Barefoot Wedge 100B Switch Platform User Guide

This document explains how to connect to and manage the Wedge 100BF-65X and Wedge 100BF-32X reference switch systems.

ATTENTION:

The information contained in this document is preliminary and subject to change without notice

Document Revision History

Revision	Date	Description
10k-UG2-018EA	03/26/2018	Added information on CPU ports and a new section, “Generating test traffic.” Explained log file locations. Add details on loading switch config.
10k-UG2-017EA	11/17/2017	Update caveats list; issue 9.4 is fixed; 9.1 and 9.2 less likely to occur.
10k-UG2-016EA	09/08/2017	Minor updates.
10k-UG2-015EA	08/28/2017	Fix incorrect port auto-negotiation (an-set) doc. To disable, correct setting is 2 not 0.
10k-UG2-014EA	08/16/2017	Add packet flow troubleshooting tips and snapshotting.
10k-UG2-013EA	07/27/2017	Add OpenBMC bundle name.
10k-UG2-012EA	07/17/2017	Accommodate ONL in mentions of OS and admin credentials. Fix bad cross references.
10k-UG2-011EA	07/07/2017	Add SDE prerequisites including HugePages.
10k-UG2-010EA	06/23/2017	Add missing SDE dependencies installation instructions for ONL-based systems. Clarify NOS options.
10k-UG2-008EA	06/14/2017	Add BMC firmware update steps. Add ONIE and ONL installation steps.
10k-UG2-007EA	05/24/2017	Minor edits.
10k-UG2-006EA	05/17/2017	Wedge 100B with SDE 4.0.x. Add LED control data diagram.
10k-UG2-005EA	04/28/2017	Add SYS-4 issue workaround. Add troubleshooting tips.
10k-UG2-004EA	04/19/2017	Add autonegotiation configuration notes. Add traffic_mgr commands.
10k-UG2-003EA	04/06/2017	Wedge 100B with SDE 3.3.0, new Tofino kernel mode driver bf_kdrv.ko replaces bf_uio.ko driver.
10k-UG2-002EA	03/23/2017	Wedge 100B with SDE 3.2.2
10k-UG2-001EA	03/17/2017	Early access release

Table of Contents

Table of Contents 3

1 Platform Overview 6

1.1 Main CPU Module 6

1.2 BMC CPU Module 7

1.3 Tofino Switch ASIC 7

1.3.1 PCIe x4 Gen 3 7

1.3.2 Tofino Ports 7

1.4 AC/DC Power Supply Units 8

1.5 Fans 8

1.6 LED status lamps 8

1.6.1 Using LED Port Channel Status Lamps 8

1.6.2 Key to Port Status LED Colors 9

1.6.3 LED Control: Overview 9

1.6.4 LED Control: Software Implementation 9

1.6.5 Device Hardware Support 10

1.6.6 Detailed Device Register Information 10

2 Management Access 13

2.1 Accessing the BMC CPU and main CPUs through the serial console port 13

2.1.1 Connecting to main CPU via BMC CPU: sol.sh 13

2.2 Appliance default login credentials 14

2.3 Management Ethernet Port 14

2.3.1 Configuring the main CPU's IP address 14

2.3.2 Logging into the main CPU over SSH 15

2.4 USB Port 15

3 Using the Capilano™ SDE on the Wedge 100B 16

3.1 Getting the SDE 16

3.2 Make sure SDE dependencies are present 16

3.2.1 Installing SDE Dependencies 16

3.3 Building the SDE 17

3.4 Loading the Tofino kernel mode driver (`bf_kdrv.ko`) 18

3.4.1 Minimizing rebuilds of bf_kdrv.ko 18

3.5 Set HugePages memory settings 19

3.6 Start your P4-defined switch on the Tofino hardware 19

3.6.1 Troubleshooting the SDE and run_switchd 20

3.6.2 How a Switch Configuration is Loaded on the Wedge 100B 21

4 Managing Tofino Ports 22

4.1 Tofino Internal Ports 22

4.2 Tofino Switch Ports 22

4.3 Wedge 100B Port Naming 23

4.4 Summary of bf-sde CLI port management commands 24

4.5 Adding and enabling ports 25

4.5.1 QSFP lane restrictions on port creation 25
4.5.2 Adding a port 25
4.5.3 Enabling a port 26
4.6 Managing port features 26
4.6.1 Port autonegotiation 26
4.7 Disabling and deleting ports 27
4.7.1 Disabling a port 27
4.7.2 Deleting a port 27
4.8 Wildcard port operations 27
4.9 Getting information about ports 28
4.9.1 show command in bf_pltfm.pm 28
4.9.2 Show all ports 29
4.9.3 Get connector information using the dump-info command 29
4.10 Managing ports from a PTF script 30
5 Exercising PTF scripts to configure Tofino 31
5.1 Example 1: Setting up P4 table rules for basic_switching.p4 31
5.1.1 Start bf_switchd with basic_switching.p4 31
5.1.2 Run the PTF script to set up the P4 table rules 31
5.2 Example 2: Setting up IPv4 Routing Table for switch.p4 using switchAPI 31
5.2.1 Start bf_switchd with switch.p4 and switchAPI library 32
5.2.2 Run the PTF script to set up the IPv4 Routing Table 32
5.3 Example 3: Setting up IPv4 Routing Table for switch.p4 using P4-PD-API 32
5.3.1 Start bf_switchd with switch.p4 and PD-API library 32
5.3.2 Run the PTF script to set up the IPv4 Routing Table 32
5.3.3 Dumping the IPv4 Routing Table 32
6 Managing the Wedge 100B Platform 33
6.1 Overview of the bf-sde CLI 33
6.1.1 Launching the bf-sde CLI 33
6.1.2 Navigating the bf-sde CLI 33
6.1.3 Top-level CLI commands in bf-sde 33
6.2 Board Port Configuration 33
6.3 Chassis Management 34
6.3.1 Get MAC addresses 35
6.3.2 Dump EEPROM Contents 35
6.3.3 Tofino Temperature Monitoring 35
6.3.4 Power Supply Monitoring 35
6.3.5 Fan Monitoring 35
6.4 QSFP Management 36
7 Firmware and Software Updates 37
7.1 Main CPU Boot Loader Updates 37
7.1.1 Installing BF-Wedge-ONIE 37
7.2 Main CPU Operating System Updates 40
7.2.1 Installing ONL 40
7.3 BMC CPU Operating System Updates 41

7.3.1 BMC Image Architecture 41
7.3.2 Procedure: BMC Update 42
7.3.3 Checking BMC image version 43
8 Tofino Testing and Troubleshooting 44
8.1 Generating test traffic 44
8.1.1 Generating packets with the Diags test dataplane 44
8.1.2 Injecting Packets from the CPU 45
8.1.3 Generating Packets in Tofino with pktgen 45
8.2 Log file locations 45
8.3 Dumping match-action table entries 45
8.3.1 List all match-action tables 45
8.3.2 Show information about a match-action table 46
8.3.3 Show information about a match-action table entry 47
8.4 Troubleshooting Packet Drops 48
8.5 Checking memory register values 48
8.6 Packet Snapshots 48
8.6.1 Snapshot Commands 49
8.6.2 Creating a snapshot trigger 49
8.6.3 Viewing a snapshot 50
8.7 Port and Pipeline Counters 52
8.7.1 Dumping Port (MAC) counters 52
8.7.2 Pipeline Traffic Counters 52
8.8 Traffic-Manager (TM) and Traffic Counters 53
8.8.1 Traffic Manager cfg_table Commands 53
8.8.2 Traffic Manager cfg Commands 56
8.8.3 Traffic Manager usage Commands 56
8.8.4 Traffic Manager watermark Commands 57
8.8.5 Traffic Manager dropstatus Commands 58
8.8.6 Traffic Manager dl_dropstatus Commands 60
8.8.7 Traffic Manager pfcstatus Commands 62
8.8.8 Traffic Manager Counter Commands 62
8.8.9 Traffic Manager clr_counter Commands 63
8.9 Interrupt Monitoring 64
9 Known Caveats and Workarounds 65
9.1 SYS-1: Main CPU is not powered up by the time OpenBMC is up 65
9.2 SYS-2: Main CPU stuck at BIOS screen 65
9.3 SYS-3: Switchd fails with BF_PLTFM CURL ERROR 65
10 Copyright and Trademarks 67
11 Contacting Support 68
12 Index 69

1 Platform Overview

The Barefoot Wedge 100BF-65X provides 65 QSFP28 network ports for a total of 6.5 Tbps throughput. Each QSFP28 port supports 100GE (4x25G), 50GE (2x25G), 25GE, 40GE (4x10G), or 4x10GE. The 4x10GE option is achieved via a QSFP-to-SFP+ breakout cable.

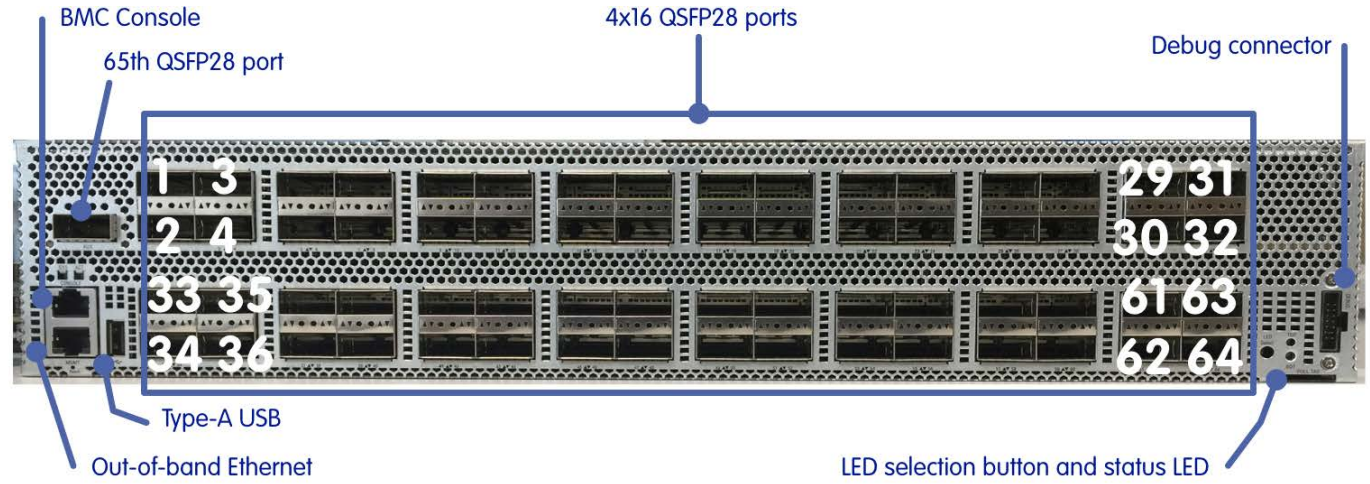


Figure 1: Barefoot Wedge 100BF-65X Front Panel View

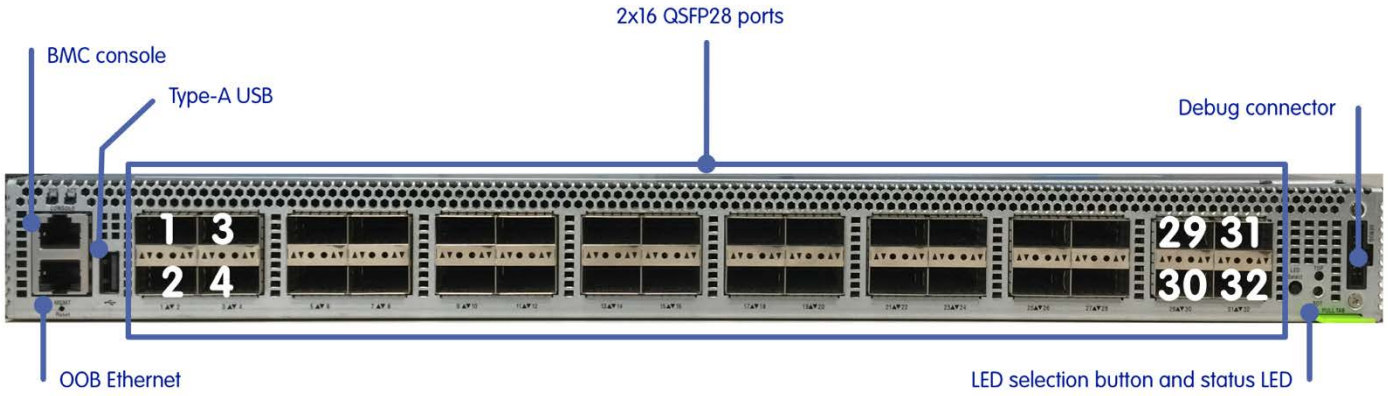


Figure 2: Barefoot Wedge 100BF-32X Front Panel View

1.1 Main CPU Module

The main CPU of the Barefoot Wedge 100B is a COM Express Type 6 module. This is an industry-standard, embedded CPU board running an Intel x86 Broadwell-DE family CPU with the following configuration:

- Broadwell-DE, Pentium D1517, 4 cores, 1.6 GHz, 6 MB cache, 25W
- 4, 8, or 16 GB of DDR4 DRAM memory utilizing up to three SODIMM sockets on the CPU board
- 128 GB M.2 SSD drive accessible over SATA

This is the main CPU of the system where the Barefoot Networks Capilano™ Software Development Environment (SDE) runs, either as a standalone software or integrated with a network operating system (NOS).

On the main CPU, the following software runs on power-up:

- AMI BIOS

- The operating system, which will be either Ubuntu 16.04 Linux or ONL.

More information:

- *To connect to the main CPU*, see Management Access on page 13.
- *To update the main CPU boot loader and operating system*, see Firmware and Software Updates on page 37.

1.2 BMC CPU Module

General control of the system is provided by the baseboard management controller CPU (BMC). For this, the Wedge 100B uses an Aspeed Technologies AST1250 Integrated Remote Management Processor. This CPU runs the OpenBMC software that performs the following platform management functions:

- reset orchestration
- PSU management
- voltage monitoring
- temperature monitoring and fan control

More information:

- *To connect to the BMC CPU*, see Management Access on page 13.
- *To update the BMC CPU operating system*, see BMC CPU Operating System Updates on page 41.

1.3 Tofino Switch ASIC

1.3.1 PCIe x4 Gen 3

The Tofino ASIC is connected to the CPU's PCIe root complex over PCIe 4x operating in Gen3 mode. On bootup, the BIOS and the OS enumerate the PCIe bus and set up the Base Address Registers for Tofino. You can confirm this setup from the Ubuntu shell using the `lspci` command. This should return a value showing Tofino to be enumerated at pcie_bus '5'.

> **Important!** Do not omit the trailing colon when you run the `lspci` command.

```
admin12@bfn-switch:~$ sudo lspci -vv -d 1d1c:
05:00.0 Unassigned class [ff00]: Device 1d1c:0001
      Subsystem: Device 1d1c:0001
      Physical Slot: 1
      Control: I/O- Mem- BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr-
      ...
```

1.3.2 Tofino Ports

1.3.2.1 64x100GE QSFP28 Ports

Tofino supports 64 QSFP28 ports (a total of sixteen 2x2 zQSFP cage connectors, with eight 2x2 zQSFP cage connectors on the upper board and eight on lower board). Each front panel port can be configured to support 100GE, 50GE, 40GE, 25GE, and 10GE speeds.

1.3.2.2 1x100GE QSFP28 "CPU" Port

- This is an additional 100GE network port which serves as the 65th 100GE network port
- This port is special in that it has the additional capability to operate at 1G speeds, unlike the other 64 ports.

1.4 AC/DC Power Supply Units

There are two redundant, 1100W AC/DC power supply units (PSUs) with internal 40 mm fan units. Each PSU has the following characteristics:

- input is 90V-264V AC
- output is 12V and 3.3V/5V standby (standby is not used in the Wedge 100B design)
- hot swappable
- load-sharing between the 2 PSUs

1.5 Fans

There are ten hot-swappable fan units in the system, five for the upper card and five for the lower card. Both upper and lower cards share the same fan board which consists of five fans.

The system supports both front-to-back and back-to-front airflow directions. In a given system, the fans and PSUs have the same airflow direction and are distinguished by the color-coded handles.

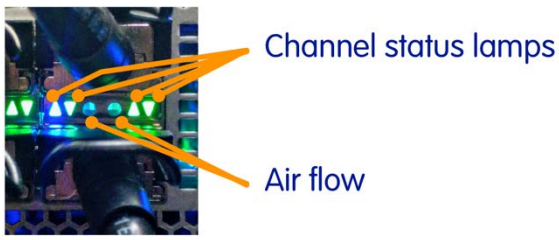
Other fan features include:

- Fans have status LEDs that show the operational status.
- Fans are individually hot swappable.
- Each fan speed is individually controllable by means of the platform software running on the BMC.

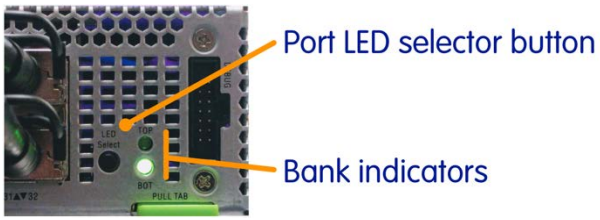
1.6 LED status lamps

1.6.1 Using LED Port Channel Status Lamps

The Wedge 100B port channel status lamps appear in a bank of four triangular lamps between a top/bottom (odd/even) pair of QSFP ports. The orientation of a triangular lamp does *not* indicate which QSFP port the lamp corresponds to. Based on the setting you've chosen using the port LED selector button, the four lamps indicate either the states of the MAC channels on the QSFP port above the lamps, or the states of the MAC channels on the QSFP port below the lamps.



The port LED selector button is located at the lower right of the front panel, labeled "LED Select," as shown in the illustration below. To the right of the button are two bank indicator lamps, TOP and BOT, indicating whether all port lamps are showing the port state for the port above them (TOP) or below them (BOT). Press the button to toggle between the two states.



For example, let's look at the panel's leftmost top/bottom pair of Wedge 100B QSFP ports and assume a configuration of 4x25G on each QSFP port (four MAC channels of 25G each). If you select the TOP bank, the four lamps show the state of the top QSFP port's four channels from left to right as:

- QSFP port 1 / MAC channel 1
- QSFP port 1 / MAC channel 2
- QSFP port 1 / MAC channel 3
- QSFP port 1 / MAC channel 4

1.6.2 Key to Port Status LED Colors

On the Wedge 100B default configuration, the color and state of an LED port channel status lamp indicates the status as shown here:

- Green: Port/channel is active (link is up)
- Blue: Port/channel is enabled but the link is not up
- Orange: Port/channel has been disabled by an administrator.
- LED off: Port/channel is not enabled or has not been added.

The Wedge 100B currently does not use the LED to display port speed.

1.6.3 LED Control: Overview

For switches with an on-chip microcontroller, it is possible for the switch to directly control the port speed indicator LED lamps. As the Tofino switch does not have a built-in microcontroller, the LED control ultimately comes from the Host CPU. The general procedure for this is as follows:

1. Host CPU polls Tofino for link status.

2. Host CPU writes the LED status info into Tofino via PCIe.

3. Host CPU triggers the **`i2c_ctrl`** register on Tofino to shift stateout bits to CPLD. (Alternatively, the host CPU can write to CPLD directly, but this is likely over I²C and therefore slow. Communicating through Tofino is faster for the Host CPU.)

The CPLD has an I²C interface to communicate with Tofino. To reduce number of IO operations needed on the CPLD, shift registers are used on the board. The CPLD will shift the data out to the shift registers.

1.6.4 LED Control: Software Implementation

> **Important**: Be aware of the role of the *port LED selector button*. If you are not seeing the expected LED state for a port or channel, press the port LED selector button to toggle the LEDs. For details, see the earlier section, "Using LED Port Channel Status Lamps."

LED control features are mostly implemented in the **`bf_platfm`** APIs, which you can find in **`$SDE/packages/bf-platforms-3.2.0.40/drivers/src`**. These functions are platform dependent. Users building their own switch platforms can take these as example code and modify them.

The host CPU periodically polls Tofino link status via **`link_poll_timer_cb()`** in **`bf_pltfm_pm_intf.c`**. To get the current operational state of each port, **`link_poll_timer_cb()`** calls **`bf_port_oper_state_get()`**.

The function to modify the LED state is **`bf_pltfm_port_led_by_cpld_set()`** (currently used by the SDE) or **`bf_pltfm_port_led_set()`** in **`bf_pltfm_mav_led.c`**. The released SDE currently uses the "alternative" way of

modifying the LED status by writing to the CPLD memory directly. To switch over to using Tofino, we basically will be calling **`bf_pltfm_port_led_set()`** in places where **`bf_pltfm_port_led_by_cpld_set()`** is called.

The method **`bf_pltfm_port_led_by_cpld_set()`** is called by the platform port management functions responsible for updating the LED status each time a port is enabled, disabled or deleted.

In the top portion of **`bf_pltfm_mav_led.c`**, you can see the definition we use for our LED. A given QSFP lane has a single LED to indicate its status, and to set the color, the system performs an OR operation on a set of four bits. The four bits are defined as:

- bit 0: green
- bit 1: blue
- bit 2: red
- bit 3: blink

1.6.5 Device Hardware Support

As explained in the document, "10k-Series Device Family Overview" (10k-DS1-008EA.pdf), there are two independent GPIO blocks on the device. Each GPIO block can support up to 6 I²C master interfaces. However, only one is needed to send the LED info. Each GPIO has a shared 2048 bits of memory organized into 64 entries of 32b registers. Each I²C interface can specify a start address (0 to 63) and number of bytes to write. If you connect GPIO_0/1 and GPIO_12/13 (that is, one from each GPIO block), you will have up to 4096b for LED control.

1.6.6 Detailed Device Register Information

This section provides information to help you navigate the device register map pertaining to GPIO controls.

Please inspect the register group, **`gpio_pair_regs`**. This group contains eleven registers for each GPIO pair. Please see the "Array Dimensions" and address fields. There are six **`gpio_pair_regs`** groups. The first group (address 0x1820000 to 0x1820028) controls the first GPIO pair of the group. The second group (0x182002C to 0x1820058) controls the second GPIO pair.

Below is a top-level view of the register groups on Tofino:

- **`Ethgpiobr`**: Bottom right of die view. This corresponds to bottom left 6 pairs of GPIO from package view. GPIO 12–23.
  - **`gpio_pair_regs`** X 6 copies
    - Controls 6 I²C pairs
    - First group controls GPIO 12–13
  - **`gpio_common_regs`**: only 1 copy
    - stateout register (32b x 64 entries = 2048 bits)
    - This is shared by GPIO12–23
- **`Ethgpiotl`**: Top left of die view. This corresponds to top right 6 pairs of GPIO from package view. GPIO 0–11
  - Same set up as above
  - First group controls GPIO 0–1

To understand the I²C frame format, please see the illustration below.

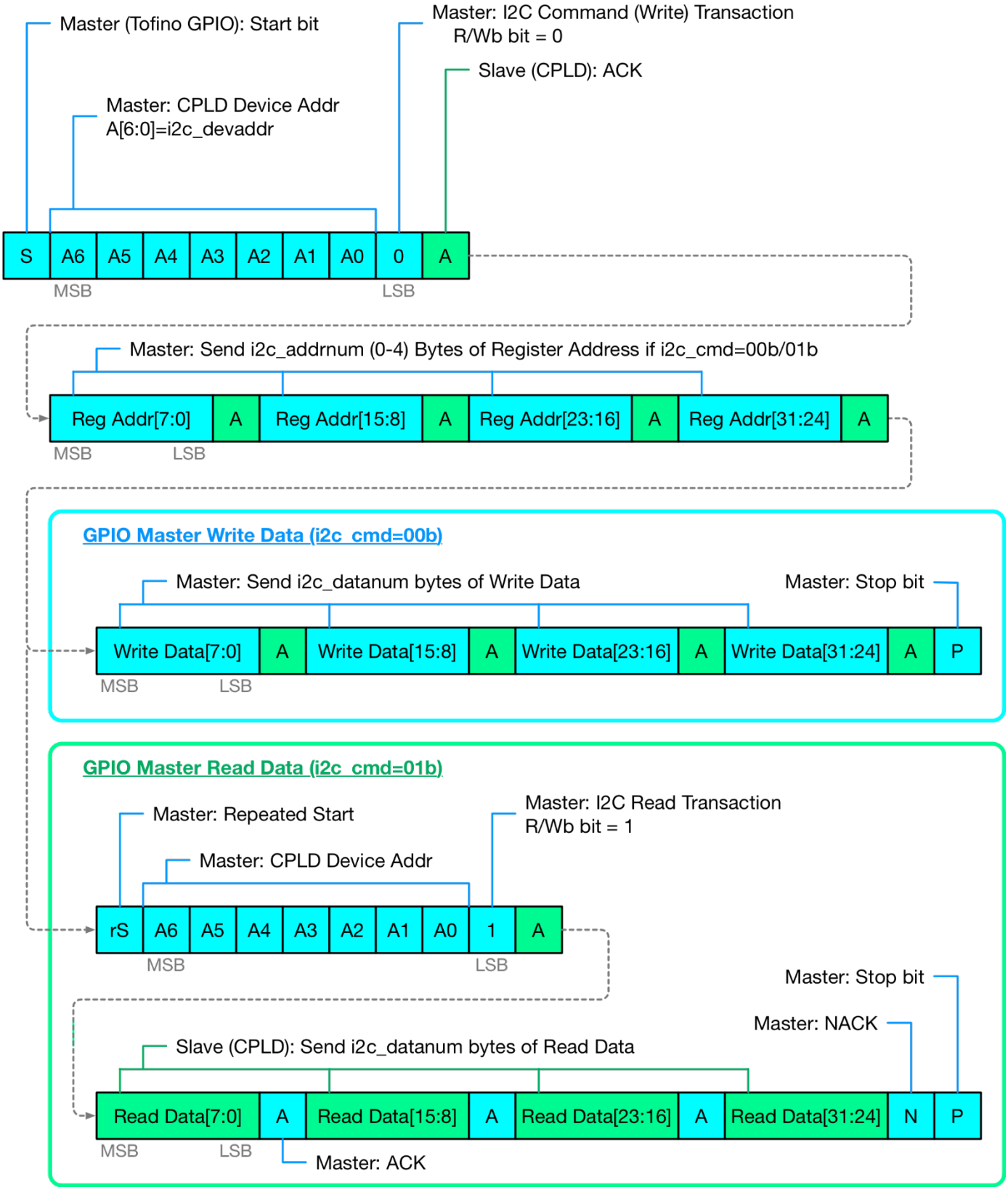


Figure 3: Slave I2C 32b Configuration Register Transaction

In an I²C transaction, you send out the following fields (register field name mapped below):

- I²C device address: **`i2c_ctrl.i2c_devaddr`**
- I²C read/write_b: **`i2c_ctrl.i2c_cmd`**. For LED data writes, this will be 00b
- I²C CPLD register address (labeled as "CPLD Device Addr" in Figure 3): **`i2c_addr.i2c_addr`**
- I²C write data: It can either be 1-4 bytes of **`i2c_wdata.i2c_wdata`** or it can be contents from the 2048b stateout register.
    - **`i2c_ctrl.type`** determines if the write data is from i2c_wdata or stateout
    - **`i2c_ctrl.i2c_stateaddr`**: where in stateout to start sending out data. This is a byte offset into the 2048b register array
    - **`i2c_ctrl.i2c_datanum`**: number of bytes to write out. All the data will be written out in a single I²C transaction. For example, if 16 bytes were to be written out, it would be sent as a burst of 16 bytes and not broken up into four 4-byte I2C transactions.
- **`i2c_ctrl.i2c_exec`**: Execute the I2C transaction

For further information, see the Barefoot knowledge base article, Slave I²C Usage.

2 Management Access

On initial startup of the Wedge 100B, connect to the BMC CPU through the CONSOLE port on the front panel. Once logged in to the BMC CPU, you can connect to the main CPU by means of the **`sol.sh`** script. After you've made the network settings to connect both CPUs to your management network, you can SSH into each CPU through the "MGMT" Ethernet port on the front panel.

2.1 Accessing the BMC CPU and main CPUs through the serial console port

You can connect to the BMC via the RJ45 console port (labeled, "CONSOLE," on the left side of the Wedge 100B front panel), or via the front panel Ethernet port (labeled "MGMT"). On power up, the CONSOLE port connects to the BMC CPU. When the system is powering up, allow approximately 2.5 minutes for this connection to be made.

Use a rollover cable and the following settings to connect to the BMC CPU:

- **Console settings**: set your serial console to 9600/8-N-1 (9600 baud, 8 data bits, no parity, and 1 stop bit)
- **Credentials**: The default BMC CPU administrator account is **`root`** with default password **`0penBmc`** (the first character in the password is a zero).

2.1.1 Connecting to main CPU via BMC CPU: sol.sh

From the BMC CPU shell (OpenBMC shell), you can start a terminal session on the main CPU by running the *sol.sh* script:

```
bmc login: root
Password:
root@bmc:~# sol.sh
Starting client to connect to microserver...
```

> **Troubleshooting tip**: If the main CPU does not respond, follow the workaround steps listed in the section, "SYS-1: Main CPU is not powered up by the time OpenBMC is up" on page 65.

2.1.1.1 Exiting sol.sh

From the main CPU's shell, you can exit *sol.sh* and return to the OpenBMC shell as follows:

- On Barefoot's OpenBMC image version 1.0.0.3 and earlier, by typing: `Ctrl-x`
- On Barefoot's OpenBMC image version 1.0.0.4 and later, by typing: `Ctrl-L` + `DEL`

Note: `DEL` is the *Delete* or *DEL* key on Mac keyboard, and it's the *Backspace* key on a Microsoft-style keyboard.

2.1.1.2 sol.sh Commands and Logging

On Barefoot's OpenBMC image version 1.0.0.4 and later, the *sol.sh* script provides the following commands:

- Display help: `Ctrl-L` + ?
- Exit the main CPU session and return to BMC CPU: `Ctrl-L` + `DEL`
- Send a break to interrupt the current operation: `Ctrl-L` + `b`

The *sol.sh* script logs its activity to **`/var/log/mTerm_wedge.log`** on the BMC CPU image.

2.2 Appliance default login credentials

- **BMC CPU**: Default administrator account is **root** with default password **0penBmc** (first character is a zero).
- **Main CPU**: Default account depends on the operating system:
  - For Ubuntu-based systems, the default administrator account is **admin12** with default password **bfn123**.
  - For ONIE/ONL-based systems, the default administrator account is **root** with default password **onl**.

2.3 Management Ethernet Port

A 10/100/1000 management port on the front panel provides management access to the BMC CPU and the main CPU.

2.3.1 Configuring the main CPU's IP address

Connect the Wedge 100B main CPU to your management network using the MGMT port on the front panel.

> **Note!** The management network *cannot* be the 192.168.0.0/24 subnet. This is because there is an internal network between the BMC CPU and the main CPU that occupies 192.168.0.1 and 192.168.0.2. Because the BMC CPU uses a non-persistent file system, its subnet cannot be changed without a recompile.

By default, the main CPU's Ubuntu OS obtains network connectivity by performing DHCP. Alternatively, you can configure the main CPU to use a static IP address as shown here:

1. Connect to the BMC CPU via serial console.

2. Connect to the main CPU by running the *sol.sh* script:

```
root@bmc:~#sol.sh
```

3. Press <Enter> to connect to the main CPU console.

4. Run *ifconfig* to show the current IP address:

```
admin12@bfn-switch:~$ ifconfig enp2s0
enp2s0    Link encap:Ethernet  HWaddr 00:90:fb:5c:cb:54
          inet addr:10.201.200.158  Bcast:10.201.201.255  Mask:255.255.254.0
          inet6 addr: fe80::638e:f7c9:d099:fd94/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

5. Assign the static IP address. Edit the interfaces file:

```
admin12@bfn-switch:~$ sudo vi /etc/network/interfaces
```

6. In the /etc/network/interfaces file, add or update the following lines with your desired network settings:

```
auto enp2s0
iface enp2s0 inet static address <ip address> netmask <netmask> gateway <gateway ip address>
sudo reboot
```

7. Restart the interface to apply your changes.

```
ifdown enp2s0; ifup enp2s0
```

Note! If you are connected over SSH, you will be logged out. Reconnect using the new IP address.

2.3.2 Logging into the main CPU over SSH

The main CPU's Ubuntu OS supports SSH, SCP, and SFTP operations after management network connectivity is established. To find the main CPU's management IP address, use the steps described in "Configuring the main CPU's IP address" on page 14. Use SSH to connect to the main CPU using the administrator account. (This is typically **admin12** or **root**. See "Appliance default login credentials," earlier in this document.):

```
$ ssh admin12@203.0.113.158
```

2.4 USB Port

The USB type-A host port on the Wedge 100B front panel allows you to connect an external USB flash drive. This port is accessible to the main CPU.

3 Using the Capilano™ SDE on the Wedge 100B

Barefoot introduces the industry's first fully programmable data plane. The Barefoot Networks Capilano™ Software Development Environment (SDE) allows you to program the data plane using the P4 language and compile your P4 program to run on the Barefoot Tofino chip or on a behavioral model that simulates the chip's behavior.

The SDE is the set of software packages you use to develop P4 programs. The SDE packages are broadly categorized into:

- Device models (ASIC and Behavioral)
- P4 compilers
- Drivers for Barefoot Networks' devices
- Switching and Routing Package (P4 and associated semantic libraries)
- P4 testing framework
- Example P4 programs and associated tests

In this section, we explain how to install and use the SDE.

3.1 Getting the SDE

Download the latest Capilano SDE from the Barefoot Networks customer portal at support.barefootnetworks.com.

1. Download the SDE bundle, **`bf-sde-<version>.tar`**
2. Copy the file to the Wedge 100B main CPU

```
me@my-desktop:~$ scp bf-sde-<version>.tar admin12@<my Wedge 100B host>:/home/admin12
```

   For the rest of this example, we'll use an example version "3.2.2-40" and an example Wedge 100B address, 203.0.113.101, so the example scp command is:

```
admin12@bfn-switch:~$ scp bf-sde-3.2.2-40.tar admin12@203.0.113.101:/home/admin12
```

3. Unpack the archive:

```
admin12@bfn-switch:~$ tar –xvf bf-sde-3.2.2-40.tar
```

4. Enter the SDE directory. This is your SDE home directory. You can source the **`set_sde.bash`** helper script to set the needed environment settings:

```
admin12@bfn-switch:~$ . ./set_sde.bash
```

Please note that in-service SDE upgrades are not currently supported.

3.2 Make sure SDE dependencies are present

Dependency packages are required for building and running the Capilano SDE and its tools such as PTF. Some Wedge 100B systems require you to install these:

- On Wedge 100B systems that run the *Ubuntu operating system*, the dependencies are already installed. Please proceed to the section, "Building the SDE," below.
- On Wedge 100B systems that run the *ONL operating system*, you must install these dependencies. See the section below, "Installing SDE Dependencies."

3.2.1 Installing SDE Dependencies

On Wedge 100B systems that run the ONL operating system, you must install additional packages before you can build and run the Capilano SDE (and all Barefoot tools such as PTF) on the system. To install:

1. Download **dependencies.tar.gz** from the "Wedge 100B ONL Operating System Download" page on the Barefoot Networks support portal.
2. Move the **dependencies.tar.gz** file to the Wedge 100B main CPU, and unpack the tar.gz archive:

```
root@localhost:~# tar zxvf dependencies.tar.gz
```

3. Change to the dependencies directory and run the **install.sh** script:

```
root@localhost:~# cd dependencies
root@localhost:~/dependencies# ./install.sh
```

3.3 Building the SDE

Barefoot Networks customer support offers scripts to simplify setting up and using the SDE (`set_sde.bash` and `sde_build.sh`). For instructions on using the scripts, see the article "Building Barefoot SDE in Two Easy Steps" on the customer support portal, and make sure you download the latest versions of these scripts.

It is not necessary to build the Capilano™ SDE on the Wedge 100B system itself. You can build it elsewhere on a build system and scp-upload the installed artifacts and required ptf-scripts to the Wedge 100B before exercising them. For the sake of convenience, the Ubuntu image on the Wedge 100B includes the packages needed to build the SDE. Below, we provide a brief description of the steps to build and use the SDE.

Keep in mind when building the SDE:

- The **bf-drivers** package needs to be built **without** specifying the `--with-model` option
- When **bf-drivers** is built offline, it produces the Tofino UIO kernel driver module (`bf_kdrv.ko`). It's essential that the Linux kernel versions of the build system and that of the Wedge 100B system match. You can confirm this by running `uname –a`.
- The `bf-platforms` artifacts are required to exercise the Tofino ports.
- When building on the Wedge 100B, there is no need to build modules that are associated with software-only behavioral models such as `bmv2` or `tofinobm`.

Refer to the **$SDE/README** file and other **README** files of the SDE for more detailed instructions. Once the SDE installation artifacts are made available on the system, read the sections below to find out how to use the SDE.

> **Building the SDE without an Internet connection**: If you're building the SDE on a system that's not connected to internet, such as a Wedge 100B system on an isolated lab network, the following command-line flags are helpful:
>
> `./sde_build.sh -r --no-deps --no-bmv2`
>
> When run with these flags, the script skips installing dependencies (which would have to be downloaded over the internet). Run in this way, the build relies on pre-installed components and skips building BMv2-related software.

Next steps: Before you can run the SDE, on most systems you must load the Tofino driver as explained in the next section.

3.4 Loading the Tofino kernel mode driver (`bf_kdrv.ko`)

The `bf_kdrv.ko` driver is the standard kernel mode driver for the Tofino device. It provides the user space `mmap` service, support for the full range of MSI and MSIx vectors, and the user space `wait for interrupt` and `enable interrupt` services.

The first step towards exercising the SDE is to load `bf_kdrv.ko` that is installed at `$SDE_INSTALL/lib/modules/bf_kdrv.ko`

```
admin12@bfn-switch:~$ cd $SDE
admin12@bfn-switch:~$ sudo ./install/bin/bf_kdrv_mod_load $SDE_INSTALL
```

Confirm the loading of the driver using the following steps

```
admin12@bfn-switch:~$ ls /dev/bf0
/dev/bf0
```

The kernel driver needs to be loaded every time the CPU is rebooted. Also, when there is an upgraded SDE, the kernel driver needs to be unloaded using the following step, before reloading the newer version.

```
admin12@bfn-switch:~$ cd $SDE
admin12@bfn-switch:~$ sudo ./install/bin/bf_kdrv_mod_unload
```

> **Attention**: An earlier version of the kernel mode driver, bf_uio.ko is deprecated but remains in the distribution to ensure backwards compatibility. The bf_uio.ko driver is not recommended and is no longer supported.

Next steps:

- If you've just upgraded the SDE, proceed to "Start your P4-defined switch on the Tofino hardware," below.
- If this is a new Mavericks system or if you've upgraded the operating system on it, you must set the HugePages settings correctly Before you can run the SDE. See "Set HugePages memory settings," below.

3.4.1 Minimizing rebuilds of bf_kdrv.ko

If kernel updates are applied to the main CPU's operating system, the Tofino kernel mode driver (bf_kdrv.ko) must be rebuilt. To avoid having to rebuild the driver, you must refrain from updating the main CPU's kernel and disable Ubuntu's automatic OS updates feature. Follow the steps below to disable automatic updates:

1. Working in a terminal shell on the Wedge 100B main CPU, use a text editor to open the file, `/etc/apt/apt.conf.d/10periodic`
2. In the file, disable automatic package downloads by editing the following line, setting the parameter to zero as shown here: `APT::Periodic::Update-Package-Lists "0";`
3. Make sure the **`Download-Upgradeable-Packages`** and **`AutocleanInterval`** settings are set to zero.
4. Save the file.
5. Disable kernel updates by typing:

```
admin12@bfn-switch:~$ sudo apt-mark hold linux-image-generic linux-headers-generic \
linux-generic
```

3.5 Set HugePages memory settings

Before you can run the SDE on a Wedge 100B system, you must perform a one-time setup to allocate huge pages required by DMA buffer allocation. The SDE provides a script for this. Run it as follows:

```
admin12@bfn-switch:~$ $SDE_INSTALL/bin/dma_setup.sh
```

You can check the HugePages setting by typing `cat /proc/meminfo`. The correct settings look like:

```
...
HugePages_Total:    128
HugePages_Free:     128
HugePages_Rsvd:       0
HugePages_Surp:       0
...
```

If the HugePages options are not set correctly, and you attempt to run a P4 program in **`run_switchd`**, you will see errors such as `alloc_huge_pages() mmap failed`, `dma_pool_create failed(-1)`, or as shown below:

```
Starting BF-PLATFORM RPC server on port Tcl server: bind done on port 8008, listening...
...
bf_switchd: drivers initialized
ERROR: bf_sys_dma_buffer_alloc for dev_id 0 failed(-1)
```

Next steps: Proceed to "Start your P4-defined switch on the Tofino hardware," below.

3.6 Start your P4-defined switch on the Tofino hardware

To start the switch on the Wedge 100B and load its logic from your P4 program, you must run a switch runtime environment with Tofino hardware drivers. Barefoot provides a sample switch runtime called **`bf_switchd`** for this.

Prerequisites:

- SDE has been built and installed and built. See "Building the SDE," above.
- **`SDE`** and **`SDE_INSTALL`** environment variables have been set and your **`PATH`** contains **`$SDE_INSTALL/bin`**
- Tofino kernel mode driver has been loaded. See "Loading the Tofino kernel mode driver," above.
- The P4 program has been compiled and all its artifacts are available in SDE home directory.

Run the **`bf_switchd`** sample application, passing it the name of the P4 program that defines your switch logic. The **`bf_switchd`** application loads the user space drivers, applies the P4 program to the Tofino device, and gives you access to the **`bf-sde`** CLI for Wedge 100B platform management.

Note! Here, we show how to start the switch on the Tofino hardware. For testing, you also have the option of starting the switch on behavioral models (for example BMV) that simulate Tofino switch behavior. For behavioral model instructions, contact the Barefoot support team.

The SDE provides a script, **`run_switchd.sh`**, for launching **`bf_switchd`**. Use it as shown below. The **`-p`** argument provides the name of the compiled P4 program that describes your switch behavior. Pass the P4 program name *without* its **`.p4`** suffix, and provide the root password when prompted:

```
admin12@bfn-switch:~$ cd $SDE
admin12@bfn-switch:~$ ./run_switchd.sh -p <p4_name>
```

For example, to run the sample P4 program, **switch.p4**, you will type:

```
admin12@bfn-switch:~$ ./run_switchd.sh -p switch
```

Wait for the switch program to start, and hit <Enter> to proceed to the bf-sde CLI prompt. This CLI allows you to add ports and manage the Wedge 100B platform.

To confirm the Tofino device has been initialized, use the following CLI command:

```
bf-sde> dev -d 0
-----------------------------------
Device|Type |#pipe|#stg|#prsr|#macblk
------|-----|-----|----|-----|-------
0     |T64Q |4    |12  |18   |65
```

You can optionally start **bf_switchd** with bare minimal initialization of Tofino using **--skip-*** options, as shown here:

```
admin12@bfn-switch:~$ ./run_switchd.sh -p <p4_name> --skip-p4 --skip-hld pmkrt --skip-port-
add
```

The initialization of the device can be confirmed using basic register PIO access diagnostics

```
bf-sde> devdiag
bf-sde.devdiag> regtest -d 0 -i 0 -t 0 -l 0
bf-sde.devdiag> regtest -d 0 -i 1 -t 0 -l 0
bf-sde.devdiag> regtest -d 0 -i 2 -t 0 -l 0
```

3.6.1 Troubleshooting the SDE and run_switchd

3.6.1.1 Resources for troubleshooting

For SDE troubleshooting help, see the knowledgebase article, Common issues during SDE build and bring-up.

3.6.1.2 Restarting bf_switchd

If you're unable to run a P4 program in run_switchd.sh, make sure nobody is using your Wedge 100B and follow these steps:

1. At the Main CPU command line, use **lsmod** to check if the Tofino kernel driver is running:

```
$ lsmod | grep bf
bf_kdrv                20480  2
```

2. Find the process number of the running **bf_switchd** instance.

```
$ ps -ax | grep switchd
26897 ?        Sl   2868:51 bf_switchd --install-dir /home/admin12/bf-sde-3.3.0.14/install --
conf-file /home/admin12/bf-sde-3.3.0.14/install/share/p4/targets/diag.conf
```

3. Kill the **bf_switchd** process.

```
$ sudo kill <process id from the previous command>
```

4. Run your P4 program as usual using **run_switchd.sh**. For example:

```
$ ./run_switchd.sh -p switch
```

3.6.2 How a Switch Configuration is Loaded on the Wedge 100B

When you start the switch data plane program with the **run_switchd.sh** script, the Wedge 100B loads switch configurations such as port mapping from a configuration file. The **run_switchd.sh** script accepts the parameter flag `-c TARGET_CONFIG_FILE` to specify the config file. If not specified, the default config file for the loaded P4 program will be used.

Loading of the config file is handled by the **bf_switchd_load_switch_conf_file()** function. In the .conf file, if "switchapi" is specified, then the specified **libswitchapi.so** will be loaded, the **dlsym()** invocation on the symbols in **libswitchapi.so** will return success, and SwitchAPI can be initialized.

A call to **switch_api_init()**adds all ports as 10G ports if **add_ports** is specified (**add_ports** is from **p4_cfg->add_ports_to_switchapi**; the default value is *true*). In **bf_switchd_switchapi_lib_init()**, we invoke **dlsym** and check the state of the **switchapi_lib_handle** for "switch_api_init") and proceed if that value is present. The actual port-add operation is at **switch_device_init()** and runs if **SWITCH_CONFIG_PORT_ADD()** is true.

The front port mapping is specified in the related board header file at **./pkgsrc/bf-platforms/drivers/src/bf_pltfm_bd_cfg/**. The header file of the system board (Wedge 100BF-32X or Wedge 100BF-65X, for example) will be used during the source code compilation process. To facilitate header file generation, the mapping can be specified in a .csv file, and tools can be used to generate the header file from the .csv file. Please see the Barefoot Support knowledge base article, *Porting Barefoot Platform Port Manager to a New Board* for more details on port mapping.

4 Managing Tofino Ports

The sections below show how to use the bf-sde CLI to manage Tofino ports. You also have the option of scripting your port management using the **<pltfm_pm_rpc>** Python package, as explained in "Managing ports from a PTF script" on page 30.

4.1 Tofino Internal Ports

By default, upon initialization **bf_switchd** instantiates only Tofino's internal ports (for example, the PCIe port, PktGen port, and recirculation port). That is, by default, **bf_switchd** instantiates only ports that are *not* associated with a QSFP/MAC device. These are ports 64-71 on all the pipes 0, 1, 2, and 3 except for ports 64-67 on pipe 0, which are associated with QSFP-65 on the front-panel. Such internal ports are instantiated in 100G mode.

> **Note!** That the P4 program may override this and instantiate QSFP ports (in addition to the internal ports **bf_switchd** instantiates by default) upon initialization. For example, the **switch.p4** sample program initiates all ports at 10G speed.

```
bf-sde> cfg
DVM:: Board configuration:
+----+------+-----+---+--------------------------------
|dev | port
+----+------+-----+---+--------------------------------
|  0 |  68
|  0 | 192
|  0 | 196
|  0 | 320
|  0 | 324
|  0 | 448
|  0 | 452
+----+------+-----+---+--------------------------------
```

4.2 Tofino Switch Ports

Tofino ports associated with QSFP devices are *not* created by default. The QSFP-associated ports can be created and managed in two ways:

- using the bf-sde CLI commands, in particular, the port manager (**pm**) commands; or
- in a PTF-script using python-thrift API.

Note! Before you use the CLI, review the section "Overview of the bf-sde CLI" on page 33.

Enter the **pm** context to use the port manager CLI commands:

```
bf-sde> pm
```

In the sections that follow, we will describe the port manager commands in detail. In summary, they are:

- Add a port: **port-add <conn_id/chnl> <speed (1G, 10G, 25G, 40G, 40G_NB, 50G, 100G, 40G_NON_BREAKABLE)> <fec (NONE, FC, RS)>**
- Delete a port: **port-del <conn_id/chnl>**
- Enable a port: **port-enb <conn_id/chnl>**
- Disable a port: **port-dis <conn_id/chnl>**
- Show port information: **show -a -p <conn_id/chnl> [-d]**

4.3 Wedge 100B Port Naming

The column headings of the port manager's **`show -a`** command allow you to map the various port names:

```
bf-sde> pm
bf-sde.bf_pltfm.pm> show -a
----+----+---+----+------+----+----+---+---+----------------+----------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX       |FRAMES TX       |E
----+----+---+----+------+----+----+---+---+----------------+----------------+-
 1/0|31/0|188|1/60|------|----| NO |---|---|----------------|----------------|-
 1/1|31/1|189|1/61|------|----| NO |---|---|----------------|----------------|-
...
```

The **`pm`** `'show -a'` command output shows the status of all ports, and from that you can see mappings among the different port designations. The Barefoot SDE uses the following port designations:

- **`PORT`**: QSFP connector number / QSFP lane number of this connection, where the QSFP connector number is the number shown on the Wedge 100B front panel, and the lane number shows the lane for a channelized connection, or 0 for a non-channelized connection such as a port configured for 100G. Use this **`PORT`** designation to refer to the port when invoking bf-sde CLI commands. For example, running the command `port-add 1/0 ...` adds a port for the left-most front panel QSFP connector. For a guide to the front panel numbers, see the illustration in the Platform Overview section, earlier in this document.
- **`MAC`**: Tofino's QUAD-MAC/block ID of this connection. In this slash-separated designation, the first number is the Tofino MAC device number (MAC block ID), and the second number is the MAC channel number on that MAC device. *Tip*: You can use the **`bf-sde.bf_pltfm.bd_cfg.mac_get`** command to find this ID for a given port and lane.
- **`D_P`**: Tofino's device port ID ("device_port_id," a 9-bit value) of this connection. This is the port designation used in P4 programming, in Python test framework (PTF) scripts, and when running Diags commands.
- **`P/PT`**: Tofino's pipe_id (2-bit value) / pipe_local_port_id (7-bit value) of this connection

In the illustration that follows, we provide an overview of MAC-to-QSFP mappings on the Wedge 100B.

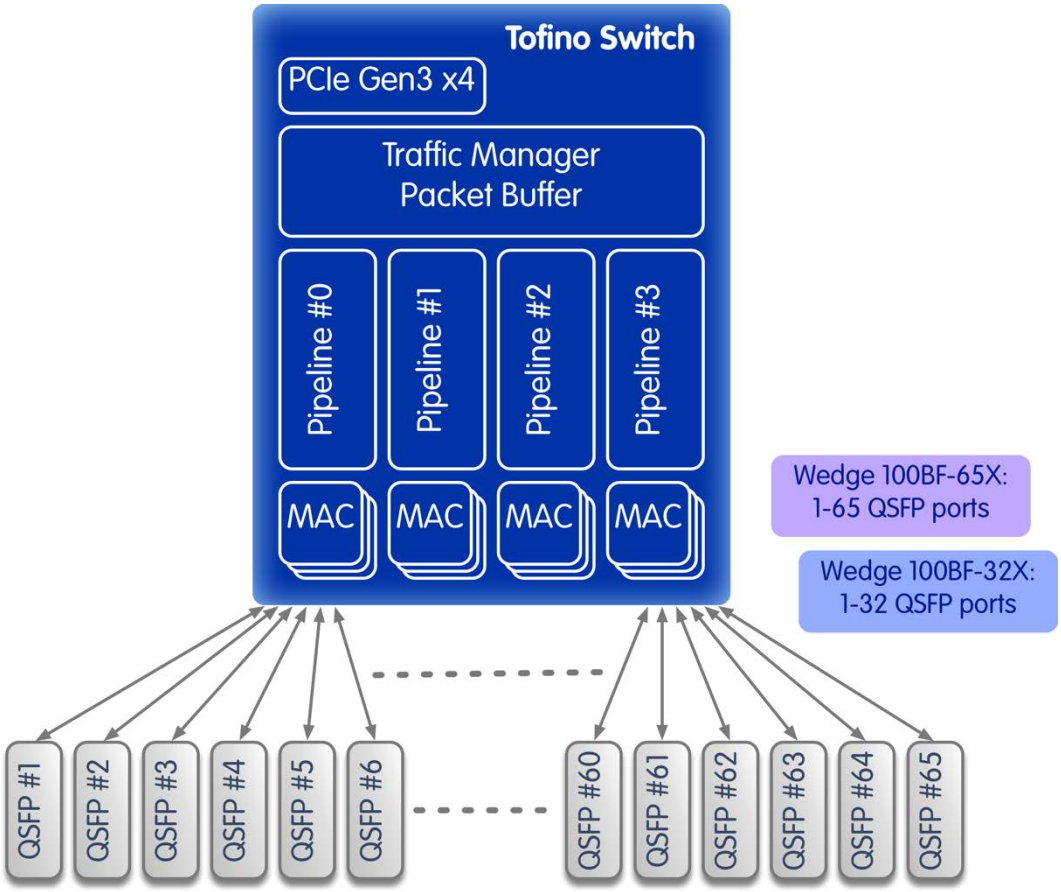


Figure 3: Pipeline, MAC, and QSFP mappings on the Barefoot Wedge 100B

4.4 Summary of bf-sde CLI port management commands

The most commonly used bf-sde CLI commands for port management are listed below. To run some of these commands, you must be in a particular CLI context, as noted. Others can be run from any CLI context.

- **`pm`**: Enter the platform port manager (**`bf_pltfm.pm`**) CLI context
- **`port-add`**: Add port(s). For example, to add all ports at 100G: **`port-add -/- 100G NONE`**
- **`port-enb`**: Enable port(s). For example, to enable all ports: **`port-enb -/-`**
- **`show`**: Show summary status of all ports. Works only in the **`bf_pltfm.pm`** CLI context.
- **`op`**: Show status of all SerDes. Works only in the root CLI context.
- **`ports`**: Show status of all ports. Works only in the root CLI context.
- **`oper`**: Show status of operational ports. Works only in the root CLI context.

4.5 Adding and enabling ports

Use the **port-add** and **port-enb** commands to add and enable ports, as shown below.

4.5.1 QSFP lane restrictions on port creation

The speed of the port determines which QSFP lane (channel) it can occupy. When creating ports, observe the following QSFP lane restrictions.

Speed	QSFP lane 0	QSFP lane 1	QSFP lane 2	QSFP lane 3
10G	Y	Y	Y	Y
25G	Y	Y	Y	Y
40G	Y			
50G	Y		Y	
100G	Y			

Table 1: QSFP lane restrictions on port creation

For example, a 10G port can be instantiated in any lane, but a 100G port can be instantiated *only* in QSFP lane 0.

At initialization, `bf_switchd` does not, by default, instantiate ports associated with QSFP/MAC. A user or application can create the required ports at the required speeds using any of the following two methods:

- Use CLI commands in the **pm** context of the bf-sde CLI
- Use a PTF script to invoke the appropriate **pm** API over Thrift

4.5.2 Adding a port

Use the **port-add** CLI command to add a port:

```
port-add <conn_id/chnl> <speed> <fec>
```

where

- **conn_id/chnl**: QSFP connector cage / QSFP lane of the port to be added
- **speed**: port nominal speed, which must be one of 1G, 10G, 25G, 40G, 50G, or 100G
- **fec**: forwarding error correction setting, which must be one of **NONE**, **FC**, or **RS**, where **NONE** disables forwarding error correction, **FC** refers to BASE-R FEC (Fire Code FEC) and **RS** refers to Reed-Solomon FEC (RS-FEC)

> **Note**: If auto-negotiation is *not* used, you must make sure the **SPEED** and **FEC** settings of the port match the settings of its link partner.

These CLI commands invoke the Barefoot platform's port management API ("port-mgmt API") defined in `<bf_pltfm_pm/bf_pltfm_pm_intf.h>`. The **bf_pltfm_pm** API maps front panel port numbers to Tofino device_port_ids before invoking the Tofino driver API to manage ports and SerDes.

For example, to create a 10G port on QSFP connector 1, lane 0, you would type:

```
bf-sde.bf_pltfm.pm> port-add 1/0 10G NONE
```

To check that the port has been created, you can retrieve port information using the **show** command

4.5.3 Enabling a port

By default, when a port is added it remains in disabled state, as shown in the **show** command output:

```
bf-sde.bf_pltfm.pm> show
----+----+---+----+------+----+----+---+---+----------------+----------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX       |FRAMES TX       |E
----+----+---+----+------+----+----+---+---+----------------+----------------+-
 1/0|31/0|188|1/60| 10G  |NONE| NO |DIS|DWN|               0|               0|
```

Enable a port using the following CLI shell command:

```
bf-sde.bf_pltfm.pm> port-enb 1/0
bf-sde.bf_pltfm.pm> show
----+----+---+----+------+----+----+---+---+----------------+----------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX       |FRAMES TX       |E
----+----+---+----+------+----+----+---+---+----------------+----------------+-
 1/0|31/0|188|1/60| 10G  |NONE| NO |ENB|DWN|               0|               0|
```

A port that does not have a QSFP module plugged in is not completely enabled to conserve power.

4.6 Managing port features

4.6.1 Port autonegotiation

Autonegotiation (AN) allows a Wedge 100B port to communicate with its link partner to choose common transmission settings. You can enable or disable AN for each QSFP port/lane combination.

> **Note**: Autonegotiation may be enabled for any port, but it takes effect only for the eligible QSFP types, which as Copper 0.5m, 1m and 3m.

When a port is created, AN is OFF by default. To enable AN for a port, use the **port-auto-neg-set** command to turn it on. You must do this after you add the port and before you enable it. The command for AN is available in the **bf-sde.bf_pltfm.pm** context:

```
bf-sde.bf_pltfm.pm> an-set <conn_id/chnl_id> <AN_setting>
```

where **AN_setting** is one of:

- 0 to accept the default determined by the SDE;
- 1 to force enablement of AN; or
- 2 to force disablement of AN.

For example, to turn AN on:

```
bf-sde.bf_pltfm.pm> an-set 1/0 1
```

To enable AN for all ports, use the wildcard options for the conn_id and chnl_id parameters:

```
bf-sde.bf_pltfm.pm> an-set -/- 1
```

4.6.1.1 Setting debounce time interval for ports that use autonegotiation

When AN is enabled, if Tofino's platform manager (**bf-platforms**) detects that a port which was autonegotiating goes down and remains down for a period longer than the debounce time interval set for that port, it will try to bring up the port. The manager does this by disabling and enabling the port internally after every **LINK_BRING_UP_TIME_INTV_MS** has elapsed. It will try to do this until the port comes up or is disabled by the user.

You can set the link debounce time interval for a port using the **link-dbn-time** CLI command.

```
bf-sde.bf_pltfm.pm> link-dbn-time <conn_id/chnl_id> <debounce time in ms>
```

This command supports wildcard options for the conn_id and chnl_id params.

4.7 Disabling and deleting ports

4.7.1 Disabling a port

An enabled port can be disabled using the following CLI shell command

```
bf-sde.bf_pltfm.pm> port-dis 1/0
bf-sde.bf_pltfm.pm> show
----+----+---+----+------+----+----+---+---+---------------+---------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX      |FRAMES TX      |E
----+----+---+----+------+----+----+---+---+---------------+---------------+-
 1/0|31/0|188|1/60| 10G  |NONE| NO |DIS|DWN|              0|              0|
bf-sde.bf_pltfm.pm>
```

4.7.2 Deleting a port

The following CLI shell command is used to delete a port

```
port-del <conn_id/chnl>
```

For example:

```
bf-sde.bf_pltfm.pm> port-del 1/0
bf-sde.bf_pltfm.pm>
```

4.8 Wildcard port operations

Instead of a specific <QSFP> and/or <lane> a hyphen character ("-") can be used as a wildcard. For example, all lanes on QSFP connector 1 can be added as 10G ports using the following commands:

```
bf-sde.bf_pltfm.pm> port-add 1/- 10G NONE
bf-sde.bf_pltfm.pm> show
----+----+---+----+------+----+----+---+---+---------------+---------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX      |FRAMES TX      |E
----+----+---+----+------+----+----+---+---+---------------+---------------+-
 1/0|31/0|188|1/60| 10G  |NONE| NO |DIS|DWN|              0|              0|
 1/1|31/1|189|1/61| 10G  |NONE| NO |DIS|DWN|              0|              0|
 1/2|31/2|190|1/62| 10G  |NONE| NO |DIS|DWN|              0|              0|
 1/3|31/3|191|1/63| 10G  |NONE| NO |DIS|DWN|              0|              0|
```

For example, all ports can be enabled in one shot using the following command

```
bf-sde.bf_pltfm.pm> port-enb -/-
```

Similarly, all ports can be deleted in a single command as shown here:

```
bf-sde.bf_pltfm.pm> port-del -/-
```

4.9 Getting information about ports

4.9.1 show command in bf_pltfm.pm

The CLI provides a port **show** command to show information about ports on Tofino. In the **bf_pltfm.pm** context, use the **show** command to retrieve the operational status and MAC statistics of switch ports. For an explanation of the command output, see the earlier section, “Wedge 100B Port Naming.”

```
bf-sde.bf_pltfm.pm> show
----+----+---+----+------+----+----+---+---+---------------+---------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX      |FRAMES TX      |E
----+----+---+----+------+----+----+---+---+---------------+---------------+-
 1/0|31/0|188|1/60| 10G  |NONE| NO |DIS|DWN|              0|              0|
bf-sde.bf_pltfm.pm>
```

The following optional parameters can be passed to the ‘show’ command

- “**-p <conn_id>/<chnl>**” to show status for a specified port (conn_id) and lane (chnl). For a non-channelized connection, you must pass a lane value of 0.
- “**-d**” to show detailed output
- “**-a**” to show all ports including physical ports that have not been configured or enabled.

Tip: You can also run the port **show** command from the root context, as follows:

```
bf-sde> pm show
----+----+---+----+------+----+----+---+---+---------------+---------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX      |FRAMES TX      |E
----+----+---+----+------+----+----+---+---+---------------+---------------+-
 5/0|27/0|172|1/44| 100G |NONE|YES |ENB|UP |              0|              0|
```

Resetting port counters: Port counters (Frames RX, Frames TX) can be reset only by deleting and re-creating the port. Use the **port-del** and **port-add** commands.

```
bf-sde.bf_pltfm.pm> show -p 1/0 -d
================================================================================
             1/0 : Port
            31/0 : MAC
             188 : Dev Port
            1/60 : Pipe/Port
           10G   : Speed
            NONE : FEC
              NO : QSFP
             DIS : Admin State
            DOWN : Operational Status
               0 : FramesReceivedOK
               0 : FramesReceivedAll
               <snip>
               <snip>
               0 : FramesTruncated
```

4.9.2 Show all ports

Use the port manager's **`show -a`** command to print current information about all physical switch ports including ports that have not been configured:

```
bf-sde> pm
bf-sde.bf_pltfm.pm> show –a
bf-sde.bf_pltfm.pm> show
----+----+---+----+------+----+----+---+---+----------------+----------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX       |FRAMES TX       |E
----+----+---+----+------+----+----+---+---+----------------+----------------+-
 1/0|31/0|188|1/60| 100G |NONE|YES |ENB|UP |               0|               0|
 2/0|30/0|184|1/56| 100G |NONE|YES |ENB|UP |               0|               0|
 3/0|29/0|180|1/52| 100G |NONE|YES |ENB|UP |               0|               0|
 4/0|28/0|176|1/48| 100G |NONE|YES |ENB|UP |               0|               0|
...
```

4.9.3 Get connector information using the dump-info command

From the **`bf_pltfm`** context, use the **`dump-info <conn_id/chnl>`** command to get connector hardware information about a QSFP port connected to Tofino.

```
bf-sde.bf_pltfm> dump-info 1/0
Port 1
  ID: 0x0d
  Status: 0x00 0x06
  Interrupt Flags:
    LOS: 0x00
    Fault: 0x00
    ...
    Vendor Defined: 0x00 0x00 0x00
  Temperature: 1.503906 C
  Supply Voltage: 0.000000 V
  Channel Data:      RX Power         TX Bias
    Channel 1:       0.000000mW      0.000000mA
    Channel 2:       0.000000mW      0.000000mA
    ...
```

4.10 Managing ports from a PTF script

The **pltfm_pm_rpc** Python package exports the following functions to perform add, delete, enable and disable functions on a port. The following arguments are used with the functions

- **device = 0** (instance of Tofino)
- **dev_port** (Tofino's 9-bit device_port_id)
- **ps** (Port's MAC Speed)
- **fec** (Port's MAC forwarding error correction setting)

```
pltfm_pm_port_add(self,
  device,
  dev_port,
  ps,
  fec)

pltfm_pm_port_del(self,
  device,
  dev_port)

pltfm_pm_port_enable(self,
  device,
  dev_port)

pltfm_pm_port_dis(self,
  device,
  dev_port)
```

Examples that use the above functions can be found in `<bf-platforms>/ptf-tests/pltfm_pm/test.py`

5 Exercising PTF scripts to configure Tofino

PTF scripts can be exercised on the Wedge 100B to configure the Tofino ASIC. The PTF script can invoke Python versions of APIs that are exposed over Thrift-RPC. The following classes of API are exported over the Python Thrift RPC interface:

- **`P4 PD API`**: P4-dependent API that's auto-generated by p4-build from the P4 program
- **`switchAPI`**: Higher level semantic API associated with switch.p4
- **`switchSAI`**: Switch abstraction interface (SAI) associated with switch.p4
- **`diagAPI`**: Higher level semantic API associated with diag.p4
- **`Platform Port Mgmt API`**: Add, edit, and monitor ports

Below, we provide a few examples that show how PTF scripts can be run on the system.

> **Note!** In this document, we use <p4-examples> to indicate the SDE's example switch programs directory. The exact path depends on the version of the SDE you are running. For example, for SDE 3.3.0, <p4-examples> has the path, **`$SDE/pkgsrc/switch-3.3.0.14`**. For information on installing the SDE and setting $SDE, see the support article, Building the SDE in Two Easy Steps.

5.1 Example 1: Setting up P4 table rules for basic_switching.p4

The following steps demonstrate configuration of P4 table rules for a simple P4 program, **`basic_switching.p4`**. The setup assumes usage of two ports.

5.1.1 Start bf_switchd with basic_switching.p4

Use the `-p` flag, and specify the name of your P4 program without its ".p4" suffix.

```
$ cd $SDE
$ ./run_switchd.sh -p basic_switching
```

5.1.2 Run the PTF script to set up the P4 table rules

```
$ cd $SDE
$ ./run_p4_tests.sh -t <p4-examples>/ptf-tests/eval/basic_switching/ --setup
```

5.2 Example 2: Setting up IPv4 Routing Table for switch.p4 using switchAPI

The following steps demonstrate configuration of a layer-3 routing table over ECMP using switchAPI. More details about the test configuration can be found in **`<p4-examples>/ptf-tests/eval/api/l3.py`**.

5.2.1 Start bf_switchd with switch.p4 and switchAPI library

```
$ cd $SDE
$ ./run_switchd.sh -p switch -c $SDE_INSTALL/share/p4/targets/switch.conf
```

5.2.2 Run the PTF script to set up the IPv4 Routing Table

```
$ cd $SDE
$ ./run_p4_tests.sh -t <p4-examples>/ptf-tests/eval/api/ --setup
```

5.3 Example 3: Setting up IPv4 Routing Table for switch.p4 using P4-PD-API

The following steps demonstrate configuration of the L3 routing table over ECMP using P4 PD API. More details about the test configuration can be found in **`<p4-examples>/ptf-tests/eval/pd/l3.py`**.

5.3.1 Start bf_switchd with switch.p4 and PD-API library

```
$ cd $SDE
$ ./run_switchd.sh -p switch -c $SDE_INSTALL/share/p4/targets/switch-pd.conf
```

5.3.2 Run the PTF script to set up the IPv4 Routing Table

```
$ cd $SDE
$ ./run_p4_tests.sh -t <p4-examples>/ptf-tests/eval/pd/ --setup
```

5.3.3 Dumping the IPv4 Routing Table

The python script **`<p4-examples>/ptf-tests/eval/dump.py`** can be used to dump tables and action profiles used in **`switch.p4`**. The script reads the entries from the hardware using P4 PD APIs. In the script, the class variables TABLES and ACTION_PROFILES enumerate the list of tables and action profiles that can be dumped. To use the script, invoke it as follows:

```
$ cd $SDE
$ python <4-examples>/ptf-tests/eval/dump.py --install-dir $SDE_INSTALL
```

6 Managing the Wedge 100B Platform

6.1 Overview of the bf-sde CLI

The bf-sde command line interface (CLI) gives you access to commands for managing and monitoring the Tofino chip and the Wedge 100B network ports and chassis.

6.1.1 Launching the bf-sde CLI

The bf-sde CLI is available as soon as you've brought up the switch by running its P4 program in bf-switchd. In the shell where you launched bf-switchd, wait for the switch initialization to finish, and then press the <Enter> key. The `bf-sde>` prompt will appear, indicating the CLI is ready to accept commands.

6.1.2 Navigating the bf-sde CLI

At any level of the CLI, type `?` to show the commands sub-packages and that are available at that level. Type the name of sub-package to enter it. Type `..` to exit a sub-package and return to its parent.

To quit the CLI, type `exit`.

6.1.3 Top-level CLI commands in bf-sde

The commands available at the top level of the CLI are:

- **`quit`**: Stop the bf-sde shell thread
- **`dump-trace`**: Display the trace logs
- **`reset-trace`**: Clear the trace logs
- **`set-trace-level`**: Set the trace level for a module
- **`set-log-level`**: Set the log level for a module
- **`get_trace`**: get_trace <size>
- **`ver`**: Display versions of all loaded/linked BF SDE components
- **`add-vdev`**: Add virtual device
- **`rmv-dev`**: Remove device

6.2 Board Port Configuration

The board port configuration commands are provided in the **`bd_cfg`** package:

```
bf-sde> bd_cfg
```

For the following commands, you must pass a slash-separated "conn_id/chnl" argument specifying the QSFP port (conn_id) and lane (chnl). For a non-channelized QSFP connection, you must pass a lane value of **`0`**. Other options are listed where they apply:

- mac_get <conn_id/chnl>
- tx_phy_lane_get <conn_id/chnl>
- rx_phy_lane_get <conn_id/chnl>
- tx_pn_swap_get <conn_id/chnl>
- rx_pn_swap_get <conn_id/chnl>
- lane_info_get <conn_id/chnl>
- rptr_info_get <conn_id/chnl> <Egress->0, Ingress->1>
- bd_type_get

- show -p <conn_id/chnl> [-d -r]

For example, to show the board port configuration, use the **bd_cfg**'s context's **show** command:

```
bf-sde.bf_pltfm.bd_cfg> show -p 1/0
----+----+----+-----+-----+-----+-----+------+------+-------+----
PORT|QUAD|CHNL|TX LN|TX PN|RX LN|RX PN|TX ATT|TX PRE|TX POST|RPTR
----+----+----+-----+-----+-----+-----+------+------+-------+----
 1/0| 31 |  0 |   2 |   0 |   2 |   0 |   0  |   4  |    0  |  0
```

Show configuration for a port:

```
bf-sde.bf_pltfm.bd_cfg> show -p 1/0 -d
================================================================================
QSFP Port : 1
QSFP Lane : 0
MAC id : 31
MAC channel id : 0
TX PHY lane id : 2
TX PN swap : 0
RX PHY lane id : 2
RX PN swap : 0
TX Attenuation : 0
TX Pre Emphasis : 4
TX Post Emphasis : 0
Repeater present : 0
Egress Repeater Number : 255
Egress Repeater I2C Address : 255
Egress Repeater Channel : 255
Egress Repeater EQ Boost 1 : 255
Egress Repeater EQ Boost 2 : 255
Egress Repeater EQ BW : 255
Egress Repeater EQ Bypass Boost 1 : 255
Egress Repeater VOD : 255
Ingress Repeater Number : 255
Ingress Repeater I2C Address : 255
Ingress Repeater Channel : 255
Ingress Repeater EQ Boost 1 : 255
Ingress Repeater EQ Boost 2 : 255
Ingress Repeater EQ BW : 255
Ingress Repeater EQ Bypass Boost 1 : 255
Ingress Repeater VOD : 255
```

6.3 Chassis Management

```
bf-sde> chss_mgmt
```

The platform chassis management commands are provided in the **chss_mgmt** package:

- sys_mac_get - System MAC address and number of extended address
- port_mac_get - MAC address of **<conn_id> <channel>** where **conn_id** is the QSFP port number and **channel** is the QSFP lane number.
- eeprom_data - system serial number and other identification information parsed by EEPROM
- tmp_show - Show the readings of all temperature sensors
- tofino_tmp_show - Show Tofino CPU temperature
- vrail_show - Show all voltage rails reading
- ps_show - Show status of power supply number **<1|2>**

- fan_show - Show all fan speed data
- fan_speed_set - Set **<fan number>** speed in **<%(0...100)>** of max speed

6.3.1 Get MAC addresses

To get the MAC address of the main CPU management interface, use sys_mac_get:

```
bf-sde.bf_pltfm.chss_mgmt> sys_mac_get
System Mac addr: 8c:ea:1b:93:ff:a7
Number of extended addr available 8
```

To get the MAC address of a front panel QSFP port and lane, use the **port_mac_get** command. For example, to get the address for QSFP port 7, lane 3, type:

```
bf-sde.bf_pltfm.chss_mgmt> port_mac_get 7 3
Port/channel:1/0  Port Mac addr: 8c:ea:1b:a8:10:0c
```

6.3.2 Dump EEPROM Contents

To retrieve the Wedge 100B system serial number, model number, and other identifying information, use the **eeprom_data** command. The **serial number** is listed as the "Product Serial Number" and the **model number** is listed as "ODM PCBA Part Number," and the **part number** is listed as the "Facebook PCB Part Number."

```
bf-sde.bf_pltfm.chss_mgmt> eeprom_data
Version: 1
Product Name: Lower MAV
Product Number: 20-000001
...
System Manufacturer: BAREFOO
System Manufacturing Date: 01-07-17
```

6.3.3 Tofino Temperature Monitoring

Use **tofino_tmp_show** to check CPU temperature:

```
bf-sde.bf_pltfm.chss_mgmt> tofino_tmp_show
TOFINO MAIN TEMP SENSOR: 40 C
TOFINO REMOTE TEMP SENSOR 43 C
```

6.3.4 Power Supply Monitoring

Use **ps_show** to check the power supply units, passing an argument of 1 or 2 to indicate the unit you wish to check:

```
bf-sde.bf_pltfm.chss_mgmt> ps_show 1
Vin             200 V
Vout            12 V
...
Load Sharing    false
```

6.3.5 Fan Monitoring

Use **fan_show** to check the main chassis fans:

```
bf-sde.bf_pltfm.chss_mgmt> fan_show
Fan tray present
FAN  FRONT RPM  REAR RPM  MAX SPEED%
 1       6150       3900        32%
...
 10       16800       16500        100%
```

6.4 QSFP Management

QSFP management is provided in the **bf_pltfm.qsfp** context. From the root CLI context, you can type **qsfp** to enter the **qsfp** context. The commands are:

- detect-xver: detect-xver <port>
- get-xver-info: get-xver-info <port>
- dump-info: dump-info <port>
- qsfp-init: qsfp-init
- get-int: get-int
- get-pres: get-pres

For example, to retrieve operating information about QSFP connector 5, you would type:

```
bf-sde.bf_pltfm.qsfp> dump-info 5
Port 5
  ID: 0x11
  Status: 0x07 0x04
...
  Vendor SN: 63222364
  Date Code: 16111
```

7 Firmware and Software Updates

7.1 Main CPU Boot Loader Updates

7.1.1 Installing BF-Wedge-ONIE

If your Barefoot Networks Wedge 100B system did not come pre-installed with the desired ONIE boot loader, follow the steps in this section to install or re-install ONIE on the Wedge 100B main CPU. To do this, you will boot your Wedge 100B system from a USB drive that contains the ONIE image. This procedure applies to the custom BF-Wedge ONIE boot loader image, available from Barefoot Networks.

> **What is ONIE?** The Open Network Install Environment (ONIE) is a small operating system for bare metal network switches that provides an environment for automatic provisioning. ONIE locates and executes a network operating system (NOS) vendor's installation program. In this case, that NOS is a version of Open Network Linux (ONL) compatible with the Wedge 100B and Tofino.
>
> When an ONIE-enabled network switch boots for the first time, the boot loader launches the kernel, which starts the ONIE discovery and execution (ODE) application.
>
> The ODE uses methods such as scanning for local files and DHCP to locate and download (via HTTP or TFTP) an operating system installer. Once the installer is found, the ODE executes the installer, which then proceeds to provision the box in a NOS-specific way.

To install ONIE, follow both procedures below.

7.1.1.1 Procedure: Create the bootable ONIE image

1. Download the ONIE image from the Barefoot Networks support portal, or get it from your Barefoot Networks account representative. As of this writing, the file names are:

```
Wedge 100BF-65 (65 port): onie-recovery-x86_64-accton_wedge100bf_65x-r0.iso
Wedge 100BF-32 (32 port): onie-recovery-x86_64-accton_wedge100bf_32x-r0.iso
```

2. Convert the ISO into a bootable image and load it onto a USB device. Use the commands below to create the bootable USB. See
   - Steps for Linux (Ubuntu) below, or
   - Steps for MacOS on page 38.

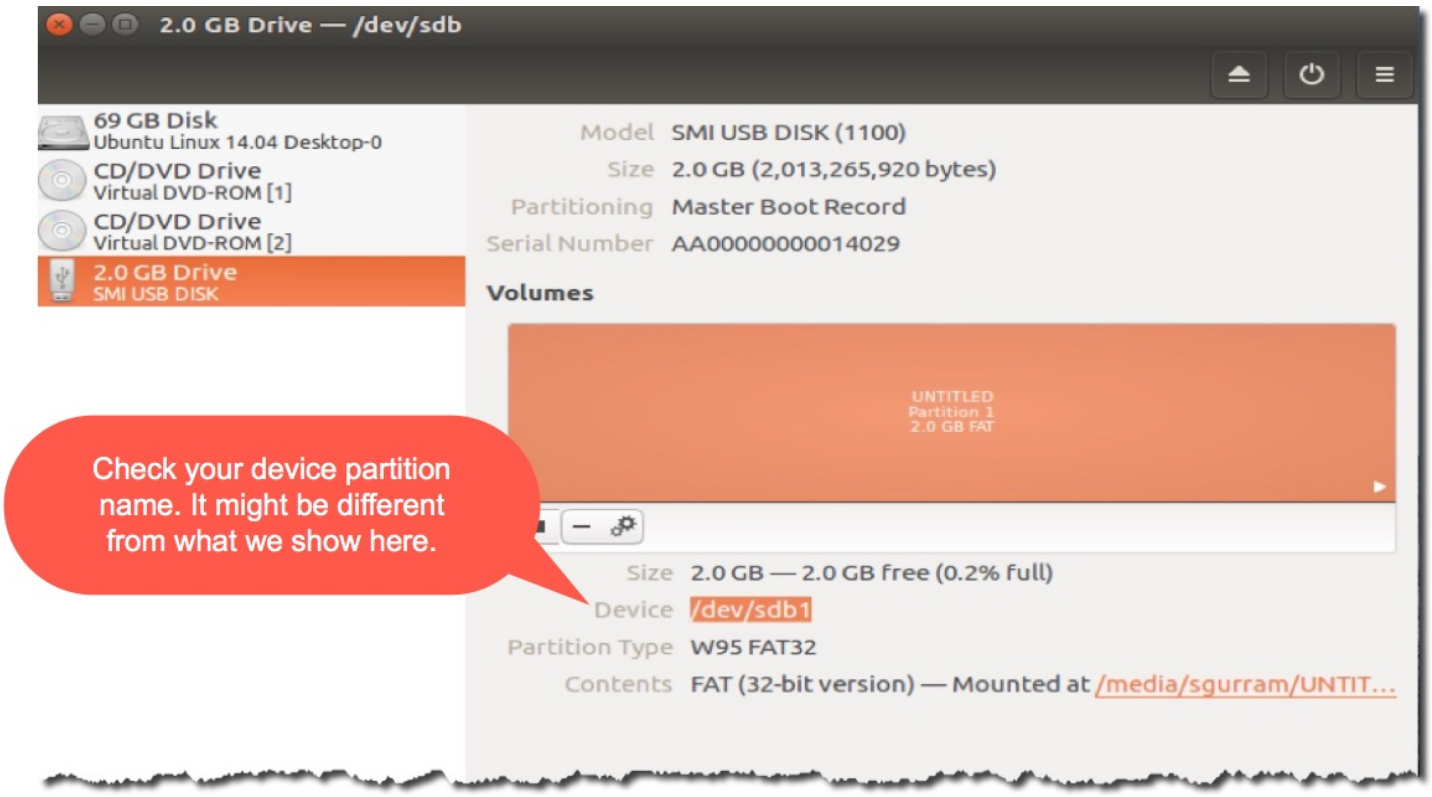
7.1.1.1.1 Steps for Linux (Ubuntu)

On Ubuntu, you can convert the ISO file into a bootable image as follows:

1. Find the partition on which your USB drive is mounted. In this example, it is **`/dev/sdb1`**. You can use a tool such as `gnome-disks` to find the name. Click on the USB drive icon and look for the name in the **Device** field.

> **Note!** On your system, the device might have a different name than **`/dev/sdb1`**. On your system, the device might be called **`/dev/sdb`** or **`/dev/sdc1`**.

```
> gnome-disks
```



2. Create the bootable image using the **dd** command as shown in the example below. Here we use **/dev/sdb1** as our example partition name. In the command below, you must replace **/dev/sdb1** with the name of your USB device partition:

```
sudo dd if=./onie-recovery-x86_64-accton_wedge100bf_65x-r0.iso of=/dev/sdb1
```

3. Proceed to “Boot the Main CPU from the ONIE image” on page 39.

7.1.1.1.2 Steps for MacOS

On Mac OS, you can convert the ISO file into a bootable image as follows:

1. Convert the ISO:

```
> hdiutil convert -format UDRW -o onie-recovery-x86_64-accton_wedge100bf_65x-r0.img onie-
recovery-x86_64-accton_wedge100bf_65x-r0.iso
```

2. Find the partition on which your USB drive is mounted. In this example, **/dev/disk2** this corresponds to the USB device. Use the **diskutil list** command as shown here:

```
> diskutil list
/dev/disk0
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *251.0 GB   disk0
   1:                        EFI EFI                     209.7 MB   disk0s1
...
/dev/disk1
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:                  Apple_HFS Macintosh HD           *249.8 GB   disk1
...
/dev/disk2
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:     FDisk_partition_scheme                        *2.0 GB     disk2
   1:                       0xEF                         213.0 KB   disk2s1
```

> **Note!** Before you run the **dd** command, make sure your USB drive has been unmounted but remains physically connected. The unmount command will be similar to "**sudo diskutil unmount /dev/disk2s1**".

3. Place the image on the USB drive using the **dd** command, as shown in the example below. This example assumes a Wedge 100BF-65:

```
sudo dd if=./onie-recovery-x86_64-accton_wedge100bf_65x-r0.img.dmg of=/dev/disk2 bs=1m
```

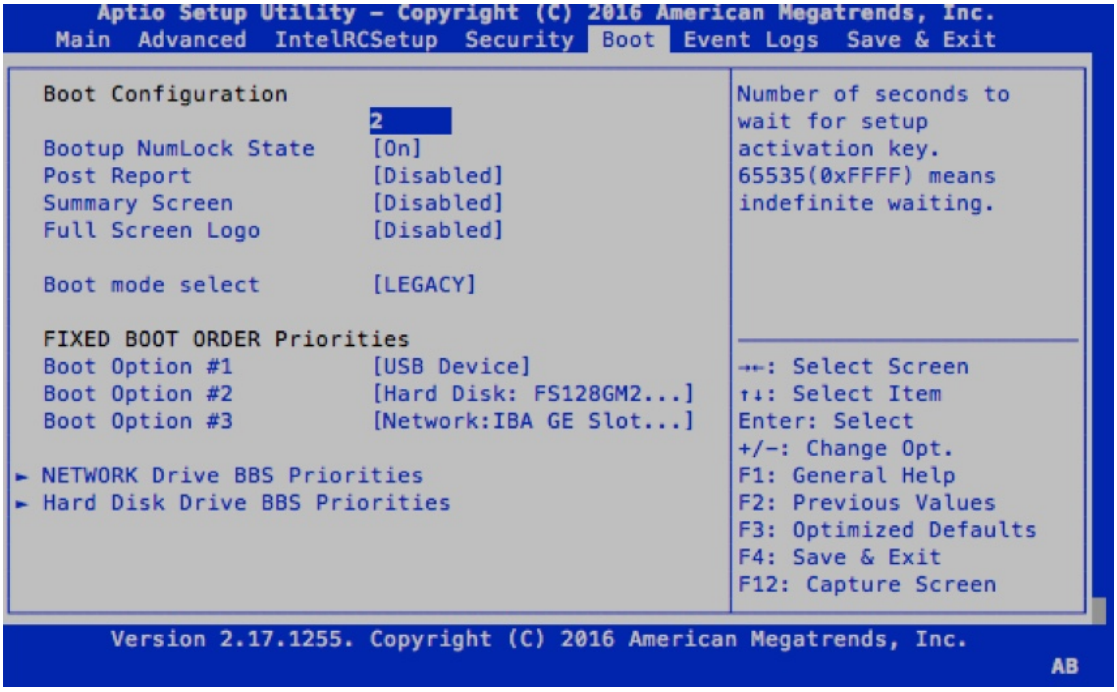
4. Proceed to "Boot the Main CPU from the ONIE image" below.

7.1.1.2 Boot the Main CPU from the ONIE image

1. Connect a console session to the BMC CPU on the Barefoot Wedge 100B. For instructions, see the section, "Accessing the BMC CPU and main CPU through the serial console port."
2. Insert your bootable USB drive into the USB port on the Wedge 100B front panel.
3. Reboot the Wedge 100B system. At the BMC prompt, type the following. After the restart, you must run the **sol.sh** script, which starts a terminal session on the Wedge 100B main CPU:

```
root@bmc:~# wedge_power.sh reset; sol.sh
Power reset microserver ... Done
```

4. Press the <ESC> or <DEL> key to enter the Aptio Setup Utility. In the **Boot** menu, set **Boot mode select** to *LEGACY* and set **Boot Option #1** to *USB Device*.



5. Press F4 to save and exit. (If you're logged in from a MacOS machine using default settings, you may need to press fn + F4).
6. When you reach the ONIE boot screen under GRUB, select *ONIE EMBED MODE*, which installs ONIE.
7. When the ONIE installation is done, remove the USB device before the switch reboots.
8. When the ONIE prompt appears (user is **root**, no password is required), type the following:

```
ONIE:/ # onie-discovery-stop
Stopping: discover… done.
```

9. Verify that the ONIE installation was successful by checking the ONIE log file:

```
ONIE:/ # tail /var/log/onie.log
```

Next steps: Install a Barefoot-specific build of the ONL network operating system as explained in "Main CPU Operating System Updates," below. As an alternative, you can install a third-party network protocol stack and toolchain. Contact Barefoot Networks customer support if you wish to install a third-party toolchain.

7.2 Main CPU Operating System Updates

7.2.1 Installing ONL

In this section, we show how to install a Wedge 100B-compatible build of the ONL operating system on the Wedge 100B main CPU. This is the recommended operating system for the Wedge 100B main CPU for exercising the Capilano SDE. If you're using a third-party network protocol stack and toolchain on the Wedge 100B, you should instead use the operating system recommended by the protocol stack vendor.

Prerequisite: Your Wedge 100B main CPU must be running the ONIE boot loader. See Installing BF-Wedge-ONIE on page 37.

Procedure: After the ONIE boot loader has been installed and the Wedge 100B has booted up, you can install the desired Open Network Linux (ONL) image by following the instructions below. In this case, we show the steps required to install the Barefoot ONL (Open Network Linux) operating system distribution.

1. Download the ONL installer from the Barefoot Networks support portal, or get it from your Barefoot Networks account representative. At the time of this writing, the ONL installer file is called **`ONL-2.0.0_ONL-OS_2017-05-10.2057-b7cf361_AMD64_INSTALLED_INSTALLER`**. Please consult the support portal to get the latest image.
2. From the ONIE prompt, copy the ONL installer file to the **`/root`** folder on the Wedge 100B system hard drive as user *root*. Do this by copying the file over the network using **`scp`** or by mounting a USB device that contains the file and copying it from the USB.

> **How to mount a USB Drive on a Wedge 100BF system where ONIE is installed:**
>
> From the ONIE prompt, do the following:
>
> ```
> ONIE:/ # fdisk –l
> ...
> ONIE:/ # mkdir /mnt/usb
> ONIE:/ # mount /dev/sdb1 /mnt/usb
> ONIE:/ # ls –l /mnt/usb
> ...
> ```
>
> The above sequence of ONIE commands assumes "/dev/sdb1" is the USB drive's filesystem, which you should confirm from the output of running the "fdisk –l" command. Once you have copied the necessary files from the USB device to the Wedge 100BF system hard drive, you can unmount the USB device by issuing the **`umount`** command:
>
> ```
> ONIE:/ # umount /mnt/usb
> ```

3. Install ONL by running the **`onie-nos-install`** command:

```
ONIE:~ # onie-nos-install ONL-2.0.0_ONL-OS_2017-05-10.2057-b7cf361_AMD64_INSTALLED_INSTALLER
```

   The same ONL installer file works on both the Mavericks (Wedge 100BF-65) and Montara (Wedge 100BF-32) boxes.

4. Once the system comes up after the ONL installation and the subsequent reboot, you can log in to the Wedge 100B main CPU with username/password: **`root/onl`**.

Next steps:

- If you will use Barefoot Networks' Capilano SDE, *you must install the SDE dependencies*, as shown in the section, "Installing SDE Dependencies" on page 16.
- If you will use a switch management/development environment from another vendor, please consult that vendor's documentation.

7.3 BMC CPU Operating System Updates

This section explains how to update the operating system of the baseboard management controller CPU (BMC CPU) on the Wedge 100B.

7.3.1 BMC Image Architecture

Barefoot Networks' Wedge 100B BMC image has four partitions. `uboot`, `uboot environment`, `kernel`, and the `root` filesystem. Only the `kernel` and `root` file systems can be field updated from the command line. The `uboot` and `uboot environment` file systems are not expected to require updates. It is possible to update the `kernel` or `root` file system independently, but we do not recommend doing this. Instead, Barefoot Networks recommends that you update both file systems before you reboot the BMC CPU, unless the Wedge 100B / SDE Release Notes explicitly state otherwise.

7.3.2 Procedure: BMC Update

Follow these steps to update the Barefoot Networks' Wedge 100B BMC image:

1. Download the OpenBMC image from the Barefoot Networks support portal. The file name will be similar to **`bf-BMC05092017.tar`**.
2. Unpack the archive. The files you need are:
   - o **`mavericks-image-mavericks.cpio.lzma.u-boot`**
   - o **`uImage`**
5. Transfer the kernel (**`uImage`**) and rootfs (**`mavericks-image-mavericks.cpio.lzma.u-boot`**, or, **`rootfs`**, for short) to the **`/tmp`** folder of the BMC filesystem. First, give the rootfs file its shorter name. For the rest of this procedure, we refer to the files as **`uImage`** and **`rootfs`**, respectively:

```
mv mavericks-image-mavericks.cpio.lzma.u-boot rootfs
```

Then place the files in the **`/tmp`** folder of the BMC filesystem

- o If your BMC CPU is reachable on the network:

```
scp uImage rootfs root@<bmc-ip-address>:/tmp/.
```

- o If your BMC CPU is not on a reachable network, then transfer the files first to the main CPU (over the network or via USB on the Wedge 100B front panel USB port) and then to the BMC. From a terminal session on the main CPU, the copy operation will be as follows. By default, the BMC CPU is reachable locally (on-box) at the address 192.168.0.1.

```
admin12@bfn-switch:~$scp uImage rootfs root@192.168.0.1:/tmp/.
```

6. Connect to the BMC shell over a serial telnet or scp connection.

```
root@bmc:~# cd /tmp
```

7. Use the `flashcp` command to write the kernel file to **`/dev/mtd2`**. IMPORTANT: Be careful to use the name "**`mtd2`**", as shown in the command below.

```
root@bmc:~# flashcp uImage /dev/mtd2
```

> The operation above takes up to **30 seconds** before returning the prompt.

8. Use the `flashcp` command to write the **`rootfs`** file to **`/dev/mtd3`**. IMPORTANT: Be careful to use the name "**`mtd3`**", as shown in the command below.

```
root@bmc:~# flashcp rootfs /dev/mtd3
```

> The operation above takes up to **two minutes** before returning the prompt.

9. Reboot the Wedge 100B BMC CPU:

```
root@bmc:~#  reboot
```

Some upgrades introduce changes to BMC commands. For BMC command line instructions, see the earlier section, "Accessing the BMC CPU and main CPUs through the serial console port" on page 13.

7.3.3 Checking BMC image version

To show the version of the Wedge 100B BMC image running on your BMC CPU, type the following commands:

```
root@bmc:~# cat /etc/issue
root@bmc:~# cat /etc/version
```

8 Tofino Testing and Troubleshooting

In this section we explain tools for testing and troubleshooting traffic forwarding on Tofino, including the bf-shell CLI troubleshooting and logging commands, as well as tools to generate packets for testing.

8.1 Generating test traffic

8.1.1 Generating packets with the Diags test dataplane

Many examples shown in this section are based on the **`diag.p4`** program, known as Diags. For more information on Diags, see the *Barefoot Diags CLI and API User Guide*. The CLI commands shown here are available in other P4 dataplane programs as well.

Tip! A fast way to create sample match-action tables and generate traffic is to use Barefoot's **`diag.p4`** demonstration program, known as "Diags." To set up the match-action tables shown in some examples below, first run Diags:

```
root@localhost:~/doc/bf-sde-5.0.0.11# . ./set_sde.bash
```

```
root@localhost:~/doc/bf-sde-5.0.0.11# ./run_switchd.sh -p diag
```

Next, add ports. For example:

```
bf-sde> pm
```

```
bf-sde.bf_pltfm.pm> port-add -/- 100G NONE
```

```
bf-sde.bf_pltfm.pm> port-enb -/-
```

Check that ports are up (check the QSFP, ADM, OPR columns):

```
bf-sde.bf_pltfm.pm> show
```

Check the status of the ports:

```
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX|FRAMES TX|E
 1/0|31/0|188|1/60| 100G |NONE|YES |ENB|UP |        0|        0|
 2/0|30/0|184|1/56| 100G |NONE|YES |ENB|UP |        0|        0|
```

Go to the Diags context and set up a VLAN:

```
bf-sde> diags
```

```
bf-sde.diags> vlan-create 0 1
```

Set the VLAN as the default VLAN for some ports:

```
bf-sde.diags> def-vlan-set 0 188 1
```

```
bf-sde.diags> def-vlan-set 0 184 1
```

You can test traffic forwarding by running a snake test in Diags using the `snake-setup`, `snake-start`, `snake-status`, `snake-stop`, and `snake-cleanup` commands. See the *Barefoot Diags CLI and API User Guide* for details.

8.1.2 Injecting Packets from the CPU

For testing, you can use a CPU port to inject packets from the control plane. Typically, you will use the PCIe port (port number 320 on a Tofino 4-pipe device (as in the Wedge 100BF-65X), and port number 192 on a 2-pipe device (as in the Wedge 100BF-32X)). On some systems, port 64 might also be connected to the CPU via a dedicated Ethernet MAC.

From P4 programming standpoint, these are regular ports, meaning that packets go through regular Tofino data plane processing. For testing, if you want such a packet to go directly to a certain port, you must design a special header that will carry the desired port number, add it to the packet and then process the packet in your P4 program accordingly (probably removing that special header before sending the packet to egress processing, among other things).

There are two options for injection:

- You can use user-level APIs, defined in **`$SDE_INSTALL/include/pkt_mgr/pkt_mgr_intf.h`** and in **`bf_pkt.h`**. In case of Thrift, the SDE provides some simplified APIs as well. See **`$SDE_INSTALL/lib/python2.7/site-packages/tofino/pkt_pd_rpc/pkt.py`** for details.
- You can use the *bf_kpkt* driver. This driver creates a Linux network interface on opt of the PCIe port. Using this, you can open a raw socket, use *libpcap* or any other standard Linux method of your choosing to send/receive packets to/from CPU PCIe port.

8.1.3 Generating Packets in Tofino with pktgen

Each pipe on Tofino has its own packet generator engine capable of injecting batches of packets based on user-specified trigger events. For information, see the **`p4_pd_pktgen`** APIs defined in **`$SDE_INSTALL/include/tofino/pdfixed/pd_conn_mgr.h`** and the **`p4_pd_pktgen_app_cfg`** struct in **`$SDE_INSTALL/include/tofino/pdfixed/pd_common.h`**.

8.2 Log file locations

When you run a P4 program, the program's log is written to a file in the directory where the program executed. The name of the log file depends on which script you used to run your P4 program:

- run_switchd.sh: **`bf_drivers.log`**
- run_p4_tests.sh: **`ptf.log`**
- run_tofino_model.sh: **`model_0.log`**

8.3 Dumping match-action table entries

For a given match-action table on Tofino, you can dump the match spec, action and action data as follows:

8.3.1 List all match-action tables

To list all the tables in use by the P4 program, use the **`pipe_mgr`** table dump command, `tbl -d <dev>` (where <dev> is the switch device ID, usually "0" on a single-ASIC system):

```
bf-sde> pipe_mgr
bf-sde.pipe_mgr> tbl -d 0

Match-Action Tables:
  Match-Action Tables:
  ------------------------|----------|----|--------|-----|----|---|---|---|---|----
  Name                    |Handle    |Type|Entries |Keysz|Stgs|Adt|Sel|Sta|Met|Sful
  ------------------------|----------|----|--------|-----|----|---|---|---|---|----
  tbl_stage0              |0x1000001 |exm |2       |0    |1   |NA |NA |NA |NA |NA
  ...
  def_vlan_mapping        |0x100000e |exm |512     |0    |1   |NA |NA |NA |NA |NA
  ...
```

In the table list output, pay attention to the values in:

- the Handle column. This is the table's unique identifier; and
- the Type column. This indicates the type of table: exm for an exact-match table, or tcam for a ternary-match table.

8.3.2 Show information about a match-action table

To dump a match action table, use the table info command, **`tbl_info`**, in one of the table-manager contexts:

- For exact-match tables (**`exm`**). use the **`exm_tbl_mgr`** context.
- For ternary-match tables (**`tcam`**). use the **`tcam_tbl`** context.

Command summary: `tbl_info -d <dev_id> -h <tbl_hdl>`

where

- <dev> is the switch device ID, usually "0" on a single-ASIC system
- <tbl_hdl> is the handle that identifies the table

For example:

```
bf-sde.pipe_mgr> exm_tbl_mgr
bf-sde.pipe_mgr.exm_tbl_mgr> tbl_info -d 0 -h 0x100000e
  ------------------------------------------------
  Exact match table info for table 0x100000e device 0
  ------------------------------------------------
  Size : 512, Number occupied : 4
  Symmetric : true
  ...
  Number of stages : 1
  Stages : 1,
  --------------------------
  Stage 1 info :
  --------------------------
    Num entries : 1024, Num Occupied : 4
    Packing format :
        Number of entries per wide-word : 1
  ...
  Entries :
  Default entry hdl for ALL PIPES : 1
  Entry Hdl |Stage|Pipe      |Mem-id(s) |Line-no
  ----------|-----|----------|----------|----------
  0x2       |    1|ALL PIPES |67,       |  188
  0x3       |    1|ALL PIPES |67,       |  184
  ...
```

In the dump of a table, the **Entry_Hdl** column provides each entry's handle, which you can use to look up information about that table entry.

8.3.3 Show information about a match-action table entry

To dump a match-action table entry, use the entry info command, **entry_info**, in the **exm_tbl_mgr** context:

Command summary: `entry_info -d <dev_id> -h <tbl_hdl> -e <entr_hdl>`

where

- <dev id> is the switch device ID, usually "0" on a single-ASIC system
- <tbl_hdl> is the handle that identifies the table
- <entr_hdl> is the handle that identifies the entry

For example:

```
bf-sde.pipe_mgr.exm_tbl_mgr> entry_info -d 0 -h 0x100000e -e 0x2
  Info for entry handle 2 for exact match table 16777230
      Match Spec :
      -----------------
      ig_intr_md_ingress_port :
            Value: 00  bc
            Mask: ff  ff

      Action Spec:
      -----------------
      Action function :set_def_vlan
      vid :
            Value: 00  01
      ingress_rid :
            Value: 00  01

      Action function handle : 0x2000002e
      Entry location info:
            Pipe id : ALL PIPES
            Stage id : 1
            Logical entry index : 188
            Ram line num : 188
```

8.4 Troubleshooting Packet Drops

To show drop cause and counters about packet drops, including packet drops at different phases (such as ingress pipeline, egress pipeline and traffic manager), use the commands of the **`pkt_path_counter`** facility. See “Port and Pipeline Counters,” below. For traffic manager (TM) packet drop counters, use the commands of the **`bf-sde.traffic_mgr.counter`** facility. See “Traffic Manager Counter Commands,” below.

8.5 Checking memory register values

When troubleshooting traffic on Tofino, you typically do not need to inspect register values if you are using P4 and the Capilano SDE APIs to program your data plane. The compiler and SDE handle register settings so they remain transparent to customers.

You can test Tofino memory modules and registers using the Diags memory tests. For instructions, see the section, "Diags Fixed Memory/Register Tests" in the *Barefoot Diags CLI and API User Guide*.

8.6 Packet Snapshots

The packet snapshot (**`pipe_mgr`**: **`snap`**) feature of Tofino allows you to trace the packet processing path through the pipeline from the incoming interface through the tables and triggered actions to the outgoing interface. This command captures nearly all the metadata and lookup results for next captured packet.

Here’s how it works: In a given stage, you can specify a ternary match for a packet header vector (PHV) that will trigger a snapshot.

> **What is a PHV?** A packet header vector, or "PHV," is a sequence of 8- 16- and 32-bit fields that store packet header data and metadata. Tofino's parser analyzes a packet and maps it into a corresponding set of fields, called a PHV. The PHV comprises a set of header fields such as IP destination, Ethernet destination, and so on. In addition, the packet header vector includes metadata fields such as the input port on which the packet arrived and other router state variables (for example, the current size of router queues). In Tofino, the PHV flows through a sequence of logical match stages (the match-action units (MAUs) of the match-action pipeline), each of which abstracts a logical unit of packet processing (for example, Ethernet or IP processing). After the PHV has been operated on by the MAUs in the pipeline, Tofino's deparser reassembles the modified packet from the PHV.

For example, you can define a ternary match like “trigger a snapshot in stage 5 if ipv4.srcAddr matches 0xC0A80100 mask 0xFFFFFF00” (192.168.1.0/24 that is).

Once such a condition occurs, a snapshot will be taken in that stage and in as many subsequent stages as you want (you can specify that). It will capture the PHV at the requested stages, so you can really observe how the metadata was changing while the packet travels through the chip. In addition to that it will also capture information about which tables were hit, and so on.

You can set multiple, *non-intersecting* snapshots at the same time if you like, e.g.

- In the ingress pipeline 0: trigger in stage 0, collect snapshots till stage 2
- In the ingress pipeline 0: trigger in stage 3, collect snapshots till stage 11
- In the egress pipeline 0: trigger in stage, collect snapshots till stage 11

Once a snapshot is taken subsequent packets will not be able to change it even if they match the same trigger condition. You will need to "re-arm" the snapshot to capture more.

8.6.1 Snapshot Commands

There are both diag shell (CLI) commands available in switchd and APIs (some of them fixed and some program-dependent) that enable you to use this facility.

When using the CLI, the snapshot-related commands are in the **`pipe_mgr`** section:

```
bf-sde> pipe_mgr
bf-sde.pipe_mgr> snap
snap-create
snap-delete
snap-trig-add
snap-trig-clr
snap-state-set
snap-timer-en
snap-intr-clr
snap-hdl-dump
snap-cfg-dump
snap-state-get
snap-capture-get
```

8.6.2 Creating a snapshot trigger

You can create and arm a snapshot trigger as follows:

1. Create a snapshot and get a handle for it. In this case. we create a snapshot on device 0 (`-d 0`), pipe 0 (`-p 0`), stages 0 through 5 (`-s 0 -e 5`), in the ingress pipeline (set it to `-i 0` for ingress, or to `-i 1` for egress):

```
bf-sde.pipe_mgr> snap-create -d 0 -p 0 -s 0 -e 5 -i 0
Snapshot created with handle 0x141
```

Important! You can set a snapshot on ingress or egress, but not both at the same time.

2. Specify the trigger conditions. Multiple conditions will be AND-ed

```
bf-sde.pipe_mgr> snap-trig-add -h 0x141 -n ipv4_srcAddr -v 0xc0a80000 -m 0xffff0000

Trigger: Adding Field ipv4_srcAddr, value 0xc0a80000, mask 0xffff0000
Success in adding field ipv4_srcAddr to trigger
```

3. Enable (arm) the snapshot.

```
bf-sde.pipe_mgr> snap-state-set -h 0x141 -e 1
Snapshot state set to 1
```

4. Monitor the snapshot state until it hits (or, if you know what you are sending into the switch, just confirm)

```
bf-sde.pipe_mgr> snap-state-get -h 0x141

Snapshot state dump for handle 0x141
Dumping snapshot state for dev 0, pipe 0, start-stage 0, end-stage 5, dir ingress
--- Dumping snapshot state for stage 0 ---
Interrupt state : Set
Admin State : Disabled
FSM state : Full
Timer state : Disabled
Current time : 0x285123ea0 clocks
Trigger time : 0x0 clocks

--- Dumping snapshot state for stage 1 ---
Interrupt state : Set
Admin State : Disabled
FSM state : Full
Timer state : Disabled
Current time : 0x253aa23960 clocks
Trigger time : 0x0 clocks
...
```

8.6.3 Viewing a snapshot

If the interrupt state is "Set" then you can view the capture. The capture can be viewed for a single stage or for all stages that it was created for. Each stage is captured separately. The capture contains the values of all the fields after the stage has executed, the summary of all tables (including the addresses of the entries that hit, but they are NOT entryIDs) and the Next-Table pointer. Note also, that since PHV containers are overlaid, you will see many fields that you might think make no sense (e.g. ipv6 fields in IPv4 packet). Ignore those.

```
bf-sde.pipe_mgr> snap-capture-get -h 0x141

Snapshot capture for handle 0x141
Dumping snapshot capture for dev 0, pipe 0, start-stage 0, end-stage 5, dir ingress
-------------- Snapshot Capture for stage 0, direction Ingress ---------------
Snapshot trigger type:
Prev stage : No
Local stage : Yes
Timer : No
------------------- Captured Fields -------------------------------------------
Field name Value
-------------------------------------------------------------------------------
l2_metadata_lkp_mac_sa 00 00 00 00 00 00
multicast_metadata_multicast_route_mc_index 0x0
tunnel_metadata_ingress_tunnel_type 0x0
l3_metadata_l3_copy 0x0
l2_metadata_bd_stats_idx 0x0
acl_metadata_if_label 0x0
multicast_metadata_ipv4_multicast_enabled 0x0
ipv4_fragOffset 0x0
multicast_metadata_outer_mcast_route_hit 0x0
ipv6___srcAddr 00 69 00 00 00 00 00 00 00 00 00 00 c0 a8 00 01
l3_metadata_lkp_l4_sport 0x0
l3_metadata_rmac_group 0x0
ipv6_version 0x40
inner_ipv4_ihl 0x40
hash_metadata_entropy_hash 0x0
mpls_2__bos 0x0
l2_metadata_learning_enabled 0x0
...
```

Table info:

```
---------------------- Table info ------------------------------------------
-------------------------------------------------------------------------------
Table Name Type Hit Hit-Addr Inhibited Executed
_condition_48 Exact No 0x0 No No
ingress_port_properties Exact Yes 0x2 No Yes
mirror Tcam No 0x0 No No
rid Exact No 0x0 No No
validate_outer_ethernet Tcam Yes 0x1c8 No Yes
validate_outer_ipv4_packet Tcam Yes 0x1f0 No Yes
validate_outer_ipv6_packet Tcam Yes 0x0 Yes No
validate_mpls_packet Tcam Yes 0x0 Yes No
switch_config_params Tcam No 0x0 No Yes
port_vlan_mapping Exact Yes 0x4d No Yes
-------------------------------------------------------------------------------
Next Table: spanning_tree
...
```

8.7 Port and Pipeline Counters

8.7.1 Dumping Port (MAC) counters

Using the commands shown below, you can print a quick summary of MAC receive and MAC transmit counters for a port. The column `'E'` indicates whether non-zero error counters are observed

```
bf-sde.bf_pltfm.pm> show -p 1/0
----+----+---+----+------+----+----+---+---+----------------+----------------+-
PORT|MAC |D_P|P/PT|SPEED |FEC |QSFP|ADM|OPR|FRAMES RX       |FRAMES TX       |E
----+----+---+----+------+----+----+---+---+----------------+----------------+-
 1/0|31/0|188|1/60| 10G  |NONE| NO |DIS|DWN|               0|               0|
```

A detailed view of the MAC counters of a port can be dumped as follows.

```
bf-sde.bf_pltfm.pm> show -p 1/0 -d
```

8.7.2 Pipeline Traffic Counters

Traffic counters at various points in the pipeline can be dumped using the CLI commands of the **`pkt_path_counter`** facility.

```
bf-sde> pkt_path_counter
bf-sde.pipe_mgr.pkt_path_counter>
```

A summary of the pipeline counters can be printed using the **`pipe_cnt`** command:

```
bf-sde.pipe_mgr.pkt_path_counter> pipe_cnt -p <pipe>
```

The commands in this facility accept the following option flags, as specified in each command description:

- -d <devid>
- -p <pipe>
- -m <port>
- -z <print zero counters too>

The counter retrieval commands in **`pkt_path_counter`** are:

- ibuf - Prints ibuf counters. Accepts flags: -d, -p, -m
- iprsr - Prints ingress parser counters. Accepts flags: -d, -p, -m
- idprsr - Prints ingress deparser counters. Accepts flags: -d, -p
- eprsr - Prints egress parser counters. Accepts flags: -d, -p, -m
- epb_ebuf - Prints egress parser buffer counters. Accepts flags: -d, -p, -m
- edprsr - Prints egress deparser counters. Accepts flags: -d, -p
- pipe_cnt - Prints ingress and egress deparser counters. Accepts flags: -d, -p
- device - Prints all device-level counters. Accepts flags: -d, -z
- pipe - Prints all pipe-level counters. Accepts flags: -d, -z
- port - Prints all port-level counters in a pipe. Accepts flags: -d, -p, -m, -z

Counter clear commands in pkt_path_counter:

- clear_pipe - Clears ingress and egress deparser counters. Accepts flags: -d, -p
- clear_ibuf - Clears ibuf counters. Accepts flags: -d, -p, -m
- clear_iprsr - Clears ingress parser counters. Accepts flags: -d, -p, -m

- clear_idprsr - Clears ingress deparser counters. Accepts flags: -d, -p
- clear_eprsr - Clears egress parser counters. Accepts flags: -d, -p, -m
- clear_edprsr - Clears egress deparser counters. Accepts flags: -d, -p
- clear_epb_ebuf - Clears egress parser buffer counters. Accepts flags: -d, -p, -m
- clear_all_pipe - Clears all port counters in data path. Accepts flags: -d, -p

Parser commands in pkt_path_counter:

- intr_status - Interrupts of parser-deparser logic. Accepts flags: -d, -p, -m
- deparser_intr_status - Interrupts of parser-deparser logic. Accepts flags: -d, -p

8.8 Traffic-Manager (TM) and Traffic Counters

The traffic manager (TM) troubleshooting commands and counters allow you to read TM status, queue and port pass/drop count, XON/XOFF status, and other useful traffic management information. These commands are provided in the **`traffic_mgr`** package.

```
bf-sde> traffic_mgr
bf-sde.traffic_mgr>  counter
bf-sde.traffic_mgr.counter> blocklevel -z
```

The traffic manager command subsections and commands:

- log
- cfg_table
- usage
- watermark
- dropstatus
- dl_dropstatus
- pfcstatus
- counter
- clr_counter
- ut_hitless_cfg
- mirror

8.8.1 Traffic Manager cfg_table Commands

These commands appear in the CLI section, `bf-sde.traffic_mgr.cfg_table`. They return information about the packet buffer management settings of Tofino. See the Tofino *Switch Architecture Specification* for details about these settings.

8.8.1.1 ppg_min Command

Shows PFC Priority Group (PPG) minimum limits used for Priority-based Flow Control (PFC) traffic.

Usage:

```
ppg_min -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.2 ppg_hdr Command

Shows PPG headroom limits.

Usage:

```
ppg_hdr -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.3 port_limit Command

Shows write admission control (WAC) Max Port Limits of the Tofino packet data buffer.

Usage:

```
port_limit -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.4 ppg_resume_limit Command

Shows PPG Resume Limits.

Usage:

```
ppg_resume_limit -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.5 port_ppg Command

Shows Port to PPG Mapping Table

Usage:

```
port_ppg -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.6 ppg_shared_limit Command

Shows PPG Shared Limits

Usage:

```
ppg_shared_limit -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.7 ppg_icos_map Command

Shows PPG iCOS Mapping

Usage:

```
ppg_icos_map -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.8 ppg_offset_prof Command

Shows PPG Offset Profile

Usage:

```
ppg_offset_prof -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.9 q_min_thrd Command

Q Min Limit

Usage:

```
q_min_thrd -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.10 q_shr_thrd Command

Q Shared Limit

Usage:

```
q_shr_thrd -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.11 q_ap Command

Q App pool Mapping

Usage:

```
q_ap -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.12 q_color_limit Command

Q Color Limit

Usage:

```
q_color_limit -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.13 qac_port_limit Command

QAC Port Limit

Usage:

```
qac_port_limit -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.14 qac_qid_profile Command

QAC Q Profile ID

Usage:

```
qac_qid_profile -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.15 qac_qid_map Command

QAC Q Mapping

Usage:

```
qac_qid_map -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.16 q_min_shaper Command

SCH Q Min Rate

Usage:

```
q_min_shaper -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.17 q_max_shaper Command

SCH Q Max Rate

Usage:

```
q_max_shaper -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.18 Port_max_shaper Command

SCH Port Max Rate

Usage:

```
Port_max_shaper -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.1.19 wac_eg_qid_map Command

WAC Egress QID mapping

Usage:

```
wac_eg_qid_map -d <devid>, -X [print in hex] -z [print zero counters too]
```

8.8.2 Traffic Manager cfg Commands

CLI section: bf-sde.traffic_mgr.cfg

8.8.2.1 ppg Command

PPG Details

Usage:

```
ppg -d <devid> -p <pipe> -r <ppg#/q#/port#> -X [print in hex] -z [print zero counters too]
```

8.8.2.2 q Command

Queue Details

Usage:

```
q -d <devid> -p <pipe> -r <ppg#/q#/port#> -X [print in hex] -z [print zero counters too]
```

8.8.2.3 port Command

Port Details

Usage:

```
port -d <devid> -p <pipe> -r <ppg#/q#/port#> -X [print in hex] -z [print zero counters too]
```

8.8.3 Traffic Manager usage Commands

CLI section: bf-sde.traffic_mgr.usage

8.8.3.1 ppg_gmin_usage Command

PPG cell usage in Gmin Pool

Usage:

```
ppg_gmin_usage -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.3.2 ppg_shrd_usage Command

PPG cell usage in shared Pool

Usage:

```
ppg_shrd_usage -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.3.3 ppg_skid_usage Command

PPG cell usage in Skid pool

Usage:

```
ppg_skid_usage -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.3.4 wac_portusage Command

Port cell usage in wac

Usage:

```
wac_portusage -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.3.5 qac_portusage Command

Port cell usage in QAC

Usage:

```
qac_portusage -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.3.6 qac_q_usage Command

Queue cell usage in QAC

Usage:

```
qac_q_usage -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.4 Traffic Manager watermark Commands

CLI section: bf-sde.traffic_mgr.watermark

8.8.4.1 ppg_wm Command

PPG usage watermark

Usage:

```
ppg_wm -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.4.2 wac_port_wm Command

Port usage watermark in WAC

Usage:

```
wac_port_wm -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.4.3 q_wm Command

Q usage watermark

Usage:

```
q_wm -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.4.4 qac_port_wm Command

Port usage watermark in QAC

Usage:

```
qac_port_wm -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5 Traffic Manager dropstatus Commands

CLI section: bf-sde.traffic_mgr.dropstatus

8.8.5.1 ppg0_31 Command

Drop status of PPG 0 to 31

Usage:

```
ppg0_31 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.2 ppg32_63 Command

Drop status of PPG 32 to 63

Usage:

```
ppg32_63 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.3 ppg64_95 Command

Drop status of PPG 64 to 95

Usage:

```
ppg64_95 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.4 ppg96_127 Command

Drop status of PPG 96 to 127

Usage:

```
ppg96_127 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.5 ppg128_159 Command

Drop status of PPG 128 to 159

Usage:

```
ppg128_159 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.6 ppg160_191 Command

Drop status of PPG 160 to 191

Usage:

```
ppg160_191 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.7 ppg192_200 Command

Drop status of PPG 192 to 200

Usage:

```
ppg192_200 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.8 port0_31 Command

Drop status of Port 0 to 31 in Wac

Usage:

```
port0_31 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.9 port32_63 Command

Drop status of Port 32 to 63 in Wac

Usage:

```
port32_63 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.10 port64_72 Command

Drop status of Port 64 to 71 in Wac

Usage:

```
port64_72 -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.11 q_green_drop Command

QAC Q Drop State(Green)

Usage:

```
q_green_drop -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.12 q_yellow_drop Command

QAC Q Drop State(Yellow)

Usage:

```
q_yellow_drop -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.13 q_red_drop Command

QAC Q Drop State(Red)

Usage:

```
q_red_drop -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.5.14 port_drop_State Command

Port Drop Status in QAC

Usage:

```
port_drop_State -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.6 Traffic Manager dl_dropstatus Commands

CLI section: bf-sde.traffic_mgr.dl_dropstatus

8.8.6.1 colordrop Command

Color Drop State in wac

Usage:

```
colordrop -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.2 skiddrop Command

Skid Pool Drop State in wac

Usage:

```
skiddrop -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.3 wac_qshadowstate Command

Queue Shadow State in wac

Usage:

```
wac_qshadowstate -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.4 ap_drop_state Command

Application Pool Drop status in qac

Usage:

```
ap_drop_state -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.5 apg_green_drop_state Command

Application Pool Drop status of green packets in qac

Usage:

```
apg_green_drop_state -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.6 apy_yel_drop_state Command

Application Pool Drop status of green packets in qac

Usage:

```
apy_yel_drop_state -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.7 apr_red_drop_state Command

Application Pool Drop status of green packets in qac

Usage:

```
apr_red_drop_state -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.8 p0_prefifo_drop_state Command

Pre FIFO Drop status in pipe0

Usage:

```
p0_prefifo_drop_state -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.9 p1_prefifo_drop_state Command

Pre FIFO Drop status in pipe1

Usage:

```
p1_prefifo_drop_state -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.10 p2_prefifo_drop_state Command

Pre FIFO Drop status in pipe2

Usage:

```
p2_prefifo_drop_state -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.6.11 p3_prefifo_drop_state Command

Pre FIFO Drop status in pipe3

Usage:

```
p3_prefifo_drop_state -d <devid>, -X [print in hex] -z [print zero values too]
```

8.8.7 Traffic Manager pfcstatus Commands

CLI section: bf-sde.traffic_mgr.pfcstatus

8.8.7.1 port_pfc Command

Per port pfc State generated in wac

Usage:

```
port_pfc -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.7.2 port_pfc_rx Command

Per port pfc state received in qac

Usage:

```
port_pfc_rx -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.7.3 q_pfc_rx Command

Per q pfc state received in qac

Usage:

```
q_pfc_rx -d <devid> -p <pipe> -X [print in hex] -z [print zero values too]
```

8.8.7.4 log Command

8.8.8 Traffic Manager Counter Commands

CLI section: bf-sde.traffic_mgr.counter

8.8.8.1 blocklevel Command

Dump all block-level counters

Usage:

```
blocklevel -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.8.2 wac_perport Command

Dump per port drop counter as seen in wac

Usage:

```
wac_perport -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.8.3 wac_perppg Command

Dump per ppg drop counter as seen in wac

Usage:

```
wac_perppg -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.8.4 qac_q_drop Command

Dump per queue drop counter as seen in qac

Usage:

```
qac_q_drop -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.8.5 qac_port_drop Command

Dump per port drop counter as seen in qac

Usage:

```
qac_port_drop -d <devid> -p <pipe> -X [print in hex] -z [print zero counters too]
```

8.8.9 Traffic Manager clr_counter Commands

CLI section: bf-sde.traffic_mgr.clr_counter

8.8.9.1 clr_blocklevel Command

Clear block level counters in all blocks

Usage:

```
clr_blocklevel -d <devid> -p <pipe>
```

8.8.9.2 clr_wacperport Command

Clear per port drop counter as seen in wac

Usage:

```
clr_wacperport -d <devid> -p <pipe>
```

8.8.9.3 clr_wacperppg Command

Clear per ppg drop counter as seen in wac

Usage:

```
clr_wacperppg -d <devid> -p <pipe>
```

8.8.9.4 clr_qac_qdrop Command

Clear per queue drop counter as seen in qac

Usage:

```
clr_qac_qdrop -d <devid> -p <pipe>
```

8.8.9.5 clr_qac_portdrop Command

Clear per port drop counter as seen in qac

Usage:

```
clr_qac_portdrop -d <devid> -p <pipe>
```

8.9 Interrupt Monitoring

Tofino interrupts can provide clues towards potential problems that are being encountered. The following sequence of commands can be used to dump newly fired interrupts since last time they were checked.

```
bf-sde> int_poll 0
bf-sde> int_new
000400a0 : 000400a4 : ffffffff : 03000000 : 0x7f8d8c0fdf18 :
__device_select__misc_regs__int_stat
: [12:12] : 1 : sbm0_dbe_err             : Set when the SBus Master 0 report dual bit error
: [15:15] : 1 : sbm1_sbe_err             : Set when the SBus Master 1 report single bit error
```

Barefoot Networks recommends that, upon initialization, you clear interrupts once using the above commands. This ensures you see only newly fired interrupts that happen after initialization.

9 Known Caveats and Workarounds

9.1 SYS-1: Main CPU is not powered up by the time OpenBMC is up

Symptom:

On power-up, the Wedge 100B main CPU does not respond over serial console or over management IP.

Condition:

This is intermittently observed on system power up. This issue is not expected to happen if NOS bootup delays are less than 40 seconds.

Workaround:

Reset the BMC's **pwr_main_n** flag by redirecting "1" and then "0" into it as follows:

```
root@bmc:~#
echo 0 > /sys/class/i2c-adapter/i2c-12/12-0031/pwr_main_n
echo 1 > /sys/class/i2c-adapter/i2c-12/12-0031/pwr_main_n; sol.sh
```

Note: Be sure to add the **sol.sh** command as shown above. By doing this, you ensure the BMC CPU will establish a connection to the main CPU at an appropriate time.

In about 40 seconds, you should expect to see the BIOS start and boot into Ubuntu.

9.2 SYS-2: Main CPU stuck at BIOS screen

Symptom:

On power-up, the main CPU is stuck at the BIOS screen

Condition:

This is intermittently observed on system power up. This issue is not expected to happen if NOS bootup delays are less than 40 seconds.

Workaround:

Hit **<ESC>** and navigate the BIOS menu to proceed with booting of Ubuntu without changing or saving BIOS configuration.

9.3 SYS-3: Switchd fails with BF_PLTFM CURL ERROR

Symptom:

The **switchd** application fails to run and exits with this message on the console:

```
...BF_PLTFM ERROR - CURL ERROR: 7
```

Condition:

If the USB-Ethernet interface is not functional on the main CPU, this condition will occur each time you attempt to run **switchd**.

Workaround:

1. Find the name of USB-Ethernet interface on the main CPU:

```
admin12@bfn-switch:~$ ifconfig | grep enp0
enp0s20u1u2 Link encap:Ethernet  HWaddr 02:00:00:00:00:02
```

The name might be **enp0s20u1u2**, as shown above, or **enp0s20u1u1**.

2. Make sure the IPv6 protocol is enabled on the USB-Ethernet interface. If it is disabled, edit the **/etc/sysctl.conf** file and comment out or remove the lines that disable IPv6 for **all** and **default** interfaces. For details, see the *SDE 6.0.x Release Notes*, in the section, "Upgrade procedure: Enabling IPv6 on main CPU."
3. Check the USB-Ethernet interface settings in the **interfaces** file:

```
admin12@bfn-switch:~$ sudo vim /etc/network/interfaces
```

Make sure the settings for your USB-Ethernet interface match the bold text, below. Here, we use an example interface name, **enp0s20u1u2**. Yours will be **enp0s20u1u2** or **enp0s20u1u1**.

```
auto lo
iface lo inet loopback

auto enp0s20u1u2
iface enp0s20u1u2 inet6 static
  address fe80::2
  netmask 64
  pre-up sysctl -w net.ipv6.conf.enp0s20u1u2.autoconf=0
  pre-up sysctl -w net.ipv6.conf.enp0s20u1u2.router_solicitations=0
  up ifconfig enp0s20u1u2 txqueuelen 64 up
```

4. Save the **interfaces** file, exit, and reboot the main CPU if you made changes to the configuration.
5. The USB-Ethernet interface should be up and reachable at its IPv6 address after the reboot. If it is not, then open the file, **/etc/rc.local** and add this line at the end of the file:

```
ifconfig enp0s20u1u2 txqueuelen 64 up
```

If the interface is configured correctly but the problem persists, log in to the OpenBMC shell and do an "ifconfig usb0 down" and "ifconfig usb0 up" on the USB-Ethernet interface.

10 Copyright and Trademarks

11 Contacting Support

You can reach the Barefoot Networks customer support team at the support portal, https://support.barefootnetworks.com/hc/en-us or by emailing support@barefootnetworks.com.

12 Index

administrator access, 7, 13
BMC specifications, 7, 41
CLI, 22, 33
  navigating, 33
  port management, 24
connect to BMC CPU, 13, 42
connect to main CPU, 14
CPU specifications, 6
device register map, 10
front panel, 6
kernel mode driver, 18
known issues, 64
LED control software, 9
LED port status lamps, 8
P4 program, 16
  run on Tofino, 19
password, 14
pm context of CLI, 24
port, 22
  diagram, 24
  internal, 22
  management, 24
  switch port, 22
port manager, 24
port naming convention, 23
SDE, 16
  build, 17
Switch ASIC specifications, 7
switch port, 22

BAREFOOT NETWORKS
info@barefootnetworks.com
4750 Patrick Henry Drive, Santa Clara, CA 95054