XPliant Family of Programmable Multilayer Switches

xpShell User Guide

CNX-SH-V3.2P

Contents of this document are subject to change without notice.

Cavium Proprietary and Confidential DO NOT COPY

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

February 2016

PUBLISHED BY
Cavium, Inc.
2315 North First Street
San Jose, CA 95131-1010
PhoneLength 408-943-7100
FaxLength 408-577-1992
EmailLength sales@cavium.com
WebLength httpLength//www.cavium.com

© 2016 by Cavium, Inc.

All rights reserved. No part of this manual may be reproduced in any form, or transmitted by any means, without the written permission of Cavium.

XPliant is a registered trademark of Cavium. All other trademarks or service marks referred to in this manual are the property of their respective owners.

Cavium makes no warranty about the use of its products, and reserves the right to change this document at any time, without notice. Whereas great care has been taken in the preparation of this manual, Cavium, the publisher, and the authors assume no responsibility for errors or omissions.

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table of Contents

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

List of Tables

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Chapter 1

Introduction

XPliant is a family of integrated, multilayer, software-defined switches. Software defines the XPliant switch “personality”, tailored to address the various requirements in different place-in-network (PIN) specifics. The personality is expressed in network profiles, typically configured into the switch at start-up. The network profile defines the forwarding pipeline, its functional behavior, and associated on-chip memory resources partitioning and entry formatting. The forwarding tables can be modified at runtime.

This document describes XPliant xpShell, a shell that provides an interactive command-line interface to enable, manage, populate, and display table entries.

The contents of this document are organized into the following sectionsLength

- Chapter 1, Introduction—Overview of guide contents, including the conventions used in the code examples.
- Chapter 2, Overview of xpShell—A high-level description of xpShell.
- Chapter 3, XPShell Use Cases—Feature-specific descriptions of xpShell functionality with example use cases.

1.1 Revision History

The revision history is shown in Table 1–1.

Table 1–1 Revision History

Version	Date	Remark
3.2	February 2016	Updated for 3.2 release.
3.1	August 2015	First release as separate document.

1.2 Conventions

This document uses the following conventionsLength

Table 1–2 Conventions

Convention	Description
`Monospace font`	Commands, information the system displays, and file paths/ names appear in `monospace font`.
`Italic monospace font`	Arguments for which the user provides a value are in *`italic monospace font`*.
`Bold monospace font`	Commands that the user must enter exactly are in **`bold monospace font`**.

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table 1–2 Conventions

Convention	Description
{} Braces	{} are used to enclose a list of pipe-delimited values from which the user must include one; for example {1|2|3}.
Bold font	Screen selections are in bold font; for example, “Select the page **Type Based Boot Priority**”.

1.3 Related Documentation

This document should be used in conjunction with the documents listed in Table 1–3: *Publications*. This document may contain information that was not previously published.

Table 1–3 Publications

Publication	Document Number
XPliant Functional Specification	
XPliant Software Programmer and Configuration Guide	CNX-PG-V3.2P
XPliant Software Theory of Operation	CNX-TOO-V3.2P
XPliant API Guide	CNX-API-V3.2P.html (located in XDK doc/ folder)

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Chapter 2

Overview of xpShell

xpShell is an XPliant command-line interface (CLI) for XDK. xpShell uses the Python `cmd2` package. xpShell enables calling SDK APIs in interactive mode, as well as adding runtime commands. It provides commands to manage and configure features on the devices through XDK; for example, for debugging or diagnostics purposes. In addition, xpShell provides a Python interface and supports running scripts. xpShell uses SWIG interface to invoke XDK APIs.

CLI commands can be auto-generated using header files and API definitions and are easy to integrate by linking with a library.

xpShell can run on the XPliant hardware or on the Simulation Model after establishing an IPC connection with xpSim.

- Running on the hardware, xpShell is typically launched to run within a user application or xpApp when the `-u` flag is used (refer to the section “Command-line Usage” in Chapter 6, XPliant Simulator and XP Application Sample of the *XPliant Software Programmer and Configuration Guide*), In this mode, unless explicitly called from xpShell, it does not trigger any hardware or software accesses and uses the process context for device operations.
- For development or debug purposes, xpShell can be launched to intercommunicate with the XPliant Device Simulator, being invoked as `python xpShell.py withwm`. In this mode, xpShell requires connecting to the XP Simulator (xpSim) and operates on shadows both local to xpShell and local to the xpSim.

In addition, for debugging purposes, xpShell can also be launched in a standalone mode, being invoked as `xpShell.py standalone`. In this mode, xpShell operates on the local shadow and does not require running or connecting to actual hardware or the XP Simulator (xpSim).

The CLI hierarchy is divided into several modes; the commands available at any given time depend on the current mode. Entering a question mark (`?`) or typing `help` at the CLI prompt returns a list of commands available for each command mode.

The software recognizes a command as soon as you enter enough characters of the command to uniquely identify it.

To exit CLI, `quit` or `exit` commands are available. The xpShell can be exited in the following waysLength

- Exit 0—Exits and restarts xpShell. This is useful if one edits the xpShell python script and wants to see the changes immediately without restarting the entire white model.
- Exit 1—Exit, back quit. This kills xpShell but xpApp continues to run. It is recommended that this option not be used unless it is required.

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

- Exit 3—Kills both xpShell and xpApp. This is the recommended way to exit the white model.

CLI command availability is evolving; the supporting command implementation is done in `cli/xpShell.py`.

xpShell imports XP and XPS APIs, enabling users to prototype customized CLI commands on their basis.

The following table provides examples of available commandsLength

Table 2–1 xpShell Commands

Command	Description
display_tables	Commands to display software-defined tables such as FDB, Tunnel-IVIF, Port-VLAN, Bridge Domain, IPv4/v6 Host, IPv4/v6 Route Prefix, IPv4 Route Next Hop, etc.
xps	Exposes XPS Layer APIs to CLI commandsLength acm, init, l3, mirror, packetdrv, stp, iacl/eacl, interface, lag, mpls, policer, tunnel, fdb, link, multicast, port, serdes, vlan, geneve, ipgre, nat, sflow, vxlan, ipinip, mac, nvgre, qos, etc.
l2_domain	Configure and print corresponding Layer 2 configurations and statuses.
link_manager	Configure and print corresponding link layer configurations and statuses.
acm	● Print all non-zero corresponding registers, if any. ● Print ACM counter value at given counter index. ● Print all non-zero counters, if any, per mode (XP_ACM_COUNTING, XP_ACM_POLICING, XP_ACM_SAMPLING). ● Print ACM counter value for a given VLAN and port. ● Configure and print ACM sampling state and corresponding configurations. ● Configure and print ACM global configurations.
qos	● Configure shapers and corresponding configurations. ● Configure AQM profiles and corresponding configurations. ● Commands to show queues information for a given port and queueLength drop count, queue forward length count in pages, queue length depth, etc. ● Commands to show PFC counts.
regAccess	Dump, read, and write registers.
diags	Various device diagnostic commands.
debug	● Various blocks debug commands, including printing data structures, statuses, and internal states. ● Print and control GPIO, service CPU, DMAs, and management interfaces statuses and configurations. ● Print PLL statuses and configurations. ● Print and control reset statuses and configurations. ● Print interrupts statuses and configurations.

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table 2–1 xpShell Commands

Command	Description
`dal_debug`	- Print and set the DAL-Type and DAL-Mode, especially for debug operations at the DAL. - Commands to redirect reads and writes to DAL shadow. - Commands to restore DAL shadow. - Various debug commands. - Commands to count reads/writes.
Utilities	- `py`—Launch a Python shell and execute a Python script. - `save`—Record the executed commands. - Send a packet to the simulator, from `.pcap` file to a specified device and ingress port.
Basic commands	- Set SAL type. - Print XDK version. - Add/remove device and initialize/release the XDK for that device.

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Chapter 3

xpShell Use Cases

xpShell script can easily accommodate adding new commands or editing commands on-the-fly on a running system without requiring reboot or recompilation. Commands executed on xpShell can be saved or copied into a file to load and run at a future time.

Debugging and troubleshooting of the system can be done through the CLI. xpShell also provides commands to upgrade firmware or compare the software shadow with hardware and the ability to take a complete snapshot of the hardware register/tables. Similarly, the complete register/table dump from the file can be loaded on to the hardware. MAC/link-level configuration and MAC statistics/counters can be read from xpShell commands.

Register access commands provide the ability to read and write each register and their fields for supported device types. Since the application can interface to hardware or the simulator through different interfaces (PCI, MDIO, and IPC), xpShell can select the corresponding DAL interface for the correct interface. Similarly, the DAL debug layer can be configured from the CLI on top of the selected interface for troubleshooting the read/write access to a specific register on the device.

In a given mode, xpShell comes up with available commands (and sub-commands) in various categories.

```
(xpShell):xps)?

Available commands (type help <topic>):
======================================
acm        erspan  interface  lag        multicast  policer  stp
aging      fdb     internal   link       nat        port     tunnel
br         geneve  ipgre      mac        nhgrp      qos      vlan
eacl       iacl    ipinip     mirror     nvgre      serdes   vxlan
egressflt  init    l3         mpls       packetdrv  sflow

Utility commands
================
back  clear  help  ls   pause  py   save   shortcuts
cd    eof    load  nop  pwd    run  shell
```

Help is available for each command usage.

```
(xpShell) help reg_access
Register and Table operation commands

(xpShell) reg_access
(xpShell):regAccess)?

Available commands (type help <topic>):
=======================================
check_reg_volatile               print_reg_id_by_name  write_reg_name
compare_hw_sh                    print_reg_ptr         write_reg_name_in
drv_pipe                         print_shadow_mem_ptr  write_reg_off
```

Cavium Confidential For Chen Peng Teraspek 01/24/2018

```
drv_usb                          print_shadow_reg_ptr  write_reg_off_in
drv_wrapper                      print_table_entry     write_reg_off_name
enable_pipe_access               print_volatile_regs   write_reg_off_name_in
load_scpu_firmware               read_mac_reg          write_table_entry
print_all_rxdma0_regs            read_reg
print_all_txdma0_regs            read_reg_field
print_attr_of_all_regs           read_reg_name
print_attr_of_all_static_tables  read_reg_off
print_attr_of_reg                read_reg_off_name
print_attr_of_reg_name           reg_compare_hw_sh
print_attr_of_static_table       set_force_hw_read
print_attr_table_of_all_regs     table_compare_hw_sh
print_complete_mem_to_file       write_from_file
print_mem_to_file                write_from_hex_file
print_mem_to_hex_file            write_mac_reg
print_reg                        write_reg
print_reg_attr_at_offset         write_reg_field

Utility commands
================
back  clear  help  ls   pause  py   save   shortcuts
cd    eof    load  nop  pwd    run  shell

(xpShell):regAccess)

(xpShell):regAccess)print_attr_of_reg 0 232
Input Arguments are devId=0, registerId=232
=================================================================
Register Id = 232
Name of the Register = XP_SKPU_CFG_SKPU_DEBUG
Num of Instances = 8
Register Type = 2
Num repeat = 1
Mem Depth = 0
Word width = 4
Bit width = 119
SW Name = 698
Functional = 0
Hardware Address Range :
Contents of Reg : XP_SKPU_CFG_SKPU_DEBUG (size = 16 bytes)
RegId = 232, Inst = 0, rep = 0, off = 0, HW Addr = 0x560a20c
Raw data: zeroth word at left, sz=4 words
0x00000000  0x00000000  0x00000000  0x00000000

----------------------------------------------------------------
FieldName (pos,len)         :  Value
------------------------------------
```

See the following sections for examples of using basic xpShell CLI commands.

3.1 Layer 2 xpShell Programming

All feature managers and tables are initialized during initialization. Refer to the initialization section in the *XPliant Software Programmer and Configuration Guide*.

The programming of the Layer 2 feature includes creating VLANs, managing the FDB table, setting STP states, and enabling learning.

The following XPS xpShell commands are used to program the Layer 2 feature.

Cavium Confidential For
Chan Peng
Teraspek
01/24/2018

3.1.1 VLAN

Table 3–1 Layer 2 VLAN Programming

Action	Command
Create VLAN.	(xpShell):xps:vlan)**vlan_create** *devId vlanId*
Set VLAN configuration.	(xpShell):xps:vlan)**vlan_set_config** *devId vlanId stpId countMode enableMirror mirrorAnalyzerId saMissCmd bcCmd unknownUcCmd arpBcCmd ipv4McbridgeMode ipv6McbridgeMode unRegMcCmd*
Enable/disable learning.	The SA learning mode is enabled by default but can be disabled using the following command: (xpShell):xps:vlan)**vlan_set_unknown_sa_cmd 0 100 1**
Add port to VLAN.	(xpShell):xps:vlan)**vlan_add_interface** *devId vlanId intfId tagType*
Add the endpoint	(xpShell):xps:vlan)**vlan_add_endpoint** *devId vlanId intfId tagType data*
Create primary VLAN, secondary VLAN, community VLAN, and isolated VLAN.	(xpShell):xps:vlan)**vlan_create** *devId priVlanId* (xpShell):xps:vlan)**p_vlan_create_primary** *devId priVlanId* (xpShell):xps:vlan)**vlan_create** *devId secvlanId* (xpShell):xps:vlan)**p_vlan_create_secondary** *devId secVlanId priVlanId vlanType*
Provide interface ID.	The interface ID (intfid) is required for vlan_add_interface to add an interface to VLAN. (xpShell):xps:port)**port_get_port_intf_id** *devId portNum* (xpShell):xps:vlan)**vlan_add_interface** *devId vlanId intfId tagType*
Add community and isolated VLANs using primary and secondary VLAN.	(xpShell):xps:vlan)**p_vlan_add_interface_community** *devId priVlanId secVlanId intfId tagType* (xpShell):xps:vlan)**p_vlan_add_interface_isolated** *devId priVlanId secVlanId intfId tagType* (xpShell):xps:vlan)**p_vlan_add_interface_primary** *devId priVlanId intfId tagType*
Set VLAN ARP broadcast.	The vlan_set_config configuration is used to set all VLAN parameters. The vlan_set_arp_bc_cmd configuration sets VLAN ARP broadcast. (xpShell):xps:vlan)**vlan_set_arp_bc_cmd** *devId vlanId arpBcCmd*

Example

Table 3–2 Example: Layer 2 VLAN Programming

Action	Command
Layer 2 Flooding. Create VLAN number 500 and enable the Layer 2 feature on ports 0 and 1. Flooding will happen on both ports.	**1.** Create VLAN. (xpShell):xps:vlan)**vlan_create 0 500** **2.** Set VLAN configuration. (xpShell):xps:vlan) **vlan_set_config 0 500 1 0 0 0 0 0 0 1 1 1 1 0 0 1** (xpShell):xps:vlan) **vlan_set_bc_cmd 0 1 1** (xpShell):xps:vlan) **vlan_set_sa_learning_mode 0 1 0** (xpShell):xps:vlan) **Vlan_set_unknown_uc_cmd 0 1 1** (xpShell):xps:vlan) **vlan_set_unknown_sa_cmd 0 1 1** **3.** Add ports to VLAN. (xpShell):xps:vlan)**vlan_add_interface 0 500 0 0** (xpShell):xps:vlan)**:vlan_add_interface 0 500 1 0** Send traffic on one port and both ports will be flooded.

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

3.1.2 LAG

The following xpShell commands are used to create LAGs, add a port to a LAG, and deploy a LAG.

Table 3–3 Layer 2 LAG Programming

Action	Command
Create a LAG interface.	`(xpShell):xps:lag)lag_create`
Add a port to an existing LAG.	`(xpShell):xps:lag)lag_add_port` *`lagIntf portIntf`*
Deploy the LAG.	`(xpShell):xps:lag)lag_deploy` *`devId lagIntf autoEnable`*
Verify port membership of LAG interface.	`(xpShell):xps:lag)lag_is_port_member` *`devId port lagIntf`*
Remove port from LAG interface.	`(xpShell):xps:lag)lag_remove_port` *`lagIntf portIntf`*
Destroy LAG.	`(xpShell):xps:lag)lag_destroy` *`lagIntf`*

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Example

Table 3–4 Example: LAG Programming

Action	Command

Create a LAG interface.
Add ports 1 and 2 to lag interface.
Configure VLAN 20 and add LAG and port 3.
	1. Create LAG.
	`(xpShell):xps:lag)lag_create`
	`Output:laglntf = 51200`
	2. Add **ports 1 and 2** to created LAG interface and deploy it**.**
	`(xpShell):xps:lag)lag_add_port 51200 1`
	`(xpShell):xps:lag)lag_add_port 51200 2`
	`(xpShell):xps:lag)lag_deploy 0 51200 1`
	`Prepare VLAN configuration for traffic forwarding:`
	3. Create **VLAN,** add configured **LAG** and **port 3** to created VLAN.
	`vlan_create 0 20`
	`vlan_add_interface 0 20 51200 0`
	`vlan_add_interface 0 20 3 0`
	4. Configure VLAN to **forward unknown unicast** traffic.
	`vlan_set_config 0 20 1 0 0 0 0 0 1 0 0 0 0`
	`or`
	`vlan_set_unknown_uc_cmd 0 1 1`
	5. To add **port 5** in runtime to already configured **LAG 51200**, use the following commands:
	`lag_add_port 51200 5`
	`lag_deploy 0 51200 1`
	6. To remove **port 5** from **LAG 51200** in runtime, use the following commands:
	`lag_remove_port 51200 5`
	`lag_deploy 0 51200 1`
	7. To destroy LAG (all ports should be removed from LAG), use the following commands:
	`lag_remove_port 51200 1`
	`lag_remove_port 51200 2`
	`lag_deploy 0 51200 1`
	`lag_destroy 51200`

3.1.3 FDB Table

Table 3–5 Layer 2 FDB Table Programming

Action	Command
Create FDB entry.	`(xpShell):xps:fdb)fdb_add_entry devId vlanId macAddr pktCmd isControl isRouter isStatic intfId serviceInstId`
Write FDB entry at a given index.	`(xpShell):xps:fdb)fdb_write_entry devId index vlanId macAddr pktCmd isControl isRouter isStatic intfId serviceInstId`
Get FDB entry with index.	`(xpShell):xps:fdb)fdb_get_entry devId vlanId macAddr intfId`
Get FDB entry using index.	`(xpShell):xps:fdb)fdb_get_entry_by_index devId index`
Trigger FDB aging.	`(xpShell):xps:fdb)fdb_trigger_aging devId`
Find FDB entry.	`(xpShell):xps:fdb)fdb_find_entry devId vlanId macAddr intfId`
Remove FDB entry.	`(xpShell):xps:fdb)fdb_remove_entry devId vlanId macAddr intfId`
Return Layer 2 encapsulation.	`(xpShell):xps:fdb)fdb_get_l2_encap_type devId intfId vlan`
Flush FDB entry.	`(xpShell):xps:fdb)fdb_flush_entry_by_intf devId intfId`
Flush FDB entry using VLAN.	`(xpShell):xps:fdb)fdb_flush_entry_by_vlan devId vlanId`
Set FDB attribute.	`(xpShell):xps:fdb)fdb_set_attribute devId vlanId macAddr pktCmd isControl isRouter isStatic intfId serviceInstId,field`
Set FDB attribute based on index.	`(xpShell):xps:fdb)fdb_set_attribute_by_index devId index field`
Get FDB attribute based on index.	`(xpShell):xps:fdb)fdb_get_attribute_by_index devId index field`

Example

Table 3–6 Example: Traffic Forwarding through FDB

Action	Command

Traffic forwarding through the FDB table.	**1.** Create VLAN.
	`(xpShell):xps:vlan)vlan_create 0 7`
	2. Add egress tagged (tag type=`1`) and egress untagged (tag type=`0`) interfaces to VLAN.
	`(xpShell):xps:vlan)vlan_add_interface 0 7 0 0`
	`(xpShell):xps:vlan)vlan_add_interface 0 7 1 1`
	3. Set VLAN configuration.
	`(xpShell):xps:vlan)vlan_set_config 0 7 1 0 0 0 0 0 0 1 1 0 0 0 0`
	4. Add FDB entry
	`(xpShell):xps:fdb)fdb_add_entry 0 7 00:00:00:11:12:33 1 0 0 1 1 7`
	Traffic sent on port 0 will be forwarded to programmed MAC address in the FDB table attached to port 1.

Cavium Confidential For
Chen Peng
Feraspek
01/24/2018

3.1.4 Aging

Table 3–7 Layer 2 FDB Aging

Action	Command
Enable FDB aging.	(xpShell):xps:fdb)**fdb_configure_aging** *devId enable*
Set FDB age time.	(xpShell):xps:fdb)**fdb_set_aging_time** *devId sec*
Trigger FDB aging.	(xpShell):xps:fdb)**fdb_trigger_aging** *devId*
Register aging handler.	xpShell):xps:fdb)**fdb_register_aging_handler** *devId fdbAgingHandler*
Unregister aging handler.	(xpShell):xps:fdb)**fdb_unregister_aging_handler** *devId*
Register learning handler.	(xpShell):xps:fdb)**fdb_register_learn_handler** *devId fdbLearnHandler*
Unregister learning handler.	(xpShell):xps:fdb)**fdb_unregister_learn_handler** *devId*

3.1.5 STP

Table 3–8 Layer 2 STP Programming

Action	Command
Create, STP state.	(xpShell):xps:stp)**stp_create**
Destroy STP.	(xpShell):xps:stp)**stp_destroy** *stpId*
Set STP state.	(xpShell):xps:stp)**stp_set_state** *devId stpId intfId stpState*
Get STP state.	(xpShell):xps:stp)**stp_get_state** *devId stpId intfId*

Example

Table 3–9 Example: Layer 2 STP Programming

Action	Command

STP
Basic scenario for traffic flooding.
	1. Define STP instance.
	`(xpShell):xps:stp)stp_create`
	`# stpId = 1`
	2. Define VLAN 1086.
	`(xpShell):xps:vlan)vlan_create 0 1086`
	3. Configure VLAN parameters to enable flooding.
	`(xpShell):xps:vlan)vlan_set_unknown_sa_cmd 0 1086 3`
	`(xpShell):xps:vlan)vlan_set_bc_cmd 0 1086 1`
	`(xpShell):xps:vlan)vlan_set_arp_bc_cmd 0 1086 0`
	`(xpShell):xps:vlan)vlan_set_stp_enable 0 1086 1`
	`(xpShell):xps:vlan)vlan_bind_stp 0 1086 1`
	`(xpShell):xps:vlan)vlan_set_count_mode 0 1086 0`
	`(xpShell):xps:vlan)vlan_set_mirror_to_analyzer 0 1086 0 0`
	`(xpShell):xps:vlan)vlan_set_unknown_sa_cmd 0 1086 1`
	`(xpShell):xps:vlan)vlan_set_unknown_uc_cmd 0 1086 1`
	`(xpShell):xps:vlan)vlan_set_ipv4_mc_bridge_mode 0 1086 1`
	`(xpShell):xps:vlan)vlan_set_ipv6_mc_bridge_mode 0 1086 0`
	`(xpShell):xps:vlan)vlan_set_unreg_mc_cmd 0 1086 1`

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table 3–9 Example: Layer 2 STP Programming

Action	Command

	4. Add ports (10, 11) into VLAN 1086.
	`(xpShell):xps:vlan)vlan_add_interface 0 1086 10 0`
	`(xpShell):xps:vlan)vlan_add_interface 0 1086 11 0`
	5. Enable stp bpdu forwarding to CPU globally.
	`(xpShell):xps:vlan)vlan_set_global_control_mac 0 01:80:c2:00:00:00`
	6. Set STP state for ports: [0 - DISABLED; 1 - LEARNING; 2 - FORWARD; 3 - BLOCKING].
	`# STP BPDUs are Forwarded to CPU over states : Learning, Forwarding, Blocking`
	`# Traffic is forwarded only over STP states Forwarding and Disabled`
	`vlan_set_stp_state 0 1086 10 2`
	`vlan_set_stp_state 0 1086 11 0`
	7. Configure FDB entry for each port in VLAN 1086.
	`fdb_add_entry 0 1086 22:33:44:01:02:03 1 0 0 0 10 1086`
	`fdb_add_entry 0 1086 22:55:aa:01:02:03 1 0 0 0 11 1086`
	Traffic can be sent and will be flooded.

3.2 Layer 3 xpShell Programming

The Layer 3 software feature managers are initialized during initialization. Refer to the initialization section in the *XPliant Software Programmer and Configuration Guide*.

Host and route programming are required to program the Layer 3 feature. The routing and VRF are enabled after programming the host and route.

The ingress router MAC address is configured for both routing and hosting.

Table 3–10 Layer 3 Ingress Router MAC Address Configuration

Action	Command
Add interface-independent ingress router MAC.	`(xpShell):xps:l3)`**`l3_add_ingress_router_mac`** *`devId mac`*
Add interface-specific ingress router MAC.	`(xpShell):xps:l3)`**`l3_add_intf_ingress_router_mac`** *`devId 3IntfId mac`*

3.2.1 Host Programming

Table 3–11 Layer 3 Host Programming

Action	Command
Add host entry.	`(xpShell):xps:l3)`**`l3_add_ip_host_entry`** *`devId vrfId type ipv4Addr ipv6Addr pktCmd serviceInstId vpnLabel propTTL l3InterfaceId macDa egressIntfId reasonCode`*

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

3.2.2 Route Programming

Table 3–12 Layer 3 Route Programming

Action	Command

Configure route programming.	`(xpShell):xps:l3)l3_add_ingress_router_mac devId mac`
	`(xpShell):xps:l3)l3_set_egress_router_mac_m_sbs devId macHi`
	`(xpShell):xps:l3)l3_create_vlan_intf vlanId`
	`(xpShell):xps:l3)l3_set_intf_egress_router_mac_lsb devId l3IntfId macSa`

Add entry in IPv4 NH table.	`(xpShell):xps:l3)l3_create_route_next_hop nhEcmpSize`
	`(xpShell):xps:l3)l3_set_route_next_hop devId nhId pktCmd serviceInstId vpnLabel propTTL l3InterfaceId macDa egressIntfId reasonCode`

Add route entry.	`(xpShell):xps:l3)l3_add_ip_route_entry devId vrfId type pv4Addr ipv6Addr ipMaskLen nhEcmpSize nhId`

3.2.3 Enabling Routing and VRF

After adding the entries in tables, enable routing and VRF.

Table 3–13 Layer 3 Routing and VRF Enabling

Action	Command
Enable VRF.	`(xpShell):xps:l3)l3_set_intf_vrf devId l3IntfId vrfId`

Enable UC routing.	`(xpShell):xps:l3)l3_set_intf_ipv4_uc_routing_en devId l3IntfId enable`
	`(xpShell):xps:l3)l3_set_intf_ipv6_uc_routing_en devId l3IntfId enable`

Enable multicast routing.	`(xpShell):xps:l3)l3_set_intf_ipv4_mc_routing_en devId l3IntfId enable`
	`(xpShell):xps:l3)l3_set_intf_ipv6_mc_routing_en devId l3IntfId enable`

3.2.4 Examples

The following examples show programming of a Layer 3 feature using xpShell commands. In these examples, packets are routed between VLAN 1 and VLAN 2.

Table 3–14 Example: Layer 3 Programming

Action	Command
Add ingress router MAC address.	`(xpShell):xps:vlan)l3_add_ingress_router_mac 0 00:aa:aa:aa:aa:00`

Create L3 interface on VLAN 1 with router MAC.	`(xpShell):xps:l3)l3_create_vlan_intf 1`
	`(xpShell):xps:l3)l3_add_intf_ingress_router_mac 0 65537 00:aa:aa:aa:aa:00`
	`(xpShell):xps:l3)l3_set_egress_router_mac_m_sbs 0 00:aa:aa:aa:aa:00`
	`(xpShell):xps:l3)l3_set_intf_egress_router_mac_lsb 0 65537 0`
	`(xpShell):xps:l3)l3_set_intf_vrf 0 65537 1`
	`(xpShell):xps:l3)l3_set_intf_ipv4_uc_routing_en 0 65537 1`

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Table 3–14 Example: Layer 3 Programming

Action	Command
Create L3 interface on VLAN 2 with router MAC.	(xpShell):xps:l3)**l3_create_vlan_intf 2** (xpShell):xps:l3)**l3_add_intf_ingress_router_mac 0 65538 00:aa:aa:aa:aa:00** (xpShell):xps:l3)**l3_set_egress_router_mac_m_sbs 0 00:aa:aa:aa:aa:00** (xpShell):xps:l3)**l3_set_intf_egress_router_mac_lsb 0 65538 0** (xpShell):xps:l3)**l3_set_intf_vrf 0 65538 1** (xpShell):xps:l3)**l3_set_intf_ipv4_uc_routing_en 0 65538 1**
Create Next Hop.	(xpShell):xps:l3)**l3_create_route_next_hop 1** (xpShell):xps:l3)**l3_set_route_next_hop 0 0 1 2 0 65538 00:11:22:22:11:00 3 0**
Add a route.	(xpShell):xps:l3)**l3_add_ip_route_entry 0 1 0 10.20.20.0 0.0.0.0 24 1 0**

3.3 Multicast (L2)

The following xpShell commands program the multicast feature. .

NOTE: VLAN must be created before multicast_create_l2_interface_list.

Table 3–15 Multicast Programming

#	Action	Command
1.	Create VLAN.	(xpShell):xps:multicast:vlan)**vlan_create** *devid vlanId*
2.	Create interface list.	(xpShell):xps:multicast)**multicast_create_l2_interface_list** *vlanId*
3.	Add interface list to device.	(xpShell):xps:multicast)**multicast_add_l2_interface_list_to_device** *devId l2IntfListId*
4.	Add bridge entry.	(xpShell):xps:multicast)**multicast_add_i_pv4_bridge_entry** *devId bdId sourceAddress groupAddress multicastVifIdx mirrorMask countMode counterIdx isControl isStatic pktCmd*
5.	Call API vlan_add_interface to add all intfId used to corresponding VLAN.	(xpShell):xps:multicast)**multicast_add_interface_to_l2_interface_list** *devId ifListId intfId*

Cavium Confidential For
Chen Peng
Teraspek
01/24/201

Example

Table 3–16 Example: Multicast Programming

#	Action	Command
1.	Create VLAN 1.	(xpShell):xps:vlan)**vlan_create 0 1**
2.	Create L2 interface list.	(xpShell):xps:multicast)**multicast_create_l2_interface_list 1** Input arguments are: vlanId = 1
3.	Add interface list to device.	(xpShell):xps:multicast)**multicast_add_l2_interface_list_to_device 0 55311**
4.	Add IPv4 bridge entry.	(xpShell):xps:multicast)**multicast_add_i_pv4_bridge_entry 0 1 10.2.168.192 10.0.0.239 55311 0 1 0 0 0 1**

Table 3–16 Example: Multicast Programming

#	Action	Command
5.	Add interface ID 1 to layer interface list.	`(xpShell):xps:multicast)`**`multicast_add_interface_to_l2_interface_list 0 55311 1`**

6.	**(Optional)** Add other interfaces IDs if needed. Before adding interface ID 2 to multicast list, interface ID 2 is added to VLAN using `vlan_add_interface`**.**	`(xpShell):xps:vlan)`**`vlan_add_interface 0 1 2 0`**
		`(xpShell):xps:multicast)`**`multicast_add_interface_to_l2_interface_list 0 55311 2`**

3.4 MPLS

3.4.1 Prerequisites

Before setting the MPLS VPN configuration (`mpls_set_vpn_config`), the following sequence of steps must be executed.

Table 3–17 Prerequisites to Set MPLS VPN configuration

#	Action	Command
1.	Create VPN interface ID.	`(xpShell):xps:l3)`**`l3_create_vpn_intf`**
2.	Initialize given VPN layer 3 interface ID.	`(xpShell):xps:l3)`**`l3_init_vpn_intf`** *`devId l3IntfId`*
3.	Bind interface ID to label.	`(xpShell):xps:l3)`**`l3_bind_vpn_intf_to_label`** *`devId label l3Intfld`*
4.	Set MPLS VPN configuration.	`(xpShell):xps:mpls)`**`mpls_set_vpn_config`** *`devId vpnLabel flagg pktCmd countMode cntId paclId raclId`*

Before setting the next hop data (`mpls_set_tunnel_next_hop_data`), the following sequence of steps must have been executed.

Table 3–18 Prerequisites to Set Next Hop

#	Action	Command
1.	Create STP.	`(xpShell):xps:stp)`**`stp_create`**
2.	Create VLAN.	`(xpShell):xps:vlan)`**`vlan_create`** *`devId vlanId`*
3.	Create VLAN config	`(xpShell):xps:vlan)`**`vlan_set_config`** *`devId vlanId stpId countMode enableMirror mirrorAnalyzerId saMissCmd bcCmd unknownUcCmd arpBcCmd ipv4McbridgeMode ipv6McbridgeMode unRegMcCmd`*
4.	Add VLAN interface.	`(xpShell):xps:vlan)`**`vlan_add_interface`** *`devId vlanId intfId tagType`*
5.	Create L3 VLAN interface.	`(xpShell):xps:l3)`**`l3_create_vlan_intf`** *`vlanId`*

6.	Create L3 next route next hop.	`(xpShell):xps:l3)`**`l3_create_route_next_hop`** *`nhEcmpSize`*
		`(xpShell):xps:l3)`**`l3_set_route_next_hop`** *`devId nhId pktCmd serviceInstId vpnLabel propTTL l3InterfaceId macDa egressIntfId reasonCode`*

Cavium Confidential For
Chen Peng
Feraspek
3/24/2018

3.4.2 Programming MPLS

The MPLS feature is configured by completing the following xpShell steps.

Table 3–19 Multicast Programming

#	Action	Command
1.	Create MPLS tunnel.	(xpShell):xps:mpls)**mpls_create_tunnel_interface** *mplsTnlId*
2.	Add tunnel entry into tunnel database.	(xpShell):xps:mpls)**mpls_add_tunnel_entry** *devId isP2MP numOfLables firstLabel secondLabel mplsTnlId*
3.	Set MPLS VPN configuration.	(xpShell):xps:mpls)**mpls_set_vpn_config** *devId vpnLabel flagg pktCmd countMode cntId paclId raclId*
4.	Set MPLS tunnel Next Hop data.	(xpShell):xps:mpls)**mpls_set_tunnel_next_hop_data** *devId mplsTnlId nextHopId*
5.	Add label entry.	(xpShell):xps:mpls)**mpls_add_label_entry** *devId keyLabel pktCmd mirrorMask countMode counterId propTTL swapLabel mplsOper l3InterfaceId macDa egressIntfId*
6.	Set tunnel configuration.	(xpShell):xps:mpls)**mpls_set_tunnel_config** *devId mplsTnlId p2pLabelTnl.propTTL countMode cntId p2mpLabelTnl.propTTL countMode cntId isBudNode*

Example

Table 3–20 Example: MPLS Programming

#	Action	Command
1.	Create MPLS tunnel.	(xpShell):xps:mpls)**mpls_create_tunnel_interface** mplsTnlId = 47104 The MPLS tunnel ID mplsTnlId = 47104 is created.
2.	Add tunnel entry into tunnel database.	(xpShell):xps:mpls)**mpls_add_tunnel_entry 0 0 1 5555 0 47104**
3.	Set MPLS VPN configuration.	(xpShell):xps:mpls)**mpls_set_vpn_config 0 5555 1 1 0 1 31 51**
4.	Set MPLS tunnel Next Hop data.	(xpShell):xps:mpls)**mpls_set_tunnel_next_hop_data 0 47104 1**
5.	Add label entry.	(xpShell):xps:mpls)**mpls_add_label_entry 0 1 1 1 1 1 1 1 66046 04:04:05:05:06:06 51**
6.	Set tunnel configuration.	(xpShell):xps:mpls)**mpls_set_tunnel_config 0 47104 1 2 1 1 5 6 1**

3.5 NAT

The NAT is configured using the following xpShell commands in sequence.

Table 3–21 NAT Programming

#	Action	Command
1.	Initialize NAT.	(xpShell):xps:nat)**init_nat**
2.	Device initialization for NAT.	(xpShell):xps:nat)**device_init_nat** *devId initType*

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table 3–21 NAT Programming

#	Action	Command
3.	Add external entry.	(xpShell):xps:nat)**nat_add_external_entry** *devId index SrcAddress SrcPort DestAddress DestPort Bd Flag Protocol SrcAddress SrcPort DestAddress DestPort Bd Flag Protocol SIPAddress srcPort DIPAddress destPort vif*
4.	Add NAT entry.	(xpShell):xps:nat)**nat_add_entry** *devId index SrcAddress SrcPort DestAddress DestPort Bd Flag Protocol SrcAddress SrcPort DestAddress DestPort Bd Flag Protocol SIPAddress srcPort IPAddress destPort vif*
5.	Add internal entry.	(xpShell):xps:nat)**nat_add_internal_entry** *devId index SrcAddress SrcPort DestAddress DestPort Bd Flag Protocol SrcAddress SrcPort DestAddress DestPort Bd Flag Protocol SIPAddress srcPort DIPAddress destPort vif*
6.	Add filter rule.	(xpShell):xps:nat)**nat_add_filter_rule** *devId index SrcAddress SrcPort DestAddress DestPort Bd Flag Protocol SrcAddress SrcPort DestAddress DestPort Bd Flag Protocol SIPAddress srcPort DIPAddress destPort vif*
7.	Set NAT in MDT table.	(xpShell):xps:nat)**set_mdt_nat_config** *devId index value*
8.	Get NAT table entry.	(xpShell):xps:nat)**nat_get_entry** *devId index*
9.	Delete NAT table entry if needed using index.	(xpShell):xps:nat)**nat_del_entry_data** *devId index*

Example

The following example programs the NAT feature.:

Table 3–22 Example: Programming NAT Feature

#	Action	Command
1.	Initialize NAT.	(xpShell):xps:nat)**init_nat**
2.	Device initialization for NAT.	(xpShell):xps:nat)**device_init_nat 0 0**
3.	Add external entry.	(xpShell):xps:nat)**nat_add_external_entry 0 0 192.168.2.62 0 173.194.36.18 0 894 0 0 255.255.255.255 65535 255.255.255.255 65535 65535 511 255 27.109.14.158 0 173.194.36.18 0 100**
4.	Add NAT entry.	(xpShell):xps:nat)**nat_add_entry 0 0 192.168.2.62 0 173.194.36.18 0 894 0 0 255.255.255.255 65535 255.255.255.255 65535 65535 511 255 27.109.14.158 0 173.194.36.18 0 100**
5.	Add internal entry.	(xpShell):xps:nat)**nat_add_internal_entry 0 0 192.168.2.62 0 173.194.36.18 0 894 0 0 255.255.255.255 65535 255.255.255.255 65535 65535 511 255 27.109.14.158 0 173.194.36.18 0 100**
6.	Add filter rule.	(xpShell):xps:nat)**nat_add_filter_rule 0 0 192.168.2.62 0 173.194.36.18 0 894 0 0 255.255.255.255 65535 255.255.255.255 65535 65535 511 255 27.109.14.158 0 173.194.36.18 0 100**
7.	Set NAT in MDT table.	(xpShell):xps:nat)**set_mdt_nat_config 0 0 0**

Cavium Confidential For Chen Peng Teraspek 01/24/2018

3.6 QoS

3.6.1 AQM

The following AQM APIs are used to configure AQM functionality.

Table 3–23 AQM Programming

Action	Command

Create AQM profile.	The created profile ID is used in binding the port to profile ID.
	`(xpShell):xps:qos)qos_aqm_create_profile devId`
	Set shared pool parameters:
	`(xpShell):xps:qos)qos_aqm_set_port_shared_pool_id devID devPort sharedPoolId`
	`(xpShell):xps:qos)qos_aqm_set_queue_shared_pool_enable devId devPort queueNum enable`
	Set DTCP parameters:
	`(xpShell):xps:qos)qos_aqm_set_port_dctcp_enable devId profileId enable`
	`(xpShell):xps:qos)qos_aqm_configure_port_dctcp_mark_threshold devId profileId markThreshold`
	Set different thresholds:
	`(xpShell):xps:qos)qos_aqm_set_global_packet_threshold devId Threshold`
	`(xpShell):xps:qos)qos_aqm_set_global_page_threshold devId Threshold`
	`(xpShell):xps:qos)qos_aqm_set_shared_pool_threshold devId sharedPoolId Threshold`
	`(xpShell):xps:qos)qos_aqm_configure_port_packet_tail_drop_threshold devId devPort threshold`
	`(xpShell):xps:qos)qos_aqm_configure_port_page_tail_drop_threshold devId profileId lengthMaxThreshold`

Bind port profile to a port.	`(xpShell):xps:qos)qos_aqm_port_bind_profile_to_port devId devPort queueNum profileId`

Create and bind queue profile if required.	`(xpShell):xps:qos)qos_aqm_create_aqm_q_profile devId`
	`(xpShell):xps:qos)qos_aqm_bind_aqm_q_profile_to_queue devId devPort queueNum profileId`

Display AQM profile.	`(xpShell):xps:qos)qos_aqm_display_aqm_profile devId profileId`

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

3.6.2 Scheduling

The following scheduling APIs are used to configure scheduling functionality.

Table 3–24 Scheduling Programming

Action	Command
Enable DWRR scheduling.	`(xpShell):xps:qos)qos_set_queue_scheduler_dwrr devId portNum queueNum enable`
Enable DWRR weight scheduling.	`(xpShell):xps:qos)qos_set_queue_scheduler_dwrr_weight devId portNum queueNum weight`
Enable Strict Priority scheduling.	`(xpShell):xps:qos)qos_set_queue_scheduler_sp devId portNum queueNum enable`

3.6.3 Shaping

The following shaping APIs are used to configure shaping functionality.

Table 3–25 Shaping Programming

Action	Command
Set port shaper on a port.	(xpShell):xps:qos)**qos_shaper_configure_port_shaper** *devId portNum rateKbps maxBurstByteSize*
Enable port shaper on a port.	(xpShell):xps:qos) **qos_shaper_set_port_shaper_enable** *devId devPort enableShaper*
Set port maximum burst multiplier.	(xpShell):xps:qos)**qos_set_port_shaper_max_burst_multiplier** *devId portNum maxBurstMult*
Set the MTU configuration for port shaper.	(xpShell):xps:qos)**qos_shaper_set_port_shaper_mtudevId** *mtuInBytes*
Get shaper UPD rate.	(xpShell):xps:qos)**qos_get_port_shaper_upd_rate** *devId portNum*
Get shaper number of tokens.	(xpShell):xps:qos)**qos_shaper_get_port_shaper__table_index** *devId devPort*
Get shaper MTU configuration.	(xpShell):xps:qos)**qos_shaper_get_port_shaper_mtu** *devId*

3.6.4 Priority Flow Control

The following APIs are used to configure XON and XOFF threshold to support control flow.

Table 3–26 Priority Flow Control Programming

Action	Command
Set port XON threshold.	(xpShell):xps:qos)**qos_fc_set_port_pfc_xon_threshold** *devId portPfcProfileId xonThreshold*
Set port XOFF threshold.	(xpShell):xps:qos)**qos_fc_set_port_pfc_xoff_threshold**

3.6.5 QOS Counter

The following APIs are used to enable and read counters.

Table 3–27 QoS Programming

Action	Command
Enable clear on read for packet forward counters.	This configures clear on read capabilities for each q packet forward counter. (xpShell):xps:qos)**qos_aqm_enable_fwd_pkt_count_clear_on_read** *devId enable*

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Table 3–27 QoS Programming

Action	Command
Enable clear on read for page forward counters.	(xpShell):xps:qos)**qos_aqm_enable_fwd_pkt_page_count_clear_on_read** *devId enable*
Return requested information.	(xpShell):xps:qos)**qos_get_queue_fwd_packet_count_for_port** *devId port queue count wrap* (xpShell):xps:qos)**qos_get_queue_drop_packet_count_for_port** *devId port queue* (xpShell):xps:qos)**qos_get_queue_fwd_page_count_for_port** *devId port queue* (xpShell):xps:qos)**qos_get_queue_drop_page_count_for_port** *devId port queue* (xpShell):xps:qos)**qos_get_current_queue_packet_depth** *devId port queue* (xpShell):xps:qos)**qos_get_current_queue_page_depth** *devId port queue* (xpShell):xps:qos)**qos_get_queue_old_page_length** *devId port queue* (xpShell):xps:qos)**qos_get_pfc_page_count** *devId port priority*

3.6.6 Marking

The following APIs are used to configure marking functionality.

Table 3–28 QoS Programming

Action	Command
Set traffic class for Layer 2.	(xpShell):xps:qos)**qos_port_ingress_set_traffic_class_for_l2_qos_profile** *devId profile pcpVal deiVal tc*
Set traffic class for Layer 3.	(xpShell):xps:qos)**qos_port_ingress_set_traffic_class_for_l3_qos_profile** *devId profile dscpVal tc*
Set traffic class for MPLS.	(xpShell):xps:qos)**qos_port_ingress_set_traffic_class_for_mpls_qos_profile** *devId profile expVal tc*
Set drop precedence for Layer 2.	(xpShell):xps:qos)**qos_port_ingress_set_drop_precedence_for_l2_qos_profile** *devId profile pcpVal deiVal dp*
Set drop precedence for Layer 3.	(xpShell):xps:qos)**qos_port_ingress_set_drop_precedence_for_l3_qos_profile** *devId profile dscpVal dp*
Set drop precedence for MPLS.	(xpShell):xps:qos)**qos_port_ingress_set_drop_precedence_for_mpls_qos_profile** *devId profile expVal dp*
Set default Layer 2 QoS priority on a port.	(xpShell):xps:qos)**qos_port_ingress_set_port_default_l2_qos_priority** *devId devPort pcpVal deiVal*

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table 3–28 QoS Programming

Action	Command
Set default Layer 3 QoS priority on a port	`(xpShell):xps:qos)qos_port_ingress_set_port_default_l3_qos_priority devId devPort dscpVal`

Return requested information.	`(xpShell):xps:qos)qos_port_ingress_get_traffic_class_for_l2_qos_profile devId profile pcpVal deiVal`
	`(xpShell):xps:qos)qos_port_ingress_get_traffic_class_for_l3_qos_profile devId profile dscpVal`
	`(xpShell):xps:qos)qos_port_ingress_get_traffic_class_for_mpls_qos_profile devId profile expVal`
	`(xpShell):xps:qos)qos_port_ingress_get_drop_precedence_for_l2_qos_profile devId profile pcpVal deiVal`
	`(xpShell):xps:qos)qos_port_ingress_get_drop_precedence_for_l3_qos_profile devId profile dscpVal`
	`(xpShell):xps:qos)qos_port_ingress_get_drop_precedence_for_mpls_qos_profile devId profile expVal`
	`(xpShell):xps:qos)qos_port_ingress_get_port_default_l2_qos_priority devId devPort`
	`(xpShell):xps:qos)qos_port_ingress_get_port_default_l3_qos_priority devId devPort`

Example 1: Layer 2 (COS)-based QoS

This example sets Layer 2 (COS)-based QoS on port number 2.

Table 3–29 Example 1: Layer 2 (COS)-based QoS

#	Action	Command
1.	Set Layer 2 trust on a port.	Set trust only the L2 Priorities and keeps the incoming PCP/DEI. Traffic class and drop precedence must be set separately. `(xpShell):xps:qos)qos_port_ingress_set_trust_l2_for_port 0 2`
2.	Set port drop precedence.	`(xpShell):xps:qos)qos_port_ingress_set_port_default_drop_precedence 0 2 3`
3.	Set Port Traffic class.	`(xpShell):xps:qos)qos_port_ingress_set_port_default_traffic_class 0 2 2`

Example 2: Layer 3 (DSCP)-based QoS

This example sets Layer 3 (DSCP) based QoS on port number 2.

Table 3–30 Example 2: Layer 3 (DSCP)-based QoS

#	Action	Command
1.	Set Layer 3 trust on a port.	This profile by default trusts only the L3 priorities and keeps the incoming DSCP. Traffic class and drop precedence must be set separately. `(xpShell):xps:qos)qos_port_ingress_set_trust_l3_for_port 0 2`
2.	Set port drop precedence.	`(xpShell):xps:qos)qos_port_ingress_set_port_default_drop_precedence 0 2 3`
3.	Set port traffic class.	`(xpShell):xps:qos)qos_port_ingress_set_port_default_traffic_class 0 2 2`

Cavium Confidential For Chen Peng Teraspek 01/24/2018

3.7 Link

The following xpShell commands verify link status.

Table 3–31 Link Commands

Action	Command
Initialize link.	(xpShell):xps:link)**link_init**
Add device to link.	All links come up after executing the following command: (xpShell):xps:link)**link_add_device** *devId initType* cpuRestart
Remove link initialization.	(xpShell):xps:link)**link_de_init**
Remove device from link.	(xpShell):xps:link)**link_remove_device**

3.8 ACM

The following commands configure ACM functionality.

Table 3–32 ACM Programming

#	Action	Command
1.	Initialize ACM.	(xpShell):xps:acm)**acm_init**
2.	Add device to ACM.	(xpShell):xps:acm)**acm_add_device** *devId initType*
3.	Configure sampling.	(xpShell):xps:acm)**acm_set_sampling_config** *devId index nSample mBase mExpo*
4.	Verify sampling configuration.	(xpShell):xps:acm)**acm_get_sampling_config** *devId index*
5.	Set sampling state.	(xpShell):xps:acm)**acm_set_sampling_state** *devId index totalCnt interEventCnt interSampleStart*
6.	Set bucket configuration parameters.	(xpShell):xps:acm)**acm_cnt_set_global_config_bucketization** *devId enable startRange endRange numBkts granularity addAddr bktUseAddr*
7.	Set configuration mode.	(xpShell):xps:acm)**acm_cnt_set_global_config_mode_pol** *devId refreshEnable unitTime refrTimeGranularity updateWeight billingCntrEnable*
8.	Print counter values.	(xpShell):xps:acm)**acm_print_counter_value** *devId countIndex printZeros*
9.	Get counter values.	(xpShell):xps:acm)**acm_get_counter_value** *devId countIndex*

The following commands are used to de-initialize and remove the device.

Table 3–33 ACM Commands

Action	Command
De-initialize device.	(xpShell):xps:acm)**acm_de_init**
Remove device.	(xpShell):xps:acm)**acm_remove_device** *devId*

Cavium Confidential For
Chen Peng
Teraspek
01/24/201

3.8.1 sFlow

The following commands configure sFlow.

Table 3–34 Configuring sFlow

#	Action	Command
1.	Initialize sflow.	`(xpShell):xps:sflow)`**`sflow_init`**
2.	Add device to sflow	`(xpShell):xps:sflow)`**`sflow_add_device`** *`devId initType`*
3.	Set sflow interface ID.	`(xpShell):xps:sflow)`**`sflow_set_intf_id`** *`devId intfId`*
4.	Verify sflow interface ID.	`(xpShell):xps:sflow)`**`sflow_get_intf_id`** *`devId`*
5.	Set sflow packet command.	`(xpShell):xps:sflow)`**`sflow_set_pkt_cmd`** *`devId pktCmd`*
6.	Verify sflow packet command.	`(xpShell):xps:sflow)`**`sflow_get_pkt_cmd`** *`devId`*
7.	Set port sampling configuration.	`(xpShell):xps:sflow)`**`sflow_set_port_sampling_config`** *`portIntfId nSample mBase mExpo`*
8.	Verify port sampling configuration.	`(xpShell):xps:sflow)`**`sflow_get_port_sampling_config`** *`portIntfId`*
9.	Enable port sampling.	`(xpShell):xps:sflow)`**`sflow_enable_port_sampling`** *`portIntfId enable`*
10.	Verify port sampling status.	`(xpShell):xps:sflow)`**`sflow_get_port_sampling_status`** *`portIntfId`*

The following commands are used to de-initialize and remove the device from sFlow.

Table 3–35 De-initialize/Remove Device from sFlow

Action	Command
De-initialize the device.	`(xpShell):xps:sflow)`**`sflow_de_init`**
Remove the device.	`(xpShell):xps:sflow)`**`sflow_remove_device`** *`devId`*

3.8.2 Policer

The following sequence of steps configures policer.

Table 3–36 Policer Programming

#	Action	Command
1.	Add policer entry.	`(xpShell):xps:policer)`**`policer_add_entry`** *`devId index cir pir cbs pbs colorAware dropRed dropYellow updateResultRed updateResultYellow updateResultGreen polResult`*
2.	Verify policer entry.	`(xpShell):xps:policer)`**`policer_get_entry`** *`devId index`*
3.	Enable port policing.	`(xpShell):xps:policer)`**`policer_enable_port_policing`** *`portIntfId enable`*
4.	Add port for policing.	`(xpShell):xps:policer)`**`policer_add_port_policing_entry`** *`portIntfId cir pir cbs pbs colorAware dropRed dropYellow updateResultRed updateResultYellow updateResultGreen polResult`*
5.	Set policing based on color.	`(xpShell):xps:policer)`**`policer_set_result_by_color`** *`devId index resultType color dp tc pcp dei dscp`*

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Table 3–36 Policer Programming

#	Action	Command
6.	Set policing based on type.	`(xpShell):xps:policer)policer_set_result_by_type devId index resultType Red_dp Red_tc Red_pcp Red_dei Red_dscp Red_exp Yellow_dp Yellow_tc Yellow_pcp Yellow_dei Yellow_dscp Yellow_exp Green_dp Green_tc Green_pcp Green_dei Green_dscp Green_exp`
7.	Set attribute.	`(xpShell):xps:policer)policer_set_attribute devId index field data`

The following commands are used to de-initialize and remove policing.

Table 3–37 De-initialize/Remove Policing

Action	Command
De-initialize policing.	`((xpShell):xps:policer)policer_de_init`
Remove policing entry.	`(xpShell):xps:policer)policer_remove_port_policing_entry portIntfId`
Remove device from policing.	`(xpShell):xps:policer)policer_remove_device devId`

3.9 Mirroring

The following sequence of steps configures mirroring.

Table 3–38 Mirror Programming

#	Action	Command
1.	Initialize the mirroring.	`(xpShell):xps:mirror)mirror_init`
2.	Add device for mirroring.	`(xpShell):xps:mirror)mirror_add_device devId initType = INIT_COLD`
3.	Create analyzer session.	`(xpShell):xps:mirror)mirror_create_analyzer_session TYPE DATA DIR`
4.	Add analyzer ID created in previous command to analyzer interface ID.	`(xpShell):xps:mirror)mirror_add_analyzer_interface analyzerId analyzerIntfId`
5.	Write analyzer session using analyzer ID.	`(xpShell):xps:mirror)mirror_write_analyzer_session deviceId analyzerId`

The analyzer session and interface are removed as follows. .

Table 3–39 De-initialize/Remove Mirroring

Action	Command
Remove analyzer interface.	`(xpShell):xps:mirror)mirror_remove_analyzer_interface deviceId analyzerId analyzerIntfId`
Delete analyzer interface.	`(xpShell):xps:mirror)mirror_delete_analyzer_interface analyzerId analyzerIntfId`
Remove analyzer session.	`(xpShell):xps:mirror)mirror_remove_analyzer_session deviceId analyzerId`
Destroy analyzer session using analyzer ID.	`(xpShell):xps:mirror)mirror_destroy_analyzer_session analyzerId`

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Examples The following examples configure the mirroring feature.

Table 3–40 Example 1: Mirroring

#	Action	Command
1.	Initialize the mirroring.	`(xpShell):xps:mirror)`**`mirror_init`**
2.	Add device for mirroring.	`(xpShell):xps:mirror)`**`mirror_add_device 0 INIT_COLD`**
3.	Create analyzer session.	`(xpShell):xps:mirror)`**`mirror_create_analyzer_session 0`**
4.	Add analyzer ID created in previous command to analyzer interface ID.	`(xpShell):xps:mirror)`**`mirror_add_analyzer_interface 0 12`**
5.	Write analyzer session using analyzer ID.	`(xpShell):xps:mirror)`**`mirror_write_analyzer_session 0 0`**

Table 3–41 Example 2: Mirroring

#	Action	Command

Basic pre-configuration for mirroring functionality verification:
Create VLAN **20**, add ports **1** and **2** to created VLAN, enable VLAN forwarding (`bcCmd`, `unknownUcCmd` flags).
`(xpShell):xps:vlan)vlan_create 0 20`
`(xpShell):xps:vlan)vlan_add_interface 0 20 1 0`
`(xpShell):xps:vlan)vlan_add_interface 0 20 2 0`
`(xpShell):xps:vlan)vlan_set_config 0 20 1 0 0 0 0 0 0 1 1 0 0 0 0`		

Basic Mirroring configuration scenarios		

1.	Create **ingress Mirroring Session**: [0-Egress, 1-Ingress].	`(xpShell):xps:mirror)mirror_create_analyzer_session 1`
		`output: analyzerId = 0`

2.	Add **Analyzer Interface (as port 4)** to created **mirroring session 0**.	`(xpShell):xps:mirror)mirror_add_analyzer_interface 0 4`
3.	Write **analyzer session 0**.	`(xpShell):xps:mirror)mirror_write_analyzer_session 0 0`
Enable configured mirroring session for specified interface [PORT, VLAN, LAG]		

4.	**Mirror PORT:**
	Enable/disable **mirroring session 0** on **port 1**.
		`(xpShell):xps:port)port_enable_mirroring 1 0`
		`port_disable_mirroring 1 0`

5.	**Mirror VLAN:**
	Enable/disable **mirroring session 0** on **VLAN 20**.
	[enable: 1, disable: 0]
		`(xpShell):xps:vlan)vlan_set_mirror_to_analyzer 0 20 1 0`
		`(xpShell):xps:vlan)vlan_set_mirror_to_analyzer 0 20 0 0`
		
		`----------------------------------------------------- Mirror LAG -`
		`------------------------------------------------------`

Basic pre-configuration for mirroring on LAG interface verification:
Create LAG, add two ports `6` and `7`, add LAG to VLAN `20`.
`(xpShell):xps:lag)lag_create`
`output:` `lagIntf = 51200`
`(xpShell):xps:lag)lag_add_port 51200 6`
`(xpShell):xps:lag)lag_add_port 51200 7`
`(xpShell):xps:lag)lag_deploy 0 51200 1`
`(xpShell):xps:vlan)vlan_add_interface 0 20 51200 0`		

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table 3–41 Example 2: Mirroring

#	Action	Command
6.	Enable/disable **mirroring session 0** on **LAG 51200**.	(xpShell):xps:lag)lag_enable_mirroring 0 51200 0 (xpShell):xps:lag)lag_disable_mirroring 0 51200 0
MIRROR manipulations		
7.	Remove **analyzer interface 4** from **mirroring session 0**.	(xpShell):xps:mirror)mirror_remove_analyzer_interface 0 0 4 (xpShell):xps:mirror)mirror_delete_analyzer_interface 0 4
8.	Remove **mirroring session 0** from DUT.	(xpShell):xps:mirror)mirror_remove_analyzer_session 0 0 (xpShell):xps:mirror)mirror_destroy_analyzer_session 0

3.10 IACL

The sequence for IACL configuration and programming is as given below.

Table 3–42 IACL Programming

#	Action	Command
1.	Create IACL table for the desired lookup types.	(xpShell):xps:iacl)**iacl_create_table** *devId numTables iaclTblType keySize numDb*
2.	Set the key format for BACL, PACL, and RACL tables.	**BACL**: (xpShell):xps:iacl)**iacl_define_bacl_key** *devId keyType numFlds isValid flds* **PACL**: (xpShell):xps:iacl)**iacl_define_pacl_key** *devId keyType numFlds isValid flds* **RACL**: (xpShell):xps:iacl)**iacl_define_racl_key** *devId keyType numFlds isValid flds*

3.	Enable IACL configuration with IACL ID programming on the interface.	Set ACL Enable fNum is 22:
		(xpShell):xps:port) **port_set_field** *devId PortIfID fNum{22} fdata{1}*
		To set port ACL ID. Set ACL ID fNum is 24:
		(xpShell):xps:port) **port_set_field** *devId PortIfId fNum{24} fdata {1}*

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table 3–42 IACL Programming

#	Action	Command
4.	Program the IACL rule on BACL, PACL, and RACL tables.	**BACL:** (xpShell):xps:iacl)**iacl_write_bacl_entry** *devId camIndex numFlds isValid type isTerminal enPktCmdUpd enRedirectToEvif enRsnCodeUpd enPolicer enCnt enMirrorSsnUpd remarkTcp remarkDscp remarkPcp pktCmd TC mirrorSessionId encapType eVifId policerId rsnCode PCP DSCP instanceId flds value mask flds value mask ...append flds value mask for number of keys to be programmed* Example: iacl_write_bacl_entry 0 0 6 1 XP_IACL_V4_TYPE 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 19 0 0 852 XP_IACL_KEY_TYPE_V4,0,0 XP_IACL_ID,1,0 XP_IACL_CTAG_VID_DEI_PCP,21763,0 XP_IACL_MAC_DA,0:0:0:0:0:0,0 XP_IACL_MAC_SA,0:0:0:0:0:0,0 **PACL:** (xpShell):xps:iacl)**iacl_write_pacl_entry** *devId camIndex numFlds isValid type isTerminal enPktCmdUpd enRedirectToEvif enRsnCodeUpd enPolicer enCnt enMirrorSsnUpd remarkTcp remarkDscp remarkPcp pktCmd TC mirrorSessionId encapType eVifId policerId rsnCode PCP DSCP instanceId flds value mask flds value mask ...append flds value mask for number of keys to be programmed* Example: iacl_write_pacl_entry 0 0 6 1 XP_IACL_V4_TYPE 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 19 0 0 852 XP_IACL_KEY_TYPE_V4,0,0 XP_IACL_ID,1,0 XP_IACL_CTAG_VID_DEI_PCP,21763,0 XP_IACL_MAC_DA,0:0:0:0:0:0,0 XP_IACL_MAC_SA,0:0:0:0:0:0,0 **RACL:** (xpShell):xps:iacl)**iacl_write_racl_entry** *devId camIndex numFlds isValid type isTerminal enPktCmdUpd enRedirectToEvif enRsnCodeUpd enPolicer enCnt enMirrorSsnUpd remarkTcp remarkDscp remarkPcp pktCmd TC mirrorSessionId encapType eVifId policerId rsnCode PCP DSCP instanceId flds value mask flds value mask ...append flds value mask for number of keys to be programmed* Example: iacl_write_racl_entry 0 0 6 1 XP_IACL_V4_TYPE 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 19 0 0 852 XP_IACL_KEY_TYPE_V4,0,0 XP_IACL_ID,1,0 XP_IACL_CTAG_VID_DEI_PCP,21763,0 XP_IACL_MAC_DA,0:0:0:0:0:0,0 XP_IACL_MAC_SA,0:0:0:0:0:0,0
5.	Program the IACL data on BACL, PACL, and RACL tables.	**BACL:** (xpShell):xps:iacl)**iacl_write_bacl_data** *devId camIndex isTerminal enPktCmdUpd enRedirectToEvif enRsnCodeUpd enPolicer enCnt enMirrorSsnUpd remarkTcp remarkDscp remarkPcp pktCmd TC mirrorSessionId encapType eVifId policerId rsnCode PCP DSCP* **PACL:** (xpShell):xps:iacl)**iacl_write_pacl_data** *devId camIndex isTerminal enPktCmdUpd enRedirectToEvif enRsnCodeUpd enPolicer enCnt enMirrorSsnUpd remarkTcp remarkDscp remarkPcp pktCmd TC mirrorSessionId encapType eVifId policerId rsnCode PCP DSCP* **RACL:** (xpShell):xps:iacl)**iacl_write_racl_data** *devId camIndex isTerminal enPktCmdUpd enRedirectToEvif enRsnCodeUpd enPolicer enCnt enMirrorSsnUpd remarkTcp remarkDscp remarkPcp pktCmd TC mirrorSessionId encapType eVifId policerId rsnCode PCP DSCP*
6.	Set IACL rule valid.	(xpShell):xps:iacl)**iacl_set_rule_valid** *devId iaclType index valid* (xpShell):xps:iacl)**iacl_set_rule_valid** *devId iaclType index valid* (xpShell):xps:iacl)**iacl_set_rule_valid** *devId iaclType index valid*

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Example

The following example configures the ACL feature.

Table 3–43 Example: ACL Configuration

Action	Command
ACL: Add VLAN	**For VLAN table**: (xpShell):xps:vlan)vlan_create 0 850 (xpShell):xps:stp)stp_create (xpShell):xps:vlan)vlan_set_config 0 850 0 1 1 0 0 0 0 1 0 0 0 0 0 0 (xpShell):xps:port)port_get_port_intf_id 0 68 (xpShell):xps:port)port_get_port_intf_id 0 76 (xpShell):xps:vlan)vlan_add_interface 0 850 68 1 (xpShell):xps:vlan)vlan_add_interface 0 850 76 1 (xpShell):xps:vlan)vlan_set_hairpin 0 850 68 0 (xpShell):xps:vlan)vlan_set_hairpin 0 850 76 0 (xpShell):xps:stp)stp_set_state 0 1 68 2 (xpShell):xps:stp)stp_set_state 0 1 76 2 (xpShell):xps:port)port_set_config 0 68 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 2 0 1 0 1 1 0 850 0 0 68 1 850 0 0 0 0
	For FDB configuration (xpShell):xps:fdb)fdb_add_entry 0 850 00:04:23:C5:43:C8 1 0 1 0 68 0 (xpShell):xps:fdb)fdb_add_entry 0 850 03:80:01:03:dd:e3 1 0 0 0 76 0
	For port configuration: (xpShell):xps:port)port_set_field 0 68 22 1 (xpShell):xps:port)port_set_field 0 68 24 0 (xpShell):xps:port)port_set_field 0 76 22 1 (xpShell):xps:port)port_set_field 0 76 24 1
	IACL Configuration (xpShell):xps:iacl)iacl_create_table 0 XP_IACL0 1 208 XP_IACL1 1 208 XP_IACL2 1 208 (xpShell):xps:iacl)iacl_define_pacl_key 0 XP_IACL_V4_TYPE 9 1 XP_IACL_KEY_TYPE_V4,XP_IACL_ID,XP_IACL_MAC_DA,XP_IACL_MAC_SA (xpShell):xps:iacl)iacl_write_pacl_data 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 19 0 0 850 (xpShell):xps:iacl)iacl_write_pacl_key 0 0 9 1 XP_IACL_V4_TYPE XP_IACL_KEY_TYPE_V4,XP_IACL_ID,XP_IACL_MAC_DA,XP_IACL_MAC_SA,XP_IACL_V4_ETHER_TYPE,XP_IACL_DIP_V4,XP_IACL_SIP_V4,XP_IACL_PROTOCOL,XP_IACL_DSCP_HAS_CTAG_STAG 0,1,03:80:01:03:dd:e3,00:04:23:C5:43:C8,0,0.0.0.0,0.0.0.0,0,63 0,0,0,0,1,1,1,0,63

3.11 Tunnel

This section describes the sequence of steps to program tunnels for different tunnel features.

3.11.1 Geneve Tunnel

To program the Geneve tunnel the following sequence of commands are required.

Table 3–44 Geneve Tunnel Programming

#	Action	Command
1.	Create the tunnel.	(xpShell):xps:geneve)**geneve_create_tunnel_interface** *lclEpIpAddr rmtEpIpAddr optionFormat*
2.	Add local endpoint.	(xpShell):xps:geneve)**geneve_add_local_endpoint** *devId localIp*
3.	Add tunnel entry.	(xpShell):xps:geneve)**geneve_add_tunnel_entry** *devId intfId*
4.	Set tunnel configuration.	(xpShell):xps:geneve)**geneve_set_tunnel_config** *devId tnlIntfId pktCmd paclEn paclId*

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Table 3–44 Geneve Tunnel Programming

#	Action	Command
5.	Set next hop data.	(xpShell):xps:geneve)**geneve_set_tunnel_next_hop_data** *devId tnlIntfId nhId*
6.	The tunnel configuration can be verified using following command.	(xpShell):xps:geneve)**geneve_get_tunnel_config** *devId tnlIntfId pktCmd paclEn paclId*
7.	Update next hop data.	(xpShell):xps:geneve)**geneve_update_tunnel_next_hop_data** *devId tnlIntfId*
8.	Bind the tunnel.	(xpShell):xps:geneve)**geneve_tunnel_bind_option** *devId lclEpIpAddr rmtEpIpAddr optionForma baseIntfId*

The following commands are used to remove and destroy tunnel interfaces.

Table 3–45 Remove/Destroy Tunnel Commands

Action	Command
Get tunnel configuration.	(xpShell):xps:geneve)**geneve_get_tunnel_config** *devId tnlIntfId pktCmd paclEn paclId*
Remove local endpoint.	(xpShell):xps:geneve)**geneve_remove_local_endpoint** *devId localIp*
Remove tunnel interface.	(xpShell):xps:geneve)**geneve_remove_tunnel_interface** *devId tnlIntfId*
Destroy tunnel interface.	(xpShell):xps:geneve)**geneve_destroy_tunnel_interface** *tnlIntfId*>

Example

Table 3–46 Example: Geneve Tunnel Programming

#	Action	Command
1.	Create the tunnel.	(xpShell):xps:geneve)**geneve_create_tunnel_interface**
2.	Add local endpoint.	(xpShell):xps:geneve)**geneve_add_local_endpoint 0 22.22.22.02**
3.	Add tunnel entry.	(xpShell):xps:geneve)**geneve_add_tunnel_entry 0 47104 22.22.22.02 21.21.21.02 0**
4.	Set tunnel configuration.	(xpShell):xps:geneve)**geneve_set_tunnel_config 0 47104 1 0 0**
5.	Set next hop data.	(xpShell):xps:geneve)**geneve_set_tunnel_next_hop_data 0 47104 0**
6.	The tunnel configuration can be verified using following command.	(xpShell):xps:geneve)**geneve_get_tunnel_config 0 47104 1 0 0**
7.	Update next hop data.	(xpShell):xps:geneve)**geneve_update_tunnel_next_hop_data 0 47104**
8.	Bind the tunnel.	(xpShell):xps:geneve)**geneve_tunnel_bind_option 0 22.22.22.02 21.21.21.02 0 47104**

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

3.11.2 VXLAN Tunnel

The sequence of commands to configure a VXLAN tunnel is as follows.

Table 3–47 VXLAN Tunnel Programming

#	Action	Command
1.	Create VXLAN tunnel interface.	(xpShell):xps:vxlan)**vxlan_create_tunnel_interface** *lclEpIpAddr rmtEpIpAddr* (xpShell):xps:vxlan)**vxlan_destroy_tunnel_interface** *tnlIntfId*
2.	Add tunnel entry.	(xpShell):xps:vxlan)**vxlan_add_tunnel_entry** *devId intfId*
3.	Add local VTEP.	(xpShell):xps:vxlan)**vxlan_add_local_vtep** *devId localIp*
4.	Add VNI.	(xpShell):xps:vxlan)**vxlan_add_vni** *devId vni vlanId*
5.	Set UDP port.	(xpShell):xps:vxlan)**vxlan_set_udp_port** *devId udpPort*
6.	Add MC tunnel entry.	(xpShell):xps:vxlan)**vxlan_add_mc_tunnel_entry** *devId tnlIntfId lclEpIpAddr rmtEpIpAddr l3IntfId portIntfId*
7.	Set tunnel configuration.	(xpShell):xps:vxlan)**vxlan_set_tunnel_config** *devId tnlIntfId pktCmd*
8.	Verify tunnel configuration.	(xpShell):xps:vxlan)**vxlan_get_tunnel_config** *devId tnlIntfId*
9.	Verify remote tunnel IP address.	(xpShell):xps:vxlan)**vxlan_get_tunnel_remote_ip** *devId tnlIntfId*
10.	Set tunnel next hop data.	(xpShell):xps:vxlan)**vxlan_set_tunnel_next_hop_data** *devId tnlIntfId nhId* **NOTE:** The following commands must be executed before setting the tunnel next hop data: (xpShell):xps:vlan)**vlan_create** *devId vlanId* (xpShell):xps:vlan)**vlan_add_interface** *devId vlanId intfId tagType* (xpShell):xps:vlan)**vlan_add_endpoint** *devId vlanId intfId tagType data* (xpShell):xps:l3)**l3_create_route_next_hop** *nhEcmpSize* (xpShell):xps:l3)**l3_set_route_next_hop** *devId nhId pktCmd serviceInstId vpnLabel propTTL l3InterfaceId macDa egressIntfId reasonCode* (xpShell):xps:l3)**l3_get_route_next_hop** *devId nhId* (xpShell):xps:interface)**interface_create_router_over_vlan** *vlanId* (xpShell):xps:l3)**l3_set_intf_egress_router_mac_lsb** *devId l3IntfId macSa*
11.	Update next hop data.	(xpShell):xps:vxlan)**vxlan_update_tunnel_next_hop_data** *devId tnlIntfId*

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

The following xpShell commands are used to remove the tunnel entry and tunnel interface.

Table 3–48 Remove/Destroy Tunnel Commands

Action	Command
Remove tunnel entry.	(xpShell):xps:vxlan)**vxlan_remove_tunnel_entry** *devId tnlIntfId*
Destroy tunnel interface.	(xpShell):xps:vxlan)**vxlan_destroy_tunnel_interface** *tnlIntfId*
Remove local VTEP.	(xpShell):xps:vxlan)**vxlan_remove_local_vtep** *devId localIp*

Example The following example configures the VXLAN tunnel.

Table 3–49 Example: VXLAN Tunnel Programming

#	Action	Command
1.	Create VxLAN tunnel interface.	`(xpShell):xps:vxlan)`**`vxlan_create_tunnel_interface`**` 64.101.203.156 253.138.194.52` This creates `tnlInitfId` = 47104.
2.	Add tunnel entry.	`(xpShell):xps:vxlan)`**`vxlan_add_tunnel_entry`**` 0 47104`
3.	Add local VTEP.	`(xpShell):xps:vxlan)`**`vxlan_add_local_vtep`**` 0 192.168.2.1`
4.	Add VNI.	`(xpShell):xps:vxlan)`**`vxlan_add_vni`**` 0 11 102`
5.	Set UDP port.	`(xpShell):xps:vxlan)`**`vxlan_set_udp_port`**` 0 500`

6.	Set tunnel configuration.	`(xpShell):xps:vxlan)`**`vxlan_set_tunnel_config`**` 0 47104 1 0 0`
		NOTE: The following commands must be executed before setting the tunnel configuration:
		`(xpShell):xps:vxlan)`**`vxlan_create_tunnel_interface`**
		`(xpShell):xps:vxlan)`**`vxlan_add_tunnel_entry 0 47104 98.65.175.130 049.200.222.12`**

7.	Verify tunnel configuration.	`(xpShell):xps:vxlan)`**`vxlan_get_tunnel_config 0 47105`**
		Input Arguments are:
		`devId=0`
		`tnlIntfId=47105`
		`pktCmd = 1`
		`paclEn = 0`
		`paclId = 0`
		`Command Success`
		The remote tunnel IP address is verified using following command.
		`(xpShell):xps:vxlan)`**`vxlan_get_tunnel_remote_ip 0 47104`**
		Input Arguments are:
		`devId=0`
		`tnlIntfId=47104`
		`rmtEpIpAddr = 64.210.182.247`
		`Command Success`

8.	Set tunnel next hop data.	`(xpShell):xps:vxlan)`**`vxlan_set_tunnel_next_hop_data 0 47104 0`**
		NOTE: The following commands must be executed before setting the tunnel next hop data:
		`(xpShell):xps:vlan)`**`vlan_create 1 107`**
		`(xpShell):xps:vlan)`**`vlan_add_interface 0 107 13 1`**
		`(xpShell):xps:vlan)`**`vlan_add_endpoint 0 107 47104 XP_L2_ENCAP_VXLAN 20`**
		`(xpShell):xps:l3)`**`l3_create_route_next_hop 1`**
		`(xpShell):xps:l3)`**`l3_set_route_next_hop 0 0 1 20 1 107 04:10:20:20:30:30 13`**
		`(xpShell):xps:l3)`**`l3_get_route_next_hop 0 0`**
		`(xpShell):xps:interface)`**`interface_create_router_over_vlan 107`**
		`(xpShell):xps:l3)`**`l3_set_intf_egress_router_mac_lsb 0 65643 64`**

9.	Update next hop data.	`(xpShell):xps:vxlan)`**`vxlan_update_tunnel_next_hop_data 0 47104`**

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

3.11.3 IPGRE Tunnel

The following commands are executed in sequence to configure IPGRE tunnel:

Table 3–50 IPGRE Tunnel Programming

#	Action	Command
1.	Create GRE tunnel interface.	(xpShell):xps:ipgre)**ip_gre_create_tunnel_interface** *lclEpIpAddr rmtEpIpAddr*
2.	Add GRE tunnel entry.	(xpShell):xps:ipgre)**ip_gre_add_tunnel_entry** *devId tnlIntfId*
3.	Set tunnel configuration.	(xpShell):xps:ipgre)**ip_gre_set_tunnel_config** *devId tnlIntfId pktCmd baclEn baclId raclEn raclId*
4.	Verify tunnel configuration.	(xpShell):xps:ipgre)**ip_gre_get_tunnel_config** *devId tnlIntfId*
5.	Set next hop data.	(xpShell):xps:ipgre)**ip_gre_set_tunnel_next_hop_data** *devId tnlIntfId nhId* **NOTE:** The following series of commands must have been executed before setting next hop data: : (xpShell):xps:vlan)**vlan_create** *devId vlanId* (xpShell):xps:vlan)**vlan_add_interface** *devId vlanId intfId tagType* (xpShell):xps:l3)**l3_create_route_next_hop** *nhEcmpSize* (xpShell):xps:l3)**l3_set_route_next_hop** *devId nhId pktCmd serviceInstId vpnLabel propTTL l3InterfaceId macDa egressIntfId reasonCode* (xpShell):xps:l3)**l3_get_route_next_hop** *devId nhId* (xpShell):xps:interface)**interface_create_router_over_vlan** *vlanId* (xpShell):xps:l3)**l3_set_intf_egress_router_mac_lsb** *devId l3IntfId macSa*
6.	Update next hop data.	(xpShell):xps:ipgre)**ip_gre_update_tunnel_next_hop_data** *devId tnlIntfId*
7.	Verify tunnel remote IP address.	(xpShell):xps:ipgre)**ip_gre_get_tunnel_remote_ip** *devId tnlIntfId*

The following commands are used to remove a tunnel entry and destroy the tunnel interface.

Table 3–51 Remove/Destroy IPGRE Tunnel Commands

Action	Command
Remove tunnel interface.	(xpShell):xps:ipgre)**ip_gre_remove_tunnel_entry** *devId tnlIntfId*
Destroy tunnel interface.	(xpShell):xps:ipgre)**ip_gre_destroy_tunnel_interface** *tnlIntfId*

Example The following example creates IPGRE tunnel.

Table 3–52 Example: IPGRE Tunnel Programming

#	Action	Command
1.	Create GRE tunnel interface.	(xpShell):xps:ipgre)**ip_gre_create_tunnel_interface 11.70.168.50 11.222.200.100**
2.	Add GRE tunnel entry.	(xpShell):xps:ipgre)**ip_gre_add_tunnel_entry 0 47104**
3.	Set tunnel configuration.	(xpShell):xps:ipgre)**ip_gre_set_tunnel_config 0 47104 XP_PKTCMD_FWD 1 10 1 20**

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Table 3–52 Example: IPGRE Tunnel Programming

#	Action	Command

4.	Verify tunnel configuration.	(xpShell):xps:ipgre)**ip_gre_get_tunnel_config 0 47104**
		Input Arguments are:
		devId=0
		tnlIntfId=47104
		pktCmd = 1
		baclEn = 1
		baclId = 10
		raclEn = 1
		raclId = 20
		Command Success

5.	Set next hop data.	(xpShell):xps:ipgre)**ip_gre_set_tunnel_next_hop_data 0 47104 0** **NOTE:** The following sequence of commands must have been executed before setting next hop data: (xpShell):xps:vlan)**vlan_create 0 107** (xpShell):xps:vlan)**vlan_add_interface 0 107 13 1** (xpShell):xps:l3)**l3_create_route_next_hop 1** (xpShell):xps:l3)**l3_set_route_next_hop 0 0 1 20 1 107 04:10:20:20:30:30 13** (xpShell):xps:l3)**l3_get_route_next_hop 0 0** (xpShell):xps:interface)**interface_create_router_over_vlan 107** (xpShell):xps:l3)**l3_set_intf_egress_router_mac_lsb 0 65643 64**
6.	Update next hop data.	(xpShell):xps:ipgre)**ip_gre_update_tunnel_next_hop_data 0 47104**

7.	Verify tunnel remote IP address.	(xpShell):xps:ipgre)**ip_gre_get_tunnel_remote_ip 0 47104**
		Input Arguments are:
		devId=0
		tnlIntfId=47104
		rmtEpIpAddr = 64.210.182.247
		Command Success

3.11.4 IPinIP Tunnel

The following commands are executed to configure IPinIP tunnel.

Table 3–53 IPinIP Tunnel Programming

#	Action	Command
1.	Create IPinIP tunnel interface.	(xpShell):xps:ipinip)**ipin_ip_create_tunnel_interface** *lclEpIpAddr rmtEpIpAddr*
2.	Add tunnel entry.	(xpShell):xps:ipinip)**ipin_ip_add_tunnel_entry** *devId tnlIntfId*
3.	Set tunnel configuration.	(xpShell):xps:ipinip)**ipin_ip_set_tunnel_config** *devId tnlIntfId baclEn baclId raclEn raclId*
4.	Verify the tunnel configuration.	(xpShell):xps:ipinip)**ipin_ip_get_tunnel_config** *devId tnlIntfId*
5.	Verify the remote IP address.	(xpShell):xps:ipinip)**ipin_ip_get_tunnel_remote_ip** *devId tnlIntfId*

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Table 3–53 IPinIP Tunnel Programming

#	Action	Command
6.	Set tunnel next hop data.	(xpShell):xps:ipinip)**ipin_ip_set_tunnel_next_hop_data** *devId tnlIntfId nhId* **NOTE:** The following sequence of commands must have been executed prior to setting the tunnel next hop data: (xpShell):xps:vlan)**vlan_create** *devId vlanId* (xpShell):xps:interface)**interface_create** *type* (xpShell):xps:port)**port_get_port_intf_id** *devId portNum* (xpShell):xps:vlan)**vlan_add_interface** *devId vlanId intfId tagType* (xpShell):xps:l3)**l3_create_tunnel_intf** (xpShell):xps:l3)**l3_init_tunnel_intf** *devId l3IntfId* (xpShell):xps:l3)**l3_create_vlan_intf** *vlanId* (xpShell):xps:l3)**l3_create_route_next_hop** *nhEcmpSize* (xpShell):xps:l3)**l3_set_route_next_hop** *devId nhId pktCmd serviceInstId vpnLabel propTTL l3InterfaceId macDa egressIntfId reasonCode*
7.	Update next hop data.	(xpShell):xps:ipinip)**ipin_ip_update_tunnel_next_hop_data** *devId tnlIntfId*

The following commands are used to remove and destroy the tunnel.

Table 3–54 Remove/Destroy IPinIP Tunnel Commands

Action	Command
Remove tunnel.	(xpShell):xps:ipinip)**ipin_ip_remove_tunnel_entry** *devId tnlIntfId*
Destroy tunnel.	(xpShell):xps:ipinip)**ipin_ip_destroy_tunnel_interface** *tnlIntfId*

Example

The following example creates an IPinIP tunnel.

Table 3–55 Example: IPinIP Tunnel Programming

#	Action	Command
1.	Create IPinIP tunnel interface	(xpShell):xps:ipinip)**ipin_ip_create_tunnel_interface 199.56.199.195 199.56.199.195**
2.	Add tunnel entry.	(xpShell):xps:ipinip)**ipin_ip_add_tunnel_entry 0 47104**
3.	Set tunnel configuration.	(xpShell):xps:ipinip)**ipin_ip_set_tunnel_config 0 47104 2 0 2 0 2**

4.	Verify the tunnel configuration.	(xpShell):xps:ipinip)**ipin_ip_get_tunnel_config 0 47104**
		Input arguments are:
		devId=0
		tnlIntfId=47104
		pktCmd = 2
		baclEn = 0
		baclId = 2
		raclEn = 0
		raclId = 2
		Command Success

5.	Verify the remote IP address.	(xpShell):xps:ipinip)**ipin_ip_get_tunnel_remote_ip 0 47104**
		Input arguments are:
		devId=0
		tnlIntfId=47104
		rmtEpIpAddr = 64.210.182.247
		Command Success

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Table 3–55 Example: IPinIP Tunnel Programming

#	Action	Command
6.	Set tunnel next hop data.	(xpShell):xps:ipinip)**ipin_ip_set_tunnel_next_hop_data 0 47104** **NOTE:** The following sequence of commands must have been executed prior setting the tunnel next hop data: (xpShell):xps:vlan)**vlan_create 0 100** (xpShell):xps:interface)**interface_create 8** (xpShell):xps:port)**port_get_port_intf_id 0 1** (xpShell):xps:vlan)**vlan_add_interface 1 100 47105 0** (xpShell):xps:l3)**l3_create_tunnel_intf** (xpShell):xps:l3)**l3_init_tunnel_intf 0 69632** (xpShell):xps:l3)**l3_create_vlan_intf 100** (xpShell):xps:l3)**l3_create_route_next_hop 1** (xpShell):xps:l3)**l3_set_route_next_hop 0 0 1 0 1 65636 E3:DB:DF:DB:0C:00 47105**
7.	Update next hop data.	(xpShell):xps:ipinip)**ipin_ip_update_tunnel_next_hop_data 0 47104**
8.	Remove the tunnel.	(xpShell):xps:ipinip)**ipin_ip_remove_tunnel_entry 0 47104**
9.	Destroy the tunnel.	(xpShell):xps:ipinip)**ipin_ip_destroy_tunnel_interface 47104**

3.11.5 MPLS Tunnel

Refer to section 3.4 MPLS.

3.11.6 MPLSoGRE

The following commands are executed in sequence to configure a MPLS over GRE tunnel.

Table 3–56 MPLSoGRE Tunnel Programming

#	Action	Command
1.	Create VPN GRE loose- or strict-mode tunnel interface.	(xpShell):xps:vpngre)**vpn_gre_create_strict_mode_ip_tunnel_interface** *lclEpIpAddr rmtEpIpAddr* or (xpShell):xps:vpngre)**vpn_gre_create_loose_mode_ip_tunnel_interface** *lclEpIpAddr rmtEpIpAddr*
2.	Add tunnel entry.	(xpShell):xps:vpngre)**vpn_gre_add_tunnel_entry** *devId tnlIntfId*
3.	Set tunnel configuration.	(xpShell):xps:vpngre)**vpn_gre_set_tunnel_config** *devId tnlIntfId baclEn baclId raclEn raclId*
4.	Verify tunnel configuration.	(xpShell):xps:vpngre)**vpn_gre_get_tunnel_config** *devId tnlIntfId baclEn ba'clId raclEn raclId*

Cavium Confidential For Chen Peng Teraspek 01/24/2018

Table 3–56 MPLSoGRE Tunnel Programming

#	Action	Command
5.	Set next hop data.	(xpShell):xps:vpngre)**vpn_gre_set_tunnel_next_hop_data** *devId tnlIntfId nhId* **NOTE:** The following series of commands must have been executed before setting next hop data: : (xpShell):xps:vlan)**vlan_create** *devId vlanId* (xpShell):xps:vlan)**vlan_add_interface** *devId vlanId intfId tagType* (xpShell):xps:vlan)**vlan_set_config** *devId vlanId stpId countMode enableMirror mirrorAnalyzerId saMissCmd bcCmd unknownUcCmd arpBcCmd ipv4McbridgeMode ipv6McbridgeMode unRegMcCmd* (xpShell):xps:l3)**l3_create_vlan_intf** *vlanId* (xpShell):xps:l3)**l3_set_intf_vrf** *devId l3IntfId vrfId* (xpShell):xps:l3)**l3_set_intf_ipv4_uc_routing_en** *devId l3IntfId enable* *(xpShell):xps:l3)***l3_set_route_next_hop** *devId nhId pktCmd serviceInstId vpnLabel propTTL l3InterfaceId macDa egressIntfId reasonCode* (xpShell):xps:l3)**l3_create_tunnel_intf** (xpShell):xps:l3)**l3_init_tunnel_int** *devId l3IntfId* (xpShell):xps:l3)**l3_set_intf_ipv4_uc_routing_en** *devId l3IntfId enable* (xpShell):xps:l3)**l3_set_intf_mpls_routing_en** *devId l3IntfId enable* (xpShell):xps:l3)**l3_set_intf_vrf** *devId l3IntfId vrfId* (xpShell):xps:l3)**l3_bind_tunnel_intf** *devId tnnlIntfId l3IntfId* (xpShell):xps:l3)**l3_add_ip_route_entry** *devId vrfId type ipv4Addr ipv6Addr ipMaskLen nhEcmpSize nhId* (xpShell):xps:l3)**l3_add_ingress_router_mac** *devId mac* (xpShell):xps:l3)**l3_set_egress_router_mac_m_sbs** *devId macHi*
6.	Update next hop data.	(xpShell):xps:vpngre)**vpn_gre_update_tunnel_next_hop_data** *devId tnlIntfId*
7.	Verify tunnel remote IP address.	(xpShell):xps:vpngre)**vpn_gre_get_tunnel_remote_ip** *devId tnlIntfId*

The following commands are used to remove a tunnel entry and destroy the tunnel interface.

Table 3–57 Remove/Destroy MPLSoGRE Tunnel Commands

Action	Command
Remove tunnel interface.	(xpShell):xps:vpngre)**vpn_gre_remove_tunnel_entry** *devId tnlIntfId*
Destroy tunnel interface.	(xpShell):xps:vpngre)**vpn_gre_destroy_tunnel_interface** *tnlIntfId*

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018

Example

The following example creates a MPLSoGRE tunnel.

Table 3–58 Example: MPLSoGRE Tunnel Programming

#	Action	Command
1.	Create VPN GRE strict-mode tunnel interface	(xpShell):xps:vpngre)**vpn_gre_create_strict_mode_tunnel_interface** 50.168.70.14 100.200.222.14 22136 **Gives tunnel id**: tnlIntf0
2.	Add tunnel entry.	(xpShell):xps:vpngre)**vpn_gre_add_tunnel_entry** 0 tnlIntf0
3.	Set tunnel configuration.	(xpShell):xps:vpngre)**vpn_gre_set_tunnel_config** 0 tnlIntf0 0 0 0 0
4.	Verify tunnel configuration.	(xpShell):xps:vpngre)**vpn_gre_get_tunnel_config 0** tnlIntf0

Table 3–58 Example: MPLSoGRE Tunnel Programming

#	Action	Command
5.	Set next hop data.	(xpShell):xps:vpngre) **vpn_gre_set_tunnel_next_hop_data** 0 tnlIntf0 0 **NOTE:** The following sequence of commands must have been executed before setting next hop data: (xpShell):xps:l3) l3_set_route_next_hop 0 0 1 0 0 1 66036 02:04:05:05:11:11 <egressPort[0]> 0 l3_set_route_next_hop 0 1 1 0 22136 1 66036 02:04:05:05:11:11 $tnlIntf0 0 l3_create_tunnel_intf Created tunnel interface: l3tnlIntf0 l3_init_tunnel_intf 0 69632 l3_set_intf_ipv4_uc_routing_en 0 l3tnlIntf0 1 l3_set_intf_mpls_routing_en 0 l3tnlIntf0 1 l3_set_intf_vrf 0 l3tnlIntf0 4 l3_bind_tunnel_intf 0 $tnlIntf0 69632 l3_add_ip_route_entry 0 4 0 172.28.88.0 00:00:00:00:00:00:00:00:00:00:00:00:00:00 24 1 1 l3_add_ingress_router_mac 0 00:00:de:ad:be:ef l3_set_egress_router_mac_m_sbs 0 aa:bb:cc:dd:dd
6.	Update next hop data.	(xpShell):xps:vpngre)**vpn_gre_update_tunnel_next_hop_data 0 47104**
7.	Verify tunnel remote IP address.	(xpShell):xps:vpngre)**vpn_gre_get_tunnel_remote_ip 0** l3tnlIntf0

3.12 Display Tables

The xpShell is used to display tables; that is, FBD, VIF, BD and MDT tables. The list of tables is seen from xpShell help command as follows. The name of display commands are self explanatory.

```
(xpShell) display_tables
(xpShell):displayTables)?

(xpShell):displayTables)?

Available commands (type help <topic>):
=======================================
display_acm_bank        display_ipv4_pim_bidir_rpf   display_port_config
display_acm_result      display_ipv4_route           display_port_counter
display_aqm_pfl         display_ipv4_route_mc        display_port_mapping_cfg
display_aqm_q_pfl       display_ipv6_bridge_mc       display_port_vlan
display_bd              display_ipv6_host            display_q_counter
display_control_mac     display_ipv6_pim_bidir_rpf   display_q_mapping
display_dwrr            display_ipv6_route           display_qos_map
display_egress_bd       display_ipv6_route_mc        display_reason_code
display_egress_filter   display_mdt                  display_shapers
display_eq_cfg          display_mit                  display_tm_h1
display_fast_shapers    display_mpls_label           display_tm_h2
display_fdb             display_mpls_tunnel          display_tm_pipe
display_h1_counter      display_nat_comp_ipv6        display_tm_port
display_h2_counter      display_nat_ipv4             display_trunk_resolution
display_iit             display_nh                   display_tunnel_id
display_insertion       display_open_flow_entries    display_tunnel_ivif
display_ipv4_bridge_mc  display_pfc_counter          display_tunnel_local_vtep
display_ipv4_host       display_pkt_limit_threshold  display_vif
```

Cavium Confidential For Chen Peng Teraspek 1/24/2018

```
Utility commands
================
back  clear  help  ls   pause  py   save   shortcuts
cd    eof    load  nop  pwd    run  shell
```

3.13 xpShell Miscellaneous Utilities

The following xpShell utilities are available as needed.

- The xpShell `xps` commands can use the same format as the API names. Both formats—i.e., camelCase and under_score—are accepted.

  For example, instead of `vlan_create`, the `vlanCreate` portion of the `xpsVlanCreate` API can be called.
- The global `home` command returns you to the xpShell root directory from any location.
- Executing the `back` command from the `home` directory does not exit xpShell.
- The `load_config` command is applicable only from the `home` directory. After loading the configuration completes, it leaves the user at the xpShell `home` directory
- The `pwd` command displays the current working directory of xpShell just as it does in Linux/UNIX.
- The Linux commands `ls`, `cd`, and `cd` *`dir`* can be used to navigate between xpShell directories.
- The `save_config` command starts saving the configuration (subsequent commands) in the given file and stops recording only after the `save_config stop` command is executed. By default, it leaves the file in the current execution directory. A file saved using `save_config` can be used again by loading it with the `load_config` command.
- The global `exec` command can be used to run any SWIG-accessible API from xpShell.
- The new `scripts` directory inside the CLI directory can be used for adding application test scripts to load using `load_config`.

Cavium Confidential For
Chen Peng
Teraspek
01/24/2018