

Quarch Technology Ltd

Torridon M.2 M-Key Vertical Card Module

Technical Manual

For use with:

QTL2034 - Torridon M.2 M-Key Vertical Card Module

Using Quarch firmware version 4.000 and above

Monday, 24 October 2016



Change History

1.0	12 October 2016	Initial Release
-----	-----------------	-----------------

Contents

Change History	2
Contents	3
Introduction	4
Technical Specifications	6
Switching Characteristics	6
Mechanical Characteristics	7
Control Interfaces	8
Basic Concepts	9
Signal Configuration	11
Power Up vs. Power Down Timing.....	13
Pin Bounce Modes	14
Constant Oscillation Frequency	14
User Pin-Bounce (Custom Oscillation)	15
Glitch Control	18
Glitch Once.....	18
Glitch Cycle.....	20
Glitch PRBS.....	21
Signal Driving.....	22
Basic Example - PERST.....	23
Glitch Example	23
Voltage Measurements.....	24
Default Startup State	25
Controlling the Module	26
Terminal Command Set	27
Appendix 1 – Signal Naming	38
Signals	38
Signal Groups	39

Introduction

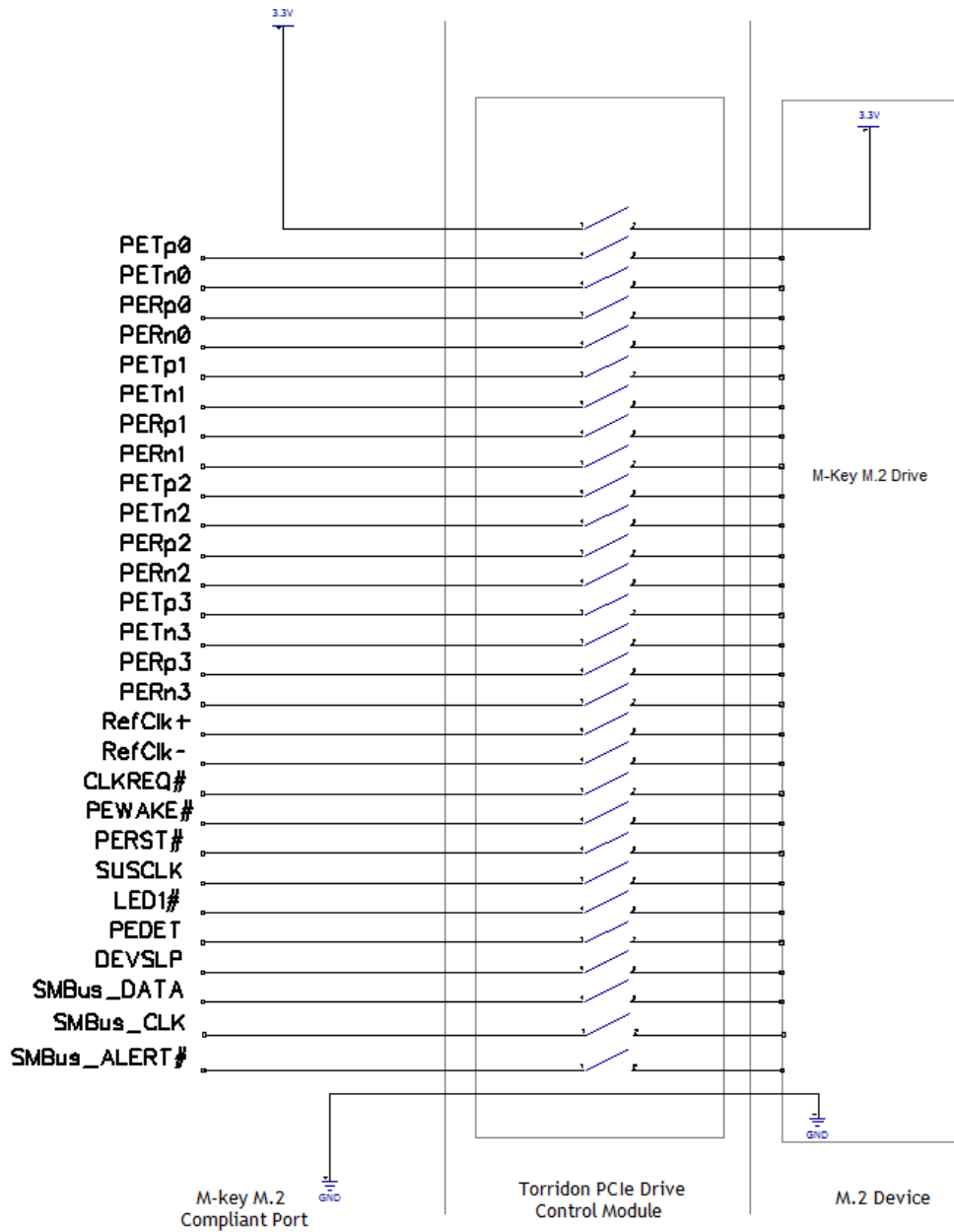
The Torridon M.2 M-Key Vertical Card Module allows remote switching of power, drive presence, PCIe data pins, and sideband signals to an M.2 M-Key device whilst plugged into a host system.

The vertical orientation of this module allows it to fit into almost any M.2 slot, with the device under test sitting vertically.

Each set of pins can be individually switched, allowing complete control over the power up sequence of a drive. The switches can be sequenced at precise timings to simulate a hot-swap event, including pin bounce. Individual pins can be broken or glitched at any time to simulate a fault in the system.

The M.2 module switches the high speed PCIe data lines at speeds up to Gen 3 (8GT/s). This greatly increases the number of faults that can be injected into the system and produces a more comprehensive hot-swap.

A number of sideband signals can also be driven high or low, for additional testing.



Schematic - Switched signals

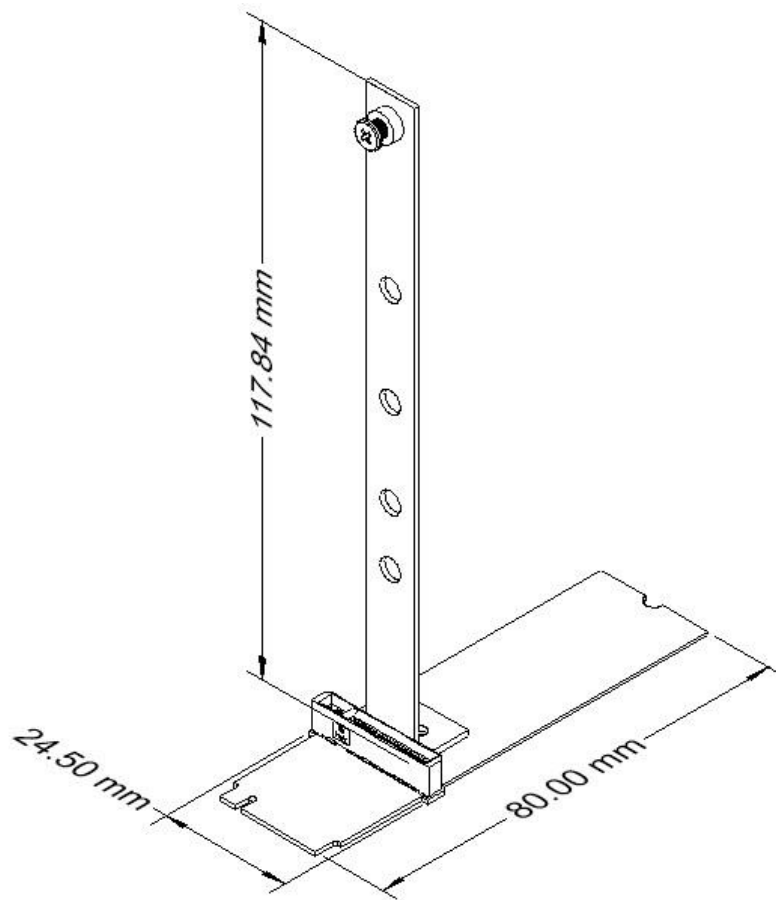
Technical Specifications

Switching Characteristics

M.2 Connector Pin	Description	Switching Action
1,3,9,15,21,27,33,39,45,51,57,71,73,75	Ground Pins	All connected to digital ground on the module
43,41,49,47,31,29,37,35,19,17,25,23,5,7,11,13	PCIe Data Signals and Reference Clocks	Each signal is individually switched by a high speed RF switch
2,4,12,14,72,70,18,74,16	3V3 Power	Connected together and switched by 5.6A power FET
52,53,55	CLKREQ#,CLK+,CLK-	Individually switched by bilateral analog switch
68	SUSCLK	Individually switched by bilateral analog switch
38	DEVSLP	Switched by bilateral analog switch
69	PEDET	Switched by bilateral analog switch
54	PEWAKE#	Switched by bilateral analog switch
50	PERST#	Switched by bilateral analog switch
10	Activity LED	Switched by bilateral analog switch
40,42,44	SM Bus Clock, Data and Alert	Individually switched by bilateral analog switch

Mechanical Characteristics

The module is designed to fit into a 2280 M-keyed M.2 slot. The module is 2.5mm wider than the allowed 22mm width of an M.2 card, with general clearances, this should not usually be a problem. The module ships with a support bracket that supports M.2 devices up to 110mm in length. This bracket may be cut shorter if required.



Control Interfaces

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

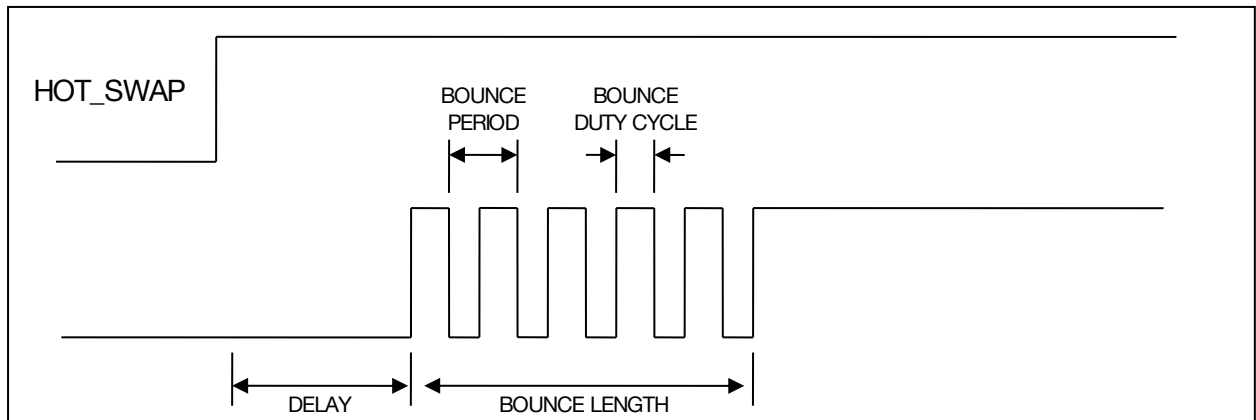
The control cable is an ultra-thin flex cable.

Control Interface	Form Factor	Torridon Ports	Control Methods Available	Interfaces
QTL1079 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
QTL1461 4 Port Array Controller	160x165x53mm Enclosure 1U Enclosure also available	4 ports on front	Terminal Scripting TestMonkey 2 GUI	Serial via RJ45 Ethernet USB
QTL1461 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 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.

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.

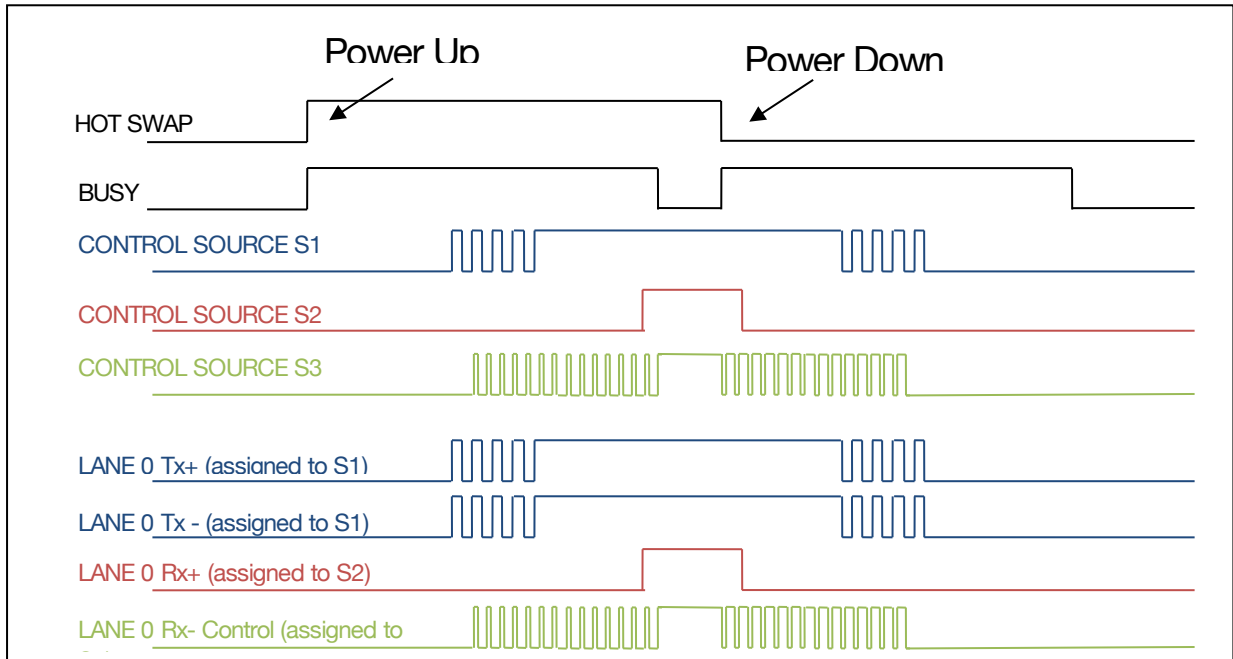


Control Source Parameters for a power up event (Basic Pin Bounce)

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:



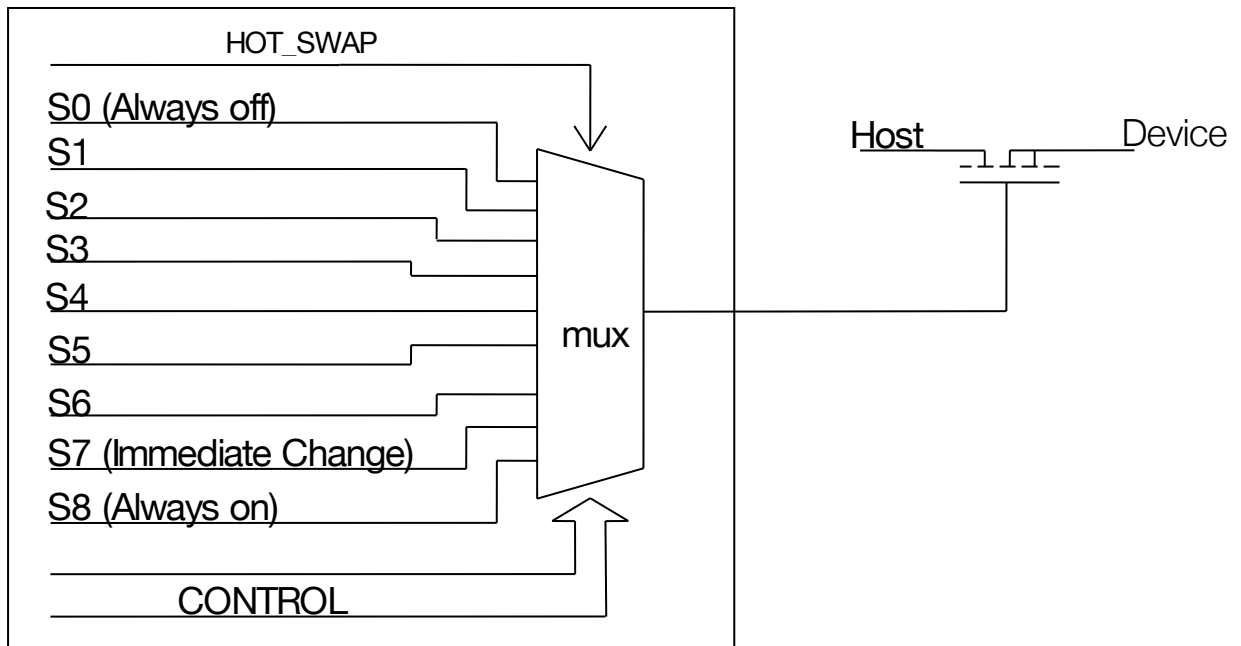
Signal Configuration

Each signal that is switched by the module is usually assigned to one of the six 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:

SIGna1:[name]:SOURce [Source#]

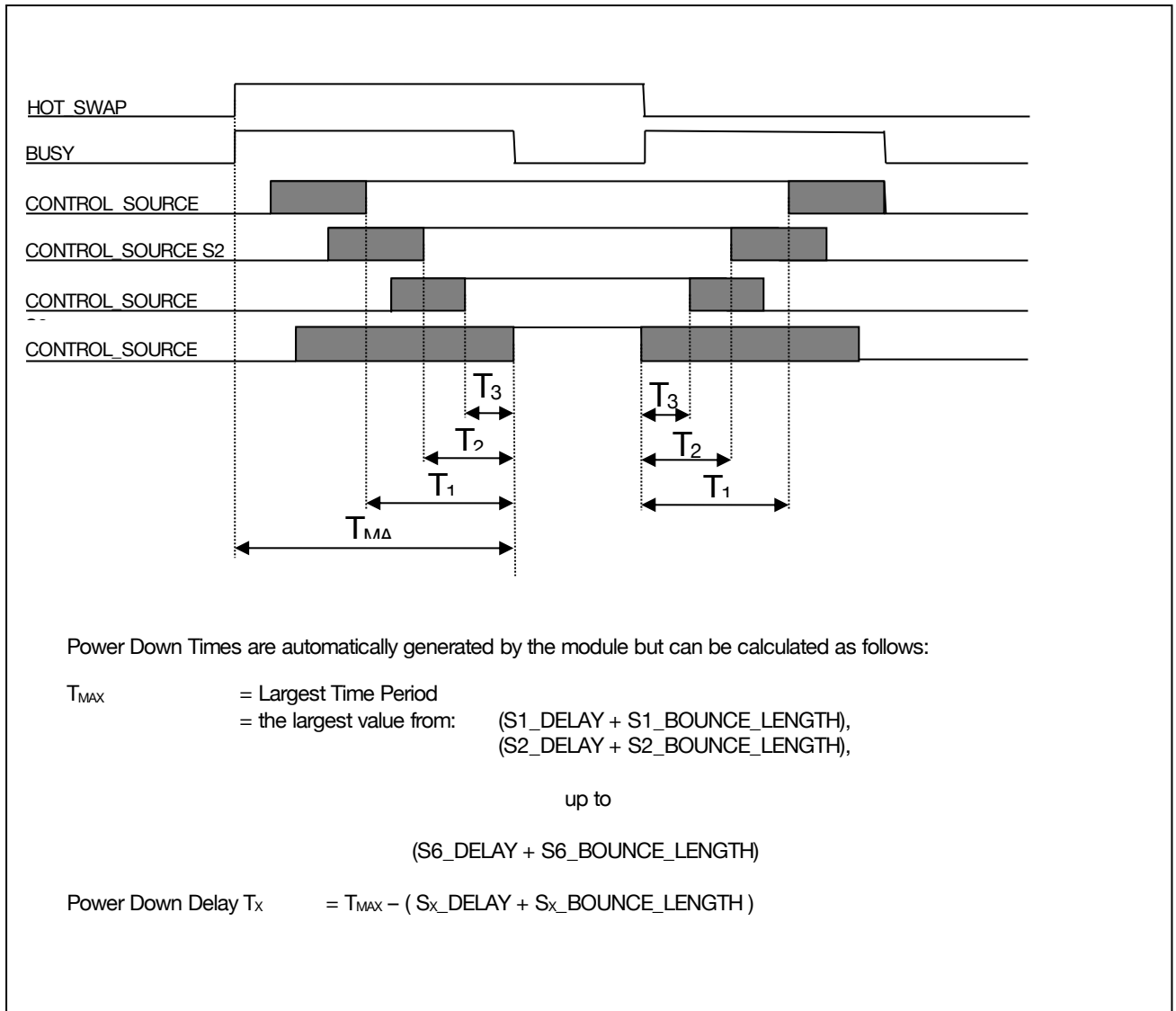
Source Number	Description
0	Signal is always OFF
1	Signal assigned to control source 1
2	Signal assigned to control source 2
3	Signal assigned to control source 3
4	Signal assigned to control source 4
5	Signal assigned to control source 5
6	Signal assigned to control source 6
7	Signal changes with HOT_SWAP 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:



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

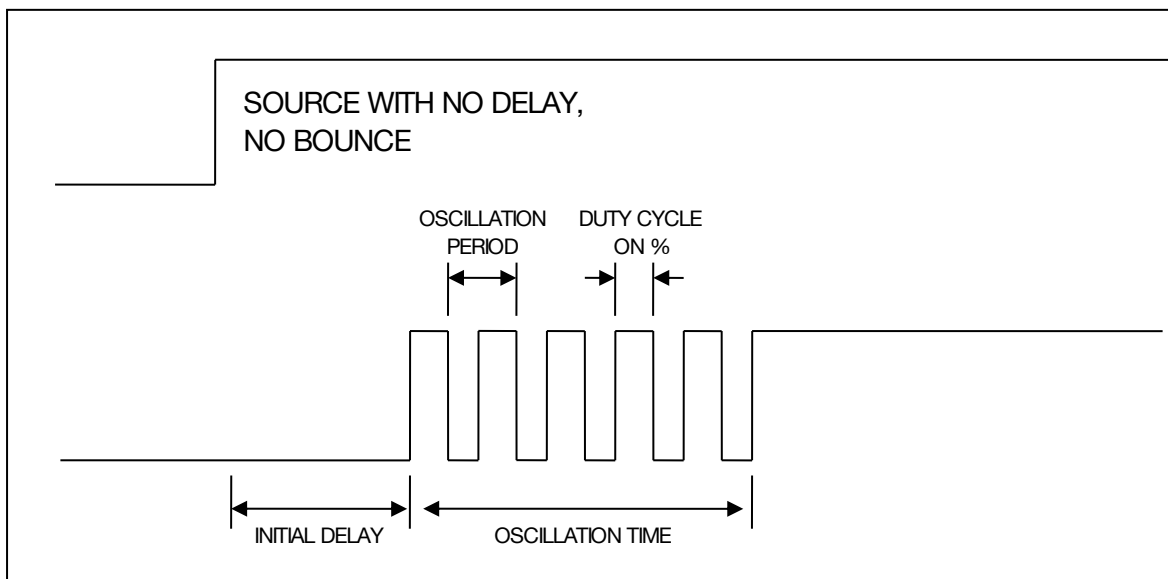
- The Oscillation Time is set in one of two ranges:
 - 0-127 milliseconds in steps of 1mS
 - 0-1.27 seconds in steps of 10mS
- The Oscillation Period is for the pattern is set in one of two ranges:
 - 0-1.27 milliseconds in steps of 10uS
 - 0-127 milliseconds in steps of 1mS

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

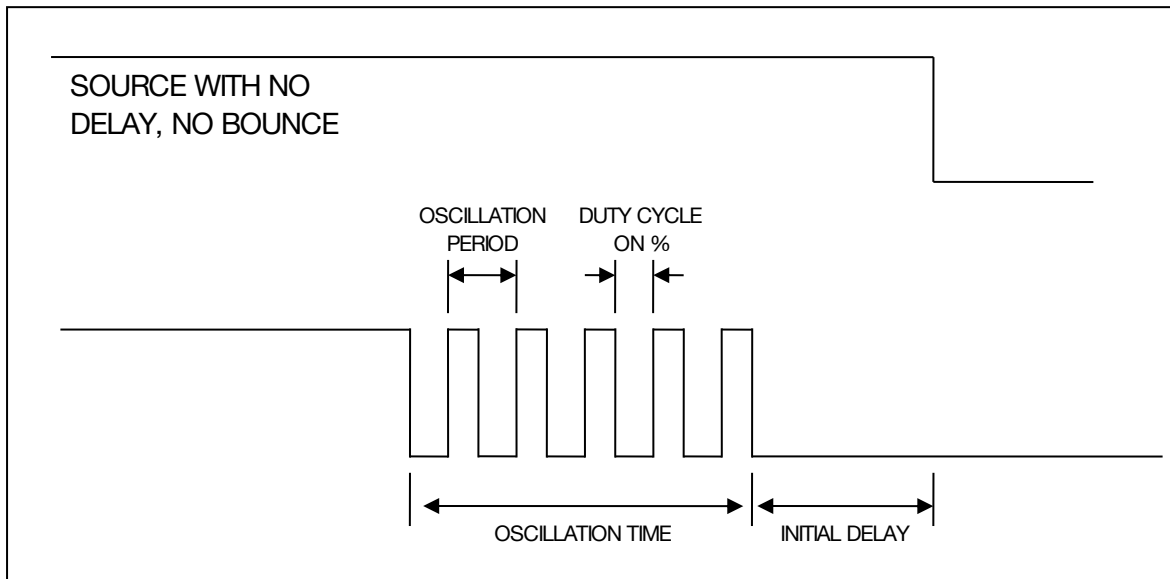
A value of 0% would leave the source off for the duration of the Oscillation Time.

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:

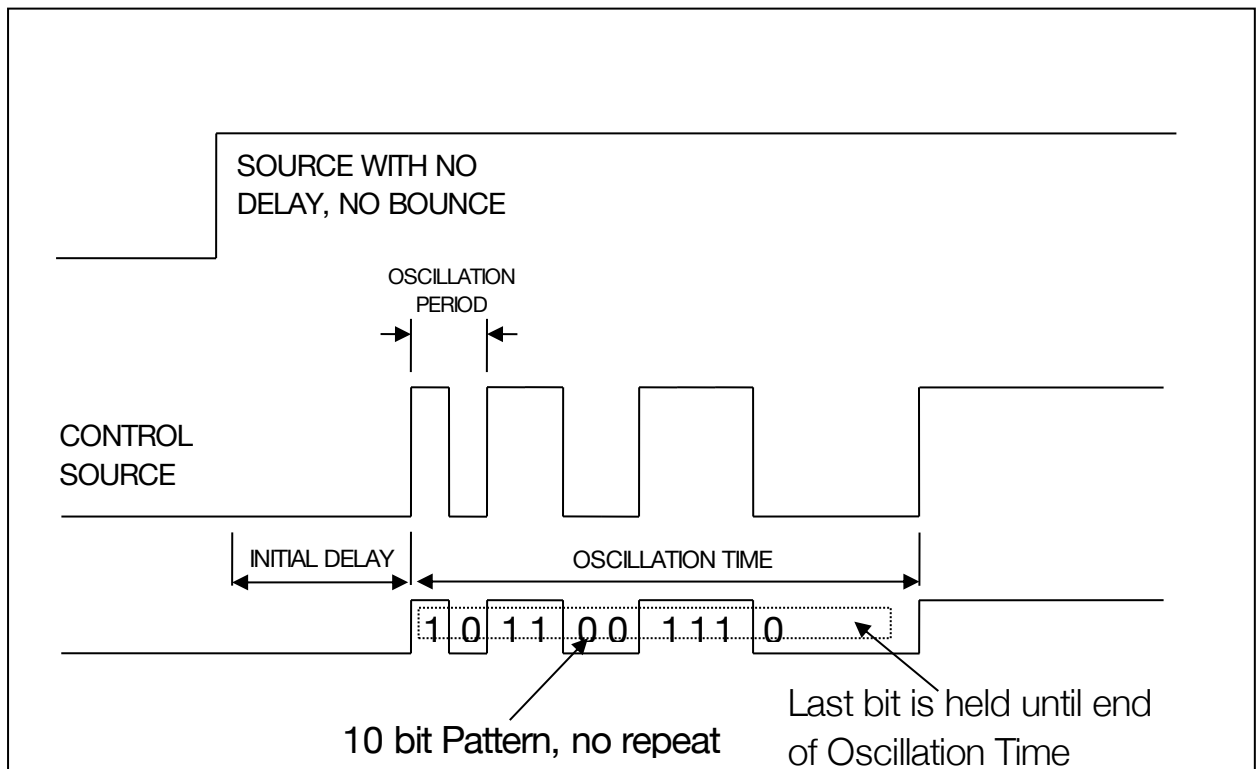
- The Oscillation Time is set in one of two ranges:
 - 0-127 milliseconds in steps of 1mS
 - 0-1.27 seconds in steps of 10mS
- The Oscillation Period is for the pattern is set in one of two ranges:
 - 0-1.27 milliseconds in steps of 10uS
 - 0-127 milliseconds in steps of 1mS
- The Pattern length may 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:

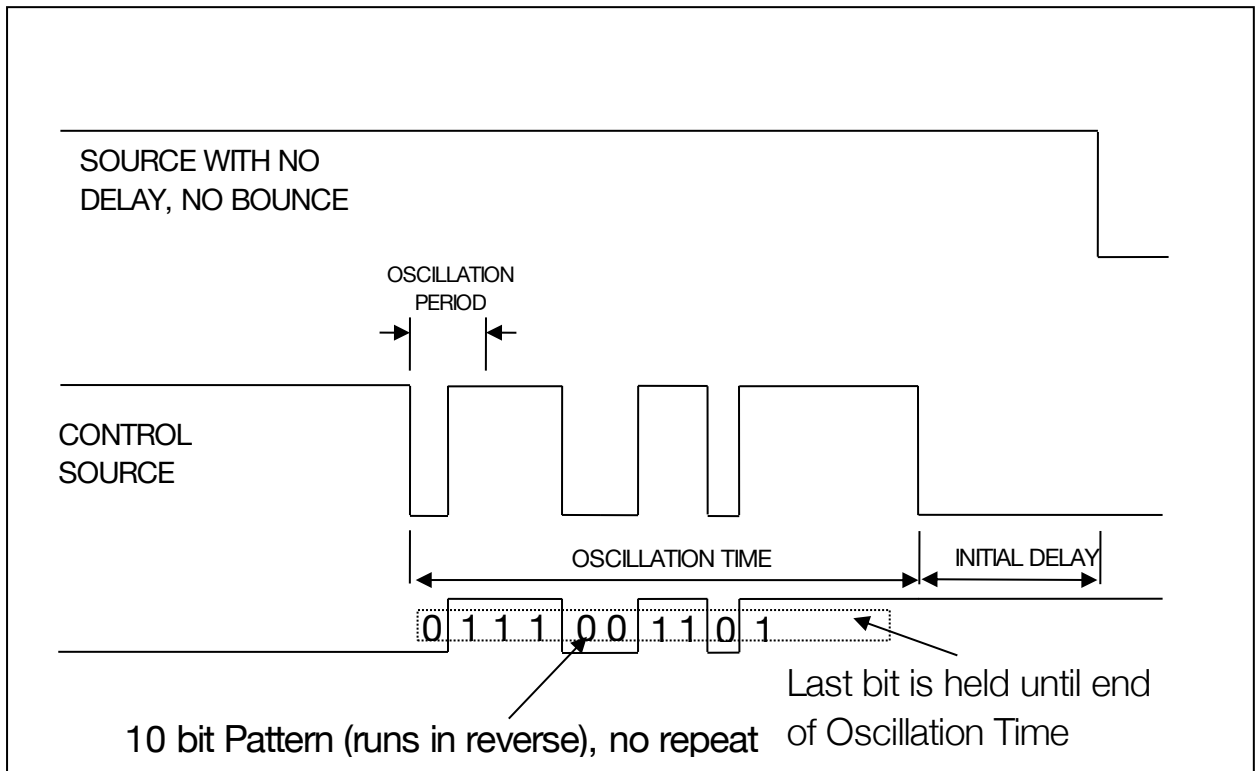
(Bit length * Number of Bits = Oscillation Time)

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.

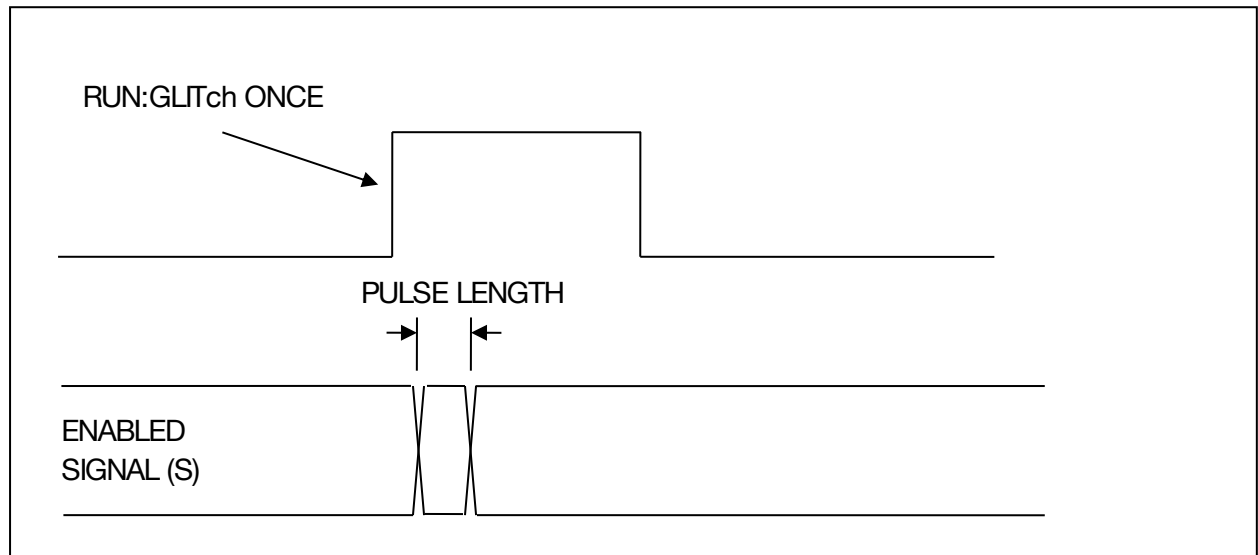
Each Signal Control register contains a “**GLITCH:ENABLE**” bit which determines whether the glitch logic will affect that signal. The setting defaults to off, so any glitches will have no effect unless explicitly set to do so.

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



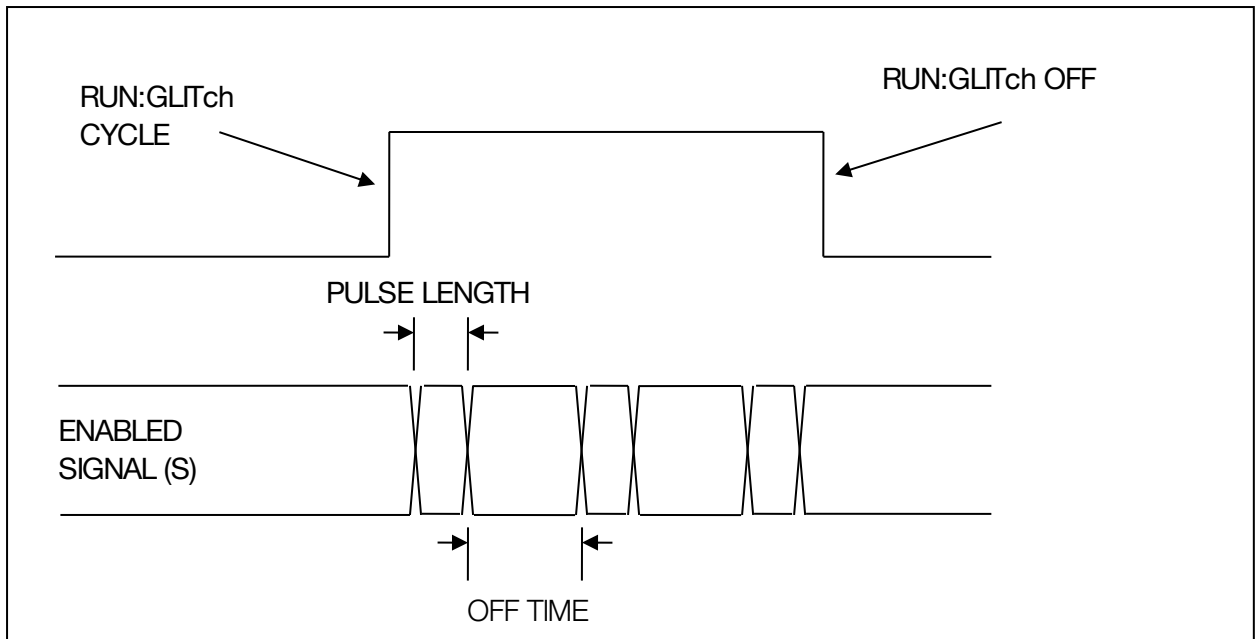
A single glitch is generated when the **RUN:GLITCh ONCE** command is executed.

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** × **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.

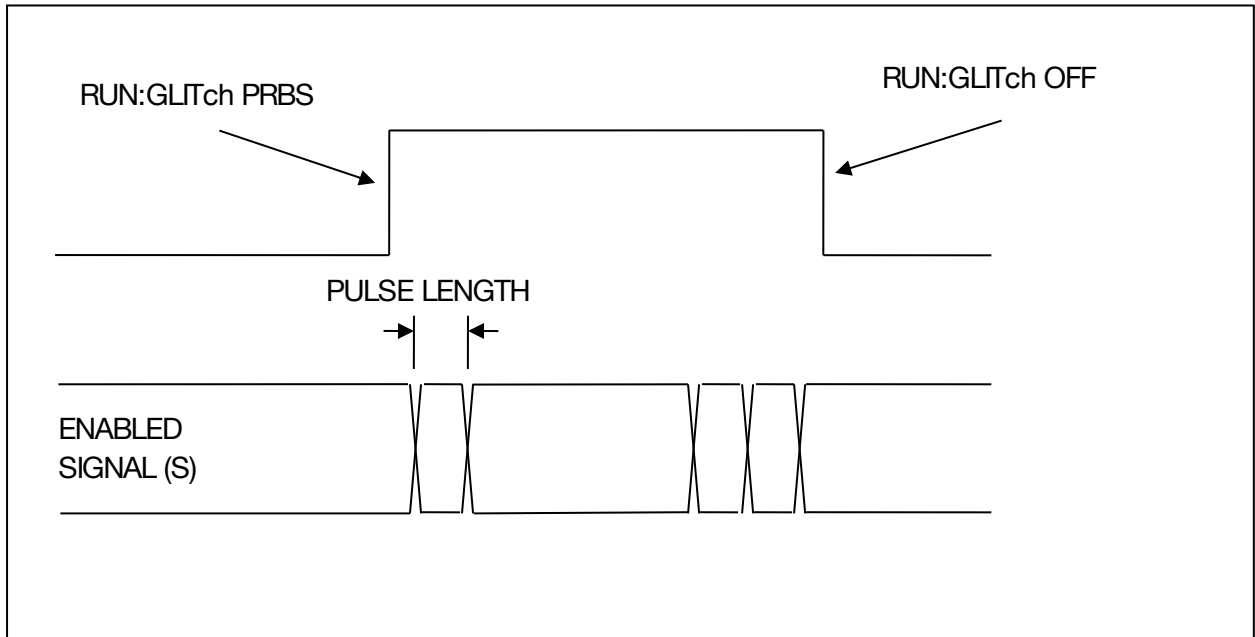
The length of the glitch is determined by using the **GLITCh:SETup** command or the **GLITCh:MULTiplier** and **GLITCh:LENGth** commands:

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

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:

- DEVSLEEP
- PERST
- PEWAKE
- PEDET
- CLKREQ

For these signals, the user can specify a behavior using the **SIGna1:[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:

Signal Name	Signal Type	Host Side Behavior		Device Side Behavior	
		High	Low	High	Low
DEVSLEEP	Open Drain output from Host	Not Driven	Not Driven	Not Driven	Driven Low
PERST	Push/Pull output from Host	Not Driven	Not Driven	Driven High	Driven Low
PEWAKE	Push/Pull output from Host	Driven High	Driven Low	Not Driven	Not Driven
PEDET	Open Drain from Module	Not Driven	Driven Low	Not Driven	Not Driven
CLKREQ	Bidirectional Open Drain	Not Driven	Driven Low	Not Driven	Driven Low

Basic Example - PERST

To issue a fundamental reset to the device under test:

PERST are active low signals 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.

```
>SIGna1:PERST:DRIVE CLOSED LOW
```

(Line is driven low: reset is asserted)

```
>SIGna1:PERST:DRIVE CLOSED NONE
```

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

Glitch Example

If we want to assert EPE_RST_# 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.

```
>SIGna1:PERST:GLITCh:ENABle ON
```

```
>SIGna1:PERST:GLITCh:SETup 500us 2
```

```
>SIGna1: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.

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 3v3?	Returns the voltage of the module's internal 3.3v rail	12mV / 3%
MEASure:VOLTage:SELF 5v?	Returns the voltage of the module's internal 5v power rail	15mV / 3%
MEASure:VOLTage 3v3in?	Returns the voltage of the 3V3 power pins on the backplane (unswitched) side of the module	46mV / 3%
MEASure:VOLTage 3v3out?	Returns the voltage of the 3v3 power pins on the drive (switched) side of the module	46mV / 3%

Default Startup State

On power up or reset, the modules enter a default state. To make the module as easy to use as possible, the default state is a ‘standard’ hot-swap scenario with preset source and signal settings such that the “**run:power up**” command will immediately power up the drive without needing any initial setup.

The default hot-swap scenario will connect long pins, then short pins, with a 25mS delay between lengths. All sources are enabled.

Source Number	Source Enabled	Initial Delay
1	YES	0mS
2	YES	25mS
3	YES	0mS
4	YES	0mS
5	YES	0mS
6	YES	0mS

Signal	Assigned Source
VCC	Source 1
All other signals	Source 2

Hot-Swap State:

Drive is in the ‘plugged’ state, waiting for a “**RUN:Power DOWN**” command to remove 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 (See application note downloads on www.quarch.com) 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

Clears 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 bootloader 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.

MEASure:VOLTage [12vin?|12vout?|12vin_chg?|12vout_chg?]

Returns the voltage on the specified rail in mV. Vin refers to the upstream or host side of the card, and Vout refers to the switched, drive side. Values are returned in the form "3300mV".

MEAS:VOLTage:SELF [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, 0 to 1270ms in steps of 10ms]

#2 = Bounce Length (mS)

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

#3 = Bounce Period (uS)

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

#4 = Duty Cycle (%)

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

SOURCE:[1-6|ALL]:DELAY [#ms]**SOURCE:[1-6|ALL]:DELAY?**

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

#1 = Initial Delay (mS)

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

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, 1000 to 127000us in steps of 1000us]

#3 = Duty Cycle (%)

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

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

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, 0 to 1270ms in steps of 10ms]

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

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, 1000 to 127000us in steps of 1000us]

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 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:[1-6]: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.

SIGna1:[SIG_NAME|ALL]:SETup [#num]

SIGna1:[SIG_NAME|ALL]:SOURce [#num]

Sets a given signal to a numbered timing source (0-8). SIGNAL_NAME is one of the items in the ‘Signal Names’ Appendix at the end of this manual.

SIGna1:[SIG_NAME]:SOURce?

Returns the source number that the signal is assigned to.

SIGna1:[SIG_NAME|ALL]:GLITCh:ENAb1e [ON|OFF]

SIGna1:[SIG_NAME|ALL]:GLITCh:ENAb1e?

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

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:ENAb1e** 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.

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]

Appendix 1 – Signal Naming

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

VCC

CLK_PL

CLK_MN

PEWAKE

DEVSLP

PEDET

CLKREQ

LED1

PERST

SUSCLK

ALERT

SMB_DATA

SMB_CLK

PETP_0

(Transmit Plus on lane 0) is the output data from the backplane to the drive, on the +ve side of the differential pair.

PETN_0

PERT_0

PERN_0

PETP_1

PETN_1

PERT_1

PERN_1

PETP_2

PETN_2

PERT_2

PERN_2

PETP_3
PETN_3
PERT_3
PERN_3

Signal Groups

ALL	(Allows change of all signals at the same time)
LANE0	(All 4 PCIe data signals that make up Lane 0)
LANE1	
LANE2	
LANE3	
CLK	(Differential clock signals)
MANAGEMENT	(Management interface signals)
DATA	(All PCIe high speed data pairs)
POWER	(VCC only)
SM_BUS	(All SM BUS signals)