# Quarch Technology Ltd

# Torridon Ethernet Cable Pull Module

# Technical Manual

For use with:

**QTL2022 – Torridon RJ-45 Cable Pull Module**

Using Quarch firmware version 4.0 and above

Monday, 16 December 2019

## Change History

| | | |
|---|---|---|
| 1.0 | 05 November 2010 | Initial Release |
| 1.1 | 25 November 2010 | Updated to new format common sections |
| 1.2 | 06 January 2010 | Added firmware version 4.0 new commands<br>- 'Config:default" command<br>- Custom pin bounce commands |
| 1.3 | 08 November 2016 | Added info on new version QTL2022 module |
| 1.4 | 16 December 2019 | Updated to show new version of glitch commands |

## Contents



## Introduction

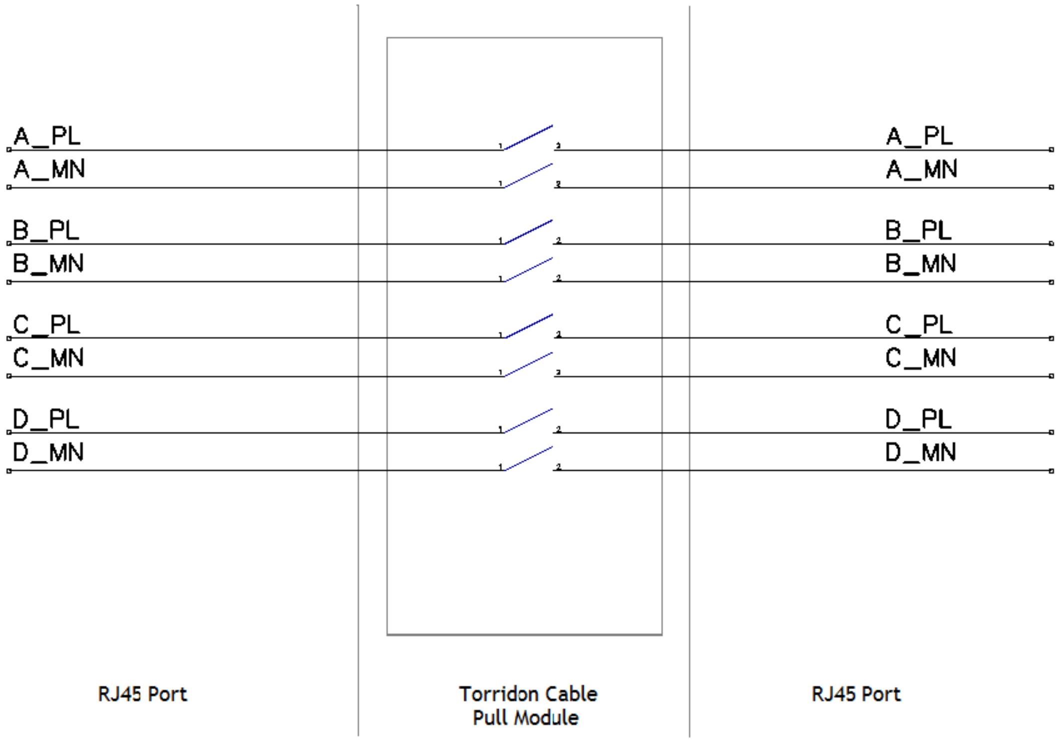
The **Torridon Ethernet Cable Pull Module** allows remote switching of the data pins in an RJ-45 Cable for test automation or fault injection purposes.

The modules support 10/100/1000 Base-T and 10G-Base-T connections

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

Quarch modules may be customized to support other proprietary signals or form factors on request.

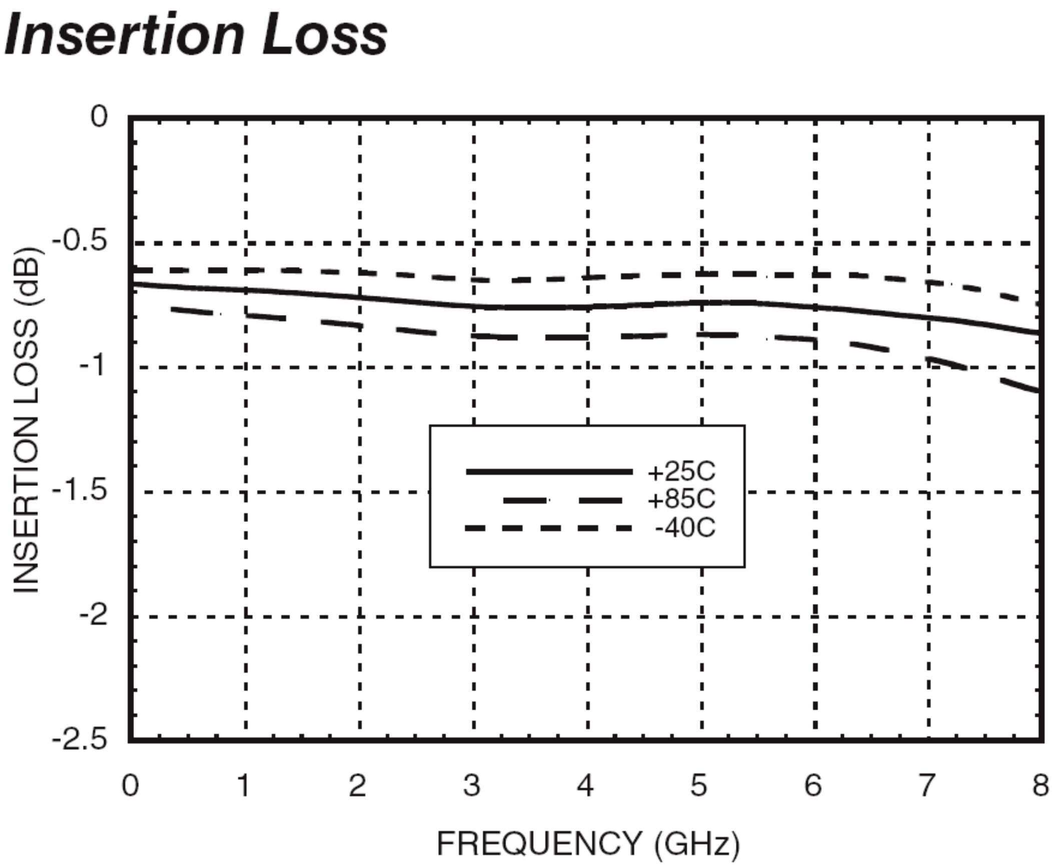


# Technical Specifications

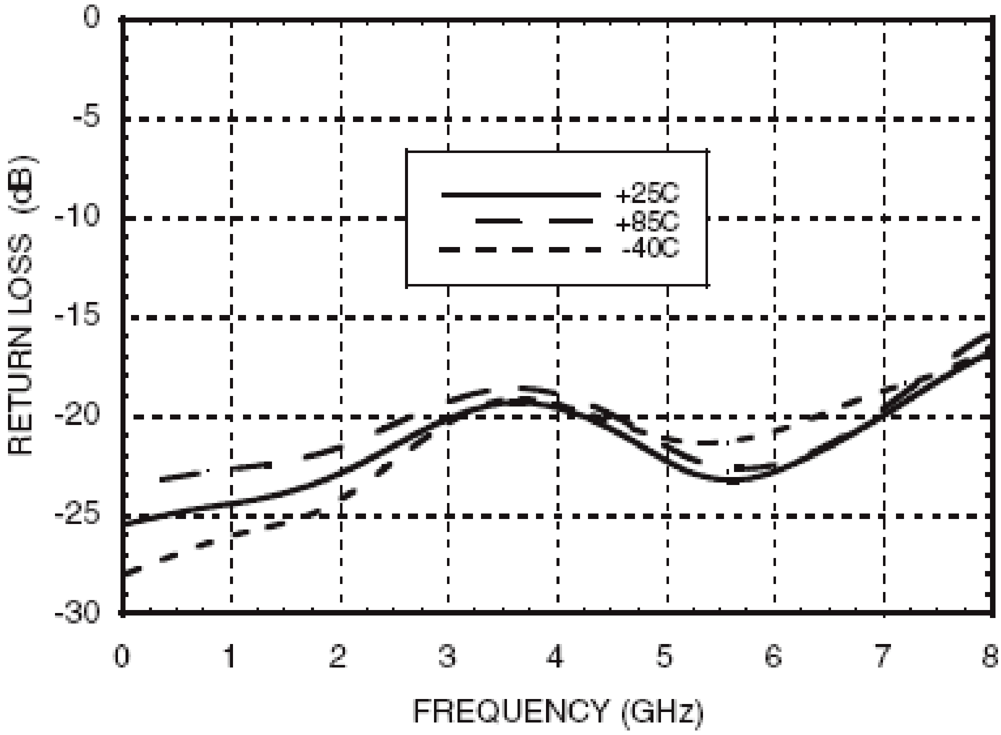
## Switching Characteristics:

| RJ-45 Connector Pin | Description | Switching Action |
| --- | --- | --- |
| A_PL, A_MN, B_PL, B_MN, C_PL, C_MN, D_PL, D_MN | RJ45 cable pairs | Each signal is individually switched by a High Speed RF Switch |

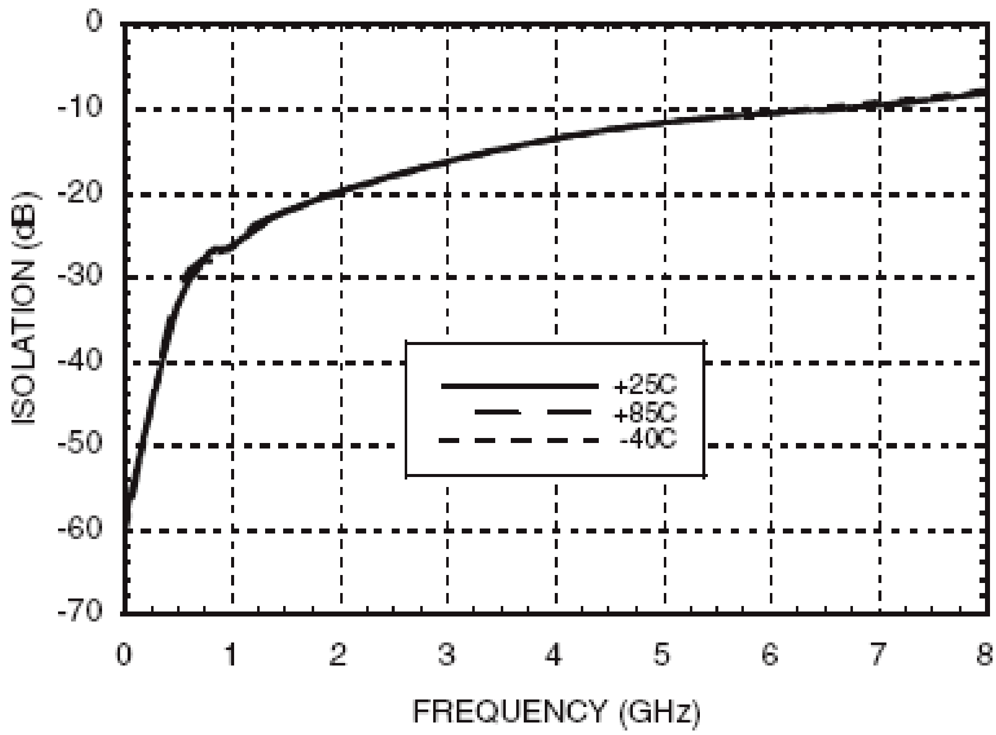
### High Speed Switch Characteristics:



## Return Loss

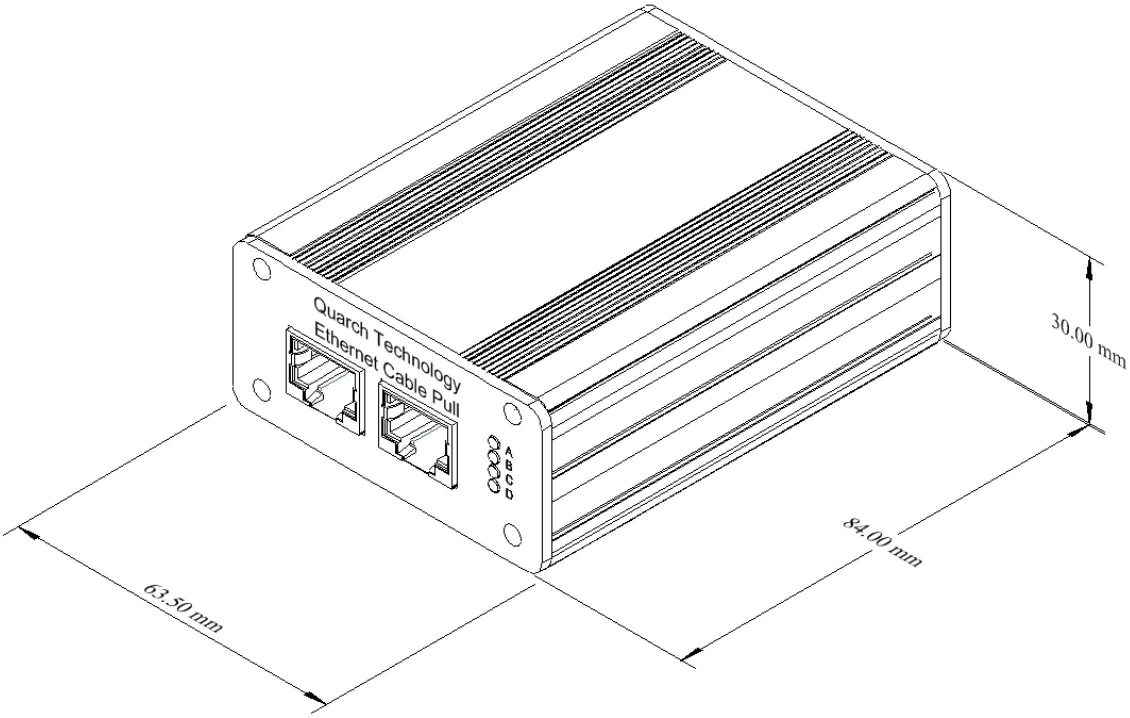



## Isolation

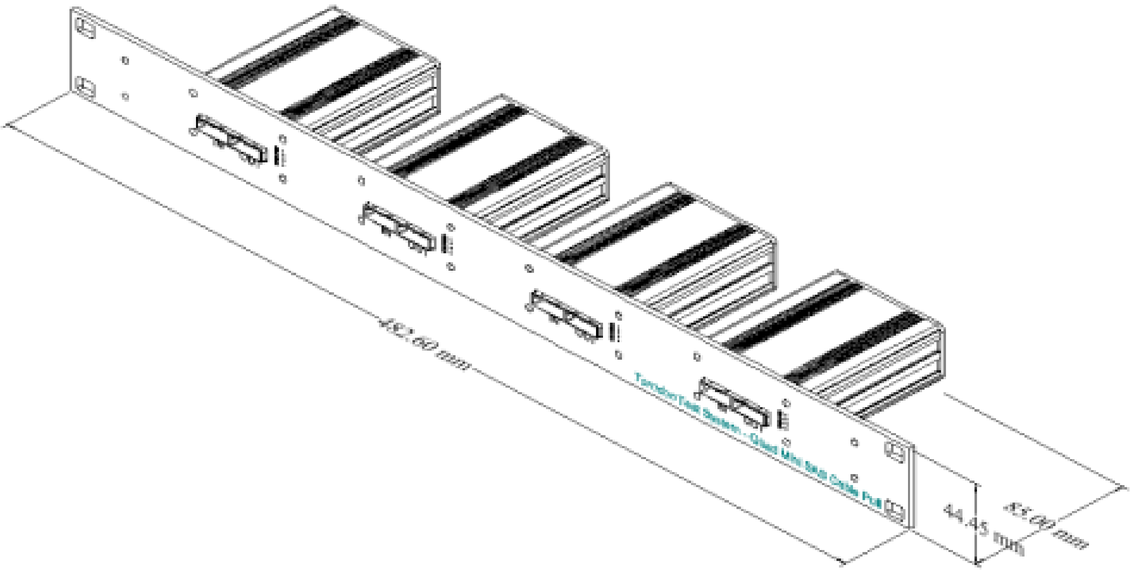


# Mechanical Characteristics

## QTL2022 Ethernet Cable Pull Module




## QTL1317 Quad Ethernet Cable Pull Module




(Mini SAS Modules shown)

## Module Versions

### QTL2022

This is an improved version, designed to also work with 10G Base T and E1 links

Cat6 cable runs are normally limited to 55m at 10G. This module does (due to the additional losses) limit the length slightly. Our testing indicates that total cable lengths up to 38m works without loss of speed. The precise maximum length may depend on your system, and should be evaluated before purchase

### Front Panel LEDs

The 4 LEDs on the front panel indicate the connection state of the signals. Each LED refers to one of the 4 data pairs in the Ethernet cable: A, B, C and D.

The light is green when all signals in the lane are connected, orange if some but not all signals are connected and off if all signals are disconnected.

## Control Interfaces

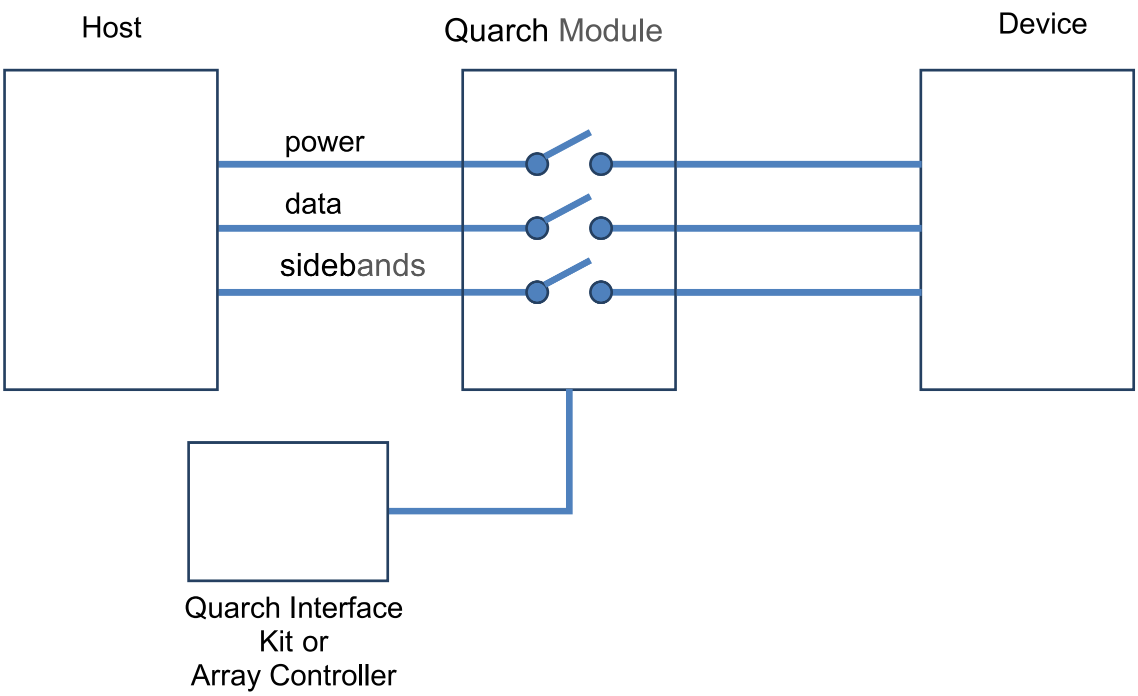
All Torridon modules are designed to be used with a Torridon Array Controller (QTL1461, QTL1079) or a single Torridon Interface Kit (QTL1260).

The control cable is an ultra-thin flex cable.

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

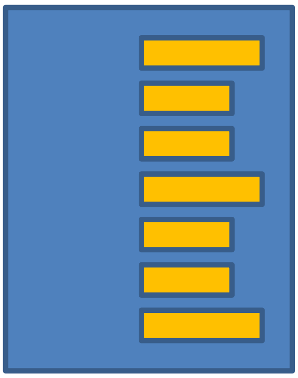
## Basic Concepts

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

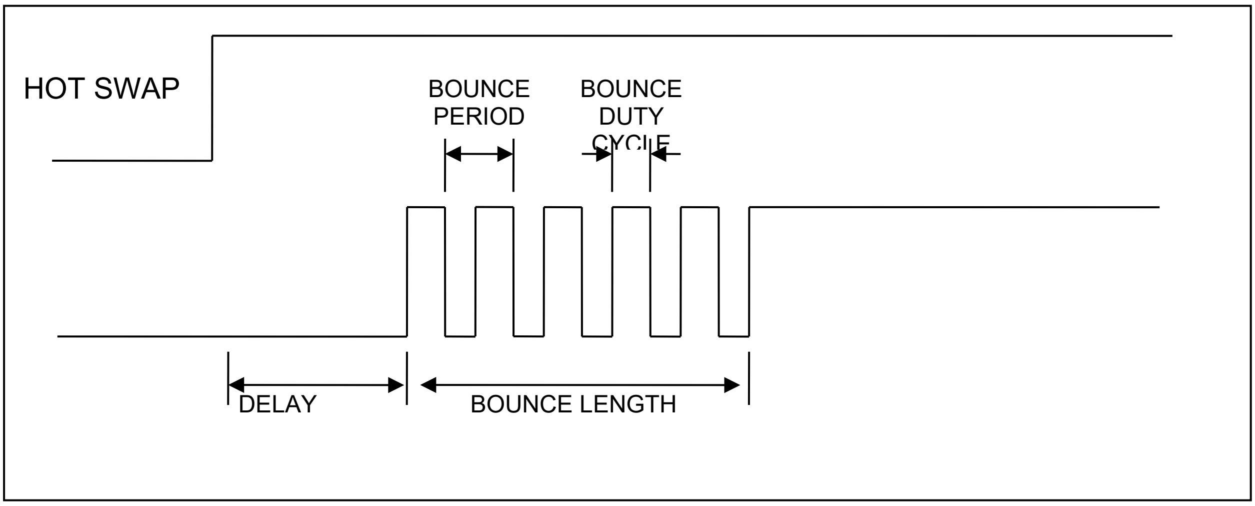



Each switch on the module is called a 'Signal' and can be programmed to follow one of six programmable delay and bounce profiles (called 'Sources'). This allows the user to sequence the signal connections in the cable in up to six programmable steps.

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.



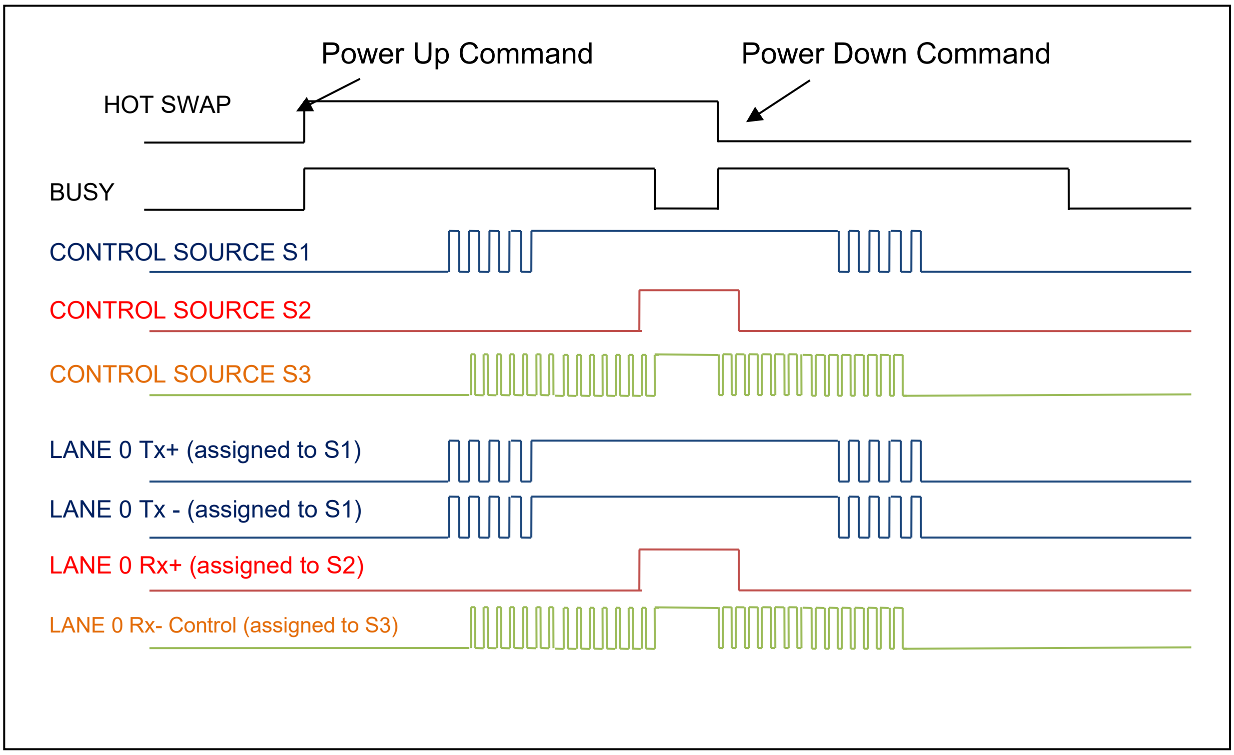
Each of the programmable delay and bounce profiles is called a control source, S1 to S6.  For each control source the user sets up a delay, and bounce parameters. Three special sources (S0, S7 and S8) are also provided as described in the table below.




*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

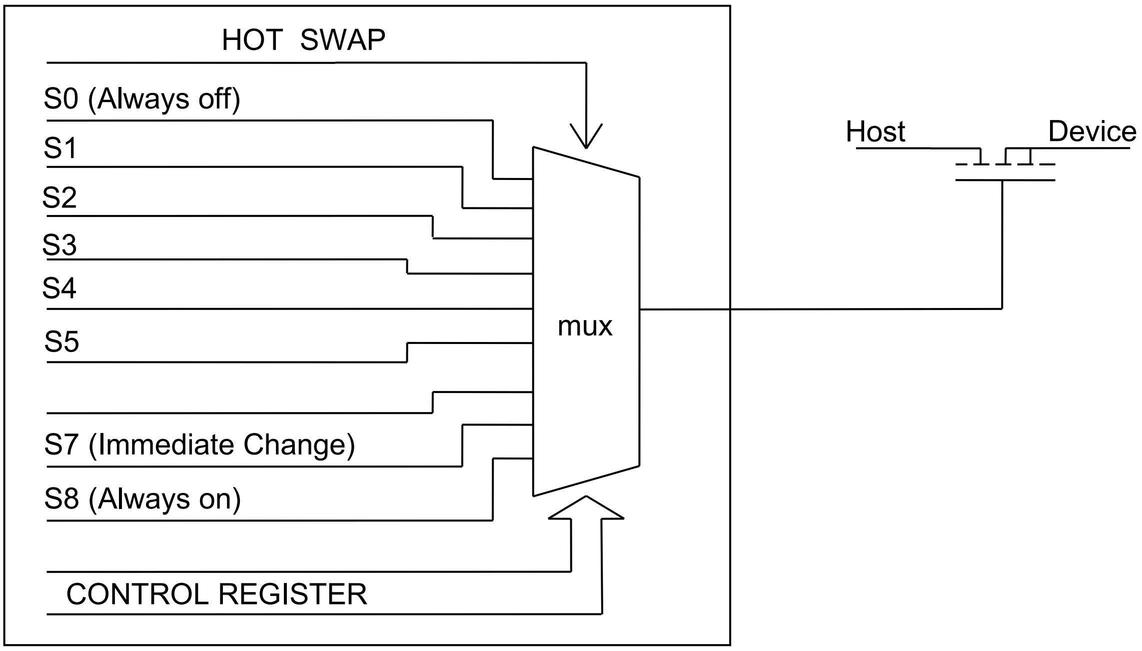
Each signal that is switched by the module is usually assigned to one of the 6 timed sources, S1 – S6. Each signal can also be assigned directly to 'always off' (source 0), 'immediate change' (source 7) or 'Always on' (source 8).

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 |

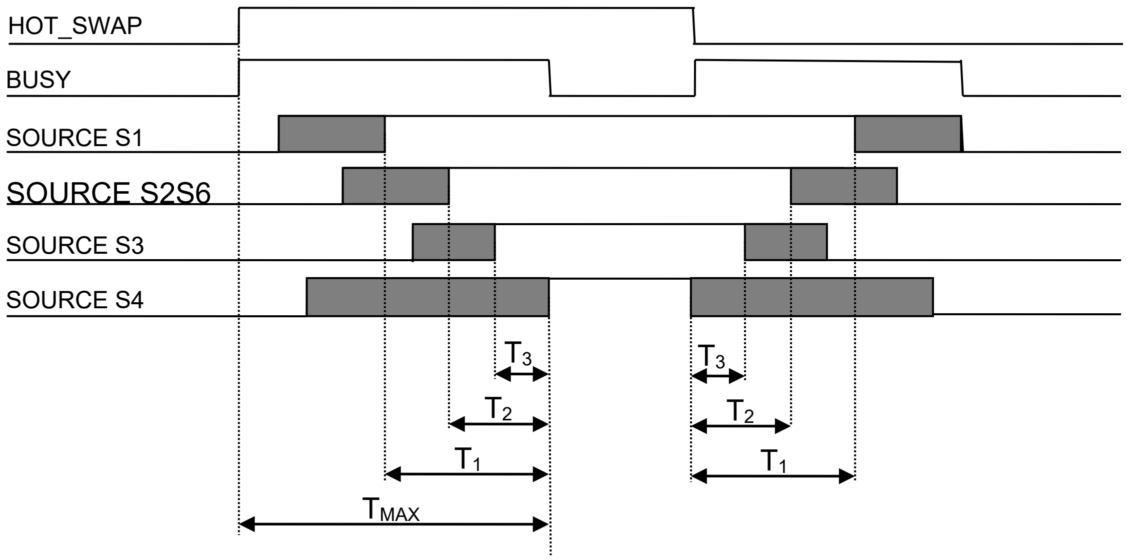
| | |
|---|---|
| 5 | Signal assigned to control source 5 |
| 6 | Signal assigned to control source 6 |
| 7 | Signal changes with HOT_SWAP state |
| 8 | Signal is always ON |




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:




Power Down Times are automatically generated by the module but can be calculated as follows:

$T_{MAX}$ = Largest Time Period

= the largest value from: (S1_DELAY + S1_BOUNCE_LENGTH),

(S2_DELAY + S2_BOUNCE_LENGTH),

up to

(S6_DELAY + S6_BOUNCE_LENGTH)

Power Down Delay $T_X$ = $T_{MAX}$ – ( $S_X$_DELAY + $S_X$_BOUNCE_LENGTH )

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.

## Pin Bounce Modes

Pin Bounce can be set in two ways:

### Constant Oscillation Frequency

- For **basic** firmware the Oscillation Time is set in one of two ranges:
  - 0 to 127 milliseconds in steps of 1mS
  - 130 milliseconds to 1.27 seconds in steps of 10mS
- For **high-resolution** firmware the Oscillation Time is set in one range:
  - 0-16,777,215uS, in steps on 1uS
  - Commands default to mS resolution but the user can add another unit as an additional parameter

- For **basic** firmware the Oscillation Period is for the pattern is set in one of two ranges:
  - 0 to 1.27 milliseconds in steps of 10uS
  - 2 to 127 milliseconds in steps of 1mS
- For **high-resolution** firmware the Oscillation Period is set in one range:
  - 0-1,677,721,500nS, in steps on 100nS
  - Commands default to uS resolution but the user can add another unit as an additional parameter

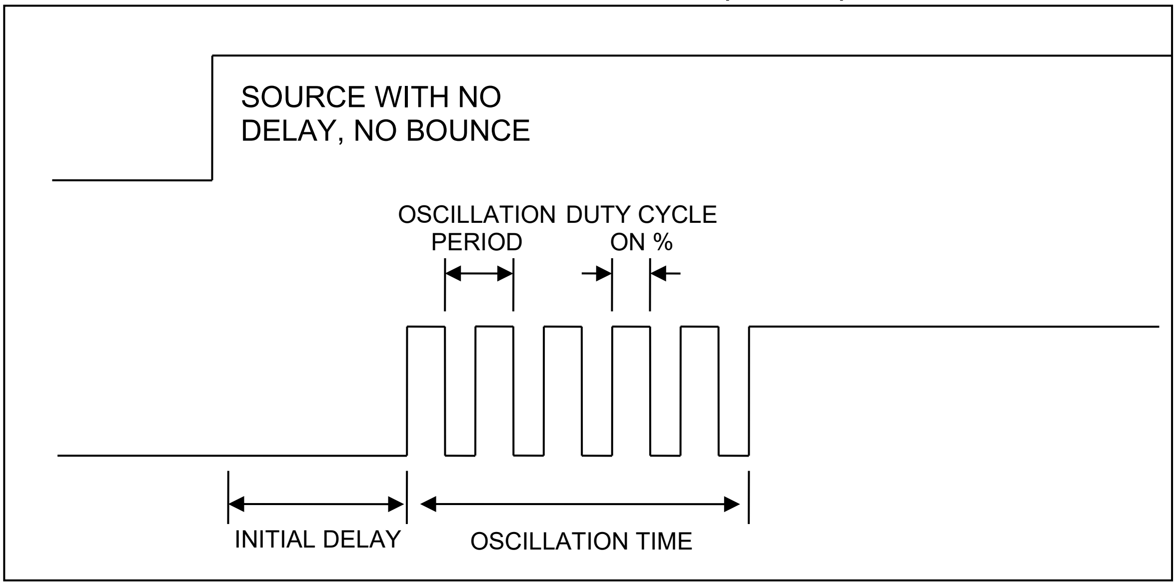
The Duty cycle (On %) is set as a percentage value in the range 0-100%.

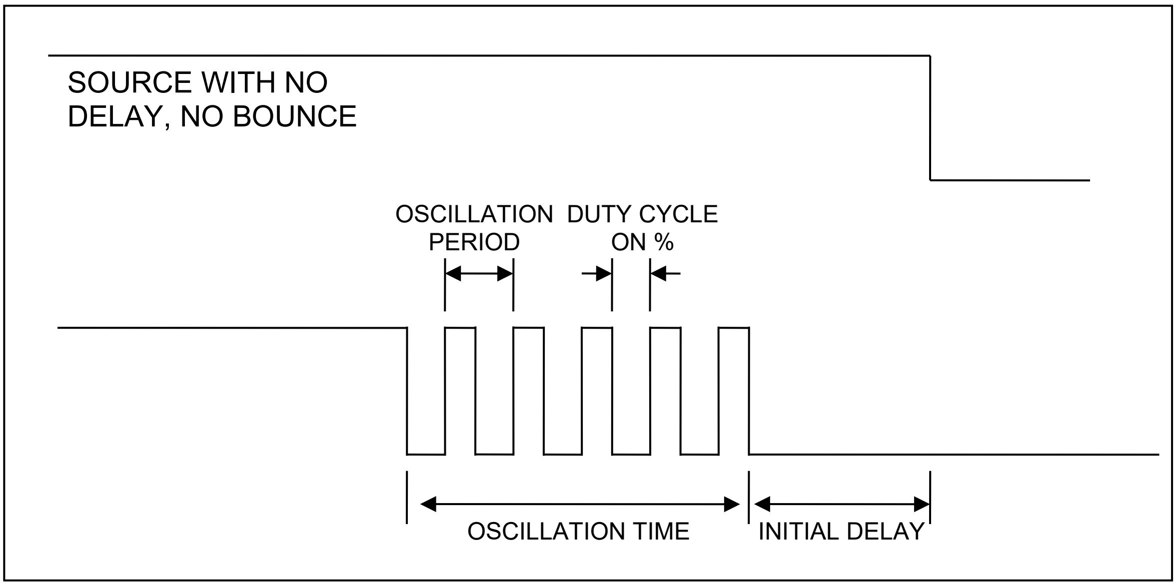
A value of 0% would leave the source off for the duration of the Oscillation Time.

A value of 50% provides a symmetrical square wave as shown below and is the default.

A value of 100% would turn the signal on for the duration of Oscillation Time.

*Basic bounce behavior on power up*




*Basic bounce behavior on power down*

### User Pin-Bounce (Custom Oscillation)

User Pin bounce allows the user to set a custom pattern of up to 112 bits which will be executed by the module instead of standard pin bounce. A custom pattern of alternating 1's and 0's would create a square wave just like the default basic bounce mode.

The executed pattern is determined by a number of factors:

- For **basic** firmware the Oscillation Time is set in one of two ranges:
  - 0 to 127 milliseconds in steps of 1mS

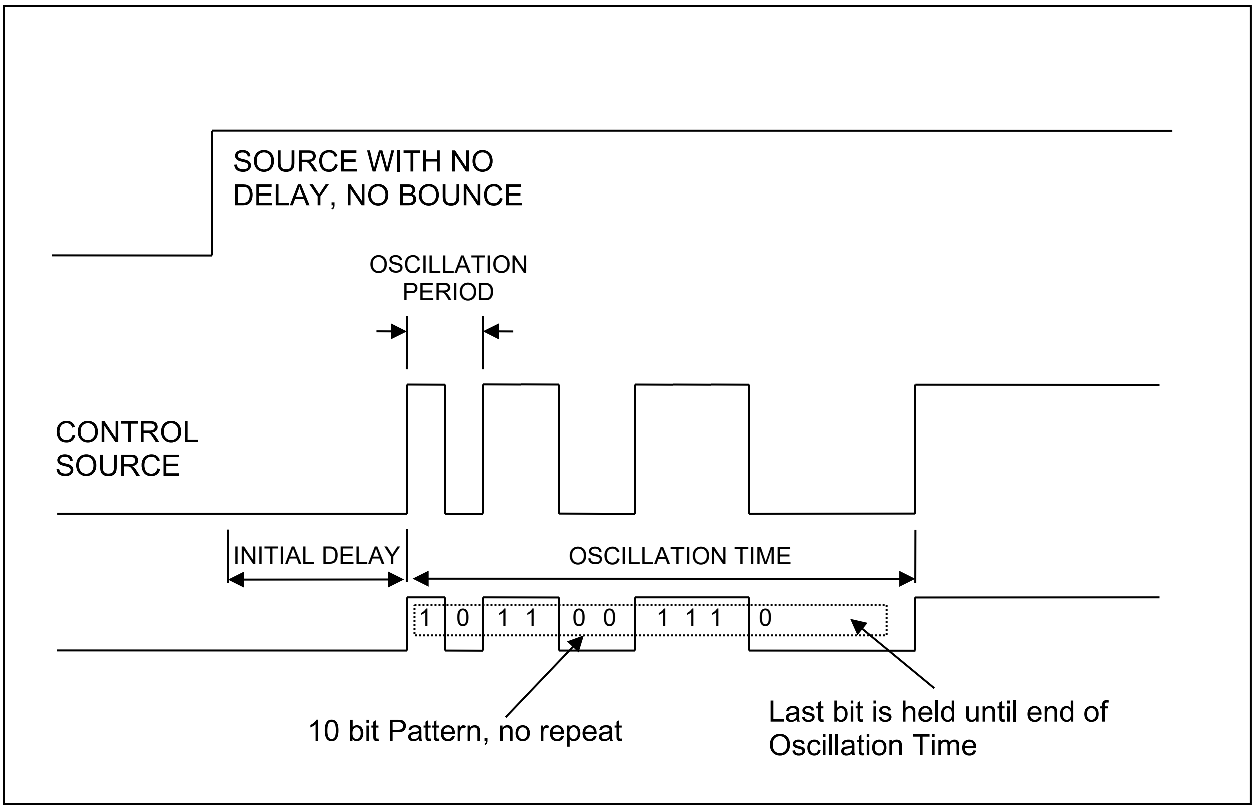
- 130 milliseconds to 1.27 seconds in steps of 10mS
- For **high-resolution** firmware the Oscillation Time is set in one range:
  - 0-16,777,215uS, in steps on 1uS
  - Commands default to mS resolution but the user can add another unit as an additional parameter
- For **basic** firmware the Oscillation Period is for the pattern is set in one of two ranges:
  - 0 to 1.27 milliseconds in steps of 10uS
  - 2 milliseconds to 127 milliseconds in steps of 1mS
- For **high-resolution** firmware the Oscillation Period is set in one range:
  - 0-1,677,721,500nS, in steps on 100nS
  - Commands default to uS resolution but the user can add another unit as an additional parameter
- The Pattern length may be set:
  - 1-112 bits
  - Choose to repeat pattern or sit on last bit until the end of Oscillation Time.

Custom Patterns can get confusing very quickly, the best way to make sure that the power down sequence is the opposite of power up, is to make sure that:

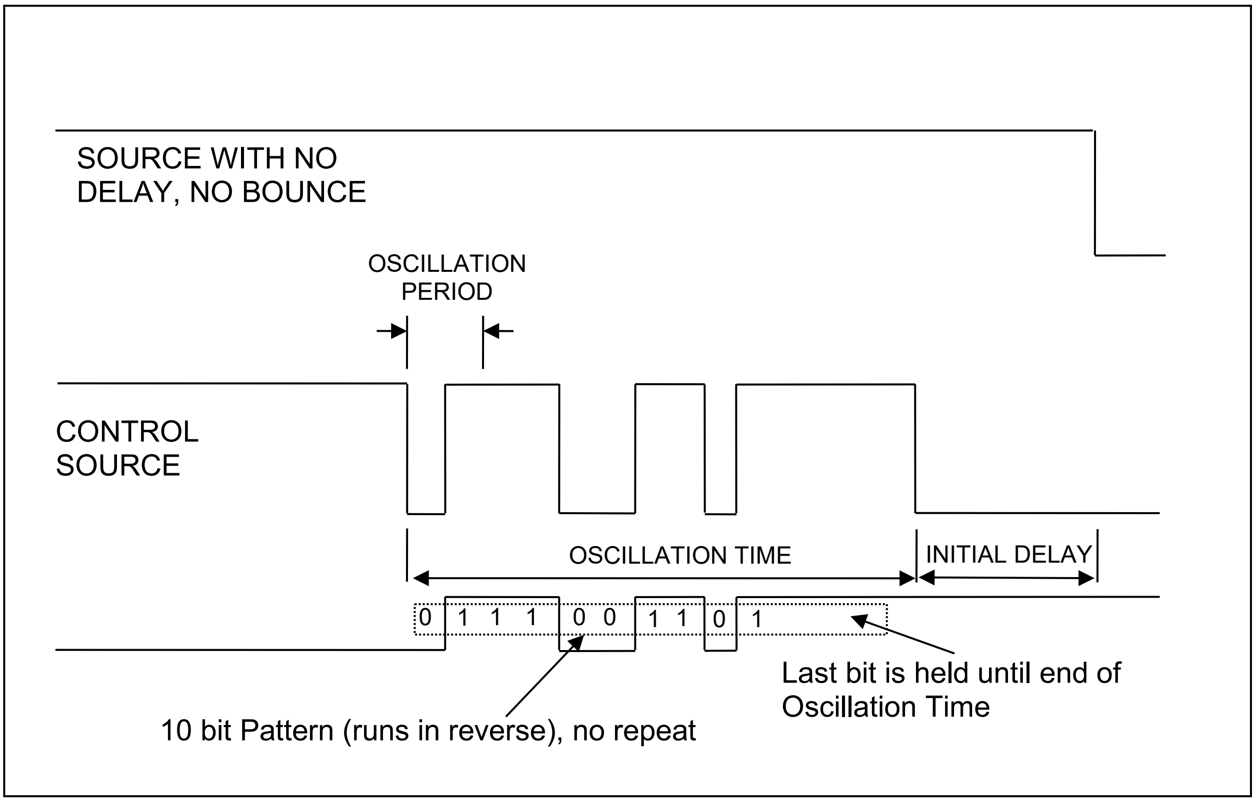
(Bit length * Number of Bits  = Oscillation Time)

so that the pattern ends exactly at the end of Oscillation time. The SOURce:[x]:BOUNce:PATtern:SETup command does this automatically.

Custom patterns run in reverse on a power down, please see the following diagrams for examples of the same pattern being run on a power up and power down situation.




*User bounce behavior on power up*




*User bounce behavior on power down*

## Glitch Control

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

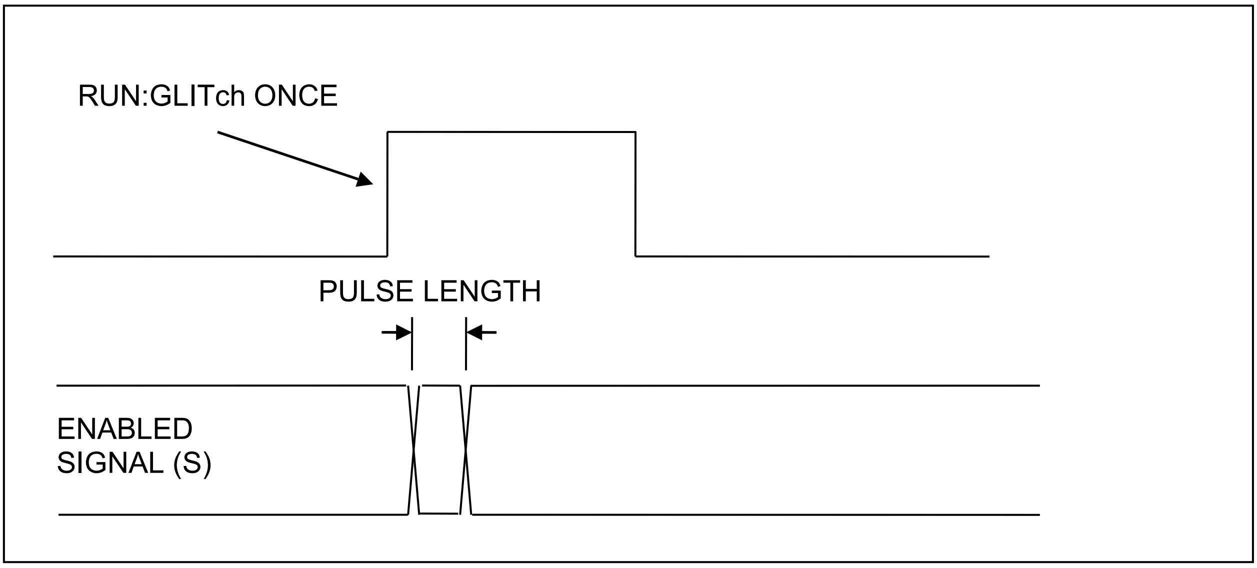
Each Signal Control register contains a "**GLITCH:ENABLE**" bit which determines whether the glitch logic will affect that signal. The setting, defaults to off, so any glitches will have no effect unless explicitly set to do so.

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



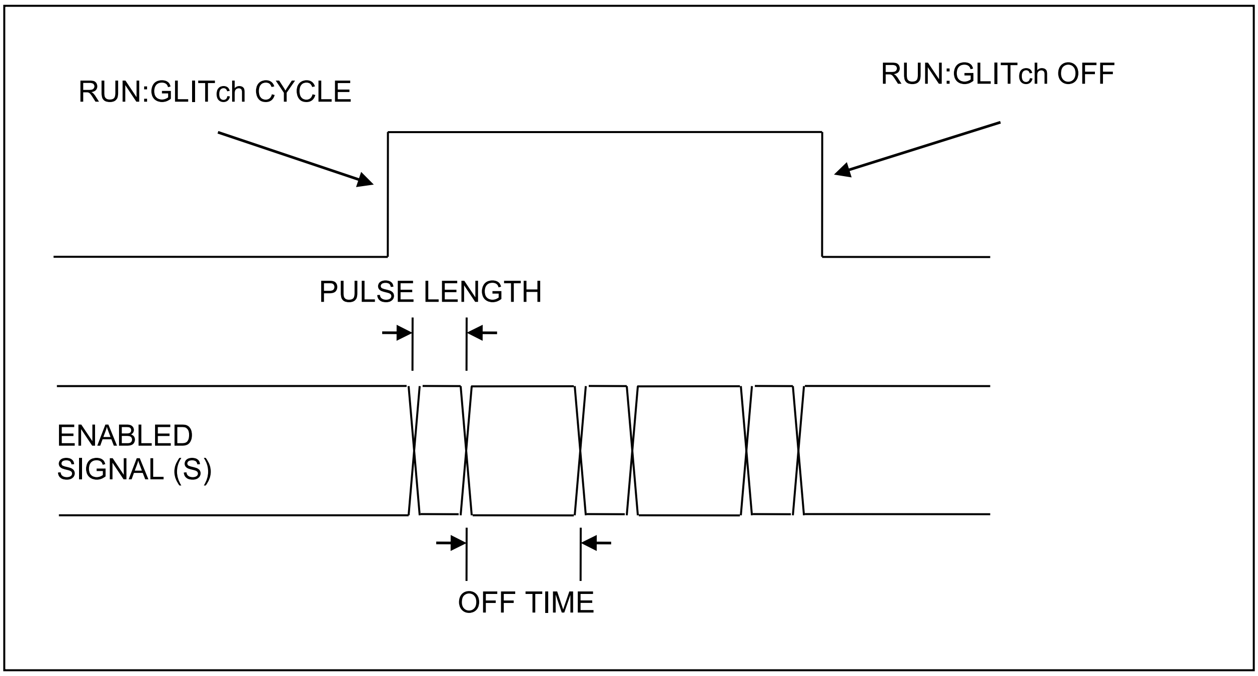

A single glitch is generated when the **RUN:GLITch ONCE** command is executed.

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

PULSE LENGTH = **GLITch:MULTiplier** x **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



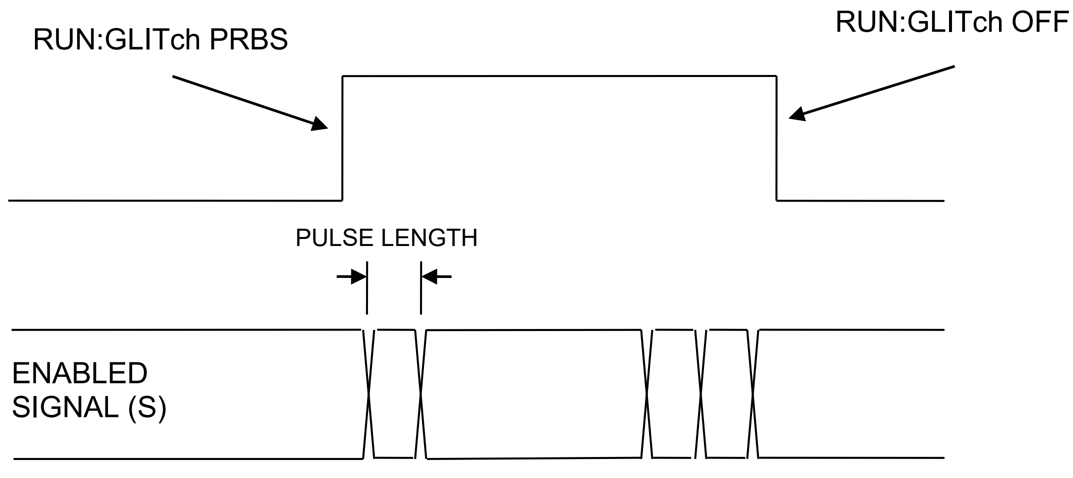

A sequence of glitches is generated when the **RUN:GLITch CYCLE** command is executed, and continues until **RUN:GLITch OFF** is executed.
The length of the glitch is determined by using the **GLITch:SETup** command or the **GLITch:MULTiplier** and **GLITch:LENgth** commands:

PULSE LENGTH = **GLITch:MULTiplier** x **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:

OFF TIME = **GLITch:CYCle:MULTiplier** x **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:

PULSE LENGTH = **GLITch:MULTiplier** x **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.

## 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 |
|---|---|---|
| **MEASure:VOLTage:SELF 1v2?** | Returns the voltage of the modules internal 1.2v power rail | 64mV / 5% |
| **MEASure:VOLTage:SELF 3v3?** | Returns the voltage of the modules internal 3.3v power rail | 64mV / 5% |
| **MEASure:VOLTage:SELF 5v?** | Returns the voltage of the modules internal 5v power rail | 64mV/ 5% |

## Default Startup State

On power up or reset, the control modules enter a default state. On the cable pull 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 all pins immediately, without delays or pin-bounce.

| 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 |
|---|---|
| A_PL | Source 1 |
| A_MN | Source 1 |
| B_PL | Source 1 |
| B_MN | Source 1 |
| C_PL | Source 1 |
| C_MN | Source 1 |
| D_PL | Source 1 |
| D_MN | 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 ':' (`LEVel1:LEVel2:LEVel3`)
- 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.

**MEAS:VOLTage:SELF [1.2?|3v3?|5v?]**

Returns the self test voltages. These are measurements of voltage rails required for correct operation of the module. The values are returned in the form “5000mV”

**SOURce:[1-6|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:[1-6|ALL]:DELAY [#ms] [#Unit*]**

**SOURce:[1-6|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:[1-6|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:[1-6|ALL]:BOUNce:LENgth [#ms] [#Unit*]**

**SOURce:[1-6|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:[1-6|ALL]:BOUNce:DUTY [#%]**

**SOURce:[1-6|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:[1-6|ALL]:BOUNce:MODE [SIMPLE|USER]**

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

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

**SOURce:[1-6|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:[1-6|ALL]:BOUNce:PATtern:READ [0xAAAA]**

Reads a word of the custom bounce pattern

0xAAAA is the address (for example 0x0002)

**SOURce:[1-6|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:[1-6|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:[1-6|ALL]:STATE [ON|OFF]**

**SOURce:[1-6|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:[1-6]:BOUNce:PATtern:LENgth [#bits]**

**SOURce:[1-6]: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:[1-6]:BOUNce:PATtern:REPeat [ON|OFF]**

**SOURce:[1-6]:BOUNce:PATtern:REPeat?**

Sets the custom pattern repeat flag. This is used when the current custom bounce pattern is shorter that 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:[1-6]: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]:SETup [#num]**
**SIGnal:[SIG_NAME|ALL]:SOURce [#num]**

Sets a given signal to a numbered timing source (0-8). SIGNAL_NAME is one of the signals/groups from 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 in on, the signal will be glitched whenever the glitch logic is in use. Multiple signals may be set for glitch at the same time.

**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 [MULTIPILER_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 [MULTIPILER_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.

# 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". 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 group names

**Signals**

A_PL
A_MN
B_PL
B_MN
C_PL
C_MN
D_PL
D_MN

**Signal Groups**

ALL (Allows change of all signals at the same time)

PAIR_A (Affect all signals relating to the 'A' differential pair)
PAIR_B
PAIR_C
PAIR_D