC (0xF) – TEDS Access Register .....	54
B.15 FIFO_CTRL (0x10) – FIFO Control Register.....	55
B.16 FIFO_AFT (0x11) – FIFO Almost Full Flag Threshold Register .....	56
B.17 FIFO_WR (0x12) – FIFO Write Register .....	56
B.18 SIG_ERR (0x13) – Signal Error Register .....	56
B.19 GAIN_COMP (0x14) – Gain Compensation Register.....	57
B.20 ERROR (0x15) – Error Register .....	57
B.21 POSTT_NOSL (0x19) – Post Trigger Number of Scans Low Register .....	58
B.22 POSTT_NOSH (0x1A) – Post Trigger Number of Scans High Register .....	58
B.23 CHNxCFG (0x20...0x2F) – Channel x Configuration Register .....	59
B.24 FCSSUB (0xFC) – Function Card Sub-Type Register .....	59
B.25 FCSERH (0xFE) – Function Card Serial Number High Register.....	59
B.26 FCSERL (0xFF) – Function Card Serial Number Low Register .....	60
B.27 FIFO (0x8000) – FIFO memory.....	60
<b>APPENDIX C: VXIPLUG&amp;PLAY DRIVER FUNCTIONS .....</b>	<b>61</b>
C.1 Introduction.....	61
C.2 Assumptions.....	61
C.3 Error and Status Information: .....	61
C.4 Function Tree Layout: .....	62
C.5 VXIplug&play Driver Function Details.....	64
C.5.1 <i>bu3416_acquireWaveform</i> .....	64
C.5.2 <i>bu3416_acquireWaveforms</i> .....	66
C.5.3 <i>bu3416_armDAQ</i> .....	69
C.5.4 <i>bu3416_burnTEDS_OTP_ROM</i> .....	70
C.5.5 <i>bu3416_calibrateAllChannels</i> .....	71
C.5.6 <i>bu3416_calibrateBoard</i> .....	72
C.5.7 <i>bu3416_checkAcquisition</i> .....	73
C.5.8 <i>bu3416_checkMultAcquisition</i> .....	75
C.5.9 <i>bu3416_clearErrors</i> .....	77
C.5.10 <i>bu3416_close</i> .....	78

C.5.11	<i>bu3416_enableLIST</i> .....	79
C.5.12	<i>bu3416_error_message</i> .....	80
C.5.13	<i>bu3416_error_query</i> .....	81
C.5.14	<i>bu3416_fcSelect</i> .....	82
C.5.15	<i>bu3416_generateITRI</i> .....	83
C.5.16	<i>bu3416_generateOTRI</i> .....	84
C.5.17	<i>bu3416_getADCMode</i> .....	85
C.5.18	<i>bu3416_getBufferSize</i> .....	87
C.5.19	<i>bu3416_getCalibData</i> .....	88
C.5.20	<i>bu3416_getDAQMode</i> .....	90
C.5.21	<i>bu3416_getDAQStatus</i> .....	92
C.5.22	<i>bu3416_getDDSFreq</i> .....	94
C.5.23	<i>bu3416_getFIFOConfig</i> .....	95
C.5.24	<i>bu3416_getFIFOStatus</i> .....	96
C.5.25	<i>bu3416_getFPTrigPolarity</i> .....	97
C.5.26	<i>bu3416_getITRIConfig</i> .....	99
C.5.27	<i>bu3416_getITRIState</i> .....	100
C.5.28	<i>bu3416_getMultFCsession</i> .....	101
C.5.29	<i>bu3416_getOTRIConfig</i> .....	103
C.5.30	<i>bu3416_getPostScans</i> .....	106
C.5.31	<i>bu3416_getSampFreq</i> .....	107
C.5.32	<i>bu3416_getSerNum</i> .....	108
C.5.33	<i>bu3416_init</i> .....	109
C.5.34	<i>bu3416_multClose</i> .....	111
C.5.35	<i>bu3416_multConfig</i> .....	112
C.5.36	<i>bu3416_multInit</i> .....	115
C.5.37	<i>bu3416_paramInit</i> .....	118
C.5.38	<i>bu3416_readAcquisition</i> .....	120
C.5.39	<i>bu3416_readFIFO</i> .....	122
C.5.40	<i>bu3416_readMultAcquisition</i> .....	123
C.5.41	<i>bu3416_readTEDS_EEPROM</i> .....	125
C.5.42	<i>bu3416_readTEDS_OTP_ROM</i> .....	126
C.5.43	<i>bu3416_readTEDS_ROM</i> .....	127
C.5.44	<i>bu3416_reset</i> .....	128
C.5.45	<i>bu3416_resetDAQ</i> .....	129
C.5.46	<i>bu3416_resetFIFO</i> .....	130
C.5.47	<i>bu3416_resetI2C</i> .....	131
C.5.48	<i>bu3416_resizeMultBuf</i> .....	132
C.5.49	<i>bu3416_revision_query</i> .....	133
C.5.50	<i>bu3416_self_test</i> .....	134
C.5.51	<i>bu3416_setAcquisitionMode</i> .....	135
C.5.52	<i>bu3416_setADCMode</i> .....	137
C.5.53	<i>bu3416_setBufferSize</i> .....	139
C.5.54	<i>bu3416_setChanConfig</i> .....	140
C.5.55	<i>bu3416_setDAQMode</i> .....	142
C.5.56	<i>bu3416_setDDSFreq</i> .....	144
C.5.57	<i>bu3416_setFIFOConfig</i> .....	145
C.5.58	<i>bu3416_setFPTrigPolarity</i> .....	146
C.5.59	<i>bu3416_setITRIConfig</i> .....	148
C.5.60	<i>bu3416_setMultChanConfig</i> .....	150

C.5.61	<i>bu3416_setMultTrigConfig</i> .....	152
C.5.62	<i>bu3416_setOTRConfig</i> .....	153
C.5.63	<i>bu3416_setPostScans</i> .....	156
C.5.64	<i>bu3416_setSampFreq</i> .....	157
C.5.65	<i>bu3416_setTrigConfig</i> .....	158
C.5.66	<i>bu3416_startAcquisition</i> .....	159
C.5.67	<i>bu3416_startAcquisitionEx</i> .....	160
C.5.68	<i>bu3416_startMultAcquisition</i> .....	161
C.5.69	<i>bu3416_startMultAcquisitionEx</i> .....	162
C.5.70	<i>bu3416_stopAcquisition</i> .....	163
C.5.71	<i>bu3416_stopDAQ</i> .....	164
C.5.72	<i>bu3416_stopMultAcquisition</i> .....	165
C.5.73	<i>bu3416_storeCalibData</i> .....	166
C.5.74	<i>bu3416_writeReadI2C</i> .....	168
C.5.75	<i>bu3416_writeTEDS_EEPROM</i> .....	169

## Table of Figures

Figure 1 – Removing the ProDAQ module cover .....	12
Figure 2 – The ProDAQ module assembly .....	14
Figure 3 – Simplified Block Diagram.....	23
Figure 4 – Analog front-end circuitry (single channel) .....	24
Figure 5 - Selecting the Connection Method .....	29
Figure 6 – Specifying the Function Card Address .....	30
Figure 7 - ProDAQ 3416 Soft Front Panel Application .....	30
Figure 8 – Channel Configuration.....	31
Figure 9 – Acquisition Configuration.....	32
Figure 10 – <i>VXIplug&amp;play</i> Driver Organization .....	33
Figure 11 - Opening a Session .....	34
Figure 12 – Acquiring a Waveform .....	36
Figure 13 – Starting the Asynchronous Acquisition .....	37
Figure 14 – Checking the Status of the Acquisition and Data Read-out.....	38
Figure 15 - Front panel connector as seen when the card is fitted in the module. ....	41

## Reference Documents

Title	Number
ProDAQ 3180 User Manual	3180-XX-UM
ProDAQ 6100 User Manual	6100-XX-UM

## Glossary

<b>ADC</b>	: Analog-to-Digital Converter
<b>CRD</b>	: Current Regulator Diode
<b>DA</b>	: Data Acquisition
<b>DAC</b>	: Digital-to-Analog Converter
<b>DDS</b>	: Direct Digital Synthesis
<b>DTC</b>	: Discharge Time Constant
<b>ECL</b>	: Emitter-Coupled Logic
<b>FIR</b>	: Finite Impulse Response digital filter
<b>FPGA</b>	: Field Programmable Gate Array
<b>H</b>	: State of the bit(s) defined by hardware (in register description)
<b>ICP</b>	: Integrated Circuit Piezoelectric
<b>LED</b>	: Light Emitting Diode
<b>LVDS</b>	: Low Voltage Differential Signal(ing)
<b>LXI</b>	: LAN eXtensions for Instrumentation
<b>PCB</b>	: Printed Circuit Board
<b>PGA</b>	: Programmable Gain Amplifier
<b>PLL</b>	: Phase-Locked Loop
<b>RO</b>	: Read-only access to register
<b>R/W</b>	: Read/Write access to register
<b>R/WSC</b>	: Read/Write access to register, Self-Clear after operation finished
<b>TEDS</b>	: Transducer Electronic Data Sheet
<b>VREF</b>	: Voltage Reference
<b>VXI</b>	: VME eXtensions for Instrumentation
<b>WO</b>	: Write-only access to register

## 1. Introduction

The ProDAQ 3416 function card is a 16-channel, 24-bit Sigma-Delta Analog-to-Digital converter function card. It is an add-on card to use together with ProDAQ motherboards and function card carriers.

It provides the following features:

- 16 analog channels with 24-bit resolution
- Simultaneous sampling
- Max. Input Range  $\pm 10V$  with overvoltage protection
- Programmable gains of 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 and 2000
- User programmable any sampling rate in a range from 1SPS to 10kSPS
- On-board FIFO memory
- Synchronization of multiple cards
- I2C master controller for an external signal conditioning unit
- IEEE 1451.4 (TEDS) Smart Transducer Interface support

This page was intentionally left blank.

## 2. Installation

ProDAQ function cards can be installed in ProDAQ VXIbus Motherboards and ProDAQ LXI Function Card Carriers. If you ordered your ProDAQ function card together with the ProDAQ motherboard or carrier, the function cards will be pre-installed to your specification. If you want to install additional cards or exchange installed cards, use the following disassembling/assembling procedure.

### 2.1. Unpacking and Inspection

The ProDAQ instrument is shipped in an antistatic package to prevent any damage from electrostatic discharge (ESD). Proper ESD handling procedures must always be used when packing, unpacking or installing any ProDAQ module, ProDAQ plug-in module or ProDAQ function card:

- Ground yourself via a grounding strap or similar, e.g. by holding to a grounded object.
- Discharge the package by touching it to a grounded object, e.g. a metal part of your VXIbus chassis, before removing the module from the package.
- Remove the ProDAQ instrument from its carton, preserving the factory packaging as much as possible.
- Inspect the ProDAQ instrument for any defect or damage. Immediately notify the carrier if any damage is apparent.

### 2.2. Reshipment Instructions

Use the original packing material when returning a ProDAQ instrument to Bustec Production Ltd. for calibration or servicing. The original shipping carton and the instrument's plastic foam will provide the necessary support for safe reshipment.

If the original anti-static packing material is unavailable, wrap the ProDAQ instrument in anti-static plastic sheeting and use plastic spray foam to surround and protect the instrument. Reship in either the original or new shipping carton.

---

#### **WARNING**

**Proper ESD handling procedures must always be used when packing, unpacking or installing any ProDAQ device or ProDAQ function card. Ground yourself via a grounding strap or similar, e.g. by holding to a grounded object and discharge the package by touching it to a grounded object, before removing the module from the package.**

---

## 2.3. ProDAQ VXIbus Module Installation

### 2.3.1. Preparing the ProDAQ Module

To install a ProDAQ function card into one of the ProDAQ motherboards, you need to remove the module's top cover:

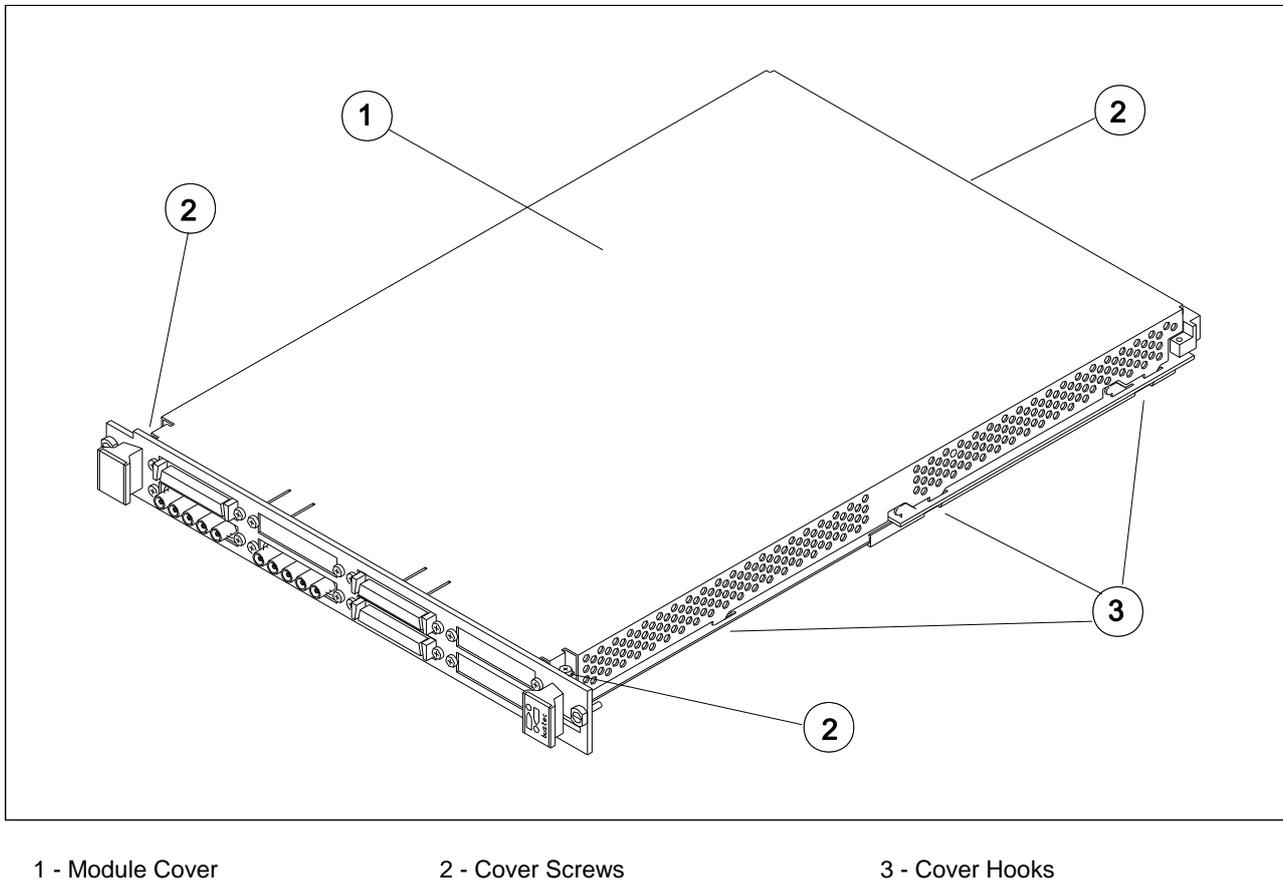


Figure 1 – Removing the ProDAQ module cover

To remove the top cover, remove the one countersunk screw in the back and the two panhead screws towards the front panel (2), that hold the cover in place. Remove the cover by sliding it out of its position towards the VXIbus connectors and up. Take special care about the hooks (3) holding it in place. Try not to lift the cover straight up. See Figure 1 for the location of the screws.

To re-install the cover, slide it back into its position by placing the small hooks over their holes and moving the cover down and forward. Secure the top cover using two panhead screws and one countersunk screw (2).

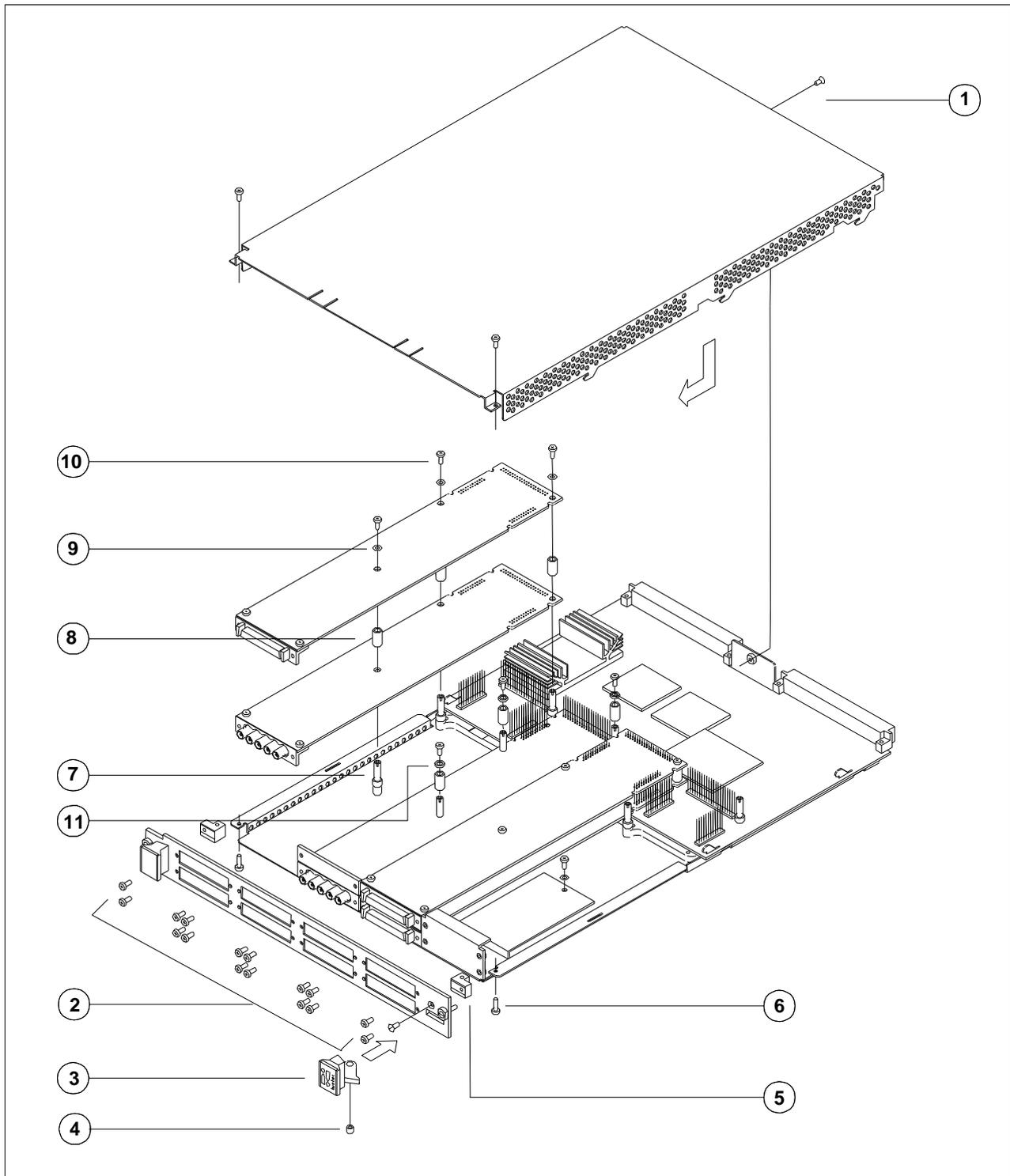
### 2.3.2. Installing a ProDAQ Function Card

The single-width ProDAQ function cards are arranged inside the ProDAQ module in four stacks of two cards each. The double-width ProDAQ function cards are arranged inside the ProDAQ module in two stacks of two cards each. The function cards are mounted face down, e.g. the front-panel connectors as well as the motherboard connectors are underneath the PCB. Single-width and double-width ProDAQ function cards can be mixed in the ProDAQ module. The ProDAQ 3416 function card is a single-width card.

To install a single-width ProDAQ function card in any of the possible positions, use the following procedure (See Figure 2 for reference):

- Remove the top cover of the module as described earlier in this chapter (Fig. 2, Pos. 1).
- Remove all screws on the front-panel holding installed function cards or double filler panels in place (Fig. 2, Pos. 2). Screws holding single filler panels don't need to be removed.
- Remove the two panhead screws that mount the front panel to the modules bottom cover (Fig. 2, Pos. 6).
- Please take special care of the module handles and the rings (Fig. 2, Pos. 3 and 4), which are also fixed by those screws. The mounting angle (Fig. 2, Pos. 5) stays fixed to the front panel.
- Remove the front panel by moving it forward carefully so as to avoid bending the installed function cards.
- Choose the stack and position (lower or upper) where you want to mount the function card. If the stack, in which the function card should be installed, is covered by a double filler panel, you have to remove it before installing the function card.
- Remove the three 2.5mm panhead screws and the crinkle washers from the stack's standoffs (Fig. 2, Pos. 9 and 10 for example).
- If you want to install a function card in the upper position of a stack without having a function card in the lower position, you need to mount both spacers (Fig. 3, Pos. 11) on each standoff. If the stack is already populated with a function card in the lower position, mount only the bigger spacer (Fig. 2, Pos. 8) onto each standoff.
- Place a bayonet (supplied) on each standoff. Align the function card over these and slide carefully down. The function card should be held parallel to the modules bottom cover all the time during its way down.
- Fix the function card by mounting the three 2.5mm panhead screws and the crinkle washers onto each standoff. If you install a function card in the lower position of a stack, you need first to mount both spacers (Fig. 2, Pos. 11) onto each standoff.
- Re-mount the modules front-panel. If there is only one function card mounted in a stack, cover the remaining opening in the front panel by a single filler panel.
- Re-mount the modules top cover.

Adjust the procedure respectively for a double-width ProDAQ function card.



- |                          |                          |                          |
|--------------------------|--------------------------|--------------------------|
| 1 - 2.5mm Panhead Screws | 2 - 2.5mm Panhead Screws | 3 - Module Handle        |
| 4 - Ring                 | 5 - Mounting Angle       | 6 - 2.5mm Panhead Screws |
| 7 - Standoff             | 8 - Spacer               | 9 - Crinkle Washer       |
| 10 - 2.5mm Panhead Screw | 11 - 2mm Spacer          |                          |

Figure 2 – The ProDAQ module assembly

### 2.3.3. Removing a ProDAQ Function Card

Removing a ProDAQ function card is exactly the reverse operation then installing it. After removing the top cover and the front panel as described previously, remove the three roundhead screws that fix the function card(s) on the standoffs.

Take special care when removing the function card(s) not to bend the motherboard connectors.

After removing the function card(s), install the correct combination of spacers on the standoffs. If a stack is populated with only one function card, each of the standoffs needs to be mounted with both spacers to cover the distance between the cards as well as the PCB thickness of the missing card. If a stack is populated with two function cards, only the bigger spacer must be mounted.

Fix any remaining function cards again by mounting the three panhead screws on the standoffs, re-mount the front panel and the modules cover.

## 2.4. ProDAQ LXI Function Card Carrier Installation

---

### **WARNING**

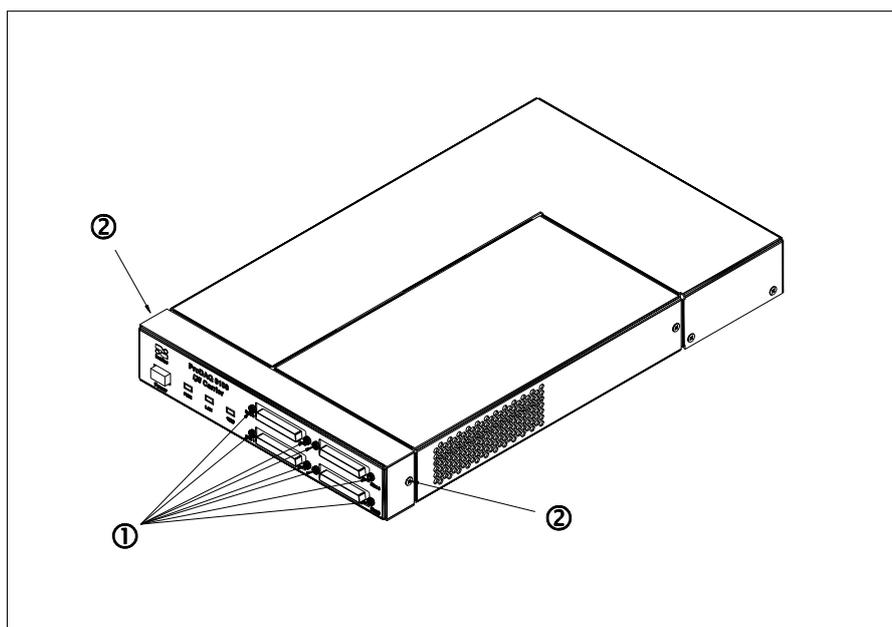
---

**Disconnect the ProDAQ 6100 from the mains before opening the enclosure!**

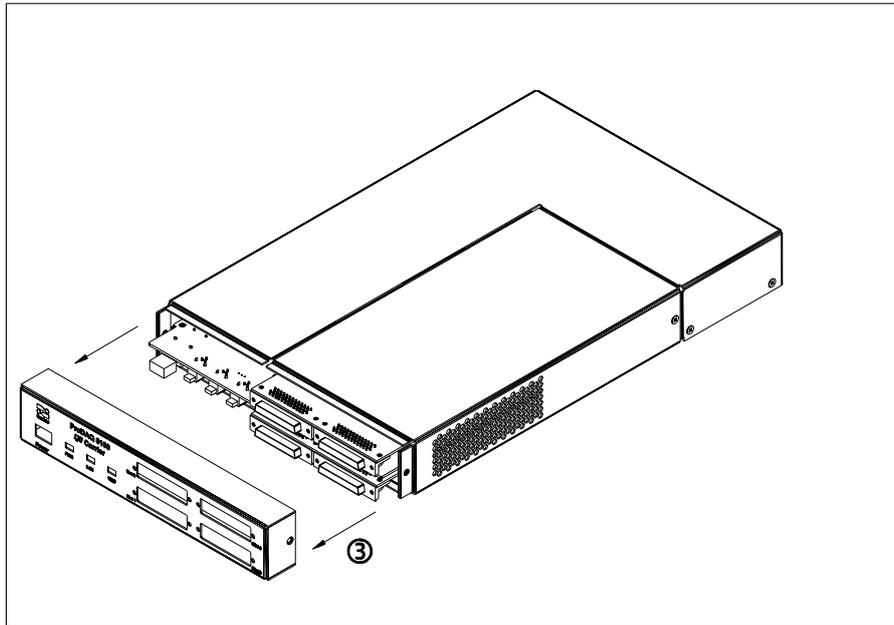
---

### 2.4.1. Opening the ProDAQ 6100 Enclosure

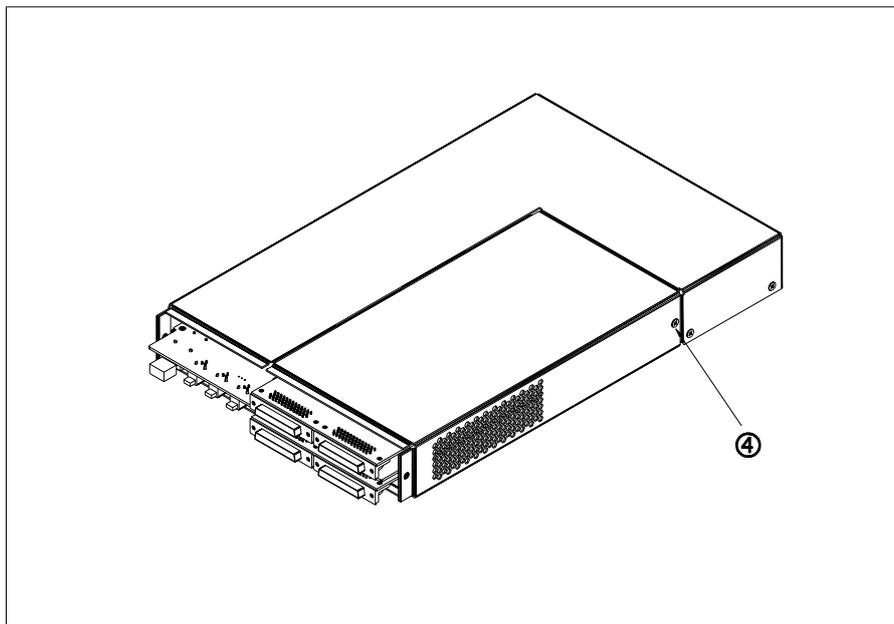
Remove the up to eight M2.5x6mm Pozidrive Panhead screws (①) attaching the front bezel to the function cards (If there is no function card installed in a slot and a blanking panel is used to cover the front bezel opening, do not remove it screws before detaching the front bezel). Then remove the two M3x6mm Torx Countersunk screws (②) attaching the front bezel to the enclosure.



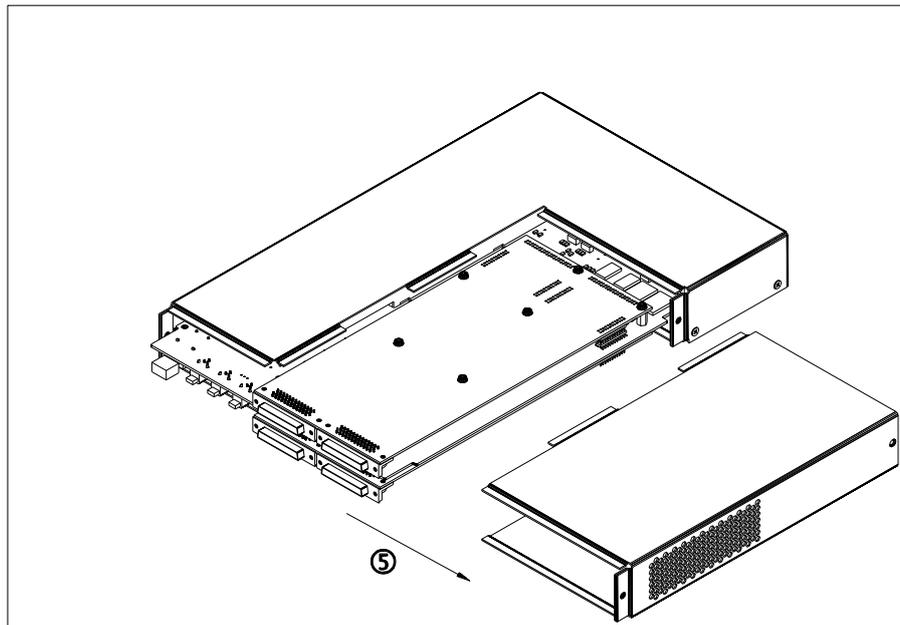
Slide the front bezel off (③) as shown below:



Remove the M3x6mm Torx Countersunk screw (④) attaching the function card cover to the enclosure:

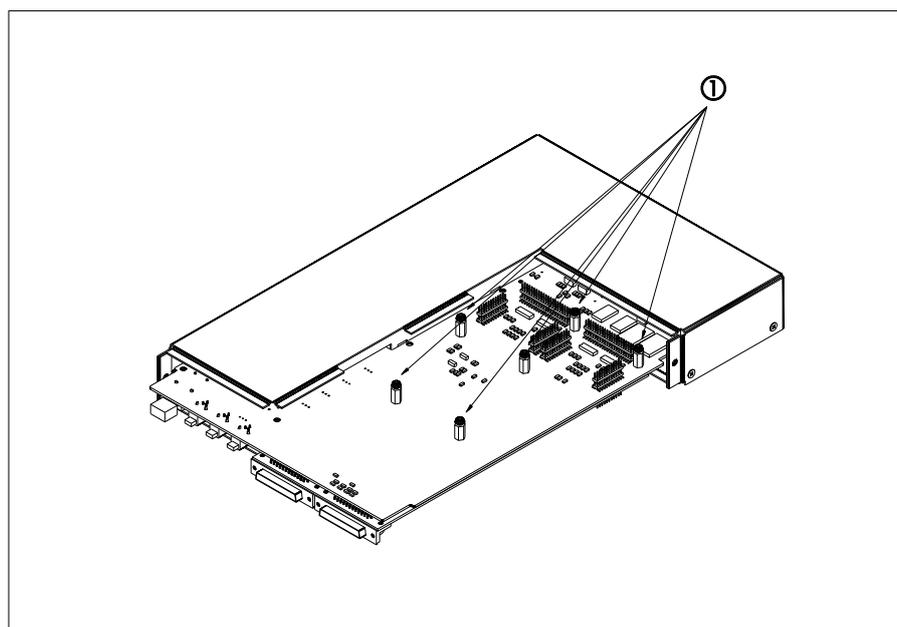


Slide the function card cover off (⑤) as shown below:

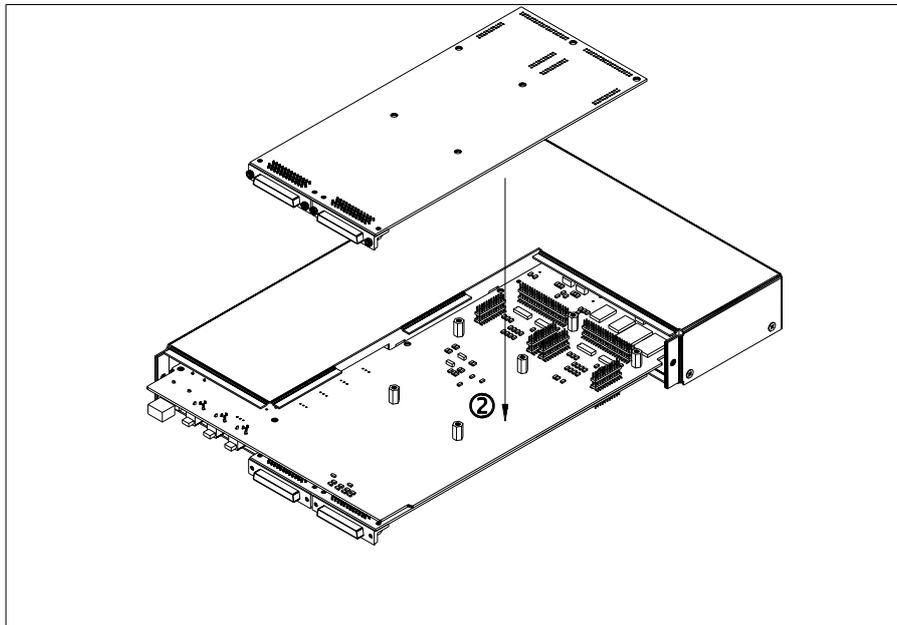


#### 2.4.2. Installing a ProDAQ Function Card

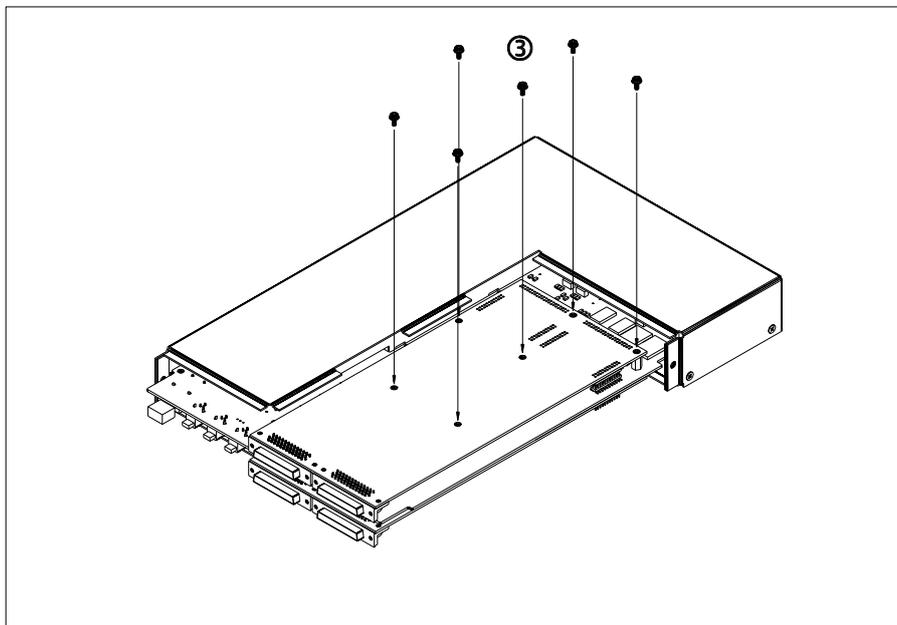
To install a ProDAQ Function Card into the ProDAQ 6100 LXI Function Card Carrier, you must first remove the front bezel and the function card cover as shown previously (see paragraph 2.4.1 Opening the ProDAQ 6100 Enclosure). The ProDAQ Function Cards are mounted inside the ProDAQ 6100 directly on the main PCB. The function cards positions two and four are located on top of the PCB and the positions one and three below. The function cards are mounted face down, e.g. the front-panel connectors as well as the motherboard connectors are underneath the PCB when mounted. Make sure that the M3x6mm screws and washers are removed from the PCB standoffs (①):



Position the function card over the function card slot you want to install it to (②), carefully aligning the connectors connecting it to the ProDAQ 6100 PCB and push it down until it seats fully onto the standoffs of the ProDAQ 6100 PCB:

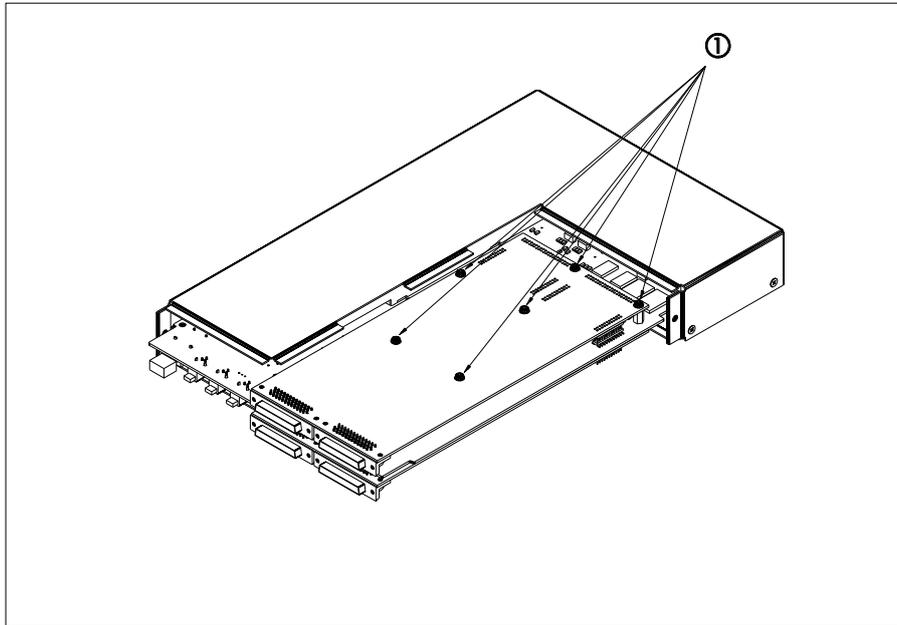


Use three M3x6mm panhead screws and washers (③) to attach the function card to the ProDAQ 6100 PCB (six screws and washers for a double wide function card):

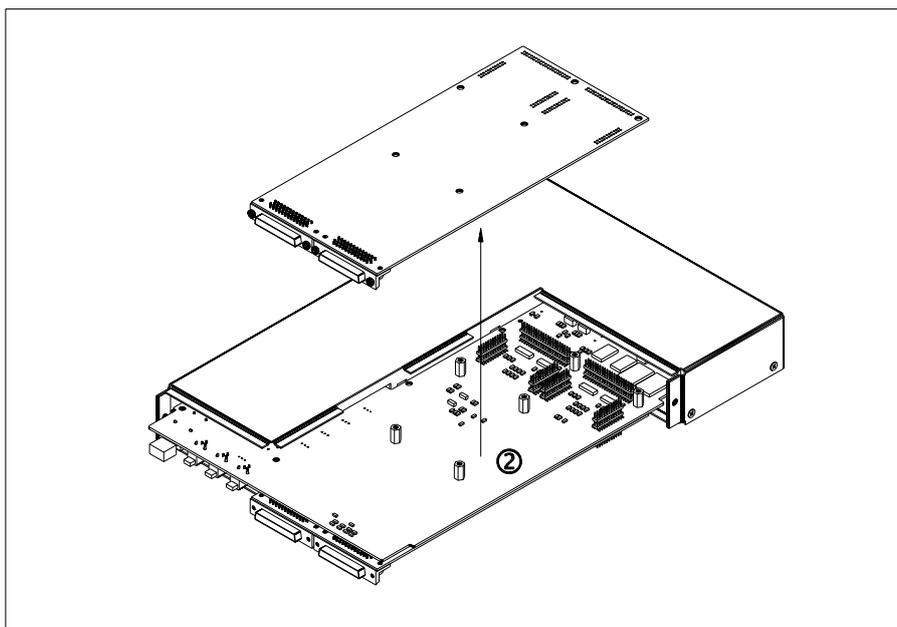


### 2.4.3. Removing a ProDAQ Function Card

If you need to remove an installed function card, remove the three M3x6mm screws (①) mounting them to the base board (six M3x6mm screws for a double wide function card).

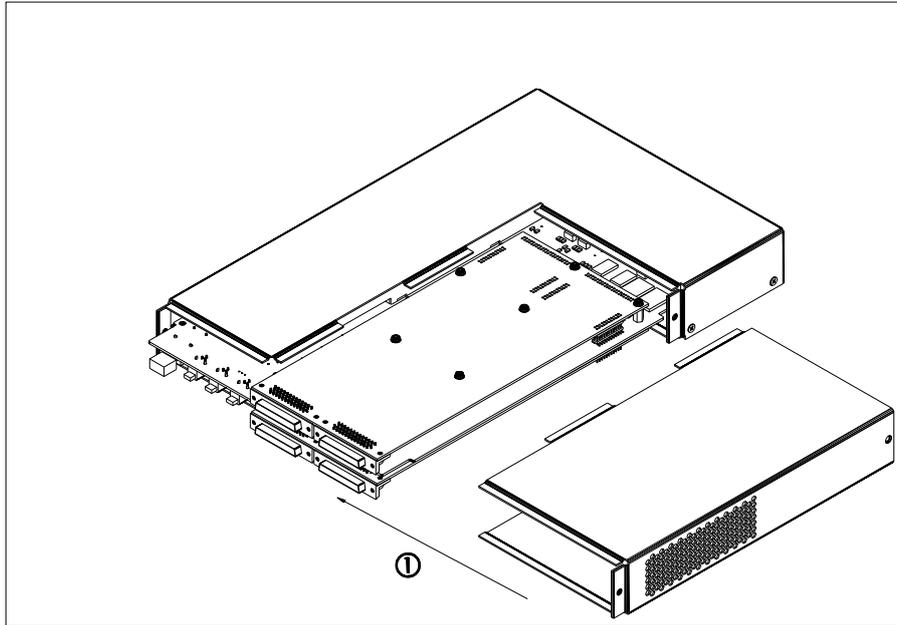


Remove the function card by pulling it (②) straight and evenly upward (or downward for a function card mounted on the bottom of the main PCB). Do not tilt the function card when doing so as it might damage the connectors connecting it to the ProDAQ 6100 PCB.

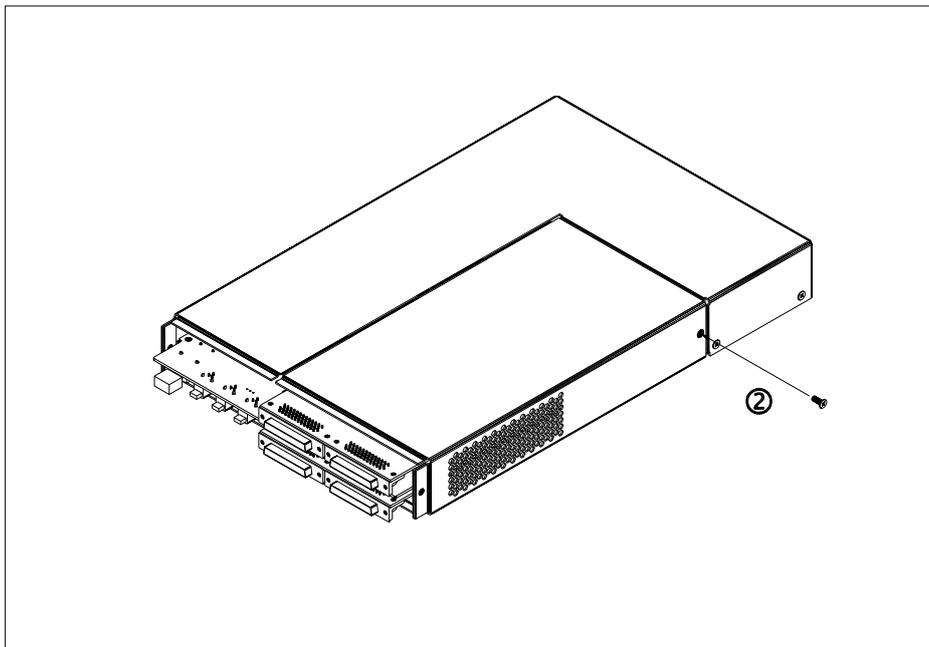


### 2.4.4. Closing the ProDAQ 6100 Enclosure

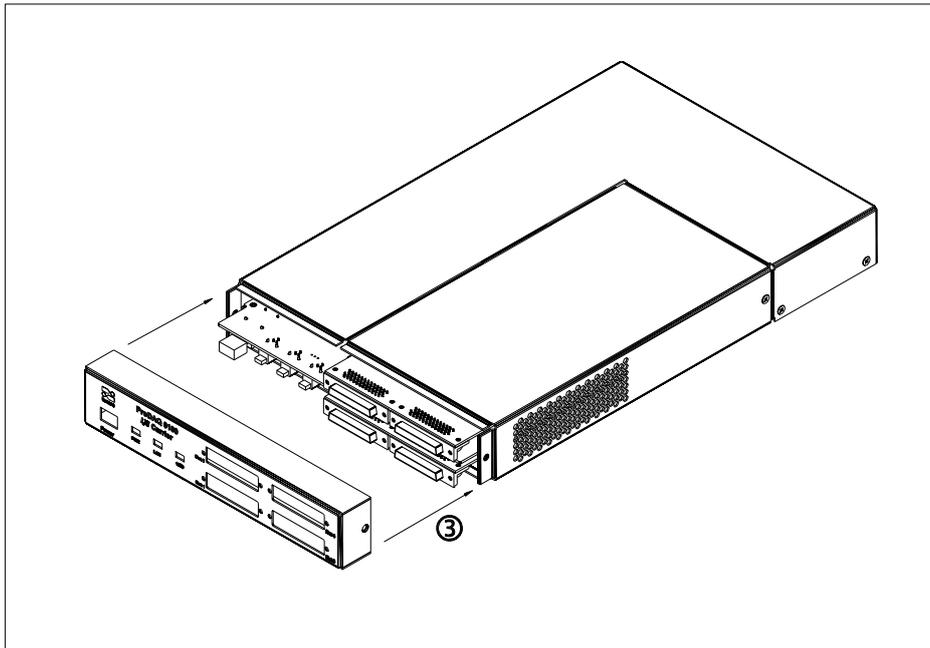
To close the enclosure after installing or removing a ProDAQ function card, first slide back on the function card cover (①):



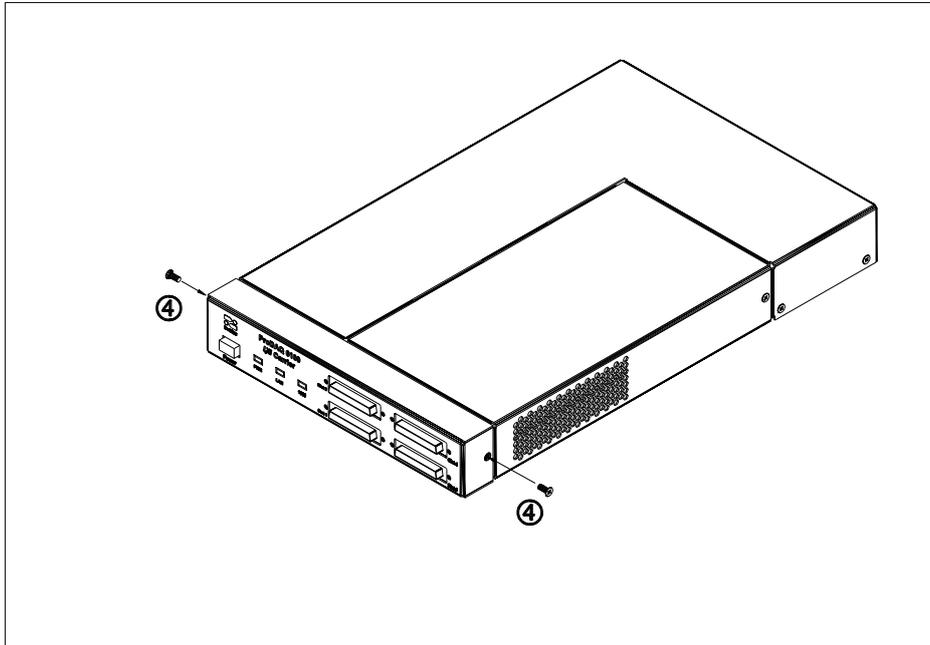
and attach it with a M3x6mm Torx screw to the enclosure:



Make sure that the cutouts for the function card connectors in the front bezel are properly opened or covered by filler panels to match the installed function cards. Slide the front bezel back on (③)



and attach it to the enclosure by two M3x6 Torx screws



This page was intentionally left blank.

## 3. Theory of Operation

### 3.1. Block Diagram

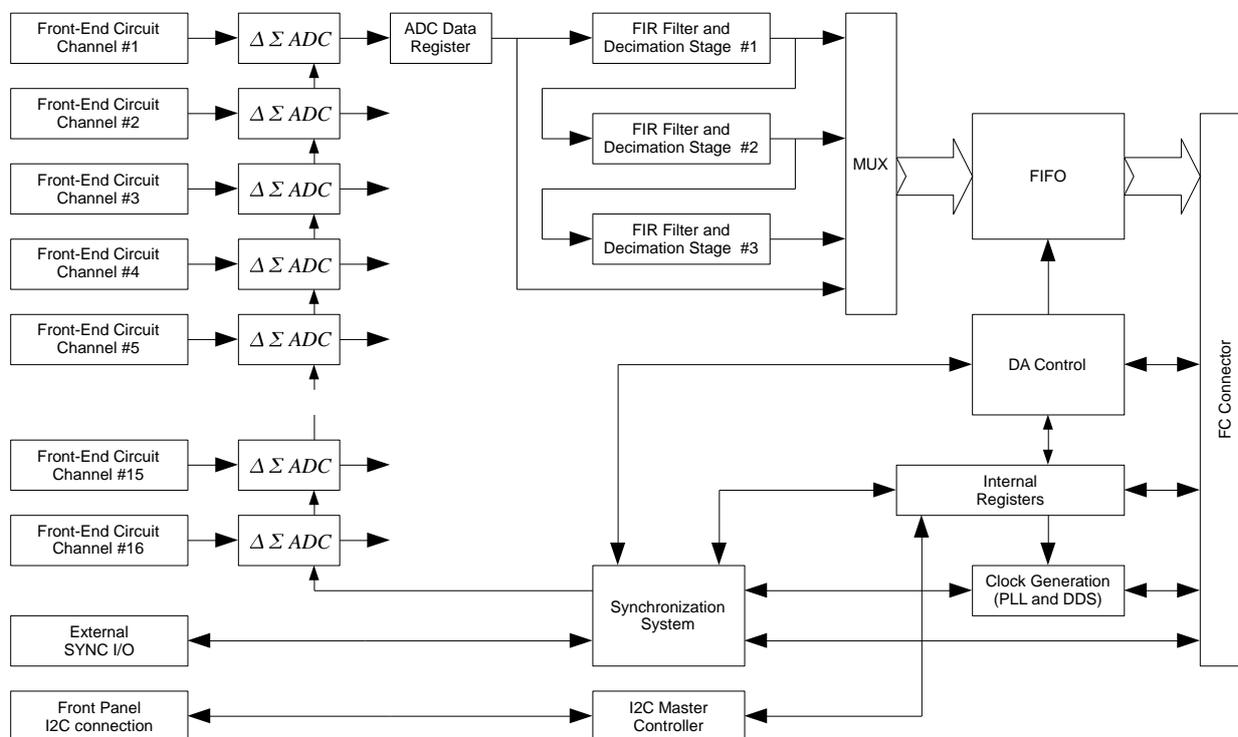


Figure 3 – Simplified Block Diagram

### 3.2. Analog Front-End Circuitry

The ProDAQ 3416 features sixteen fully differential, overvoltage protected, high impedance analog input channels numbered from 1 to 16. Analog front end circuitry condition the input signal, which is later digitized by an ADC.

Figure 3 shows a block diagram of the analog front end circuitry for a single channel. The input signal is first passed through an optional attenuator stage designed to allow for voltages of up to 60V. For calibration purposes, the input of every channel can be connected to the voltage reference bus available on ProDAQ motherboards and function card carriers via a 2:1 multiplexer.

The gain block consists of multiple stages providing gain factors of 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 and 2000; independently software selectable on a per-channel basis. In the following ADC driver block a combined differential driver, level shifter and Butterworth filter in MFB topology prepares the signal to be digitized by the 24-bit ADC available per channel.

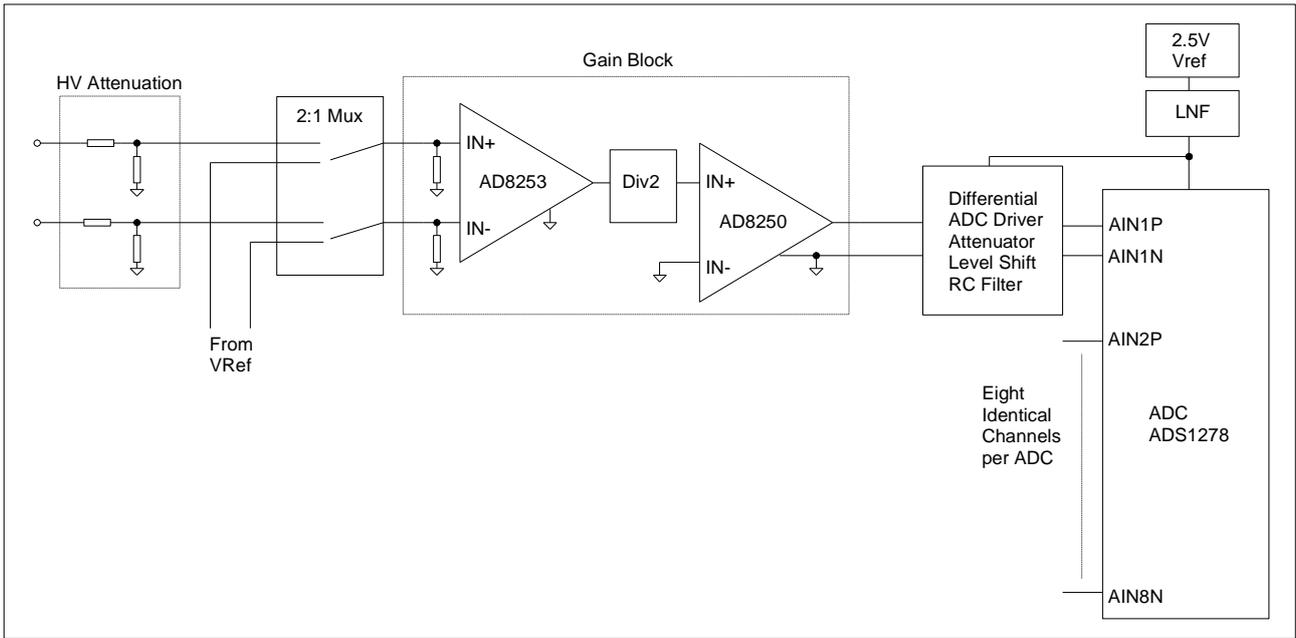


Figure 4 – Analog front-end circuitry (single channel)

### 3.3. TEDS Reader Interface

Transducer Electronic Data Sheet (TEDS) is a nonvolatile memory within a sensor that is utilized for storing information about that sensor. The manufacturer of the sensor deposits into this memory initial information such as manufacturer name, sensor type, model number, serial number, and calibration data. Memory space allocation permits the user to add additional information such as channel ID, location, position, direction, tag number, etc. The protocols and formats of the data are defined by IEEE P1451.4 standard.

The sensor operates in a “mixed mode”, i.e. analog or digital fashion. In the digital mode, the information stored in memory is downloaded. In the analog mode, the sensor functions normally, as a measurement device. A suitable TEDS signal conditioner is used to access the memory digitally, over the same wires ordinarily used for analog measurement signal transmission.

The 3416 card has a common TEDS reader interface circuitry for all sixteen channels. It is brought to the front panel connector on a separate pin. To provide a class 1 TEDS interface, the TEDS line needs to be externally multiplexed onto any of analog input lines.

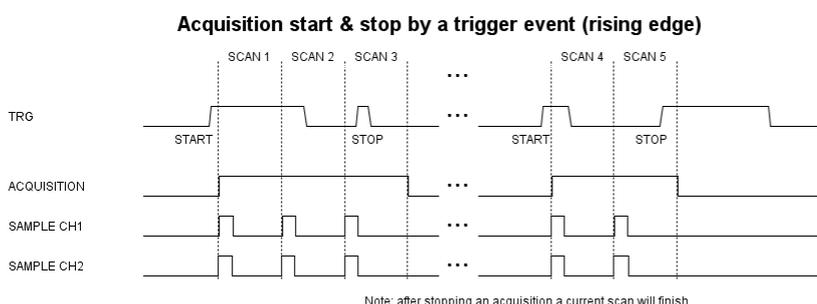
### 3.4. I<sup>2</sup>C Master Interface

The card contains an I<sup>2</sup>C master, which may be used to program an external signal conditioning unit via the two wire bus. For this purpose the bus signals of the master controller are amplified and made available on separate pins on the front panel connector.

### 3.5. Data Acquisition

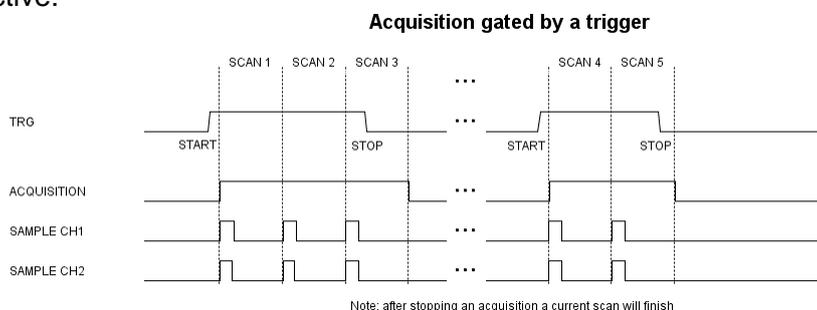
The 3416 function card allows continuous digitizing of up to 16 input channels. The acquired data is streamed into onboard FIFO memory. As each input channel uses its own ADC, all channels are sampled simultaneously. The acquisition process can be started instantly by a host application or by a trigger event after prior arming the acquisition process. The same applies to the end of the acquisition; it can be stopped on a host request or after another trigger event. Alternatively the acquisition may end after collecting a programmed number of samples. A single measurement is possible for externally triggered measurements, in such configuration a trigger event requests a measurement of all enabled input channels on one or more cards.

A DA (data acquisition) trigger event can start the acquisition and a following trigger event can stop the acquisition:



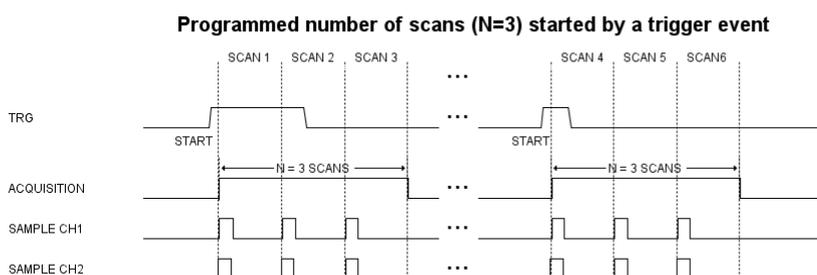
Note: the active trigger edge on the above picture is a rising edge, it can be configured according to an application needs.

Alternatively the DA trigger can act as a gate and the acquisition can take place as long as the DA trigger is active:



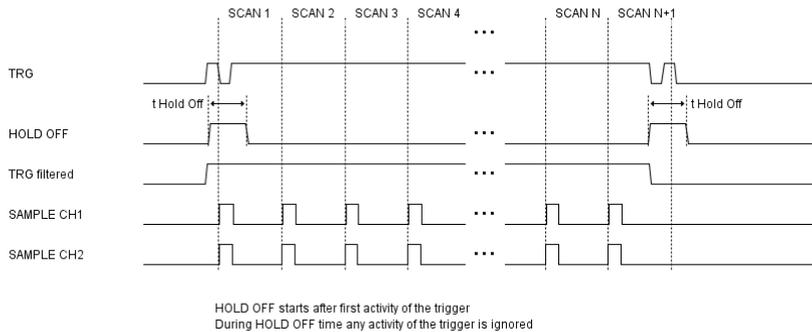
Note: the active trigger level on the above picture is a logical high level, it can be configured.

A DA trigger event can also start an acquisition configured to collect a programmed number of scans. In a default configuration it is infinity and a second DA trigger event stops the acquisition. If a number of samples to be collected is limited than a second DA trigger does not stop the acquisition but instead can retrigger a collection of new scans if a previous set of scans have already been collected.



If a number of scans to be collected is limited and a new DA trigger event happens too early before previous programmed number of samples has been collected an error flag will be set. The error will not appear if a new DA trigger event comes during the last scan of the programmed set of samples so a new acquisition can be triggered at any time after the last scan begun no matter if it has finished yet or is still in progress.

A DA trigger input, which can start and stop the acquisition has implemented a hold off feature, which protects against false trigger events (glitches) occurring near the functional trigger event. The hold off time is fixed and equals to  $1\mu\text{s}$ . False trigger events happening in the hold off zone will not generate an error condition and will be safely ignored.



A data acquisition trigger can be generated internally by a software command or accepted from an external source via the input trigger line on the front panel connector or the motherboard trigger line on the function card interface.

### 3.6. Sampling Settings

The output data rate of the acquisition is common for all input channels and can be set up to 10 kSamples/sec (-Bx versions) or 1 kSample/sec (-Ax versions). The acquisition clock can be generated onboard or can be accepted from an external source via various trigger lines. If it is generated locally, it is generated using the Direct Digital Synthesis (DDS) technique and can be programmed by the user with a very fine resolution, much lower than 1Hz. The locally generated clocks on multiple 3416 or other ProDAQ function cards using the same scheme can be synchronized to each other.

As the sigma-delta ADCs together with a fixed low-pass filter in the input stage can only sample the input data with a rate down to TBD samples/sec, additional FIR filter stages implemented in the on-board programmable logic devices provide additional decimation to allow output data rates down to 1 Sample/sec.

### 3.7. Multiple Cards Synchronization

The 3416 function card samples all 16 channels simultaneously. If more than 16 channels need to be sampled in a synchronous way and the acquisition started at the same time on all channels than multiple cards can be synchronized together.

The ProDAQ 3416 can be set as a master or a slave. If the 3416 works in stand-alone mode it need always to be configured as master. If a number of the 3416 cards should work in a multiple-card synchronization mode then card is configured as master and all other as slaves. For the synchronous acquisition, the master card generates two signals, which have to be distributed to all slaves: a clock signal and a sync signal.

## 4. Specifications

### 4.1. Input Characteristics

Number of Channels	16
Input Type	Differential
Coupling	DC
Full Scale Signal Ranges	$\pm 5$ mV, $\pm 10$ mV, $\pm 20$ mV, $\pm 50$ mV, $\pm 100$ mV, $\pm 200$ mV, $\pm 500$ mV, $\pm 1$ V, $\pm 2$ V, $\pm 5$ V and $\pm 10$ V (plus 5% for hardware calibration and over-range capability)
Gain Settings	1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000
Analog Input Filter Type	2-pole Butterworth
Input Impedance	> 10 M $\Omega$ , 25 pF
Input Protection	$\pm 25$ V
Input Offset Voltage (typical)	$\pm 30$ $\mu$ V typ. (gain 1) $\pm 6$ $\mu$ V typ. (gain 2000) $\pm 80$ $\mu$ V typ. (-Cx versions, gain 1, >25 kS/s)
Gain Error (typical)	typ. 0.002% (gain 1) typ. 0.05% (gain 2000) typ. 0.004% (-Cx versions, gain 1, >25 kS/s) typ. 0.1% (-Cx versions, gain 2000, >25 kS/s)
INL (Best Fit Method)	$\pm 0.0003\%$ FSR typ. $\pm 0.0012\%$ FSR max.
DC Accuracy	$\pm(8 + 225/\text{gain})$ $\mu$ V (typ.) $\pm(20 + 600/\text{gain})$ $\mu$ V (max.) $\pm(25 + 800/\text{gain})$ $\mu$ V (max., -Cx, >25 kS/s)
Common-mode Rejection Ratio	87 dB typ. (gain 1) 106 dB typ. (gain 2000)
0.1dB Analog Passband	DC to 450 Hz (-Ax versions) DC to 4.5 kHz (-Bx versions) DC to 23.7 kHz (-Cx versions)
3dB Analog Bandwidth ( $f_c$ )	DC to 490 Hz (-Ax versions) DC to 4.9 kHz (-Bx versions) DC to 25.8 kHz (-Cx versions)
Pass Band Ripple	$\pm 0.005$ dB
Stop Band Attenuation	95 dB min.
Signal-to-Noise Ratio	105 dB typ. (97.7 Hz, -1 dB <sub>FS</sub> )
Signal, Noise And Distortion (SINAD)	100 dB typ.
Total Harmonic Distortion (THD)	-102 dB typ. (1 kHz, -1 dB <sub>FS</sub> )
Spurious-free Dynamic Range	>103 dB
Noise	35 $\mu$ V RMS typ. (1 kHz bandwidth, gain 1) 0.3 $\mu$ V RMS typ. (1 kHz bandwidth, gain 2000)

Crosstalk	116 dB typ.
-----------	-------------

## 4.2. Sampling

Resolution	24 bit
ADC Type	Sigma-Delta
Output Data Rates	1 S/s to 1 kS/s (-Ax versions) 1 S/s to 10 kS/s (-Bx versions) 5 kS/s to 52 kS/s (-Cx versions)
Rate Selection Resolution	0.01 S/s
Oversampling	128 x
FIFO	10 kSamples

## 4.3. Triggering

Trigger Input	Motherboard or Front Panel Connector
Signal Type	TTL
Active Level	Low
Minimum Pulse Width	100 ns

## 4.4. Synchronization

Clock and Sync I/O	Motherboard or Front Panel Connector
Signal Type	TTL
Active Level	Low
Clock Input Frequency	2 MHz Reference Clock for DDS or ADC clock

## 4.5. Environmental Specifications

Power Consumption	9.7 W max.
Dimensions	230 x 52.6mm
Weight	TBD
Temperature	0 °C to +50 °C (operational) -40 °C to +70 °C (storage only)
Humidity	10% - 90% (non-condensing)
Altitude	n/a
Shock and Vibration	n/a
Warm-up Time	30 Min.

## 5. The VXIplug&play Driver

### 5.1. Installation

The ProDAQ 3416 16-Ch. Sigma-Delta ADC function card is supplied with a VXIplug&play driver. To install the driver, run the “Setup.exe” application coming with it and follow the instructions presented. Make sure that no other ProDAQ software is running when you start the setup.

The installation program will by default perform a complete installation. It will install the driver files in the directory defined by the %VXIPNPPATH% environment variable and shortcuts into the VXIPNP program group of the start menu. To choose a different path and/or custom installation options is not recommended and may result in malfunctioning of the soft front panel and any application trying to use the driver.

### 5.2. The Soft Front Panel

The purpose of soft front panel application is to demonstrate the instrument’s abilities. After the start of the soft front panel application, the user has the choice to either enter the address information (VISA resource specification and function card number) of the function card the soft front panel application shall connect to or use the build-in auto find functionality to discover accessible ProDAQ 3416 function cards.

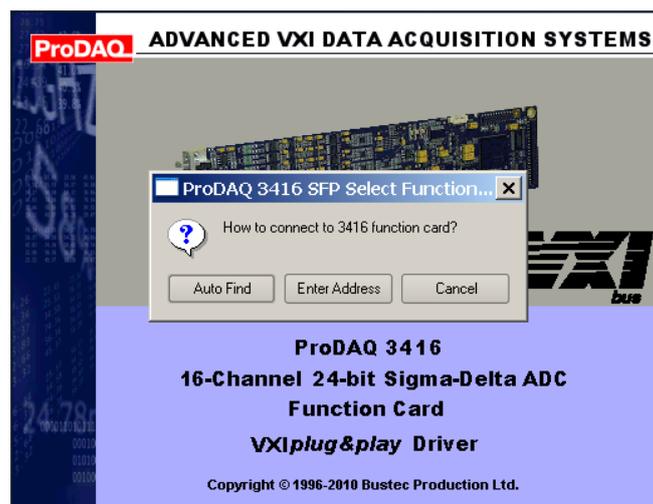


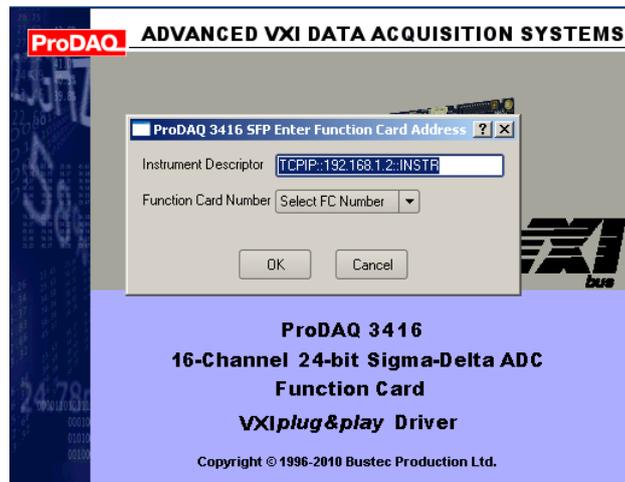
Figure 5 - Selecting the Connection Method

Please note that the auto find functionality will only inspect network resources that are known to the VISA library to avoid unwanted accesses of network resources that might be unintentionally reachable via the local network. For VXIbus resources, running the VISA resource manager prior to running the soft front panel application is necessary for both the auto find functionality to work and in general the access to the function card to be possible.

If “Auto Find” is selected, the user will be presented with a dialog box showing all available ProDAQ 3416 function cards, allowing the selection of one function card to connect to. The soft front panel is not designed to handle more than one function card at a time. If there is only one function card available, the dialog box will not appear and the soft front panel application will automatically establish the communication to this instrument. If no ProDAQ 3416 is available in your system, the soft front panel application can be run in demo mode, allowing to operate all controls as if connected to a 3416.

If “Enter Address” is selected, the user is presented with a dialog box that allows entering the VISA resource string and the function card number directly:

Figure 6 – Specifying the Function Card Address



The resource string and range of function card numbers differ depending on the ProDAQ Motherboard or Carrier the ProDAQ 3416 is installed on. Please refer to the motherboard/carrier user manual for more information.

---

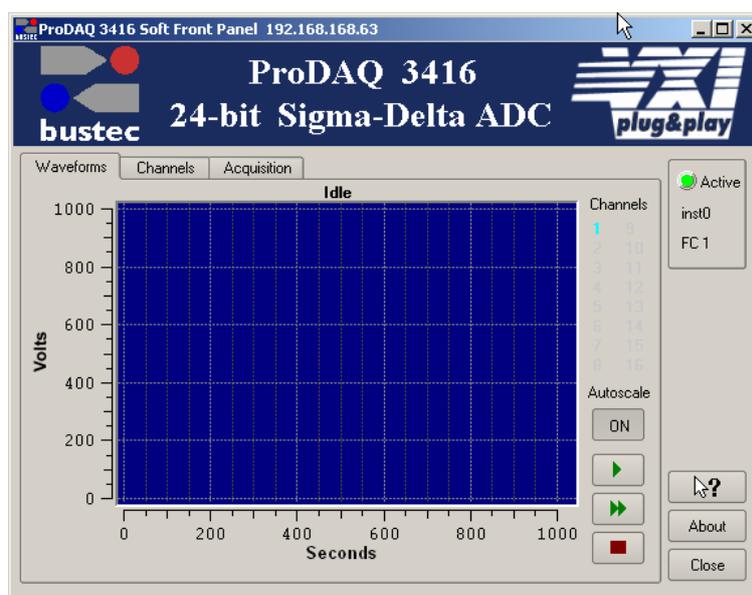
#### Note

**In some systems it might be necessary to register the network instrument via the Configuration Utility coming with the Bustec VISA before the function card can be found via “Auto Find” or accessed using a TCPIP resource descriptor.**

---

After initializing the ProDAQ 3416 function card, during which a splash screen is displayed, the soft front panel window will appear (see Figure 7 - ProDAQ 3416 Soft Front Panel Application).

Figure 7 - ProDAQ 3416 Soft Front Panel Application



### 5.2.1. “Waveforms” Tab

The “Waveforms” tab, which is shown by default, allows the user to acquire and display data from enabled channels (see 5.2.2 - “Channels” Tab). Each time the start button (  ) is clicked, the soft front panel application acquires a block of data as specified by the settings in the “Acquisition” tab (see 5.2.3 - “Acquisition” Tab) and displays it. If the run button (  ) is clicked, the soft front panel application continuously acquires blocks of data until the stop button (  ) is clicked.

With the “Autoscale On/Off” button the user can select whether the graph display is automatically scaled to the acquired signal or whether a constant scaling should be used. Clicking on the graph display with the right mouse key and dragging the mouse to select an area will allow the user to zoom in on the data.

### 5.2.2. “Channels” Tab

The “Channels” tab contains a set of tabs, one for each channel. Each channel tab allows the user to select the input source for the channel as well as the gain and to choose, whether the channel should be included when data is acquired.

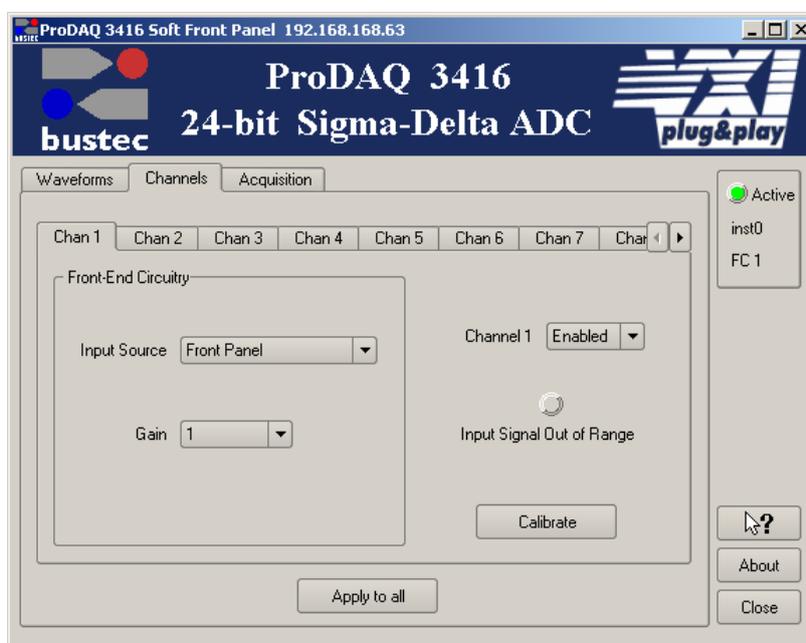


Figure 8 – Channel Configuration

The “Input Source” for each channel can be selected to be either the front panel connector or the voltage reference bus from the ProDAQ motherboard or carrier. If no voltage reference option is installed on the motherboard or carrier, selecting the voltage reference bus as input should be avoided. The gain is selectable on a per channel basis between 1 and 2000 in steps of 1, 2 and 5.

The button “Apply to all” will apply the current tabs settings for input source, gain and channel enable/disable to all channels.

### 5.2.3. “Acquisition” Tab

The “Acquisition” tab allows the user to specify the parameter for the acquisition started by the start or run button on the “Waveform” tab.

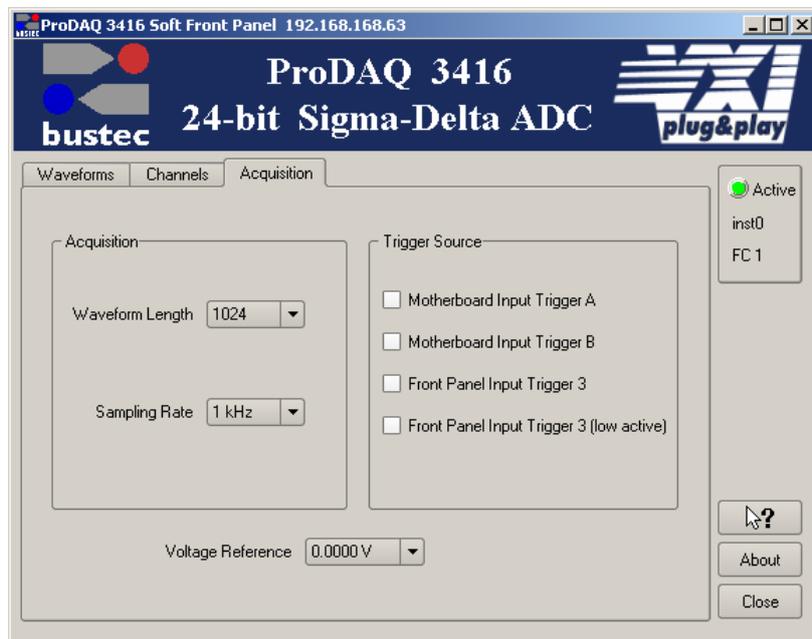


Figure 9 – Acquisition Configuration

The “Waveform Length” parameter determines how many samples per channel will be acquired each time the start button is pressed. The “Sampling Rate” selects the common sampling rate for all channels.

By default the acquisition starts immediately after the user presses the start or run button on the “Waveform” tab. By selecting one of the trigger sources the user can specify the acquisition to wait for a start trigger. Please note that if one of the motherboard input triggers is chosen, the motherboard or carrier must be configured separately to route a trigger to the function card in addition.

For convenience the optional voltage reference of the motherboard or carrier can be directly controlled from the ProDAQ 3416 soft front panel application. Selecting one of the voltages or ground via the “Voltage Reference” drop down selector will allow to sample this voltage on all channels which are configured for this input source.

## 6. Programming the ProDAQ 3416

This chapter shows how to program the ProDAQ 3416 function card using the *VXIplug&play* driver. Complete examples can be found in the “Examples” subdirectory of the driver. All functions are explained in detail in the help file coming with the driver.

### 6.1. *VXIplug&play* Driver Organization

The *VXIplug&play* driver is organized in a hierarchical manner to allow the user to quickly choose the function calls to solve the task at hand without being confronted with unnecessary details. Besides the standard connection/disconnection and utility functions it contains different levels of functionality which provide single functions or sets of functions to solve a particular data acquisition task:

Function Tree Layout:	
<b>ProDAQ 3416 16-ch 24-bit Sigma Delta ADC</b>	
Initialization	bu3416_init
Select Function Card	bu3416_fcSelect
Initialization with Parameters	bu3416_paramInit
<b>Hardware Configuration</b>	
...	
<b>Single-Card Acquisition</b>	
<b>Single-shot Acquisition</b>	
...	
<b>Continuous Acquisition</b>	
...	
<b>Multi-Card Acquisition</b>	
...	
<b>Low-Level Access</b>	
...	
<b>Utility Functions</b>	
Get Serial Number	bu3416_getSerNum
Reset	bu3416_reset
Self Test	bu3416_self_test
Error Query	bu3416_error_query
Error Message	bu3416_error_message
Revision Query	bu3416_revision_query
Close	bu3416_close

Figure 10 – *VXIplug&play* Driver Organization

The section **Hardware Configuration** contains high-level functions to configure the card (e.g. gain settings). The different sections **Single-Card Acquisition** (with the sub-sections for **Single-shot Acquisition** and **Continuous Acquisition**) and **Multi-Card Acquisition** contain functions or sets of functions to quickly program the card for different acquisition tasks. The functions from the different sections should be used together per section and not be mixed.

The section **Low-level Access** contains functions that directly change settings on a register level and are used by the higher level functions to implement their functionality. Using them directly in combination with the higher level functions might interfere with the functionality implemented and should be avoided. In general the usage of the low-level functions will require an intimate knowledge of the ProDAQ 3416 hardware as well as the hardware of the ProDAQ motherboards and function card carriers. Before you attempt to implement your data acquisition or test application using them, it is recommended to study their usage in the higher level functions in the driver sources and/or contact Bustec for support.

The following paragraphs will explain the usage of the high level functions:

## 6.2. Connecting to the Function Card

To initialize the driver and connect to the ProDAQ motherboard or function card carrier, the standard *VXIplug&play* initialization function `bu3416_init()` is used (see Figure 11, ①). (Please refer to the *VXIplug&play* standard VPP-4.3, section 4.3 for a detailed description of the address string used.)

After initializing the driver and connecting to the motherboard or carrier, the driver must be told which one of the function cards to work with. This is done by the function `bu3416_fcSelect()`. It takes as an argument the session established via the function `bu3416_init()`, the function card number and a boolean value specifying whether to reset the selected function card (see Figure 11, ②).

```
#include <visa.h>
#include <bu3416.h>

main (int argc, char **argv)
{
    ViStatus status;
    ViSession session;
    ViChar descr[256];

    ① /* connect to a ProDAQ motherboard in a VXIbus system */
    if ((status = bu3416_init("VXI0::2::INSTR", VI_TRUE, VI_TRUE, &session)) != VI_SUCCESS)
    {
        viStatusDesc (rm_session, status, descr);
        printf ("Error: bu3416_init() failed due to %s\n", descr);

        return -1;
    }

    ② /* use function card in position/slot 1 */
    if ((status = bu3416_fcSelect(session, 1, VI_TRUE)) != VI_SUCCESS)
    {
        viStatusDesc (instr_session, status, descr);
        printf ("Error: bu3416_fcSelect failed due to %s\n", descr);

        return -1;
    }

    /* OR: connect to a 3416 in position 1 in a LXI function card carrier */
    ③ if ((status = bu3416_paramInit("TCP/IP::192.168.168.63::INSTR",
                                   1, VI_TRUE, VI_TRUE, &session)) != VI_SUCCESS)
    {
        viStatusDesc (rm_session, status, descr);
        printf ("Error: bu3416_paramInit() failed due to %s\n", descr);

        return -1;
    }

    /* ... */
}
```

Figure 11 - Opening a Session

For your convenience, the driver contains a new function called `bu3416_paramInit()`, which combines the functionality of the `bu3416_init()` and `bu3416_fcSelect()` functions by extending the argument list of the standard initialization function with a parameter specifying the function card number (see Figure 11, ③).

For the driver functions to work properly, you will either have to use the function `bu3416_paramInit()` to open a session with the device, or you will have to call the function `bu3416_fcSelect()` after calling the function `bu3416_init()` and before any other driver function is called.

To close a session with the ProDAQ 3416 16-Ch. Sigma/Delta ADC function card, the standard VXiplug&play function `bu3416_close()` must be used.

### 6.3. Hardware Configuration

The input multiplexer and gain stages on the ProDAQ 3416 function card are configured using the function `bu3416_setChanConfig()`. It takes as arguments the session to the instrument, a channel number, a selection for the input multiplexer and a value for the gain setting. The channel number has to be an integer number in the range of 1...16 to select one of the channels or 0 for applying the configuration to all channels. Predefined macros from the include file `bu3416.h` can be used (`bu3416_CHAN_1` to `bu3416_CHAN_16` or `bu3416_CHAN_ALL`). The input multiplexer can be set to either connect the channel's input to the front panel connector or to the internal voltage reference bus. The selection can be made by using an integer value of 0 (front panel connector) or 1 (voltage reference bus) or again by using a macro predefined in `bu3416.h` (`bu3416_CH_FP` or `bu3416_CH_VREF`). The gain can be set in steps of 1, 2, 5 between 1 and 2000 by either using valid integer numbers (1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000) or by using the predefined macros `bu3416_GAIN_1` to `bu3416_GAIN_2000` (see Figure 12, ①).

If the acquisition shall be started by a hardware trigger, the trigger used for this purpose can be selected by using the function `bu3416_setTrigConfig()`. The trigger can be received from either the ProDAQ function card bus (`bu3416_DA_TRIG_MBA` and `bu3416_DA_TRIG_MBB`) or the front panel connector (signal `FP_TRG_IO_3`, see 0

). If the front panel connector input is used, the trigger used can be low- or high-active (`bu3416_DA_TRIG_FP3_LOW` or `bu3416_DA_TRIG_FP3`). The type of the parameter is integer and might be specified again either directly as a value or by using the predefined macros from the include file `bu3416.h`. The driver help file `bu3416.hlp` lists also both forms. Please note that the usage of the function card bus trigger lines will require you to configure their routing in the ProDAQ motherboard or function card carrier in addition.

### 6.4. Single-Card Acquisition

#### 6.4.1. Single-shot Acquisition

To acquire a consecutive number of samples from a single channel or several channels, the functions `bu3416_acquireWaveform()` (see Figure 12, ②) and `bu3416_acquireWaveforms()` (see Figure 12, ③) can be used. These functions implement the complete functionality of configuring the card, starting the acquisition, waiting for the end of the acquisition and transferring the data back to your application.

The functions take either a channel number or a channel mask as an argument to specify which channel or group of channels to acquire data from. In addition the sample rate in samples/sec/channel, a number of samples to specify the consecutive number of samples that will be acquired per channel and an output array used to store the waveform(s):

```

{
    ViSession session;
    ViInt16 mask;
    ViReal64 waveform[10240];

    /* .... */

    /* configuring all channels for gain 10, front panel connector input */
    ① if ((status = bu3416_setChanConf (session, bu3416_CHAN_ALL,
                                     bu3416_GAIN_10, bu3416_CH_FP)) < VI_SUCCESS)
    {
        bu3416_error_message (rm_session, status, descr);
        printf ("Error: bu3416_acquireWaveform() failed due to %s\n", descr);

        return -1;
    }

    /* acquire a waveform of 1024 samples from channel 3 at 1 kSa/s */
    ② if ((status = bu3416_acquireWaveform (session, 3, 1000.0, 1024, waveform)) < VI_SUCCESS)
    {
        bu3416_error_message (rm_session, status, descr);
        printf ("Error: bu3416_acquireWaveform() failed due to %s\n", descr);

        return -1;
    }

    /* acquire waveforms from channels 1-8, 12, and 13 */
    ③ mask = 0x18FF;
    if ((status = bu3416_acquireWaveforms (session, mask, 1000.0, 1024,
                                           bu3416_GROUP_BY_CHANNEL, waveform)) != VI_SUCCESS)
    {
        bu3416_error_message (rm_session, status, descr);
        printf ("Error: bu3416_acquireWaveforms() failed due to %s\n", descr);

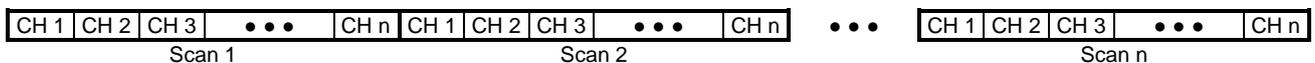
        return -1;
    }

    /* ... */
}

```

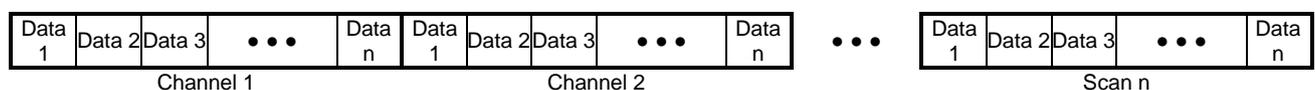
Figure 12 – Acquiring a Waveform

The function `bu3416_acquireWaveforms()` has an additional argument specifying the arrangement of the data in the output array. The function card is storing the data interleaved in the on-board FIFO. So the arrangement of the data as read from the on-board FIFO is



The number of values per scan depends on the number of channels enabled in the channel mask. If for example channels 1-8, 12 and 13 as in the above example are enabled, each scan delivers 10 values.

This is also the arrangement of the data in the output array when the parameter `fillMode` is specified as `bu3416_GROUP_BY_SCAN`. But most of the time it is more convenient to have the data arranged on a per channel basis. Therefore, the function `bu3416_acquireWaveforms()` will rearrange the data while transferring it to the output array when the parameter `fillMode` is specified as `bu3416_GROUP_BY_CHANNEL`. The result is an arrangement like



The complete number of samples as specified by the parameter *scans* for the first enabled channel is placed into the output array, then the complete number of samples for the second enabled channel and so on.

#### 6.4.2. Continuous Acquisition

To acquire data continuously, the ProDAQ 3416 needs to be configured for scanning the input channels and moving the data into the on-board FIFO. The FIFO memory stores the data until the host computer is ready to read out the data. The timing for this asynchronous read-out depends on the amount of data in the FIFO.

The driver function `bu3416_setAcquisitionMode()` can be used to configure the card for the acquisition. The parameter *mask* defines which channels should be enabled. The parameter *sampleRate* defines the scan rate used in samples per second per channel. The parameter *scansToCollect* can be used to limit the total amount of samples acquired. If 0 (zero) is specified, the acquisition will continue until stopped by using `bu3416_stopAcquisition()`. The parameter start mode specifies whether `bu3416_startAcquisition()` shall start the acquisition immediately or whether it should wait for the “start” trigger (see 6.3 - Hardware Configuration). Last not least the parameter *stopOnError* defines whether the data acquisition is stopped when an error occurs.

```

{
    ViSession session;
    ViStatus status;
    ViInt16 mask;

    /* .... */

    /*
     * configure the ProDAQ 3416 for continuous acquisition of channels 1...4,
     * 1000 Sa/s/ch, start mode 'immediate' and stop on all errors:
     */
    mask = 0x000f;
    If ((status = bu3416_setAcquisitionMode (session, mask, 1000.0, 1000000,
                                           bu3416_DA_START_IMM, bu3416_DA_STOP_ERR_ANY)) < VI_SUCCESS)
    {
        /* error handling ... */
    }

    /*
     * Start the asynchronous acquisition as configured above:
     */
    if ((status = bu3416_startAcquisition (session)) < VI_SUCCESS)
    {
        /* error handling ... */
    }

    /* ... */
}

```

Figure 13 – Starting the Asynchronous Acquisition

To read out the acquired data at the right time, the application needs to poll the status of the acquisition using the function `bu3416_checkAcquisition()`. The function returns the acquisition state, errors that may occur during the acquisition (e.g. over-range error) and the number of scans available for readout. Due to the hardware synchronisation support for multi-card configurations and the requirements of the Sigma-Delta ADC, the state machine on the function card uses number of states before the card is ready for sampling. These states (`bu3416_SM_DDSUD`, `bu3416_SM_SYNC`) will only be returned in case an error happened and must not be used in the application to follow the progress of the state machine.

If no error occurs, the state machine will either go to the state `bu3416_SM_READY`, if the acquisition is configured to wait for a trigger, or directly to the state `bu3416_SM_POST`. In this state the ProDAQ 3416 is acquiring data and storing it in the FIFO.

```

{
    ViSession session;
    ViStatus status;
    ViInt16 state, error;
    ViInt32 backlog, nread, remaining;
    ViInt32 *waveforms;

    /* .... */

    /* wait for the ProDAQ 3416 to acquire data */
    do
    {
        status = bu3416_checkAcquisition (session, &state, &error, &backlog);

        if (error != 0)
        {
            /* handle error, break loop ... */
        }
    }
    while (state < bu3416_SM_POST);

    /*
     * read out the data. Acquisition will stop automatically when total number
     * of samples is reached:
     */
    do
    {
        status = bu3416_checkAcquisition (session, &state, &error, &backlog);

        if (error != 0)
        {
            /* handle error, break loop ... */
        }

        if (backlog > 1024)
        {
            status = bu3416_readAcquisition (session, 1024, bu3416_GROUP_BY_CHANNEL,
                                             &remaining, &nread, waveforms);
        }
    }
    while (state == bu3416_SM_POST);

    /* ... */
}

```

Figure 14 – Checking the Status of the Acquisition and Data Read-out

When data is available, the function `bu3410_readAcquisition()` can be used to read out the data acquired. It takes as parameters the number of scans to read, the fill mode as described above for the function `bu3410_acquireWaveforms()` and a pointer to the data buffer. It also returns the actual number of scans read and the number of scans still in the on-board FIFO. Depending on the timing, it may be necessary to continue reading data after the ProDAQ 3416 has stopped acquiring data to read the data remaining in the FIFO.

If you want to use an asynchronous callback instead of polling, you will need to use the function `bu3416_startAcquisitionEx()` to specify a callback function and a threshold. The driver will then configure the card to generate an asynchronous event that will activate the callback function whenever the amount of data available reaches the specified threshold. The callback function must be of the type `bu3100_irqHandler_t`, see `bu3100.h`. As this is a generic handler function used for all ProDAQ functions cards, you still need to use the function `bu3416_checkAcquisition()` inside the callback function to check for errors and `bu3416_readAcquisition()` to read the data. See the example “AsynchAcquisition” coming with the driver for a complete example how to use these functions.

## 6.5. Calibration

The ProDAQ 3416 comes factory calibrated. Yet, to achieve the highest accuracy possible, it is recommended to calibrate the ProDAQ 3416 before starting an acquisition by using the optional voltage reference which can be installed on ProDAQ motherboards and function card carriers. To perform the calibration, the driver provides the function `bu3416_calibrateBoard()`. The results of the run-time calibration are stored on the card and used for further acquisitions by the gain and offset correction stage in the hardware, but get lost again when the card is powered off. If the motherboard or carrier housing the ProDAQ 3416 function card is not equipped with a voltage reference, the function returns an error. Please note that the calibration you will need to configure first the gain for the channels before calibrating.

This page was intentionally left blank.

## Appendix A: Front-panel Connector

The front panel connector used on the ProDAQ 3416 is a high-density 50-pin female SCSI connector with the following pin-out:

Signal	A	B	Signal
FP_TRG_IO_1	1	26	GND
FP_TRG_IO_2	2	27	GND
FP_TRG_IO_3	3	28	GND
TEDS	4	29	GND
I2C_DET	5	30	GND
I2C_SCL	6	31	GND
I2C_SDA	7	32	GND
n.c.	8	33	n.c.
Vref-	9	34	Vref+
In 16-	10	35	In 16+
In 15-	11	36	In 15+
In 14-	12	37	In 14+
In 13-	13	38	In 13+
In 12-	14	39	In 12+
In 11-	15	40	In 11+
In 10-	16	41	In 10+
In 9-	17	42	In 9+
In 8-	18	43	In 8+
In 7-	19	44	In 7+
In 6-	20	45	In 6+
In 5-	21	46	In 5+
In 4-	22	47	In 4+
In 3-	23	48	In 3+
In 2-	24	49	In 2+
In 1-	25	50	In 1+

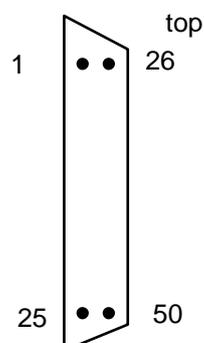


Figure 15 - Front panel connector as seen when the card is fitted in the module.

<b>Signal</b>	<b>Description</b>
FP_TRG_IO_N	Front panel trigger inputs
TEDS	TEDS Interface. This signal needs to be multiplexed into the inputs signal paths for sensor readout.
I2C_DET	I <sup>2</sup> C Detection Signal
I2C_SCL	I <sup>2</sup> C Clock Signal
I2C_SDA	I <sup>2</sup> C Data Signal
VRef+/VRef-	Buffered voltage reference bus output.
In N+/In N-	Differential channel inputs.

## Appendix B: Register Description

All addresses are given in hexadecimal notation. FC\_ADR is address in the function cards address space. VXI\_ADR is address in VXI address space (refer to the motherboard manual for more details).

---

### WARNING

**Writing directly to the registers of the function card can cause unexpected behavior and/or may render the card unusable (e.g. by overwriting calibration values). Please use the function card driver to access the card!**

---

FC_ADDR	VXI_ADDR	Register Name	Access	Function
0	00	FCID	RO	FC ID register
1	04	FCVER	RO	FC version register
2	08	FCCSR	RW	Function card Control and Status register
3	0C	MODE1	RW	
4	10	MODE2	RW	
5	14	OTRI	RW	Output trigger control register
6	18	ITRI	RW	Input trigger control register
7	1C	<i>reserved</i>		
8	20	DDS_WX	RW	DDS control register
9	24	OCOEFLL	RW	Offset coefficient register (lower part)
A	28	OCOEFH	RW	Offset coefficient register (upper part)
B	2C	GCOEFL	RW	Gain coefficient register (lower part)
C	30	GCOEFH	RW	Gain coefficient register (upper part)
D	34	<i>reserved</i>		
E	38	I2C_CTRL	RW	I2C control register
F	3C	TEDS_ACC	RW	TEDS access register
10	40	FIFO_CTRL	RW	FIFO control register
11	44	FIFO_AFT	RW	FIFO almost full flag threshold register
12	48	FIFO_WR	RW	FIFO Write register
13	4C	SIG_ERR	RO	Signal error register
14	50	<i>reserved</i>		
15	54	ERROR	RW	Second error register
16-18		<i>reserved</i>		
19	64	POSTT_NOSL	RW	Number of samples for DA (lower part)
1A	68	POSTT_NOSH	RW	Number of samples for DA (upper part)
1B-1F		<i>reserved</i>		
20	80	CHN1CFG	RW	Channel #1 configuration register
21	84	CHN2CFG	RW	Channel #2 configuration register
22	88	CHN3CFG	RW	Channel #3 configuration register
23	8C	CHN4CFG	RW	Channel #4 configuration register
24	90	CHN5CFG	RW	Channel #5 configuration register
25	94	CHN6CFG	RW	Channel #6 configuration register
26	98	CHN7CFG	RW	Channel #7 configuration register
27	9C	CHN8CFG	RW	Channel #8 configuration register

FC_ADDR	VXI_ADDR	Register Name	Access	Function
28	A0	CHN9CFG	RW	Channel #9 configuration register
29	A4	CHN10CFG	RW	Channel #10 configuration register
2A	A8	CHN11CFG	RW	Channel #11 configuration register
2B	AC	CHN12CFG	RW	Channel #12 configuration register
2C	B0	CHN13CFG	RW	Channel #13 configuration register
2D	B4	CHN14CFG	RW	Channel #14 configuration register
2E	B8	CHN15CFG	RW	Channel #15 configuration register
2F	C0	CHN16CFG	RW	Channel #16 configuration register
30-FB		reserved		
FC	3F4	FCSUB	RO	Function Card sub-type register
FE	3F8	FCSERH	RO	FC Serial Number High register
FF	3FC	FCSERL	RO	FC Serial Number Low register
8000	20000	FIFO	RO	Access to FIFO memory

### B.1 FCID (0x0) – Function Card ID Register

The FCID register contains function card identification number. Readout should always give a value of 3416H.

Bit	Access & Default	Description
15:0	RO 0x3416	<b>FCID – Function Card ID</b> FC identification number, 0x3416 for 16-channel, 24-bit Delta Sigma ADC

### B.2 FCVER (0x1) – Function Card Version Register

This is the FC version register. Readout from this register gives information about the PCB revision and the FPGA design revision.

Bit	Access & Default	Description
15:8	RO h	<b>FPGA_REV – FPGA Revision Number</b>
7:0	RO h	<b>PCB_REV – PCB Revision Number</b>

### B.3 FCCSR (0x2) – Function Card Control and Status Register

This is control and status register of the function card.

Bit	Access & Default	Description
15	R/W 0	<b>MASTER – Master</b> When the card is a Master, it generates all control signals, needed for the DA, internally. If the boards work in standalone configuration then all boards have to be set to Master. If the boards are configured for the synchronous sampling then only one board can be switched to be a Master. 1 : the board is a Slave 1 : the board is a Master
14	RO h	<b>I2C_DET – I2C device detected</b> The bit is used to detect a cable supporting communication with I2C devices is connected to the card. The bit does not change its status during normal operation. 0 : the cable supporting I2C is not present 1 : the cable supporting I2C communication is connected
13	RO h	<b>DA_END – Data Acquisition End</b> The bit is set by hardware after the normal end of data acquisition or when the DA_SKIP has been performed. It is not set if DA ends with the error. This bit is cleared on arming command. 1 : DA ended
12:10	RO h	<b>MAINSM_ST – Main State Machine States</b> The bits indicate the states of the main state machine. Read: 000 : IDLE_ST (idle state) 001 : DDSUD_ST (frequency synchronization) 010 : ADCSYNC_ST (ADC synchronization) 011 : READY4DA_ST (armed) 100 : PRET_ST (armed, ready for trigger) 101 : POSTT_ST (acquisition in progress)

		Others : reserved
9	RO 0	<b>FIFO PAFF – FIFO Programmable Almost Full Flag</b> The Almost Full Flag indicates that FIFO memory is almost full. Read: 0 : FIFO not almost full 1 : FIFO almost full Note: the same flag is present in the FIFO_CTRL register
8	RO 0	<b>ERR – a Common Error Flag</b> The bit informs that an error condition happened. The error source can be found by reading the ERROR register. Read: 0 : normal operation 1 : error condition happened
7:6	RO h	<b>MODE PINS – Mode Pins Status</b> The bits are read only and return the status of the Mode pins.
5	WSC 0	<b>DA SKIP – DA Skip</b> Writing this bit causes the end of DA. Write 1 : current DA skipped
4	R/W 0	<b>SYNC SEL – SYNC Source Selection</b> This bit selects the source of the SYNC signal. If the board is a Master then it generates the SYNC signal internally, otherwise it takes the SYNC from the external source selected by the SYNC_SEL bit. Write 0 : MB trigger input B (MB_TRIG1_B) 1 : FP trigger input 2 (FP_TRG2)
3	R/W 0	<b>SYNC NEED – Synchronisation Need</b> Defines if the DA has to be launched together with the synchronisation Write 0 : DA launched without a synchronisation 1 : DA launched with a synchronisation
2	R/W 0	<b>LONG SYNC – Long Synchronisation</b> If set when the board is a Master then a long P2 part of the SYNC pulses will be generated. Write 0 : synchronization takes about 950ms 1 : synchronization takes about 8200ms
1	WSC '0'	<b>ARM CMD – Arming Command</b> Arming command launches the data acquisition process (the main SM leaves the IDLE_ST) Write 0 : No effect 1 : Generates arming command
0	R/WSC 0	<b>SW_RST – Software Reset</b> This bit is used to reset a part of the FPGA logic which is related to the Data Acquisition. The reset doesn't change the contents of the registers. Reset is started by writing "1" to that bit. After the reset is done, the hardware clears the bit. Software should poll the bit until it is cleared. Write 0 : No effect 1 : Starts reset of the FPGA logic Read 0 : Card ready (reset finished, if previously started) 1 : Card not ready (reset in progress)

## B.4 MODE1 (0x3) – Mode 1 Register

This register is used to configure parameters of the function card.

Bit	Access & Default	Description
15		<b>Reserved</b>
14	R/W 0	<b>Reserved (SHORT SYNC)</b> This bit should be always set to '0'.
13:8	R/W 0	<b>DA TRG SEL [5:0] – Source for Data Acquisition Trigger</b> These bits enable trigger sources for starting and stopping the data acquisition process. Multiple sources can be selected at a time. Write '1' enables a source DA_TRG[5] – enables MB_TRIGI_A trigger input DA_TRG[4] – enables MB_TRIGI_B trigger input DA_TRG[3] – reserved DA_TRG[2] – reserved DA_TRG[1] – enables FP_TRG3 trigger input DA_TRG[0] – forces DA trigger to active state (software trigger)
7	RO 0	<b>DA TRG STS – DA Trigger Status</b> Current state of the DA Trigger. The status shows OR function of all enabled sources of the DA Trigger. 0 : DA Trigger inactive 1 : DA Trigger active
6	R/W 0	<b>DA TRG LEVEL – DA Trigger Edge/Level Selection</b> This bit selects between edge or level of the DA trigger 0 : DA trigger reacts on the edge 1 : DA trigger follows level
5	R/W 0	<b>DA TRG ESEL – Stop Trigger Edge Selection</b> This bit selects the edge transition for the stop trigger event 0 : DA trigger reacts on the active to inactive transition 1 : DA trigger reacts on the inactive to active transition
4:3	R/W 0	<b>STOP ON ERR – Stop On Error Mode</b> When set the bit enables stopping the data acquisition if an error happens. 00 : errors don't stop DA 01 : any error excluding OVRANGE_ERR stops DA 10 : any error including OVRANGE_ERR stops DA 11 : reserved
2	R/W 0	<b>DA STARTSEL – DA Start Mode Selection</b> This bit selects the way the Data Acquisition is started. 0 : DA starts immediately after synchronisation is done 1 : DA starts when DA Trigger goes active
1:0	R/W 0	<b>DA STOPSEL – DA Stop Mode Selection</b> This bit selects the way the Data Acquisition is stopped. 00 : DA stops when a programmed number of samples has been collected 01 : DA stops when DA Trigger stop event happens (Start Stop mode) 10 : DA stops when DA Trigger goes inactive (Gate mode) 11 : reserved

## B.5 MODE2 (0x4) – Mode 2 Register

This register is used to configure parameters of the function card.

Bit	Access & Default	Description
15	R/W '0'	<b><u>ADC_PWRDOWN – ADS1278 ADC power down</u></b> Write: 0 : ADS1278 powered up (default) 1 : ADS1278 in power down state Read Returns previously written value
14	R/W '1'	<b><u>ADC_CLK_DIV – ADS1278 ADC clock divider</u></b> Write: 0 : ADS1278 clock divider line set to 0 1 : ADS1278 clock divider line set to 1 (default) Read Returns previously written value
13:12	R/W '01'	<b><u>ADC_MODE – ADS1278 ADC mode selection</u></b> Write: 00 : High speed mode 01 : High resolution mode (default) 10 : Low Power mode 11 : Low Speed mode Read Returns previously written value
11:7		<b><u>Reserved</u></b>
6:5	R/W '00'	<b><u>DECIM_SEL – Decimation Stage Selection</u></b> These bits select the decimation factor of the chain of FIR filters. 00 : decimation switched off 01 : decimation by 10 selected 10 : decimation by 100 selected 11 : decimation by 1000 selected
4:3	R/W '00'	<b><u>ADC_CLK_SEL – Clock Selection</u></b> These bits select the source of the clock signal for the A/D converters. Write 00 : clock from on-board DDS selected 01 : common clock input from the MB (CCLK) 10 : MB input trigger 'A' (MB_TRIGI_A) 11 : FP input Trigger FP_TRG1 Read Returns previously written value
2	R/W 0	<b><u>PLL_EN – PLL Enable</u></b> This bit enables/disables on-board PLL. PLL must be enabled if onboard DDS is engaged. 0 : PLL disabled 1 : PLL enabled
1:0	R/W '00'	<b><u>PLL_RSEL – PLL Reference Clock Selection</u></b> These bits select the source of the PLL reference clock. Write 00 : local oscillator (40MHz divided down to 2MHz) 01 : common clock input from the MB (CCLK) 10 : MB input trigger 'A' (MB_TRIGI_A) 11 : FP input Trigger FP_TRG1

## B.6 OTRI (0x5) – Output Trigger Configuration Register

This register allows selection of a source for a particular trigger output.

Bit	Access & Default	Description
15:13	R/W '000'	<p><b><u>TRIGO A – Output trigger 'A' to the MB</u></b>            Select a source for the trigger            Write              000 : Output disabled (inactive)              001 : Front Panel Trigger input '3'              010 : Reference clock (2MHz)              011 : ADC clock              100 : Acquisition in progress              101 : OUTRANGE_ERR asserts trigger line              110 : FIFO trigger flag (programmable almost full PAF)              111 : Software assertion (forced to active state)            Read            Returns previously written value</p>
12:10	R/W '000'	<p><b><u>TRIGO B – Output trigger 'B' to the MB</u></b>            Select a source for the trigger            Write              000 : Output disabled (inactive)              001 : Front Panel Trigger input '2'              010 : SYNC pulses              011 : reserved              100 : Pulse per Scan              101 : OUTRANGE_ERR asserts the trigger line              110 : FIFO trigger flag (programmable almost full PAF) 111 : Software assertion (forced to active state)            Read            Returns previously written value</p>
9:7	R/W '000'	<p><b><u>FPTRIGO 1 – Front Panel Output trigger '1'</u></b>            Select a source for the trigger            Write              000 : Output disabled (high impedance)              001 : MB Trigger input 'A'              010 : Common Clock input from the MB (CCLK)              011 : Reference clock (2MHz)              100 : ADC clock              101 : reserved              110 : Acquisition in progress              111 : Software assertion (forced to active state)            Read            Returns previously written value</p>

6:4	R/W '000'	<p><b><u>FPTRIGO 2 – Front Panel Output trigger '2'</u></b></p> <p>Select a source for the trigger</p> <p>Write</p> <ul style="list-style-type: none"> <li>000 : Output disabled (high impedance)</li> <li>001 : MB Trigger input 'B'</li> <li>010 : SYNC pulses</li> <li>011 : reserved</li> <li>100 : reserved</li> <li>101 : OUTRANGE_ERR asserts trigger line</li> <li>110 : Pulse per Scan</li> <li>111 : Software assertion (forced to active state)</li> </ul> <p>Read</p> <p>Returns previously written value</p> <p>Note:</p> <p>If FP trigger output 2 is selected as the SYNC source then FPTRG_2_LOW should be set for compatibility with other FCs.</p>
3:2	R/W '00'	<p><b><u>FPTRIGO 3 – Front Panel Output trigger '3'</u></b></p> <p>Select a source for the trigger</p> <p>Write</p> <ul style="list-style-type: none"> <li>00 : Output disabled (high impedance)</li> <li>01 : MB Trigger input 'A'</li> <li>10 : MB Trigger input 'B'</li> <li>11 : Software assertion (forced to active state)</li> </ul> <p>Read</p> <p>Returns previously written value</p>
1:0	R/W '00'	<p><b><u>DI – Direct Interrupt</u></b></p> <p>Select a source for the interrupt line</p> <p>Write</p> <ul style="list-style-type: none"> <li>00 : Output disabled (inactive)</li> <li>01 : FIFO trigger flag (programmable almost full PAF)</li> <li>10 : OUTRANGE_ERR asserts DI line</li> <li>11 : Software assertion (forced to active state)</li> </ul> <p>Read</p> <p>Returns previously written value</p>

## B.7 ITRI (0x6) – Input Trigger Status Register

This is the FC version register. Readout from this register gives information about the PCB revision and the FPGA design revision.

Bit	Access & Default	Description
15:12		<b>Reserved</b>
11:9	RO H	<p><b><u>CCLK – Common clock input status</u></b></p> <p>This is a status of common clock input line.</p> <p>Read:</p> <ul style="list-style-type: none"> <li>000 : Input at a low level (for at least last 10us)</li> <li>001 : Input toggles at 2MHz</li> <li>010 : Input toggles at 5MHz</li> <li>011 : Input toggles at 10MHz</li> <li>100 : Input toggles at another frequency</li> <li>101 : reserved</li> <li>110 : reserved</li> <li>111 : Input at a high level (for at least last 10us)</li> </ul>

8	RO H	<p><b><u>I2C INT – status of the interrupt line</u></b>  This bit reflects the actual status of the I2C interrupt line from the PCA9665 controller.  Read:  0 : interrupt line not active  1 : interrupt line active (asserted by the PCA9665 controller)</p>
7	R/W 0	<p><b><u>FPTRG 3 LOW – FP trigger 3 active low</u></b>  The bit negates the front panel trigger (both directions) so the active level is LOW  Write:  0 : trigger high active  1 : trigger low active</p>
6	R/W 0	<p><b><u>FPTRG 2 LOW – FP trigger 2 active low</u></b>  The bit negates the front panel trigger (both directions) so the active level is LOW  Write:  0 : trigger high active  1 : trigger low active</p>
5	R/W 0	<p><b><u>FPTRG 1 LOW – FP trigger 1 active low</u></b>  The bit negates the front panel trigger (both directions) so the active level is LOW  Write:  0 : trigger high active  1 : trigger low active</p>
4	RO H	<p><b><u>FPTRIGI 3 – FP Trigger Input ‘3’</u></b>  This is a status bit that indicates when the current state of the trigger line.  Read:  0 : trigger not active  1 : trigger active (asserted)</p>
3	RO H	<p><b><u>FPTRIGI 2 – FP Trigger Input ‘2’</u></b>  This is a status bit that indicates when the current state of the trigger line.  Read:  0 : trigger not active  1 : trigger active (asserted)</p>
2	RO H	<p><b><u>FPTRIGI 1 – FP Trigger Input ‘1’</u></b>  This is a status bit that indicates when the current state of the trigger line.  Read:  0 : trigger not active  1 : trigger active (asserted)</p>
1	RO H	<p><b><u>TRIGI B – MB Trigger Input ‘B’</u></b>  This is a status bit that indicates when the current state of the trigger line.  Read:  0 : trigger not active  1 : trigger active (asserted)</p>
0	RO H	<p><b><u>TRIGI A – MB Trigger Input ‘A’</u></b>  This is a status bit that indicates when the current state of the trigger line.  Read:  0 : trigger not active  1 : trigger active (asserted)</p>

## B.8 DDS\_WX (0x8) – DDS Control Register

This is DDS control register. It is used to set up words written to DDS during DDS update phase if DDS is to be used.

Bit	Access & Default	Description
15:12		Reserved
11:8	WO	<b>WX_ADDR – Word X Address</b> The address of the word. WX_ADDR is in the range from 0 to 12.
7:0	WO	<b>DDS_WX – DDS Word X</b> The content of the DDS word to be stored

## B.9 OCOEFL (0x9) – Offset Coefficient Write Low Register

OCOEFL register together with OCOEFH register can be used to overwrite offset coefficients loaded during card initialisation from EEPROM.

Bit	Access & Default	Description
15:0	WO	<b>GCOEFL – Offset coefficient, bits 15..0</b> Write Stores bits 15..0 of the offset coefficient in an internal temporary register

## B.10 OCOEFH (0xA) – Offset Coefficient Write High Register

OCOEFH register together with OCOEFL register can be used to overwrite offset coefficients loaded during card initialisation from EEPROM.

Bit	Access & Default	Description
15:12	WO	<b>OCOEF_CHN – Gain coefficient write channel selection</b> Write Specifies the channel the offset coefficient is being written for: 0000 : Channel 1 0001 : Channel 2 0010 : Channel 3 0011 : Channel 4 0100 : Channel 5 0101 : Channel 6 0110 : Channel 7 0111 : Channel 8 1000 : Channel 9 1001 : Channel 10 1010 : Channel 11 1011 : Channel 12 1100 : Channel 13 1101 : Channel 14 1110 : Channel 15 1111 : Channel 16
11:8		<b>Reserved</b>

7:0	WO	<b><u>OCOEFH – Offset coefficient, bits 23..16</u></b> Write Stores the offset coefficient into the internal gain coefficient memory at a location specified by OCOEF_CHN field
-----	----	---

### B.11 GCOEFL (0xB) – Gain Coefficient Write Low Register

GCOEFL register together with GCOEFH register can be used to overwrite gain coefficients loaded during card initialisation from EEPROM.

Bit	Access & Default	Description
15:0	WO	<b><u>GCOEFL – Gain coefficient, bits 15..0</u></b> Write Stores bits 15..0 of the gain coefficient in an internal temporary register

### B.12 GCOEFH (0xC) – Gain Coefficient Write High Register

GCOEFH register together with GCOEFL register can be used to overwrite gain coefficients loaded during card initialisation from EEPROM.

Bit	Access & Default	Description
15:12	WO	<b><u>GCOEF_CHN – Gain coefficient write channel selection</u></b> Write Specifies the channel the gain coefficient is being written for: 0000 : Channel 1 0001 : Channel 2 0010 : Channel 3 0011 : Channel 4 0100 : Channel 5 0101 : Channel 6 0110 : Channel 7 0111 : Channel 8 1000 : Channel 9 1001 : Channel 10 1010 : Channel 11 1011 : Channel 12 1100 : Channel 13 1101 : Channel 14 1110 : Channel 15 1111 : Channel 16
11:8		<b><u>Reserved</u></b>
7:0	WO	<b><u>GCOEFH – Gain coefficient, bits 23..16</u></b> Write Stores gain coefficient into internal gain coefficient memory at location specified by GCOEF_CHN field

### B.13 I2C\_CTRL (0xE) – I2C Control Register

This is the I2C control register, used to communicate with the I2C controller.

Bit	Access & Default	Description
15		<b>Reserved</b>
14	WO	<b><u>I2C_RD_nWR – I2C Read / Write</u></b> This bit is used to select desired operation for the I2C controller. Write 0 : WRITE to the I2C controller 1 : READ from the I2C controller
13	RO h	<b><u>I2C_INT – status of the interrupt line</u></b> This bit reflects the actual status of the I2C interrupt line from the controller. Read: 0 : interrupt line not active 1 : interrupt line active (asserted by the controller)
12:2		<b>Reserved</b>
9:8	R/W '00'	<b><u>I2C_ADDR – I2C Controller Address</u></b> Write Selects the I2C controller register for which command is issued (range: 0..3) Read Returns previously written value
7:0	R/W h	<b><u>I2C_DATA</u></b> This the data byte that will be transferred to the I2C controller during WRITE operation or data read from the I2C controller after previous READ operation. Write Stores written data to be transmitted to the I2C controller after WRITE command is issued using the I2C_CTRL register Read Returns the data read from the I2C controller after last READ command issued through I2C_CTRL register

### B.14 TEDS\_ACC (0xF) – TEDS Access Register

This register gives the possibility to access the TEDS-enabled sensor's memory.

Bit	Access & Default	Description
15:12		<b>Reserved</b>
11	RO 1	<b><u>TEDS_READY – TEDS Ready</u></b> This bit indicates if the execution of an operation specified on the OP bits is finished. Read 0 : TEDS memory not ready, operation in progress 1 : TEDS memory ready and able to react on further actions
10	RO 0	<b><u>TEDS_PRESENT – TEDS Present</u></b> This bit indicates if the execution of a RESET operation finished successfully. Read 0 : TEDS not present 1 : TEDS present and able to react on further actions
9:8	WO	<b><u>OP – Operation Selection</u></b> The Operation bits specify the action the TEDS interface logic shall do. Write 00 : Has no effect 01 : READ - Requests a read of one data byte from TEDS sensor memory. After

		<p>operation is completed (bit TEDS_READY set to '1'), software can read this data from the CMD_DATA bits.</p> <p>10 : WRITE - Requests a write of byte specified in the CMD_DATA field to the TEDS sensor memory. Transmission of the next byte can be initiated after operation is completed (bit TEDS_READY set to '1').</p> <p>11 : RESET - Initiates TEDS sensor memory Reset operation, which is necessary before first access to TEDS memory and after some commands issued to it. Next operation can be initiated after operation is completed successfully (bit TEDS_READY set to '1' and bit TEDS_PRESENT set to '1').</p>
7:0	R/W h	<p><b>CMD_DATA – Command / Data</b></p> <p>This is the command/address/data byte that will be transferred to the TEDS sensor memory during WRITE operation or data read from memory after READ operation.</p> <p>Write Specifies the byte that has to be transferred to the TEDS memory during a Write operation</p> <p>Read Gives the last data read from the TEDS sensor memory</p>

### B.15 FIFO\_CTRL (0x10) – FIFO Control Register

This register is a control/status register of the FIFO memory.

Bit	Access & Default	Description
15	R/WSC 0	<p><b>FIFO_RST – FIFO Reset</b></p> <ul style="list-style-type: none"> <li>This is FIFO reset. Reset is done by writing "1" to that bit and waiting for "0". Reset of the FIFO means clearing read, write pointers and internal registers.</li> </ul> <p>Write 0 : no effect 1 : starts reset of FIFO</p> <p>Read 0 : resetting finished (if previously started) 1 : resetting in progress</p>
14	RO 0	<p><b>FIFO_UNF_ERR – FIFO Underflow Error Flag</b></p> <p>This signal indicates that a read request was rejected because the FIFO is empty. Underflowing to the FIFO is non-destructive to the FIFO.</p> <p>0 : FIFO not underflow 1 : FIFO underflow</p>
13	RO 0	<p><b>FIFO_OVF_ERR – FIFO Overflow Error Flag</b></p> <p>This signal indicates that a write request was rejected because the FIFO is full. Overflowing to the FIFO is non-destructive to the FIFO.</p> <p>0 : FIFO not overflow 1 : FIFO overflow</p>
12	RO 0	<p><b>FIFO_FF – FIFO Full Flag</b></p> <p>The Full Flag indicates that FIFO memory is full.</p> <p>0 : FIFO not full 1 : FIFO full</p>
11	RO 0	<p><b>FIFO_PAFF – FIFO Programmable Almost Full Flag</b></p> <p>The Almost Full Flag indicates that FIFO memory is almost full.</p> <p>0 : FIFO not almost full 1 : FIFO almost full</p>

10	RO 1	<b>FIFO_EF – FIFO Empty Flag</b> The Empty Flag indicates that FIFO memory is empty. 0 : FIFO not empty 1 : FIFO empty
9:8		<b>Reserved</b>
7:0	RO 0	<b>FIFO_COUNT[7:0] – FIFO Count</b> This is a lower part of a counter for a number of samples stored in the FIFO memory. The most significant bits are not available.

### B.16 FIFO\_AFT (0x11) – FIFO Almost Full Flag Threshold Register

This register is used to write the configuration data for programmable Almost Full FIFO flag. FIFO Reset has to be asserted for reconfiguration.

Bit	Access & Default	Description
15:14		<b>Reserved</b>
13:0	R/W 0	<b>FIFO_AFL – FIFO Almost Full Assert Level</b> Value is used to set the threshold level for programmable almost full flag, which define when the signal is asserted.

### B.17 FIFO\_WR (0x12) – FIFO Write Register

This register is used to write the data to FIFO. Writing to FIFO is allowed in IDLE\_ST only. Two writes to FIFO\_WR register (a first is 16 bit a second uses only lower 8 bits of data) cause one 24bit word stored in the onboard FIFO memory, which can be later read out using FIFO register.

Bit	Access & Default	Description
15:0	WO	<b>FIFO_DATA – FIFO Data Input</b> Writes a word to a FIFO memory for test purposes.

### B.18 SIG\_ERR (0x13) – Signal Error Register

This is signal out of range error register.

Bit	Access & Default	Description
15:0	RO 0x0	<b>SIG_ERR – Signal Out of Range Error</b> Indicates that signal out of range condition detected on (bit number + 1) channel 1 : signal out of range 0 : signal in range

## B.19 GAIN\_COMP (0x14) – Gain Compensation Register

This is the gain compensation register.

Bit	Access & Default	Description
15:0	R/W 0x0	<p><b>GAIN_COMP[15..0] – Gain compensation coefficient (lower part)</b></p> <p>Write Allows overwriting the GAIN_COMP_COEFF loaded automatically from EEPROM. Value 0x0 written to this register disables the frequency dependant gain compensation mechanism.</p> <p>Read Returns lower 16 bits of a gain compensation coefficient, upper 8 bits have a value of 0x7F (or 0x80 if read value is 0x0000). The coefficient is sampling clock dependant and is scaled automatically with the actual ADC clock frequency according to programmed in EEPROM value.</p>

## B.20 ERROR (0x15) – Error Register

Bit	Access & Default	Description
15	WSC	<p><b>CLR_CMD – Clear Command</b></p> <p>Clear command clears errors</p> <p>Write 0 : No effect 1 : clear errors command</p>
14:13	R/W 0	<p><b>DE – Direct Error</b></p> <p>Select sources for the error line DE</p> <p>Write 00 : DE output disabled 01 : OUTRANGE_ERR asserts DE line 10 : any error asserts DE line 11 : Software assertion (forced to active state)</p> <p>Read Returns previously written value</p>
12:8		<b>Reserved</b>
7	RO 0	<p><b>FIFO_UNF_ERR – FIFO Underflow Error</b></p> <p>This bit indicates that there was an attempt to read from an empty FIFO. 1 : FIFO underflow error occurred</p>
6	RO 0	<p><b>FIFO_OVF_ERR – FIFO Overflow Error</b></p> <p>The bit when reading returns the status of the FIFO overload event (occurs when trying to write to a full FIFO). This bit is cleared on the clearing command.</p> <p>Read 1 : FIFO overload error occurred</p>
5	RO h	<p><b>SCAN_ERR – Scan Error</b></p> <p>The bit is read only and is set by hardware after scan error happens. This bit is cleared on the clearing command. 1 : Scan error occurred</p>

4	RO h	<b>ADCRANGE_ERR – ADC Clock Frequency Out of Range Error</b> The bit is read only and is set by hardware after the frequency of the ADC signal goes out of range (lower than 512kHz or higher than 5.12MHz) . This bit will be set after a first arming with the synchronization. This bit is cleared on the clearing command and after arming with synchronization. 1 : ADC frequency out of range error occurred
3	RO h	<b>DDSUD_ERR – DDS External Update Signal Error</b> The bit is read only and is set by hardware after DDS external update signal error happens. This bit is cleared on the clearing command. 1 : external update signal not in phase
2	RO h	<b>ARITH_ERR – Arithmetic Overflow Error</b> Indicates that during offset and gain correction calculations, overflow condition happened on any channel. This bit is cleared on the clearing command. 1 : out of range error occurred
1	RO h	<b>DEC_ERR – Decimation error detected</b> Bit indicates that during writing data to decimation stage collision happen which is related to decimation module fatal error 1 : decimation error 0 : normal operation
0	RO h	<b>OUTRANGE_ERR – Input Signal Out of Range Error</b> The bit is read only and is set by hardware after input signal out of range error happens. This bit is cleared on the clearing command. 1 : out of range error occurred This is common signal for all channels. Read SIG_ERR register to detect which channel caused this error.

### B.21 POSTT\_NOSL (0x19) – Post Trigger Number of Scans Low Register

The POSTT\_NOSL/H registers define the number of scans to be acquired during DA if the mode with number of scans has been selected (DA\_STOPSEL='00'). The number of post-trigger scans is in the range from 0 to 16777215 (0 means unlimited number of scans).

Bit	Access & Default	Description
15:0	R/W 0x0	<b>POSTT_NOS – Post Trigger Number Of Scans</b> Lower 16 bits (POSTT_NOS[15:0]) of the post trigger number of scans to collect.

### B.22 POSTT\_NOSH (0x1A) – Post Trigger Number of Scans High Register

The POSTT\_NOSL/H registers define the number of scans to be acquired during DA if the mode with number of scans has been selected (DA\_STOPSEL='00'). The number of post-trigger scans is in the range from 0 to 16777215 (0 means unlimited number of scans).

Bit	Access & Default	Description
15:8		<b>Reserved</b>
7:0	R/W 0x0	<b>POSTT_NOS – Post Trigger Number Of Scans</b> Upper 8 bits (POSTT_NOS [23:16]) of the post trigger number of scans to collect.

### B.23 CHNxCFG (0x20...0x2F) – Channel x Configuration Register

These are the registers used to configure the settings for all channels.

Bit	Access & Default	Description
15:6		<b>Reserved</b>
5:4	R/W '00'	<b><u>GAIN2_SEL – Gain of the Second Stage Selection</u></b> These bits select the second PGA gain of the analog front end. 00 : x1 gain 01 : x2 gain 10 : x5 gain 11 : x10 gain
3:2	R/W '00'	<b><u>GAIN1_SEL – Gain of the First Stage Selection</u></b> These bits select the first PGA gain of the analog front end. 00 : x1 gain 01 : x10 gain 10 : x100 gain 11 : x1000 gain
1	R/W 0	<b><u>VREF_EN – VREF Input Enabled</u></b> This bit connects the channel to the voltage reference option fitted on the MB. 0 : channel input switched to the FP input connector 1 : channel input switched to VREF voltage
0	R/W 0	<b><u>CHN_EN – Channel Enable</u></b> This bit is used to include the channel in the DA. 0 : channel disabled 1 : channel enabled

### B.24 FCSSUB (0xFC) – Function Card Sub-Type Register

This is function card sub-type register useful for software to distinguish between versions of the board.

Bit	Access & Default	Description
15:8	RO h	<b><u>FCSUB_2CH – Sub-Type Second Character</u></b> Second ASCII character of the function card sub-type
7:0	RO h	<b><u>FCSUB_1CH – Sub-Type First Character</u></b> First ASCII character of the function card sub-type

### B.25 FCSERH (0xFE) – Function Card Serial Number High Register

This register contains the upper 16 bits of the FC serial number. Serial numbers are coded as BCD digits.

Bit	Access & Default	Description
15:0	RO h	<b><u>FCSERH – Function Card Serial Number (upper part)</u></b> Upper 16 bits (FCSER[31:16]) of the function card serial number

## B.26 FC SERL (0xFF) – Function Card Serial Number Low Register

This register contains the lower 16 bits of the FC serial number.

Bit	Access & Default	Description
15:0	RO h	<b>FC SERL – Function Card Serial Number (lower part)</b> Lower 16 bits (FC SERL[15:0]) of the function card serial number

## B.27 FIFO (0x8000) – FIFO memory

Allows the readout of the FIFO memory.

Bit	Access & Default	Description
15:0	RO 0x0	<b>FIFO – FIFO memory readout</b> The measured 24bit data is stored in the FIFO memory and can be read using this register. A first access is for lower 16 bits of the 24bit ADC sample, a second access uses only bits FIFO[7:0] for higher 8 bits (23 to 16) of the 24bit ADC sample (FIFO[15:8] bits contain repeated sign bit so a result is extended to signed 32bit value).

## Appendix C: VXIplug&play Driver Functions

### C.1 Introduction

This instrument driver provides programming support for the ProDAQ 3416 16 channel, 24-bit Sigma-Delta ADC Function card. It contains functions for opening, configuring, acquiring data with, and closing the instrument.

### C.2 Assumptions

To successfully use this function card, it must be installed onto a ProDAQ VXIbus motherboard or a ProDAQ LXI function card carrier. The ProDAQ motherboard must in turn be installed in a VXIbus system which is connected via a suitable slot-0 controller to your computer. The LXI function card carrier must be connected via network to your computer. A suitable VISA library must be installed on your computer.

### C.3 Error and Status Information:

Each function in this instrument driver returns a status code that either indicates success or describes an error or warning condition. Your program should examine the status code from each call to an instrument driver function to determine if an error occurred.

The general meaning of the status code is as follows:

Value	Meaning
0	Success
Positive Values	Warnings
Negative Values	Errors

The description of each instrument driver function lists possible error codes and their meanings.

## C.4 Function Tree Layout:

ProDAQ 3416 16-ch 24-bit Sigma Delta ADC	Function Name:
Initialization	bu3416_init
Select Function Card	bu3416_fcSelect
Initialization With Parameters	bu3416_paramInit
Hardware Configuration	
Set Channel Configuration	bu3416_setChanConfig
Set Trigger Configuration	bu3416_setTrigConfig
Calibrate Board	bu3416_calibrateBoard
Single-Card Acquisition	
Single-shot Acquisition	
Acquire Waveform	bu3416_acquireWaveform
Acquire Waveforms	bu3416_acquireWaveforms
Continuous Acquisition	
Set Acquisition Mode	bu3416_setAcquisitionMode
Start Acquisition	bu3416_startAcquisition
Start Acquisition Ex	bu3416_startAcquisitionEx
Check Acquisition	bu3416_checkAcquisition
Read Acquisition	bu3416_readAcquisition
Stop Acquisition	bu3416_stopAcquisition
Multi-Card Acquisition	
Multi-Card Initialization	bu3416_multInit
Multi-Card Configuration	bu3416_multConfig
Multi-Card Channel Config	bu3416_setMultChanConfig
Multi-Card Trigger Config	bu3416_setMultTrigConfig
Multi-Card Start Acquisition	bu3416_startMultAcquisition
Multi-Card Start Acquisition Ex	bu3416_startMultAcquisitionEx
Multi-Card Check Acquisition	bu3416_checkMultAcquisition
Multi-Card Read Acquisition	bu3416_readMultAcquisition
Multi-Card Stop Acquisition	bu3416_stopMultAcquisition
Multi-Card Set DRAM Buffer Size	bu3416_resizeMultBuf
Multi-Card Get FC Handle	bu3416_getMultFCsession
Multi-Card Close	bu3416_multClose
Low-Level Access	
Set Input Trigger Config	bu3416_setITRIConfig
Get Input Trigger Config	bu3416_getITRIConfig
Set Output Trigger Config	bu3416_setOTRIConfig
Get Output Trigger Config	bu3416_getOTRIConfig
Set FP Triggers Polarity	bu3416_setFPTrigPolarity
Get FP Triggers Polarity	bu3416_getFPTrigPolarity
Set DAQ Mode	bu3416_setDAQMode
Get DAQ Mode	bu3416_getDAQMode
Set ADC Mode	bu3416_setADCMode
Get ADC Mode	bu3416_getADCMode
Set Post-Trigger Scans	bu3416_setPostScans
Get Post-Trigger Scans	bu3416_getPostScans
Set DDS Frequency	bu3416_setDDSFreq
Get DDS Frequency	bu3416_getDDSFreq
Set Sampling Frequency	bu3416_setSampFreq
Get Sampling Frequency	bu3416_getSampFreq
Control/Status Functions	
Generate Input Trigger	bu3416_generateITRI
Generate Output Trigger	bu3416_generateOTRI
Get Input Triggers State	bu3416_getITRIState
Reset DAQ	bu3416_resetDAQ
Arm DAQ	bu3416_armDAQ
Clear Errors	bu3416_clearErrors
Stop DAQ	bu3416_stopDAQ
Get DAQ Status	bu3416_getDAQStatus
FIFO Readout / Control	
Set FIFO Configuration	bu3416_setFIFOConfig
Get FIFO Configuration	bu3416_getFIFOConfig
Get FIFO Status	bu3416_getFIFOStatus
Read FIFO	bu3416_readFIFO
Reset FIFO	bu3416_resetFIFO

Calibration Functions	
Calibrate All Channels	bu3416_calibrateAllChannels
Store Calibration Data	bu3416_storeCalibData
Get Calibration Data	bu3416_getCalibData
LIST Processor Support	
Set Buffer Size	bu3416_setBufferSize
Get Buffer Size	bu3416_getBufferSize
Enable LIST	bu3416_enableLIST
TEDS	
Read 1-Wire ROM	bu3416_readTEDS_ROM
Burn 64-bit OTP ROM	bu3416_burnTEDS_OTP_ROM
Read 64-bit OTP ROM	bu3416_readTEDS_OTP_ROM
Write 256-Bit EEPROM	bu3416_writeTEDS_EEPROM
Read 256-Bit EEPROM	bu3416_readTEDS_EEPROM
I2C	
Communicate with I2C device	bu3416_writeReadI2C
Reset I2C Controller	bu3416_resetI2C
JTAG	
Generate Sequence	bu3416_JTAG_generateSeq
Get Status	bu3416_JTAG_getStatus
Utility Functions	
Get Serial Number	bu3416_getSerNum
Reset	bu3416_reset
Self Test	bu3416_self_test
Error Query	bu3416_error_query
Error Message	bu3416_error_message
Revision Query	bu3416_revision_query
Close	bu3416_close

## C.5 VXIplug&play Driver Function Details

The following functions are in alphabetical order.

### C.5.1 bu3416\_acquireWaveform

```
ViStatus bu3416_acquireWaveform (ViSession instrumentHandle, ViInt16 channel,
                                 ViReal64 sampleRateHz, ViInt32 samples,
                                 ViReal64 waveform[], ViInt16 *errors);
```

#### Purpose

This function acquires the waveform from the specified channel. The channel should be configured using the bu3416\_setChanConfig() function prior to this function call.

For the triggered waveform acquisition the trigger must be configured using bu3416\_setTrigConfig() function prior to this function call.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### channel

Variable Type      ViInt16

This parameter specifies from which channel the waveform will be acquired.

Possible values are:

bu3416_CHAN_1	1	Channel 1
bu3416_CHAN_2	2	Channel 2
bu3416_CHAN_3	3	Channel 3
bu3416_CHAN_4	4	Channel 4
bu3416_CHAN_5	5	Channel 5
bu3416_CHAN_6	6	Channel 6
bu3416_CHAN_7	7	Channel 7
bu3416_CHAN_8	8	Channel 8
bu3416_CHAN_9	9	Channel 9
bu3416_CHAN_10	10	Channel 10
bu3416_CHAN_11	11	Channel 11
bu3416_CHAN_12	12	Channel 12
bu3416_CHAN_13	13	Channel 13
bu3416_CHAN_14	14	Channel 14
bu3416_CHAN_15	15	Channel 15
bu3416_CHAN_16	16	Channel 16

##### sampleRateHz

Variable Type      ViReal64

This parameter specifies the sample rate (in Hertz) for data acquisition process. Possible values are from 1.0 (1Hz) to 10000.0 (10KHz)

**samples**

Variable Type            ViInt32

This parameter sets the number of samples to collect.

**waveform**

Variable Type            ViReal64[]

The output buffer containing the samples from the specified channel. This buffer should be allocated by application before the function call with appropriate size to hold all data. Values of the waveform are expressed in Volts.

**errors**

Variable Type            ViInt16 (passed by reference)

This parameter contains information about any error happened during Data Acquisition process. The value is a bitmask of the following values:

bu3416_DA_OUTRANGE_ERR	0x0001	Cumulative Out Of Range error;
bu3416_DA_DECIM_ERR	0x0002	Decimation error;
bu3416_DA_ARITH_ERR	0x0004	Arithmetic error;
bu3416_DA_DDSUD_ERR	0x0008	DDS Update Signal error;
bu3416_DA_MCLK_ERR	0x0010	MCLK Clock Frequency error;
bu3416_DA_SCAN_ERR	0x0020	Scan Error;
bu3416_DA_FIFO_OV_ERR	0x0040	FIFO Overflow error;
bu3416_DA_FIFO_UF_ERR	0x0080	FIFO Underflow error;

**NOTE:**

For error bu3416\_DA\_OUTRANGE\_ERR please use function bu3416\_getDAQStatus() to get more details about error happened on channel.

**Return Value**

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.2 bu3416\_acquireWaveforms

```
ViStatus bu3416_acquireWaveforms (ViSession instrumentHandle,
                                   ViInt16 channelMask, ViReal64 scanRateHz,
                                   ViInt32 scans, ViInt16 fillMode,
                                   ViReal64 waveforms[], ViInt16 *errors);
```

### Purpose

This function acquires the waveforms from the specified channels. All channels should be configured using the bu3416\_setChanConfig() function prior to this function call.

For the triggered waveform acquisition the trigger must be configured using bu3416\_setTrigConfig() function prior to this function call.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### channelMask

Variable Type      ViInt16

Selects the channels from which the data will be taken.

bit 0 corresponds to the Channel 1

...

bit 15 corresponds to the Channel 16

"1" written in the appropriate bit means that the channel will be included in the data acquisition.

All selected channels must be configured using bu3416\_setChanConfig() function prior to this function call.

#### scanRateHz

Variable Type      ViReal64

This parameter specifies the sample rate (in Hertz) for data acquisition process. Possible values are from 1.0 (1Hz) to 10000.0 (10kHz).

#### scans

Variable Type      ViInt32

This parameter sets the number of scans to collect.

#### fillMode

Variable Type      ViInt16

The parameter specifies whether the Waveform array will be grouped by channels or grouped by scans.

Possible values are:

```
bu3416_GROUP_BY_CHANNEL  0  Group data by channel
bu3416_GROUP_BY_SCAN    1  Group data by scans
```

For example:

If you scan channels A through C and Number of Scans is 5, then the possible fill modes are:

Grouped by channel:

```
A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C4 C5
 \-----/ \-----/ \-----/
or
```

Grouped by scan:

```
A1 B1 C1 A2 B2 C2 A3 B3 C3 A4 B4 C4 A5 B5 C5
 \----/ \----/ \----/ \----/ \----/
```

If you are to pass the array to a graph, you should acquire the data grouped by channel.

If you are to pass the array to a strip chart, you should acquire the data grouped by scan.

#### waveforms

Variable Type ViReal64[]

The output buffer containing the samples (in Volts) from the specified channels. This buffer should be allocated by application before the function call with appropriate size to hold all data. Samples in this buffer are arranged according to "Fill Mode" parameter".

#### errors

Variable Type ViInt16 (passed by reference)

This parameter contains information about any error happened during Data Acquisition process. The value is a bitmask of the following values:

```
bu3416_DA_OUTRANGE_ERR 0x0001 Cumulative Out Of Range error;
bu3416_DA_DECIM_ERR    0x0002 Decimation error;
bu3416_DA_ARITH_ERR    0x0004 Arithmetic error;
bu3416_DA_DDSUD_ERR    0x0008 DDS Update Signal error;
bu3416_DA_MCLK_ERR     0x0010 MCLK Clock Frequency error;
bu3416_DA_SCAN_ERR     0x0020 Scan Error;
bu3416_DA_FIFO_OV_ERR  0x0040 FIFO Overflow error;
bu3416_DA_FIFO_UF_ERR  0x0080 FIFO Underflow error;
```

NOTE:

For error bu3416\_DA\_OUTRANGE\_ERR please use function bu3416\_getDAQStatus() to get more details about error happened on channel.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.3 bu3416\_armDAQ

```
ViStatus bu3416_armDAQ (ViSession instrumentHandle, ViInt16 syncNeed);
```

#### Purpose

This function issues the arming command, which launches the Data Acquisition process.

#### Parameter List

##### instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### syncNeed

Variable	Type	ViInt16
----------	------	---------

This parameter specifies whether DAQ Arming process will go through the Synchronization procedure or not.

Possible values are:

bu3416_OFF	0	Synchronization procedure will not be performed;
bu3416_ON	1	Synchronization procedure will be performed;
bu3416_SYNC_IF_NEED	2	Synchronization procedure will be performed only if necessary, i.e. if the function card was reconfigured before, so it must be re-synchronized (Default);

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.4 bu3416\_burnTEDS\_OTP\_ROM

```
ViStatus bu3416_burnTEDS_OTP_ROM (ViSession instrumentHandle, ViBuf buf,
                                   ViBoolean burn);
```

### Purpose

Burns TEDS 64-bit OTP ROM if not programmed yet.

**ATTENTION:** This memory can be programmed only once (OTP)  
Please make sure that data is correct and byte's  
order is right.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### buf

Variable Type      ViBuf

Buffer with bytes to write.

**NOTE:** size of this buffer must be 8 bytes (64 bits).  
Function will copy bytes in following order:

```
buf[0] into ROM address 0x00
buf[1] into ROM address 0x01
buf[2] into ROM address 0x02
buf[3] into ROM address 0x03
buf[4] into ROM address 0x04
buf[5] into ROM address 0x05
buf[6] into ROM address 0x06
buf[7] into ROM address 0x07
```

#### burn

Variable Type      ViBoolean

if "burn" is VI\_TRUE function will program ROM.  
if not function will write data into scratchpad. (for testing)  
**ATTENTION:** Be careful programming ROM can be done only once.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.5 bu3416\_calibrateAllChannels

```
ViStatus bu3416_calibrateAllChannels (ViSession instrumentHandle, ViInt16 gain,
                                     ViInt32 offsets[], ViInt32 gains[]);
```

### Purpose

This function performs the calibration of all channels for the selected gain. Please note that calibration process requires Voltage Reference module fitted on Motherboard.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### gain

Variable Type      ViInt16

This parameter specifies the gain for which all channels will be calibrated;

Possible values are:

bu3416_GAIN_1	1	Gain 1 (Default)
bu3416_GAIN_2	2	Gain 2
bu3416_GAIN_5	5	Gain 5
bu3416_GAIN_10	10	Gain 10
bu3416_GAIN_20	20	Gain 20
bu3416_GAIN_50	50	Gain 50
bu3416_GAIN_100	100	Gain 100
bu3416_GAIN_200	200	Gain 200
bu3416_GAIN_500	500	Gain 500
bu3416_GAIN_1000	1000	Gain 1000
bu3416_GAIN_2000	2000	Gain 2000

#### offsets

Variable Type      ViInt32[]

This parameter returns the calibration coefficients (Offsets) acquired during calibration process. It is array of 16 values for all 16 channels. The program should allocate this buffer with appropriate size prior to the function call.

#### gains

Variable Type      ViInt32[]

This parameter returns the calibration coefficients (Gains). It is array of 16 values for all 16 channels. The program should allocate this buffer with appropriate size prior to the function call.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.6 bu3416\_calibrateBoard

```
ViStatus bu3416_calibrateBoard (ViSession instrumentHandle, ViInt16 gain);
```

### Purpose

This function performs the calibration of all 16 channels of the ProDAQ 3416 module.

The calibration will be performed only for the selected gain setting.

If any other gain will be selected for the data acquisition process, the board should be calibrated for this gain setting as well.

The calibration coefficients will be applied to the hardware of the 3416 module immediately, but they will not be stored in any non-volatile memory on the board. So, after power-down or function card reset the board should be calibrated again.

Please note that calibration process require Voltage Reference module fitted on the Motherboard.

### Parameter List

#### instrumentHandle

Variable Type	ViSession
---------------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### gain

Variable Type	ViInt16
---------------	---------

This parameter specifies the gain for which all channels will be calibrated;

Possible values are:

bu3416_GAIN_1	1	Gain 1 (Default)
bu3416_GAIN_2	2	Gain 2
bu3416_GAIN_5	5	Gain 5
bu3416_GAIN_10	10	Gain 10
bu3416_GAIN_20	20	Gain 20
bu3416_GAIN_50	50	Gain 50
bu3416_GAIN_100	100	Gain 100
bu3416_GAIN_200	200	Gain 200
bu3416_GAIN_500	500	Gain 500
bu3416_GAIN_1000	1000	Gain 1000
bu3416_GAIN_2000	2000	Gain 2000

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.7 bu3416\_checkAcquisition

```
ViStatus bu3416_checkAcquisition (ViSession instrumentHandle,
                                  ViInt16 *acquisitionState, ViInt16 *error,
                                  ViInt32 *scanBacklog);
```

### Purpose

Returns the state of the last or current Data Acquisition.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### acquisitionState

Variable Type      ViInt16 (passed by reference)

Returns the status of the Data Acquisition process. Possible values are:

bu3416_SM_IDLE	0	DAQ in IDLE state; All needed settings (FE config, Clock and Trigger Config etc.) must be done in this state;
bu3416_SM_DDSUD	1	DAQ in DDS UPDATE state; In this state the card performs the update of the DDS settings;
bu3416_SM_SYNC	2	DAQ in SYNC state; After DDS UPDATE the card performs reset and re-synchronization of ADCs and FIR filters. This stage takes approximately 1 second;
bu3416_SM_READY	3	DAQ in READY state; The synchronization is done and card is ready to start DAQ;
bu3416_SM_POST	5	DAQ in POST-TRIGGER state; Post-trigger samples are stored in FIFO as long as the End Event of DAQ is not happened;
bu3416_SM_END	6	DAQ process terminated successfully;

#### error

Variable Type      ViInt16 (passed by reference)

This parameter contains information about any error happened during Data Acquisition process. The value is a bitmask of the following values:

bu3416_DA_OUTRANGE_ERR	0x0001	Cumulative Out Of Range error;
bu3416_DA_DECIM_ERR	0x0002	Decimation error;
bu3416_DA_ARITH_ERR	0x0004	Arithmetic error;
bu3416_DA_DDSUD_ERR	0x0008	DDS Update Signal error;
bu3416_DA_MCLK_ERR	0x0010	MCLK Clock Frequency error;
bu3416_DA_SCAN_ERR	0x0020	Scan Error;
bu3416_DA_FIFO_OV_ERR	0x0040	FIFO Overflow error;
bu3416_DA_FIFO_UF_ERR	0x0080	FIFO Underflow error;
bu3416_DA_CB_OV_ERR	0x0100	Circular Buffer Overflow error;

**NOTE:**

For error `bu3416_DA_OUTRANGE_ERR` please use function `bu3416_getDAQStatus()` to get more details about error happened on `channel`.

**scanBacklog**

Variable Type            `ViInt32` (passed by reference)

Returns the backlog of scans that have been acquired into the buffer but have not been read using `bu3416_readAcquisition`.

**Return Value**

If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return a warning or error code. Passing the status code to the function `"bu3416_error_message"` will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function `"bu3416_error_message"` will handle all three types of warning/error codes by passing them to the appropriate function if necessary (`"bu3100_error_message"` or `"viStatusDesc"`) to return the correct warning/error message:

**VISA Warnings/Errors:**

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

**BU3100 Warnings/Errors:**

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range `0x3FFC0800` to `0x3FFC0900` and errors in the range `0xBFFC0800` to `0xBFFC0900`. They are defined in the include file `bu3100.h`.

**BU3416 Warnings/Errors:**

Warning codes returned by the 3416 driver functions will be in the range `0x3FFC900` to `0x3FFC0FFF` and errors codes in the range `0xBFFC0900` to `0xBFFC0FFF`. They are defined in the include file `bu3416.h`.

## C.5.8 bu3416\_checkMultAcquisition

```
ViStatus bu3416_checkMultAcquisition (ViSession instrumentHandle,
                                      ViInt16 *acquisitionState, ViInt16 *error,
                                      ViInt32 *scanBacklog);
```

### Purpose

Returns the state of the last or current Data Acquisition process running on the Group of Function Cards operating synchronously.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

#### acquisitionState

Variable Type      ViInt16 (passed by reference)

Returns the status of the Data Acquisition process. Possible values are:

Possible values are:

bu3416_SM_IDLE	0	DAQ in IDLE state; All needed settings (FE config, Clock and Trigger Config etc.) must be done in this state;
bu3416_SM_DDSUD	1	DAQ in DDS UPDATE state; In this state the card performs the update of the DDS settings;
bu3416_SM_SYNC	2	DAQ in SYNC state; After DDS UPDATE the card performs reset and re-synchronization of ADCs and FIR filters. This stage takes approximately 1 second;
bu3416_SM_READY	3	DAQ in READY state; The synchronization is done and card is ready to start DAQ;
bu3416_SM_POST	5	DAQ in POST-TRIGGER state; Post-trigger samples are stored in FIFO as long as the End Event of DAQ is not happened;
bu3416_SM_END	6	DAQ process terminated successfully;

#### error

Variable Type      ViInt16 (passed by reference)

This parameter contains information about any error happened during Data Acquisition process. The value is a bitmask of the following values:

bu3416_DA_OUTRANGE_ERR	0x0001	Cumulative Out Of Range error;
bu3416_DA_DECIM_ERR	0x0002	Decimation error;
bu3416_DA_ARITH_ERR	0x0004	Arithmetic error;
bu3416_DA_DDSUD_ERR	0x0008	DDS Update Signal error;
bu3416_DA_MCLK_ERR	0x0010	MCLK Clock Frequency error;
bu3416_DA_SCAN_ERR	0x0020	Scan Error;

```

bu3416_DA_FIFO_OV_ERR    0x0040    FIFO Overflow error;
bu3416_DA_FIFO_UF_ERR    0x0080    FIFO Underflow error;
bu3416_DA_CB_OV_ERR      0x0100    Circular Buffer Overflow error;

```

**NOTE:**

For error `bu3416_DA_OUTRANGE_ERR` please use function `bu3416_getDAQStatus()` to get more details about error happened on channel.

**scanBacklog**

Variable Type            `ViInt32` (passed by reference)

Returns the backlog of scans that have been acquired into the buffer but have not been read using `bu3416_readAcquisition`.

**Return Value**

If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return a warning or error code. Passing the status code to the function `"bu3416_error_message"` will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function `"bu3416_error_message"` will handle all three types of warning/error codes by passing them to the appropriate function if necessary (`"bu3100_error_message"` or `"viStatusDesc"`) to return the correct warning/error message:

**VISA Warnings/Errors:**

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

**BU3100 Warnings/Errors:**

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range `0x3FFC0800` to `0x3FFC0900` and errors in the range `0xBFFC0800` to `0xBFFC0900`. They are defined in the include file `bu3100.h`.

**BU3416 Warnings/Errors:**

Warning codes returned by the 3416 driver functions will be in the range `0x3FFC900` to `0x3FFC0FFF` and errors codes in the range `0xBFFC0900` to `0xBFFC0FFF`. They are defined in the include file `bu3416.h`.

## C.5.9 bu3416\_clearErrors

```
ViStatus bu3416_clearErrors (ViSession instrumentHandle);
```

### Purpose

This function issues the Clear command, which clears Error Status information.

### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.10 bu3416\_close

```
ViStatus bu3416_close (ViSession instrumentHandle);
```

### Purpose

This function closes the instrument and reclaims the resources allocated by the call to the initialization function `bu3416_init()` or `bu3416_paramInit()`.

This should be called once for every instrument handle returned by the initialize functions prior to terminating the application program.

### Parameter List

`instrumentHandle`

Variable	Type
<code>instrumentHandle</code>	<code>ViSession</code>

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

### C.5.11 bu3416\_enableLIST

```
ViStatus bu3416_enableLIST (ViSession instrumentHandle, ViBoolean enable);
```

#### Purpose

This function forces the usage of the ProDAQ 3150 Motherboard's LIST processor. By default the LIST processor is enabled when the ProDAQ 3416 module is running on 3150 equipped with LIST processor. LIST processor cannot be used while the ProDAQ 3416 module is running on ProDAQ 3120 Motherboard.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### enable

Variable Type      ViBoolean

This parameter enables/disables the ProDAQ 3150 Motherboard's LIST processor.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.12 bu3416\_error\_message

```
ViStatus bu3416_error_message (ViSession instrumentHandle,  
                              ViStatus errorReturnValue, ViChar errorMessage[]);
```

#### Purpose

This function converts a numeric error code returned by one of the functions of this driver into a descriptive error message string.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### errorReturnValue

Variable Type      ViStatus

Accepts the error code returned by one of the functions in this instrument driver. See bu3416.h for Error Codes.

##### errorMessage

Variable Type      ViChar[]

Upon return from the function, holds a text error message which corresponds to the error code.

The VISA Warnings and VISA Errors are described in section 3.3 of the VPP 4.2.2 document and Appendix B of VPP 4.2.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.13 bu3416\_error\_query

```
ViStatus bu3416_error_query (ViSession instrumentHandle, ViInt32 *errorCode,  
                             ViChar errorMessage[]);
```

#### Purpose

This function queries the instrument for latest error code and error message.

NOTE: this function is included for VXIplug&play compatibility but is not supported by this instrument and the function always returns a VI\_WARN\_NSUP\_ERROR\_QUERY warning.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### errorCode

Variable Type      ViInt32 (passed by reference)

Returns the result of the error query.

##### errorMessage

Variable Type      ViChar[]

Upon return from the function, holds a text error message which corresponds to the error code.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.14 bu3416\_fcSelect

```
ViStatus bu3416_fcSelect (ViSession instrumentHandle, ViInt16 functionCard,  
                          ViBoolean resetFC);
```

#### Purpose

Selects the Function Card to be accessed further by the driver's functions.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### functionCard

Variable Type      ViInt16

The function card to which the instrument handler will be bound.

##### resetFC

Variable Type      ViBoolean

Specifies if the Function Card is to be reset to its power-on settings during the initialization procedure.

##### Valid Range:

1 - Yes  
0 - No

Default Value: Yes

NOTE: If you do not want the instrument reset set this control to No while initializing the instrument.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.15 bu3416\_generateITRI

ViStatus bu3416\_generateITRI (ViSession instrumentHandle, ViInt16 function);

#### Purpose

This function asserts/deasserts DA Trigger signal line.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### function

Variable Type      ViInt16

This parameter specifies what function will be performed on Input Trigger line.

Possible values are:

bu3416_OFF	0	The Trigger Line will be switched to inactive state (if all selected Output Trigger sources are in inactive state);
bu3416_ON	1	The Trigger Line will be switched to active state;
bu3416_PULSE	2	The Trigger Line will be switched subsequently to inactive-active-inactive states (if all selected Output Trigger sources are in inactive state);

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.16 bu3416\_generateOTRI

```
ViStatus bu3416_generateOTRI (ViSession instrumentHandle, ViInt16 trigger,
                             ViInt16 function);
```

### Purpose

This function asserts/deasserts selected Output Trigger.

### Parameter List

instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument. If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

trigger

Variable Type      ViInt16

This parameter specifies trigger which will be used. The parameter is a bit mask, so more than one trigger can be used at the same time using bitwise-OR of the following values:

bu3416_TRIG_MBA	0x0001	Motherboard Trigger Stack A
bu3416_TRIG_MBB	0x0002	Motherboard Trigger Stack B
bu3416_TRIG_FP1	0x0004	Front Panel Trigger 1
bu3416_TRIG_FP2	0x0008	Front Panel Trigger 2
bu3416_TRIG_FP3	0x0010	Front Panel Trigger 3

function

Variable Type      ViInt16

This parameter specifies what function will be performed on chosen Trigger. Possible values are:

bu3416_OFF	0	The Trigger Line will be switched to inactive state (if all selected Output Trigger sources are in inactive state);
bu3416_ON	1	The Trigger Line will be switched to active state;
bu3416_PULSE	2	The Trigger Line will be switched subsequently to inactive-active-inactive states (if all selected Output Trigger sources are in inactive state);

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.17 bu3416\_getADCMode

```
ViStatus bu3416_getADCMode (ViSession instrumentHandle, ViInt16 *ADCClockSource,
                             ViInt16 *PLLClockSource, ViInt16 *decimation);
```

#### Purpose

This function returns the ADC operational mode.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### ADCClockSource

Variable Type      ViInt16 (passed by reference)

This parameter returns the source for the ADC Clock.

Possible values are:

bu3416_ADC_CLK_DDS	0	Clock from DDS (Default);
bu3416_ADC_CLK_CCLK	1	Clock from CCLK (Common Clock);
bu3416_ADC_CLK_MBA	2	Clock from MB Input Trigger A;
bu3416_ADC_CLK_FP	3	Clock from FP Input Trigger 1;

##### PLLClockSource

Variable Type      ViInt16 (passed by reference)

This parameter returns the source for the PLL circuitry.

Possible values are:

bu3416_PLL_CLK_OSC	0	Clock from on-board oscillator;
bu3416_PLL_CLK_CCLK	1	Clock from CCLK (Common clock);
bu3416_PLL_CLK_MBA	2	Clock from MB Input Trigger A;
bu3416_PLL_CLK_FP	3	Clock from FP Input Trigger 1;

##### decimation

Variable Type      ViInt16 (passed by reference)

This parameter returns the decimation factor applied at ADC output data.

Possible values are:

bu3416_DECIM_OFF	0	Decimation is off;
bu3416_DECIM_10	1	Decimation is 10;
bu3416_DECIM_100	2	Decimation is 100;
bu3416_DECIM_1000	3	Decimation is 1000;

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

## C.5.18 bu3416\_getBufferSize

```
ViStatus bu3416_getBufferSize (ViSession instrumentHandle, ViInt32 *bufferSize);
```

### Purpose

This function returns the size of the buffer in ProDAQ Motherboard on-board DRAM memory.

### Parameter List

instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

bufferSize

Variable Type      ViInt32 (passed by reference)

Returns the size of the DRAM buffer allocated for the Function Card.

Default size of the DRAM buffer depends on the type of the Function Card Carrier.

### Return Value

If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return a warning or error code. Passing the status code to the function "bu3416\_error\_message" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "bu3416\_error\_message" will handle all three types of warning/error codes by passing them to the appropriate function if necessary ("bu3100\_error\_message" or "viStatusDesc") to return the correct warning/error message:

#### VISA Warnings/Errors:

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

#### BU3100 Warnings/Errors:

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range 0x3FFC0800 to 0x3FFC0900 and errors in the range 0xBFFC0800 to 0xBFFC0900. They are defined in the include file bu3100.h.

#### BU3416 Warnings/Errors:

Warning codes returned by the 3416 driver functions will be in the range 0x3FFC900 to 0x3FFC0FFF and errors codes in the range 0xBFFC0900 to 0xBFFC0FFF. They are defined in the include file bu3416.h.

### C.5.19 bu3416\_getCalibData

```
ViStatus bu3416_getCalibData (ViSession instrumentHandle, ViInt16 channel,
                              ViUInt32 *offset, ViUInt32 *gain);
```

#### Purpose

This function retrieves the calibration data from on-board EEPROM.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### channel

Variable Type      ViInt16

This parameter specifies for which channel the calibration data will be acquired.

Possible values are:

bu3416_CHAN_1	1	Channel 1
bu3416_CHAN_2	2	Channel 2
bu3416_CHAN_3	3	Channel 3
bu3416_CHAN_4	4	Channel 4
bu3416_CHAN_5	5	Channel 5
bu3416_CHAN_6	6	Channel 6
bu3416_CHAN_7	7	Channel 7
bu3416_CHAN_8	8	Channel 8
bu3416_CHAN_9	9	Channel 9
bu3416_CHAN_10	10	Channel 10
bu3416_CHAN_11	11	Channel 11
bu3416_CHAN_12	12	Channel 12
bu3416_CHAN_13	13	Channel 13
bu3416_CHAN_14	14	Channel 14
bu3416_CHAN_15	15	Channel 15
bu3416_CHAN_16	16	Channel 16

##### offset

Variable Type      ViUInt32 (passed by reference)

This parameter returns the calibration coefficient (Offset) which was stored into on-board EEPROM.

##### gain

Variable Type      ViUInt32 (passed by reference)

This parameter returns the calibration coefficient (Gain) which was stored into on-board EEPROM.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

## C.5.20 bu3416\_getDAQMode

```
ViStatus bu3416_getDAQMode (ViSession instrumentHandle, ViInt16 *boardMode,
                             ViInt16 *startMode, ViInt16 *stopMode,
                             ViInt16 *stopOnError);
```

### Purpose

This function returns the Data Acquisition operational mode.

### Parameter List

#### instrumentHandle

Variable Type            ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### boardMode

Variable Type            ViInt16 (passed by reference)

This parameter returns the Function Card operational mode.

Possible values are:

bu3416_FC_STALONE	0	The Board operates in a Stand-Alone mode (Default);
bu3416_FC_MASTER	1	The Board operates as a Master in Multi-Card operational mode;
bu3416_FC_SLAVE	2	The Board operates as a Slave in Multi-Card operational mode;

#### startMode

Variable Type            ViInt16 (passed by reference)

This Parameter returns the Start Mode of Data Acquisition.

Possible values are:

bu3416_DA_START_IMM	0	Data Acquisition starts immediately after synchronization is done (Default);
bu3416_DA_START_TRIG	1	Data Acquisition starts when Input Trigger goes active;

#### stopMode

Variable Type            ViInt16 (passed by reference)

This Parameter returns the Stop Mode of Data Acquisition.

Possible values are:

bu3416_DA_STOP_COUNT	0	Data Acquisition stops when the specified number of samples has been collected (Default);
bu3416_DA_STOP_TRIG	1	Data Acquisition stops when Input Trigger Stop Event happened;
bu3416_DA_STOP_GATE	2	DA stops when Trigger goes inactive;
bu3416_DA_STOP_UNLIM	3	Data Acquisition stops only when DAQ STOP command issued;

**stopOnError**

Variable Type            ViInt16 (passed by reference)

This parameter returns what kind of error will break data acquisition (DA).

Possible values are:

bu3416_DA_STOP_ERR_OFF	0	Errors don't stop DA;
bu3416_DA_STOP_ERR_EXOFR	1	Any error excluding OUTRANGE stops DA;
bu3416_DA_STOP_ERR_ANY	2	Any error stop DA;

**Return Value**

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.21 bu3416\_getDAQStatus

```
ViStatus bu3416_getDAQStatus (ViSession instrumentHandle, ViInt16 *state,
                              ViInt16 *errors, ViInt32 *chErrors);
```

#### Purpose

This function returns status of the Data Acquisition process.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### state

Variable Type      ViInt16 (passed by reference)

This parameter returns the DAQ State Machine state.

Possible values are:

bu3416_SM_IDLE	0	DAQ in IDLE state; All needed settings (FE config, Clock and Trigger Config etc.) must be done in this state;
bu3416_SM_DDSUD	1	DAQ in DDS UPDATE state; In this state the card performs the update of the DDS settings;
bu3416_SM_SYNC	2	DAQ in SYNC state; After DDS UPDATE the card performs reset and re-synchronization of ADCs and FIR filters. This stage takes approximately 1 second;
bu3416_SM_READY	3	DAQ in READY state; The synchronization is done and card is ready to start DAQ;
bu3416_SM_POST	5	DAQ in POST-TRIGGER state; Post-trigger samples are stored in FIFO as long as the End Event of DAQ is not happened;
bu3416_SM_END	6	DAQ process terminated successfully;

##### errors

Variable Type      ViInt16 (passed by reference)

This parameter contains information about any error happened during Data Acquisition process. The value is a bitmask of the following values:

bu3416_DA_OUTRANGE_ERR	0x0001	Cumulative Out Of Range error;
bu3416_DA_DECIM_ERR	0x0002	Decimation error;
bu3416_DA_ARITH_ERR	0x0004	Arithmetic error;
bu3416_DA_DDSUD_ERR	0x0008	DDS Update Signal error;
bu3416_DA_MCLK_ERR	0x0010	MCLK Clock Frequency error;
bu3416_DA_SCAN_ERR	0x0020	Scan Error;
bu3416_DA_FIFO_OV_ERR	0x0040	FIFO Overflow error;
bu3416_DA_FIFO_UF_ERR	0x0080	FIFO Underflow error;

**chErrors**

Variable Type            ViInt32 (passed by reference)

This parameter contains information error happened on channel during Data Acquisition process. The value is a bitmask of the following values:

bu3416_CH1_OUTRANGE	0x00000001	Channel 1	Out Of Range error
bu3416_CH2_OUTRANGE	0x00000002	Channel 2	Out Of Range error
bu3416_CH3_OUTRANGE	0x00000004	Channel 3	Out Of Range error
bu3416_CH4_OUTRANGE	0x00000008	Channel 4	Out Of Range error
bu3416_CH5_OUTRANGE	0x00000010	Channel 5	Out Of Range error
bu3416_CH6_OUTRANGE	0x00000020	Channel 6	Out Of Range error
bu3416_CH7_OUTRANGE	0x00000040	Channel 7	Out Of Range error
bu3416_CH8_OUTRANGE	0x00000080	Channel 8	Out Of Range error
bu3416_CH9_OUTRANGE	0x00000100	Channel 9	Out Of Range error
bu3416_CH10_OUTRANGE	0x00000200	Channel 10	Out Of Range error
bu3416_CH11_OUTRANGE	0x00000400	Channel 11	Out Of Range error
bu3416_CH12_OUTRANGE	0x00000800	Channel 12	Out Of Range error
bu3416_CH13_OUTRANGE	0x00001000	Channel 13	Out Of Range error
bu3416_CH14_OUTRANGE	0x00002000	Channel 14	Out Of Range error
bu3416_CH15_OUTRANGE	0x00004000	Channel 15	Out Of Range error
bu3416_CH16_OUTRANGE	0x00008000	Channel 16	Out Of Range error

**Return Value**

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.22 bu3416\_getDDSFreq

```
ViStatus bu3416_getDDSFreq (ViSession instrumentHandle, ViReal64 *frequencyHz);
```

### Purpose

This function returns the frequency (in Hz) of the DDS generator.

### Parameter List

**instrumentHandle**

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

**frequencyHz**

Variable Type      ViReal64 (passed by reference)

This parameter returns the actual frequency what DDS generator was set up for.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.23 bu3416\_getFIFOConfig

```
ViStatus bu3416_getFIFOConfig (ViSession instrumentHandle,
                               ViInt16 *affThreshold);
```

#### Purpose

This function gets FIFO configuration.

#### Parameter List

instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

affThreshold

Variable Type      ViInt16 (passed by reference)

This parameter returns the threshold level for Programmable Almost Full Flag (PAFF).

Possible values are:

2 to (FIFO\_SIZE-1) where FIFO\_SIZE is 8192 or 16384 depending on card version

#### NOTE:

The following table shows configuration for FIFO flags:

Number of samples in FIFO	Empty	Almost Full	Full
0	1	0	0
1 to (aftThreshold-1)	0	0	0
aftThreshold to (FIFO_SIZE-1)	0	1	0
FIFO_SIZE	0	0	1

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.24 bu3416\_getFIFOStatus

```
ViStatus bu3416_getFIFOStatus (ViSession instrumentHandle, ViInt16 *FIFOFlags,
                               ViInt16 *FIFOCounter);
```

### Purpose

This function returns the information about current load of FIFO.

### Parameter List

instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

FIFOFlags

Variable Type      ViInt16 (passed by reference)

This parameter returns the status of the FIFO Flags. Possible values are:

bu3416_FIFO_EMPTY	0	No Samples stored in FIFO;
bu3416_FIFO_NEMPTY	1	FIFO contains 1 to FIFO_AFT-1 samples
bu3416_FIFO_AFULL	2	FIFO contains FIFO_AFT to FIFO_SIZE-1 samples;
bu3416_FIFO_FULL	3	FIFO contains FIFO_SIZE samples;
bu3416_FIFO_OV	4	Overflow - write request was rejected because the FIFO is full;
bu3416_FIFO_UF	5	Underflow - read request was rejected because the FIFO is empty;

where FIFO\_AFT is Almost Full Threshold register.  
See bu3416\_setFIFOConfig() function.

FIFOCounter

Variable Type      ViInt16 (passed by reference)

This parameter returns a lower 8-bit part of a counter for a number of samples stored in the FIFO memory.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.25 bu3416\_getFPTrigPolarity

```
ViStatus bu3416_getFPTrigPolarity (ViSession instrumentHandle, ViInt16 *polFPT1,
                                   ViInt16 *polFPT2, ViInt16 *polFPT3);
```

#### Purpose

This function gets active state levels for Front Panel Triggers.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### polFPT1

Variable Type      ViInt16 (passed by reference)

This parameter returns level of active state of Front Panel Trigger 1  
Possible values are:

bu3416_LOW	0	active state for FP trigger 1 is low (zero)
bu3416_HIGH	1	active state for FP trigger 1 is high (one)

##### polFPT2

Variable Type      ViInt16 (passed by reference)

This parameter returns level of active state of Front Panel Trigger 2  
Possible values are:

bu3416_LOW	0	active state for FP trigger 2 is low (zero)
bu3416_HIGH	1	active state for FP trigger 2 is high (one)

##### polFPT3

Variable Type      ViInt16 (passed by reference)

This parameter returns level of active state of Front Panel Trigger 3  
Possible values are:

bu3416_LOW	0	active state for FP trigger 3 is low (zero)
bu3416_HIGH	1	active state for FP trigger 3 is high (one)

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.26 bu3416\_getITRConfig

```
ViStatus bu3416_getITRConfig (ViSession instrumentHandle,
                             ViInt16 *daTrigSource, ViInt16 *syncSource);
```

### Purpose

This function returns the configuration of the Input Trigger.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### daTrigSource

Variable Type      ViInt16 (passed by reference)

This parameter returns what sources will be chosen for Data Acquisition (DA) Trigger. The parameter is a bit mask, so more than one source can be used at the same time using bitwise-OR of the following values:

bu3416_DA_TRIG_OFF	0x0000
bu3416_DA_TRIG_MBA	0x0001
bu3416_DA_TRIG_MBB	0x0002
bu3416_DA_TRIG_FP3	0x0004

#### syncSource

Variable Type      ViInt16 (passed by reference)

This parameter returns the source for the SYNC signal.

Possible values are:

bu3416_SYNC_MBB	0	Motherboard Input Trigger Stack B;
bu3416_SYNC_FP	1	Front Panel SYNC signal;

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.27 bu3416\_getITRIState

```
ViStatus bu3416_getITRIState (ViSession instrumentHandle, ViInt16 *state,
                             ViInt16 *cclkState);
```

#### Purpose

Gets input triggers lines state.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### state

Variable Type      ViInt16 (passed by reference)

This parameter returns bit-OR mask with states of input trigger lines.

Possible mask values are:

bu3416_TRIG_MBA	0x0001	bit0 if set MB Trigger A is active;
bu3416_TRIG_MBB	0x0002	bit1 if set MB Trigger B is active;
bu3416_TRIG_FP1	0x0004	bit2 if set FP Trigger 1 is active;
bu3416_TRIG_FP2	0x0008	bit3 if set FP Trigger 2 is active;
bu3416_TRIG_FP3	0x0010	bit4 if set FP Trigger 3 is active;

##### cclkState

Variable Type      ViInt16 (passed by reference)

This parameter returns a status of Common Clock (CCLK) input line.

Possible values:

bu3416_CCLK_TOGG_0MHZ	0	CCLK at low level for at least 10 us;
bu3416_CCLK_TOGG_2MHZ	1	CCLK toggles at 2MHz;
bu3416_CCLK_TOGG_5MHZ	2	CCLK toggles at 5MHz;
bu3416_CCLK_TOGG_10MHZ	3	CCLK toggles at 10MHz;
bu3416_CCLK_TOGG_XMHZ	4	CCLK toggles at another frequency;

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.28 bu3416\_getMultFCsession

```
ViStatus bu3416_getMultFCsession (ViSession instrumentHandle,
                                  ViInt16 functionCardIndex,
                                  ViSession *FCInstrumentHandle);
```

### Purpose

This function returns the Instrument Handle for individual Function Card included into the Group.

This Handle can be used to customize some Function Card settings individually using Instrument Driver Functions not included into the Multi-Card Acquisition sub-class.

However, those function should be used carefully, as they can change Function Card settings important for synchronous operation mode.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

#### functionCardIndex

Variable Type      ViInt16

Specifies the index of the Function Card in the Group for which the Instrument Handle is required.

#### FCInstrumentHandle

Variable Type      ViSession (passed by reference)

Returns the Instrument Handle for individual Function Card included into the Group.

This Handle can be used to customize some Function Card settings individually using Instrument Driver Functions not included into the Multi-Card Acquisition sub-class.

However, those function should be used carefully, as they can change Function Card settings important for synchronous operation mode.

### Return Value

If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return a warning or error code. Passing the status code to the function "bu3416\_error\_message" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "bu3416\_error\_message" will handle all three types of warning/error codes by passing them to the appropriate function if necessary ("bu3100\_error\_message" or "viStatusDesc") to return the correct warning/error message:

#### VISA Warnings/Errors:

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

**BU3100 Warnings/Errors:**

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range 0x3FFC0800 to 0x3FFC0900 and errors in the range 0xBFFC0800 to 0xBFFC0900. They are defined in the include file bu3100.h.

**BU3416 Warnings/Errors:**

Warning codes returned by the 3416 driver functions will be in the range 0x3FFC900 to 0x3FFC0FFF and errors codes in the range 0xBFFC0900 to 0xBFFC0FFF. They are defined in the include file bu3416.h.

## C.5.29 bu3416\_getOTRConfig

```
ViStatus bu3416_getOTRConfig (ViSession instrumentHandle, ViInt16 *sourceMBA,
                              ViInt16 *sourceMBB, ViInt16 *sourceFPOT1,
                              ViInt16 *sourceFPOT2, ViInt16 *sourceFPOT3,
                              ViInt16 *sourceDE, ViInt16 *sourceDI);
```

### Purpose

This function returns the configuration of the Output Trigger.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### sourceMBA

Variable Type      ViInt16 (passed by reference)

This parameter returns what source is chosen for Motherboard Output Trigger Stack A (MBA).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_FPIT3_MBA	1	FP trigger input 3 is routed to MBA;
bu3416_RCLK_MBA	2	Ref. clock (2MHz) is routed to MBA;
bu3416_ADCC_MBA	3	ADC clock is is routed to MBA;
bu3416_DA_ST_MBA	4	DA progress signal is routed to MBA;
bu3416_OUTRANGE_MBA	5	Out of range signal is routed to MBA;
bu3416_FIFO_PAFF_MBA	6	FIFO PAF flag is routed to MBA;

#### sourceMBB

Variable Type      ViInt16 (passed by reference)

This parameter returns what source is chosen for Motherboard Output Trigger Stack B (MBB).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_FPIT2_MBB	1	FP trigger input 2 is routed to MBB;
bu3416_SYNC_MBB	2	SYNC is routed to MBB;
bu3416_SCANP_MBB	4	Pulse per Scan sig. is routed to MBB;
bu3416_OUTRANGE_MBB	5	Out of range signal is routed to MBB;
bu3416_FIFO_PAFF_MBB	6	FIFO PAF flag is routed to MBB;

#### sourceFPOT1

Variable Type      ViInt16 (passed by reference)

This parameter returns what source is chosen for Front Panel Output Trigger 1 (FPOT1).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_MBA_FPOT1	1	MB Trigger input 'A' is routed to FPOT1;
bu3416_CCLK_FPOT1	2	Common Clock (CCLK) is routed to FPOT1;
bu3416_RCLK_FPOT1	3	Ref. clock (2MHz) is routed to FPOT1;
bu3416_ADCC_FPOT1	4	ADC clock is routed to FPOT1;
bu3416_DA_ST_FPOT1	6	DA progress signal is routed to FPOT1;
bu3416_SWA_FPOT1	7	Software assertion mode.

## sourceFPOT2

Variable Type            ViInt16 (passed by reference)

This parameter returns what source is chosen for Front Panel Output Trigger 2 (FPOT2).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_MBB_FPOT2	1	MB Trigger input 'B' is routed to FPOT2;
bu3416_SYNC_FPOT2	2	SYNC is routed to FPOT2;
bu3416_OUTRANGE_FPOT2	5	Out of range signal is routed to FPOT2;
bu3416_SCANP_FPOT2	6	Pulse per Scan sig. is routed to FPOT2;
bu3416_SWA_FPOT2	7	Software assertion mode.

## sourceFPOT3

Variable Type            ViInt16 (passed by reference)

This parameter returns what source is chosen for Front Panel Output Trigger 3 (FPOT3).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_MBA_FPOT3	1	MB Trigger input 'A' is routed to FPOT3;
bu3416_MBB_FPOT3	2	MB Trigger input 'B' is routed to FPOT3;
bu3416_SWA_FPOT3	7	Software assertion mode.

## sourceDE

Variable Type            ViInt16 (passed by reference)

This parameter returns what source will be chosen for Direct Error (DE).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_OUTRANGE_DE	1	Out of range signal is routed to DE;
bu3416_ANY_DE	2	Any error asserts DE line;

## sourceDI

Variable Type            ViInt16 (passed by reference)

This parameter returns what source is chosen for Direct Interrupt (DI).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_FIFO_PAFF_DI	1	FIFO PAFF flag is routed to DI;
bu3416_OUTRANGE_DI	2	OUTRANGE signal is routed to DI;

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

### C.5.30 bu3416\_getPostScans

```
ViStatus bu3416_getPostScans (ViSession instrumentHandle, ViInt32 *scans);
```

#### Purpose

This function returns the configured post-trigger number of scans to collect.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### scans

Variable Type      ViInt32 (passed by reference)

This parameter returns the configured post-trigger number of scans to collect.

Possible values are from 0 to 0xffffffff.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.31 bu3416\_getSampFreq

```
ViStatus bu3416_getSampFreq (ViSession instrumentHandle, ViReal64 frequencyHz);
```

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### frequencyHz

Variable Type      ViReal64

This parameter returns the actual sampling frequency what was set up for.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.32 bu3416\_getSerNum

```
ViStatus bu3416_getSerNum (ViSession instrumentHandle, ViInt32 *serialNumber);
```

#### Purpose

This function returns card's serial number.

#### Parameter List

**instrumentHandle**

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

**serialNumber**

Variable Type      ViInt32 (passed by reference)

Contains the serial number of the 3416 function card.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.33 bu3416\_init

```
ViStatus bu3416_init (ViRsrc instrumentDescriptor, ViBoolean IDQuery,
                    ViBoolean resetDevice, ViSession *instrumentHandle);
```

#### Purpose

Initializes the instrument and returns an "instrument handle". The instrument handle must be used with all of the other functions of this driver.

The initialize call allows the instrument to be queried to ensure that it is a Bustec Data Acquisition System. It also resets the Module to the power-up state if the "Reset" parameter is True (ON).

This function interrogates the motherboard registers to ascertain in which locations there are function cards fitted and then checks those locations to identify the type of function card fitted.

NOTE: that for each "bu3416\_init()" call, a new unique instrument handle is returned. Thus, if four calls are made to the initialize call in succession, four unique instrument handles will be returned.

After call of "bu3416\_init()", the function "bu3416\_fcSelect()" must be called to bind the acquired instrument handler to the specific Function Card.

For each instrument handle returned by the "bu3416\_init()" function, the "bu3416\_close()" function should be called to free up the resources allocated by "bu3416\_init()". The call(s) to "bu3416\_close()" should be made before the application program terminates.

#### Parameter List

##### instrumentDescriptor

Variable Type      ViRsrc

Specifies which remote instrument to establish a communication session with. Based on the syntax of the Instr Descriptor, the Initialize function configures the I/O interface and generates an Instr Handle

The default value is for a VXI interface for logical address 7:-

Default Value: "VXI::7::INSTR"

Based on the Instrument Descriptor, this operation establishes a communication session with a device.

##### IDQuery

Variable Type      ViBoolean

Specifies if an ID Query is sent to the instrument during the initialization procedure.

Valid Range:

1 - Yes

0 - No

Default Value: Yes

NOTE: Under normal circumstances the ID Query ensures that the instrument initialized over the bus is the type supported by this driver. However, circumstances may arise where it is undesirable to

send an ID Query to the instrument. In those cases set this control to Skip Query and this function will initialize the bus and the Command arrays in the driver, without doing an ID Query.

#### resetDevice

Variable Type            ViBoolean

Specifies if the instrument is to be reset to its power-on settings during the initialization procedure.

#### Valid Range:

1 - Yes

0 - No

Default Value: Yes

NOTE: If you do not want the instrument reset, set this control to No while initializing the instrument.

#### instrumentHandle

Variable Type            ViSession (passed by reference)

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

NOTE: A new (unique) handle will be returned EACH time the Initialize function is called. The bu3416\_close() call should be used for EVERY handle returned by the bu3416\_init() function.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.34 bu3416\_multClose

```
ViStatus bu3416_multClose (ViSession instrumentHandle);
```

#### Purpose

This function closes all Function Cards included in the Group and reclaims the resources allocated by the call to the Function Card Group initialization function bu3416\_multInit().

#### Parameter List

instrumentHandle

Variable Type            ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

#### Return Value

If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return a warning or error code. Passing the status code to the function "bu3416\_error\_message" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "bu3416\_error\_message" will handle all three types of warning/error codes by passing them to the appropriate function if necessary ("bu3100\_error\_message" or "viStatusDesc") to return the correct warning/error message:

##### VISA Warnings/Errors:

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

##### BU3100 Warnings/Errors:

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range 0x3FFC0800 to 0x3FFC0900 and errors in the range 0xBFFC0800 to 0xBFFC0900. They are defined in the include file bu3100.h.

##### BU3416 Warnings/Errors:

Warning codes returned by the 3416 driver functions will be in the range 0x3FFC900 to 0x3FFC0FFF and errors codes in the range 0xBFFC0900 to 0xBFFC0FFF. They are defined in the include file bu3416.h.

### C.5.35 bu3416\_multConfig

```
ViStatus bu3416_multConfig (ViSession instrumentHandle, ViInt16 synchMode,
                           ViInt16 channelMask[], ViReal64 sampleRateHz,
                           ViInt16 startMode, ViInt32 scansToCollect,
                           ViInt16 BPTrigClock, ViInt16 BPTrigSynch);
```

#### Purpose

This function configures the group of ProDAQ 3416 function cards for synchronous operation. The group of function cards should be initialized prior to this function call by using bu3416\_multInit() function. If the group will be configured in Triggered mode, the input trigger for this group should be configured prior to this function call by using bu3416\_setMultTrigConfig() function.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

##### synchMode

Variable Type      ViInt16

This parameter specifies how the Function Cards will be synchronized. Possible values are:

bu3416\_MB\_SYNC\_MODE 0 Synchronization will be done using ProDAQ Motherboard triggering system. If the Function Cards are located in different ProDAQ Motherboards then two VXI Backplane TTL trigger lines will be used as well;

bu3416\_FP\_SYNC\_MODE 1 Synchronization will be done using Clock and Sync Front Panel connectors which should be connected appropriately using the external cables;

##### channelMask

Variable Type      ViInt16[]

Selects the channels from which the data will be taken. This is an array of 16-bit values, each element corresponds to the appropriate Function Card from the list of Function Cards used for Multiple Card Initialization.

The array must contain the number of elements corresponding to the number of Function Cards initialized in the Group;

Each element in this array is a bitmask of 16 channels:

bit 0 corresponds to the Channel 1  
 ...  
 bit 15 corresponds to the Channel 16

"1" written in the appropriate bit means that the channel will be included in the data acquisition.

#### sampleRateHz

Variable Type            ViReal64

This parameter specifies the sample rate (in Hertz) for data acquisition process. Possible values are from 1.0 (1Hz) to 10000.0 (10kHz)

#### startMode

Variable Type            ViInt16

This Parameter specifies the Start Mode of Data Acquisition.

Possible values are:

bu3416_DA_START_IMM	0	Data Acquisition starts immediately after synchronization is done (Default);
bu3416_DA_START_TRIG	1	Data Acquisition starts when Input Trigger goes active;

#### scansToCollect

Variable Type            ViInt32

This parameter specifies total number of scans to collect by data acquisition (including pre-triggered scans). If this parameter contains '0', the data acquisition will run in unlimited (continuous) mode.

If the trigger is enabled then the number of scans should not be less than the number of pre-triggered scans.

#### BPTrigClock

Variable Type            ViInt16

This control allows to select the VXI Backplane TTL Trigger line if one should be used for Multiple Function Card synchronization (See description of "Synch Mode" control for more information). This line will propagate common Clock signal.

Possible values are:

VI\_TRIG\_TTL0  
to  
VI\_TRIG\_TTL7

VI\_TRIG\_ALL - VXI Backplane TTL Trigger line will be selected automatically.

#### BPTrigSynch

Variable Type            ViInt16

This control allows to select the VXI Backplane TTL Trigger line if one should be used for Multiple Function Card synchronization (See description of "Synch Mode" control for more information). This line will propagate common Synch signal.

Possible values are:

VI\_TRIG\_TTL0  
to  
VI\_TRIG\_TTL7

VI\_TRIG\_ALL - VXI Backplane TTL Trigger line will be selected automatically.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.36 bu3416\_multInit

```
ViStatus bu3416_multInit (ViRsrc instrumentDescriptor, ViInt16 functionCards[],
                        ViBoolean IDQuery, ViBoolean resetDevice,
                        ViSession *instrumentHandle);
```

#### Purpose

Initializes the group of specified Function Cards for synchronous operations and returns the "instrument handle" associated with this group of Function Cards.

**IMPORTANT NOTE:** The instrument handle must be used with all of the other functions of Multi-Card Acquisition sub-class and it cannot be used with any other function of the Instrument driver.

The function cards can be fitted in the same or in different ProDAQ motherboard modules, but they must be located in the same VXI crate. First function card in the list will operate as a Master, while all others will operate as Slaves.

If the function cards are located on the same ProDAQ motherboard module, they will be synchronized over ProDAQ motherboard internal trigger system. If the function cards are located on different ProDAQ motherboard modules, two VXI backplane TTL trigger lines will be used for synchronization.

"Instrument Descriptor" and "Function Cards" parameters describe the list of function cards which should be initialized for synchronous operation. For example, there are three Function Cards:

two are fitted to the same motherboard with logical address 3 as FC1 and FC5 and the third one is fitted to another motherboard with logical address 5 as FC2.

To initialize them, the comma-separated list of instrument descriptors for all three function cards should be passed to the "Instrument Descriptor" parameter and array of appropriate function card positions should be passed to "Function Cards" control:

```
ViInt16 fcs[3]={1,5,2};
...
bu3416_multInit("VXI0::3::INSTR,VXI0::5::INSTR", fcs, ...);
```

#### Parameter List

instrumentDescriptor

Variable	Type	ViRsrc
instrumentDescriptor		

Comma-separated instrument descriptors list for each 3416 function card which should be initialized for synchronous operation.

"Instrument Descriptor" and "Function Cards" parameters describe the list of function cards which should be initialized for synchronous operation.

For example, there are three Function Cards:

two are fitted to the same motherboard with logical address 3 as FC1 and FC5 and the third one is fitted to another motherboard with logical address 5 as FC2.

To initialize them, the comma-separated list of instrument descriptors for all three function cards should be passed to the "Instrument Descriptor" parameter and array of appropriate function card positions should be passed to "Function Cards" control:

```
ViInt16 fcs[3]={1,5,2};
...
bu3416_multInit("VXI0::3::INSTR,VXI0::5::INSTR", fcs, ...);
```

## functionCards

Variable Type            ViInt16[]

This control should contain an array of the Function Cards positions (numbered from 1 to 8).

"Instrument Descriptor" and "Function Cards" parameters describe the list of function cards which should be initialized for synchronous operation.

For example, there are three Function Cards:

two are fitted to the same motherboard with logical address 3 as FC1 and FC5 and the third one is fitted to another motherboard with logical address 5 as FC2.

To initialize them, the comma-separated list of instrument descriptors for all three function cards should be passed to the "Instrument Descriptor" parameter and array of appropriate function card positions should be passed to "Function Cards" control:

```
ViInt16 fcs[3]={1,5,2};
...
bu3416_multInit("VXI0::3::INSTR,VXI0::5::INSTR", fcs, ...);
```

## IDQuery

Variable Type            ViBoolean

Specifies if an ID Query is sent to the instrument during the initialization procedure.

Valid Range:

1 - Yes  
0 - No

Default Value: Yes

NOTE: Under normal circumstances the ID Query ensures that the instrument initialized over the bus is the type supported by this driver. However, circumstances may arise where it is undesirable to send an ID Query to the instrument. In those cases set this control to Skip Query and this function will initialize the bus and the Command arrays in the driver, without doing an ID Query.

## resetDevice

Variable Type            ViBoolean

Specifies if the instrument is to be reset to its power-on settings during the initialization procedure.

Valid Range:

1 - Yes  
0 - No

Default Value: Yes

NOTE: If you do not want the instrument reset, set this control to No while initializing the instrument.

## instrumentHandle

Variable Type            ViSession (passed by reference)

The Instrument Handle is used to identify the unique session or communication channel between the driver and the selected group of the function cards.

IMPORTANT NOTE: The instrument handle must be used with all of the

other functions of Multi-Card Acquisition sub-class and it cannot be used with any other function of the Instrument driver.

A `bu3416_multClose()` function should be called to close this instrument handle.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

### C.5.37 bu3416\_paramInit

```
ViStatus bu3416_paramInit (ViRsrc instrumentDescriptor, ViInt16 functionCard,
                          ViBoolean IDQuery, ViBoolean resetDevice,
                          ViSession *instrumentHandle);
```

#### Purpose

Initializes the specified Function Card on the specified ProDAQ module and returns an "instrument handle". The instrument handle must be used with all of the other functions of this driver.

The Initialize With Parameters call allows the VXI module to be queried to ensure that it is a Bustec Data Acquisition System and the selected Function Card is one of the appropriate type (bu3416 Function Card). It also resets the Module to the power-up state if the "Reset" parameter is True (ON).

#### Parameter List

##### instrumentDescriptor

Variable Type      ViRsrc

Specifies which remote instrument to establish a communication session with. Based on the syntax of the Instr Descriptor, the Initialize function configures the I/O interface and generates an Instr Handle

##### functionCard

Variable Type      ViInt16

The function card to which the instrument handler will be bound.

##### IDQuery

Variable Type      ViBoolean

Specifies if an ID Query is sent to the instrument during the initialization procedure.

Valid Range:

1 - Yes

0 - No

Default Value: Yes

NOTE: Under normal circumstances the ID Query ensures that the instrument initialized over the bus is the type supported by this driver. However, circumstances may arise where it is undesirable to send an ID Query to the instrument. In those cases set this control to Skip Query and this function will initialize the bus and the Command arrays in the driver, without doing an ID Query.

##### resetDevice

Variable Type      ViBoolean

Specifies if the instrument is to be reset to its power-on settings during the initialization procedure.

Valid Range:

1 - Yes

0 - No

Default Value: Yes

NOTE: If you do not want the instrument reset, set this control to No while initializing the instrument.

#### instrumentHandle

Variable Type            ViSession (passed by reference)

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

NOTE: A new (unique) handle will be returned EACH time the Initialize function is called. The bu3416\_close() call should be used for EVERY handle returned by the bu3416\_init() function.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.38 bu3416\_readAcquisition

```
ViStatus bu3416_readAcquisition (ViSession instrumentHandle,
                                 ViInt32 scanstoRead, ViInt16 fillMode,
                                 ViInt32 *scanBacklog, ViInt32 *actualScansRead,
                                 ViReal64 waveforms[]);
```

#### Purpose

Fetches the specified amount of data from the function card.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### scanstoRead

Variable Type      ViInt32

Specifies the number of scans to be fetched from Function Card.

Each scan contains one sample per selected channel. For instance, if 3 channels were selected for Data Acquisition and number of scans is 1000, then 3000 samples will be stored in the output buffer.

##### fillMode

Variable Type      ViInt16

The parameter specifies whether the Waveform array will be grouped by channels or grouped by scans.

Possible values are:

```
bu3416_GROUP_BY_CHANNEL 0 Group data by channel
bu3416_GROUP_BY_SCAN   1 Group data by scans
```

For example:

If you scan channels A through C and Number of Scans is 5, then the possible fill modes are:

Grouped by channel:

```
A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C4 C5
 \-----/ \-----/ \-----/
or
```

Grouped by scan:

```
A1 B1 C1 A2 B2 C2 A3 B3 C3 A4 B4 C4 A5 B5 C5
 \----/ \----/ \----/ \----/ \----/
```

If you are to pass the array to a graph, you should acquire the data grouped by channel.

If you are to pass the array to a strip chart, you should acquire the data grouped by scan.

**scanBacklog**

Variable Type            ViInt32 (passed by reference)

Returns the backlog of scans that have been acquired into the buffer but have not been read using `bu3416_readAcquisition`.

**actualScansRead**

Variable Type            ViInt32 (passed by reference)

Returns the number of scans fetched from the function card and stored in the output buffer. Each scan contains one sample per selected channel. For instance, if 3 channels were selected for Data Acquisition and number of scans is 1000, then 3000 samples will be stored in the output buffer.

**waveforms**

Variable Type            ViReal64[]

The output buffer containing the samples fetched from Function Card FIFO. This buffer should be allocated by application before the function call with appropriate size to hold all data. Samples in this buffer are arranged according to "Fill Mode" parameter".

**Return Value**

If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return a warning or error code. Passing the status code to the function "`bu3416_error_message`" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "`bu3416_error_message`" will handle all three types of warning/error codes by passing them to the appropriate function if necessary ("`bu3100_error_message`" or "`viStatusDesc`") to return the correct warning/error message:

**VISA Warnings/Errors:**

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

**BU3100 Warnings/Errors:**

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range `0x3FFC0800` to `0x3FFC0900` and errors in the range `0xBFFC0800` to `0xBFFC0900`. They are defined in the include file `bu3100.h`.

**BU3416 Warnings/Errors:**

Warning codes returned by the 3416 driver functions will be in the range `0x3FFC900` to `0x3FFC0FFF` and errors codes in the range `0xBFFC0900` to `0xBFFC0FFF`. They are defined in the include file `bu3416.h`.

### C.5.39 bu3416\_readFIFO

```
ViStatus bu3416_readFIFO (ViSession instrumentHandle, ViInt32 count,  
                          ViInt32 data[]);
```

#### Purpose

This function reads the specified amount of data from FIFO. The required amount of data should not be bigger than actual number of samples stored in FIFO.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### count

Variable Type      ViInt32

This parameter specifies how many 32-bit samples should be read from FIFO. The required amount of samples should not be bigger than actual number of samples stored in FIFO.

##### data

Variable Type      ViInt32[]

This parameter contains the pointer to the user buffer which will hold the requested data upon successful function call. The buffer should be allocated prior to function call with size big enough to hold all requested data.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.40 bu3416\_readMultAcquisition

```
ViStatus bu3416_readMultAcquisition (ViSession instrumentHandle,
                                     ViInt32 scanstoRead, ViReal64 waveforms[]);
```

### Purpose

Fetches the specified amount of data from all Function Cards operating synchronously in the Group.

The amount of data available for reading should be obtained by using bu3416\_checkMultAcquisition() prior to this function call.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

#### scanstoRead

Variable Type      ViInt32

Specifies the number of scans to be fetched from Function Card.

The amount of scans available for reading should be obtained by using bu3416\_checkMultAcquisition() prior to this function call.

Each scan contains one sample per selected channel. For instance, if 3 channels were selected for Data Acquisition and number of scans is 1000, then 3000 samples will be stored in the output buffer.

#### waveforms

Variable Type      ViReal64[]

The output buffer containing the samples fetched from Function Cards. This buffer should be allocated by application before the function call with appropriate size to hold all data.

Samples in this buffer are grouped by channel, for example:

If you scan channels A through C and Number of Scans is 5, then data is grouped in the following way:

```

A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C4 C5
 \-----/ \-----/ \-----/
```

### Return Value

If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return a warning or error code. Passing the status code to the function "bu3416\_error\_message" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "bu3416\_error\_message" will handle all three types of warning/error codes by passing them to the appropriate

function if necessary ("bu3100\_error\_message" or "viStatusDesc") to return the correct warning/error message:

**VISA Warnings/Errors:**

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

**BU3100 Warnings/Errors:**

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range 0x3FFC0800 to 0x3FFC0900 and errors in the range 0xBFFC0800 to 0xBFFC0900. They are defined in the include file bu3100.h.

**BU3416 Warnings/Errors:**

Warning codes returned by the 3416 driver functions will be in the range 0x3FFC900 to 0x3FFC0FFF and errors codes in the range 0xBFFC0900 to 0xBFFC0FFF. They are defined in the include file bu3416.h.

### C.5.41 bu3416\_readTEDS\_EEPROM

```
ViStatus bu3416_readTEDS_EEPROM (ViSession instrumentHandle, ViChar *buf,  
                                ViUInt16 addr, ViUInt16 cnt);
```

#### Purpose

Reads "cnt" bytes from TEDS 256-bit EEPROM starting at "addr" address.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### buf

Variable Type      ViChar (passed by reference)

Buffer for result.

##### addr

Variable Type      ViUInt16

TEDS EEPROM memory address / offset. Valid values: 0x00-0x1F

##### cnt

Variable Type      ViUInt16

Number of bytes to read from TEDS EEPROM. Valid values: 0-32

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.42 bu3416\_readTEDS\_OTP\_ROM

```
ViStatus bu3416_readTEDS_OTP_ROM (ViSession instrumentHandle, ViChar *buf,
                                   ViUInt16 addr, ViUInt16 cnt);
```

### Purpose

Function reads "cnt" bytes from TEDS OTP ROM starting at "addr" address.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### buf

Variable Type      ViChar (passed by reference)

Buffer for result.

#### addr

Variable Type      ViUInt16

TEDS ROM memory address / offset. Valid values: 0x00-0x07

#### cnt

Variable Type      ViUInt16

Number of bytes to read from TEDS ROM. Valid values: 0-8

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.43 bu3416\_readTEDS\_ROM

```
ViStatus bu3416_readTEDS_ROM (ViSession instrumentHandle, ViChar *buf,  
                             ViUInt16 cnt);
```

#### Purpose

Function reads "cnt" bytes from TEDS 1-Wire ROM.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### buf

Variable Type      ViChar (passed by reference)

Buffer for result.

##### cnt

Variable Type      ViUInt16

Number of bytes to read from TEDS 1-Wire ROM. Valid values: 0-8

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.44 bu3416\_reset

```
ViStatus bu3416_reset (ViSession instrumentHandle);
```

#### Purpose

This function resets the function card to its power-on state.

#### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.45 bu3416\_resetDAQ

```
ViStatus bu3416_resetDAQ (ViSession instrumentHandle);
```

#### Purpose

This function resets the Data Acquisition State Machine. The reset doesn't change contents of the registers.

#### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.46 bu3416\_resetFIFO

```
ViStatus bu3416_resetFIFO (ViSession instrumentHandle);
```

### Purpose

This function resets the FIFO pointers. All configuration remains unchanged.

### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.47 bu3416\_resetI2C

```
ViStatus bu3416_resetI2C (ViSession instrumentHandle);
```

#### Parameter List

instrumentHandle

Variable Type          ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.48 bu3416\_resizeMultBuf

```
ViStatus bu3416_resizeMultBuf (ViSession instrumentHandle, ViInt32 newSize);
```

### Purpose

This function re-allocates new buffer in 3150 Motherboard on-board DRAM memory for each Function Card in the Group.

### Parameter List

instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

newSize

Variable Type      ViInt32

Specifies the new size of the DRAM buffer allocated for each Function Card in the Group. The maximum size depends on the amount of DRAM available and number 3416 Function Cards fitted to each 3180 module.

Default size of the DRAM buffer is 0x20000 samples

### Return Value

If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return a warning or error code. Passing the status code to the function "bu3416\_error\_message" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "bu3416\_error\_message" will handle all three types of warning/error codes by passing them to the appropriate function if necessary ("bu3100\_error\_message" or "viStatusDesc") to return the correct warning/error message:

#### VISA Warnings/Errors:

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

#### BU3100 Warnings/Errors:

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range 0x3FFC0800 to 0x3FFC0900 and errors in the range 0xBFFC0800 to 0xBFFC0900. They are defined in the include file bu3100.h.

#### BU3416 Warnings/Errors:

Warning codes returned by the 3416 driver functions will be in the range 0x3FFC900 to 0x3FFC0FFF and errors codes in the range 0xBFFC0900 to 0xBFFC0FFF. They are defined in the include file bu3416.h.

### C.5.49 bu3416\_revision\_query

```
ViStatus bu3416_revision_query (ViSession instrumentHandle,  
                               ViChar driverRevision[],  
                               ViChar instrumentFirmwareRevision[]);
```

#### Purpose

This function returns the driver revision.

However, because the instrument revision query function is not supported this function always returns the VI\_WARN\_NSUP\_REV\_QUERY warning.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### driverRevision

Variable Type      ViChar[]

Returns the Instrument Driver revision.

##### instrumentFirmwareRevision

Variable Type      ViChar[]

Because the instrument revision query function is not supported this control always returns the message "Not Available".

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.50 bu3416\_self\_test

```
ViStatus bu3416_self_test (ViSession instrumentHandle, ViInt16 *testResult,  
                          ViChar testMessage[]);
```

#### Purpose

This function performs a self-test on the instrument.

\*\* NOTE: Self\_test is not supported by the hardware.

#### Parameter List

##### instrumentHandle

Variable Type          ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### testResult

Variable Type          ViInt16 (passed by reference)

Returns the result of the self test.

Valid Range:

0 - no error (test passed)

1 - test failed

##### testMessage

Variable Type          ViChar[]

Returns description of result of self-test.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.51 bu3416\_setAcquisitionMode

```
ViStatus bu3416_setAcquisitionMode (ViSession instrumentHandle,
                                     ViInt16 channelMask, ViReal64 sampleRateHz,
                                     ViInt32 scansToCollect, ViInt16 startMode,
                                     ViInt16 stopOnError);
```

#### Purpose

This function sets the Data Acquisition operational mode.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### channelMask

Variable Type      ViInt16

Selects the channels from which the data will be taken.

bit 0 corresponds to the Channel 1

...

bit 15 corresponds to the Channel 16

"1" written in the appropriate bit means that the channel will be included in the data acquisition.

##### sampleRateHz

Variable Type      ViReal64

This parameter specifies the sample rate (in Hertz) for data acquisition process. Possible values are from 1.0 (1Hz) to 10000.0 (10kHz)

##### scansToCollect

Variable Type      ViInt32

This parameter specifies total number of scans to collect by data acquisition. If this parameter contains '0', the data acquisition will run in unlimited (continuous) mode.

##### startMode

Variable Type      ViInt16

This Parameter specifies the Start Mode of Data Acquisition.

Possible values are:

bu3416_DA_START_IMM	0	Data Acquisition starts immediately after synchronization is done (Default);
bu3416_DA_START_TRIG	1	Data Acquisition starts when Input Trigger goes active;

**stopOnError**

Variable Type            ViInt16

This parameter specifies what kind of error will break data acquisition (DA).

Possible values are:

bu3416_DA_STOP_ERR_OFF	0	Errors don't stop DA;
bu3416_DA_STOP_ERR_EXOFR	1	Any error excluding OUTFRANGE stops DA;
bu3416_DA_STOP_ERR_ANY	2	Any error stop DA;

**Return Value**

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.52 bu3416\_setADCMode

```
ViStatus bu3416_setADCMode (ViSession instrumentHandle, ViInt16 ADCClockSource,
                             ViInt16 PLLClockSource, ViInt16 decimation);
```

### Purpose

This function sets the ADC operational mode.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### ADCClockSource

Variable Type      ViInt16

This parameter specifies the source for the ADC Clock.

Possible values are:

bu3416_ADC_CLK_DDS	0	Clock from DDS (Default);
bu3416_ADC_CLK_CCLK	1	Clock from CCLK (Common Clock);
bu3416_ADC_CLK_MBA	2	Clock from MB Input Trigger A;
bu3416_ADC_CLK_FP	3	Clock from FP Input Trigger 1;

#### PLLClockSource

Variable Type      ViInt16

This parameter specifies the source for the PLL circuitry.

Possible values are:

bu3416_PLL_CLK_OSC	0	Clock from on-board oscillator;
bu3416_PLL_CLK_CCLK	1	Clock from CCLK (Common clock);
bu3416_PLL_CLK_MBA	2	Clock from MB Input Trigger A;
bu3416_PLL_CLK_FP	3	Clock from FP Input Trigger 1;

#### decimation

Variable Type      ViInt16

This parameter specifies the decimation factor applied at ADC output data.

Possible values are:

bu3416_DECIM_OFF	0	Decimation is off;
bu3416_DECIM_10	1	Decimation is 10;
bu3416_DECIM_100	2	Decimation is 100;
bu3416_DECIM_1000	3	Decimation is 1000;

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

### C.5.53 bu3416\_setBufferSize

```
ViStatus bu3416_setBufferSize (ViSession instrumentHandle, ViInt32 newSize);
```

#### Purpose

This function re-allocates new buffer in ProDAQ Motherboard on-board DRAM memory.

#### Parameter List

instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

newSize

Variable Type      ViInt32

Specifies the new size of the DRAM buffer allocated for the Function Card.

Default size of the DRAM buffer depends on the type of the Function Card Carrier.

#### Return Value

If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return a warning or error code. Passing the status code to the function "bu3416\_error\_message" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "bu3416\_error\_message" will handle all three types of warning/error codes by passing them to the appropriate function if necessary ("bu3100\_error\_message" or "viStatusDesc") to return the correct warning/error message:

##### VISA Warnings/Errors:

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

##### BU3100 Warnings/Errors:

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range 0x3FFC0800 to 0x3FFC0900 and errors in the range 0xBFFC0800 to 0xBFFC0900. They are defined in the include file bu3100.h.

##### BU3416 Warnings/Errors:

Warning codes returned by the 3416 driver functions will be in the range 0x3FFC900 to 0x3FFC0FFF and errors codes in the range 0xBFFC0900 to 0xBFFC0FFF. They are defined in the include file bu3416.h.

### C.5.54 bu3416\_setChanConfig

```
ViStatus bu3416_setChanConfig (ViSession instrumentHandle, ViInt16 channel,
                               ViInt16 source, ViInt16 gain);
```

#### Purpose

This function configures the specified channel or all 16 channels.

#### Parameter List

##### instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### channel

Variable	Type	ViInt16
----------	------	---------

This parameter specifies which channel will be configured.

Possible values are:

bu3416_CHAN_ALL	0	All 16 channels will be configured.
bu3416_CHAN_1	1	Channel 1
bu3416_CHAN_2	2	Channel 2
bu3416_CHAN_3	3	Channel 3
bu3416_CHAN_4	4	Channel 4
bu3416_CHAN_5	5	Channel 5
bu3416_CHAN_6	6	Channel 6
bu3416_CHAN_7	7	Channel 7
bu3416_CHAN_8	8	Channel 8
bu3416_CHAN_9	9	Channel 9
bu3416_CHAN_10	10	Channel 10
bu3416_CHAN_11	11	Channel 11
bu3416_CHAN_12	12	Channel 12
bu3416_CHAN_13	13	Channel 13
bu3416_CHAN_14	14	Channel 14
bu3416_CHAN_15	15	Channel 15
bu3416_CHAN_16	16	Channel 16

##### source

Variable	Type	ViInt16
----------	------	---------

This parameter specifies the source of the input signal.

The possible values are:

bu3416_CH_FP	0	Channel is connected to FP SCSI connector;
bu3416_CH_VREF	1	Channel is connected to Voltage Reference;

#### NOTE:

Only one channel can be connected to VREF in the same time.

Please do not use 'Channel' = bu3416\_CHAN\_ALL with  
'Source' =bu3416\_CH\_VREF.

**gain**

Variable Type            ViInt16

This parameter specifies the gain for the input channel;  
Possible values are:

bu3416_GAIN_1	1	Gain 1 (Default)
bu3416_GAIN_2	2	Gain 2
bu3416_GAIN_5	5	Gain 5
bu3416_GAIN_10	10	Gain 10
bu3416_GAIN_20	20	Gain 20
bu3416_GAIN_50	50	Gain 50
bu3416_GAIN_100	100	Gain 100
bu3416_GAIN_200	200	Gain 200
bu3416_GAIN_500	500	Gain 500
bu3416_GAIN_1000	1000	Gain 1000
bu3416_GAIN_2000	2000	Gain 2000

**Return Value**

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.55 bu3416\_setDAQMode

```
ViStatus bu3416_setDAQMode (ViSession instrumentHandle, ViInt16 boardMode,
                             ViInt16 startMode, ViInt16 stopMode,
                             ViInt16 stopOnError);
```

#### Purpose

This function sets the Data Acquisition operational mode.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### boardMode

Variable Type      ViInt16

This parameter specifies the Function Card operational mode.

Possible values are:

bu3416_FC_STALONE	0	The Board operates in a Stand-Alone mode (Default);
bu3416_FC_MASTER	1	The Board operates as a Master in Multi-Card operational mode;
bu3416_FC_SLAVE	2	The Board operates as a Slave in Multi-Card operational mode;

##### startMode

Variable Type      ViInt16

This Parameter specifies the Start Mode of Data Acquisition (DA).

Possible values are:

bu3416_DA_START_IMM	0	Data Acquisition starts immediately after synchronization is done (Default);
bu3416_DA_START_TRIG	1	Data Acquisition starts when Input Trigger goes active;

##### stopMode

Variable Type      ViInt16

This Parameter specifies the Stop Mode of Data Acquisition (DA).

Possible values are:

bu3416_DA_STOP_COUNT	0	Data Acquisition stops when the specified number of samples has been collected (Default);
bu3416_DA_STOP_TRIG	1	Data Acquisition stops when Input Trigger Stop Event happened;
bu3416_DA_STOP_GATE	2	DA stops when Trigger goes inactive;
bu3416_DA_STOP_UNLIM	3	Data Acquisition stops only when DAQ STOP command issued;

**stopOnError**

Variable Type            ViInt16

This parameter specifies what kind of error will break data acquisition (DA).

Possible values are:

bu3416_DA_STOP_ERR_OFF	0	Errors don't stop DA;
bu3416_DA_STOP_ERR_EXOFR	1	Any error excluding OUTFRANGE stops DA;
bu3416_DA_STOP_ERR_ANY	2	Any error stop DA;

**Return Value**

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.56 bu3416\_setDDSFreq

```
ViStatus bu3416_setDDSFreq (ViSession instrumentHandle, ViReal64 frequencyHz);
```

#### Purpose

This function sets the frequency (in Hz) of the DDS generator.

#### Parameter List

##### instrumentHandle

Variable Type	ViSession
---------------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### frequencyHz

Variable Type	ViReal64
---------------	----------

This parameter specifies the frequency (in Hz) of the DDS generator. Possible values are:

512000.0 - 5120000.0 (512kHz - 5.12MHz)

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.57 bu3416\_setFIFOConfig

```
ViStatus bu3416_setFIFOConfig (ViSession instrumentHandle, ViInt16 affThreshold);
```

#### Purpose

This function sets FIFO configuration.

#### Parameter List

instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

affThreshold

Variable Type      ViInt16

This parameter specifies the threshold level for Programmable Almost Full Flag (PAFF).

Possible values are:

2 to (FIFO\_SIZE-1) where FIFO\_SIZE is 8192 or 16384 depending on card version

#### NOTE:

The following table shows configuration for FIFO flags:

Number of samples in FIFO	Empty	Almost Full	Full
0	1	0	0
1 to (aftThreshold-1)	0	0	0
aftThreshold to (FIFO_SIZE-1)	0	1	0
FIFO_SIZE	0	0	1

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.58 bu3416\_setFPTrigPolarity

```
ViStatus bu3416_setFPTrigPolarity (ViSession instrumentHandle, ViInt16 polFPT1,
                                   ViInt16 polFPT2, ViInt16 polFPT3);
```

#### Purpose

This function sets active state levels for Front Panel Triggers.

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### polFPT1

Variable Type      ViInt16

This parameter specifies level of active state of Front Panel Trigger 1  
Possible values are:

bu3416_LOW	0	active state for FP trigger 1 is low (zero)
bu3416_HIGH	1	active state for FP trigger 1 is high (one)

##### polFPT2

Variable Type      ViInt16

This parameter specifies level of active state of Front Panel Trigger 2  
Possible values are:

bu3416_LOW	0	active state for FP trigger 2 is low (zero)
bu3416_HIGH	1	active state for FP trigger 2 is high (one)

##### polFPT3

Variable Type      ViInt16

This parameter specifies level of active state of Front Panel Trigger 3  
Possible values are:

bu3416_LOW	0	active state for FP trigger 3 is low (zero)
bu3416_HIGH	1	active state for FP trigger 3 is high (one)

## Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

### C.5.59 bu3416\_setITRConfig

```
ViStatus bu3416_setITRConfig (ViSession instrumentHandle, ViInt16 daTrigSource,
                             ViInt16 syncSource);
```

#### Purpose

This function configures the Input Trigger.

#### Parameter List

##### instrumentHandle

Variable Type	ViSession
---------------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### daTrigSource

Variable Type	ViInt16
---------------	---------

This parameter specifies what sources will be chosen for Data Acquisition (DA) Trigger. The parameter is a bit mask, so more than one source can be used at the same time using bitwise-OR of the following values:

bu3416_DA_TRIG_OFF	0x0000
bu3416_DA_TRIG_MBA	0x0001
bu3416_DA_TRIG_MBB	0x0002
bu3416_DA_TRIG_FP3	0x0004

#### Note:

This configuration is used if a board is configured as a STANDALONE or MASTER, otherwise are don't care.

##### syncSource

Variable Type	ViInt16
---------------	---------

This parameter specifies the source for the SYNC signal.

Possible values are:

bu3416_SYNC_MBB	0	Motherboard Input Trigger Stack B;
bu3416_SYNC_FP	1	Front Panel SYNC signal;

#### NOTE:

This configuration is don't care when the board is a MASTER or STANDALONE, it is only required for the SLAVES. Setting this parameter to bu3416\_SYNC\_FP will automatically change FP Trigger input 2 polarity to low level as active state.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

**VISA Errors:**

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

**bu3416 Driver Errors:**

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.60 bu3416\_setMultChanConfig

```
ViStatus bu3416_setMultChanConfig (ViSession instrumentHandle, ViInt16 channel,
                                   ViInt16 source, ViInt16 gain);
```

### Purpose

This function configures the specified channel or all channels for the Function Card Group.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

#### channel

Variable Type      ViInt16

This parameter specifies which channel will be configured.

Possible values are:

bu3416_CHAN_ALL	0	All channels for all Function Card will be configured.
	1	Channel 1 of Function Card 1
	2	Channel 2 of Function Card 1
	...	
	16	Channel 16 of Function Card 1
	17	Channel 1 of Function Card 2
	18	Channel 2 of Function Card 2
	...	
	32	Channel 16 of Function Card 2
	...	

#### source

Variable Type      ViInt16

This parameter specifies the source of the input signal. The possible values are:

bu3416_CH_FP	0	Channel is connected to FP SCSI connector;
bu3416_CH_VREF	1	Channel is connected to Voltage Reference;

#### gain

Variable Type      ViInt16

This parameter specifies the gain for the input channel; Possible values are:

bu3416_GAIN_1	1	Gain 1 (Default)
bu3416_GAIN_2	2	Gain 2
bu3416_GAIN_5	5	Gain 5

bu3416_GAIN_10	10	Gain 10
bu3416_GAIN_20	20	Gain 20
bu3416_GAIN_50	50	Gain 50
bu3416_GAIN_100	100	Gain 100
bu3416_GAIN_200	200	Gain 200
bu3416_GAIN_500	500	Gain 500
bu3416_GAIN_1000	1000	Gain 1000
bu3416_GAIN_1000	2000	Gain 2000

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

### C.5.61 bu3416\_setMultTrigConfig

```
ViStatus bu3416_setMultTrigConfig (ViSession instrumentHandle, ViInt16 source);
```

#### Purpose

This function configures the Trigger for Data Acquisition.

#### Parameter List

instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

source

Variable Type      ViInt16

This parameter specifies the source for the Data Acquisition (DA) Trigger signal. The Trigger can be applied only to the Master Function Card. All Slave Function Cards will be triggered by Master Function Card.

Possible values are:

bu3416_DA_TRIG_OFF	0x0000	
bu3416_DA_TRIG_MBA	0x0001	
bu3416_DA_TRIG_MBB	0x0002	
bu3416_DA_TRIG_FP3	0x0004	
bu3416_DA_TRIG_FP3_LOW	0x0008	use this setting with bu3416_DA_TRIG_FP3 to set trigger active state to low.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.62 bu3416\_setOTRConfig

```
ViStatus bu3416_setOTRConfig (ViSession instrumentHandle, ViInt16 sourceMBA,
                              ViInt16 sourceMBB, ViInt16 sourceFPOT1,
                              ViInt16 sourceFPOT2, ViInt16 sourceFPOT3,
                              ViInt16 sourceDE, ViInt16 sourceDI);
```

### Purpose

This function configures the Output Trigger.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### sourceMBA

Variable Type      ViInt16

This parameter specifies what source will be chosen for Motherboard Output Trigger Stack A (MBA).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_FPIT3_MBA	1	FP trigger input 3 is routed to MBA;
bu3416_RCLK_MBA	2	Ref. clock (2MHz) is routed to MBA;
bu3416_ADCC_MBA	3	ADC clock is is routed to MBA;
bu3416_DA_ST_MBA	4	DA progress signal is routed to MBA;
bu3416_OUTRANGE_MBA	5	Out of range signal is routed to MBA;
bu3416_FIFO_PAFF_MBA	6	FIFO PAF flag is routed to MBA;

#### sourceMBB

Variable Type      ViInt16

This parameter specifies what source will be chosen for Motherboard Output Trigger Stack B (MBB).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_FPIT2_MBB	1	FP trigger input 2 is routed to MBB;
bu3416_SYNC_MBB	2	SYNC is routed to MBB;
bu3416_SCANP_MBB	4	Pulse per Scan sig. is routed to MBB;
bu3416_OUTRANGE_MBB	5	Out of range signal is routed to MBB;
bu3416_FIFO_PAFF_MBB	6	FIFO PAF flag is routed to MBB;

#### sourceFPOT1

Variable Type      ViInt16

This parameter specifies what source will be chosen for Front Panel Output Trigger 1 (FPOT1).

Possible values are:

bu3416_OFF	0	Output disabled;
bu3416_MBA_FPOT1	1	MB Trigger input 'A' is routed to FPOT1;
bu3416_CCLK_FPOT1	2	Common Clock (CCLK) is routed to FPOT1;

```

bu3416_RCLK_FPOT1 3 Ref. clock (2MHz) is routed to FPOT1;
bu3416_ADCC_FPOT1 4 ADC clock is routed to FPOT1;
bu3416_DA_ST_FPOT1 6 DA progress signal is routed to FPOT1;
bu3416_SWA_FPOT1 7 Software assertion mode used to
                    generate trigger by bu3416_generateOTRI
                    function.

```

## sourceFPOT2

```
Variable Type      ViInt16
```

This parameter specifies what source will be chosen for Front Panel Output Trigger 2 (FPOT2).

Possible values are:

```

bu3416_OFF          0 Output disabled;
bu3416_MBB_FPOT2    1 MB Trigger input 'B' is routed to FPOT2;
bu3416_SYNC_FPOT2   2 SYNC is routed to FPOT2;
bu3416_OUTRANGE_FPOT2 5 Out of range signal is routed to FPOT2;
bu3416_SCANP_FPOT2  6 Pulse per Scan sig. is routed to FPOT2;
bu3416_SWA_FPOT2    7 Software assertion mode used to
                    generate trigger by bu3416_generateOTRI
                    function;

```

## sourceFPOT3

```
Variable Type      ViInt16
```

This parameter specifies what source will be chosen for Front Panel Output Trigger 3 (FPOT3).

Possible values are:

```

bu3416_OFF          0 Output disabled;
bu3416_MBA_FPOT3    1 MB Trigger input 'A' is routed to FPOT3;
bu3416_MBB_FPOT3    2 MB Trigger input 'B' is routed to FPOT3;
bu3416_SWA_FPOT3    7 Software assertion mode used to
                    generate trigger by bu3416_generateOTRI
                    function.

```

## sourceDE

```
Variable Type      ViInt16
```

This parameter specifies what source will be chosen for Direct Error (DE).

Possible values are:

```

bu3416_OFF          0 Output disabled;
bu3416_OUTRANGE_DE  1 Out of range signal is routed to DE;
bu3416_ANY_DE       2 Any error asserts DE line;

```

## sourceDI

```
Variable Type      ViInt16
```

This parameter specifies what source will be chosen for Direct Interrupt (DI).

Possible values are:

```

bu3416_OFF          0 Output disabled;
bu3416_FIFO_PAFF_DI 1 FIFO PAFF flag is routed to DI;
bu3416_OUTRANGE_DI  2 OUTRANGE signal is routed to DI;

```

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

### C.5.63 bu3416\_setPostScans

```
ViStatus bu3416_setPostScans (ViSession instrumentHandle, ViInt32 scans);
```

#### Purpose

This function sets the post-trigger number of scans to collect.

#### Parameter List

**instrumentHandle**

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

**scans**

Variable Type      ViInt32

This parameter sets the post-trigger number of scans to collect. Possible values are from 0 to 0xffffffff.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.64 bu3416\_setSampFreq

```
ViStatus bu3416_setSampFreq (ViSession instrumentHandle, ViReal64 frequencyHz);
```

#### Purpose

Sets sampling frequency.

#### Parameter List

##### instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### frequencyHz

Variable	Type	ViReal64
----------	------	----------

This parameter specifies the sampling frequency (in Hz). Possible values are:

1.0 - 10000.0 (1Hz - 10 kHz)

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.65 bu3416\_setTrigConfig

```
ViStatus bu3416_setTrigConfig (ViSession instrumentHandle, ViInt16 source);
```

#### Purpose

This function configures the Trigger for Data Acquisition.

#### Parameter List

##### instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### source

Variable	Type	ViInt16
----------	------	---------

This parameter specifies what sources will be chosen for Data Acquisition (DA) Trigger. The parameter is a bit mask, so more than one source can be used at the same time using bitwise-OR of the following values:

bu3416_DA_TRIG_OFF	0x0000	
bu3416_DA_TRIG_MBA	0x0001	
bu3416_DA_TRIG_MBB	0x0002	
bu3416_DA_TRIG_FP3	0x0004	
bu3416_DA_TRIG_FP3_LOW	0x0008	use this setting with bu3416_DA_TRIG_FP3 to set trigger active state to low.

#### Note:

This configuration is used if a board is configured as a STANDALONE or MASTER, otherwise don't care.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.66 bu3416\_startAcquisition

```
ViStatus bu3416_startAcquisition (ViSession instrumentHandle);
```

### Purpose

This function starts the continuous acquisition.

### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.67 bu3416\_startAcquisitionEx

```
ViStatus bu3416_startAcquisitionEx (ViSession instrumentHandle,
                                     ViInt32 threshold,
                                     bu3100_irqHandler_t callback,
                                     ViAddr parameter);
```

### Purpose

This function starts the continuous acquisition. It uses Interrupt Callback Routine to read out periodically the 3416 FIFO contents.

### Parameter List

#### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### threshold

Variable Type      ViInt32

The interrupt service routine will be called every time when the number of scans, specified by this parameter is collected in Function Card FIFO or Motherboard Circular Buffer (on 3180).

#### callback

Variable Type      bu3100\_irqHandler\_t

This interrupt service routine should readout data from function card.

#### parameter

Variable Type      ViAddr

Pointer to a parameter which will be transferred to the interrupt service routine.

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.68 bu3416\_startMultAcquisition

```
ViStatus bu3416_startMultAcquisition (ViSession instrumentHandle);
```

### Purpose

This function starts the Data Acquisition on the Group of Function Cards operating synchronously.

### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.69 bu3416\_startMultAcquisitionEx

```
ViStatus bu3416_startMultAcquisitionEx (ViSession instrumentHandle,
                                         ViInt32 threshold,
                                         bu3100_irqHandler_t callback,
                                         ViAddr parameter);
```

#### Purpose

This function starts the Data Acquisition Process on the Group of the Function Cards operating synchronously. It uses Interrupt Callback Routine to read out periodically the acquired data.

#### Parameter List

##### instrumentHandle

Variable Type            ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

##### threshold

Variable Type            ViInt32

The interrupt service routine will be called every time when the number of scans, specified by this parameter is collected in Function Card FIFO or Motherboard Circular Buffer (on 3180).

##### callback

Variable Type            bu3100\_irqHandler\_t

This interrupt service routine should readout data from function card.

##### parameter

Variable Type            ViAddr

Pointer to a parameter which will be transferred to the interrupt service routine.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.70 bu3416\_stopAcquisition

```
ViStatus bu3416_stopAcquisition (ViSession instrumentHandle);
```

### Purpose

Stops the Data Acquisition process.

### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

### Return Value

If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return a warning or error code. Passing the status code to the function "bu3416\_error\_message" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "bu3416\_error\_message" will handle all three types of warning/error codes by passing them to the appropriate function if necessary ("bu3100\_error\_message" or "viStatusDesc") to return the correct warning/error message:

#### VISA Warnings/Errors:

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

#### BU3100 Warnings/Errors:

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range 0x3FFC0800 to 0x3FFC0900 and errors in the range 0xBFFC0800 to 0xBFFC0900. They are defined in the include file bu3100.h.

#### BU3416 Warnings/Errors:

Warning codes returned by the 3416 driver functions will be in the range 0x3FFC900 to 0x3FFC0FFF and errors codes in the range 0xBFFC0900 to 0xBFFC0FFF. They are defined in the include file bu3416.h.

### C.5.71 bu3416\_stopDAQ

```
ViStatus bu3416_stopDAQ (ViSession instrumentHandle);
```

#### Purpose

This function issues the DAQ Stop command, which will terminate the Data Acquisition process.

#### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

## C.5.72 bu3416\_stopMultAcquisition

```
ViStatus bu3416_stopMultAcquisition (ViSession instrumentHandle);
```

### Purpose

Stops the Data Acquisition process running on the Group of the Function Cards operating synchronously.

### Parameter List

instrumentHandle

Variable	Type	ViSession
----------	------	-----------

The Instrument Handle is used to identify the unique session or communication channel between the driver and the Group of the Function Cards operating synchronously.

This Handle can be obtained only from initialization of Function Card Group by using bu3416\_multInit() function, not from initialization of individual Function Cards (bu3416\_init() or bu3416\_paramInit()).

### Return Value

If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return a warning or error code. Passing the status code to the function "bu3416\_error\_message" will return a string describing the warning or error.

A driver function can return three different types of warnings or errors. The function "bu3416\_error\_message" will handle all three types of warning/error codes by passing them to the appropriate function if necessary ("bu3100\_error\_message" or "viStatusDesc") to return the correct warning/error message:

#### VISA Warnings/Errors:

See section 3.3 of the VPP 4.3.2 document for a complete list of VISA status codes and their values. The VPP 4.3 document contains detailed descriptions of all VISA functions and the status codes returned by each of them.

#### BU3100 Warnings/Errors:

These are warning or error codes returned by the common motherboard interface library, which is used by the 3416 driver to access a ProDAQ motherboard. Warnings returned by the library will be in the range 0x3FFC0800 to 0x3FFC0900 and errors in the range 0xBFFC0800 to 0xBFFC0900. They are defined in the include file bu3100.h.

#### BU3416 Warnings/Errors:

Warning codes returned by the 3416 driver functions will be in the range 0x3FFC900 to 0x3FFC0FFF and errors codes in the range 0xBFFC0900 to 0xBFFC0FFF. They are defined in the include file bu3416.h.

### C.5.73 bu3416\_storeCalibData

```
ViStatus bu3416_storeCalibData (ViSession instrumentHandle, ViInt16 channel,
                                ViUInt32 offset, ViUInt32 gain);
```

#### Purpose

This function stores the calibration data into on-board EEPROM. The data should be acquired before using bu3416\_calibrate() function. Once data is stored into EEPROM it will be downloaded at every power-up automatically to perform analog data correction.

#### Parameter List

##### instrumentHandle

Variable Type          ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### channel

Variable Type          ViInt16

This parameter specifies for which channel the calibration data will be stored.

Possible values are:

bu3416_CHAN_1	1	Channel 1
bu3416_CHAN_2	2	Channel 2
bu3416_CHAN_3	3	Channel 3
bu3416_CHAN_4	4	Channel 4
bu3416_CHAN_5	5	Channel 5
bu3416_CHAN_6	6	Channel 6
bu3416_CHAN_7	7	Channel 7
bu3416_CHAN_8	8	Channel 8
bu3416_CHAN_9	9	Channel 9
bu3416_CHAN_10	10	Channel 10
bu3416_CHAN_11	11	Channel 11
bu3416_CHAN_12	12	Channel 12
bu3416_CHAN_13	13	Channel 13
bu3416_CHAN_14	14	Channel 14
bu3416_CHAN_15	15	Channel 15
bu3416_CHAN_16	16	Channel 16

##### offset

Variable Type          ViUInt32

This parameter specifies the calibration coefficient (Offset) which will be stored into on-board EEPROM.

##### gain

Variable Type          ViUInt32

This parameter specifies the calibration coefficient (Gain) which will be stored into on-board EEPROM.

## Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function `"bu3416_error_message"`, will return a string describing the error.

### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between `0xBFFC0900` and `0xBFFC0FFF`.

### C.5.74 bu3416\_writeReadI2C

```
ViStatus bu3416_writeReadI2C (ViSession instrumentHandle, ViInt32 address,
                              ViInt32 nWrite, ViChar dataWrite[], ViInt32 nRead,
                              ViChar dataRead[]);
```

#### Parameter List

##### instrumentHandle

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument. If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

##### address

Variable Type      ViInt32

This control contains the address of the I2C device.

##### nWrite

Variable Type      ViInt32

This control specifies the number of bytes to be written to the I2C device. This number should not exceed bu3416\_PCA\_COUNT\_MAX-1.

##### dataWrite

Variable Type      ViChar[]

This buffer contains the data to be written to the I2C device.

##### nRead

Variable Type      ViInt32

This control specifies the number of bytes to read from the I2C device. This number should not exceed bu3416\_PCA\_COUNT\_MAX.

##### dataRead

Variable Type      ViChar[]

On successful call this buffer will contain the data read from the I2C device. The buffer should be allocated prior to the function call with appropriate size to hold all read data.

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of VI\_SUCCESS, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

##### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

##### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

### C.5.75 bu3416\_writeTEDS\_EEPROM

```
ViStatus bu3416_writeTEDS_EEPROM (ViSession vi, ViBuf buf, ViUInt16 addr,  
                                   ViUInt16 cnt);
```

#### Purpose

Write "cnt" bytes into TEDS 256-bit EEPROM starting at "addr" address.

#### Parameter List

vi

Variable Type      ViSession

The Instrument Handle is used to identify the unique session or communication channel between the driver and the instrument.

If more than one instrument of the same model type is used, this Handle will be used to differentiate between them.

buf

Variable Type      ViBuf

Buffer with data to write.

addr

Variable Type      ViUInt16

TEDS EEPROM memory address / offset. Valid values: 0x00-0x1F

cnt

Variable Type      ViUInt16

Number of bytes to write into TEDS EEPROM. Valid values: 0-32

#### Return Value

Displays the return status of the function call. If the function was successful, it will return a status of `VI_SUCCESS`, otherwise it will return an error code. Passing the error code into the function "bu3416\_error\_message", will return a string describing the error.

#### VISA Errors:

See section 3.3 of the VPP 4.2.2 document for a complete list of the WIN Framework error codes and their values. Appendix B of VPP 4.2 contains descriptions and a breakdown of the error codes returned by each of the VISA functions.

#### bu3416 Driver Errors:

Errors returned from the bu3416 drivers will be between 0xBFFC0900 and 0xBFFC0FFF.

This page was intentionally left blank.

**Bustec Production, Ltd.**  
**Bustec House, Shannon Business Park,**  
**Shannon, Co. Clare, Ireland**  
**Tel: +353 (0) 61 707100, FAX: +353 (0) 61 707106**

**Bustec, Inc.**  
**34428 Yucaipa Blvd, Yucaipa, CA 92399, U.S.A**  
**Tel. +1 909 797 0484, Fax: +1 760 751 1284**

