

# Quarch Technology Ltd

## Gen5 OCP NIC x16 Breakers

### Technical Manual

For use with:

**QTL3095 – Gen5 EDSFF E3 x8 Breaker**

**QTL3096 – Gen5 EDSFF E3 x8 Breaker + Triggering**

**Using Quarch firmware version 5.000 and above**

Wednesday, 21 May 2025



## Change History

1.0	21/05/2025	Initial Release

## Contents

<b>Change History</b> .....	<b>2</b>
<b>Introduction</b> .....	<b>4</b>
<b>Technical specifications</b> .....	<b>5</b>
Power requirements .....	5
Switching characteristics.....	5
Mechanical characteristics .....	6
<b>Control Interfaces</b> .....	<b>7</b>
<b>Basic Concepts</b> .....	<b>8</b>
Signal Configuration.....	11
Power Up vs. Power Down Timing.....	13
Glitch Control .....	14
Signal Driving.....	17
Signal Monitoring .....	19
Voltage Measurements .....	21
Default Startup State.....	22
<b>Controlling the Module</b> .....	<b>23</b>
Terminal Command Set .....	23
<b>Appendix 1 - Signal Names</b> .....	<b>37</b>
<b>Appendix 2 – Signals supporting ‘Monitoring’</b> .....	<b>41</b>
<b>Appendix 3 – Signals supporting ‘Driving’</b> .....	<b>42</b>

## Introduction

The **GEN5 OCP NIC x16 Breaker** enables remote switching of all data and control pins on an OCP NIC 3.0 (x16) interface, providing powerful capabilities for test automation and fault injection.

This module supports data rates up to **32GT/s** per lane, making it suitable for full-speed PCIe Gen5 testing.

Each pin is individually switched, allowing complete control over the mating sequence of a drive connector.

The switches can be sequenced at precise timings to simulate a hot-swap event, including pin bounce. Individual pins can also be broken or glitched at any time to simulate a fault in the system.

The module fits directly between the OCP NIC card and its host system using the standard x16 OCP NIC 3.0 connector.

This module supports the 'High Resolution' firmware feature, which allows for more detailed time resolution for initial delay and bounce period timing.

The triggering version of the module includes trigger IN and OUT connectors to allow sync with other test equipment, such as analyzers or power modules.

Two versions of the module are available:

QTL3095 – The basic module

QTL3096 – Module with additional external triggering, allowing sync with external equipment and additional monitoring of sideband signals

## Technical specifications

### Power requirements

The modules take power from its controller: Either a QTL1260 Interface Kit, or a QTL1461/QTL1079 Array Controller. No power is required from the host/device.

### Switching characteristics

Drive pins	Description	Switching Action
PCIe data	16 lanes of GEN5 PCIe data	Each signal is individually switched by a high-speed Switch
12v 3v3_aux	Power supply	Individually switched by power FET
REFCLK	4x differential RefClk signals	Each signal is individually switched by a high-speed Switch
Sideband	PRSNT_A, PRSNT_B0, PERST#, NIC_PWR_GOOD, SM BUS, etc	Each signal is individually switched

### Protocol Compatibility

The modules are protocol agnostic and can switch to any protocol which is compatible with EDSFF electrical specification.

Depending on the quality of your host, device and cabling, there may be some cases where the highest speed protocols do not work error-free through the modules. We always recommend that you evaluate a module in your live system before purchasing, especially when long traces are used, or where the mechanical fit may be tight.

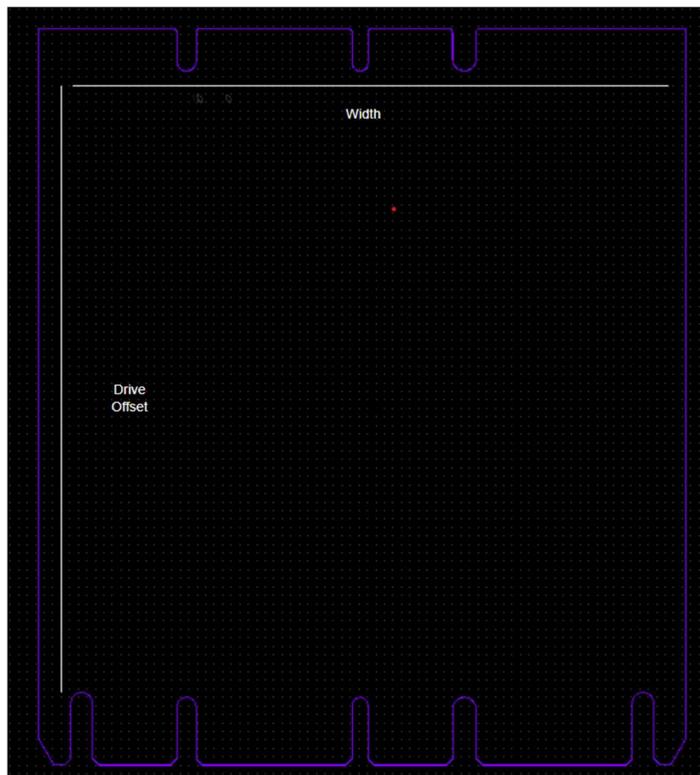
## Mechanical characteristics

The size:

Width 75.4mm

Height 7.5mm

Drive Offset 86mm



## Control Interfaces

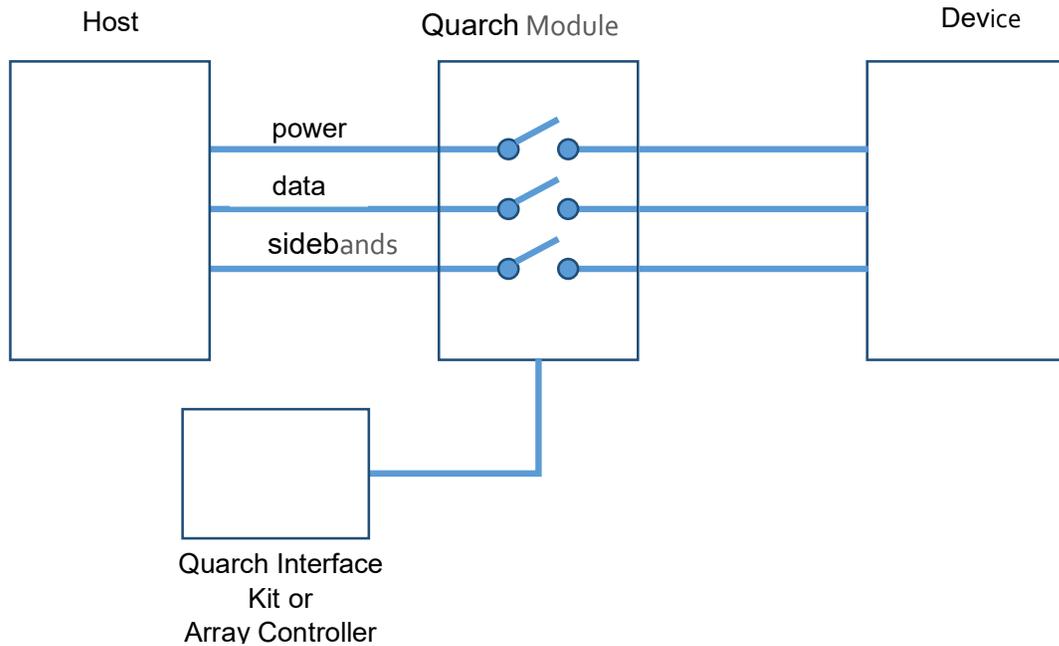
All Torridon modules are designed to be used with a Torridon Array Controller (QTL1461, QTL1079) or a single Torridon Interface Kit (QTL1260).

The control cable is an ultra-thin flex cable.

Control Interface	Form Factor	Torridon Ports	Control Methods Available	Interfaces
<b>QTL1079</b> 28 Port Torridon Array Controller	1U 19" Rack Mounted unit	24 at the front 4 at the rear	Terminal Scripting  TestMonkey 2 GUI	Serial via DB9 or RJ45  Ethernet  USB
<b>QTL1461</b> 4 Port Array Controller	160x165x53mm Enclosure  1U Enclosure also available	4 ports on front	Terminal Scripting  TestMonkey 2 GUI	Serial via RJ45  Ethernet  USB
<b>QTL1461</b> Torridon Interface Kit	60mm x 45mm x 30mm Box	1 port	Terminal Scripting  TestMonkey 2 GUI	Serial via RJ- 45  Serial via USB/Serial convertor  USB

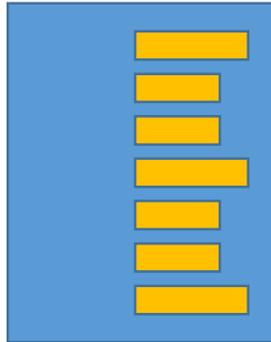
## Basic Concepts

Each controlled pin is connected to a separate switch on the module, so it can be connected or isolated on command.

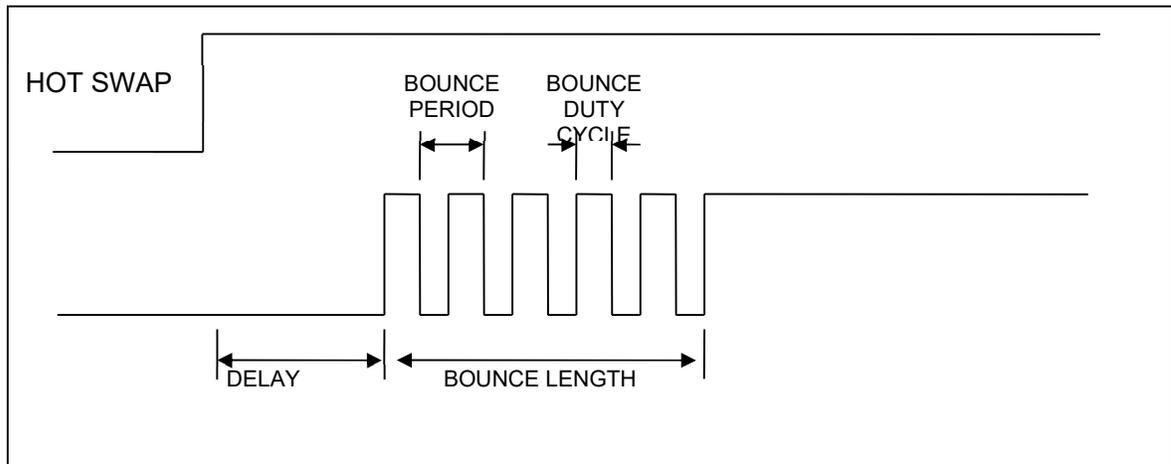


Each switch on the module is called a ‘Signal’ and can be programmed to follow one of six programmable delay and bounce profiles (called ‘Sources’). This allows the user to sequence the signal connections in the cable in up to six programmable steps.

This allows us to create virtually any hot-swap scenario. The default scenario on the module is based on the pin lengths on the connector, so that the long pins mate first, followed by shorter pins.



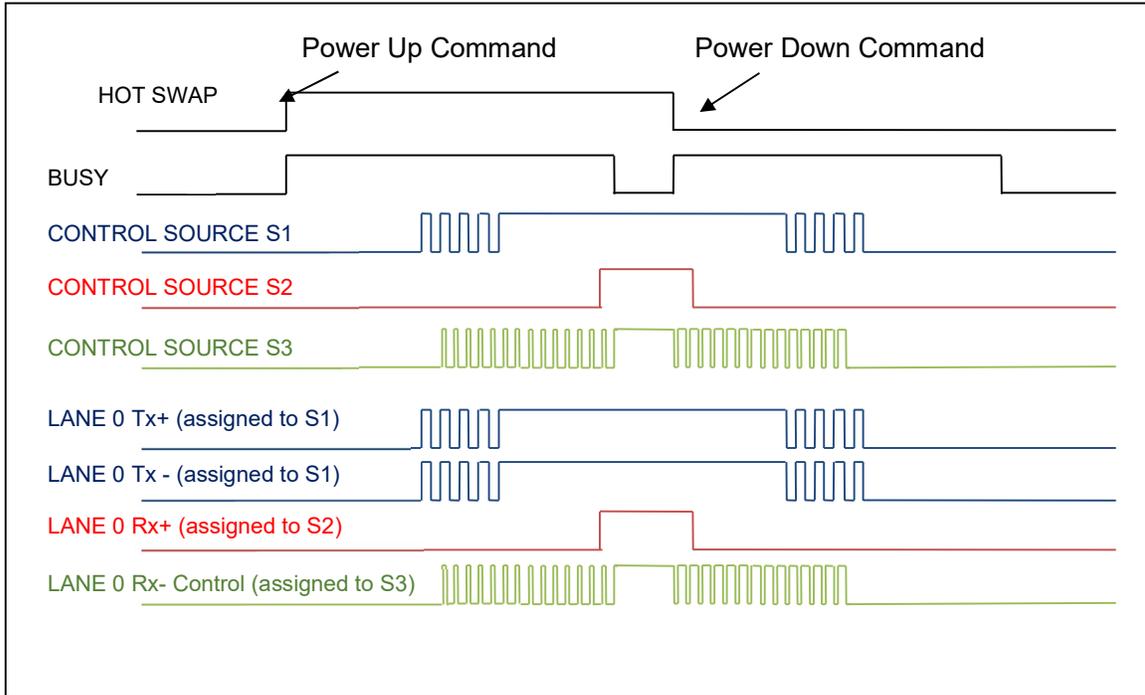
Each of the programmable delay and bounce profiles is called a control source, S1 to S4. For each control source the user sets up a delay, and bounce parameters. Three special sources (S0, S7 and S8) are also provided as described in the table below.



Control Source Parameters for a power up event (Basic Pin Bounce)

Once each delay period is set up, the user assigns each signal to follow the relevant control source, then uses the “**run:power up**” and “**run:power down**” commands to initiate the hot-swap.

The BUSY bit 1 in the control register is set during a power up, power down and short operation. This may be used to monitor for the completion of timed events.



Power up and Power down example

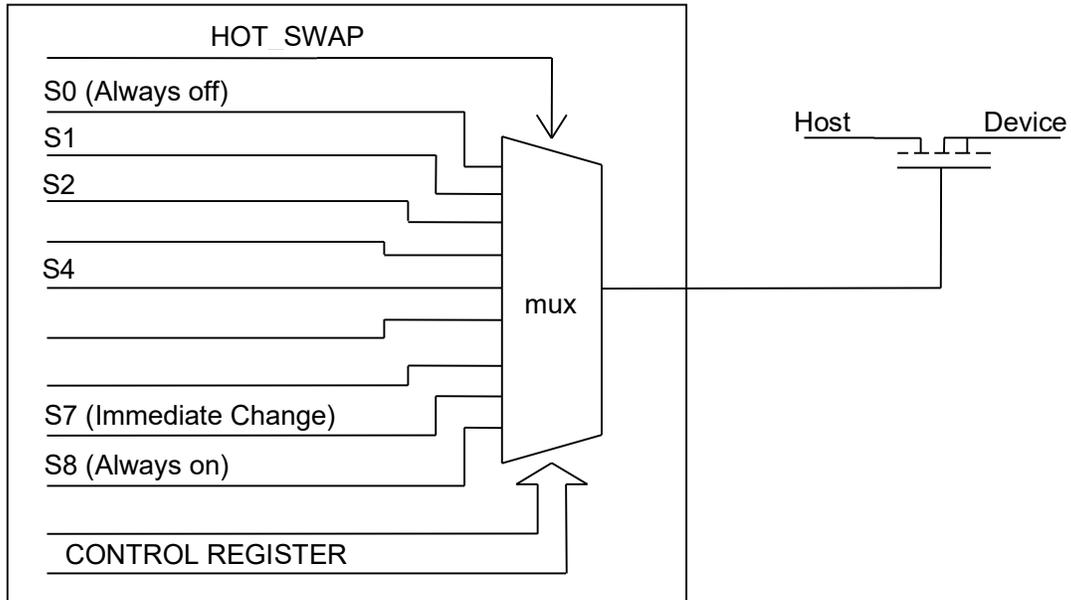
## Signal Configuration

Each signal that is switched by the module is usually assigned to one of the 6 timed sources, S1 – S4. Each signal can also be assigned directly to 'always off' (source 0), 'immediate change' (source 7) or 'Always on' (source 8).

Signals assignment is done through the command:

**SIGnal:[name]:SOURce [Source#]**

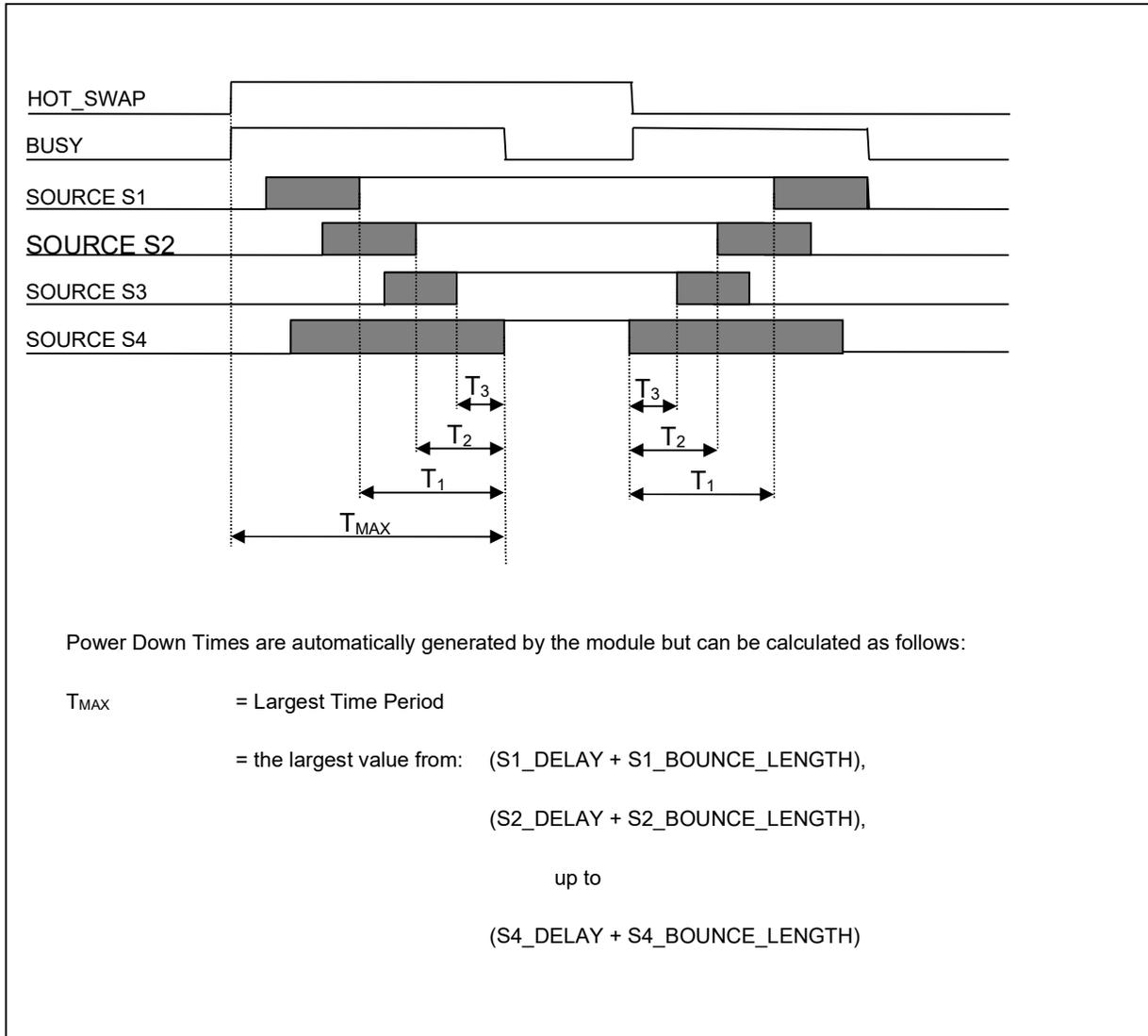
Source Number	Description
0	Signal is always OFF
1	Signal assigned to control source 1
2	Signal assigned to control source 2
3	Signal assigned to control source 3
4	Signal assigned to control source 4
7	Signal changes with HOT_SWAP state
8	Signal is always ON



This diagram shows the 9 possible source settings entering the control MUX for a switched signal. The value of the control register will determine which of the sources are used to control the signal. When enabled, the hot-swap line will cause the MUX to pass the control signal from that source through to the switch.

## Power Up vs. Power Down Timing

Each control source is always configured with power-up parameters. The power-down profile is automatically generated by the module, and is the mirror image of the power up:



If you require a different power down sequence then you can alter any of the source timing values, pin bounce or signal assignments while the module is in the plugged state. When you initiate the 'pull' action, the new settings will be used.

## Glitch Control

Any control signal may be glitched for a pre-determined length of time using the glitch generator logic.

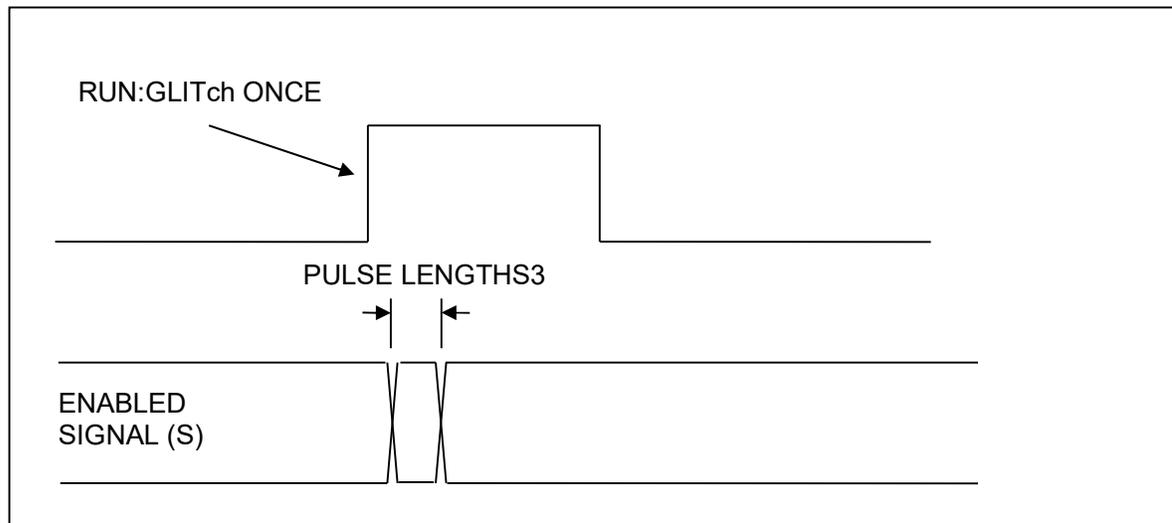
Each Signal Control register contains a “**GLITCH:ENABLE**” bit which determines whether the glitch logic will affect that signal. The setting, defaults to off, so any glitches will have no effect unless explicitly set to do so.

Glitches will invert the current state of the switched signal. Therefore if a switch is currently OFF, a glitch will turn it ON, and if the switch is ON, it will turn OFF.

For modules that support signal driving, the glitch action will drive the signal following the ‘**DRIVE:OPEN**’ and ‘**DRIVE:CLOSED**’ settings

Glitches may be applied in 3 modes:

### Glitch Once



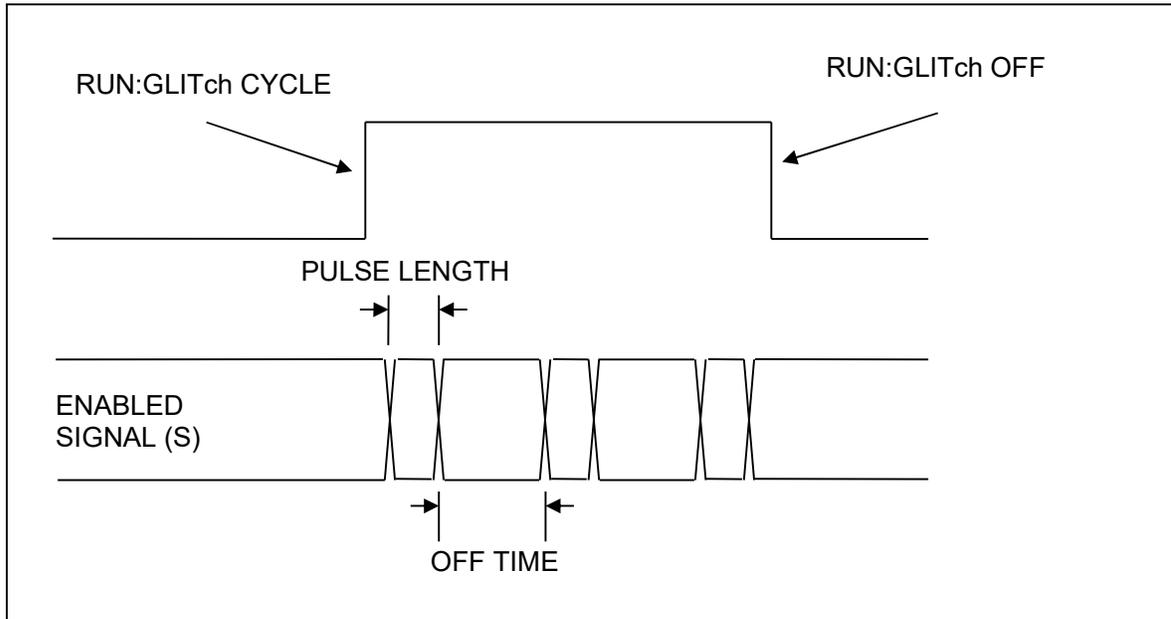
A single glitch is generated when the **RUN:GLITCh ONCE** command is executed.

The length of the glitch is determined by using the **GLITCh:SETup** command or the **GLITCh:MULTiplier** and **GLITCh:LENgth** commands:

$$\text{PULSE LENGTH} = \text{GLITCh:MULTiplier} \times \text{GLITCh:LENgth}$$

Repeated use of the **RUN:GLITCh:ONCE** command will generate multiple glitches, it is not necessary to use the **RUN:GLITCh OFF** command after a single glitch.

### Glitch Cycle



A sequence of glitches is generated when the **RUN:GLITCh CYCLE** command is executed, and continues until **RUN:GLITCh OFF** is executed.

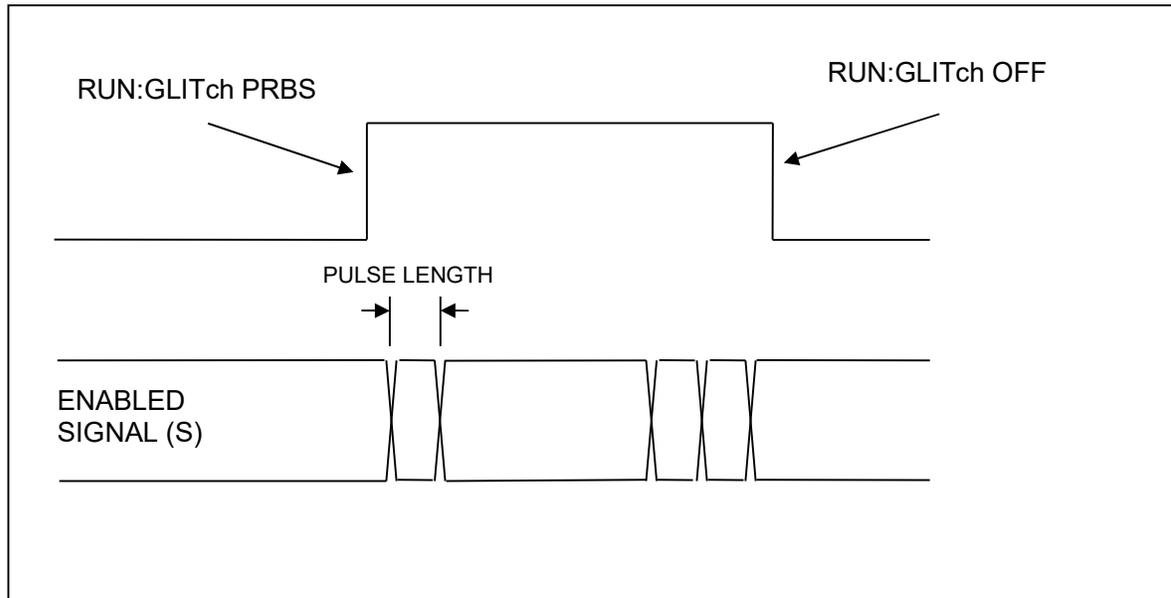
The length of the glitch is determined by using the **GLITCh:SETup** command or the **GLITCh:MULTiplier** and **GLITCh:LENgth** commands:

$$\text{PULSE LENGTH} = \text{GLITCh:MULTiplier} \times \text{GLITCh:LENgth}$$

The length of time between each glitch pulse is set in the same way as the glitch length, The length of the gap is determined by using the **GLITCh:CYCle:SETup** command or the **GLITCh:CYCle:MULTiplier** and **GLITCh:CYCle:LENgth** commands:

$$\text{OFF TIME} = \text{GLITCh:CYCle:MULTiplier} \times \text{GLITCh:CYCle:LENgth}$$

### Glitch PRBS



A pseudo random sequence of glitches is generated when the **RUN:GLITCh PRBS** command is executed, and continues until **RUN:GLITCh OFF** is executed.

The length of the glitch is determined by using the **GLITCh:SETup** command or the **GLITCh:MULTiplier** and **GLITCh:LENGTh** commands:

$$\text{PULSE LENGTH} = \text{GLITCh:MULTiplier} \times \text{GLITCh:LENGTh}$$

The number of glitches in a set length of time is determined by the **GLITCh:PRBS** command. A value of 2 will result in glitches at a ratio of 1:2 (the line will be in a glitched state 50% of the time), whilst a value of 256 will produce glitches in a ratio of 1:256.

## Signal Driving

The module has the ability to drive the following sideband signals in certain configurations. See the appendix for a list.

For these signals, the user can specify a behaviour using the **SIGNAL:[SIG\_NAME]:DRIVE:[OPEN|CLOSED] [NONE|HIGH|LOW]** command. The **OPEN** parameter is used to specify the action that the module should take when the switch is open (following a **RUN:POWER DOWN** command or when the signal is assigned to source 0), and the **CLOSED** parameter is used to specify the action to take when the switch should be closed (following a **RUN:POWER UP** command or when the signal is assigned to Source 8). The default behavior for both **OPEN** and **CLOSED** states is **NONE**, which tells the module not to drive the signal lines at all (just open and close the switch as usual).

The behavior of the module when signal driving is enabled (set to **HIGH** or **LOW**) is different depending on the signal being driven to avoid hardware conflicts:

### Examples:

#### *PERST*

Note that PERST is used as a general example, your module may have signals with different names. Please check the list of available signals in the appendix of this document

To issue a fundamental reset to the device under test:

PERST is an active low signal so to assert one we need to drive it low. Assuming the module is already powered up then we need to change the **CLOSED** behavior from **NONE** to **LOW**, and then back again to clear the reset.

**>SIGNAL:PERST:DRIVE CLOSED LOW**

*(Line is driven low: reset is asserted)*

**>SIGNAL:PERST:DRIVE CLOSED NONE**

*(Line driving is disabled: reset is de-asserted)*

*Advanced Usage:*

If we want to assert PERST for a set period of time, we can set the **OPEN** behavior and use the glitch function on the PERST signal to control the “open” time.

**>SIGnal:PERST:GLITch:ENable ON**

**>SIGnal:PERST:GLITch:SETup 500us 2**

**>SIGnal:PERST:DRIVE OPEN LOW**

**>RUN:GLITch ONCE**

During the glitch event, the switch would normally be open for 1mS. The addition of the driving setting changes this to instead drive the signal low for 1mS

## Signal Monitoring

The 'signal monitoring' feature allows specific signals on a module (normally sideband signals) to be tracked.

The state of a monitored signal can be requested from the module at any time via a command

On 'triggering' modules, the state of a signal can be output in real time to one of the triggering ports. As there are two trigger ports, two signals can be monitored at a time. This is ideal for diverting SM\_BUS to an analyser.

For a list of signals on the module that support triggering, see the annex at the end of the manual.

### Requesting signal state

To get the state of a monitored signal:

**SIGnAl:[SIGNAL-NAME]:STATus:[HOST?|DEVIce?]**

Returns the current state of the monitored signal as **HIGH** or **LOW**. The signal state can (if supported) be monitored independently on both the host and device side of the module.

### Live monitoring

This feature is supported on 'Triggering' modules only. Both the trigger IN and OUT ports can be used to monitor a signal.

**WARNING:** As the trigger IN port can be ordered to OUTPUT a status, there is a risk of two devices driving against each other and causing damage. Before using the live monitoring feature, you must ensure that you do not have any equipment attached that may try to drive the trigger IN port.

To begin live monitoring, first enable the trigger ports you want to use. This is done via additional options to the existing trigger setup commands:

Trigger OUT port:

```
# Set the trigger mode to sideband monitor  
TRIGger:OUT:MODE:SIDEband
```

Trigger IN port (requires double verification)

```
# Set the trigger mode to sideband monitor  
TRIGger:IN:MODE:SIDEband  
# Also set the trigger IN source to sideband out  
TRIGger:IN:SOURCE:SIDEband
```

The commands to control monitoring are:

```
# Select a signal for live monitoring
```

```
TRIGger:MONitor[IN|OUT]:[SIGNAL-NAME]:[HOST|DEvice]
```

Sets a trigger port to activate live monitoring for a given signal. The host/device parameter selects the side of the module to monitor on.

```
TRIGger:MONitor[IN?|OUT?]
```

Returns the selection for live monitoring on the given trigger port. The response will be in the form **PERST:HOST** or similar (**SIGNAL\_NAME:SIDE**)

Note that the triggering mode must also be set before the live monitoring will start.

## Voltage Measurements

The modules are capable of measuring various voltages both for self test and to assist in the testing of a customer’s system. The following measurement points are available:

Measurement Command	Description	Resolution / Accuracy
<b>MEASure:VOLTage:SELF 3v3?</b>	Returns the voltage of the module’s internal 3.3v power rail	64mV / 5%
<b>MEASure:VOLTage:SELF 5v?</b>	Returns the voltage of the module’s internal 5v internal power rail	64mV / 5%
<b>MEASure:VOLTage:SELF -5v?</b>	Returns the voltage of the module’s internal -5V power rail	64mV / 5%
<b>MEASure:VOLTage 12vin?</b>	Returns the host side 12V power rail	64mV / 5%
<b>MEASure:VOLTage 12vout?</b>	Returns the drive side 12V power rail	64mV / 5%
<b>MEASure:VOLTage 3v3_auxin?</b>	Returns the host side 3V3_aux power rail	64mV / 5%
<b>MEASure:VOLTage 3v3_auxout?</b>	Returns the drive side 3V3_aux power rail	64mV / 5%

## Default Startup State

On power up or reset, the control modules enter a default state. On the cable module all signals are connected at startup. The “run:power down” command will immediately disconnect the cable without needing any initial setup.

The default hot-swap scenario will disconnect data pins immediately, followed by the management interface (as the management interface uses longer pins)

Source Number	Initial Delay	Pin Bounce Mode	Bounce Length	Bounce Period	Bounce Duty Cycle
1	0mS	Standard	0mS	0uS	50%
2	0mS	Standard	0mS	0uS	50%
3	0mS	Standard	0mS	0uS	50%
4	0mS	Standard	0mS	0uS	50%
5	0mS	Standard	0mS	0uS	50%
6	0mS	Standard	0mS	0uS	50%

Signal	Assigned Source
REFCLKp0, REFCLKn0, REFCLKp1, REFCLKn1, REFCLKp2, REFCLKn2, REFCLKp3, REFCLKn3	Source 1
SMBRST, SMBCLK, SMBDAT	Source 1
PWRBRK0, BIF0, BIF1, BIF2	Source 1
PRSNT_A, PRSNT_B0, PRSNT_B1, PRSNT_B2, PRSNT_B3, PERST0, PERST1, PERST2, PERST3	Source 1
12V, 3V3_AUX	Source 1
All PCIe Data signals	Source 1
All RBT signals	Source 1

All SCAN signals	Source 1
------------------	----------

### Hot-Swap State:

The cable is in the 'plugged' state, waiting for a **RUN:POWER DOWN** command to disconnect it.

## Controlling the Module

The module can be controlled either by:

- Serial ASCII terminal (such as HyperTerminal)  
  
This is normally used with scripted commands to automate a series of tests. The commands are normally generated by a script or user code (PERL, TCL, C, C# or similar).
- Telnet Terminal (Only when connected to an Array Controller).  
  
This mode uses exactly the same commands as the serial ASCII terminal, but run over a standard Telnet connection.
- REST API (Only when connected to an Array Controller).  
  
Controllers provide a basic REST API, allowing multi-user control over Torridon products.
- USB  
  
Quarch's TestMonkey application can control a single module via USB, this allows simple graphical control of the module. The Quarch C# API and Python examples allow automation via USB.

## Terminal Command Set

These commands are based on the SCPI style control system that is used by many manufacturers of test instruments. The entire SCPI specification has NOT been implemented but the command structure will be very familiar to anyone who has used it before.

- SCPI commands are NOT case sensitive
- SCPI commands are in a hierarchy separated by ':'  
(**LEVe11:LEVe12:LEVe13**)
- Most words have a short form (e.g. '**register**' shortens to '**reg**'). This will be documented as **REGister**, where the short form is shown in capitals.

- Some commands take parameters. These are separated by spaces after the main part of the command (e.g. “`meas:volt:self 3v3?`” obtains the 3v3 self test measurement).
- Query commands that return a value all have a ‘?’ on the end
- Commands with a preceding ‘\*’ are basic control commands, found on all devices.
- Commands that do not return a particular value will return “OK” or “FAIL”. Unless disabled, the fail response will also append a text description for the failure if it can be determined.

### # [comments]

Any line beginning with a # character is ignored as a comment. This allows commenting of scripts for use with the module.

### \*RST

Triggers a reset, the module will behave as if it had just been powered on.

### \*CLR

Clear the terminal window and displays the normal start screen. Also runs the internal self test. The same action can be performed by pressing return on a blank line.

### \*IDN?

Displays a standard set of information, identifying the device. An example return is shown below:

Family:	Torridon System	[The parent family of the device]
Name:	Ethernet Cable Pull Module	[The name of the device]
Part#:	QTL1271-01	[The part number of the hardware]
Processor:	QTL1159-01,3.50	[Part# and version of firmware]
Bootloader:	QTL1170-01,1.00	[Part# and version of bootloader]

FPGA 1: 1.0 [Version of FPGA core]

#### **\*TST?**

Runs a set of standard tests to confirm the device is operating correctly, these tests are also performed at start up. Returns 'OK' or 'FAIL' followed by a list of errors that occurred, each on a new line.

#### **CONFig:MODE BOOT**

Configures the card for boot loader mode (to update the firmware), requires an update utility on the PC.

#### **CONFig:MESSAgES [SHORT|USER]**

#### **CONFig:MESSAgES?**

Gets or sets the mode for messages that are returned to the user's terminal

**Short:** Only a "FAIL" or "OK" will be returned.

**User:** Full error messages are returned to the user on failure.

#### **CONFig:TERMinAl USER**

Sets the terminal response mode to the default 'User' setting. This is intended for use with HyperTerminal or similar and manually typed commands.

#### **CONFig:TERMinAl SCRIPT**

Sets the terminal response mode for easier parsing. Especially useful from a UNIX/LINUX based system. Characters sent from the PC are not echoed by the device and a <CR><LF> is sent after the cursor to force a flush of the USART buffer.

#### **CONFig:TERMinAl?**

Returns the current terminal mode.

#### **CONFig:DEFault STATE**

Resets the state of the module. This will set all source/signal/glitch etc logic to its default power-on values. Terminal setting will not be affected. This command allows the module to be brought back to a known state without resetting it.



**SOURce:[1-4|ALL]:SETup [#1] [#2] [#3] [#4]**

Sets up the source in a single command. All parameters are positive integer numbers:

#1 = Initial delay (mS)

[Limits: 0 to 127ms in steps of 1ms, 130ms to 1270ms in steps of 10ms]

#2 = Bounce length (mS)

[Limits: 0 to 127ms in steps of 1ms, 130ms to 1270ms in steps of 10ms]

#3 = Bounce Period (uS)

[Limits: 10 to 1270us in steps of 10us, 2000 to 127000us in steps of 1000us]

#4 = Duty Cycle (%)

[Limits: 0 to 100% in steps of 1%]

SOURce:[1-4|ALL]:DELAY [#ms] [#Unit\*]

SOURce:[1-4|ALL]:DELAY?

Sets the initial delay of a source in mS. The delay is entered as a integer number with no units. E.g. "Source:1:delay 300".

#1 = Initial delay (mS)

[Limits: 0 to 127ms in steps of 1ms, 130ms to 1270ms in steps of 10ms]

**#2 = Optional unit specifier (High resolution firmware only) [uS, mS, S]. High resolution firmware allows initial delay of 0 to 16,775mS in 1uS resolution. This parameter is optional, to be back-compatible with older firmware**

SOURce:[1-4|ALL]:BOUNce:SETup [#1] [#2] [#3]

Sets up the bounce parameters in a single command. All parameters are positive integer numbers:

#1 = Bounce length (mS)

[Limits: 0 to 127ms in steps of 1ms, 0 to 1270ms in steps of 10ms]

#2 = Bounce Period (uS)

[Limits: 10 to 1270us in steps of 10us, 2000 to 127000us in steps of 1000us]

#3 = Duty Cycle (%)

[Limits: 0 to 100% in steps of 1%]

**SOURce:[1-4|ALL]:BOUNce:LENgth [#ms] [#Unit\*]**

**SOURce:[1-4|ALL]:BOUNce:LENgth?**

Sets the length of the pin bounce in mS. The delay is entered as a decimal number with no units. E.g. "Sour:2:boun:len 50".

#1 = Bounce length (mS)

[Limits: 0 to 127ms in steps of 1ms, 130ms to 1270ms in steps of 10ms]

#2 = Optional unit specifier (High resolution firmware only) [uS, mS, S]. High resolution firmware allows initial delay of 0 to 16,775mS in 1uS resolution. This parameter is optional, to be back-compatible with older firmware

**SOURce:[1-6|ALL]:BOUNce:PERiod [#us] [#Unit\*]**

**SOURce:[1-6|ALL]:BOUNce:PERiod?**

Sets the bounce period of the pin bounce in uS. The value is entered as a decimal number **with no units**. E.g. "Sour:6:boun:period 300".

#1 = Bounce Period (uS)

[Limits: 10 to 1270us in steps of 10us, 2000 to 127000us in steps of 1000us]

#2 = **Optional unit specifier (High resolution firmware only) [uS, mS, S]. High resolution firmware allows initial delay of 0 to 1,677mS in 100nS resolution. This parameter is optional, to be back-compatible with older firmware**

**SOURce:[1-4|ALL]:BOUNce:DUTY [#%]**

**SOURce:[1-4|ALL]:BOUNce:DUTY?**

Sets the duty cycle of the pin bounce as a %. The value is entered as a decimal number with no units. E.g. "source:3:bounce:duty 50".

#1 = Duty Cycle (%)

[Limits: 0 to 100% in steps of 1%]

**SOURce:[1-4|ALL]:BOUNce:MODE [SIMPLE|USER]**

**SOURce:[1-4|ALL]:BOUNce:MODE?**

Sets the bounce pattern to **SIMPLE** (Duty cycle driven oscillation) or **USER** (User defined custom pattern).

**SOURce:[1-4|ALL]:BOUNce:PATtern:WRITe [0xAAAA] [0xDDDD]**

Writes a word of the custom bounce pattern to the give address within the pattern

0xAAAA is the address (for example 0x0002)

0xDDDD is the pattern data (for example 0x13F2)

**SOURce:[1-4|ALL]:BOUNce:PATtern:READ [0xAAAA]**

**Reads a word of** the custom bounce pattern

0xAAAA is the address (for example 0x0002)

**SOURce:[1-4|ALL]:BOUNce:PATtern:DUMP [0xAAAA] [0xAAAA]**

Reads a range of words from the custom bounce pattern

0xAAAA is the start and end address range (for example 0x0002)

**SOURce:[1-4|ALL]:BOUNce:CLEAR**

Removes any pin bounce from the source and sets all bounce settings to default values. See “Default Startup State” for details for the default settings.

**SOURce:[1-4|ALL]:STATE [ON|OFF]**

**SOURce:[1-4|ALL]:STATE?**

Sets or returns the enable state of the source. Any signals assigned to a disabled (off) source will immediately be disconnected and vice versa. If a source state is changed, all signals assigned to it will change at exactly the same time (if a change is required).

**SOURce:[1-4]:BOUNce:PATtern:LENgth [#bits]**

**SOURce:[1-4]:BOUNce:PATtern:LENgth?**

Sets or returns the number of bits of the custom bounce pattern that are to be used. This defaults to the maximum (112) and can be reduced to create more accurate patterns.

**SOURce:[1-4]:BOUNce:PATtern:REPeat [ON|OFF]**

**SOURce:[1-4]:BOUNce:PATtern:REPeat?**

Sets the custom pattern repeat **flag**. This is used when the current custom bounce pattern is shorter than the specified bounce length. When the flag is set (default) the pattern will wrap. When this flag is off, the last bit of the pattern will be repeated.

**SOURce:[1-4]:BOUNce:PATtern:SETup [#us] [#binarypattern]**

Sets a basic custom pattern from a single command. This command will alter the bounce period, bounce length, pattern length and the custom pattern.

[#uS] – Integer value of uS to specify the period. The length of each bit in the pattern will be half of this value. 20uS is the minimum value (10uS per bit)

[#binarypattern] – String parameter containing 1s and 0s, for example “001” is a 2 bit pattern that is low for 2 bits then high for 1. The given pattern will always be padded up to the nearest millisecond. This is because the total glitch length has a 1mS resolution.

**SIGnal:[SIG\_NAME|ALL]:SOURce [#num]**

**SIGnal:[SIG\_NAME|ALL]:SOURce?**

Assigns a given signal to a numbered timing source (0-8). SIGNAL\_NAME is one of the signals/groups as found in the ‘Signal Names’ appendix at the end of this manual

**SIGnal:[SIG\_NAME|ALL]:GLITch:ENABle [ON|OFF]**

**SIGnal:[SIG\_NAME|ALL]:GLITch:ENABle?**

Enables a signal for glitching. If this is on, the signal will be glitched whenever the glitch logic is in use. Multiple signals may be set to glitch at the same time.

**SIGnal:[SIG\_NAME]:DRIVE:OPEN [HIGH|LOW|NONE]**

**SIGnal:[SIG\_NAME]:DRIVE:OPEN?**

Sets the 'drive' mode for a signal when the switch would normally be 'open'. Only the 'driving' capable signals (listed in the appendix) can support this.

**SIGnal:[SIG\_NAME]:DRive:CLosed [HIGH|LOW|NONE]**

**SIGnal:[SIG\_NAME]:DRive:CLosed?**

Sets the 'drive' mode for a signal when the switch would normally be 'closed'. Only the 'driving' capable signals (listed in the appendix) can support this.

**SIGnal:[SIG\_NAME]:STATus:HOST?**

**SIGnal:[SIG\_NAME]:STATus:DEVice?**

Requests the current state of the sideband signal on the host or device side, from the monitoring system. Only the 'monitoring' capable signals (listed in the appendix) can support this.

**GLITch:SETup [MULTIPLIER\_STEP] [#count]**

Sets up the length of the glitch in a single command.

#1 = Multiplier factor for glitch length (mS)

[50ns|500ns|5us|50us|500us|5ms|50ms|500ms]

#2 = Length of the glitch (number of times the multiplication factor will be run)

[Limits: 0 to 255 in steps of 1]

This gives a maximum glitch of 127.5 Seconds.

**GLITch:MULTIplier [MULTIPLIER\_STEP]****GLITch:MULTIplier?**

Sets the multiplier value for the glitch time to one of the specified durations.

This factor is multiplied with the **GLITch:LENgth** value to give the actual glitch time.

#1 = Multiplier factor for glitch length (mS)

[50ns|500ns|5us|50us|500us|5ms|50ms|500ms]

**GLITch:LENgth [#count]****GLITch:LENgth?**

This value is multiplied by **GLITch:MULTIplier** to give the glitch duration.

#1 = Length of the glitch (number of times the multiplication factor will be run)

[Limits: 0 to 255 in steps of 1]

**GLITch:CYCLE:SETup [MULTIPLIER\_STEP] [#count]**

Sets up the length of the glitch cycle in a single command.

#1 = Multiplier factor for glitch cycle length (mS)

[50ns|500ns|5us|50us|500us|5ms|50ms|500ms]

#2 = Length of the glitch cycle (number of times the multiplication factor will be run)

[Limits: 0 to 255 in steps of 1]

This gives a maximum glitch cycle time of 127.5 Seconds.

**GLITch:CYCLe:MULTIplier [MULTIPLIER\_STEP]**

**GLITch:CYCLe:MULTIplier?**

Sets the multiplier value for the glitch cycle time to one of the specified durations.

This factor is multiplied with the **GLITch:CYCLe:LENgth** value to give the actual time between cycled glitches.

#1 = Multiplier factor for glitch length (mS)

[50ns|500ns|5us|50us|500us|5ms|50ms|500ms]

**GLITch:CYCLe:LENgth [#count]**

**GLITch:CYCLe:LENgth?**

This value is multiplied by **GLITch:CYCLe:MULTIplier** to give the actual time between cycled glitches.

#1 = Length of the glitch (number of times the multiplication factor will be run)

[Limits: 0 to 255 in steps of 1]

**GLITch:PRBS [#1]**

Sets the PRBS rate for Pseudo Random repeat glitching, this is a ratio, 2 means 1:2 (approximately 50% of the time the signal will be glitched), 256 means 1:256.

#1 = PRBS Ratio

[2|4|8|16|32|64|128|256|512|1024|2048|4096|8192|16384|32768|65536]

**RUN:POWer [UP|DOWN]**

Initiates a plug or pull operation (legacy name used to preserve compatibility between Torridon modules). This is the master control for all switches on the card.

The command will fail if you order a power up when the module is already in the connected state and vice-versa as the action cannot be performed.

The “OK” response will be returned as soon as the hot-swap event has begun. If your timing sequence is very long you may have to poll the BUSY bit in register 0 to check when it has completed.

**RUN:POWER?**

Returns the current plugged/pulled state of the module.

**RUN:GLITCh ONCE**

Triggers a single glitch with length:

**GLITCh:MULTIplier x GLITCh:LENGth.**

**RUN:GLITCh CYCLE**

Triggers a sequence of repeated glitches that run until the **RUN:GLITCh STOP** command is executed. All signals with **GLITCh:ENABle** set to **ON** are glitched for **GLITCh:MULTIplier x GLITCh:LENGth** and then released for a duration of **GLITCh:CYClE:MULTIplier x GLITCh:CYClE:LENGth**. This is repeated until the **RUN:GLITCh STOP** command is run.

**RUN:GLITCh PRBS**

Triggers a PRBS glitch sequence which runs until the **RUN:GLITCh STOP** command is issued.

**RUN:GLITCh STOP**

Stops any running glitch sequence.

**RUN:GLITCh?**

Returns the state of the current glitch sequence running on the module.

**TRIGger:IN:TYPE [EDGE|LEVEL]**

Sets the trigger type

EDGE = Actions will start on the asserted edge and complete in full

LEVEL = Actions will run for as long as the trigger signal is asserted

**TRIGger:IN:INVERT [ON|OFF]**

Sets the trigger invert mode for input triggers

OFF = Trigger acts as normal

ON = Trigger responds to an inverted input

**TRIGger:OUT:INVERT [ON|OFF]**

Sets the trigger invert mode for output trigger

OFF = Trigger acts as normal

ON = Trigger outputs in an inverted form

**TRIGger:IN:SOURce [EXTErnal|??\_host]**

Sets the source of the trigger in event

EXTErnal = Uses the trigger in connector

??\_host = Uses the output of the host power detect system. ?? is the voltage channel, generally 3v3\_host or 12v\_host, though the channel selection options will vary between modules and is not supported on all devices.

**TRIGger:IN:MODE [OFF|POWER|GLITCH|SIDEband]**

Sets the action to perform on a trigger in event

OFF = No action (default mode)

POWER = Power cycle will be performed

GLITCH = Glitch action will be performed

SIDEband= Sideband monitor mode (if 'monitoring' is supported)

**TRIGger:OUT:MODE [OFF|POWER|GLITCH|???\_host|SIDEband]**

Sets the action to perform on a trigger out event

OFF = No action (default mode)

POWER = Trigger out shows power state

GLITCH = Trigger out on glitch

???\_host = Trigger out on detect of host power. ??? is the voltage channel, generally 3v3\_host or 12v\_host, though the channel selection options will vary between modules and is not supported on all device.

SIDEband= Sideband monitor mode (if 'monitoring' is supported)

**TRIGger:MONitor:IN [#SignalName] [HOST|DEVIce]****TRIGger:MONitor:IN?**

If monitoring is supported and enabled for trigger IN, this command sets the sideband signal to be output on the trigger IN line.

You **MUST** ensure that nothing external will try to drive this trigger line when the monitor feature is enabled, as this may cause damage.

The signals available for monitoring are listed in the appendix

**TRIGger:MONitor:OUT [#SignalName] [HOST|DEVIce]****TRIGger:MONitor:OUT?**

If monitoring is supported and enabled for trigger IN, this command sets the sideband signal to be output on the trigger IN line.

You **MUST** ensure that nothing external will try to drive this trigger line when the monitor feature is enabled, as this may cause damage.

The signals available for monitoring are listed in the appendix

## Appendix 1 - Signal Names

The following signal names are used to specify a single signal or a group of signals. These may be used in commands that take a parameter “SIGNAL\_NAME”.

These signals names are based on the interface specification.

Note that some commands, such as those returning a value, only accept a parameter that resolves to a single signal. In this case you cannot use the groups

### Signals

PETpN (Data transmitted from the ‘input’ port on Lane n (+ve side of differential pair))

PETnN

PERpN

PERnN

12V

3V3\_AUX

REFCLKp0

REFCLKn0

REFCLKp1

REFCLKn1

REFCLKp2

REFCLKn2

REFCLKp3

REFCLKn3

PRSNT\_A

PRSNT\_B0

PRSNT\_B1

PRSNT\_B2

PRSNT\_B3

PERST0

PERST1

PERST2

PERST3

BIF0

BIF1

BIF2

SMBRST

SMBCLK

SMBDAT

WAKE

PWRBRK0

RBT\_TX\_EN

RBT\_CLK

RBT\_TXD0

RBT\_TXD1

RBT\_ARB\_IN

RBT\_ARB\_OUT

RBT\_RXD0

RBT\_CRS\_DV

SLOT\_ID0

SLOT\_ID1

SCAN\_CLK

SCAN\_DATA\_OUT

SCAN\_DATA\_IN

SCAN\_LD

USB\_DAT<sub>n</sub>\_N

USB\_DAT<sub>n</sub>\_P

AUX\_PWR\_EN

MAIN\_PWR\_EN

NIC\_PWR\_GOOD

RFU\_B68

RFU\_B69

### **Signal Groups**

ALL	(Allows change of all signals at the same time)
LANE0	(Affect all signals relating to a specific lane)
LANE1	
LANE2	
LANE3	
LANE4	
LANE5	
LANE6	
LANE7	
LANE8	
LANE9	
LANE10	
LANE11	
LANE12	
LANE13	
LANE14	
LANE15	
POWER	(Affects all power signals)
DATA	(Affects all data lanes)
SMB_BUS	(All SM BUS signals)
CLK_0	(REFCLKp0 and REFCLKn0)
CLK_1	(REFCLKp1 and REFCLKn1)
CLK_2	(REFCLKp2 and REFCLKn2)
CLK_3	(REFCLKp3 and REFCLKn3)
MANAGEMENT	(Controls all signals in the management interface)
RBT	(All RBT signals)
SCAN	(All SCAN signals)
DATA_0	(Affects lanes 0 to 3)
DATA_1	(Affects lanes 4 to 7)

DATA_2	(Affects lanes 8 to 11)
DATA_3	(Affects lanes 12 to 15)

## Appendix 2 – Signals supporting ‘Monitoring’

Signal Name	Side that can be monitored
PRSNTA	Host and Drive
PRSNTB0	Host and Drive
PRSNTB1	Host and Drive
PRSNTB2	Host and Drive
PRSNTB3	Host and Drive
SMBRST	Host and Drive
BIF0	Host and Drive
BIF1	Host and Drive
BIF2	Host and Drive
SMBDAT	Host and Drive
SMBCLK	Host and Drive
PWRBRK0	Host and Drive
RBT_TX_EN	Host and Drive
MAIN_PWR_EN	Host and Drive
NIC_PWR_GOOD	Host and Drive
AUX_PWR_EN	Host and Drive
PERST0	Host and Drive

PERST1	Host and Drive
PERST2	Host and Drive
PERST3	Host and Drive

### Appendix 3 – Signals supporting ‘Driving’

Signal Name	Signal Type	Host Side Behavior		Device Side Behavior	
		High	Low	High	Low
PRSNTA	Open Drain output from Host	Not Driven	Not Driven	Not Driven	Driven Low
PRSNTB0	Open Drain output from Device	Not Driven	Driven Low	Not Driven	Not Driven
PRSNTB1	Open Drain output from Device	Not Driven	Driven Low	Not Driven	Not Driven
PRSNTB2	Open Drain output from Device	Not Driven	Driven Low	Not Driven	Not Driven
PRSNTB3	Open Drain output from Device	Not Driven	Driven Low	Not Driven	Not Driven
SMBRST	Driven high by Host	Not Driven	Not Driven	Driven High	Driven Low
SMBCLK	Driven high by Host	Not Driven	Not Driven	Not Driven	Driven Low

SMBDAT	Bi-directional Open Drain	Not Driven	Driven Low	Not Driven	Driven Low
BIF0	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low
BIF1	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low
BIF2	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low
PWRBRK0	Open Drain Output from device	Not Driven	Not Driven	Not Driven	Driven Low
RBT_TX_EN	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low
MAIN_PWR_EN	Driven High by Host	Not Driven	Not Driven	Driven High	Driven Low
NIC_PWR_GOOD	Driven High by Device	Driven High	Driven Low	Not Driven	Not Driven
AUX_PWR_EN	Driven High by Host	Not Driven	Not Driven	Driven High	Driven Low
PERST0	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low
PERST1	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low
PERST2	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low
PERST3	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low

WAKE	Open Drain Output from device	Not Driven	Driven Low	Not Driven	Not Driven
------	-------------------------------------	---------------	---------------	---------------	------------