

Change History

1.0	8th July 2022	First Release
1.1	6 th January 2023	Updated with information about the REFCLK buffering

Contents

Change History	2
Contents.....	3
Introduction	5
Technical Specifications	6
Switching Characteristics	6
Power Injection.....	7
Power Injection Maximum Continuous Current	8
Triggering.....	9
Trigger IN	9
Trigger OUT	9
Mechanical Characteristics	10
Dimensions and Connector Locations	10
Control Interfaces	11
Basic Concepts.....	12
Signal Configuration	13
Power Up vs. Power Down Timing.....	15
Pin Bounce Modes	16
Glitch Control.....	20
Signal Driving	23
Example – PERST	23
Advanced Usage:	24
Signal Monitoring	25
Requesting signal state.....	25
Live monitoring.....	26

Voltage Measurements	27
Default Startup State	28
Controlling the Module	29
Serial Command Set	29
SCPI Style Commands	29
Triggering Commands.....	39
Control Register Map	40
Appendix 1 - Signal Names	41
Appendix 2 – Signals Supporting ‘Monitoring’	43

Introduction

The **Torridon Gen5 AIC to U.2 Breaker Module** allows remote switching of PCIe data, power and supporting control pins between an AIC Slot and a U.2 device, for test automation or fault injection purposes.

The module supports PCIe Gen 2, Gen 3, Gen 4, and Gen 5 devices at data rates up to 32 GT/s.

Each pin is individually switched, allowing complete control over the mating of the card connector. Any individual pin(s) can be disconnected or glitched to simulate failures or configuration changes. If the devices support hot-swap, a sequenced hot-swap event can be run, connecting pins in sequence and simulating pin-bounce if required.

WARNING: Some systems DO NOT support hot swap of a PCIe card while the system is powered up. You should verify that your hardware supports this feature before using it.

The module can disrupt power, data, and control signals down to a minimum glitch length of 50nS.

The module includes power injection to the U.2 Device Under Test via either a Quarch Programmable Power Module (QTL1995 or QTL1999).

(*The practical minimum time for some pins, such as power pins, may be longer than this)

Technical Specifications

Switching Characteristics

AIC Edge Finger	U.2 Connector Pin	Description	Switching Action
A4, A12, A15, A18, B4, B7, B13, B16, B18, A20, B21, B22, A23, A24, B25, B26, A27, A28, B29, A31, B32	P5, P6, P12, S1, S4, S7, S8, S11, S14, S16, S19, S22, S25, S28, E9, E12, E15, E19, E22	PCIe Data and Power Ground Pins	All connected to ground on the Module
A16, A17, A21, A22, A25, A26, A29, A30, B14, B15, B19, B20, B23, B24, B27, B28	S17, S18, S20, S21, S23, S24, S26, S27, E10, E11, E13, E14, E17, E18, E20, E21	PCIe Data Signal pins	Individually switched by a High-Speed RF Switch, each signal is AC coupled on the receiver side of the switch with a 1uF capacitor
A13, A14	E1, E2, E7, E8	PCIe Reference Clock pins	Individually switched by a dual analog switch
A2, A3, B1, B2, B3	P14, P15	12V Power	Connected together and switched by 8A power FET
A2, A3, B1, B2, B3	P13	12V Precharge	Switched by 8A power FET
B10	E3	3.3V Aux Power	Switched by 0.7A FET
B5, B6	E23, E24	SM Bus Clock and Data	Individually switched by an analog switch
A11, B12, B11, B31	E4, E5, P1, P4	PERST, PERSTB/CLKREQ, WAKE, IFDET	Individually switched by an analog switch
	P2, P3, P10, P11, S15, E6, E16, E25	Sideband Signals	Digitally monitored/driven

Power Injection

A device connected to the U.2 will receive power via the 12V, 12V_CHARGE and 3V3_AUX pins of the connector. The breaker module can provide these power supplies either from the host (the PCIe slot) or from a separate Quarch instrument called a Programmable Power Module (commonly called a PPM, product QTL1999 or QTL1995).

Jumper J5 is used to select between the source for the power supplies:

J5	Operating mode	Power source
Not present	Host power	12V and 12V_CHARGE are supplied from 12V on the PCIe slot. 3.3V_AUX is supplied from 3V3AUX on the PCIe slot.
Present	Power injection	The PPM must be configured to output 12V and 3.3V and be connected to the breaker module using cable QTL1983. 12V and 12V_CHARGE are supplied to the U.2 connector from the 12V channel on the PPM. 3.3V_AUX is supplied to the U.2 connector from 3.3V channel on the QTL1999.

Power injection mode is intended to be used to capture data about the power consumed by the device using the PPM and Quarch’s QPS software package. Synchronization of the power signals with all the other data and sideband signals is described below.

Since the PCIe slot does not support hot-swap, the use case for power injection would follow this procedure:

- The breaker module should be connected to a PPM
- The breaker module should be connected to an unpowered Torridon Interface Unit
- The breaker module should be inserted into an unpowered host system
- The U.2 device should be inserted into the breaker module
- The PPM should be configured to output 12V and 3.3V and these outputs should be enabled
- Power should be applied to the Torridon Interface Unit
- By default, the breaker module will default into the powered up state, so all breaker switches will be enabled and the breaker will be connecting any PCIe slot connections through to the U.2 device. At this point, the host system is unpowered.

- The host system should be turned on. When the host system generates power on the 12V and 3.3V connections, these will be detected by the module, and the U.2 power will be connected to the PPM power source when they pass their voltage thresholds.
- The PCIe module synchronises 3v3 and 12v rails separately. 3v3_AUX is activated when the host 3V3AUX supply rises above 3.0V. 12V (and 12V_CHARGE) are activated when the 12V supply from the host rises above 11.09V.

This function can be turned on/off with the CONFig:INJection:SYNC [ON|OFF] command. Turning sync mode off is an advanced function and not normally required. If you turn sync mode off then it may be difficult to synchronise the power connected to the drive with all the other signals, and the user must take care to ensure that this is done safely.

Power Injection Maximum Continuous Current

Voltage Rail	Voltage Range	Max Current
12V	0-14.4V	4.0A (voltages > 10V)
3.3V	0-4V	4.0A (voltages > 1.5V)

For full specification of Power Margining Module output capabilities see product technical manual; available from <https://quarch.com/products/hd-programmable-power-module>.

Triggering

Triggering connectors and cables come supplied on triggering enabled modules (QTL2814). Both trigger connectors are MCX receptacles and are for connection to 3.3V equipment.

Trigger IN

Trigger IN allows you to control the hot-swap state of the module. This can be EDGE triggered (each rising edge swaps the state, causing a power-up or power-down action) or LEVEL triggered (The hot-swap state follows the input signal, every time it changes).

Alternatively, trigger in can be used to control the glitch engine. This can be EDGE triggered (each rising edge starts the currently selected glitch) or LEVEL triggered (each rising edge starts the currently selected glitch and the falling edge will end it, assuming it has not already completed).

Trigger OUT

Trigger OUT allows external equipment to follow the actions applied by the unit. This can be set to follow the hot-swap state, or to follow the glitch state. The trigger out signal will change whenever the action occurs, regardless of the means (trigger in or manual control) used to start the action.

REFCLK

This is an asymmetric breaker which has a single REFCLK pair on the host side, and two REFCLK pairs on the device side. The module uses a clock buffer to generate the two device-side pairs (referred to in this document as REFCLK and REFCLKB) from the single host-side pair. Breaker switches are placed between the outputs of the REFCLK buffer and the connector for the U.2 device.

The default for this module will be that it will operate as a single-port U.2 breaker. Breaker control of REFCLK for this single-port operation is implemented using the sideband isolation switches controlled by registers REFCLK_PL and REFCLK_MN.

In line with the default single-port operation, REFCLKB should not be presented to the U.2 device. Breaker operation for the REFCLKB channel is implemented using the sideband isolation switches controlled by registers REFCLKB_PL and REFCLKB_MN. These breaker switch controls are assigned to source 0 by default, so that they are permanently isolating the REFCLKB signals at the U.2 device.

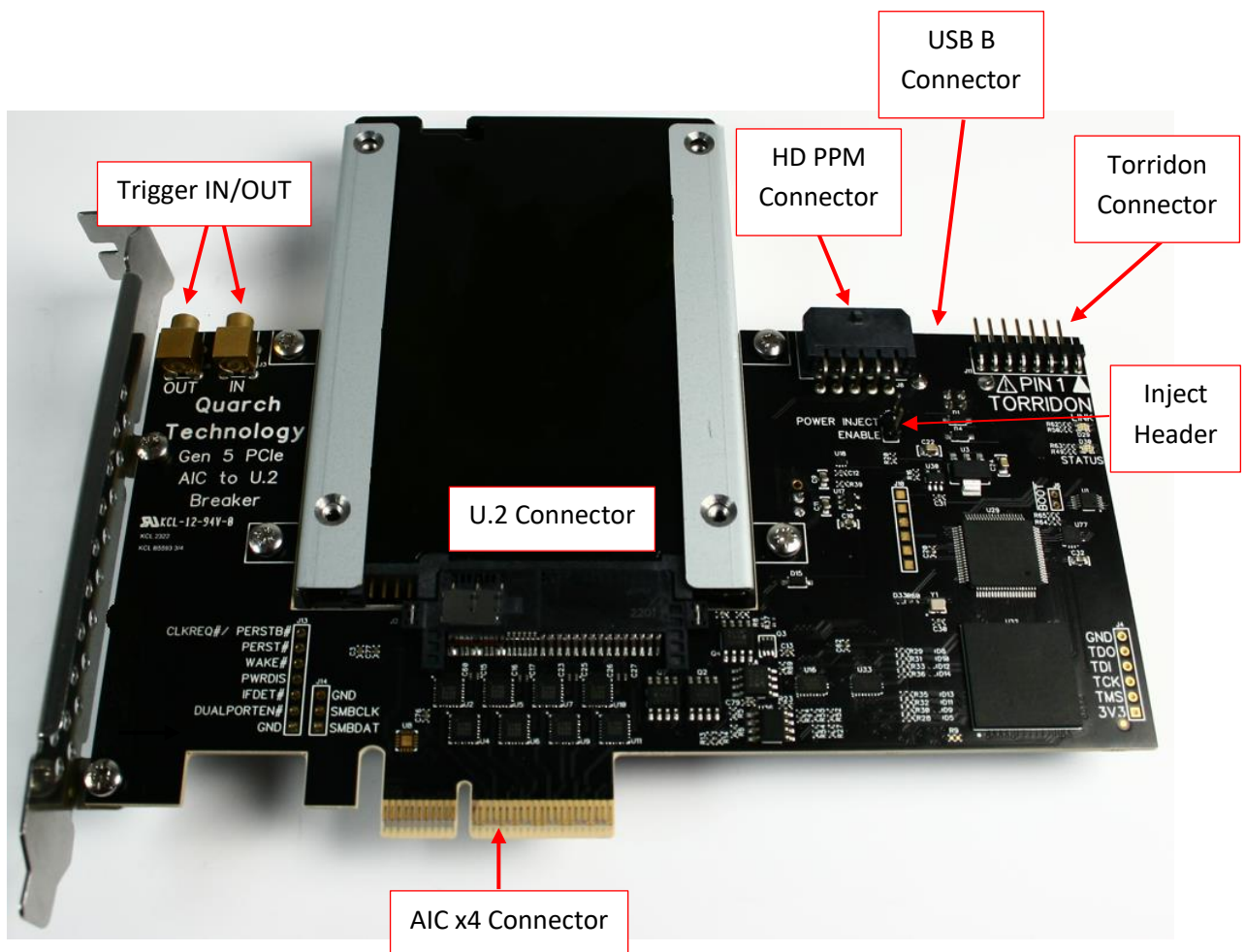
If the user wishes to test dual port operation, the signals for the REFCLKB_PL and REFCLKB_MN breaker controls can be assigned to different sources.

Mechanical Characteristics

Dimensions and Connector Locations

The right-hand connector with 'IN' marked next to it is 'Trigger IN'.

The left-hand connector with 'OUT' marked next to it is 'Trigger OUT'.



Control Interfaces

All Torridon Control Modules are designed to be used with a Torridon Array Controller (QTL1079) or a single Torridon Interface Card (QTL1144). This module also includes communication over a USB B connector.

The control cable is an ultra-thin Flex cable or USB cable.

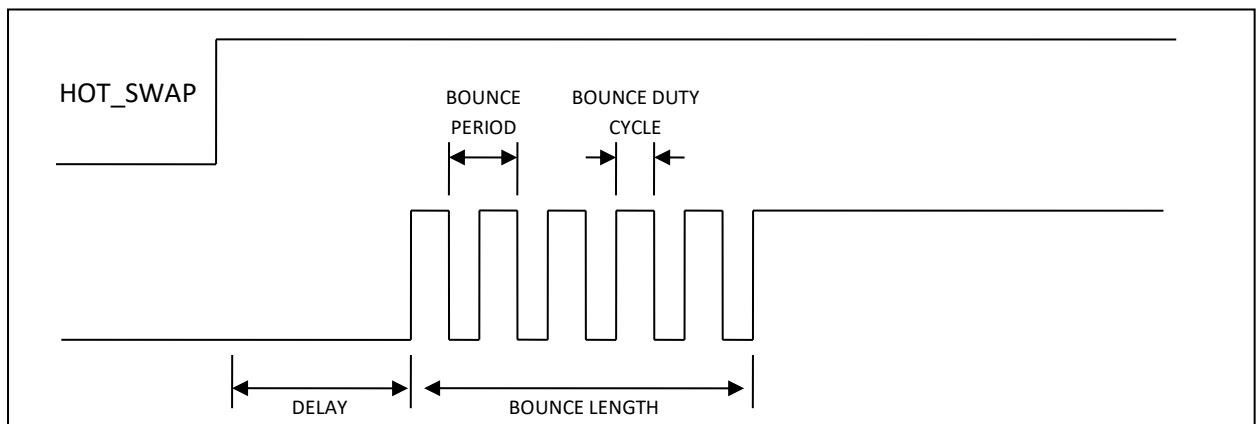
Control Interface	Form Factor	Torridon Module Ports	Control Methods Available	Interfaces
Torridon Array Controller	1U 19" Rack Mounted unit	24 at the front, 4 at the rear	Serial Scripting Script Generation through TestMonkey GUI	Serial via DB9 or RJ45 Ethernet
Torridon Interface Card	102mm x 26mm PCB	1 port	Serial Scripting Script Generation through TestMonkey GUI Real-time USB Control via TestMonkey GUI	Serial via DB9 or RJ45 USB

Basic Concepts

Each switch on the module is called a ‘Signal’ and can be programmed to follow one of 6 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.

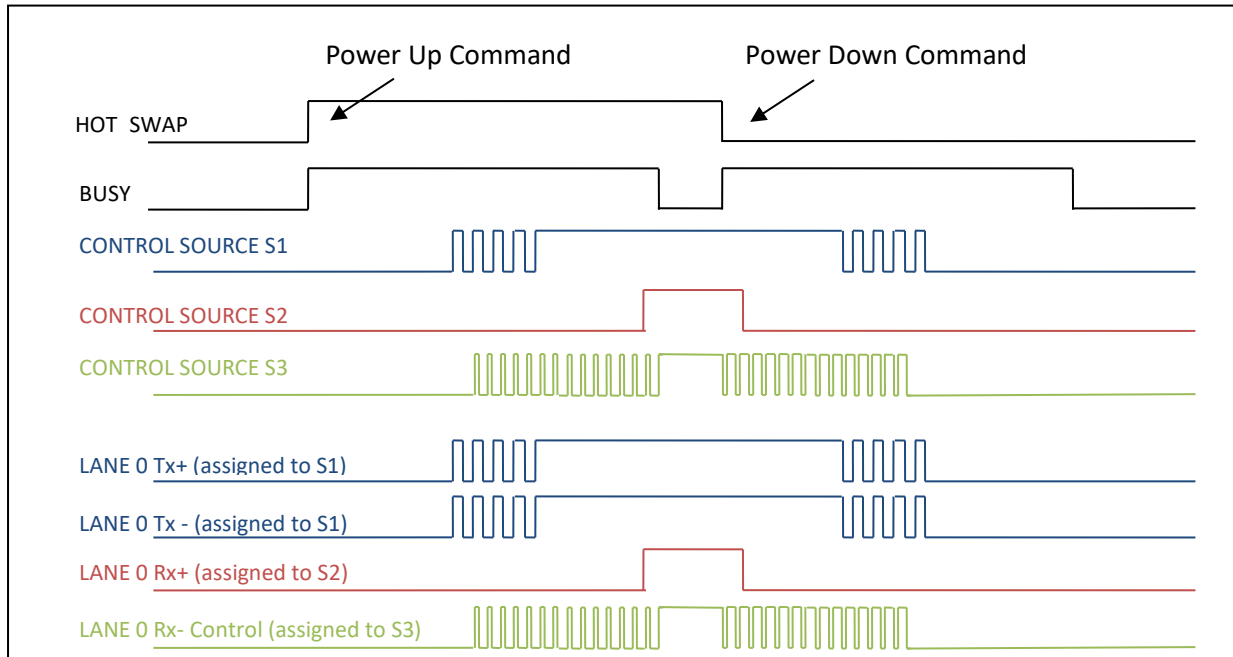
Control Source Parameters for a power up event (Basic Pin Bounce):



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:

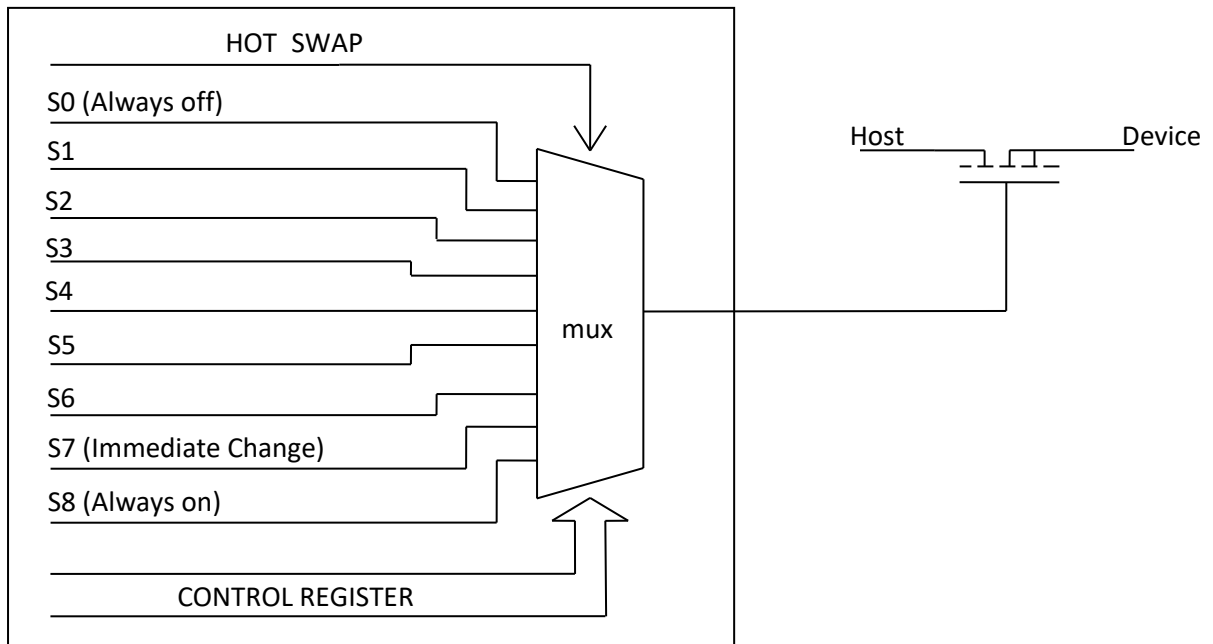


Signal Configuration

Each signal that is switched by the module is usually assigned to one of the 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).

To assign a signal to a control source, write to its CONTROL_REGISTER:

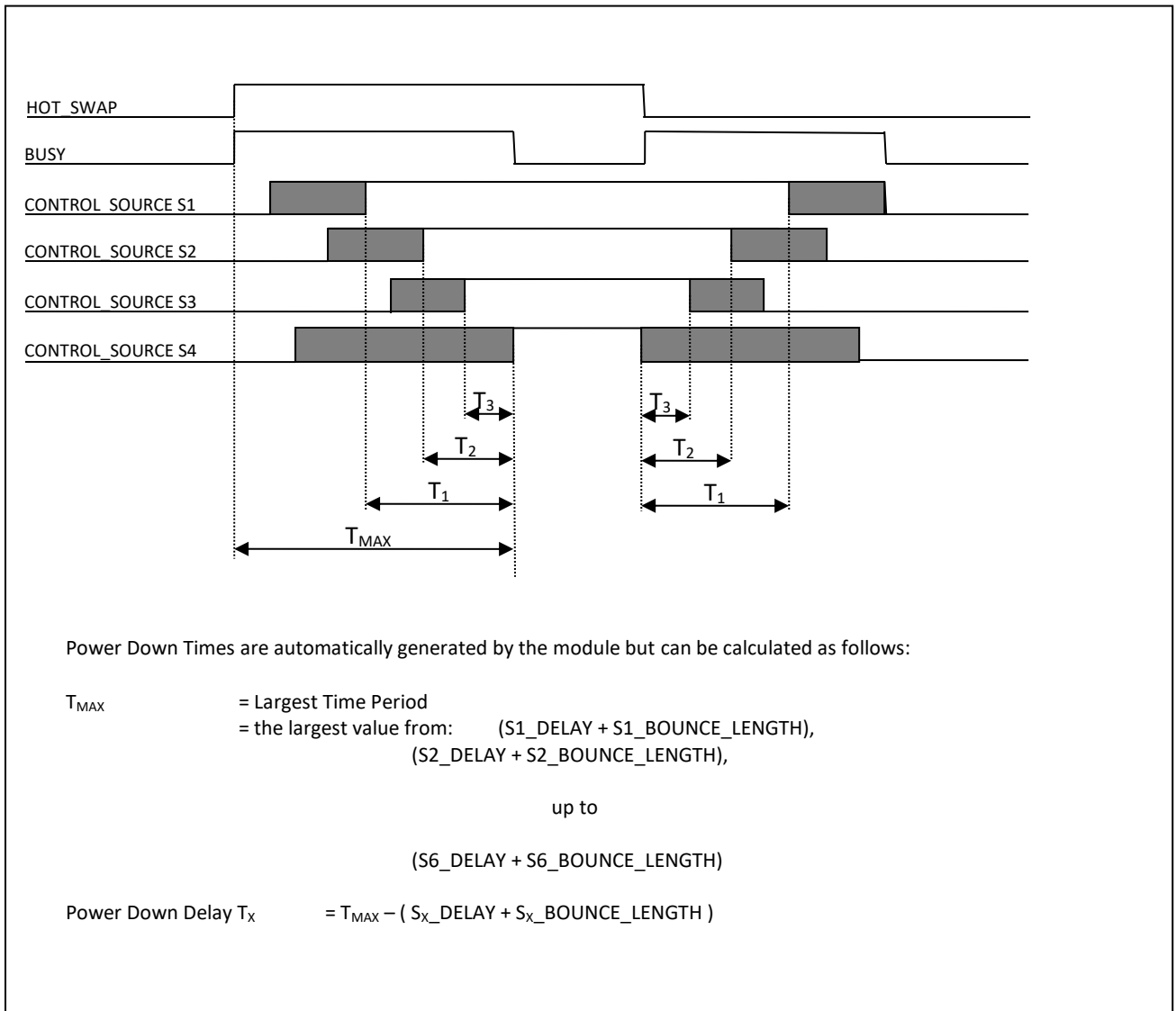
CONTROL_REGISTER Value	Description
0	Signal is always OFF
1	Signal assigned to control source 1
2	Signal assigned to control source 2
3	Signal assigned to control source 3
4	Signal assigned to control source 4
5	Signal assigned to control source 5
6	Signal assigned to control source 6
7	Signal changes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



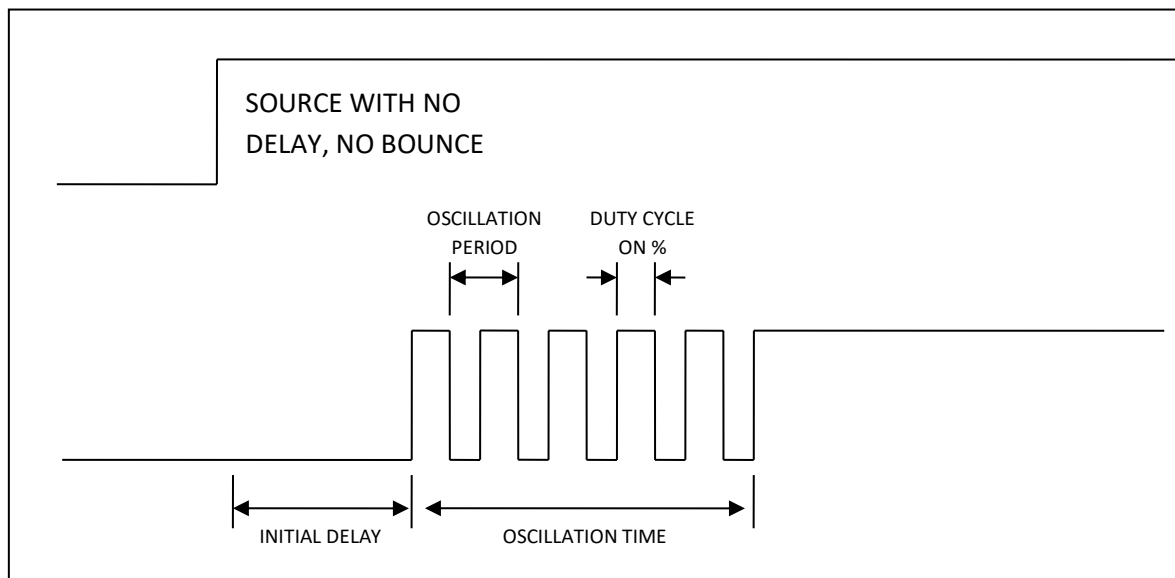
If you require a different power down sequence then you can alter any of the source timing values, pin bounce or signal assignments while the module is in the plugged state. When you initiate the 'pull' action, the new settings will be used.

Pin Bounce Modes

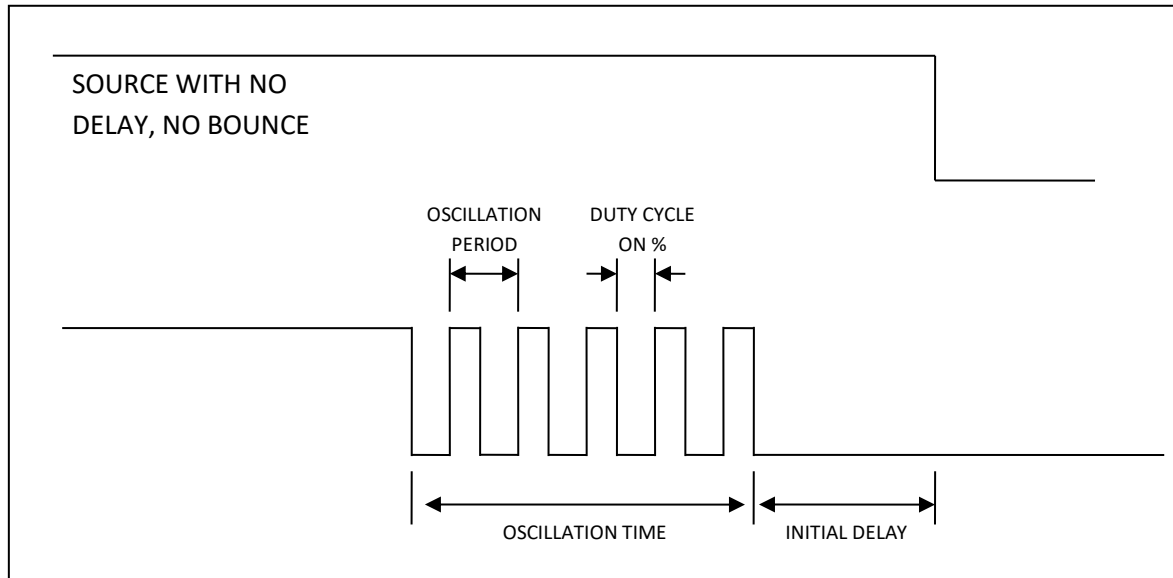
Pin Bounce can be set in two ways:

1. Basic Pin-Bounce (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



2. 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 on one of the 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

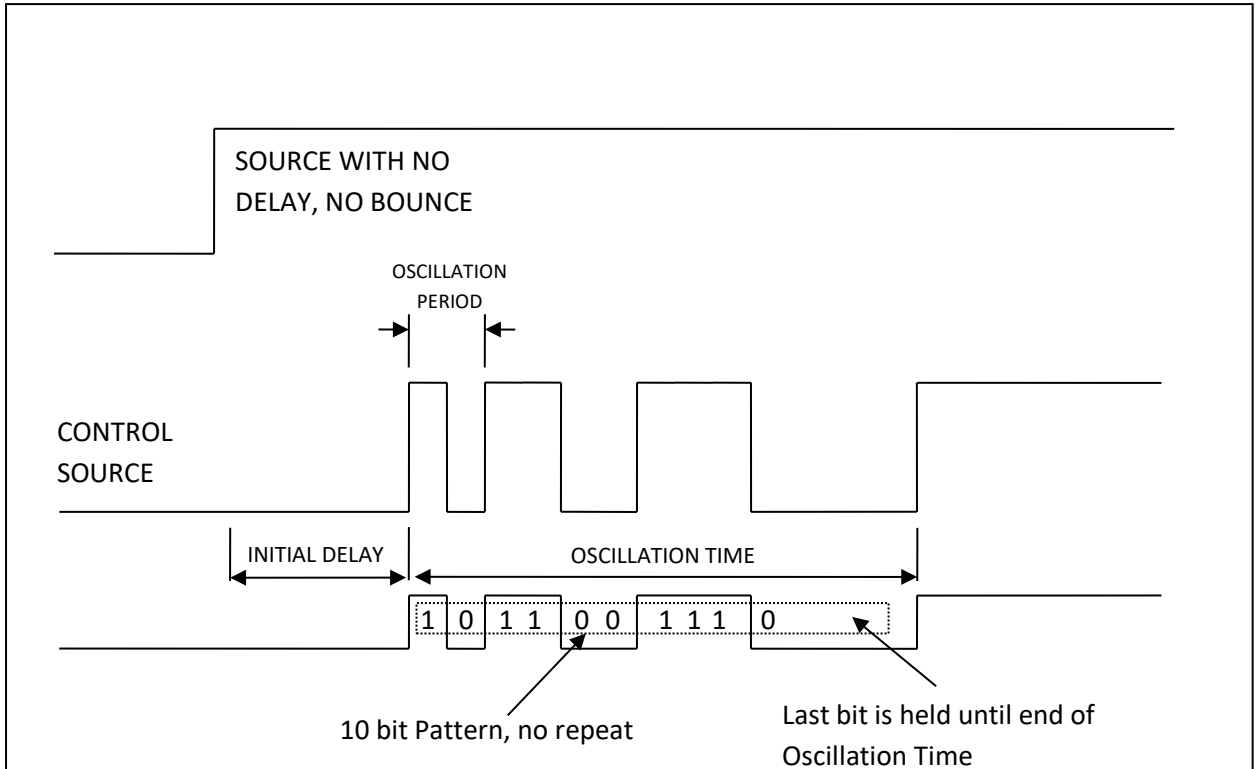
Custom Patterns can get confusing very quickly, the best way to make sure that the power down sequence is the opposite of power up, is to make sure that:

$$\text{(Bit length * Number of Bits = Oscillation Time)}$$

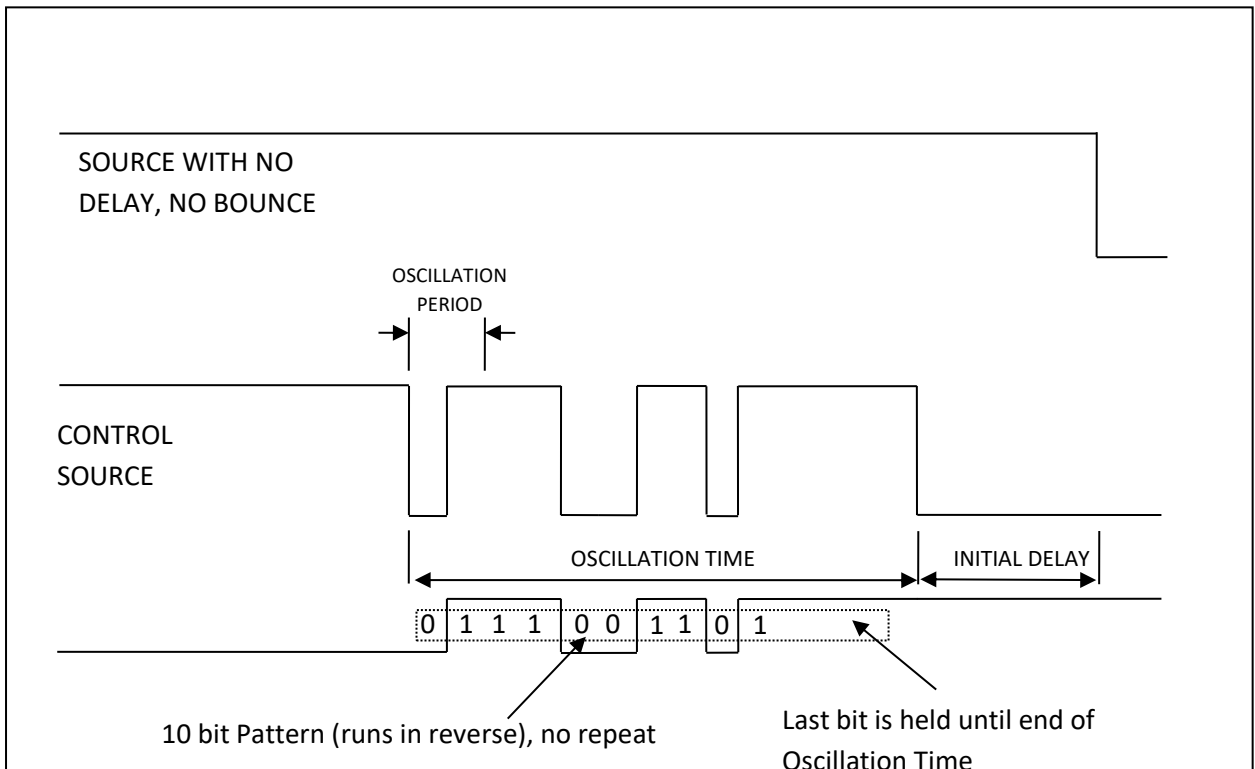
so that the pattern ends exactly at the end of Oscillation time. The SOURCE:[x]:BOUNce:PATtern:SETup command does this automatically.

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

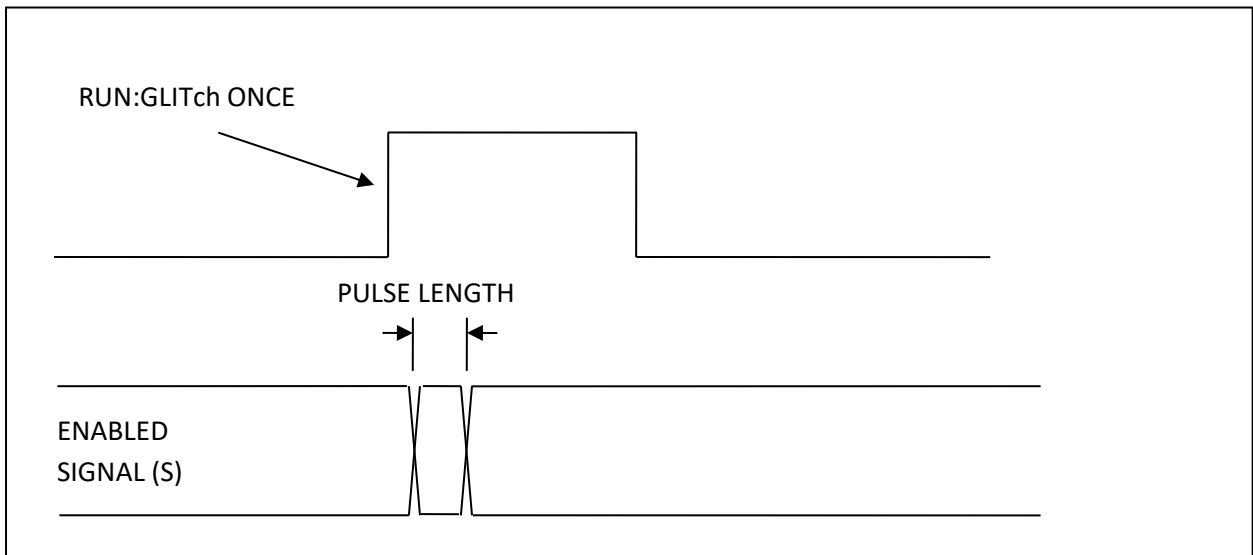
Any control signal may be glitched for a pre-determined length of time using the glitch generator logic.

Each Signal Control register contains a “GLITCH_ENABLE” bit which determines whether the glitch logic will affect that signal. The GLITCH_ENABLE bit, 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.

Glitches may be applied in 3 modes:

Glitch Once:



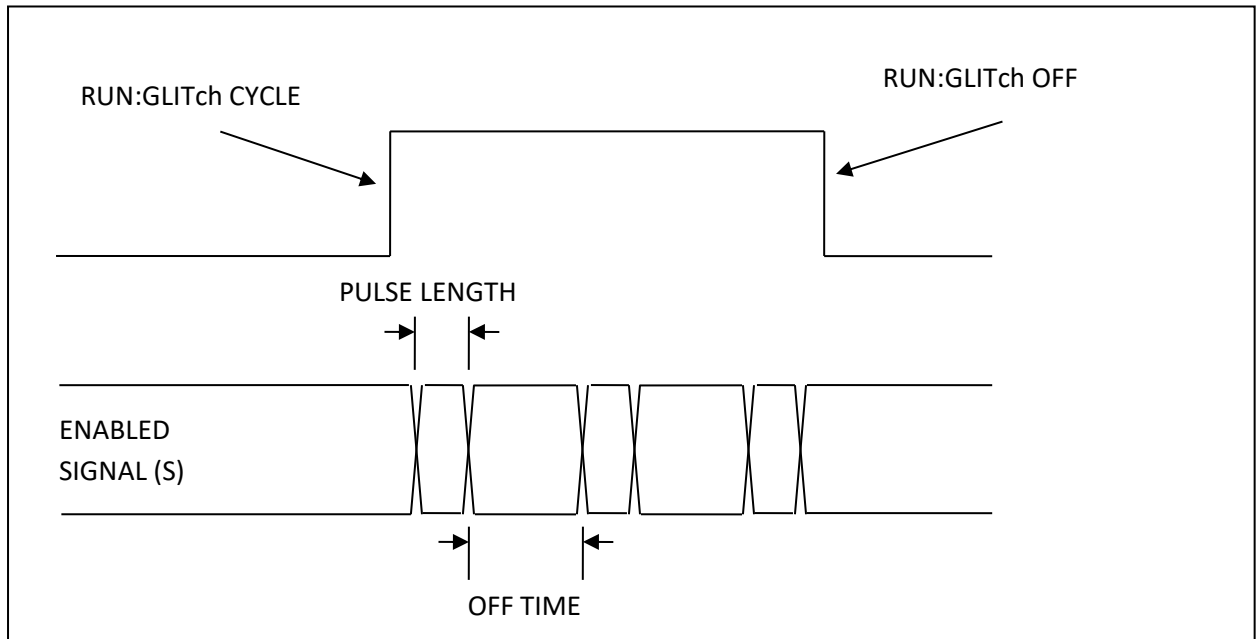
A single glitch is generated when the **RUN:GLITCh ONCE** command is executed

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.

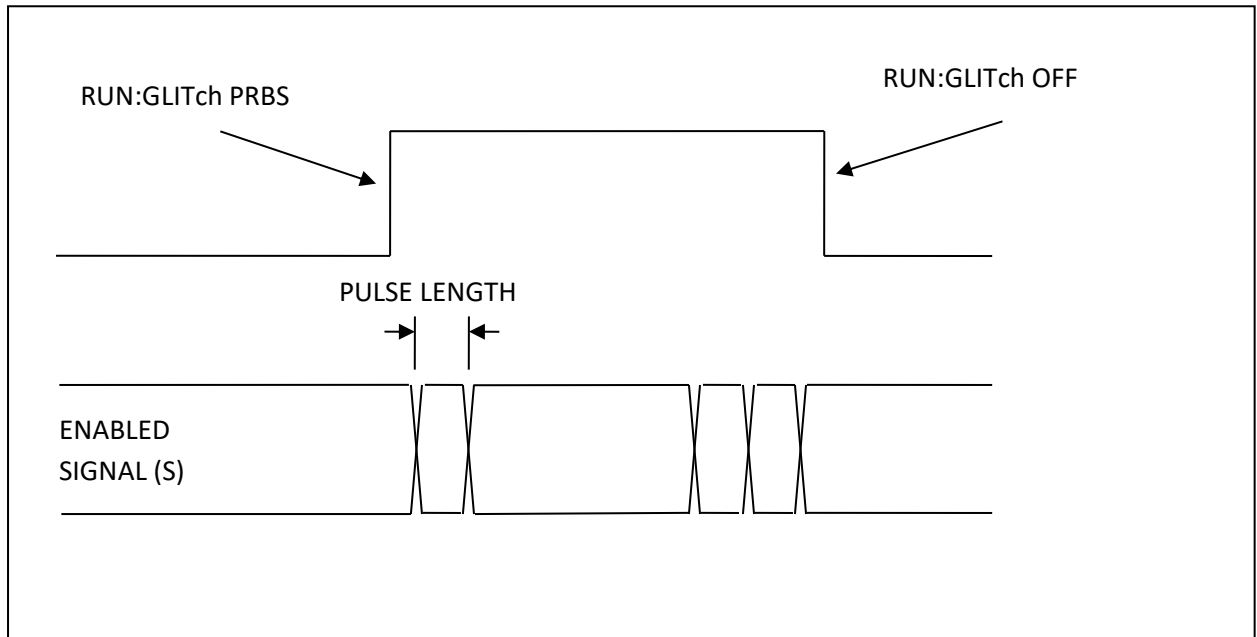
The length of the glitch is determined by using the **GLITCh:SETup** command or the **GLITCh:MULTiplier** and **GLITCh:LENgth** commands:

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

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:

- PERST
- PERSTB / CLKREQ
- DUALPORTEN
- HPT0
- HPT1
- PWR_DIS

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 behaviour 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 behaviour 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 Behaviour		Device Side Behaviour	
		High	Low	High	Low
PERST	Push/Pull output from host	Not Driven	Not Driven	Driven High	Driven Low
PERSTB / CLKREQ	Driven low by the system	Not Driven	Not Driven	Driven High	Driven Low
DUALPORTEN	Open Drain output from host	Not Connected		Drive High	Driven Low
PWR_DIS	Push/Pull output from host	Not Connected		Drive High	Driven Low
HPT0	Open drain output from host	Not Connected		Drive High	Driven Low
HPT1	Open drain output from host	Not Connected		Drive High	Driven Low

Example – PERST

To issue a fundamental reset to the device under test:

PERST is an active low signal so to assert it we need to drive it low. The module is already powered up so we need to change the **CLOSED** behaviour 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** behaviour and use the glitch function on the PERST signal to control the “open” time.

>SIGNAL:PERST:GLITCH:ENABLE ON

> 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:

SIGna1:[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 system monitoring. 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	12mV / 5%
MEASure:VOLTage:SELF 3v3?	Returns the voltage of the modules internal 3.3v power rail – This powers the modules internal circuitry, and the active circuitry on the IN connector	15mV / 5%
MEASure:VOLTage:SELF -5v?	Returns the voltage of the modules internal -5v power rail	15mV/ 5%
MEASure:VOLTage 12v_host?	Returns the voltage inputted from the 2.1mm 12V power socket	64mV/ 5%
MEASure:VOLTage 12v_power_device?	Returns the voltage being supplied to the device on the 12v power rail	64mV/ 5%
MEASure:VOLTage 12v_charge_device?	Returns the voltage being supplied to the device on the 12v charge rail	64mV/ 5%
MEASure:VOLTage 3v3_aux_host?	Returns the internal voltage created from the 2.1mm 12V power socket	64mV/ 5%
MEASure:VOLTage 3v3_aux_device?	Returns the voltage being supplied to the device on the 3v3 aux rail	64mV/ 5%

Default Startup State

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

The default hot-swap scenario will disconnect all pins based on the pin length, without any pin-bounce. All data and control pins are assigned to source 1 and will change immediately. Presence pins are assigned to source 2 and will change 25ms after the other pins.

Source Number	Initial Delay	Pin Bounce Mode	Bounce Length	Bounce Period	Bounce Duty Cycle
1	0mS	Standard	0mS	0uS	50%
2	25mS	Standard	0mS	0uS	50%
3	50mS	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
IF_DET, IF_DET2	Source 1
12V_CHARGE, PWR_DIS, PRSNT	Source 2
All other signals	Source 3

Hot-Swap State:

The module 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 or on a module with a PoE/LAN option). This mode uses exactly the same commands as the serial ASCII terminal
- USB
Quarch's TestMonkey application can control a single module via USB, this allows simple graphical control of the module.

Serial Command Set

When connected via a serial terminal, the module has a simple command line interface

SCPI Style Commands

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:WIDth [1|2|4|8|16]

Sets the lane width of the module. This deactivates the unused lanes and prevents them from being connected, overriding the normal signal settings.

CONFig:WIDth?

Returns the current lane width setting

DEPRECATED COMMANDS – Provided for backwards compatibility, we strongly suggest you use the ‘Signal’ and ‘Source’ commands instead.

REGister:READ [0xAA]

Returns the value of the register with address [0xAA]. [0xAA] should be in hex format and preceded by the suffix “0x”. e.g. “0x6D”. The value is returned in the same form as the address.

REGister:DUMP [0xA1] [0xA2]

Returns the value of each register in a range, starting at the first register address, up to the second. [0xA1] and [0xA2] should be in hex format and preceded by the suffix “0x”. Each data value will be returned on a new line.

REGister:WRITe [0xAA] [0xDD]

Writes the byte [0xDD] to register [0xAA], both [0xDD] and [0xAA] should be in hex format and preceded by the suffix “0x”. The command returns “OK” or “FAIL”.

CONFig:INJection:SYNC [ON|OFF]**CONFig:INJection:SYNC?**

Sets or returns the power injection sync state. When on AND Jumper J5 is connected, the PCIe card will not have power supplied unless the host power is detected

CONFig:INJection:MODE?

Returns the state of the J5 jumper. When ON, the power injection jumper (J5) is connected.

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 a 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.

SIGnal:[SIG_NAME|ALL]:SETup [#num]
SIGnal:[SIG_NAME|ALL]:SOURce [#num]

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|CLOsed] [NONE|HIGH|LOW]
SIGnal:[SIG_NAME]:GLITch:[OPEn|CLOsed]?

Sets the drive state for signals that can be driven (forced to a high or low state). See the section on signal driving for full details

GLITch:SETup [MULTIPLIER_STEP] [#count]

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

#1 = Multiplier factor for glitch length (mS)
[50ns|500sn|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|500sn|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 | 500sn | 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 | 500sn | 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 [2|4|8|16|32|64|128|256|512|1024|2048|4096|8192|16384|32768|65536]

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 Torricon modules). This is done by changing the HOT_SWAP bit, register 0x00 bit 0. This is the master control for all switches on the card. The same action can be performed by writing this bit directly.

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:MULTIplier** x **GLITCh:LENgth** x **GLITCh:CYCLE**. 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.

RUN:GLITCh STOP

Stops any running glitch sequence.

RUN:GLITCh?

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

Triggering Commands

These commands are found only on the triggering version of the module

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

TRIGger:IN:MODE?

Sets/Return the trigger in mode. This chooses the action to be performed when a trigger in is received.

POWER : Trigger in will alter the hot-plug state

GLITCH : Trigger in will start a glitch

TRIGger:IN:TYPE [EDGE|LEVEL]

TRIGger:IN:TYPE?

Sets/Returns the trigger in signal type. This describes how the trigger in signal should be interpreted. See the triggering details for information on how each trigger mode is affected

TRIGger:OUT:MODE [OFF|POWER|GLITCH]

TRIGger:OUT:MODE?

Sets/Returns the trigger out mode. This chooses the action that will cause a trigger out to occur.

POWER : Trigger out will occur on hot-swap

GLITCH : Trigger out will occur on glitch

RUN:INTerrupt?

Returns a list of active interrupt flags and also clears those flags. Flags are:

COMPLETE : An action (such as hot-swap or glitch completed fully)

TRIGGERED : An external trigger occurred

GLITCh:MODE [ONCE|CYCLE|PRBS]

GLITCh:MODE?

Sets/Returns the current glitch mode. This allows you to choose the glitch action that will be performed if the trigger:in:mode is set to 'GLITCH'.

Control Register Map

Access to the FPGA registers should not be required for the majority of operations and customers are strongly encouraged to use the high level commands in order to maintain compatibility with future firmware versions. If you require details of the register map, please contact:

support@quarch.com

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

12V_CHARGE
12V_POWER
3V3_AUX

PERST
REFCLK_PL
REFCLK_MN

PETP0 (Data transmitted from the 'input' port on Lane 0 (+ve side of differential pair))

PETN0

PERP0 (Data received at the 'input' port on Lane 0 (+ve side of differential pair))

PERN0

PETP1

PETN1

PERP1

PERN1

PETP2

PETN2

PERP2

PERN2

PETP3

PETN3

PERP3

PERN3

REFCLKB_PL

REFCLKB_MN

PERSTB

SMCLK

SMDAT

DUALPORTEN

IF_DET

ACTIVITY

WAKE
PWR_DIS
PRSNT
HPT0
HPT1
IF_DET2

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

POWER (All power and pre-charge signals)
SMBUS (All SM BUS signals)

CLK_A (Ref clock signals for port A)
CLK_B
PORT_A (All signals for port A: 8x Data signals plus REF_CLK and PERST signals)
PORT_B
DATA_A (All data signals for port A, PCIe data lanes only)
DATA_B

Appendix 2 – Signals Supporting ‘Monitoring’

Signal name	Side that can be monitored
PERST	Host and Drive
PERSTB	Host and Drive
SMBCLK	Host and Drive
SMDAT	Host and Drive
WAKE	Host and Drive
DUALPORTEN	Drive
HPT0	Drive
HPT1	Drive
IF_DET	Host and Drive
IF_DET2	Drive
ACTIVITY	Drive
PRSNT	Drive
PWR_DIS	Drive