United Electronic Industries®

The High-Performance Alternative

DNx-MUX-414 / DNR-MUX-418 — User Manual

14 channel 1x3 multiplexer / switch interface board
for the PowerDNA Cube and RACK chassis
and
18 channel 1x3 multiplexer / switch interface board
for the RACK chassis

May 2019

PN Man-DNx-MUX-414-418

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue
Walpole, MA 02081
U.S.A.

For a list of our distributors and partners in the US and around the world, please contact a member of our support team:

Support:

Telephone:	(508) 921-4600
Fax:	(508) 668-2350

Also see the FAQs and online “Live Help” feature on our web site.

Internet Support:

Support:	support@ueidaq.com
Website:	www.ueidaq.com
FTP site:	ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronic Industries Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Specifications in this document are subject to change without notice. Check with UEI for current status.

Table of Contents

List of Figures

Chapter 1 Introduction

This document outlines the feature set and use of the DNx-MUX-414 and DNR-MUX-418 boards.

MUX-414 / MUX-418 boards are 1x3 multiplexer / cross point switch interface modules, for interfacing with switching and digital control applications.

NOTE: The DNR-MUX-418 is designed for use with UEI's RACKtangle chassis and is not available for FLATRACK or Cube chassis.
The DNx-MUX-414 boards are designed for use with all UEI chassis.

This chapter includes the following sections:

- Organization of Manual (Section 1.1)
- Manual Conventions (Section 1.2)
- MUX-414 / MUX-418 Board Overview (Section 1.3)
- Features (Section 1.4)
- Indicators (Section 1.5)
- Specification (Section 1.6)
- Device Architecture (Section 1.7)
- Connectors and Wiring (Pinout) (Section 1.8)

1.1 Organization of Manual

This *MUX-414-418 User Manual* is organized as follows:

- **Introduction**
  This chapter provides an overview of DNx-MUX-414 and DNR-MUX-418 1x3 multiplexer / cross point switch board features, device architecture, connectivity, and logic.
- **Programming with the High-Level API**
  This chapter provides an overview of how to create a session, configure the session, and interpret results on the MUX-414 / MUX-418 series boards for programming with the high-level Framework.
- **Programming with the Low-Level API**
  This chapter provides an overview of low-level API commands for configuring and using the MUX-414 / MUX-418.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for use with the DNx-MUX-414 and DNR-MUX-418 boards.
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

1.2 Manual Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:

Tips are designed to highlight quick ways to get the job done or to reveal good ideas you might not discover on your own.

NOTE: Notes alert you to important information.

CAUTION! *Caution advises you of precautions to take to avoid injury, data loss, and damage to your boards or a system crash.*

Text formatted in **bold** typeface generally represents text that should be entered verbatim. For instance, it can represent a command, as in the following example: “You can instruct users how to run setup using a command such as **setup.exe**.”

Bold typeface will also represent field or button names, as in “Click **Scan Network**.”

Text formatted in `fixed` typeface generally represents source code or other text that should be entered verbatim into the source code, initialization, or other file.

Before you begin:

Before plugging any I/O connector into the Cube or RACK chassis, be sure to remove power from all field wiring. Failure to do so may cause severe damage to the equipment.

No HOT SWAP

Always turn POWER OFF before performing maintenance on a UEI system. Failure to observe this warning may result in damage to the equipment and possible injury to personnel.

Usage of Terms

Throughout this manual, the term “Cube” refers to either a PowerDNA Cube product or to a PowerDNR RACKtangle^{TM} rack mounted system, whichever is applicable. The term DNR is a specific reference to the RACKtangle, DNA to the PowerDNA I/O Cube, and DNx to refer to both.

1.3 MUX-414 / MUX-418 Board Overview

The **DNx-MUX-414** boards provide 14 independent 1 x 3 switches for use with UEI Cube and RACK systems.

The DNA-MUX-414, DNR-MUX-414, and DNF-MUX-414 board versions are compatible with the UEI Cube, RACKtangle, and FLATRACK respectively. All board versions are functionally identical except for the mounting hardware. The DNA version is designed to stack in a Cube chassis. The DNR/F versions are designed to plug into the backplane of a RACK chassis.

The **DNR-MUX-418** boards provide 18 independent 1 x 3 switches for use with UEI RACKtangle and HalfRACK systems.

MUX-414 / MUX-418 boards are designed for use in a wide variety of switching and digital control applications.

1.3.1 Multiplexing Modes

Each channel provides a “common” terminal connected to three independent SPST (Form A) contacts. In a typical SIL application, this allows each flight computer signal to be connected to the actual trainer, a simulated device, a third test or error signal, or left open to simulate a broken wire or other open circuit condition.

1.3.2 Switch Conditions

Each channel is capable of switching voltages up to ±48 VDC, AC waveforms with peaks less than ±48 VDC or sinusoidal signals up to 34 V_{rms}.

Each channel is rated for continuous operation at 1 A DC or AC rms (at -40 to 85°C) with a switch resistance of less than 0.2 Ω (typical, not including external cables). All relays default to “open” on power up/reset.

Switching rates up to 250 Hz are supported, and all channels default to break-before-make relay operation.

1.3.3 Synchronization Input and Output Pins

MUX-414 / MUX-418 boards can synchronize relay switching via the sync in pin and sync out pins.

The sync out pin can be configured to change state when programming a relay state change, and the sync in pin can be configured to delay the switching of relays until the sync in pulses. Wiring the sync out pin of the final board configured with the sync in pins of multiple MUX-414 / MUX-418 boards allows hardware synchronization of relay switching on all outputs.

1.3.4 Diagnostic Capabilities

Users have the capability of reading onboard 2.5 V and 3.3 V supply levels, as well as onboard temperatures. Users can also read back the current state of each relay.

1.3.5 Isolation & Over-voltage Protection

Each board provides 350 VDC isolation between channels, as well as between the board, chassis and other installed I/O boards.

1.3.6 Support Accessories

All **DNx-MUX-414** connections are made through a 62-pin D connector. Users may also connect the boards to our DNA-STP-62 screw terminal panel via the DNA-CBL-62 cables. The cables are fully shielded and are available in 1, 3, 6, 10 and 20 foot lengths.

All **DNx-MUX-418** connections are made through a 78-pin D connector. Users may connect the boards to our DNA-STP-78 screw terminal panel via the DNA-CBL-78 cables. The cables are fully shielded and are available in 3 and 10 foot lengths.

1.3.7 Software Support

The MUX-414 / MUX-418 series includes software drivers supporting all popular operating systems including: Windows, Linux, QNX, VXWorks, RTX, and other popular Real-Time Operating Systems. Windows users may use the UEIDAQ Framework which provides a simple and complete software interface to all popular Windows programming languages and data acquisition and control applications (e.g. LabVIEW, MATLAB).

1.4 Features

The MUX-414 / MUX-418 board provides the following features:

- DNx-MUX-14: 14 fully isolated 1 x 3 channel multiplexer/switch
  DNR-MUX-18: 18 fully isolated 1 x 3 channel multiplexer/switch
- ±48 VDC / 34 V_{rms} (sinusoidal) maximum operating voltage
- 0.2 Ohm resistance (not including cabling)
- 1 A continuous load current rating (at -40 to 85°C)
- 3 A surge current (<100 mS)
- 250 Hz update rate
- All switches Normally Off power up / reboot state
- 350 V_{AC} isolation

1.5 Indicators

The MUX-414 / MUX-418 indicators are described in **Table 1-1** and illustrated in **Figure 1-1**.

Table 1-1 MUX-414 / MUX-418 Indicators

LED Name	Description
RDY	Indicates board is powered up and operational
STS	Indicates which mode the board is running in: • **OFF**: Configuration mode (e.g., configuring channels, running in point-by-point mode) • **ON**: Operation mode

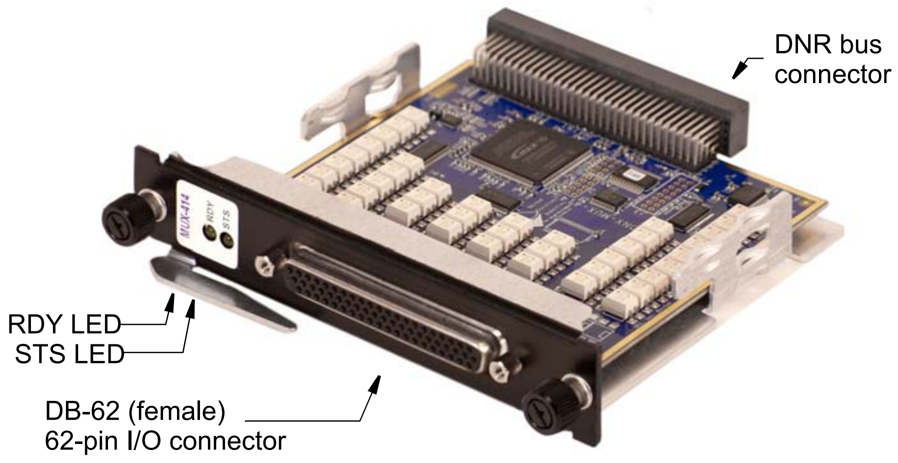


Figure 1-1. Photo of MUX-414 Multiplexer Board

1.6 Specification

The technical specifications for the MUX-414 are listed in **Table 1-2**, and technical specifications for the MUX-418 are listed in **Table 1-3**.

Table 1-2. DNx-MUX-414 Technical Specifications

Output configuration	14 independent 1 x 3 switches
Output specifications	
Rated Load (continuous)	1 A (-40 to +85°C) 2 A (-40 to +25°C)
Rated Load (peak)	3 A < 0.1 second
Max Operating Voltage	48 VDC, 48 V peak in AC waveforms, 34 Vrms (sinusoidal signals)
Absolute Max Voltage	55 VDC
Contact type	Solid State
Contact ON impedance	0.2 Ohm typical, 0.25 Ohm max (at the I/O connector)
Contact OFF impedance	>100 MOhm
Off Leakage Current	< 5 nA typical, <3 µA max over full temp range
Max update rate	250 Hz (including break-before-make timing)
Turn-Off Time	<0.2 mS typical (1 mS max)
Turn-On Time	< 0.45 mS typical (2 mS max)
Power up / reboot state	All Switches Off
Sync in/out specifications	
Sync in High Voltage	2.8 V min
Sync in Low Voltage	1.0 V max
Sync out High Voltage	3.55 V min / 4.0 V max @ 3 mA
Sync out Low Voltage	0.4 V max @ 3 mA
Power dissipation	< 5 W
Isolation	350 Vrms
Isolation resistance	>1 GOhm
Operating Temp. Range	Tested -40 to +85 °C
Operating Humidity	95%, non-condensing
Vibration *IEC 60068-2-6* *IEC 60068-2-64*	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500 Hz, broad-band random
Shock *IEC 60068-2-27*	50 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations

Table 1-3. DNR-MUX-418 Technical Specifications

Output configuration	18 independent 1 x 3 switches
Output specifications	
Rated Load (continuous)	1A (-40 to +85°C) 2A (-40 to +25°C)
Rated Load (peak)	3 A < 0.1 second
Max Operating Voltage	48 VDC, 48V peak in AC waveforms, 34 Vrms (sinusoidal signals)
Absolute Max Voltage	55 VDC
Contact type	Solid State
Contact ON impedance	0.2 Ohm typical, 0.25 Ohm max (at the I/O connector)
Contact OFF impedance	>100 MOhm
Off Leakage Current	5 nA typical, 3 µA max over full temp range
Max update rate	250 Hz (including break-before-make timing)
Turn-Off Time	<0.2 mS typical (1 mS max)
Turn-On Time	< 0.45 mS typical (2 mS max)
Power up / reboot state	All Switches Off
Sync in/out specifications	
Sync in High Voltage	2.8 V min
Sync in Low Voltage	1.0 V max
Sync out High Voltage	3.55 V min / 4.0 V max @ 3 mA
Sync out Low Voltage	0.4 V max @ 3 mA
Power dissipation	< 5 W not including output switches
Isolation	350 Vrms
Isolation resistance	>1 GOhm
Operating Temp. Range	Tested -40 to +85 °C
Operating Humidity	95%, non-condensing
Vibration *IEC 60068-2-6* *IEC 60068-2-64*	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500 Hz, broad-band random
Shock *IEC 60068-2-27*	50 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations
MTBF	>400,000 hours

1.7 Device Architecture

The MUX-414 / MUX-418 supports 14 and 18 independent 1x3 switches respectively.

A block diagram of the MUX-414 / MUX-418 is shown below:

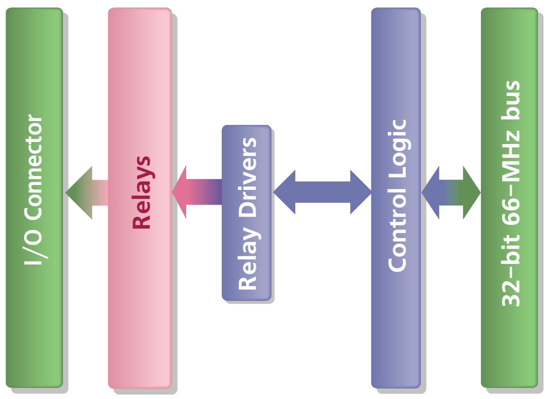


Figure 1-2. Architecture Block Diagram of DNA-MUX-414 / MUX-418

1.7.1 Input Circuitry

Each channel provides a common (COM) terminal connected to three independent SPST “A”/”B”/”C” (Form A) contacts. Refer to **Figure 1-3** below for a simplified block diagram of channel muxing.

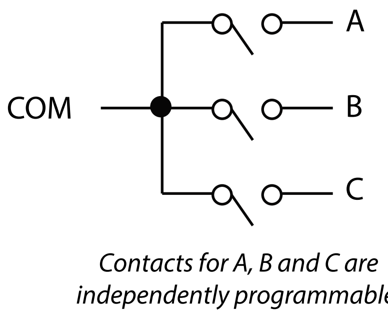


Figure 1-3. Single Channel Block Diagram for MUX-414 / MUX-418

1.7.2 Controlling Multiplexers

Channel relays are opened or closed based on a single write, written as a port. This means that each channel can have “A”, “B”, “C” or “none” closed independently, but when you write the configuration state, all 14 / 18 channels are written at once as a group.

By default, when users write new relay states, all closed relays open before new relay states are closed. This break-before-make functionality is user-configurable. Users can disable this feature, as well as program a delay in microseconds for how long the contacts will be open before new connections are made.

1.7.3 Synchro-nization I/O

Users have access to synchronization functionality via a sync in pin on the DB connector and sync out functionality via a sync out pin on the DB connector or via internal chassis bus lines.

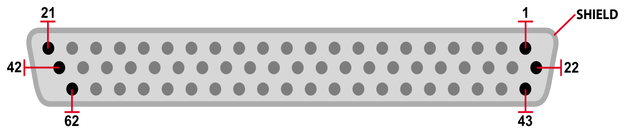
The sync out functionality can be enabled to change state when programming a relay state change. The pulse width and level of the pulse is user configurable.

The sync in functionality can be enabled, which delays relay state changes until a pulse is received on the sync in pin. Triggering on a level or edge change, as well as the polarity (high/rising or low/falling) is user configurable.

1.8 Connectors and Wiring (Pinout)

Figure 1-4 shows the pinout of the 62-pin female connector of the MUX-414.

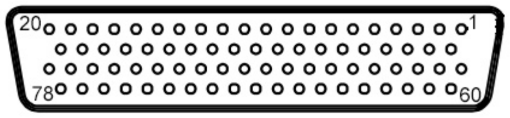
NOTE: *Pins marked 'rsvd' should be left open.



Pin	Signal	Pin	Signal	Pin	Signal
1	ch 11-A	22	ch 11 -com	43	ch 11 -C
2	ch 13 -B	23	ch 13 -C	44	ch 11 -B
3	ch 13 -A	24	ch 13 -com	45	ch 12 -A
4	ch 12 -B	25	ch 12 -C	46	ch 12 -com
5	sync out	26	sync gnd	47	sync +3.75V
6	ch 10 -B	27	ch 10 -C	48	sync in
7	ch 10 -A	28	ch 10 -com	49	ch 9 -A
8	ch 9 -B	29	ch 9 -C	50	ch 9 -com
9	ch 8 -B	30	ch 8 -C	51	ch 8 -A
10	ch 7 -B	31	ch 7 -C	52	ch 8 -com
11	ch 7 -A	32	ch 7 -com	53	ch 6 -A
12	ch 6 -B	33	ch 6 -C	54	ch 6 -com
13	ch 5 -B	34	ch 5 -C	55	ch 2 -C
14	ch 5 -A	35	ch 5 -com	56	ch 2 -B
15	ch 2 -A	36	ch 2 -com	57	ch 1 -A
16	ch 1 -B	37	ch 1 -C	58	ch 1 -com
17	ch 4 -B	38	ch 4 -C	59	ch 4 -A
18	ch 3 -B	39	ch 3 -C	60	ch 4 -com
19	ch 3 -A	40	ch 3 -com	61	ch 0 -A
20	ch 0 -B	41	ch 0 -C	62	ch 0 -com
21	rsvd*	42	rsvd*		

Figure 1-4. DB-62 I/O Connector Pinout for DNx-MUX-414

Figure 1-5 shows the pinout of the 78-pin female connector of the MUX-418:



Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	ch 17 -A	21	ch 11 -A	40	ch 11 -Com	60	ch 11 -C
2	ch 17 -B	22	ch 13 -B	41	ch 13 -C	61	ch 11 -B
3	ch 17 -C	23	ch 13 -A	42	ch 13 -Com	62	ch 12 -A
4	ch 17 -Com	24	ch 12 -B	43	ch 12 -C	63	ch 12 -Com
5	ch 16 -A	25	sync out	44	sync gnd	64	sync +3.75V
6	ch 16 -B	26	ch 10 -B	45	ch 10 -C	65	sync in
7	ch 16 -C	27	ch 10 -A	46	ch 10 -Com	66	ch 9 -A
8	ch 16 -Com	28	ch 9 -B	47	ch 9 -C	67	ch 9 -Com
9	ch 15 -A	29	ch 8 -B	48	ch 8 -C	68	ch 8 -A
10	ch 15 -B	30	ch 7 -B	49	ch 7 -C	69	ch 8 -Com
11	ch 15 -C	31	ch 7 -A	50	ch 7 -Com	70	ch 6 -A
12	ch 15 -Com	32	ch 6 -B	51	ch 6 -C	71	ch 6 -Com
13	ch 14 -A	33	ch 5 -B	52	ch 5 -C	72	ch 2 -C
14	ch 14 -B	34	ch 5 -A	53	ch 5 -Com	73	ch 2 -B
15	ch 14 -C	35	ch 2 -A	54	ch 2 -Com	74	ch 1 -A
16	ch 14 -Com	36	ch 1 -B	55	ch 1 -C	75	ch 1 -Com
17	ch 0 -Com	37	ch 4 -B	56	ch 4 -C	76	ch 4 -A
18	rsvd*	38	ch 3 -B	57	ch 3 -C	77	ch 4 -Com
19	rsvd*	39	ch 3 -A	58	ch 3 -Com	78	ch 0 -A
20	ch 0 -B			59	ch 0 -C		

Figure 1-5. DB-78 I/O Connector Pinout for DNR-MUX-418

Chapter 2 Programming with the High-Level API

This chapter provides the following information about using the UeiDaq Framework High-Level API to control the MUX-414 / MUX-418:

- Creating a Session (Section 2.1)
- Configuring Mux Port (Section 2.2)
- Configuring the Timing (Section 2.3)
- Configuring Sync Input / Sync Output (Section 2.4)
- Writing Data (Section 2.5)
- Monitoring Supply Voltage, Temperature & Status (Section 2.6)
- Monitoring Relay States & Status (Section 2.7)
- Cleaning-up the Session (Section 2.8)

UeiDaq Framework is object oriented and its objects can be manipulated in the same manner from different development environments, such as Visual C++, Visual Basic, or LabVIEW.

The following section focuses on the C++ API, but the concept is the same no matter what programming language you use.

Please refer to the “UeiDaq Framework User Manual” for more information on use of other programming languages.

2.1 Creating a Session

The Session object controls all operations on your PowerDNx device. The first task when programming using the high-level Framework is to create a session object:

```
// create a session object

CUeiSession muxSession;
```

2.2 Configuring Mux Port

You use a Mux session to configure MUX-414 / MUX-418 channels. All channels are configured as one port (`mux0`).

The MUX-414 / MUX-418 provides break-before-make functionality, which defaults to opening all relays (A, B, and C) for a channel when opening or closing any one relay (A, B, or C). You can disable this break-before-make default functionality if you wish when creating the session.

The following call configures the mux port of a DNx-MUX-414 / MUX-418 set as device 1 with break-before-make functionality enabled.

```
// Configure session to write to port 0 on device 1

muxSession.CreateMuxPort("pdna://192.168.100.2/Dev1/mux0", true);
```

`CreateMuxPort` configures the following parameters:

- **breakBeforeMake:** true to enable break-before-make, false otherwise (boolean).

2.2.1 Configuring Break-before-make or Port On Delay

Users can configure the delay that relays will stay open before closing (break-before-make delay), whether or not break-before-make functionality is enabled, and the delay before a new write will be accepted (port on delay).

By default, break-before-make circuitry is enabled, and the break delay is set to 250 µS. The default port on delay is 100 µS.

To enable/disable break-before-make and/or program delays, you first get a pointer to the mux port object of the MUX-414 / MUX-418 board:

```
CUeiMuxPort* pChan =
   dynamic_cast<CUeiMuxPort *>(muxSession.GetChannel(0));
```

You can enable or disable break-before-make when you first create the channel (see Section 2.2 above) or by using the `EnableBreakBeforeMake` method.

The following example shows you how to turn break-before-make off using the `EnableBreakBeforeMake` method:

```
pChan->EnableBreakBeforeMake(false);
```

You can change the amount of time the A, B, and C relays for a channel open before they close with the `SetOffDelay` method.

The following example sets the off delay to 350 µS:

```
pChan->SetOffDelay(350);
```

You can change the amount of time before a new write command will be accepted with the `SetOnDelay` method.

The following example sets the on delay to 150 µS:

```
pChan->SetOnDelay(150);
```

2.3 Configuring the Timing

You can configure the MUX-414 / MUX-418 to run in simple mode (point by point).

In simple mode, the delay between samples is determined by software on the host computer.

The following sample shows how to configure the simple mode. Please refer to the *UeiDaq Framework User Manual* to learn how to use other timing modes.

```
// configure timing for point-by-point (simple mode)

muxSession.ConfigureTimingForSimpleIO();
```

2.4 Configuring Sync Input / Sync Output

Users can optionally configure the MUX-414 / MUX-418 to synchronize relay switching to use an external hardware trigger via the sync in pins. Alternatively, you can configure sync out pins to pulse on a relay write.

As an example, you can use the sync out pin on one MUX-414 / MUX-418 to synchronize other MUX-414 / MUX-418 boards via their sync in pins.

To use this option, connect the sync out pin of the one board to the sync in pins of the others (and connect sync gnd of all boards).

In your application, program multiple MUX-414 / MUX-418 boards to turn channel relays on or off as needed, but configure the sync in flag to delay the actual hardware switching until those boards receive a synchronization pulse on their sync in pins. Then configure the MUX-414 / MUX-418 board that is controlling sync out to pulse its MUX-414 / MUX-418 sync output pin when its relay write is programmed.

2.4.1 Configure Sync Out

To program this, first get a pointer to the mux port object of the MUX-414 / MUX-418 board that you wish to configure the sync out pin for:

```
CUeiMuxPort* pChanOut =
   dynamic_cast<CUeiMuxPort *>(muxSessionOut.GetChannel(0));
```

Configure the sync output mode with the `SetSyncOutputMode` API.
The following configures the sync out pin to pulse low when the relays are ready (otherwise it is high).:

```
pChanOut->SetSyncOutputMode(UeiMuxSyncOutputRelaysReadyPulse1);
```

`SetSyncOutputMode` accepts the following options for configuration:

Option Name	Description
`UeiMuxSyncOutputLogic0`	drive constant logic '0'
`UeiMuxSyncOutputLogic1`	drive constant logic '1'
`UeiMuxSyncOutputSyncLine0 through UeiMuxSyncOutputSyncLine3`	driven by internal SYNC BUS[0] through [3]
`UeiMuxSyncOutputRelaysReadyPulse0`	pulse logic ‘1’ on "relays ready" for a pulse width duration of programmed with `SetSyncOutputPulseWidth` (100 µs default)
`UeiMuxSyncOutputRelaysReadyPulse1`	pulse logic ‘0’ on "relays ready" for a pulse width duration of programmed with `SetSyncOutputPulseWidth` (100 µs default)
`UeiMuxSyncOutputRelaysReadyLogic0`	drive logic ‘0’ while "relays ready" (UEI internal test mode)
`UeiMuxSyncOutputRelaysReadyLogic1`	drive logic ‘1’ while "relays ready" (UEI internal test mode)

You configure the sync out pulse width with the `SetSyncOutputPulseWidth` method. Note that the pulse width is only configurable when using the following sync modes:

- `UeiMuxSyncOutputRelaysReadyPulse0`
- `UeiMuxSyncOutputRelaysReadyPulse1`

In this example, we configure the sync out pin to pulse low for 1 ms when the relays are ready (otherwise it is high):

```
pChanOut->SetSyncOutputPulseWidth(1000);
```

`SetSyncOutputPulseWidth` accepts 1 µS, 10 µS, 100 µS, and 1000 µS options as the pulse width parameter.

2.4.2 Configure Sync In

To configure sync in, get the pointer(s) to the mux port object(s) of the MUX-414 / MUX-418 board(s) that you wish to configure the sync in pin(s) for.

To program this, first get a pointer to the mux port object of the MUX-414 / MUX-418 board that you wish to configure the sync in pin for:

```
CUeiMuxPort* pChan =
   dynamic_cast<CUeiMuxPort *>(muxSession.GetChannel(0));
```

Enable sync in:

```
pChan->EnableSyncInput(true);
```

Configure the sync in circuitry to trigger the relay writes on a rising edge:

```
pChan->SetSyncInputEdgePolarity(UeiDigitalEdgeRising);
pChan->EnableSyncInputEdgeMode(true);
```

Accepted values for `SetSyncInputEdgePolarity` are:

- `UeiDigitalEdgeRising`: Detect rising edge
- `UeiDigitalEdgeFalling`: Detect falling edge

2.5 Writing Data

Writing data is done using a `MuxWriter` object.

The following sample shows how to create a writer object and write multiplexer switches:

```
// create a writer and link it to the session’s stream

CUeiMuxWriter muxWriter(muxSession.GetDataStream());
```

You program the multiplexer channels using the `WriteMux` method, which gets passed how many channels you wish to program, which channels, and what states.

You can program one of 4 mux states for each MUX-414 / MUX-418 channel:

- 0: Open A, B, and C relays
- 1: Close the A relay, open B and C
- 2: Close the B relay, open A and C
- 3: Close the C relay, open A and B

The following example of writes 5 channels: 0, 2, 5, 11, and 12, which are reprogrammed with the following changes:

- channel 0: close relay A (1)
- channel 2: close relay B (2)
- channel 5: close relay C (3)
- channel 11: open all relays (0)
- channel 12 close relay C (3)

```
// declare arrays to hold channels and states (up to 14 channels)

int channels[14];
int relays[14]

// set up list of channels as described in example:

channels[0] = 0;
channels[1] = 2;
channels[2] = 5;
channels[3] = 11;
channels[4] = 12;


// set up list of relay states as described in example:

relays[0] = 1;
relays[1] = 2;
relays[2] = 3;
relays[3] = 0;
relays[4] = 3;


// write the 5 channels

writer.WriteMux(5, channels, relays);
```

2.6 Monitoring Supply Voltage, Temperature & Status

The MUX-414 / MUX-418 provides the diagnostic capability of monitoring onboard supply voltages and temperature using an onboard ADC. Additionally, you can also retrieve status.

To monitor diagnostic data, use the `ReadADC` method.

You can read up to 5 diagnostic channels:

- ADC channel 0: <Reserved>
- ADC channel 1: The 3.3 V supply in volts
- ADC channel 2: The 2.5 V supply in volts
- ADC channel 3: The temperature in degrees C
- ADC channel 4: The status (uint32, see description in Section 2.7)

The following code shows how to read the voltage and temperature:

```
// read all 5 values

double adcBuffer[5];
muxWriter.ReadADC(5, adcBuffer, NULL);

// print supplies and temp

for(int j = 1; j<4; j++)
{
     std::cout << " adc" << j << "= " << adcBuffer[j];
}
std::cout << std::endl;
```

NOTE: You can also retrieve status data with the `ReadStatus` method (see Section 2.7).

2.7 Monitoring Relay States & Status

You can monitor current relay states and status data with the `ReadStatus` method:

```
ReadStatus(uInt32 *relayA, uInt32 *relayB,
           uInt32 *relayC, uInt32 *status)
```

where

- `relayA`: bitwise representation of relay A states (1 is closed, 0 open)
- `relayB`: bitwise representation of relay B states
- `relayC`: bitwise representation of relay C states
- `status`: bitwise representation of board status

Status is returned as:

- `Bit 17`: '1' means data is ready from ADC
- `Bit 16`: '1' means overrun (write while busy)
- `Bit 3`: '1' means state machine is busy
- `Bit 2`: '1' means output state machine is waiting for the external SYNC/ready
- `Bit 1`: '1' means relays are settled
- `Bit 0`: reports the logic state of the sync in pin
- all other bits are <Reserved>

The following code shows how to read the relay states and status:

```
// read all current relay state and status

uInt32 stRelayA, stRelayB, stRelayC, status;
muxWriter.ReadStatus(&stRelayA, &stRelayB, &stRelayC, &status);

// print results

std::cout << std::hex << " relayA=" << stRelayA <<
          " relayB=" << stRelayB <<
          " relayC=" << stRelayC <<
          " status=" << status << std::dec << std::endl << std::endl;
```

2.8 Cleaning-up the Session

The session object will clean itself up when it goes out of scope or when it is destroyed. To reuse the object with a different set of channels or parameters, you can manually clean up the session as follows:

```
// clean up the session

muxSession.CleanUp();
```

Chapter 3 Programming with the Low-Level API

This chapter provides the following information about programming the MUX-414 / MUX-418 using the low-level API:

- About the Low-level API (Section 3.1)
- Low-level Functions (Section 3.2)
- Low-level Programming Techniques (Section 3.3)
- Programming the MUX-414 / MUX-418 (Immediate Mode) (Section 3.4)
- Writing MUX Relays & Control Bits (Section 3.5)
- Reading Diagnostic Voltage, Temperature, and Status (Section 3.6)
- Reading Status and Relay States (Section 3.7)
- Configuring Sync and Delays (Section 3.8)
- Configuring Break-before-make Functionality (Section 3.9)

3.1 About the Low-level API

The low-level API provides direct access to the DAQBIOS protocol structure and registers in C. The low-level API is intended for speed-optimization, when programming unconventional functionality, or when programming under Linux or real-time operating systems.

When programming in Windows OS, however, we recommend that you use the UeiDaq high-level Framework API (see **Chapter 2**). The Framework extends the low-level API with additional functionality that makes programming easier and faster.

For additional information regarding low-level programming, refer to the "PowerDNA API Reference Manual" located in the following directories:

- On Linux systems:
  <PowerDNA-x.y.z>/docs
- On Windows systems:
  Start » All Programs » UEI » PowerDNA » Documentation

3.2 Low-level Functions

Table 3-1 provides a summary of MUX-414 / MUX-418-specific functions. All low-level functions are described in detail in the *PowerDNA API Reference Manual*.

Table 3-1 Summary of Low-level API Functions for DNx-MUX-414 / MUX-418

Function	Description
`DqAdv414Write`	Writes to output port(s) (mux control); also accepts OR'ed in optional control flags to disable break-before-make functionality and program sync in and sync out
`DqAdv414SetCfg`	Configures break-before-make delay characteristics and sync in pin and sync out pin characteristics
`DqAdv414ReadADC`	Allows users to read diagnostic internal voltage and board temperature
`DqAdv414ReadStatus`	Allows users to read back the position of relay A, B, and C and read board status

3.3 Low-level Programming Techniques

Application developers are encouraged to explore the existing source code examples when first programming the MUX-414 / MUX-418. Sample code provided with the installation is self-documented and serves as a good starting point.

Code examples are located in the following directories:

- On Linux systems: <PowerDNA-x.y.z>/src/DAQLib_Samples
- On Windows: *Start » All Programs » UEI » PowerDNA » Examples*

Sample code for data acquisition modes have the name of the mode and the name of the I/O boards being programmed embedded in the sample name. Note that immediate mode samples are named Sample<I/O board name>, (i.e., Sample414_418).

3.3.1 Data Collection Modes

The MUX-414 / MUX-418 supports the following acquisition mode:

- Immediate (point-to-point): Designed to provide easy access to a single I/O board at a non-deterministic pace. Acquires a single data point per channel. Runs at a maximum of 100 Hz.

API that implement data acquisition modes and additional mode descriptions are provided in the *PowerDNA API Reference Manual*.

3.4 Programming the MUX-414 / MUX-418 (Immediate Mode)

The following sections provide an overview of how to set up and use your MUX-414 / MUX-418 in Immediate Mode using the low-level API.

For best results, use this overview in conjunction with actual sample code, (i.e., Sample414_418). This overview does not address typical initialization or error handling. Refer to Section 3.3 for sample code location.

3.5 Writing MUX Relays & Control Bits

Set A, B, or C relays on or off for the MUX-414 / MUX-418 channels using the `DqAdv414Write` API:

```
DqAdv414Write(hd0, DEVN, rly_wr);
```

where

- `hd0` is the handle to the IOM
- `DEVN` is the MUX-414 / MUX-418 position in the chassis
- `rly_wr` is a data structure of type `DQ414W`.

The `DQ414W` data structure has elements for closing or opening the A, B, and/or C relays and programming sync in, sync out, and/or break-before-make configuration.

```
typedef struct {
   int32 rflags;
   int32 mux_select[2];
   int32 relay_select[3];
} DQ414W, *pDQ414W;
```

3.5.1 Programming Relays

You have two options of how to program the A, B, or C relays.

The typical application will use the `mux_select` array, which closes a single relay per channel. With this, you can connect COM to A, B, or C (or open all 3 for open circuit tests).

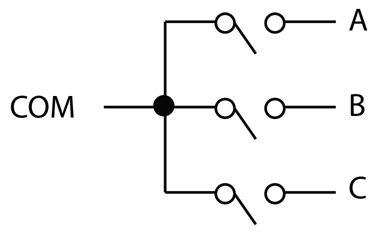


Figure 3-1. Block Diagram for MUX-414 / MUX-418

Alternatively, you can program more than one relay closed per channel using `relay_select`.

NOTE: You must use extreme caution if you choose to program using `relay_select. relay_select` allows users to close multiple mux relays per channel. This means if you have a source on A, B, and C, you will short your source equipment. This mode is meant for diagnostic purposes: for example, a sensor connected to COM, and listening devices on A, B, and/or C.

You set the `rflags` parameter to tell the firmware whether you are programming with `mux_select` or `relay_select`.

The `rflags` parameter must include one (and only one) of the `rflags` listed in **Table 3-2**.

Note that `rflags` can optionally have the following flags ORed in also:

- `DQ_MUX414_W_OPTION_SOUT:` Programs the sync out pin to pulse when `DqAdv414Write` executes (refer to Section 3.8)
- `DQ_MUX414_W_OPTION_SIN:` Uses sync in as a trigger to delay the relays from changing state when `DqAdv414Write` until the sync in pin transitions (refer to Section 3.8)
- `DQ_MUX414_W_OPTION_NO_BBM:` disables break-before-make circuitry (Section 3.9)

Table 3-2 Settings for rflags (mux_select or relay_select)

Name	Description
`DQ_MUX414_W_PORT0`	Program A, B, or C relay connect for channel 0 through 13; Allows write to `mux_select[0]` (414 or 418)
`DQ_MUX414_W_PORT1`	Program A, B, or C relay connect for channel 14 through 17; Allows write to `mux_select[1]` (418 only)
`DQ_MUX414_W_PORT10`	Program A, B, or C relay connect for channel 0 through 17; Allows write to `mux_select[1]` and `mux_select[0]`
`DQ_MUX414_W_RELAY_A`	Program A relays to connect or disconnect (all channels); Allows write to `relay_select[0]`
`DQ_MUX414_W_RELAY_B`	Program B relays to connect or disconnect (all channels); Allows write to `relay_select[1]`
`DQ_MUX414_W_RELAY_BA`	Program A & B relays to connect or disconnect (all channels); Allows write to `relay_select[1]` and `relay_select[0]`
`DQ_MUX414_W_RELAY_C`	Program C relays to connect or disconnect (all channels); Allows write to `relay_select[2]`
`DQ_MUX414_W_RELAY_CA`	Program C & A relays to connect or disconnect (all channels); Allows write to `relay_select[2]` and `relay_select[0]`
`DQ_MUX414_W_RELAY_CB`	Program C & B relays to connect or disconnect (all channels); Allows write to `relay_select[2]` and `relay_select[1]`
`DQ_MUX414_W_RELAY_CBA`	Program A, B & C relays to connect or disconnect (all channels); Allows write to `relay_select[2]` and `relay_select[1]` and `relay_select[0]`

3.5.1.1 Example of Using mux_select

When using `mux_select`, the following mux states for each channel are programmed to its corresponding 2-bits in `mux_select:`

- 1 to close A relay
- 2 to close B relay
- 3 to close C relay
- 0 to open all 3 relays

To program channel 0, you set bits 1 and 0 to 1,2,3 or 0 mux state;
to program channel 1, you set bits 3 and 2 to 1,2,3 or 0; etc.

As an example, to program

- channel 0 to close the B-relay (2),
- channel 2 to close the C-relay (3),
- channel 8 to close the A relay (1),
- channel 9 to close the C-relay (3) and
- all other channels up to channel 13 to open all the relays (0),

you set the `rflags` for using `mux_select[0]` and program the following:

```
pdata.rflags= DQ_MUX414_W_PORT0;
pdata.mux_select[0] = 0x00D0032;
DqAdv414Write(hd0, DEVN, pdata);
```

3.5.1.2 Example of Using relay_select

When using `relay_select`, you can close more than 1 relay per channel.

- `relay_select[0]` closes A relays.
- `relay_select[1]` closes B relays.
- `relay_select[2]` closes C relays.

NOTE: Each channel is programmed by setting or resetting its corresponding bit in the `relay_select` array, (e.g. the A relay for channel 0 corresponds to bit 0 in `relay_select[0]`, the B relay for channel 5 corresponds to bit 5 in `relay_select[1]`, etc.).
A ‘1’ corresponds to closing the relay, ‘0’ to opening.

Use caution if you choose to program using `relay_select.`
Using `relay_select` gives you the capability of closing more than one mux switch for a channel. This means if you have a source on A, B, and C, you could short your source equipment. This mode is meant for systems with a single source on COM, node A, node B, or node C.

Example: To program channel 0 through 13 to close the A relay, and channel 2 to close the C-relay, and all other channels to open all other relays, you set the `rflags` for using `relay_select`:

```
pdata.rflags= DQ_MUX414_W_RELAY_CBA;
pdata.relay_select[0] = 0x00003FFF;  //A
pdata.relay_select[1] = 0x00000000;  //B
pdata.relay_select[2] = 0x00000004;  //C
DqAdv414Write(hd0, DEVN, pdata);
```

3.6 Reading Diagnostic Voltage, Temperature, and Status

You can read diagnostic data for the MUX-414 / MUX-418 using the `DqAdv414ReadADC` API.

Data is returned as `DQ414ADC` data structure and includes onboard voltage supply readings, onboard temperature readings, and status:

```
typedef struct {
  double adc_in;    // Reserved
  double adc_3_3;   // monitor internal 3.3V supply
  double adc_2_5;   // monitor internal 2.5V supply
  double adc_deg_c; // adc temperature in degrees C
  uint32 status;    // status identical to .status
                    // returned by DqAdv414ReadStatus()
} DQ414ADC, *pDQ414ADC;
```

NOTE: Before reading data, call `DqAdv414ReadADC(hd, devn, NULL)` once to initialize the ADC.

Example:

```
// startup ADC reads
DqAdv414ReadADC(hd, DEVN, NULL);

// Get voltages and temperature along with status
DqAdv414ReadADC(hd, DEVN, &adc_reads);
         printf(" Internal 3.3V supply =  %lf\n", adc_reads.adc_3_3);
         printf(" Internal 2.5V supply =  %lf\n", adc_reads.adc_2_5);
         printf(" Internal temperature =  %lf\n", adc_reads.adc_deg_c);
         printf(" status               =  %x\n", adc_reads.status);
```

NOTE: Refer to `DqAdv414ReadStatus` API for `status` bit meanings.

3.7 Reading Status and Relay States

You can read board status and current A, B, and C relay states for the MUX-414 / MUX-418 using the `DqAdv414ReadStatus` API.

The `DQ414STATUS` data structure holds status data, as well as the current state the A, B, and C relays for each channel:

```
typedef struct {
   uint32 relay_a;
   uint32 relay_b;
   uint32 relay_c;
   uint32 status;
} DQ414STATUS, *pDQ414STATUS;
```

Relay position data is returned as a ‘1’ for closed and a ‘0’ for open.

Status is returned as:

- `Bit 17, DQ_MUX414_STS_ADCDR`: ‘1’ means data is ready from ADC
- `Bit 16, DQ_MUX414_STS_OVR`: ‘1’ means overrun (write while busy)
- `Bit 3, DQ_MUX414_STS_BUSY`: ‘1’ means state machine is busy
- `Bit 2, DQ_MUX414_STS_SYNCWAIT`: ‘1’ means output state machine is waiting for the external SYNC/ready
- `Bit 1, DQ_MUX414_STS_RDY`: ‘1’ means relays are settled
- `Bit 0, DQ_MUX414_STS_DI_STS`: reports the logic state of the sync in pin

For example:

```
// Get relay states and status
DqAdv414ReadStatus(hd, DEVN, &r_sts);
printf(" relays A = %x\n", r_sts.relay_a);
printf(" relays B = %x\n", r_sts.relay_b);
printf(" relays C = %x\n", r_sts.relay_c);
printf(" status = %x\n", r_sts.status);
```

3.8 Configuring Sync and Delays

The MUX-414 / MUX-418 provides a sync in and sync out pin.

The sync out pin can be configured to change state when calling the API to program relay state changes, and the sync in pin can be configured to delay switching relays until the sync in is triggered.

Users can program the sync out pin to pulse / change state when the `DQ_MUX414_W_OPTION_SOUT` flag is ORed into `rflags` with the relay flags when calling the `DqAdv414Write` API (see Section 3.5 for more information about the `DqAdv414Write` API).

Users can program using the sync in pin as a gating device when the `DQ_MUX414_W_OPTION_SIN` flag is ORed into `rflags` with the relay flags when calling the `DqAdv414Write` API (see Section 3.5 for more information about the `DqAdv414Write` API).

You can configure delays and sync in / sync out modes with the `DqAdv414Cfg` API.

```
typedef struct {
  uint32 on_delay;
  uint32 off_delay;
  uint32 di_mode;
  uint32 di_polarity;
  uint32 sync_out_pw;
  uint32 sync_out_mode;
} DQ414CFG, *pDQ414CFG;
```
:

Table 3-3 Settings for DQ414CFG Elements

Name	Description
`on_delay`	program the time before next command is accepted in 10uS units, range 1..256
`off_delay`	program the breaking time of break-before-make in 10uS units, range 1..256
`di_mode`	program the sync in pin operation mode, 0 = level / 1 = edge
`di_polarity`	program the polarity of sync in strobe: 0-falling (low) / 1-rising (high)
`sync_out_pw`	program the sync out pin pulse length for `sync_out_mode` 6 and 7 0 - 1uS; 1 - 10uS; 2 - 100uS; 3 - 1mS; 4..15 - reserved
`sync_out_mode`	program mode of operation for the sync out pin 0 - drive constant logic '0' 1 - drive constant logic '1' 2..5 - driven by internal SYNC BUS[0]..[3 6 - positive transitioning pulse on "relays ready" 7 - negative transitioning pulse on "relays ready" 8 - logic '1' level on "relays ready" (UEI test adapter expects this) 9 - logic '0' level on "relays ready"

3.8.1 Example of Using Sync In / Sync Out Handshaking

The following example programs relays on a MUX-414 board located in DEVN=0 to switch at the same time as relays on a MUX-414 board located in DENV=1.

3.8.1.1 Configure Sync In

Configure delays and pulse characteristics of the sync in pin for the first MUX-414 in DEVN0:

```
// fill a DQ414CFG struct
    r_cfg.on_delay      = 2;  // (20us) time before next command is accepted
    r_cfg.off_delay     = 2;  // (20uS) breaking time of break-before-make
    r_cfg.di_mode       = 0;  // sync in pin operation mode,  0 = level
    r_cfg.di_polarity   = 0;  // Polarity of sync_in strobe: 0=falling(low)
    r_cfg.sync_out_pw   = 0;  // not using, program 0
    r_cfg.sync_out_mode = 0;  // not using, program 0

// set configuration
DqAdv414SetCfg(hd, DEVN0, &r_cfg);
```

3.8.1.2 Configure Sync Out

Configure delays and pulse characteristics of the sync out pin for the second MUX-414 in DEVN1:

```
// fill a DQ414CFG struct
    r_cfg.on_delay      =  2;  // (20us)time before next command is accepted
    r_cfg.off_delay     =  25; // (250uS) breaking time of break-before-make
    r_cfg.di_mode       =  0;  // not using, program 0
    r_cfg.di_polarity   =  0;  // not using, program 0
    r_cfg.sync_out_pw   =  3;  // sync_out pin pulse length
                               //    for sync_out_modes 6 and 7
                               //    0 - 1uS, 1 - 10uS, 2 - 100uS, 3 - 1mS
    r_cfg.sync_out_mode =  7;  // sync_out pin mode of operation
                               // 0 - drive constant logic '0'
                               // 1 - drive constant logic '1'
                               // 2..5 - driven by internal SYNC BUS[0]..[3]
                               // 6 - '1' going pulse on "relays ready"
                               // 7 - '0' going pulse on "relays ready"
                               // 8 - logic '1' level on "relays ready"
                               // 9 - logic '0' level on "relays ready"
// set configuration
DqAdv414SetCfg(hd, DEVN1, &r_cfg);
```

3.8.1.3 Program Writing Relays Gated by Sync In

Program all B relays to close for DEVN0 and OR in `DQ_MUX414_W_OPTION_SIN` to gate the relays from actually switching until a sync in pulse is accepted:

```
pdata.rflags= DQ_MUX414_W_PORT0 | DQ_MUX414_W_OPTION_SIN;
pdata.mux_select[0] = 0xAAAAAAA;
   DqAdv414Write(hd0, DEVN0, pdata);
```

3.8.1.4 Program Writing Relays and Pulsing Sync Out

Program all A relays to open for DEVN1 and OR in `DQ_MUX414_W_OPTION_SOUT` to pulse when relays are written:

```
pdata.rflags= DQ_MUX414_W_PORT0| DQ_MUX414_W_OPTION_SOUT;
pdata.mux_select[0] = 0x5555555;
   DqAdv414Write(hd0, DEVN1, pdata);
```

3.9 Configuring Break-before-make Functionality

By default, when users program a relay state change for a channel, MUX-414 / MUX-418 break-before-make circuitry will open the A, B, and C relays for that channel before connecting.

Users can adjust the duration of time you break on a relay state change, as well as disable break-before-make functionality completely.

3.9.1 Changing Break Duration

To change the duration of the break time, you configure the `off_delay` element in the `DQ414CFG` structure using the `DqAdv414SetCfg` API.

```
typedef struct {
  uint32 on_delay;
  uint32 off_delay;
  uint32 di_mode;
  uint32 di_polarity;
  uint32 sync_out_pw;
  uint32 sync_out_mode;
} DQ414CFG, *pDQ414CFG;
```
:

Table 3-4 Setting for DQ414CFG Break-before-make Delay

Name	Description
`off_delay`	program the breaking time of break-before-make in 10uS units, range 1..256

For example, to set 1 ms as the amount of time you break connections for A, B, and C relays before you reconnect to a new configuration, you program the following:

```
// set delay in DQ414CFG struct
r_cfg.off_delay  =  100;  // 100*10us(1000uS) breaking time
                          //    of break-before-make

// set configuration
DqAdv414SetCfg(hd, DEVN1, &r_cfg);
```

3.9.2 Disabling Break-before-make

To disable the break-before-make functionality on MUX-414 / MUX-418 relay switches, you set the `DQ_MUX414_W_OPTION_NO_BBM` flag using the `DqAdv414Write` API.

For example, the following disables break-before-make circuitry for the following `DqAdv414Write`:

```
pdata.rflags=DQ_MUX414_W_PORT0 | DQ_MUX414_W_OPTION_NO_BBM;
pdata.mux_select[0] = 0x000D032;
DqAdv414Write(hd0, DEVN, pdata);
```

Appendix A

A. Accessories

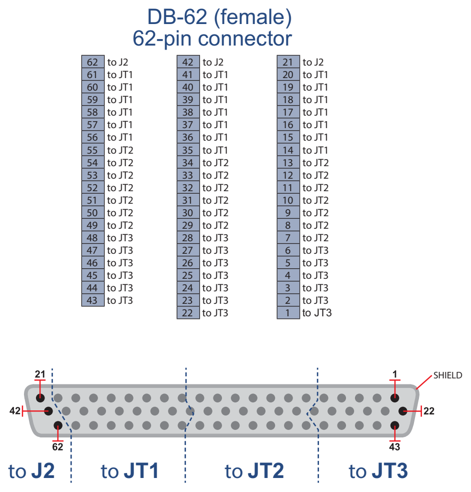
The following cables and STP boards are available for the MUX-414 / MUX-418 board.

DNA-CBL-62

This is a 62-conductor round shielded cable with 62-pin male D-sub connectors on both ends. It is made with round, heavy-shielded cable; 2.5 ft (75 cm) long, weight of 9.49 ounces or 269 grams; up to 10ft (305cm) and 20ft (610cm).

DNA-STP-62

The STP-62 is a Screw Terminal Panel with three 20-position terminal blocks (JT1, JT2, and JT3) plus one 3-position terminal block (J2). The dimensions of the STP-62 board are 4w x 3.8d x1.2h inch or 10.2 x 9.7 x 3 cm (with standoffs). The weight of the STP-62 board is 3.89 ounces or 110 grams.



DB-62 (female)
62-pin connector

62	to J2	42	to J2	21	to J2
61	to JT1	41	to JT1	20	to JT1
60	to JT1	40	to JT1	19	to JT1
59	to JT1	39	to JT1	18	to JT1
58	to JT1	38	to JT1	17	to JT1
57	to JT1	37	to JT1	16	to JT1
56	to JT1	36	to JT1	15	to JT1
55	to JT2	35	to JT1	14	to JT1
54	to JT2	34	to JT2	13	to JT2
53	to JT2	33	to JT2	12	to JT2
52	to JT2	32	to JT2	11	to JT2
51	to JT2	31	to JT2	10	to JT2
50	to JT2	30	to JT2	9	to JT2
49	to JT2	29	to JT2	8	to JT2
48	to JT3	28	to JT3	7	to JT2
47	to JT3	27	to JT3	6	to JT3
46	to JT3	26	to JT3	5	to JT3
45	to JT3	25	to JT3	4	to JT3
44	to JT3	24	to JT3	3	to JT3
43	to JT3	23	to JT3	2	to JT3
		22	to JT3	1	to JT3

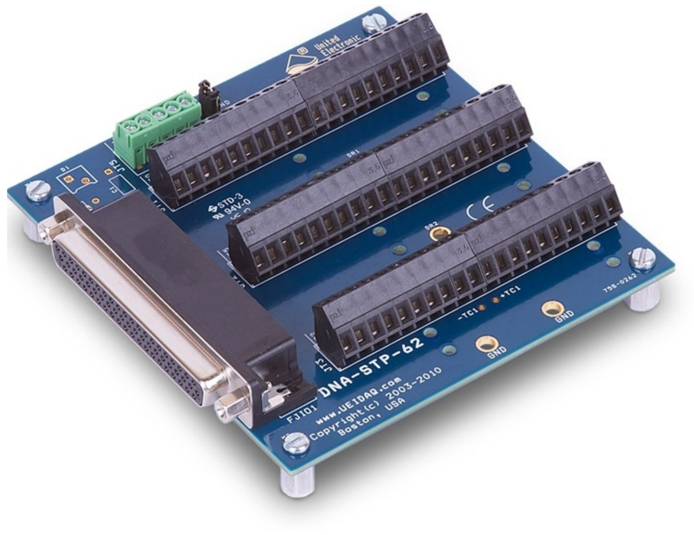


Figure A-1. Pinout and Photo of DNA-STP-62 Screw Terminal Panel

Index