Quarch Technology Ltd

AN-003 – NVMe Plugfest hotswap testing Application Note

For use with:

QTL1743 – PCIe Drive Control Module

Change History

1.0		Initial Release
1.1		Added additional hot-plug speeds
1.2		Added Python examples code
1.3		Fixed bug in python code where link speed was read incorrectly
1.4		Updated with fixes for hosts requiring lspci ‘-M’ option. Improved setup instructions
1.5		Added support for 4-port controllers and python 3.x

Contents

NVMe Plugfest – Hotplug Testing

Hotswap testing requirements for PCIe SFF (U.2) devices has been steadily increasing, due the number of issues

Plugfest #4

From NVMe Plugfest #4 (June 2014), hotplug testing was mandatory. All PCIe SFF devices were required to pass a hot-swap test, using Quarch's 'Torridon' modules.

Not all of the devices were able to pass the test station, so clearly this is an important test case to consider.

Only a basic test was required at this time. This test is intended to validate the hot-swap ability of an NVMe device into a host system.

This current test is intended for basic verification only, and does NOT include:

- Corner cases for all hot plug scenarios
- Broken connector scenarios or other faults
- Pin-bounce testing

Quarch would recommend that anyone planning to pass the hot-swap part of the Plugfest should implement these additional tests before the event.

Plugfest #5

From the February 2016 Plugfest, an expanded range of hot-plug speeds were added.

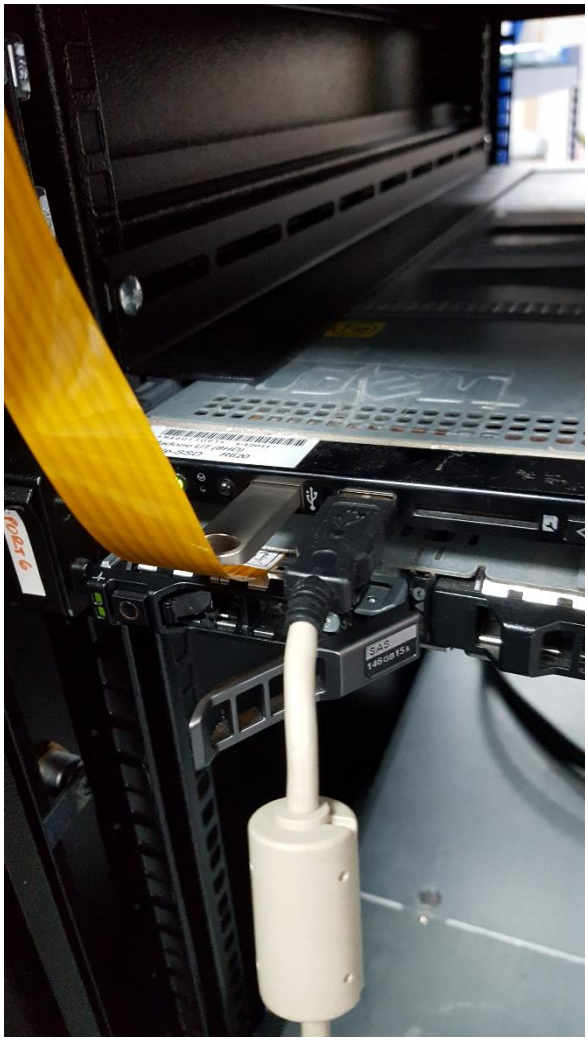
Quarch would recommend that anyone planning to pass the hot-swap part of the Plugfest should implement these additional tests before the event.

Requirements

1. Host System
2. NVME Device (PCIe SFF device)
3. QTL1260 – Interface Kit
4. Quarch PCIe module (QTL1743 PCIe Drive Control Module)

Setup

1. Place the Quarch control module behind the SFF drive and insert it into the host enclosure



2. Connect the control module to the QTL1260 interface kit, noting the alignment of the arrows

3. Attach the 12v power to the QTL1260
4. Attach the USB cable from the QTL1260 to the host
5. Push the button on the QTL1260 in the way (which will generate a USB virtual COM port on the host)
6. Power on the host
7. Verify that the drive can be seen by the OS

8. Copy the example power shell or Python version scripts to the host (additional install may be required, detailed below)
9. Run the script or follow the manual procedure, as required

Test Procedure

- Power up the host system and verify the drive is functioning normally
- Perform each of the hot-plug speeds 10 times, verifying the drive and checking for errors after each plug/removal (PLUGFEST MANDATORY)
- The current hot-plug speeds required are
  - ‘Standard’ Plug is 25ms between pin lengths
  - ‘Fast’ plug is 10ms between pin lengths
  - ‘Slow’ plug is 100ms between pin lengths
  - ‘Very Slow’ plug is 500ms between pin lengths

For each of the hot-plugs we ‘pull’ the drive, wait 10 seconds (and verify the drive has removed without error), ‘plug’ the drive and verify that the drive returns correctly.

The timing parameters for the test specify the time between each length of pin in the connector mating (longest pins mate first).

The test is normally down via the Python or PowerShell version of the scripts which are provided with this application note. It is also possible to run the test manually in TestMonkey

Automating with Python script

- Python is available as standard on Linux installs, you will need to install **Python 3.x** for Windows (Python 2.x should work but is less well tested). The 'Python' folder provided with this application note contains a Python script for automating the testing. This uses the standard QuarchPy package to control the Quarch module, and again uses lspci.exe to monitor the PCIe device.

  If you do not currently use the QuarchPy then you will need to install it via

  ```
  Pip install quarchpy
  ```

  Further details, manual and local install code can be found on our site: https://quarch.com/products/quarchpy-python-package

- You will also need the FTDI driver to support the virtual COM port in the interface kit. Windows will normally download this automatically, but it can also be installed from here:

  http://www.ftdichip.com/Drivers/VCP.htm

- If you chose to use a USB connection, you will require the Quarch USB driver:

  https://quarch.com/file/torridon-driver-win8

- Using an elevated command prompt (cmd.exe run in administrator mode) navigate to the folder containing the Python script and execute:

  ```
  >Python Hotplug cycle test.py
  ```

- After the run, a time-stamped log file in the script folder will contain the full results of the tests.

Warning: If have are using a PCIe switch in your host, lspci may not be able to view the device you want to hot-swap. In the man script set: `mappingMode = True` to allow vision of the devices beyond the switch. Note that this may greatly increase the number of devices shown, so is defaulted to 'False'

Automating with PowerShell script

- PowerShell in included in most windows installation and should be available without additional install

- You will also need the FTDI driver to support the virtual COM port in the interface kit. Windows will normally download this automatically, but it can also be installed from here:

  http://www.ftdichip.com/Drivers/VCP.htm

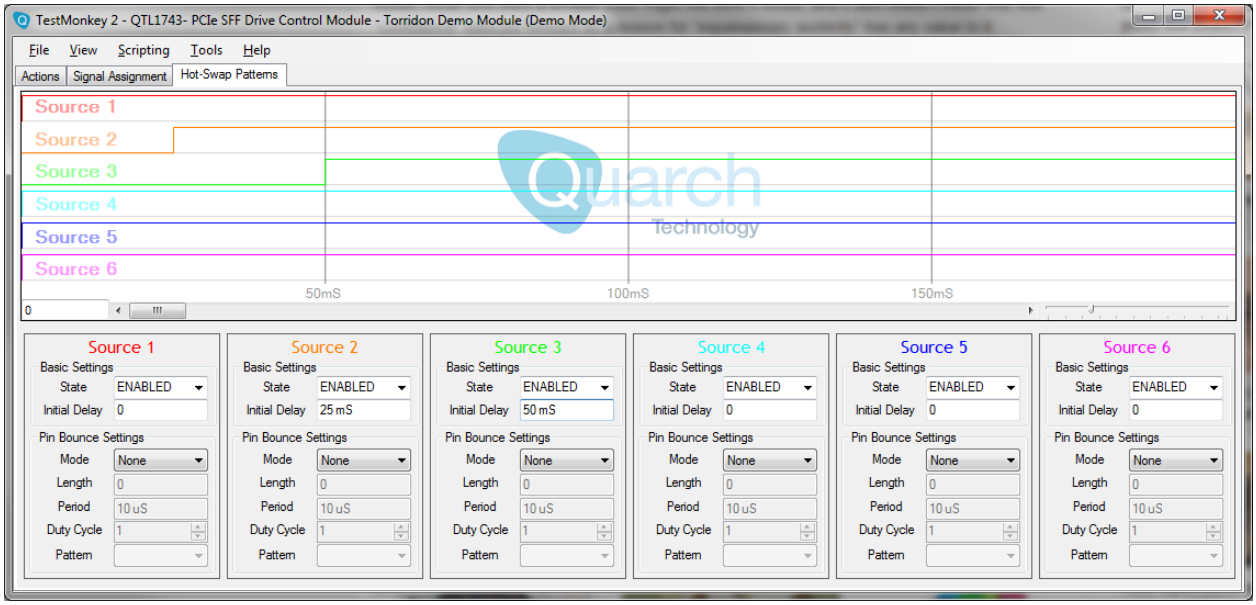
- Run a PowerShell terminal in administrator mode, navigate to the folder containing the scripts and run:

  .\Hotplug.ps1

Warning: If have are using a PCIe switch in your host, lspci may not be able to view the device you want to hot-swap. In the man script set: `mappingMode = True` to allow vision of the devices beyond the switch. Note that this may greatly increase the number of devices shown, so is defaulted to 'False'

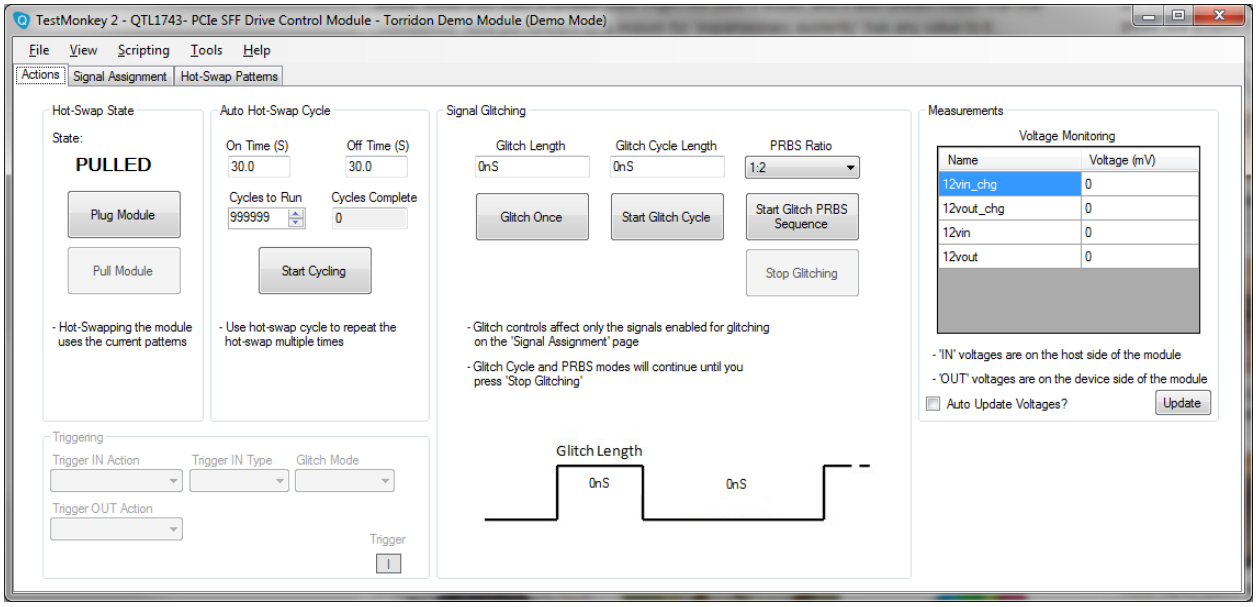
Manual test – TestMonkey

Assuming that the Quarch module was power cycled or otherwise reset into default state before the test, then the 'Initial Delay' setting for sources 2 and 3 are the only things we need to change:



Here we can see the 0, 25mS, 50mS timing sequence for the standard speed hot-swap. Changing these numbers allow you to create the fast and slow sequences as required.

To perform the actual hot-plug, we use the 'Plug' and 'Pull' buttons on the actions tab:



Automated test – Basic commands

If you wish to automate the test in a different script language, the full command listing used is below. These can be send via USB, Serial, Telnet or ReST depending on the controller you choose to use.

Initial Power Up

No commands are required. The module will start up in the 'plugged' state by default.

Basic Hot Swap

Tests that the Drive/Host combination can perform a standard surprise hot-swap.

Set the module to its default state (if not already done):

```
CONFig:DEFault STATE
```

Set sources 1, 2 and 3 to the correct increment time for the test (10/25/100 ms) between each of the sources in order:

```
SOURce:1:DELay 0
SOURce:2:DELay 25
SOURce:3:DELay 50
```

In the default state, the module will already be set to the 25ms delays.

Now 'pull' the drive by running the power down sequence:

```
RUN:POWer DOWN
```

'Plug' the drive by running the power up sequence

```
RUN:POWer UP
```

Verify the operation of the drive in each state and log any unexpected error messages. Repeat for the fast and slow speed plugs.

Hot-plug recommendations

The plugfest tests are a basic minimum, and the number of cycles are limited due to time constraints.

1. Ideally you should run each of the hot-plug scenarios *at least* 100 times. This will make it more likely that you will see faults that only happen on occasion.
2. You should run tests that are beyond the required fast/slow parameters. Perhaps using 5ms for fast plugs and 250ms for slow plugs.
3. Ideally you should test more unusual cases as well, these will not happen as often in the real world, but they can occur, so it is important your device can handle them:
    a. Reverse the source timing, so source 2 has a lower value than source 1 (and similar). This simulates some pins mating later than expected.
    b. Add pin bounce to each of the sources in turn. This simulates the imperfect nature of copper connections, where the pins do not make cleanly.

Additional Tests

Below are a number of additional tests that we recommend as a good start. These are not currently required at the Plugfest, but this may change in the future. In any case, they are valuable scenarios to perform.

Dual Port tests

a. Hot-plug with port A only connected
b. Hot-plug with port B only connected
c. Break then make port A while running dual ported
d. Break them make port B while running dual ported

Method (tests a/b)

Set the module back to default state:
CONFig:DEFault STATE

Power it down:

RUN:POWer DOWN

Set the port(s)/lane(s) that we are NOT going to connect, to source 0 (always off). This will prevent them connecting:

SIGnal:PORT_B:SOURce 0

Now power up the device, only port A will come up in this case:

RUN:POWer UP

Repeat for other ports/lanes

Method (tests c/d)

Set the module back to default state:
CONFig:DEFault STATE

Disconnect the port/lane that we want to drop:

SIGnal:PORT_A:SOURce 0

Reconnect the port/lane:

SIGnal:PORT_A:SOURce 1

Repeat this for the other ports/lanes

Basic pin-bounce tests

a. Hot-plug with pin-bounce on IF_DETECT only
b. Hot-plug with pin-bounce on data

Method (test a)

Set the module back to default state then power it down:
CONFig:DEFault STATE

RUN:POWer DOWN

Set pin bounce on IF_DETECT (or MATED), by moving it to a different source and setting a 15ms long at 500us period oscillation:

SOURce:6:BOUNce:PERiod 500

SOURce:6:BOUNce:LENgth 15

SOURce:6:DELay 0

SIGnal:IF_DETECT:SOURce 6

Now run the power up sequence which will contain the pin bounce, as part of the pattern.

Method (test b)

Set the module back to default state then power it down:
CONFig:DEFault STATE

RUN:POWer DOWN

Set pin bounce on DATA, by moving all data signals to a different source and setting a 15ms long at 500us period oscillation:

SOURce:6:BOUNce:PERiod 500

SOURce:6:BOUNce:LENgth 15

SOURce:6:DELay 0

SIGnal:DATA:SOURce 6

Now run the power up sequence which will contain the pin bounce, as part of the pattern.