NVIDIA BlueField DPU BSP v4.5.6 (2023 LTS U7)

Table of Contents

1 About This Document

NVIDIA® BlueField® DPU software is built from the BlueField BSP (Board Support Package) which includes the operating system and the DOCA framework. BlueField BSP includes the bootloaders and other essentials for loading and setting software components. The BSP loads the official BlueField operating system (Ubuntu reference Linux distribution) to the DPU. DOCA is the software framework and SDK for the development of applications and infrastructure services. DOCA includes runtime libraries; the DOCA Runtime stack for Arm supports various accelerations for storage, networking, and security. As such, customers can run any Linux-based application in the BlueField software environment seamlessly.

This guide provides product release notes as well as information on the BSP and how to develop and/or customize applications, system software, and file system images for the BlueField platform.

Important: Make sure to download the latest available software packages for the procedures documented in this guide to run as expected.

1.1 Intended Audience

This document is intended for software developers and DevOps engineers interested in creating and/or customizing software applications and system software for the NVIDIA BlueField DPU platform.

1.2 Software Download

To download product software, refer to the DOCA SDK developer zone.

1.3 Technical Support

Firmware Compatibility

For BlueField-3, a firmware version of 32.38.1002 or greater requires a BFB version of 2.2.0 or higher. Downgrading to lower BFB/firmware versions may result in anomalous behavior.

Proper Power Cycle Procedure

Make sure to perform a graceful shutdown of the Arm OS in advance of performing system/host power cycle when required by the manual.

Customers who purchased NVIDIA products directly from NVIDIA are invited to contact us through the following methods:

- E-mail: enterprisesupport@nvidia.com
- Enterprise Support page: https://www.nvidia.com/en-us/support/enterprise

Customers who purchased NVIDIA M-1 Global Support Services, please see your contract for details regarding technical support.

Customers who purchased NVIDIA products through an NVIDIA-approved reseller should first seek assistance through their reseller.

1.4 Glossary

Term	Description
ACE	AXI coherency extensions
ACPI	Advanced configuration and power interface
AMBA®	Advanced microcontroller bus architecture
ARB	Arbitrate
ATF	Arm-trusted firmware
AXI4	Advanced eXtensible Interface 4
BDF address	Bus, device, function address. This is the device's PCIe bus address to uniquely identify the specific device.
BERT	Boot error record table
BF_INST_DIR	The directory where the BlueField software is installed
BFB	BlueField bootstream
BMC	Board management controller
BSD	BlueField software distribution
BSP	BlueField support package
BUF	Buffer
CBS	Committed burst size
CHI	Coherent hub interface; Arm® protocol used over the BlueField Skymesh specification
CIR	Committed information rate
CL	Cache line
CMDQ	Command queue
CMO	Cache maintenance operation
COB	Collision buffer
DAT	Data
DEK	Data encryption key
DHCP	Dynamic host configuration protocol
DMA	Direct memory access
DOCA	DPU SDK
DORA	Discover; Offer; Request; Acknowledgment
DOT	Device ownership transfer
DPA	Data path accelerator; an auxiliary processor designed to accelerate data-path operations
DPDK	Data plane development kit
DPI	Deep packet inspection

Term	Description
DPU	Data processing unit, the third pillar of the data center with CPU and GPU
DVM	Distributed virtual memory
DW	Dword
EBS	Excess burst size
ECPF	Embedded CPU physical function
EIR	Excess information rate
EMEM/ EMI	External memory interface; block in the MSS which performs the actual read/write from the DDR device
eMMC	Embedded multi-media card
ESP	EFI system partition
ESP header	Encapsulating security payload
EU	Execution unit. HW thread; a logical DPA processing unit.
FIPS	Federal Information Processing Standards
FPGA	Field-programmable gate arrays
FS	File system
FW	Firmware
GDB	GNU debugger
GPT	GUID partition table
HCA	Host-channel adapter
HNF	Home node interface
Host	When referring to "the host" this documentation is referring to the **server host**. When referring to the Arm based host, the documentation will specifically call out "Arm host". • Server host OS refers to the Host Server OS (Linux or Windows) • Arm host refers to the AARCH64 Linux OS which is running on the BlueField Arm Cores
HW	Hardware
hwmon	Hardware monitoring
IB	InfiniBand
ICM	Interface configuration memory
IKE	Internet key exchange
IPMB	Intelligent platform management bus
IPMI	Intelligent platform management interface
IR	Intermediate representation
KGDB	Kernel debugger
KGDBOC	Kernel debugger over console
LAT	Latency
LCRD	Link credit
LSO	Large send offload

Term	Description
LTO	Link-time optimization
MMIO	Memory-mapped I/O
MSB	Most significant bit
MSS	Memory subsystem
MST	Mellanox software tools
NAT	Network address translation
NIC	Network interface card
NIST	National Institute of Standards and Technology
NS	Namespace
OCD	On-chip debugger
OOB	Out-of-band
OS	Operating system
OVS	Open vSwitch
PBS	Peak burst size
PCIe	PCI Express; Peripheral Component Interconnect Express
PF	Physical function
PIR	Peak information rate
PK	Platform key
PKA	Public key accelerator
POC	Point of coherence
RD	Read
RDMA	Remote direct memory access
RegEx	Regular expression
REQ	Request
RES	Response
RMC	Remote management controller
RN	Request node RN-F – Fully coherent request node RN-D – IO coherent request node with DVM support RN-I – IO coherent request node
RNG	Random number generator/generation
RoCE	Ethernet and RDMA over converged Ethernet
RQ	Receive queue
RShim	Random Shim
RTT	Round-trip time
RX	Receive
SA	Security association

Term	Description
SBSA	Server base system architecture
SDK	Software development kit
SF	Sub-function or scalable function
SG	Scatter-gather
SHA	Secure hash algorithm
SMMU	System memory management unit
SNP	Snooping
SQ	Send queue
SR-IOV	Single-root IO virtualization
STL	Stall
Sync event	Synchronization event
TBU	Translation buffer unit
TIR	Transport interface receive
TIS	Transport interface send
TLS	Transport layer security
TRB	Trail buffer
TSO	TCP send offload
TSO	Total store order
TX	Transmit
UDS	Unix domain socket
UEFI	Unified extensible firmware interface
UPVS	UEFI persistent variable store
VF	Virtual function
VFE	Virtio full emulation
VM	Virtual machine
VPI	Virtual protocol interconnect
VST	Virtual switch tagging
WorkQ or workq	Work queue
WQE	Work queue elements
WR	Write
WRDB	Write data buffer

1.5 Related Documentation

Document Name	Description
InfiniBand Architecture Specification, Vol. 1, Release 1.3.1	The InfiniBand Architecture Specification that is provided by IBTA
Firmware Release Notes	See Firmware Release Notes
MFT Documentation	See Firmware Tools Release Notes and User Manual
NVIDIA OFED for Linux User Manual	Intended for system administrators responsible for the installation, configuration, management and maintenance of the software and hardware of VPI adapter cards
WinOF Documentation	See WinOF Release Notes and User Manual
NVIDIA BlueField BMC Software User Manual	This document provides general information concerning the BMC on the NVIDIA® BlueField® DPU, and is intended for those who want to familiarize themselves with the functionality provided by the BMC
NVIDIA BlueField-3 DPU User Guide	This document provides details as to the interfaces of the board, specifications, required software and firmware for operating the board, and a step-by-step plan of how to bring up BlueField-3 DPUs
NVIDIA BlueField-2 Ethernet DPU User Guide	This document provides details as to the interfaces of the board, specifications, required software and firmware, and a step-by-step plan of how to bring up BlueField-2 Ethernet DPUs
NVIDIA BlueField-2 InfiniBand/Ethernet DPU User Guide	This document provides details as to the interfaces of the board, specifications, required software and firmware, and a step-by-step plan of how to bring up BlueField-2 InfiniBand/Ethernet DPUs
NVIDIA BlueField InfiniBand/Ethernet DPU User Guide	This document provides details as to the interfaces of the board, specifications, required software and firmware, and a step-by-step plan of how to bring up BlueField InfiniBand/Ethernet DPUs
NVIDIA DOCA SDK	The NVIDIA DOCA™ SDK enables developers to rapidly create applications and services on top of NVIDIA® BlueField® data processing units (DPUs), leveraging industry-standard APIs. With DOCA, developers can deliver breakthrough networking, security, and storage performance by harnessing the power of NVIDIA's DPUs.
NVIDIA BlueField Reference Platform Hardware User Manual	Provides details as to the interfaces of the reference platform, specifications and hardware installation instructions
NVIDIA BlueField Ethernet Controller Card User Manual	This document provides details as to the interfaces of the board, specifications, required software and firmware for operating the card, hardware installation, driver installation and bring-up instructions
NVIDIA BlueField UEFI Secure Boot User Guide	This document provides details and directions on how to enable UEFI secure boot and sign UEFI images
NVIDIA BlueField Secure Boot User Guide	This document provides guidelines on how to enable the Secure Boot on BlueField DPUs
NVIDIA BlueField SNAP and virtio-blk SNAP Documentation	This document describes the configuration parameters of NVMe SNAP and virtio-blk SNAP in detail
PKA Driver Design and Implementation Architecture Document	This document provides a description of the design and implementation of the Public Key accelerator (PKA) hardware driver. The driver manages and controls the EIP-154 Public Key Infrastructure Engine, an FIPS 140-3 compliant PKA and operates as a co-processor to offload the processor of the host.

Document Name	Description
PKA Programming Guide	This document is intended to guide a new crypto application developer or a public key user space driver. It offers programmers the basic information required to code their own PKA-based application for NVIDIA® BlueField® DPU.

2 Release Notes

The release note pages provide information for NVIDIA® BlueField® DPU family software such as changes and new features, supported platforms, and reports on software known issues as well as bug fixes.

2.1 Changes and New Features

For an archive of changes and features from previous releases, refer to Release Notes Change Log History.

NVIDIA® BlueField® DPUs support configuring network ports as either Ethernet only or InfiniBand only.

2.1.1 Changes and New Features in 4.5.6 LTS

- Optimizations and routine maintenance to improve overall DOCA stability and performance

2.2 Supported Platforms and Interoperability

2.2.1 Supported NVIDIA BlueField-3 DPU Platforms

SKU	PSID	Description
900-9D3D4-00NN-HA0	MT_0000001070	Nvidia BlueField-3 B3140H E-series HHHL DPU; 400GbE(default mode)/NDR IB; Single-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on board DDR; integrated BMC; Crypto Disabled
900-9D3B6-00CV-AA0	MT_0000000884	NVIDIA BlueField-3 B3220 P-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B6-00CC-AA0	MT_0000001024	NVIDIA BlueField-3 B3210 P-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC;Crypto Enabled
900-9D3B6-H1CN-AB0	MT_0000000883	NVIDIA BlueField-3 B3240 P-Series Dual-slot FHHL DPU; 400GbE / NDR IB (default mode); Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled

SKU	PSID	Description
900-9D3C6-00SV-DA0	MT_0000001102	NVIDIA BlueField-3 B3220SH E-Series FHHL Storage Controller; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 48GB on-board DDR; integrated BMC; Crypto Disabled;
900-9D3B4-00PN-EA0	MT_0000001011	NVIDIA BlueField-3 B3140L E-Series FHHL DPU; 400GbE / NDR IB (default mode); Single-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B6-00SN-AB0	MT_0000000964	NVIDIA BlueField-3 B3240 P-Series Dual-slot FHHL DPU; 400GbE / NDR IB (default mode); Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B4-00SC-EA0	MT_0000000967	NVIDIA BlueField-3 B3210L E-series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual port QSFP112; PCIe Gen4.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Disabled
699-21014-0230	NVD0000000038	NVIDIA A800T WITH BLUEFIELD-3; P1014 SKU 230; GENERIC; GA100 80GB HBM2E; PASSIVE DUAL SLOT 350W GEN5; DPU CRYPTO ON
900-9D3B6-00SC-EA0	MT_0000001117	NVIDIA BlueField-3 B3210E E-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3C6-00SV-GA0	MT_0000001101	NVIDIA BlueField-3 B3220SH E-Series No Heatsink FHHL Storage Controller; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 48GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B6-00SV-AA0	MT_0000000965	NVIDIA BlueField-3 B3220 P-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B4-00SV-EA0	MT_0000001094	NVIDIA BlueField-3 B3220L E-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B4-00CC-EA0	MT_0000000966	NVIDIA BlueField-3 B3210L E-series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual port QSFP112; PCIe Gen4.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B4-00CV-EA0	MT_0000001093	NVIDIA BlueField-3 B3220L E-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3D4-00EN-HA0	MT_0000001069	Nvidia BlueField-3 B3140H E-series HHHL DPU; 400GbE(default mode)/NDR IB; Single-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on board DDR; integrated BMC; Crypto Enabled
900-9D3B4-00EN-EA0	MT_0000001010	NVIDIA BlueField-3 B3140L E-Series FHHL DPU; 400GbE / NDR IB (default mode); Single-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B6-00SC-AA0	MT_0000001025	NVIDIA BlueField-3 B3210 P-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3C6-00CV-GA0	MT_0000001083	NVIDIA BlueField-3 B3220SH E-Series No heatsink FHHL Storage Controller; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 48GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B6-00CC-EA0	MT_0000001115	NVIDIA BlueField-3 B3210E E-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled

SKU	PSID	Description
900-9D3C6-00CV-DA0	MT_0000001075	NVIDIA BlueField-3 B3220SH E-Series FHHL Storage Controller; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 48GB on-board DDR; integrated BMC; Crypto Enabled; Secure Boot

2.2.2 Supported NVIDIA BlueField-2 DPU Platforms

NVIDIA SKU	Legacy OPNs	PSID	Description
900-9D206-0083-ST3	MBF2H332A-AECOT	MT_0000000541	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D208-0086-ST4	MBF2M516C-EECOT	MT_0000000728	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-ST3	MBF2H536C-CESOT	MT_0000000767	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0086-ST2	MBF2H536C-CECOT	MT_0000000768	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0076-STA	MBF2H516C-CEUOT	MT_0000000973	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled with UEFI disabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management
900-9D219-0086-ST1	MBF2M516A-CECOT	MT_0000000375	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0086-ST0	MBF2M516A-EECOT	MT_0000000376	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0086-SQ0	MBF2H516C-CECOT	MT_0000000729	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-ST6	MBF2M516C-EESOT	MT_0000000732	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D218-0083-ST2	MBF2H512C-AECOT	MT_0000000724	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D218-0073-ST4	MBF2H512C-AEUOT	MT_0000000972	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled with UEFI disabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management
6991402800000	N/A	NVD0000000020	ZAM/NAS
900-9D250-0048-ST1	MBF2M345A-HECOT	MT_0000000716	BlueField-2 E-Series DPU; 200GbE/HDR single-port QSFP56; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL

NVIDIA SKU	Legacy OPNs	PSID	Description
900-9D208-0076-ST1	MBF2H516C-CESOT	MT_0000000738	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D218-0083-ST4	MBF2H532C-AECOT	MT_0000000765	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0076-STB	MBF2H536C-CEUOT	MT_0000001008	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled with UEFI Disabled; Crypto Disabled; 32GB on-board DDR; 1GbE OOB management; FHHL
P1004 / 699210040230	N/A	NVD0000000015	ROY BlueField-2 + GA100 PCIe Gen4 x8; two 100Gbe/EDR QSFP28 ports; FHFL
900-9D250-0038-ST1	MBF2M345A-HESOT	MT_0000000715	BlueField-2 E-Series DPU; 200GbE/HDR single-port QSFP56; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D218-0073-ST1	MBF2H512C-AESOT	MT_0000000723	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0076-ST5	MBF2M516C-CESOT	MT_0000000731	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0086-ST3	MBF2M516C-CECOT	MT_0000000733	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-ST2	MBF2H516C-EESOT	MT_0000000737	BlueField-2 P-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D218-0073-ST0	MBF2H532C-AESOT	MT_0000000766	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Disabled; 32GB on-board DDR; 1GbE OOB management; FHHL

2.2.3 Embedded Software

The BlueField DPU installation DOCA local repo package for DPU for this release is `DOCA_2.5.5_BSP_4.5.5_Ubuntu_22.04-2.23-07.prod.bfb`.

The following software components are embedded in it:

Component	Version	Description
ATF	v2.2(release):4.5.4-0-g6280f57a6	Arm-trusted firmware is a reference implementation of secure world software for Arm architectures
UEFI	4.5.5-0-ga20921c832	UEFI is a specification that defines the architecture of the platform firmware used for booting and its interface for interaction with the operating system
BlueField-3 NIC firmware	32.39.5124	Firmware is used to run user programs on the BlueField-3 which allow hardware to run

Component	Version	Description
BlueField-2 NIC firmware	24.39.5124	Firmware is used to run user programs on the BlueField-2 which allow hardware to run
BMC firmware	23.10-10	BlueField BMC firmware
BlueField-3 eROT (Glacier)	00.02.0195.0000	BlueField-3 eROT firmware
BlueField-2 eROT (CEC)	04.0f	BlueField-2 eROT firmware

For more information about embedded software components and drivers, refer to the DOCA 2.5.3 Release Notes.

2.2.4 Supported DPU Linux Distributions (aarch64)

- Ubuntu 22.04

2.2.5 Supported DPU Host OS Distributions

The default operating system of the BlueField DPU (Arm) is Ubuntu 22.04.

The supported operating systems on the host machine per DOCA profile are the following:

Only the following generic kernel versions are supported for DOCA local repo package for host installation.

DOCA for Host	Kernel	Arch	doca-all	doca-cx	doca-ofed
CTYunOS3 23.01	5.10	aarch64	✔	✔	✔
Ubuntu 20.04	5.4	x86	✔	✔	✔
Ubuntu 22.04	5.15	x86 / aarch64	✔	✔	✔
Debian 10.8	4.19	x86	✔	✔	✔
Debian 10.13	4.19	x86	✔	✔	✔
Alinux 3.2	5.10	x86	✔	✔	✔
Oracle Linux 8.7	5.15	x86	✔	✔	✔
BCLinux 21.10 SP2	4.19.90	x86 / aarch64			✔
CTYunOS2.0	4.19.90	x86 / aarch64			✔
Debian 10.9	4.19.0-16	x86			✔

Debian 11.3	5.10.0-13	x86 / aarch64			✔
Debian 12.1	6.1.0-10	x86 / aarch64			✔
Debian 12.5	6.1.0-18	x86 / aarch64	✔	✔	✔
Kylin 10 SP2	4.19.90	x86 / aarch64			✔
Oracle Linux 8.6	5.4	x86			✔
openEuler 20.03 SP3	4.19.90	x86 / aarch64			✔
openEuler 22.03	5.10.0	x86 / aarch64			✔
RHEL/CentOS 8.0	4.18.0-80.el8	x86			✔
RHEL/CentOS 8.2	4.18	x86	✔	✔	✔
RHEL/CentOS 8.2	4.18.0-193.el8	aarch64			✔
RHEL/CentOS 8.4	4.18.0-305.el8	x86 / aarch64			✔
RHEL/Rocky 8.6	4.18	x86	✔	✔	✔
RHEL/Rocky 8.6	4.18.0-372.41.1.el8	aarch64			✔
RHEL/Rocky 8.8	4.18.0-477.10.1.el8_8	x86 / aarch64	✔	✔	✔
RHEL/Rocky 8.9	4.18.0-513.5.1.el8_9	x86 / aarch64			✔
RHEL/Rocky 8.10	4.18.0-553.el8_10	x86 / aarch64			✔
RHEL/Rocky 9.0	5.14.0-70.46.1.el9_0	x86 / aarch64			✔
RHEL/Rocky 9.1	5.14	x86	✔	✔	✔
RHEL/Rocky 9.1	5.14.0-162.19.1.el9_1	x86 / aarch64			✔
RHEL/Rocky 9.2	5.14.0-284.11.1.el9_2	x86 / aarch64			✔
RHEL/Rocky 9.3	5.14.0-362.8.1.el9_3	x86 / aarch64			✔
RHEL/Rocky 9.4	5.14.0-427.13.1.el9_4	x86 / aarch64			✔
SLES 15 SP3	5.3.18-57	x86 / aarch64			✔
SLES 15 SP4	5.14.21-150400.22	x86 / aarch64			✔
SLES 15 SP5	5.14.21-150500.53	x86 / aarch64			✔
SLES 15 SP6	6.4.0-150600.21-default	x86 / aarch64			✔

2.2.6 Supported Open vSwitch

- 2.15.1

2.3 Bug Fixes In This Version

For an archive of bug fixes from previous releases, please see "Bug Fixes History".

Ref #	Issue Description
N/A	Description: N/A
	Keywords: N/A
	Reported in version: N/A

2.4 Known Issues

Ref #	Issue
4824938	Description: Following an `MmcBootCap` capsule update, the DPU may reboot unexpectedly on the second boot. Interfaces remain down and the host cannot detect the device.
	Workaround: Restart Host mlx5 driver.
	Keyword: Secure boot; update
	Reported in version: 4.5.5
3880194	Description: `mlxbf-bootctl` command failed to install `default.bfb`.

	Workaround: The following are possible options -
	• Boot the BFB file `doca_2.5.0_bsp_4.5.0_ubuntu_22.04-1.23-10.prod.bfb` to update the platform software
	• Download, compile, and install latest `mlxbf-bootctl` command from GitHub
	• Edit `default.bfb` by using the `mlx-mkbfb` command to incorporate the platform-specific images and filtering out unused images. Example for a BlueField-2 device:
	`$ mlx-mkbfb -x default.bfb`
	`$ mlx-mkbfb \`
	`--bl2r-v1=dump-bl2r-v1 \`
	`--bl2r-cert-v1=dump-bl2r-cert-v1 \`
	`--bl2-v1=dump-bl2-v1 \`
	`--bl2-cert-v1=dump-bl2-cert-v1 \`
	`--bl31-v1=dump-bl31-v1 \`
	`--bl31-cert-v1=dump-bl31-cert-v1 \`
	`--bl31-key-cert-v1=dump-bl31-key-cert-v1 \`
	`--bl33-v0=dump-bl33-v0 \`
	`--bl33-cert-v1=dump-bl33-cert-v1 \`
	`--bl33-key-cert-v1=dump-bl33-key-cert-v1 \`
	`--boot-acpi-v0=dump-boot-acpi-v0 \`
	`--boot-args-v0=dump-boot-args-v0 \`
	`--boot-desc-v0=dump-boot-desc-v0 \`
	`--boot-path-v0=dump-boot-path-v0 \`
	`--ddr_ini-v1=dump-ddr_ini-v1 \`
	`--ddr-cert-v1=dump-ddr-cert-v1 \`
	`--ddr_ate_imem-v1=dump-ddr_ate_imem-v1 \`
	`--ddr_ate_dmem-v1=dump-ddr_ate_dmem-v1 \`
	`--snps_images-v1=dump-snps_images-v1 \`
	`--trusted-key-cert-v1=dump-trusted-key-cert-v1 \`
	`default_min.bfb`

	Keywords: Software; upgrade
	Discovered in version: 4.5.1
3204153	Description: On BlueField-2, the OOB may not get an IP address due to the interface being down.

Ref #	Issue
	Workaround: Restart auto-negotiation using the command `ethtool -r oob_net0`.
	Keyword: OOB; IP
	Reported in version: 4.5.0
3601491	Description: Symmetric pause must be enabled in the DHCP server for the OOB to be able to reliably get an IP address assigned.
	Workaround: N/A
	Keyword: OOB; IP
	Reported in version: 4.5.0
3673330	Description: On Debian 12, Arm ports remain in Legacy mode after multiple Arm reboot iterations. The following error message appears in `/var/log/syslog`: `mlnx_bf_configure[2601]: ERR: Failed to configure switchdev mode for 0000:03:00.0 after 61 retries`

	Workaround: Run:
	`$ echo SET_MODE_RETRY_NUM=300 >> /etc/mellanox/mlnx-bf.conf`
	`$ reboot`

	Keyword: Debian; Arm
	Reported in version: 4.5.0
3695543	Description: PXE boot may fail after a firmware upgrade from 32.36.xxxx, 32.37.xxxx, to 32.38.xxxx and above.

	Workaround: Create `/etc/bf.cfg` with the following lines, then run `bfcfg` to recreate the PXE boot entries:
	`BOOT0=DISK`
	`BOOT1=NET-NIC_P0-IPV4`
	`BOOT2=NET-NIC_P0-IPV6`
	`BOOT3=NET-NIC_P1-IPV4`
	`BOOT4=NET-NIC_P1-IPV6`
	`BOOT5=NET-OOB-IPV4`
	`BOOT6=NET-OOB-IPV6`

	Keyword: MAC allocation; PXE boot
	Reported in version: 4.5.0
3663398	Description: On rare occasions, OP-TEE may panic upon boot.
	Workaround: Perform graceful shutdown and power cycle the DPU.
	Keyword: fTPM over OP-TEE
	Reported in version: 4.5.0
3647476	Description: Debian 12 OS does not support CT tunnel offload.
	Workaround: Recompile the kernel with `CONFIG_NET_TC_SKB_EXT` set.
	Keyword: Connection tracking; Linux
	Reported in version: 4.5.0

Ref #	Issue
3007696	Description: When configuring a static IP address for `tmfifo_net0` interface in `/etc/network/interfaces`, the IP address is lost after restarting the RShim driver on Debian Linux.

	Workaround: Use netplan configuration. For example
	`# cat /etc/netplan/tmfifo_net0.yaml`
	`network:`
	`  version: 2`
	`  renderer: networkd`
	`  ethernets:`
	`  tmfifo_net0:`
	`  addresses:`
	`  - 192.168.100.1/30`
	`  dhcp4: false`
	Then run "netplan apply".

	Keyword: IP address; tmfifo_net0; host
	Reported in version: 4.5.0
3633453	Description: Jumbo MTU is only supported on a guest OS with kernel 4.11 and above.
	Workaround: N/A
	Keyword: Virtio-net; jumbo MTU
	Reported in version: 4.5.0
3021967	Description: When rebooting a DPU with a large number of VFs created on host, VF recovery may fail due to timeout.
	Workaround: Restart the driver on the host after the DPU is up.
	Keyword: Reboot; VFs
	Reported in version: 4.5.0
3670628	Description: When NIC subsystem is in recovery mode, the interface towards to NVMe is not accessible. Thus, the SSD boot device would not be available.
	Workaround: The admin must configure the Arm subsystem boot device to boot from the eMMC, for example.
	Keyword: mlxfwreset; RShim
	Reported in version: 4.5.0
3702393	Description: On rare occasions, the boot process part of SWRESET (via RShim) or FWRESET (via mlxfwreset) may result in a device hanging on the boot flow or cause the host server to reboot.
	Workaround: Perform graceful shutdown and then a power cycle.
	Keyword: mlxfwreset; RShim
	Reported in version: 4.5.0
3695367	Description: For BlueField-2, although an option to configure "large ICM size" appears in the UEFI menu it is not functional as large ICM size is not supported on it.
	Workaround: N/A
	Keyword: UEFI
	Reported in version: 4.5.0
3665724	Description: If the UEFI password is an empty string (""), then it cannot be changed via Redfish.
	Workaround: UEFI; password; Redfish

Ref #	Issue
	Keyword: UEFI; password; Redfish
	Reported in version: 4.5.0
3677366	Description: On rare occasions, the devices `/dev/tpm0` and `/dev/tpmrm0` are not created triggering an fTPM panic during boot. This message indicates that the fTPM over OP-TEE feature is not functional.
	Workaround: Reboot the DPU.
	Keyword: fTPM over OP-TEE
	Reported in version: 4.5.0
3671185	Description: XFRM rules must be deleted before driver restart or warm reboot are performed.
	Workaround: N/A
	Keyword: IPsec
	Reported in version: 4.5.0
3666160	Description: Installing BFB using `bfb-install` when `mlxconfig` `PF_TOTAL_SF` >1700, triggers server reboot immediately.
	Workaround: Change `PF_TOTAL_SF` to 0, perform graceful shutdown, then power cycle, and then install the BFB.
	Keyword: SF; PF_TOTAL_SF; BFB installation
	Reported in version: 4.2.2
3618936	Description: When moving to DPU mode from NIC mode, it is necessary to reinstall the BFB and perform a graceful reboot to the DPU by shutting down the Arm cores before rebooting the host system.
	Workaround: N/A
	Keyword: NIC mode
	Reported in version: 4.2.2
3605254	Description: Following a system power cycle, both the DPU and BMC boot independently which may lead to the DPU's UEFI boot process to complete before the BMC's. As a result, when attempting to establish Redfish communication, the BMC may not yet be prepared to respond.
	Workaround: Wait until the BMC is done booting before issuing a reset command to the DPU.
	Keyword: Power cycle; Redfish; boot
	Reported in version: 4.2.1
3603146	Description: Running `mlxfwreset` on BlueField-3 may cause the external host to crash when the RShim driver is running on that host.
	Workaround: Stop the RShim driver on the external host using `systemctl stop rshim` before performing `mlxfwreset`.
	Keyword: RShim; mlxfwreset
	Reported in version: 4.2.1

Ref #	Issue
3602044	Description: When the public key is deleted while Redfish is enabled, UEFI secure boot is disabled and UEFI reverts to Setup Mode (i.e., the `SecureBootEnable` Redfish property is reset to `false`). If later, the public key is re-enrolled, the platform does not implement UEFI secure boot until the `SecureBootEnable` Redfish property is explicitly changed to `true`.
	Workaround: Set `SecureBootEnable` to true using the Redfish API.
	Keyword: Redfish; UEFI secure boot
	Reported in version: 4.2.1
3592080	Description: When using UEK8 on the host in DPU mode, creating a VF on the host consumes about 100MB memory on the DPU.
	Workaround: N/A
	Keyword: UEK; VF
	Reported in version: 4.2.1
3568341	Description: Downgrading BSP software from 4.2.0 fails if UEFI secure boot is enabled.
	Workaround: Disable UEFI secure boot before downgrading.
	Keyword: Software; downgrade
	Reported in version: 4.2.0
3566042	Description: Virtio hotplug is not supported in GPU-HOST mode on the NVIDIA Converged Accelerator.
	Workaround: N/A
	Keyword: Virtio; Converged Accelerator
	Reported in version: 4.2.0
3546474	Description: PXE boot over ConnectX interface might not work due to an invalid MAC address in the UEFI boot entry.
	Workaround: On the DPU, create `/etc/bf.cfg` file with the relevant PXE boot entries, then run the command `bfcfg`.
	Keyword: PXE; boot; MAC
	Reported in version: 4.2.0
3306489	Description: After rebooting a BlueField-3 DPU running Rocky Linux 8.6 BFB, the kernel log shows the following error: `[ 3.787135] mlxbf_gige MLNXBF17:00: Error getting PHY irq. Use polling instead` This message indicates that the Ethernet driver will function normally in all aspects, except that PHY polling is enabled.
	Workaround: N/A
	Keyword: Linux; PHY; kernel
	Reported in version: 4.2.0
3529297	Description: Enhanced NIC mode is not supported on BlueField-2 DPUs.
	Workaround: N/A
	Keyword: Operation; mode

Ref #	Issue
	Reported in version: 4.2.0
3306489	Description: When performing longevity tests (e.g., mlxfwreset, DPU reboot, burning of new BFBs), a host running an Intel CPU may observer errors related to "CPU 0: Machine Check Exception".
	Workaround: Add `intel_idle.max_cstate=1` entry to the kernel command line.
	Keyword: Longevity; mlxfwreset; DPU reboot
	Reported in version: 4.2.0
3538486	Description: When removing LAG configuration from the DPU, a kernel warning for `uverbs_destroy_ufile_hw` is observed if virtio-net-controller is still running.
	Workaround: Stop virtio-net-controller service before cleaning up bond configuration.
	Keyword: Virtio-net; LAG
	Reported in version: 4.2.0
3444073	Description: `mlxfwreset` is not supported in this release.
	Workaround: Perform graceful shutdown and power cycle the host.
	Keyword: mlxfwreset; support
	Reported in version: 4.0.2

3462630	Description: When trying to perform a PXE installation when UEFI Secure Boot is enabled, the following error messages may be observed:
	`error: shim_lock protocol not found.`
	`error: you need to load the kernel first.`

	Workaround: Download a Grub EFI binary from the Ubuntu website. For further information on Ubuntu UEFI Secure Boot PXE Boot, please visit Ubuntu's official website.
	Keyword: PXE; UEFI Secure Boot
	Reported in version: 4.0.2
3412847	Description: Socket-Direct is currently not supported on BlueField-3 devices.
	Workaround: N/A
	Keyword: Socket-Direct; support
	Reported in version: 4.0.2
3448841	Description: While running CentOS 8.2, switchdev Ethernet DPU runs in "shared" RDMA net namespace mode instead of "exclusive".
	Workaround: Use `ib_core` module parameter `netns_mode=0`. For example: `echo "options ib_core netns_mode=0" >> /etc/modprobe.d/mlnx-bf.conf`
	Keywords: RDMA; isolation; Net NS
	Reported in version: 4.0.2
3413938	Description: Using `mlnx-sf` script, creating and deleting an SF with same ID number in a stressful manner may cause the setup to hang due to a race between create and delete commands.
	Workaround: N/A

Ref #	Issue
	Keywords: Hang; `mlnx-sf`
	Reported in version: 4.0.2
3452740	Description: Ovs-pki is not working due to two versions of OpenSSL being installed, causing the PKA engine to not load properly.
	Workaround: N/A
	Keywords: PKA; OpenSSL
	Reported in version: 4.0.2
3232444	Description: After live migration of virtio-net devices using the VFE driver, the `max_queues_size` output from the `virtnet list` may be wrong. This does not affect the actual value.
	Workaround: N/A
	Keywords: Virtio-net; live migration
	Reported in version: 4.0.2
3441287	Description: Failure occurs when attempting to raise static LAG with `ifenslave_2.10ubuntu3` package.
	Workaround: Use `ifenslave_2.9ubuntu1`.
	Keywords: `ifenslave`; bonding
	Reported in version: 4.0.2
3341481	Description: RShim console may hang after pushing BFB or running reboot command from the DPU Arm Linux.
	Workaround: Restart the RShim driver on the host side using `systemctl restart rshim`.
	Keywords: RShim console; hang; BFB push
	Reported in version: 4.0.2
3273435	Description: Changing the mode of operation between NIC and DPU modes results in different capabilities for the host driver which might cause unexpected behavior.
	Workaround: Reload the host driver or reboot the host.
	Keywords: Modes of operation; driver
	Reported in version: 4.0.2
2706803	Description: When an NVMe controller, SoC management controller, and DMA controller are configured, the maximum number of VFs is limited to 124.
	Workaround: N/A
	Keywords: VF; limitation
	Reported in version: 4.0.2
3948009	Description: The command `bfcfg -d` may show an incorrect out-of-band MAC.
	Workaround: Read the OOB MAC address from `/sys/devices/platform/MLNXBF04:00/oob_mac` instead.
	Keywords: OOB; MAC
	Reported in version: 3.9.7

Ref #	Issue
3264224	Description: When trying to change boot order using efibootmgr, BlueField fails to attempt PXE boot from `p0` even though efibootmgr returns a successful result.
	Workaround: Drop into the UEFI menu and regenerate all the EFI entries.
	Keywords: PXE; efibootmgr
	Reported in version: 3.9.3.1
3188415	Description: An Arm firmware update to the same version that is installed will fail and is not supported.
	Workaround: N/A
	Keywords: Arm; firmware; update
	Reported in version: 3.9.2
N/A	Description: The `BootOptionEnabled` attribute changes back to true after DPU-force reset.
	Workaround: N/A
	Keywords: Redfish; `BootOptionEnabled`
	Reported in version: 3.9.2
3012182	Description: The command `ethtool -I --show-fec` is not supported by the DPU with kernel 5.4.
	Workaround: N/A
	Keywords: Kernel; show-fec
	Reported in version: 3.9.0
2855986	Description: After disabling SR-IOV VF on a virtio device, removing virtio-net/PCIe driver from guest OS may render the virtio controller unusable.
	Workaround: Restart the virtio-net controller to recover it. To avoid this issue, monitor the log from controller and make sure VF resources are destroyed before unloading virtio-net/PCIe drivers.
	Keywords: Virtio-net; VF
	Reported in version: 3.9.0
2863456	Description: SA limit by packet count (hard and soft) are supported only on traffic originated from the ECPF. Trying to configure them on VF traffic removes the SA when hard limit is hit. However, traffic could still pass as plain text due to the tunnel offload used in such configuration.
	Workaround: N/A
	Keywords: ASAP2; IPsec Full Offload
	Reported in version: 3.9.0
2982184	Description: When multiple BlueField resets are issued within 10 seconds of each other, EEPROM error messages are displayed on the console and, as a result, the BlueField may not boot from the eMMC and may halt at the UEFI menu.
	Workaround: Power-cycle the BlueField to fix the EEPROM issue. Manual recovery of the boot options and/or SW installation may be needed.
	Keywords: Reset; EEPROM
	Reported in version: 3.9.0
2853408	Description: Some pre-OS environments may fail when sensing a hot plug operation during their boot stage.

Ref #	Issue
	Workaround: Run "`mlxconfig -d <mst dev> set PF_LOG_BAR_SIZE=0`".
	Keywords: BIOS; hot-plug; Virtio-net
	Reported in version: 3.9.0
2934833	Description: Running I/O traffic and toggling both physical ports status in a stressful manner on the receiving-end machine may cause traffic loss.
	Workaround: N/A
	Keywords: MLNX_OFED; RDMA; port toggle
	Reported in version: 3.8.5
2911425	Description: ProLiant DL385 Gen10 Plus server with BIOS version 1.3 hangs when large number of SFs (`PF_TOTAL_SF=252`) are configured.
	Workaround: Update the BIOS version to 2.4 which should correctly detect the PCIe device with the bigger BAR size.
	Keywords: Scalable functions; BIOS
	Reported in version: 3.8.5
2801780	Description: When running virtio-net-controller with host kernel older than 3.10.0-1160.el7, host virtio driver may get error (`Unexpected TXQ (13) queue failure: -28`) from dmesg in traffic stress test.
	Workaround: N/A
	Keywords: Virtio-net; error
	Reported in version: 3.8.0
2870213	Description: Servers do not recover after configuring `PCI_SWITCH_EMULATION_NUM_PORT` to 32 followed by power cycle.
	Workaround: N/A
	Keywords: VirtIO-net; power cycle
	Reported in version: 3.8.0
-	Description: Only QP queues are supported for GGA accelerators from this version onward.
	Workaround: N/A
	Keywords: Firmware; SQ; QP
	Reported in version: 3.8.0
2846108	Description: Setting `VHCA_TRUST_LEVEL` does not work when there are active SFs or VFs.
	Workaround: N/A
	Keywords: Firmware; SF; VF
	Reported in version: 3.8.0
2750499	Description: Some devlink commands are only supported by mlnx devlink (`/opt/mellanox/iproute2/sbin/devlink`). The default devlink from the OS may produce failure (e.g., `devlink port show -j`).
	Workaround: N/A

Ref #	Issue
	Keywords: Devlink
	Reported in version: 3.7.1
2730157	Description: Kernel upgrade is not currently supported on BlueField as there are out of tree kernel modules (e.g., ConnectX drivers that will stop working after kernel upgrade).
	Workaround: Kernel can be upgraded if there is a matching DOCA repository that includes all the drivers compiled with the new kernel or as a part of the new BFB package.
	Keywords: Kernel; upgrade
	Reported in version: 3.7.0
2706710	Description: Call traces are seen on the host when recreating VFs before the controller side finishes the deletion procedure.
	Workaround: N/A
	Keywords: Virtio-net controller
	Reported in version: 3.7.0
2685478	Description: 3rd party (netkvm.sys) Virtio-net drivers for Windows do not support SR-IOV.
	Workaround: N/A
	Keywords: Virtio-net; SR-IOV; WinOF-2
	Reported in version: 3.7.0
2685191	Description: Once Virtio-net is enabled, the mlx5 Windows VF becomes unavailable.
	Workaround: N/A
	Keywords: Virtio-net; virtual function; WinOF-2
	Reported in version: 3.7.0
2702395	Description: When a device is hot-plugged from the virtio-net controller, the host OS may hang when warm reboot is performed on the host and Arm at the same time.
	Workaround: Reboot the host OS first and only then reboot DPU.
	Keywords: Virtio-net controller; hot-plug; reboot
	Reported in version: 3.7.0
2684501	Description: Once the contiguous memory pool, a limited resource, is exhausted, fallback allocation to other methods occurs. This process triggers `cma_alloc` failures in the dmesg log.
	Workaround: N/A
	Keywords: Log; cma_alloc; memory
	Reported in version: 3.7.0
2590016	Description: ibdev2netdev tool is not supported for PCIe PF operating in switchdev mode or on SFs.
	Workaround: N/A
	Keywords: ibdev2netdev
	Reported in version: 3.6.0.11699
2590016	Description: A "double free" error is seen when using the "curl" utility. This error is from libcrypto.so library which is part of the OpenSSL package. This happens only when OpenSSL is configured to use a dynamic engine (e.g. Bluefield PKA engine).

Ref #	Issue
	Workaround: Set `OPENSSL_CONF=/etc/ssl/openssl.cnf.orig` before using the curl utility. For example: `# OPENSSL_CONF=/etc/ssl/openssl.cnf.orig curl -O https://tpo.pe/pathogen.vim` ⚠ OPENSSL_CONF is aimed at using a custom config file for applications. In this case, it is used to point to a config file where dynamic engine (PKA engine) is not enabled.
	Keywords: OpenSSL; curl
	Reported in version: 3.6.0.11699
2407897	Description: The host may crash when the number of PCIe devices overflows the PCIe device address. According to the PCIe spec, the device address space is 8 bits in total—device (5 bits) and function (3 bits)—which means that the total number of devices cannot be more than 256. The second PF maximum number of VFs is limited by the total number of additional PCIe devices that precedes it. By default, the preceding PCIe devices are 2 PFs + RShim DMA + 127 VFs of the first PF. This means that the maximum valid number of VFs for the second port will be 126.
	Workaround: Use the maximum allowed VFs on the 2nd PCIe PF of BlueField instead of the maximum of 127 VFs.
	Keywords: Emulated devices; VirtIO-net; VirtIO-blk; VFs; RShim
	Reported in version: 3.6.0.11699
2445289	Description: If secure boot is enabled, MFT cannot be installed on the BlueField DPU independently from BlueField drivers (MLNX_OFED).
	Workaround: N/A
	Keywords: MFT; secure boot
	Reported in version: 3.5.1.11601
2377021	Description: Executing `sudo poweroff` on the Arm side causes the system to hang.
	Workaround: Perform graceful shutdown, then reboot your BlueField device or power cycle the server.
	Keywords: Hang; reboot
	Reported in version: 3.5.0.11563
2350132	Description: Boot process hangs at BIOS (version 1.2.11) stage when power cycling a server (model Dell PowerEdge R7525) after configuring "PCI_SWITCH_EMULATION_NUM_PORT" > 27.
	Workaround: N/A
	Keywords: Server; hang; power cycle
	Reported in version: 3.5.0.11563
2581408	Description: On a BlueField device operating in Embedded CPU mode, PXE driver will fail to boot if the Arm side is not fully loaded and the OVS bridge is not configured.
	Workaround: Run warm reboot on the host side and boot again via the device when Arm is up and the OVS bridge is configured.
	Keywords: Embedded CPU; PXE; UEFI; Arm
	Reported in version: 2.5.0.11176

Ref #	Issue
1859322	Description: On some setups, DPU does not power on following server cold boot when UART cable is attached to the same server.
	Workaround: As long as the RShim driver is loaded on the server and the RShim interface is visible, the RShim driver will detect this and auto-reset the card into normal state.
	Keywords: DPU; Arm; Cold Boot
	Reported in version: 2.4.0.11082
1899921	Description: Driver restart fails when SNAP service is running.
	Workaround: Stop the SNAP services nvme_sf and nvme_snap@nvme0, then restart the driver. After the driver loads restart the services.
	Keywords: SNAP
	Reported in version: 2.2.0.11000
1911618	Description: Defining namespaces with certain Micron disks (Micron_9300_MTFDHAL3T8TDP) using consecutive attach-ns commands can cause errors.
	Workaround: Add delay between attach-ns commands.
	Keywords: Micron; disk; namespace; attach-ns
	Reported in version: 2.2.0.11000

2.5 Validated and Supported Cables and Modules

2.5.1 Cables Lifecycle Legend

Lifecycle Phase	Definition
EOL	End of Life
LTB	Last Time Buy
HVM	GA level
MP	GA level
P-Rel	GA level
Preliminary	Engineering Sample
Prototype	Engineering Sample

2.5.2 Supported Cables and Modules for BlueField-3

2.5.2.1 NDR / 400GbE Cables

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	400GE	980-9I08L-00W003	C-DQ8FNM003-NML	NVIDIA Select 400GbE QSFP-DD AOC 3m	Preliminary
N/A	400GE	980-9I08N-00W005	C-DQ8FNM005-NML	NVIDIA Select 400GbE QSFP-DD AOC 5m	Preliminary
N/A	400GE	980-9I08P-00W010	C-DQ8FNM010-NML	NVIDIA Select 400GbE QSFP-DD AOC 10m	Preliminary
N/A	400GE	980-9I08R-00W020	C-DQ8FNM020-NML	NVIDIA Select 400GbE QSFP-DD AOC 20m	Preliminary
N/A	400GE	980-9I08T-00W050	C-DQ8FNM050-NML	NVIDIA Select 400GbE QSFP-DD AOC 50m	Preliminary
NDR	NA	980-9I068-00NM00	MMS1X00-NS400	NVIDIA single port transceiver, 400Gbps, NDR, QSFP112, MPO, 1310nm SMF, up to 500m, flat top	Early BOM
NDR	N/A	980-9I81B-00N004	MCA7J65-N004	NVIDIA Active copper splitter cable, IB twin port NDR 800Gb/s to 2x400Gb/s, OSFP to 2xQSFP112, 4m	Prototype
NDR	N/A	980-9I81C-00N005	MCA7J65-N005	NVIDIA Active copper splitter cable, IB twin port NDR 800Gb/s to 2x400Gb/s, OSFP to 2xQSFP112, 5m	Prototype
NDR	N/A	980-9I76G-00N004	MCA7J75-N004	NVIDIA Active copper splitter cable, IB twin port NDR 800Gb/s to 4x200Gb/s, OSFP to 4xQSFP112, 4m	Prototype
NDR	N/A	980-9I76H-00N005	MCA7J75-N005	NVIDIA Active copper splitter cable, IB twin port NDR 800Gb/s to 4x200Gb/s, OSFP to 4xQSFP112, 5m	Prototype
NDR	N/A	980-9I928-00N001	MCP7Y10-N001	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 2x400Gb/s, OSFP to 2xQSFP112,1m	P-Rel
NDR	N/A	980-9I929-00N002	MCP7Y10-N002	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 2x400Gb/s, OSFP to 2xQSFP112,2m	P-Rel
NDR	N/A	980-9I80P-00N003	MCP7Y10-N003	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 2x400Gb/s, OSFP to 2xQSFP112,3m	P-Rel
NDR	N/A	980-9I80A-00N01A	MCP7Y10-N01A	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 2x400Gb/s, OSFP to 2xQSFP112,1.5m	P-Rel

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
NDR	N/A	980-9I80Q-00N02A	MCP7Y10-N02A	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 2x400Gb/s, OSFP to 2xQSFP112,2.5m	P-Rel
NDR	N/A	980-9I80B-00N001	MCP7Y40-N001	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 4x200Gb/s, OSFP to 4xQSFP112, 1m	P-Rel
NDR	N/A	980-9I80C-00N002	MCP7Y40-N002	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 4x200Gb/s, OSFP to 4xQSFP112, 2m	P-Rel
NDR	N/A	980-9I75R-00N003	MCP7Y40-N003	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 4x200Gb/s, OSFP to 4xQSFP112, 3m	P-Rel
NDR	N/A	980-9I75D-00N01A	MCP7Y40-N01A	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 4x200Gb/s, OSFP to 4xQSFP112, 1.5m	P-Rel
NDR	N/A	980-9I75S-00N02A	MCP7Y40-N02A	NVIDIA passive copper splitter cable, IB twin port NDR 800Gb/s to 4x200Gb/s, OSFP to 4xQSFP112, 2.5m	P-Rel
NDR	N/A	980-9I73U-000003	MFP7E10-N003	NVIDIA passive fiber cable, MMF , MPO12 APC to MPO12 APC, 3m	MP
NDR	N/A	980-9I73V-000005	MFP7E10-N005	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 5m	MP
NDR	N/A	980-9I57W-000007	MFP7E10-N007	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 7m	MP
NDR	N/A	980-9I57X-00N010	MFP7E10-N010	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 10m	MP
NDR	N/A	980-9I57Y-000015	MFP7E10-N015	NVIDIA passive fiber cable, MMF , MPO12 APC to MPO12 APC, 15m	MP
NDR	N/A	980-9I57Z-000020	MFP7E10-N020	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 20m	MP
NDR	N/A	980-9I573-00N025	MFP7E10-N025	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 25m	MP
NDR	N/A	980-9I570-00N030	MFP7E10-N030	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 30m	MP
NDR	N/A	980-9I570-00N035	MFP7E10-N035	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 35m	MP
NDR	N/A	980-9I570-00N040	MFP7E10-N040	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 40m	MP
NDR	N/A	980-9I57Y-00N050	MFP7E10-N050	NVIDIA passive fiber cable, MMF, MPO12 APC to MPO12 APC, 50m	MP
NDR	N/A	980-9I571-00N003	MFP7E20-N003	NVIDIA passive fiber cable, MMF, MPO12 APC to 2xMPO12 APC, 3m	MP
NDR	N/A	980-9I572-00N005	MFP7E20-N005	NVIDIA passive fiber cable, MMF, MPO12 APC to 2xMPO12 APC, 5m	MP

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
NDR	N/A	980-9I573-00N007	MFP7E20-N007	NVIDIA passive fiber cable, MMF, MPO12 APC to 2xMPO12 APC, 7m	MP
NDR	N/A	980-9I554-00N010	MFP7E20-N010	NVIDIA passive fiber cable, MMF, MPO12 APC to 2xMPO12 APC, 10m	MP
NDR	N/A	980-9I555-00N015	MFP7E20-N015	NVIDIA passive fiber cable, MMF, MPO12 APC to 2xMPO12 APC, 15m	MP
NDR	N/A	980-9I556-00N020	MFP7E20-N020	NVIDIA passive fiber cable, MMF, MPO12 APC to 2xMPO12 APC, 20m	MP
NDR	N/A	980-9I557-00N030	MFP7E20-N030	NVIDIA passive fiber cable, MMF, MPO12 APC to 2xMPO12 APC, 30m	MP
NDR	N/A	980-9I55Z-00N050	MFP7E20-N050	NVIDIA passive fiber cable, MMF, MPO12 APC to 2xMPO12 APC, 50m	MP
NDR	N/A	980-9I558-00N001	MFP7E30-N001	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 1m	MP
NDR	N/A	980-9I559-00N002	MFP7E30-N002	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 2m	MP
NDR	N/A	980-9I55A-00N003	MFP7E30-N003	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 3m	MP
NDR	N/A	980-9I55B-00N005	MFP7E30-N005	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 5m	MP
NDR	N/A	980-9I58C-00N007	MFP7E30-N007	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 7m	MP
NDR	N/A	980-9I58D-00N010	MFP7E30-N010	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 10m	MP
NDR	N/A	980-9I58E-00N015	MFP7E30-N015	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 15m	MP
NDR	N/A	980-9I58F-00N020	MFP7E30-N020	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 20m	MP
NDR	N/A	980-9I58G-00N030	MFP7E30-N030	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 30m	MP
NDR	N/A	980-9I580-00N030	MFP7E30-N040	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 40m	MP
NDR	N/A	980-9I58H-00N050	MFP7E30-N050	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 50m	MP
NDR	N/A	980-9I581-00N050	MFP7E30-N060	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 60m	MP
NDR	N/A	980-9I582-00N050	MFP7E30-N070	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 70m	MP
NDR	N/A	980-9I58I-00N100	MFP7E30-N100	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 100m	MP
NDR	N/A	980-9I58J-00N150	MFP7E30-N150	NVIDIA passive fiber cable, SMF, MPO12 APC to MPO12 APC, 150m	MP
NDR	N/A	980-9I58K-00N003	MFP7E40-N003	NVIDIA passive fiber cable, SMF, MPO12 APC to 2xMPO12 APC, 3m	MP

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
NDR	N/A	980-9I58L-00N005	MFP7E40-N005	NVIDIA passive fiber cable, SMF, MPO12 APC to 2xMPO12 APC, 5m	MP
NDR	N/A	980-9I58M-00N007	MFP7E40-N007	NVIDIA passive fiber cable, SMF, MPO12 APC to 2xMPO12 APC, 7m	MP
NDR	N/A	980-9I58N-00N010	MFP7E40-N010	NVIDIA passive fiber cable, SMF, MPO12 APC to 2xMPO12 APC, 10m	MP
NDR	N/A	980-9I56O-00N015	MFP7E40-N015	NVIDIA passive fiber cable, SMF, MPO12 APC to 2xMPO12 APC, 15m	MP
NDR	N/A	980-9I56P-00N020	MFP7E40-N020	NVIDIA passive fiber cable, SMF, MPO12 APC to 2xMPO12 APC, 20m	MP
NDR	N/A	980-9I56Q-00N030	MFP7E40-N030	NVIDIA passive fiber cable, SMF, MPO12 APC to 2xMPO12 APC, 30m	MP
NDR	N/A	980-9I56R-000050	MFP7E40-N050	NVIDIA passive fiber cable, SMF, MPO12 APC to 2xMPO12 APC, 50m	MP
NDR	N/A	980-9I693-00NS00	MMA1Z00-NS400	NVIDIA single port transceiver, 400Gbps,NDR, QSFP112, MPO12 APC, 850nm MMF, up to 50m, flat top	P-Rel

2.5.2.2 HDR / 200GbE Cables

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
HDR	200GE	980-9I548-00H001	MCP1650-H001E30	Nvidia Passive Copper cable, up to 200Gbps, QSFP56 to QSFP56, 1m	HVM
HDR	200GE	980-9I549-00H002	MCP1650-H002E26	Nvidia Passive Copper cable,up to 200Gbps, QSFP56 to QSFP56, 2m	HVM
HDR	200GE	980-9I54A-00H00A	MCP1650-H00AE30	Nvidia Passive Copper cable, up to 200Gbps, QSFP56 to QSFP56, 0.5m	HVM
HDR	200GE	980-9I54B-00H01A	MCP1650-H01AE30	Nvidia Passive Copper cable, up to 200Gbps, QSFP56 to QSFP56, 1.5 m	HVM
N/A	200GE	980-9I54C-00V001	MCP1650-V001E30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 1m, black pulltab, 30AWG	LTB [HVM]
N/A	200GE	980-9I54D-00V002	MCP1650-V002E26	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 2m, black pulltab, 26AWG	LTB [HVM]
N/A	200GE	980-9I54G-00V003	MCP1650-V003E26	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 3m, black pulltab, 26AWG	EOL [HVM]
N/A	200GE	980-9I54H-00V00A	MCP1650-V00AE30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 0.5m, black pulltab, 30AWG	LTB [HVM]
N/A	200GE	980-9I54I-00V01A	MCP1650-V01AE30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 1.5m, black pulltab, 30AWG	LTB [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	200GE	980-9I54L-00V02A	MCP1650-V02AE26	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 2.5m, black pulltab, 26AWG	LTB [HVM]
HDR	200GE	980-9I39E-00H001	MCP7H50-H001R30	Nvidia Passive copper splitter cable, 200Gbps to 2x100Gbps, QSFP56 to 2xQSFP56, 1m	HVM
HDR	200GE	980-9I99F-00H002	MCP7H50-H002R26	Nvidia Passive copper splitter cable, 200Gbps to 2x100Gbps, QSFP56 to 2xQSFP56, 2m	HVM
HDR	200GE	980-9I98G-00H01A	MCP7H50-H01AR30	Nvidia Passive copper splitter cable, 200Gbps to 2x100Gbps, QSFP56 to 2xQSFP56, 1.5m	HVM
N/A	200GE	980-9I98H-00V001	MCP7H50-V001R30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 1m, 30AWG	LTB [HVM]
N/A	200GE	980-9I98I-00V002	MCP7H50-V002R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 2m, 26AWG	LTB [HVM]
N/A	200GE	980-9I98J-00V003	MCP7H50-V003R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 3m, 26AWG	EOL [HVM]
N/A	200GE	980-9I98K-00V01A	MCP7H50-V01AR30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 1.5m, 30AWG	EOL [HVM]
N/A	200GE	980-9I98M-00V02A	MCP7H50-V02AR26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 2.5m, 26AWG	LTB [HVM]
N/A	200GE	980-9IA3X-00V001	MCP7H70-V001R30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 4x50Gb/s, QSFP56 to 4xSFP56, colored, 1m, 30AWG	EOL [P-Rel]
N/A	200GE	980-9IA3Y-00V002	MCP7H70-V002R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 4x50Gb/s, QSFP56 to 4xSFP56, colored, 2m, 26AWG	EOL [P-Rel]
N/A	200GE	980-9I43Z-00V003	MCP7H70-V003R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 4x50Gb/s, QSFP56 to 4x4SFP56, colored, 3m, 26AWG	EOL [P-Rel]
N/A	200GE	980-9I430-00V01A	MCP7H70-V01AR30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 4x50Gb/s, QSFP56 to 4xSFP56, colored, 1.5m, 30AWG	EOL [P-Rel]
N/A	200GE	980-9I431-00V02A	MCP7H70-V02AR26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to4x50Gb/s, QSFP56 to 4xSFP56, colored, 2.5m, 26AWG	EOL [P-Rel]
HDR	200GE	980-9I46K-00H001	MCP7Y60-H001	NVIDIA passive copper splitter cable, 400(2x200)Gbps to 2x200Gbps, OSFP to 2xQSFP56, 1m, fin to flat	MP
HDR	200GE	980-9I46L-00H002	MCP7Y60-H002	NVIDIA passive copper splitter cable, 400(2x200)Gbps to 2x200Gbps, OSFP to 2xQSFP56, 2m, fin to flat	MP
HDR	200GE	980-9I93M-00H01A	MCP7Y60-H01A	NVIDIA passive copper splitter cable, 400(2x200)Gbps to 2x200Gbps, OSFP to 2xQSFP56, 1.5m, fin to flat	MP

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
HDR	200GE	980-9I93N-00H001	MCP7Y70-H001	NVIDIA passive copper splitter cable, 400(2x200)Gbps to 4x100Gbps, OSFP to 4xQSFP56, 1m, fin to flat	MP
HDR	200GE	980-9I93O-00H002	MCP7Y70-H002	NVIDIA passive copper splitter cable, 400(2x200)Gbps to 4x100Gbps, OSFP to 4xQSFP56, 2m, fin to flat	MP
HDR	200GE	980-9I47P-00H01A	MCP7Y70-H01A	NVIDIA passive copper splitter cable, 400(2x200)Gbps to 4x100Gbps, OSFP to 4xQSFP56, 1.5m, fin to flat	MP
HDR	N/A	980-9I124-00H003	MFS1S00-H003E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 3m	EOL [HVM]
HDR	200GE	980-9I457-00H003	MFS1S00-H003V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 3m	MP
HDR	N/A	980-9I45A-00H005	MFS1S00-H005E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 5m	EOL [HVM]
HDR	200GE	980-9I45D-00H005	MFS1S00-H005V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 5m	MP
HDR	N/A	980-9I45G-00H010	MFS1S00-H010E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 10m	EOL [HVM]
HDR	200GE	980-9I45J-00H010	MFS1S00-H010V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 10m	MP
HDR	N/A	980-9I45M-00H015	MFS1S00-H015E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 15m	EOL [HVM]
HDR	200GE	980-9I45O-00H015	MFS1S00-H015V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 15m	MP
HDR	N/A	980-9I45R-00H020	MFS1S00-H020E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 20m	EOL [HVM]
HDR	200GE	980-9I45T-00H020	MFS1S00-H020V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 20m	MP
HDR	N/A	980-9I45Y-00H030	MFS1S00-H030E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 30m	EOL [HVM]
HDR	200GE	980-9I440-00H030	MFS1S00-H030V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 30m	MP
HDR	N/A	980-9I455-00H050	MFS1S00-H050E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 50m	EOL [HVM]
HDR	200GE	980-9I447-00H050	MFS1S00-H050V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 50m	MP
HDR	N/A	980-9I44G-00H100	MFS1S00-H100E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 100m	EOL [HVM]
HDR	200GE	980-9I44H-00H100	MFS1S00-H100V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 100m	MP
HDR	N/A	980-9I44I-00H130	MFS1S00-H130E	NVIDIA active fiber cable, IB HDR, up to 200Gb/s, QSFP56, LSZH, black pulltab, 130m	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
HDR	200GE	980-9I44K-00H130	MFS1S00-H130V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 130m	MP
HDR	200GE	980-9I44N-00H150	MFS1S00-H150V	Nvidia active optical cable, up to 200Gbps , QSFP56 to QSFP56, 150m	MP
N/A	200GE	980-9I44P-00V003	MFS1S00-V003E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 3m	LTB [HVM]
N/A	200GE	980-9I45Q-00V005	MFS1S00-V005E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 5m	LTB [HVM]
N/A	200GE	980-9I45R-00V010	MFS1S00-V010E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 10m	LTB [HVM]
N/A	200GE	980-9I44S-00V015	MFS1S00-V015E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 15m	LTB [HVM]
N/A	200GE	980-9I44T-00V020	MFS1S00-V020E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 20m	LTB [HVM]
N/A	200GE	980-9I44U-00V030	MFS1S00-V030E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 30m	LTB [HVM]
N/A	200GE	980-9I44V-00V050	MFS1S00-V050E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 50m	LTB [HVM]
N/A	200GE	980-9I44W-00V100	MFS1S00-V100E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 100m	EOL [HVM] [HIBERN/ ATE]
HDR	N/A	980-9I452-00H003	MFS1S50-H003E	NVIDIA active fiber splitter cable, IB HDR, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56 , LSZH, 3m	EOL [HVM]
HDR	200GE	980-9I445-00H003	MFS1S50-H003V	Nvidia active optical splitter cable, 200Gbps to 2x100Gbps , QSFP56 to 2x QSFP56, 3m	HVM
HDR	N/A	980-9I956-00H005	MFS1S50-H005E	NVIDIA active fiber splitter cable, IB HDR, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56 , LSZH, 5m	EOL [HVM]
HDR	200GE	980-9I969-00H005	MFS1S50-H005V	Nvidia active optical splitter cable, 200Gbps to 2x100Gbps , QSFP56 to 2x QSFP56, 5m	HVM
HDR	N/A	980-9I95A-00H010	MFS1S50-H010E	NVIDIA active fiber splitter cable, IB HDR, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56 , LSZH, 10m	EOL [HVM]
HDR	200GE	980-9I96D-00H010	MFS1S50-H010V	Nvidia active optical splitter cable, 200Gbps to 2x100Gbps , QSFP56 to 2x QSFP56, 10m	HVM
HDR	N/A	980-9I95E-00H015	MFS1S50-H015E	NVIDIA active fiber splitter cable, IB HDR, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56 , LSZH, 15m	EOL [HVM]
HDR	200GE	980-9I96H-00H015	MFS1S50-H015V	Nvidia active optical splitter cable, 200Gbps to 2x100Gbps , QSFP56 to 2x QSFP56, 15m	HVM
HDR	N/A	980-9I95I-00H020	MFS1S50-H020E	NVIDIA active fiber splitter cable, IB HDR, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56 , LSZH, 20m	EOL [HVM]
HDR	200GE	980-9I96L-00H020	MFS1S50-H020V	Nvidia active optical splitter cable, 200Gbps to 2x100Gbps , QSFP56 to 2x QSFP56, 20m	HVM
HDR	N/A	980-9I95M-00H030	MFS1S50-H030E	NVIDIA active fiber splitter cable, IB HDR, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56 , LSZH, 30m	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
HDR	200GE	980-9I96P-00H030	MFS1S50-H030V	Nvidia active optical splitter cable, 200Gbps to 2x100Gbps , QSFP56 to 2x QSFP56, 30m	HVM
HDR	200GE	980-9I95S-00H040	MFS1S50-H040V	Nvidia active optical splitter cable, 200Gbps to 2x100Gbps , QSFP56 to 2x QSFP56, 40m	Prototype
HDR	200GE	980-9I95T-00H050	MFS1S50-H050V	Nvidia active optical splitter cable, 200Gbps to 2x100Gbps , QSFP56 to 2x QSFP56, 50m	Prototype
N/A	200GE	980-9I95Q-00V003	MFS1S50-V003E	NVIDIA active fiber splitter cable, 200GbE, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, LSZH, black pulltab, 3m	EOL [HVM]
N/A	200GE	980-9I96R-00V005	MFS1S50-V005E	NVIDIA active fiber splitter cable, 200GbE, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, LSZH, black pulltab, 5m	EOL [HVM]
N/A	200GE	980-9I96S-00V010	MFS1S50-V010E	NVIDIA active fiber splitter cable, 200GbE, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, LSZH, black pulltab, 10m	EOL [HVM]
N/A	200GE	980-9I96T-00V015	MFS1S50-V015E	NVIDIA active fiber splitter cable, 200GbE, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, LSZH, black pulltab, 15m	EOL [HVM]
N/A	200GE	980-9I95U-00V020	MFS1S50-V020E	NVIDIA active fiber splitter cable, 200GbE, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, LSZH, black pulltab, 20m	EOL [HVM]
N/A	200GE	980-9I95V-00V030	MFS1S50-V030E	NVIDIA active fiber splitter cable, 200GbE, 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, LSZH, black pulltab, 30m	EOL [HVM]
HDR	N/A	980-9I961-00H010	MFS1S90-H010E	NVIDIA active fiber splitter cable, IB HDR, 2x200Gb/s to 2x200Gb/s, 2xQSFP56 to 2xQSFP56 , LSZH, 10m	LTB [HVM]
HDR	N/A	980-9I423-00H020	MFS1S90-H020E	NVIDIA active fiber splitter cable, IB HDR, 2x200Gb/s to 2x200Gb/s, 2xQSFP56 to 2xQSFP56 , LSZH, 20m	LTB [HVM]
HDR	N/A	980-9I424-00H030	MFS1S90-H030E	NVIDIA active fiber splitter cable, IB HDR, 2x200Gb/s to 2x200Gb/s, 2xQSFP56 to 2xQSFP56 , LSZH, 30m	EOL [HVM]
HDR	N/A	980-9I17S-00HS00	MMA1T00-HS	NVIDIA transceiver, HDR, QSFP56, MPO, 850nm, SR4, up to 100m	HVM
N/A	200GE	980-9I20T-00V000	MMA1T00-VS	NVIDIA transceiver, 200GbE, up to 200Gb/s, QSFP56, MPO, 850nm, SR4, up to 100m	HVM

2.5.2.3 EDR / 100GbE Cables

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	100GE	980-9I62O-00C001	MCP1600-C001	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 1m 30AWG	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	100GE	980-9I620-00C001	MCP1600-C001E30N	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 1m, Black, 30AWG, CA-N	HVM
N/A	100GE	980-9I62S-00C001	MCP1600-C001LZ	NVIDIA Passive Copper Cable, ETH 100GbE, 100Gb/s, QSFP, 1m, LSZH, 30AWG	EOL [MP]
N/A	100GE	980-9I621-00C002	MCP1600-C002	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 2m 30AWG	EOL [HVM]
N/A	100GE	980-9I622-00C002	MCP1600-C002E26N	NVIDIA® Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 2m, Black, 26AWG, CA-N	Preliminary
N/A	100GE	980-9I62V-00C002	MCP1600-C002E30N	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 2m, Black, 30AWG, CA-N	HVM
N/A	100GE	980-9I62X-00C003	MCP1600-C003	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 3m 28AWG	EOL [HVM]
N/A	100GE	980-9I62Z-00C003	MCP1600-C003E26N	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 3m, Black, 26AWG, CA-N	EOL [HVM]
N/A	100GE	980-9I620-00C003	MCP1600-C003E30L	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 3m, Black, 30AWG, CA-L	HVM
N/A	100GE	980-9I622-00C003	MCP1600-C003LZ	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP, 3m, LSZH, 26AWG	EOL [MP]
N/A	100GE	980-9I625-00C005	MCP1600-C005E26L	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 5m, Black, 26AWG, CA-L	HVM
N/A	100GE	980-9I626-00C00A	MCP1600-C00A	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 0.5m 30AWG	EOL [HVM]
N/A	100GE	980-9I627-00C00A	MCP1600-C00AE30N	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 0.5m, Black, 30AWG, CA-N	EOL [HVM]
N/A	100GE	980-9I629-00C00B	MCP1600-C00BE30N	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 0.75m, Black, 30AWG, CA-N	EOL [HVM]
N/A	100GE	980-9I62B-00C01A	MCP1600-C01A	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 1.5m 30AWG	EOL [HVM]
N/A	100GE	980-9I62C-00C01A	MCP1600-C01AE30N	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 1.5m, Black, 30AWG, CA-N	HVM
N/A	100GE	980-9I62G-00C02A	MCP1600-C02A	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 2.5m 30AWG	EOL [HVM]
N/A	100GE	980-9I62H-00C02A	MCP1600-C02AE26N	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28, 2.5m, Black, 26AWG, CA-N	EOL [HVM]
N/A	100GE	980-9I62I-00C02A	MCP1600-C02AE30L	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP28,2.5m, Black, 30AWG, CA-L	HVM
N/A	100GE	980-9I62M-00C03A	MCP1600-C03A	NVIDIA Passive Copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 3.5m 26AWG	EOL [P-Rel]
EDR	100GE	980-9I62P-00C001	MCP1600-E001	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 1m 30AWG	EOL [HVM]
EDR	N/A	980-9I62Q-00E001	MCP1600-E001E30	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 1m, Black, 30AWG	HVM
EDR	100GE	980-9I62S-00C002	MCP1600-E002	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 2m 28AWG	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
EDR	N/A	980-9I62T-00E002	MCP1600-E002E26	NVIDIA® Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 2m, Black, 26AWG	Preliminary
EDR	N/A	980-9I62U-00E002	MCP1600-E002E30	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 2m, Black, 30AWG	HVM
EDR	100GE	980-9I62V-00C003	MCP1600-E003	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 3m 26AWG	EOL [HVM]
EDR	N/A	980-9I62W-00E003	MCP1600-E003E26	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 3m, Black, 26AWG	HVM
EDR	N/A	980-9I62Y-00E004	MCP1600-E004E26	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 4m, Black, 26AWG	EOL [HVM]
EDR	N/A	980-9I62Z-00E005	MCP1600-E005E26	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 5m, Black, 26AWG	HVM
EDR	N/A	980-9I620-00E00A	MCP1600-E00A	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 0.5m 30AWG	EOL [HVM]
EDR	N/A	980-9I621-00E00A	MCP1600-E00AE30	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 0.5m, Black, 30AWG	EOL [HVM]
EDR	N/A	980-9I622-00E00B	MCP1600-E00BE30	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 0.75m, Black, 30AWG	EOL [HVM] [HIBERN/ ATE]
EDR	100GE	980-9I623-00C01A	MCP1600-E01A	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 1.5m 30AWG	EOL [HVM]
EDR	N/A	980-9I624-00E01A	MCP1600-E01AE30	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 1.5m, Black, 30AWG	HVM
EDR	N/A	980-9I625-00E01C	MCP1600-E01BE30	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 1.25m, Black, 30AWG	EOL [HVM] [HIBERN/ ATE]
EDR	100GE	980-9I626-00C02A	MCP1600-E02A	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 2.5m 26AWG	EOL [HVM]
EDR	N/A	980-9I627-00E02A	MCP1600-E02AE26	NVIDIA Passive Copper cable, IB EDR, up to 100Gb/s, QSFP28, 2.5m, Black, 26AWG	HVM
N/A	100GE	980-9I645-00C001	MCP7F00-A001R	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, colored pulltabs, 1m, 30AWG	EOL [HVM]
N/A	100GE	980-9I486-00C001	MCP7F00-A001R30N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 1m, Colored, 30AWG, CA-N	LTB [HVM]
N/A	100GE	980-9I48A-00C002	MCP7F00-A002R	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, colored pulltabs, 2m, 30AWG	EOL [HVM]
N/A	100GE	980-9I48B-00C002	MCP7F00-A002R30N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 2m, Colored, 30AWG, CA-N	LTB [HVM]
N/A	100GE	980-9I48G-00C003	MCP7F00-A003R26N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 3m, Colored, 26AWG, CA-N	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	100GE	980-9I48H-00C003	MCP7F00-A003R30L	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 3m, Colored, 30AWG, CA-L	LTB [HVM]
N/A	100GE	980-9I48J-00C005	MCP7F00-A005R26L	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 5m, Colored, 26AWG, CA-L	LTB [HVM]
N/A	100GE	980-9I48M-00C01A	MCP7F00-A01AR	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, colored pulltabs,1.5m, 30AWG	EOL [HVM]
N/A	100GE	980-9I48N-00C01A	MCP7F00-A01AR30N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 1.5m, Colored, 30AWG, CA-N	LTB [HVM]
N/A	100GE	980-9I48S-00C02A	MCP7F00-A02AR26N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 2.5m, Colored, 26AWG, CA-N	EOL [HVM]
N/A	100GE	980-9I48T-00C02A	MCP7F00-A02AR30L	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 2.5m, Colored, 30AWG, CA-L	LTB [HVM]
N/A	100GE	980-9I48U-00C02A	MCP7F00-A02ARLZ	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 2.5m, LSZH, Colored, 28AWG	EOL [P-Rel]
N/A	100GE	980-9I48X-00C03A	MCP7F00-A03AR26L	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 3.5m, Colored, 26AWG, CA-L	EOL [HVM]
N/A	100GE	980-9I61C-00C005	MCP7H00-G00000	NVIDIA® passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 5m, Colored, 26AWG, CA-L	Preliminary
N/A	100GE	980-9I61D-00C001	MCP7H00-G001	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 1m, 30AWG	EOL [HVM]
N/A	100GE	980-9I99F-00C001	MCP7H00-G001R	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pulltabs, 1m, 30AWG	EOL [HVM]
N/A	100GE	980-9I99G-00C001	MCP7H00-G001R30N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 1m, Colored, 30AWG, CA-N	LTB [HVM]
N/A	100GE	980-9I99J-00C002	MCP7H00-G002R	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pulltabs, 2m, 30AWG	EOL [HVM]
N/A	100GE	980-9I99K-00C002	MCP7H00-G002R26N	NVIDIA® passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 2m, Colored, 26AWG, CA-N	Preliminary
N/A	100GE	980-9I99L-00C002	MCP7H00-G002R30N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 2m, Colored, 30AWG, CA-N	LTB [HVM]
N/A	100GE	980-9I99O-00C003	MCP7H00-G003R	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pulltabs, 3m, 28AWG	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	100GE	980-9I99Q-00C003	MCP7H00-G003R26N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 3m, Colored, 26AWG, CA-N	EOL [HVM]
N/A	100GE	980-9I39R-00C003	MCP7H00-G003R30L	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 3m, Colored, 30AWG, CA-L	LTB [HVM]
N/A	100GE	980-9I99S-00C004	MCP7H00-G004R26L	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 4m, Colored, 26AWG, CA-L	EOL [HVM]
N/A	100GE	980-9I99W-00C01A	MCP7H00-G01AR	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pulltabs, 1.5m, 30AWG	EOL [HVM]
N/A	100GE	980-9I99X-00C01A	MCP7H00-G01AR30N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 1.5m, Colored, 30AWG, CA-N	LTB [HVM]
N/A	100GE	980-9I992-00C02A	MCP7H00-G02AR	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pulltabs, 2.5m, 30AWG	EOL [HVM]
N/A	100GE	980-9I994-00C02A	MCP7H00-G02AR26N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 2.5m, Colored, 26AWG, CA-N	EOL [HVM]
N/A	100GE	980-9I395-00C02A	MCP7H00-G02AR30L	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 2.5m, Colored, 30AWG, CA-L	LTB [HVM]
N/A	100GE	980-9I13S-00C003	MFA1A00-C003	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 3m	HVM
N/A	100GE	980-9I13X-00C005	MFA1A00-C005	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 5m	HVM
N/A	100GE	980-9I134-00C010	MFA1A00-C010	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 10m	HVM
N/A	100GE	980-9I13A-00C015	MFA1A00-C015	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 15m	HVM
N/A	100GE	980-9I13F-00C020	MFA1A00-C020	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 20m	HVM
N/A	100GE	980-9I13N-00C030	MFA1A00-C030	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 30m	HVM
N/A	100GE	980-9I130-00C050	MFA1A00-C050	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 50m	HVM
N/A	100GE	980-9I13B-00C100	MFA1A00-C100	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 100m	LTB [HVM]
EDR	N/A	980-9I13D-00E001	MFA1A00-E001	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 1m	HVM
EDR	N/A	980-9I13F-00E003	MFA1A00-E003	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 3m	HVM

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
EDR	N/A	980-9I13J-00E005	MFA1A00-E005	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 5m	HVM
EDR	N/A	980-9I13M-00E007	MFA1A00-E007	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 7m	LTB [HVM]
EDR	N/A	980-9I13O-00E010	MFA1A00-E010	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 10m	HVM
EDR	N/A	980-9I13S-00E015	MFA1A00-E015	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 15m	HVM
EDR	N/A	980-9I13V-00E020	MFA1A00-E020	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 20m	HVM
EDR	N/A	980-9I13Y-00E030	MFA1A00-E030	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 30m	HVM
EDR	N/A	980-9I133-00E050	MFA1A00-E050	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 50m	HVM
EDR	N/A	980-9I135-00E100	MFA1A00-E100	NVIDIA active fiber cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 100m	LTB [HVM]
N/A	100GE	980-9I37H-00C003	MFA7A20-C003	NVIDIA active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 3m	EOL [HVM]
N/A	100GE	980-9I37I-00C005	MFA7A20-C005	NVIDIA active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 5m	EOL [HVM]
N/A	100GE	980-9I40J-00C010	MFA7A20-C010	NVIDIA active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 10m	EOL [HVM]
N/A	100GE	980-9I40K-00C020	MFA7A20-C020	NVIDIA active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 20m	EOL [HVM]
N/A	100GE	980-9I40L-00C002	MFA7A20-C02A	NVIDIA® active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 2.5m	Preliminary
N/A	100GE	980-9I40M-00C003	MFA7A20-C03A	NVIDIA® active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 3.5m	Preliminary
N/A	100GE	980-9I40N-00C003	MFA7A50-C003	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 3m	EOL [HVM]
N/A	100GE	980-9I40O-00C005	MFA7A50-C005	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 5m	EOL [HVM]
N/A	100GE	980-9I49P-00C010	MFA7A50-C010	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 10m	EOL [HVM]
N/A	100GE	980-9I49Q-00C015	MFA7A50-C015	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 15m	EOL [HVM]
N/A	100GE	980-9I49R-00C020	MFA7A50-C020	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 20m	EOL [HVM]
N/A	100GE	980-9I49S-00C030	MFA7A50-C030	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 30m	EOL [HVM]
N/A	100GE	980-9I149-00CS00	MMA1B00-C100D	NVIDIA transceiver, 100GbE, QSFP28, MPO, 850nm, SR4, up to 100m, DDMI	HVM
N/A	100GE	980-9I17D-00CS00	MMA1B00-C100T	NVIDIA® transceiver, 100GbE, QSFP28, MPO, 850nm, up to 100m, OTU4	Preliminary

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
EDR	N/A	980-9I17L-00E000	MMA1B00-E100	NVIDIA transceiver, IB EDR, up to 100Gb/s, QSFP28, MPO, 850nm, SR4, up to 100m	HVM
N/A	100GE	980-9I17P-00CR00	MMA1L10-CR	NVIDIA optical transceiver, 100GbE, 100Gb/s, QSFP28, LC-LC, 1310nm, LR4 up to 10km	HVM
N/A	100GE	980-9I17Q-00CM00	MMA1L30-CM	NVIDIA optical module, 100GbE, 100Gb/s, QSFP28, LC-LC, 1310nm, CWDM4, up to 2km	MP
N/A	100GE	980-9I16X-00C000	MMS1C10-CM	NVIDIA active optical module, 100Gb/s, QSFP, MPO, 1310nm, PSM4, up to 500m	EOL [MP]
N/A	100GE	980-9I53X-00C000	SPQ-CE-ER-CDFL-M	40km 100G QSFP28 ER Optical Transceiver	P-Rel
N/A	100GE	980-9I63F-00CM00	X65406	NVIDIA® optical module, 100GbE, 100Gb/s, QSFP28, LC-LC, 1310nm, CWDM4, up to 2km	Preliminary

2.5.2.4 FDR / 56GbE Cables

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
FDR	56GE	980-9I679-00L004	MC2207126-004	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 4m	EOL [HVM]
FDR	56GE	980-9I67A-00L003	MC2207128-003	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 3m	EOL [HVM]
FDR	56GE	980-9I67C-00L02A	MC2207128-0A2	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 2.5m	EOL [MP]
FDR	56GE	980-9I67D-00L001	MC2207130-001	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 1m	EOL [HVM]
FDR	56GE	980-9I67E-00L002	MC2207130-002	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 2m	EOL [HVM]
FDR	56GE	980-9I67F-00L00A	MC2207130-00A	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 0.5m	EOL [HVM]
FDR	56GE	980-9I67G-00L01A	MC2207130-0A1	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 1.5m	EOL [HVM]
FDR	56GE	980-9I15U-00L003	MC220731V-003	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 3m	EOL [HVM]
FDR	56GE	980-9I15V-00L005	MC220731V-005	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 5m	EOL [HVM]
FDR	56GE	980-9I15W-00L010	MC220731V-010	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 10m	EOL [HVM]
FDR	56GE	980-9I15X-00L015	MC220731V-015	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 15m	EOL [HVM]
FDR	56GE	980-9I15Y-00L020	MC220731V-020	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 20m	EOL [HVM]
FDR	56GE	980-9I15Z-00L025	MC220731V-025	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 25m	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
FDR	56GE	980-9I150-00L030	MC220731V-030	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 30m	EOL [HVM]
FDR	56GE	980-9I151-00L040	MC220731V-040	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 40m	EOL [HVM] [HIBERN/ATE]
FDR	56GE	980-9I152-00L050	MC220731V-050	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 50m	EOL [HVM]
FDR	56GE	980-9I153-00L075	MC220731V-075	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 75m	EOL [HVM]
FDR	56GE	980-9I154-00L100	MC220731V-100	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 100m	EOL [HVM]
FDR	56GE	980-9I675-00L001	MCP170L-F001	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 1m	EOL [P-Rel]
FDR	56GE	980-9I676-00L002	MCP170L-F002	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 2m	EOL [P-Rel]
FDR	56GE	980-9I677-00L003	MCP170L-F003	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 3m	EOL [P-Rel] [HIBERN/ATE]
FDR	56GE	980-9I678-00L00A	MCP170L-F00A	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 0.5m	EOL [P-Rel]
FDR	56GE	980-9I679-00L01A	MCP170L-F01A	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 1.5m	EOL [P-Rel] [HIBERN/ATE]
FDR	N/A	980-9I17M-00FS00	MMA1B00-F030D	NVIDIA transceiver, FDR, QSFP+, MPO, 850nm, SR4, up to 30m, DDMI	LTB [HVM]

2.5.2.5 25GbE Cables

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
NA	25GE	980-9I78I-00A000	MAM1Q00A-QSA28	Mellanox cable module, ETH 25GbE, 100Gb/s to 25Gb/s, QSFP28 to SFP28	HVM
NA	25GE	980-9I63J-00A001	MCP2M00-A001	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 1m, 30AWG	EOL [HVM]
NA	25GE	980-9I63L-00A001	MCP2M00-A001E30N	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 1m, Black, 30AWG, CA-N	LTB [HVM]
NA	25GE	980-9I63N-00A002	MCP2M00-A002E26N	Mellanox® Passive Copper cable, ETH, up to 25Gb/s, SFP28, 2m, Black, 26AWG, CA-N	Preliminary
NA	25GE	980-9I63O-00A002	MCP2M00-A002E30N	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 2m, Black, 30AWG, CA-N	LTB [HVM]
NA	25GE	980-9I63R-00A003	MCP2M00-A003E26N	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 3m, Black, 26AWG, CA-N	EOL [HVM]
NA	25GE	980-9I63S-00A003	MCP2M00-A003E30L	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 3m, Black, 30AWG, CA-L	LTB [HVM]
NA	25GE	980-9I63T-00A004	MCP2M00-A004E26L	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 4m, Black, 26AWG, CA-L	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
NA	25GE	980-9I63V-00A005	MCP2M00-A005E26L	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 5m, Black, 26AWG, CA-L	LTB [HVM]
NA	25GE	980-9I63W-00A00A	MCP2M00-A00A	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 0.5m, 30AWG	EOL [HVM]
NA	25GE	980-9I63X-00A00A	MCP2M00-A00AE30N	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 0.5m, Black, 30AWG, CA-N	EOL [HVM]
NA	25GE	980-9I63Z-00A01A	MCP2M00-A01AE30N	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 1.5m, Black, 30AWG, CA-N	LTB [HVM]
NA	25GE	980-9I631-00A02A	MCP2M00-A02AE26N	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 2.5m, Black, 26AWG, CA-N	EOL [HVM]
NA	25GE	980-9I632-00A02A	MCP2M00-A02AE30L	Mellanox Passive Copper cable, ETH, up to 25Gb/s, SFP28, 2.5m, Black, 30AWG, CA-L	LTB [HVM]
NA	25GE	980-9IA1T-00A003	MFA2P10-A003	Mellanox active optical cable 25GbE, SFP28, 3m	EOL [HVM]
NA	25GE	980-9I53W-00A005	MFA2P10-A005	Mellanox active optical cable 25GbE, SFP28, 5m	EOL [HVM]
NA	25GE	980-9I53Z-00A007	MFA2P10-A007	Mellanox active optical cable 25GbE, SFP28, 7m	EOL [HVM]
NA	25GE	980-9I532-00A010	MFA2P10-A010	Mellanox active optical cable 25GbE, SFP28, 10m	EOL [HVM]
NA	25GE	980-9I535-00A015	MFA2P10-A015	Mellanox active optical cable 25GbE, SFP28, 15m	EOL [HVM]
NA	25GE	980-9I536-00A020	MFA2P10-A020	Mellanox active optical cable 25GbE, SFP28, 20m	EOL [HVM]
NA	25GE	980-9I539-00A030	MFA2P10-A030	Mellanox active optical cable 25GbE, SFP28, 30m	EOL [HVM]
NA	25GE	980-9I53A-00A050	MFA2P10-A050	Mellanox active optical cable 25GbE, SFP28, 50m	EOL [HVM]
NA	25GE	980-9I094-00AR00	MMA2L20-AR	Mellanox optical transceiver, 25GbE, 25Gb/s, SFP28, LC-LC, 1310nm, LR up to 10km	MP
NA	25GE	980-9I595-00AM00	MMA2P00-AS	Mellanox transceiver, 25GbE, SFP28, LC-LC, 850nm, SR	HVM

2.5.2.6 10GbE Cables

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	10GE	980-9I71G-00J000	MAM1Q00A-QSA	NVIDIA cable module, ETH 10GbE, 40Gb/s to 10Gb/s, QSFP to SFP+	HVM
N/A	10GE	980-9I65P-00J005	MC2309124-005	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 5m	EOL [P-Rel]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	10GE	980-9I65Q-00J007	MC2309124-007	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 7m	EOL [P-Rel]
N/A	10GE	980-9I65R-00J001	MC2309130-001	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 1m	EOL [HVM]
N/A	10GE	980-9I65S-00J002	MC2309130-002	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 2m	EOL [HVM]
N/A	10GE	980-9I65T-00J003	MC2309130-003	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 3m	EOL [HVM]
N/A	10GE	980-9I65U-00J00A	MC2309130-00A	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 0.5m	EOL [HVM] [HIBERN/ATE]
N/A	10GE	980-9I682-00J004	MC3309124-004	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 4m	EOL [HVM]
N/A	10GE	980-9I683-00J005	MC3309124-005	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 5m	EOL [HVM]
N/A	10GE	980-9I684-00J006	MC3309124-006	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 6m	EOL [HVM]
N/A	10GE	980-9I685-00J007	MC3309124-007	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 7m	EOL [HVM]
N/A	10GE	980-9I686-00J001	MC3309130-001	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1m	EOL [HVM]
N/A	10GE	980-9I688-00J002	MC3309130-002	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2m	EOL [HVM]
N/A	10GE	980-9I68B-00J003	MC3309130-003	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 3m	EOL [HVM]
N/A	10GE	980-9I68F-00J00A	MC3309130-00A	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 0.5m	EOL [HVM]
N/A	10GE	980-9I68G-00J01A	MC3309130-0A1	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1.5m	EOL [HVM]
N/A	10GE	980-9I68H-00J02A	MC3309130-0A2	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2.5m	EOL [HVM]
N/A	10GE	980-9I68A-00J001	MCP2100-X001B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1m, Blue Pulltab, Connector Label	EOL [HVM] [HIBERN/ATE]
N/A	10GE	980-9I68B-00J002	MCP2100-X002B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2m, Blue Pulltab, Connector Label	EOL [HVM] [HIBERN/ATE]
N/A	10GE	980-9I68C-00J003	MCP2100-X003B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 3m, Blue Pulltab, Connector Label	EOL [HVM]
N/A	10GE	980-9I68E-00J001	MCP2104-X001B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1m, Black Pulltab, Connector Label	EOL [HVM] [HIBERN/ATE]
N/A	10GE	980-9I68F-00J002	MCP2104-X002B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2m, Black Pulltab, Connector Label	EOL [HVM]

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	10GE	980-9I68G-00J003	MCP2104-X003B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 3m, Black Pulltab, Connector Label	EOL [HVM]
N/A	10GE	980-9I68H-00J01A	MCP2104-X01AB	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1.5m, Black Pulltab, Connector Label	EOL [HVM]
N/A	10GE	980-9I68I-00J02A	MCP2104-X02AB	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2.5m, Black Pulltab, Connector Label	EOL [HVM]
N/A	10GE	930-9O000-0000-343	MFM1T02A-LR	NVIDIA SFP+ optical module for 10GBASE-LR	HVM
N/A	10GE	MFM1T02A-LR-F	MFM1T02A-LR-F	NVIDIA optical module, ETH 10GbE, 10Gb/s, SFP+, LC-LC, 1310nm, LR up to 10km	HVM
N/A	10GE	930-9O000-0000-409	MFM1T02A-SR	NVIDIA SFP+ optical module for 10GBASE-SR	HVM
N/A	10GE	MFM1T02A-SR-F	MFM1T02A-SR-F	NVIDIA optical module, ETH 10GbE, 10Gb/s, SFP+, LC-LC, 850nm, SR up to 300m	HVM
N/A	10GE	MFM1T02A-SR-P	MFM1T02A-SR-P	NVIDIA optical module, ETH 10GbE, 10Gb/s, SFP+, LC-LC, 850nm, SR up to 300m	HVM

2.5.2.7 1GbE Cables

IB Data Rate	Eth Data Rate	NVIDIA P/N	Legacy P/N	Description	LifeCycle Phase
N/A	1GE	980-9I270-00IM00	MC3208011-SX	NVIDIA Optical module, ETH 1GbE, 1Gb/s, SFP, LC-LC, SX 850nm, up to 500m	EOL [P-Rel]
N/A	1GE	980-9I251-00IS00	MC3208411-T	NVIDIA module, ETH 1GbE, 1Gb/s, SFP, Base-T, up to 100m	HVM

2.5.2.8 Supported 3rd Party Cables and Modules

Speed	Cable OPN	Description
400GbE	DME8811-EC07	400G-2x200G split 7M AOC cables (400G QSFP-DD breaking out to 2x 200G QSFP56) (Rev 12)
400GbE	RTXM500-910	400G-2x200G split 10M AOC cables (400G QSFP-DD breaking out to 2x 200G QSFP56) (Rev 10)
200GbE	RTXM500-905	400G-2x200G split 5M AOC cables (400G QSFP-DD breaking out to 2x 200G QSFP56)
100GbE	1AT-3Q4M01XX-12A	O-NET QSFP28 100G Active cable/module
100GbE	AQPMANQ4EDMA0784	QSFP28 100G SMF 500m Transceiver

Speed	Cable OPN	Description
100GbE	CAB-Q-Q-100G-3M	Passive 3 meter, QSFP+ to QSFP+ QSFP100 TWINAX 103.125Gbps-CR4
100GbE	CAB-Q-Q-100GbE-3M	Passive 3 meter , QSFP+ to QSFP+ QSFP100 TWINAX 103.125Gbps-CR4
100GbE	FCBN425QE1C30-C1	100GbE Quadwire® QSFP28 Active Optical Cable 30M
100GbE	FTLC1151RDPL	TRANSCIEVER 100GBE QSFP LR4
100GbE	FTLC9152RGPL	100G 100M QSFP28 SWDM4 OPT TRANS
100GbE	FTLC9555REPM3-E6	100m Parallel MMF 100GQSFP28Optical Transceiver
100GbE	NDAAFJ-C102	SF-NDAAFJ100G-005M
100GbE	QSFP-100G-AOC30M	30m (98ft) Cisco QSFP-100G-AOC30M Compatible 100G QSFP28 Active Optical Cable
100GbE	QSFP28-LR4-AJ	CISCO-PRE 100GbE LR4 QSFP28 Transceiver Module
100GbE	SFBR-89BDDZ-CS2	CISCO-PRE 100G AOM BiDi
100GbE	SQF1002L4LNC101P	Cisco-SUMITOMO 100GbE AOM
40GbE	2231254-2	Cisco 3m 40GbE copper
40GbE	AFBR-7QER15Z-CS1	Cisco 40GbE 15m AOC
40GbE	BN-QS-SP-CBL-5M	PASSIVE COPPER SPLITTER CABLE ETH 40GBE TO 4X10GBE 5M
40GbE	NDCCGJ-C402	15m (49ft) Avago AFBR-7QER15Z Compatible 40G QSFP+ Active Optical Cable
40GbE	QSFP-40G-SR-BD	Cisco 40GBASE-SR-BiDi, duplex MMF

2.5.3 Supported Cables and Modules for BlueField-2

2.5.3.1 NDR / 400GbE Cables

Speed	Part Number	Marketing Description
400GE	MCP1660-W001E30	NVIDIA Direct Attach Copper cable, 400GbE, 400Gb/s, QSFP-DD, 1m, 30AWG
400GE	MCP1660-W002E26	NVIDIA Direct Attach Copper cable, 400GbE, 400Gb/s, QSFP-DD, 2m, 26AWG
400GE	MCP1660-W003E26	NVIDIA Direct Attach Copper cable, 400GbE, 400Gb/s, QSFP-DD, 3m, 26AWG
400GE	MCP1660-W00AE30	NVIDIA Direct Attach Copper cable, 400GbE, 400Gb/s, QSFP-DD, 0.5m, 30AWG
400GE	MCP1660-W01AE30	NVIDIA Direct Attach Copper cable, 400GbE, 400Gb/s, QSFP-DD, 1.5m, 30AWG
400GE	MCP1660-W02AE26	NVIDIA Direct Attach Copper cable, 400GbE, 400Gb/s, QSFP-DD, 2.5m, 26AWG

Speed	Part Number	Marketing Description
400GE	MCP7F60-W001R30	NVIDIA DAC splitter cable, 400GbE, 400Gb/s to 4x100Gb/s, QSFP-DD to 4xQSFP56, 1m, 30AWG
400GE	MCP7F60-W002R26	NVIDIA DAC splitter cable, 400GbE, 400Gb/s to 4x100Gb/s, QSFP-DD to 4xQSFP56, 2m, 26AWG
400GE	MCP7F60-W02AR26	NVIDIA DAC splitter cable, 400GbE, 400Gb/s to 4x100Gb/s, QSFP-DD to 4xQSFP56, 2.5m, 26AWG
400GE	MCP7H60-W001R30	NVIDIA DAC splitter cable, 400GbE, 400Gb/s to 2x200Gb/s, QSFP-DD to 2xQSFP56, 1m, 30AWG
400GE	MCP7H60-W002R26	NVIDIA DAC splitter cable, 400GbE, 400Gb/s to 2x200Gb/s, QSFP-DD to 2xQSFP56, 2m, 26AWG
400GE	MCP7H60-W01AR30	NVIDIA DAC splitter cable, 400GbE, 400Gb/s to 2x200Gb/s, QSFP-DD to 2xQSFP56, 1.5m, 30AWG
400GE	MCP7H60-W02AR26	NVIDIA DAC splitter cable, 400GbE, 400Gb/s to 2x200Gb/s, QSFP-DD to 2xQSFP56, 2.5m, 26AWG

2.5.3.2 HDR / 200GbE Cables

Speed	Part Number	Marketing Description
200GE	MFS1S00-V003E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 3m
200GE	MFS1S00-V005E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 5m
200GE	MFS1S00-V010E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 10m
200GE	MFS1S00-V015E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 15m
200GE	MFS1S00-V020E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 20m
200GE	MFS1S00-V030E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 30m
200GE	MFS1S00-V050E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 50m
200GE	MFS1S00-V100E	NVIDIA active fiber cable, 200GbE, 200Gb/s, QSFP56, LSZH, black pulltab, 100m
200GE	MCP1650-V001E30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 1m, black pulltab, 30AWG
200GE	MCP1650-V002E26	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 2m, black pulltab, 26AWG
200GE	MCP1650-V00AE30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 0.5m, black pulltab, 30AWG
200GE	MCP1650-V01AE30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 1.5m, black pulltab, 30AWG
200GE	MCP1650-V02AE26	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 2.5m, black pulltab, 26AWG
200GE	MCP7H50-V001R30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 1m, 30AWG
200GE	MCP7H50-V002R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 2m, 26AWG

Speed	Part Number	Marketing Description
200GE	MCP7H50-V01AR30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 1.5m, 30AWG
200GE	MCP7H50-V02AR26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 2.5m, 26AWG
200GE	MMA1T00-VS	NVIDIA transceiver, 200GbE, up to 200Gb/s, QSFP56, MPO, 850nm, SR4, up to 100m
200GE	MCP1650-V001E30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 1m, black pulltab, 30AWG
200GE	MCP1650-V002E26	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 2m, black pulltab, 26AWG
200GE	MCP1650-V003E26	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 3m, black pulltab, 26AWG
200GE	MCP1650-V00AE30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 0.5m, black pulltab, 30AWG
200GE	MCP1650-V01AE30	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 1.5m, black pulltab, 30AWG
200GE	MCP1650-V02AE26	NVIDIA Passive Copper cable, 200GbE, 200Gb/s, QSFP56, LSZH, 2.5m, black pulltab, 26AWG
200GE	MCP7H50-V001R30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 1m, 30AWG
200GE	MCP7H50-V002R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 2m, 26AWG
200GE	MCP7H50-V003R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 3m, 26AWG
200GE	MCP7H50-V01AR30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 1.5m, 30AWG
200GE	MCP7H50-V02AR26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 2x100Gb/s, QSFP56 to 2xQSFP56, colored, 2.5m, 26AWG
200GE	MCP7H70-V001R30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 4x50Gb/s, QSFP56 to 4xSFP56, colored, 1m, 30AWG
200GE	MCP7H70-V002R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 4x50Gb/s, QSFP56 to 4xSFP56, colored, 2m, 26AWG
200GE	MCP7H70-V003R26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 4x50Gb/s, QSFP56 to 4x4SFP56, colored, 3m, 26AWG
200GE	MCP7H70-V01AR30	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to 4x50Gb/s, QSFP56 to 4xSFP56, colored, 1.5m, 30AWG
200GE	MCP7H70-V02AR26	NVIDIA passive copper hybrid cable, 200GbE 200Gb/s to4x50Gb/s, QSFP56 to 4xSFP56, colored, 2.5m, 26AWG

2.5.3.3 EDR / 100GbE Cables

Speed	Part Number	Marketing Description
100GbE	MCP1600-C001	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 1m 30AWG
100GbE	MCP1600-C001E30N	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 1m, black, 30AWG, CA-N
100GbE	MCP1600-C001LZ	NVIDIA passive copper Cable, ETH 100GbE, 100Gb/s, QSFP, 1m, LSZH, 30AWG
100GbE	MCP1600-C002	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 2m 30AWG
100GbE	MCP1600-C002E30N	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 2m, black, 30AWG, CA-N
100GbE	MCP1600-C003	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 3m 28AWG
100GbE	MCP1600-C003E26N	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 3m, black, 26AWG, CA-N
100GbE	MCP1600-C003E30L	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 3m, black, 30AWG, CA-L
100GbE	MCP1600-C003LZ	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, 3m, LSZH, 26AWG
100GbE	MCP1600-C005AM	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, 5m, 26AWG
100GbE	MCP1600-C005E26L	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 5m, black, 26AWG, CA-L
100GbE	MCP1600-C00A	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 0.5m 30AWG
100GbE	MCP1600-C00AE30N	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 0.5m, black, 30AWG, CA-N
100GbE	MCP1600-C00BE30N	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 0.75m, black, 30AWG, CA-N
100GbE	MCP1600-C01A	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 1.5m 30AWG
100GbE	MCP1600-C01AE30N	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 1.5m, black, 30AWG, CA-N
100GbE	MCP1600-C02A	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 2.5m 30AWG
100GbE	MCP1600-C02AE26N	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28, 2.5m, black, 26AWG, CA-N
100GbE	MCP1600-C02AE30L	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP28,2.5m, black, 30AWG, CA-L
100GbE	MCP1600-C03A	NVIDIA passive copper cable, ETH 100GbE, 100Gb/s, QSFP, PVC, 3.5m 26AWG
100GbE	MCP1600-E001	NVIDIA passive copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 1m 30AWG

Speed	Part Number	Marketing Description
100GbE	MCP1600-E002	NVIDIA passive copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 2m 28AWG
100GbE	MCP1600-E003	NVIDIA passive copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 3m 26AWG
100GbE	MCP1600-E01A	NVIDIA passive copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 1.5m 30AWG
100GbE	MCP1600-E02A	NVIDIA passive copper cable, IB EDR, up to 100Gb/s, QSFP, LSZH, 2.5m 26AWG
100GbE	MCP7F00-A001R	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, colored pull-tabs, 1m, 30AWG
100GbE	MCP7F00-A001R30N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 1m, colored, 30AWG, CA-N
100GbE	MCP7F00-A002R	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, colored pull-tabs, 2m, 30AWG
100GbE	MCP7F00-A002R30N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 2m, colored, 30AWG, CA-N
100GbE	MCP7F00-A003R26N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 3m, colored, 26AWG, CA-N
100GbE	MCP7F00-A003R30L	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 3m, colored, 30AWG, CA-L
100GbE	MCP7F00-A005R26L	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 5m, colored, 26AWG, CA-L
100GbE	MCP7F00-A01AR	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, colored pull-tabs,1.5m, 30AWG
100GbE	MCP7F00-A01AR30N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 1.5m, colored, 30AWG, CA-N
100GbE	MCP7F00-A02AR26N	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 2.5m, colored, 26AWG, CA-N
100GbE	MCP7F00-A02AR30L	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 2.5m, colored, 30AWG, CA-L
100GbE	MCP7F00-A02ARLZ	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 2.5m, LSZH, colored, 28AWG
100GbE	MCP7F00-A03AR26L	NVIDIA passive copper hybrid cable, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 3.5m, colored, 26AWG, CA-L
100GbE	MCP7H00-G001	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 1m, 30AWG
100GbE	MCP7H00-G001R	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pull-tabs, 1m, 30AWG
100GbE	MCP7H00-G001R30N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 1m, colored, 30AWG, CA-N
100GbE	MCP7H00-G002R	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pull-tabs, 2m, 30AWG
100GbE	MCP7H00-G002R30N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 2m, colored, 30AWG, CA-N

Speed	Part Number	Marketing Description
100GbE	MCP7H00-G003R	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pull-tabs, 3m, 28AWG
100GbE	MCP7H00-G003R26N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 3m, colored, 26AWG, CA-N
100GbE	MCP7H00-G003R30L	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 3m, colored, 30AWG, CA-L
100GbE	MCP7H00-G004R26L	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 4m, colored, 26AWG, CA-L
100GbE	MCP7H00-G01AR	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pull-tabs, 1.5m, 30AWG
100GbE	MCP7H00-G01AR30N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 1.5m, colored, 30AWG, CA-N
100GbE	MCP7H00-G02AR	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, colored pull-tabs, 2.5m, 30AWG
100GbE	MCP7H00-G02AR26N	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 2.5m, colored, 26AWG, CA-N
100GbE	MCP7H00-G02AR30L	NVIDIA passive copper hybrid cable, ETH 100Gb/s to 2x50Gb/s, QSFP28 to 2xQSFP28, 2.5m, colored, 30AWG, CA-L
100GbE	MFA1A00-C003	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 3m
100GbE	MFA1A00-C005	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 5m
100GbE	MFA1A00-C010	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 10m
100GbE	MFA1A00-C015	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 15m
100GbE	MFA1A00-C020	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 20m
100GbE	MFA1A00-C030	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 30m
100GbE	MFA1A00-C050	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 50m
100GbE	MFA1A00-C100	NVIDIA active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 100m
100GbE	MFA7A20-C003	NVIDIA active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 3m
100GbE	MFA7A20-C005	NVIDIA active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 5m
100GbE	MFA7A20-C010	NVIDIA active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 10m
100GbE	MFA7A20-C020	NVIDIA active fiber hybrid solution, ETH 100GbE to 2x50GbE, QSFP28 to 2xQSFP28, 20m
100GbE	MFA7A50-C003	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 3m

Speed	Part Number	Marketing Description
100GbE	MFA7A50-C005	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 5m
100GbE	MFA7A50-C010	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 10m
100GbE	MFA7A50-C015	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 15m
100GbE	MFA7A50-C020	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 20m
100GbE	MFA7A50-C030	NVIDIA active fiber hybrid solution, ETH 100GbE to 4x25GbE, QSFP28 to 4xSFP28, 30m
100GbE	MMA1B00-C100D	NVIDIA transceiver, 100GbE, QSFP28, MPO, 850nm, SR4, up to 100m, DDMI
100GbE	MMA1B00-C100D_FF	NVIDIA transceiver, 100GbE, QSFP28, MPO, 850nm, SR4, up to 100m, DDMI
100GbE	MMA1L10-CR	NVIDIA optical transceiver, 100GbE, 100Gb/s, QSFP28, LC-LC, 1310nm, LR4 up to 10km
100GbE	MMA1L30-CM	NVIDIA optical module, 100GbE, 100Gb/s, QSFP28, LC-LC, 1310nm, CWDM4, up to 2km
100GbE	MMS1C10-CM	NVIDIA active optical module, 100Gb/s, QSFP, MPO, 1310nm, PSM4, up to 500m

2.5.3.4 FDR / 56GbE Cables

Speed	Part Number	Marketing Description
56GbE	MC2207126-004	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 4m
56GbE	MC2207128-003	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 3m
56GbE	MC2207128-0A2	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 2.5m
56GbE	MC2207130-001	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 1m
56GbE	MC2207130-002	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 2m
56GbE	MC2207130-00A	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 0.5m
56GbE	MC2207130-0A1	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 1.5m
56GbE	MC220731V-003	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 3m
56GbE	MC220731V-005	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 5m
56GbE	MC220731V-010	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 10m
56GbE	MC220731V-015	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 15m
56GbE	MC220731V-020	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 20m
56GbE	MC220731V-025	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 25m
56GbE	MC220731V-030	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 30m
56GbE	MC220731V-040	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 40m

Speed	Part Number	Marketing Description
56GbE	MC220731V-050	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 50m
56GbE	MC220731V-075	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 75m
56GbE	MC220731V-100	NVIDIA active fiber cable, VPI, up to 56Gb/s, QSFP, 100m
56GbE	MCP1700-F001C	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 1m, Red pull-tab
56GbE	MCP1700-F001D	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 1m, Yellow pull-tab
56GbE	MCP1700-F002C	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 2m, Red pull-tab
56GbE	MCP1700-F002D	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 2m, Yellow pull-tab
56GbE	MCP1700-F003C	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 3m, Red pull-tab
56GbE	MCP1700-F003D	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, 3m, Yellow pull-tab
56GbE	MCP170L-F001	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 1m
56GbE	MCP170L-F002	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 2m
56GbE	MCP170L-F003	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 3m
56GbE	MCP170L-F00A	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 0.5m
56GbE	MCP170L-F01A	NVIDIA passive copper cable, VPI, up to 56Gb/s, QSFP, LSZH, 1.5m

2.5.3.5 50GbE Cables

Speed	Part Number	Marketing Description
50GE	MAM1Q00A-QSA56	NVIDIA cable module, ETH 50GbE, 200Gb/s to 50Gb/s, QSFP56 to SFP56
50GE	MCP2M50-G001E30	NVIDIA Passive Copper cable, 50GbE, 50Gb/s, SFP56, LSZH, 1m, black pulltab, 30AWG
50GE	MCP2M50-G002E26	NVIDIA Passive Copper cable, 50GbE, 50Gb/s, SFP56, LSZH, 2m, black pulltab, 26AWG
50GE	MCP2M50-G003E26	NVIDIA Passive Copper cable, 50GbE, 50Gb/s, SFP56, LSZH, 3m, black pulltab, 26AWG
50GE	MCP2M50-G00AE30	NVIDIA Passive Copper cable, 50GbE, 50Gb/s, SFP56, LSZH, 0.5m, black pulltab, 30AWG
50GE	MCP2M50-G01AE30	NVIDIA Passive Copper cable, 50GbE, 50Gb/s, SFP56, LSZH, 1.5m, black pulltab, 30AWG
50GE	MCP2M50-G02AE26	NVIDIA Passive Copper cable, 50GbE, 50Gb/s, SFP56, LSZH, 2.5m, black pulltab, 26AWG

2.5.3.6 FDR10 / 40GbE Cables

Speed	Part Number	Marketing Description
40GbE	MC2206128-004	NVIDIA passive copper cable, VPI, up to 40Gb/s, QSFP, 4m
40GbE	MC2206128-005	NVIDIA passive copper cable, VPI, up to 40Gb/s, QSFP, 5m
40GbE	MC2206130-001	NVIDIA passive copper cable, VPI, up to 40Gb/s, QSFP, 1m

Speed	Part Number	Marketing Description
40GbE	MC2206130-002	NVIDIA passive copper cable, VPI, up to 40Gb/s, QSFP, 2m
40GbE	MC2206130-003	NVIDIA passive copper cable, VPI, up to 40Gb/s, QSFP, 3m
40GbE	MC2206130-00A	NVIDIA passive copper cable, VPI, up to 40Gb/s, QSFP, 0.5m
40GbE	MC2210126-004	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 4m
40GbE	MC2210126-005	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 5m
40GbE	MC2210128-003	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 3m
40GbE	MC2210130-001	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 1m
40GbE	MC2210310-003	NVIDIA active fiber cable, ETH 40GbE, 40Gb/s, QSFP, 3m
40GbE	MC2210310-005	NVIDIA active fiber cable, ETH 40GbE, 40Gb/s, QSFP, 5m
40GbE	MC2210310-010	NVIDIA active fiber cable, ETH 40GbE, 40Gb/s, QSFP, 10m
40GbE	MC2210310-015	NVIDIA active fiber cable, ETH 40GbE, 40Gb/s, QSFP, 15m
40GbE	MC2210310-020	NVIDIA active fiber cable, ETH 40GbE, 40Gb/s, QSFP, 20m
40GbE	MC2210310-030	NVIDIA active fiber cable, ETH 40GbE, 40Gb/s, QSFP, 30m
40GbE	MC2210310-050	NVIDIA active fiber cable, ETH 40GbE, 40Gb/s, QSFP, 50m
40GbE	MC2210310-100	NVIDIA active fiber cable, ETH 40GbE, 40Gb/s, QSFP, 100m
40GbE	MC2210411-SR4E	NVIDIA optical module, 40Gb/s, QSFP, MPO, 850nm, up to 300m
40GbE	MC2609125-005	NVIDIA passive copper hybrid cable, ETH 40GbE to 4x10GbE, QSFP to 4xSFP+, 5m
40GbE	MC2609130-001	NVIDIA passive copper hybrid cable, ETH 40GbE to 4x10GbE, QSFP to 4xSFP+, 1m
40GbE	MC2609130-003	NVIDIA passive copper hybrid cable, ETH 40GbE to 4x10GbE, QSFP to 4xSFP+, 3m
40GbE	MCP1700-B001E	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 1m, black pull-tab
40GbE	MCP1700-B002E	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 2m, black pull-tab
40GbE	MCP1700-B003E	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 3m, black pull-tab
40GbE	MCP1700-B01AE	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 1.5m, black pull-tab
40GbE	MCP1700-B02AE	NVIDIA passive copper cable, ETH 40GbE, 40Gb/s, QSFP, 2.5m, black pull-tab
40GbE	MCP7900-X01AA	NVIDIA passive copper hybrid cable, ETH 40GbE to 4x10GbE, QSFP to 4xSFP+, 1.5m, blue pull-tab, customized label
40GbE	MCP7904-X002A	NVIDIA passive copper hybrid cable, ETH 40GbE to 4x10GbE, QSFP to 4xSFP+, 2m, black pull-tab, customized label
40GbE	MCP7904-X003A	NVIDIA passive copper hybrid cable, ETH 40GbE to 4x10GbE, QSFP to 4xSFP+, 3m, black pull-tab, customized label
40GbE	MCP7904-X01AA	NVIDIA passive copper hybrid cable, ETH 40GbE to 4x10GbE, QSFP to 4xSFP+, 1.5m, black pull-tab, customized label
40GbE	MCP7904-X02AA	NVIDIA passive copper hybrid cable, ETH 40GbE to 4x10GbE, QSFP to 4xSFP+, 2.5m, black pull-tab, customized label
40GbE	MMA1B00-B150D	NVIDIA transceiver, 40GbE, QSFP+, MPO, 850nm, SR4, up to 150m, DDMI

2.5.3.7 25GbE Cables

Speed	Part Number	Marketing Description
25GbE	MAM1Q00A-QSA28	NVIDIA cable module, ETH 25GbE, 100Gb/s to 25Gb/s, QSFP28 to SFP28
25GbE	MCP2M00-A001	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 1m, 30AWG
25GbE	MCP2M00-A001E30N	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 1m, black, 30AWG, CA-N
25GbE	MCP2M00-A002	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 2m, 30AWG
25GbE	MCP2M00-A002E30N	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 2m, black, 30AWG, CA-N
25GbE	MCP2M00-A003E26N	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 3m, black, 26AWG, CA-N
25GbE	MCP2M00-A003E30L	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 3m, black, 30AWG, CA-L
25GbE	MCP2M00-A004E26L	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 4m, black, 26AWG, CA-L
25GbE	MCP2M00-A005E26L	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 5m, black, 26AWG, CA-L
25GbE	MCP2M00-A00A	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 0.5m, 30AWG
25GbE	MCP2M00-A00AE30N	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 0.5m, black, 30AWG, CA-N
25GbE	MCP2M00-A01AE30N	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 1.5m, black, 30AWG, CA-N
25GbE	MCP2M00-A02AE26N	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 2.5m, black, 26AWG, CA-N
25GbE	MCP2M00-A02AE30L	NVIDIA passive copper cable, ETH, up to 25Gb/s, SFP28, 2.5m, black, 30AWG, CA-L
25GbE	MFA2P10-A003	NVIDIA active optical cable 25GbE, SFP28, 3m
25GbE	MFA2P10-A005	NVIDIA active optical cable 25GbE, SFP28, 5m
25GbE	MFA2P10-A007	NVIDIA active optical cable 25GbE, SFP28, 7m
25GbE	MFA2P10-A010	NVIDIA active optical cable 25GbE, SFP28, 10m
25GbE	MFA2P10-A015	NVIDIA active optical cable 25GbE, SFP28, 15m
25GbE	MFA2P10-A020	NVIDIA active optical cable 25GbE, SFP28, 20m
25GbE	MFA2P10-A030	NVIDIA active optical cable 25GbE, SFP28, 30m
25GbE	MFA2P10-A050	NVIDIA active optical cable 25GbE, SFP28, 50m
25GbE	MMA2P00-AS	NVIDIA transceiver, 25GbE, SFP28, LC-LC, 850nm, SR, up to 150m

2.5.3.8 10GbE Cables

Speed	Part Number	Marketing Description
10GbE	MAM1Q00A-QSA	NVIDIA cable module, ETH 10GbE, 40Gb/s to 10Gb/s, QSFP to SFP+

Speed	Part Number	Marketing Description
10GbE	MC2309124-005	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 5m
10GbE	MC2309124-007	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 7m
10GbE	MC2309130-001	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 1m
10GbE	MC2309130-002	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 2m
10GbE	MC2309130-003	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 3m
10GbE	MC2309130-00A	NVIDIA passive copper hybrid cable, ETH 10GbE, 10Gb/s, QSFP to SFP+, 0.5m
10GbE	MC3309124-004	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 4m
10GbE	MC3309124-005	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 5m
10GbE	MC3309124-006	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 6m
10GbE	MC3309124-007	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 7m
10GbE	MC3309130-001	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1m
10GbE	MC3309130-002	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2m
10GbE	MC3309130-003	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 3m
10GbE	MC3309130-00A	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 0.5m
10GbE	MC3309130-0A1	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1.5m
10GbE	MC3309130-0A2	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2.5m
10GbE	MCP2100-X001B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1m, blue pull-tab, connector label
10GbE	MCP2100-X002B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2m, blue pull-tab, connector label
10GbE	MCP2100-X003B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 3m, blue pull-tab, connector label
10GbE	MCP2101-X001B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1m, Green pull-tab, connector label
10GbE	MCP2104-X001B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1m, black pull-tab, connector label
10GbE	MCP2104-X002B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2m, black pull-tab, connector label
10GbE	MCP2104-X003B	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 3m, black pull-tab, connector label
10GbE	MCP2104-X01AB	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 1.5m, black pull-tab, connector label
10GbE	MCP2104-X02AB	NVIDIA passive copper cable, ETH 10GbE, 10Gb/s, SFP+, 2.5m, black pull-tab, connector label
N/A	MFM1T02A-LR	NVIDIA SFP+ optical module for 10GBASE-LR
N/A	MFM1T02A-SR	NVIDIA SFP+ optical module for 10GBASE-SR

2.5.3.9 1GbE Cables

Speed	Part Number	Marketing Description
1GbE	MC3208011-SX	NVIDIA optical module, ETH 1GbE, 1Gb/s, SFP, LC-LC, SX 850nm, up to 500m
1GbE	MC3208411-T	NVIDIA module, ETH 1GbE, 1Gb/s, SFP, Base-T, up to 100m

2.6 Release Notes Change Log History

2.6.1 Changes and New Features in 4.5.4 LTS

- Increased BlueField-2 UEFI database size for storing additional certificates
- Bug fixes

2.6.2 Changes and New Features in 4.5.2

- Bug fixes

2.6.3 Changes and New Features in 4.5.1

- Bug fixes

2.6.4 Changes and New Features in 4.5.0

- Added Redfish support for configuring all UEFI secure boot settings (disable, enable, enroll user keys, etc.) at scale, remotely, and securely
- For FHHL DPUs, added support for performing PCIe bifurcation configuration via MFT tool

  > ⚠ Only a subset of configurations are supported.

- Updated the print of the manufacturing (MFG) setting, `MFG_OOB_MAC`, displayed by the command `bfcfg -d` to appear in lower-case to align with standard Linux tools

2.6.5 Changes and New Features in 4.2.0

> ⚠ **Important!**
>
> Upgrading to this BSP version installs a new version of Ubuntu GRUB. This version of GRUB revokes the old UEFI secure boot certificates and install new ones. The new certificates will not validate older images and boot will fail. Therefore, to roll back to older software versions, users must disable UEFI secure boot.

- BFB installation chooses the on-chip NVMe (`/dev/nvme0n1`) by default for the EFI system partition and Linux rootfs installation and can be overloaded with `device=/dev/mmcblk0` in `bf.cfg` to push together with the BFB.

Installing on NVMe causes DPU booting to stay at the UEFI shell when changing to Livefish mode.

A previously installed OS on the eMMC device stays intact. Only the EFI boot entry is updated to boot from the SSD device.

2.6.6 Changes and New Features in 4.0.3

- BlueField-3 tuning update for power and performance

2.6.7 Changes and New Features in 4.0.2

- BlueField-3 power-capping and thermal-throttling
- Added Linux `fsck` to boot flow
- Log PCIe errors (to RShim log)
- Halt uncorrectable double-bit ECC error on DDR

2.6.8 Changes and New Features in 3.9.3

- Added support for live migration of VirtIO-net and VirtIO-blk VFs from one VM to another. Requires working with the new vDPA driver.
- OS configuration – enabled tmpfs in `/tmp`

2.6.9 Changes and New Features in 3.9.2

- Added support for Arm host
- Enroll new NVIDIA certificates to DPU UEFI database

Important: User action required! See known issue #3077361 for details.

2.6.10 Changes and New Features in 3.9.0

This is the last release to offer GA support for first-generation NVIDIA® BlueField® DPUs.

- Added support for NIC mode of operation
- Added password protection to change boot parameters in GRUB menu
- Added IB support for DOCA runtime and dev environment
- Implemented RShim PF interrupts
- Virtio-net-controller is split to 2 processes for fast recovery after service restart
- Added support for live virtio-net controller upgrade instead of performing a full restart
- Expanded BlueField-2 PCIe bus number range to 254 (0-253)

- Added a new CAP field, `log_max_queue_depth` (value can be set to `2K` / `4K`), to indicate the maximal NVMe SQ and CQ sizes supported by firmware. This can be used by NVMe controllers or by non-NVMe drivers which do not rely on NVMe CAP field.
- Added ability for the RShim driver to still work when the host is in secure boot mode
- Added `bfb-info` command which provides the breakdown of the software components bundled in the BFB package
- Added support for rate limiting VF groups

2.6.11 Changes and New Features in 3.8.5

- PXE boot option is enabled automatically and is available for the ConnectX and OOB network interfaces
- Added Vendor Class option "BF2Client" in DHCP request for PXE boot to identify card
- Updated the "force PXE" functionality to continue to retry PXE boot entries until successful. A configuration called "boot override retry" has been added. With this configured, UEFI does not rebuild the boot entries after all boot options are attempted but loops through the PXE boot options until booting is successful. Once successful, the boot override entry configuration is disabled and would need to be reenabled for future boots.
- Added ability to change the CPU clock dynamically according to the temperature and other sensors of the DPU. If the power consumption reaches close to the maximum allowed, the software module decreases the CPU clock rate to ensure that the power consumption does not cross the system limit.

  > ⚠ This feature is relevant only for OPNs MBF2H516C-CESOT, MBF2M516C-EECOT, MBF2H516C-EESOT, and MBF2H516C-CECOT.

- Bug fixes

2.6.12 Changes and New Features in 3.8.0

- Added ability to perform warm reboot on BlueField-2 based devices
- Added support for DPU BMC with OpenBMC
- Added support for NVIDIA Converged Accelerator (900-21004-0030-000)

2.7 Bug Fixes History

Ref #	Issue Description
4403055	Description: Repeated power cycles cause corruption in the EXT4 file system.
	Keywords: Power cycle; FS corruption
	Fixed in version: 4.5.5
4146553	Description: Fixed the PMD crash while dumping a rule with invalid rule pointer. Check the validity of the pointer.
	Keywords: Invalid rule pointer; dumping rule
	Fixed in version: 4.5.4

Ref #	Issue Description
4412 7420	Description: When calling `…_is_equal_pci_addr()` with a PCIe address of BDF format (i.e., without the domain component), the assumed domain is `0000` (e.g., if the input PCIe address is `03:00.0`, the input is treated as `0000:03:00.0`).
	Keywords: BDF
	Fixed in version: 4.5.4
3901 193	Description: Host PCIe driver hangs when hot plugging a device due to SF creation and error flow handling failure.
	Keywords: Subfunction; hot-plug
	Fixed in version: 4.5.4
3901 190	Description: Virtio-net may see TX timeout on specific queues.
	Keywords: Emulated devices
	Fixed in version: 4.5.4
3899 526	Description: On BlueField-2, the OOB may not get an IP address due to the interface being down.
	Keywords: OOB; IP
	Fixed in version: 4.5.4
3894 575	Description: Kernel updated to include the latest security fix for CVE-2023-52340.
	Keywords: Security; vulnerability
	Fixed in version: 4.5.4
3996 822	Description: Added support flash-less booting to make up for GPIO degradation mitigation
	Keyword: Boot; degradation
	Fixed in version: 4.5.4
3894 907	Description: eMMC clock should be disabled at runtime when using NIC mode.
	Keyword: eMMC
	Fixed in version: 4.5.4
3682 873	Description: BlueField-3 FT test shows abnormal sensor temperature readings.
	Keyword: Thermal
	Fixed in version: 4.5.4
3992 563	Description: V2F sensors hang after boot.
	Keyword: Thermal; hang
	Fixed in version: 4.5.4
3774 088	Description: When enrolling a certificate to the UEFI DB, a failure message `ERROR: Unsupported file type!` is displayed when the DB was full.
	Keyword: SNAP; UEFI
	Fixed in version: 4.5.1
3739 089	Description: Ipmitool from DPU OS toward the DPU BMC is failing to open `/dev/ipmi0` or `/dev/ipmi1/0` error.

Ref #	Issue Description
	Keyword: Ipmitool
	Fixed in version: 4.5.1
3730478	Description: In secure boot user mode, self-signed db certs, and self-signed dbx certs cannot be enrolled via Redfish.
	Keyword: Secure boot
	Fixed in version: 4.5.1
3762951	Description: When the I^2C EEPROM is no longer available (e.g., EOF), data loss (e.g., MFG data loss, etc.) is not optimal for the system to function properly.
	Keyword: EEPROM
	Fixed in version: 4.5.1

3571285	Description: Intermittent UEFI/grub exception after many power-cycles:
	`Call Stack: Synchronous Exception at 0xF4B72E0C`
	`ERR[UEFI]: PC=0xF4B72E0C`
	`ERR[UEFI]: PC=0xF4B72E70`
	`ERR[UEFI]: PC=0xF4B73570`
	`ERR[UEFI]: PC=0xF4B74904`
	`ERR[UEFI]: PC=0xF4F04444`
	`ERR[UEFI]: PC=0xF4F044F8`
	`ERR[UEFI]: PC=0xF4F05160`
	`ERR[UEFI]: PC=0xF4F02030`
	`ERR[UEFI]: PC=0xFDFC3A38 (0xFDFB0000+0x13A38) [ 1] DxeCore.dll`
	`ERR[UEFI]: PC=0xF56E3594 (0xF56D4000+0xF594) [ 2] BdsDxe.dll`
	`ERR[UEFI]: PC=0xF56F1FFC (0xF56D4000+0x1DFFC) [ 2] BdsDxe.dll`
	`ERR[UEFI]: PC=0xF56F40D4 (0xF56D4000+0x200D4) [ 2] BdsDxe.dll`
	`ERR[UEFI]: PC=0xFDFC6E50 (0xFDFB0000+0x16E50) [ 3] DxeCore.dll`
	`ERR[UEFI]: PC=0x880092E0`
	`ERR[UEFI]: PC=0x8800947C`
	`ERR[UEFI]: X0=0x0 X1=0xF4B78FC3 X2=0xE X3=0x0`
	`ERR[UEFI]: X4=0x0 X5=0xFFFFFFFFFFFFFFF8 X6=0x0 X7=0xFFFFFFF5`
	`ERR[UEFI]: X8=0xF4B79480 X9=0x2 X10=0xFFFFFFFFFFFFFFFF X11=0xFFFFDC00`

	Keyword: Security
	Fixed in version: 4.5.0
3599839	Description: On a reboot following BFB install, the error message "Boot Image update completed, Status: Volume Corrupt" is observed. The error is non-functional and may be safely ignored.
	Keyword: Software provisioning; EFI capsule update; eMMC boot partitions
	Fixed in version: 4.5.0
3556795	Description: The first uplink representor interface may not be renamed to `p0` from `ethX`.
	Keyword: Representors
	Fixed in version: 4.5.0
3629875	Description: Fixed base address of static ICM.
	Keyword: ICM
	Fixed in version: 4.5.0
3365363	Description: On BlueField-3, when booting virtio-net emulation device using a GRUB2 bootloader, the bootloader may attempt to close and re-open the virtio-net device. This can result in unexpected behavior and possible system failure to boot.
	Keywords: BlueField-3; virtio-net; UEFI
	Fixed in version: 4.5.0

Ref #	Issue Description
3373849	Description: Different OVS-based packages can include their own systemd services which prevents `/sbin/mlnx_bf_configure` from identifying the right one.
	Keywords: OVS; systemd
	Fixed in version: 4.5.0
3605332	Description: A dmseg is printed due to the OVS bridge interface being configured DOWN by default.
	Keyword: OVS
	Fixed in version: 4.2.1
3479040	Description: For non-LSO data, a max chain of 4 descriptors is posted onto the send queue resulting in a partial packet going out on the wire.
	Keyword: Send; LSO
	Fixed in version: 4.2.1
3549785	Description: NVMe and mlx5_core drivers fail during BFB installation. As a result, Anolis OS cannot be installed on the SSD and the `mlxfwreset` command does not work during Anolis BFB installation.
	Keyword: Linux; NVMe; BFB installation
	Fixed in version: 4.2.1
3393316	Description: When LSO is enabled, if the header and data appear in the same fragment, the following warning is given from tcpdump: `truncated-ip - 9 bytes missing`
	Keyword: Virtio-net; large send offload
	Fixed in version: 4.2.1
3554128	Description: "`dmidecode`" output does not match "`ipmitool fru print`" output.
	Keywords: IPMI; print
	Fixed in version: 4.2.1
3508018	Description: Failure to ssh to Arm via 1GbE OOB interface is experienced after performing warm reboot on the DPU.
	Keywords: SSH; reboot
	Fixed in version: 4.2.0
3451539	Description: BSP build number (fourth digit in version number) does not appear in UEFI menu.
	Keywords: UEFI; software
	Fixed in version: 4.2.0
3259805	Description: Following many power cycles on the BlueField DPU, the virtio-net controller may fail to start with the error `failed to register epoll` in the log.
	Keywords: Virtio-net; power cycle; epoll
	Fixed in version: 4.2.0
3266180	Description: Enabled reset on MMC to enhance recovery on error.
	Keywords: MMC; reset

Ref #	Issue Description
	Fixed in version: 4.2.0
3448217	Description: The PKA engine is not working on CentOS 7.6 due to multiple OpenSSL versions (1.0.2k 1.1.1k) being installed and the library loader not selecting the correct version of the openssl library.
	Keywords: PKA; OpenSSL
	Fixed in version: 4.2.0
3448228	Description: On virtio-net devices with LSO (large send offload) enabled, bogus packets may be captured on the SF representor when running heavy `iperf` traffic.
	Keywords: Virtio-net; iperf
	Fixed in version: 4.2.0
3452583	Description: OpenSSL is not working with PKA engine on CentOS 7.6 with 4.23 5.4 5.10 kernels due to multiple versions of OpenSSL(1.0.2k and 1.1.1k) are installed.
	Keywords: OpenSSL; PKA
	Fixed in version: 4.2.0
3455873	Description: 699140280000 OPN is not supported.
	Keywords: SKU; support
	Fixed in version: 4.2.0
3519341	Description: Populate the vGIC maintenance interrupt number in MADT to avoid harmless.
	Keywords: Error
	Fixed in version: 4.2.0
3522652	Description: The timer frequency is measured using the c0 fmon feature causing new kernels to complain if CNTFRQ_EL0 has a different value on different cores.
	Keywords: Timer frequency
	Fixed in version: 4.2.0
3531965	Description: Memory info displayed via `dmidecode` is not correct for memory sizes 32G and above.
	Keywords: Memory; dmidecode
	Fixed in version: 4.2.0
3362181	Description: A customized BFB with an older kernel does not support bond speed above 200Gb/s.
	Keywords: Bond; LAG; speed
	Fixed in version: 4.2.0
3177569	Description: DCBX configuration may not take effect.
	Keywords: DCBX; QoS; lldpad
	Fixed in version: 4.2.0
2824859	Description: Hotplug/unplug of virtio-net devices during host shutdown/bootup may result in failure to do plug/unplug.
	Keywords: Virtio-net, hotplug
	Fixed in version: 4.2.0
3252083	Description: Assert errors may be observed in the RShim log after reset/reboot. These errors are harmless and may be ignored.

Ref #	Issue Description
	Keywords: RShim; log; error
	Fixed in version: 4.0.3
3240060	Description: Hotplug of a modern virtio-net device is not supported when `VIRTIO_EMULATION_HOTPLUG_TRANS` is `TRUE` from mlxconfig.
	Keywords: Virtio-net; hotplug; legacy
	Fixed in version: 4.0.3
3240182	Description: Virtio-net full emulation is not supported in CentOS 8.2 with inbox-kernel 4.18.0-193.el8.aarch64.
	Keywords: Virtio-net; CentOS
	Fixed in version: 4.0.3
3151884	Description: If secure boot is enabled, the following error message is observed while installing Ubuntu on the DPU: `ERROR: need to use capsule in secure boot mode`. This message is harmless and may be safely ignored.
	Keywords: Error message; installation
	Fixed in version: 3.9.3
2793005	Description: When Arm reboots or crashes after sending a virtio-net unplug request, the hotplugged devices may still be present after Arm recovers. The host, however, will not see those devices.
	Keywords: Virtio-net; hotplug
	Fixed in version: 3.9.3
3107227	Description: BlueField with secured BFB fails to boot up if the `PART_SCHEME` field is set in `bf.cfg` during installation.
	Keywords: Installation; bf.cfg
	Fixed in version: 3.9.2
3109270	Description: If the RShim service is running on an external host over the PCIe interface then, in very rare cases, a soft reset of the BlueField can cause a poisoned completion to be returned to the host. The host may treat this as a fatal error and crash.
	Keywords: RShim; ATF
	Fixed in version: 3.9.2
2790928	Description: Virtio-net-controller recovery may not work for a hot-plugged device because the system assigns a BDF (string identifier) of 0 for the hot-plugged device, which is an invalid value.
	Keywords: Virtio-net; hotplug; recovery
	Fixed in version: 3.9.0
2780819	Description: Eye-opening is not supported on 25GbE integrated-BMC BlueField-2 DPU.
	Keywords: Firmware, eye-opening
	Fixed in version: 3.9.0
2876447	Description: Virtio full emulation is not supported by NVIDIA® BlueField®-2 multi-host cards.
	Keywords: Virtio full emulation; multi-host
	Fixed in version: 3.9.0

Ref #	Issue Description
2855485	Description: After BFB installation, Linux crash may occur with `efi_call_rts` messages in the call trace which can be seen from the UART console.
	Keywords: Linux crash; `efi_call_rts`
	Fixed in version: 3.9.0
2901514	Description: Relaxed ordering is not working properly on virtual functions.
	Keywords: MLNX_OFED; relaxed ordering; VF
	Fixed in version: 3.9.0
2852086	Description: On rare occasions, the UEFI variables in UVPS EEPROM are wiped out which hangs the boot process at the UEFI menu.
	Keywords: UEFI; hang
	Fixed in version: 3.9.0
2934828	Description: PCIe device address to RDMA device name mapping on x86 host may change after the driver restarts in Arm.
	Keywords: RDMA; Arm; driver
	Fixed in version: 3.9.0
-	Description: RShim driver does not work when the host is in secure boot mode.
	Keywords: RShim; Secure Boot
	Fixed in version: 3.9.0
2787308	Description: At rare occasions during Arm reset on BMC-integrated DPUs, the DPU will send "PCIe Completion" marked as poisoned. Some servers treat that as fatal and may hang.
	Keywords: Arm reset; BMC integrated
	Fixed in version: 3.9.0
2585607	Description: Pushing the BFB image fails occasionally with a "bad magic number" error message showing up in the console.
	Keywords: BFB push; installation
	Fixed in version: 3.9.0
2802943	Description: SLD detection may not function properly.
	Keywords: Firmware
	Fixed in version: 3.9.0
2580945	Description: External host reboot may also reboot the Arm cores if the DPU was configured using mlxconfig.
	Keywords: Non-volatile configuration; Arm; reboot
	Fixed in version: 3.9.0
2899740	Description: BlueField-2 may sometimes go to PXE boot instead of Linux after installation.
	Keywords: Installation; PXE
	Fixed in version: 3.8.5
2870143	Description: Some DPUs may get stuck at GRUB menu when booting due to the GRUB configuration getting corrupted when board is powered down before the configuration is synced to memory.

Ref #	Issue Description
	Keywords: GRUB; memory
	Fixed in version: 3.8.5
2873700	Description: The available RShim logging buffer may not have enough space to hold the whole register dump which may cause buffer wraparound.
	Keywords: RShim; logging
	Fixed in version: 3.8.5
2801891	Description: IPMI EMU service reports cable link as down when it is actually up.
	Keywords: IPMI EMU
	Fixed in version: 3.8.0
2779861	Description: Virtio-net controller does not work with devices other than `mlx5_0/1`.
	Keywords: Virtio-net controller
	Fixed in version: 3.8.0
2801378	Description: No parameter validation is done for feature bits when performing hotplug.
	Keywords: Virtio-net; hotplug
	Fixed in version: 3.8.0
2802917	Description: When secure boot is enabled, PXE boot may not work.
	Keywords: Secure boot; PXE
	Fixed in version: 3.8.0
2827413	Description: Updating a BFB could fail due to congestion.
	Keywords: Installation; congestion
	Fixed in version: 3.8.0
2829876	Description: For virtio-net device, modifying the number of queues does not update the number of MSIX.
	Keywords: Virtio-net; queues
	Fixed in version: 3.8.0
2597790	Description: A "double free" error is seen when using the "curl" utility. This happens only when OpenSSL is configured to use a dynamic engine (e.g. Bluefield PKA engine).
	Keywords: OpenSSL; curl
	Fixed in version: 3.8.0
2853295	Description: UEFI secure boot enables the kernel lockdown feature which blocks access by mstmcra.
	Keywords: Secure boot
	Fixed in version: 3.8.0
2854472	Description: Virtio-net controller may fail to start after power cycle.
	Keywords: Virtio-net controller
	Fixed in version: 3.8.0
2854995	Description: Memory consumed for a representor exceeds what is necessary making scaling to 504 SF's not possible.

Ref #	Issue Description
	Keywords: Memory
	Fixed in version: 3.8.0
2856652	Description: Modifying VF bits yields an error.
	Keywords: Virtio-net controller
	Fixed in version: 3.8.0
2859066	Description: Arm hangs when user is thrown to livefish by FW (e.g. secure boot).
	Keywords: Arm; livefish
	Fixed in version: 3.8.0
2866082	Description: The current installation flow requires multiple resets after booting the self-install BFB due to the watchdog being armed after capsule update.
	Keywords: Reset; installation
	Fixed in version: 3.8.0
2866537	Description: Power-off of BlueField shows up as a panic which is then stored in the RShim log and carried into the BERT table in the next boot which is misleading to the user.
	Keywords: RShim; log; panic
	Fixed in version: 3.8.0
2868944	Description: Various errors related to the UPVS store running out of space are observed.
	Keywords: UPVS; errors
	Fixed in version: 3.8.0
2754798	Description: `oob_net0` cannot receive traffic after a network restart.
	Keywords: `oob_net0`
	Fixed in version: 3.8.0
2691175	Description: Up to 31 hot-plugged virtio-net devices are supported even if `PCI_SWITCH_EMULATION_NUM_PORT=32`. Host may hang if it hot plugs 32 devices.
	Keywords: Virtio-net; hotplug
	Fixed in version: 3.8.0

2597973	Description: Working with CentOS 7.6, if SF network interfaces are statically configured, the following parameters should be set.
	`NM_CONTROLLED="no"`
	`DEVTIMEOUT=30`
	For example:
	`# cat /etc/sysconfig/network-scripts/ifcfg-p0m0`
	`NAME=p0m0`
	`DEVICE=p0m0`
	`NM_CONTROLLED="no"`
	`PEERDNS="yes"`
	`ONBOOT="yes"`
	`BOOTPROTO="static"`
	`IPADDR=12.212.10.29`
	`BROADCAST=12.212.255.255`
	`NETMASK=255.255.0.0`
	`NETWORK=12.212.0.0`
	`TYPE=Ethernet`
	`DEVTIMEOUT=30`

Ref #	Issue Description
	Keywords: CentOS; subfunctions; static configuration
	Fixed in version: 3.7.0
2581534	Description: When shared RQ mode is enabled and offloads are disabled, running multiple UDP connections from multiple interfaces can lead to packet drops.
	Keywords: Offload; shared RQ
	Fixed in version: 3.7.0
2581621	Description: When OVS-DPDK and LAG are configured, the kernel driver drops the LACP packet when working in shared RQ mode.
	Keywords: OVS-DPDK; LAG; LACP; shared RQ
	Fixed in version: 3.7.0
2601094	Description: The gpio-mlxbf2 and mlxbf-gige drivers are not supported on 4.14 kernel.
	Keywords: Drivers; kernel
	Fixed in version: 3.7.0
2584427	Description: Virtio-net-controller does not function properly after changing uplink representor MTU.
	Keywords: Virtio-net controller; MTU
	Fixed in version: 3.7.0
2438392	Description: VXLAN with IPsec crypto offload does not work.
	Keywords: VXLAN; IPsec crypto
	Fixed in version: 3.7.0
2406401	Description: Address Translation Services is not supported in BlueField-2 step A1 devices. Enabling ATS can cause server hang.
	Keywords: ATS
	Fixed in version: 3.7.0
2402531	Description: PHYless reset on BlueField-2 devices may cause the device to disappear.
	Keywords: PHY; firmware reset
	Fixed in version: 3.7.0
2400381	Description: When working with strongSwan 5.9.0bf, running `ip xfrm state show` returns partial information as to the offload parameters, not showing "`mode full`".
	Keywords: strongSwan; ip xfrm; IPsec
	Fixed in version: 3.7.0
2392604	Description: Server crashes after configuring PCI_SWITCH_EMULATION_NUM_PORT to a value higher than the number of PCIe lanes the server supports.
	Keywords: Server; hang
	Fixed in version: 3.7.0
2293791	Description: Loading/reloading NVMe after enabling VirtIO fails with a PCI bar memory mapping error.
	Keywords: VirtIO; NVMe
	Fixed in version: 3.7.0

Ref #	Issue Description
2245983	Description: When working with OVS in the kernel and using Connection Tracking, up to 500,000 flows may be offloaded.
	Keywords: DPU; Connection Tracking
	Fixed in version: 3.7.0
1945513	Description: If the Linux OS running on the host connected to the BlueField DPU has a kernel version lower then 4.14, MLNX_OFED package should be installed on the host.
	Keywords: Host OS
	Fixed in version: 3.7.0
1900203	Description: During heavy traffic, ARP reply from the other tunnel endpoint may be dropped. If no ARP entry exists when flows are offloaded, they remain stuck on the slow path.
	Workaround: Set a static ARP entry at the BlueField Arm to VXLAN tunnel endpoints.
	Keywords: ARP; Static; VXLAN; Tunnel; Endpoint
	Fixed in version: 3.7.0
2082985	Description: During boot, the system enters systemctl emergency mode due a corrupt root file system.
	Keywords: Boot
	Fixed in version: 3.6.0.11699
2278833	Description: Creating a bond via NetworkManager and restarting the driver (openibd restart) results in no pf0hpf and bond creation failure.
	Keywords: Bond; LAG; network manager; driver reload
	Fixed in version: 3.6.0.11699
2286596	Description: Only up to 62 host virtual functions are currently supported.
	Keywords: DPU; SR-IOV
	Fixed in version: 3.6.0.11699
2397932	Description: Before changing SR-IOV mode or reloading the mlx5 drivers on IPsec-enabled systems, make sure all IPsec configurations are cleared by issuing the command `ip x s f && ip x p f`.
	Keywords: IPsec; SR-IOV; driver
	Fixed in version: 3.6.0.11699
2405039	Description: In Ubuntu, during or after a reboot of the Arm, manually, or as part of a firmware reset, the network devices may not transition to switchdev mode. No device representors would be created (pf0hpf, pf1hpf, etc). Driver loading on the host will timeout after 120 seconds.
	Keywords: Ubuntu; reboot; representors; switchdev
	Fixed in version: 3.6.0.11699
2403019	Description: EEPROM storage for UEFI variables may run out of space and cause various issues such as an inability to push new BFB (due to timeout) or exception when trying to enter UEFI boot menu.
	Keywords: BFB install; timeout; EEPROM UEFI Variable; UVPS
	Fixed in version: 3.6.0.11699

Ref #	Issue Description

2458040	Description: When using OpenSSL on BlueField platforms where Crypto support is disabled, the following errors may be encountered:
	`PKA_ENGINE: PKA instance is invalid`
	`PKA_ENGINE: failed to retrieve valid instance`
	This happens due to OpenSSL configuration being linked to use PKA hardware, but that hardware is not available since crypto support is disabled on these platforms.

	Keywords: PKA; Crypto
	Fixed in version: 3.6.0.11699
2456947	Description: All NVMe emulation counters (Ctrl, SQ, Namespace) return "0" when queried.
	Keywords: Emulated devices; NVMe
	Fixed in version: 3.6.0.11699
2411542	Description: Multi-APP QoS is not supported when LAG is configured.
	Keywords: Multi-APP QoS; LAG
	Fixed in version: 3.6.0.11699
2394130	Description: When creating a large number of VirtIO VFs, hung task call traces may be seen in the dmesg.
	Keywords: VirtIO; call traces; hang
	Fixed in version: 3.5.1.11601
2398050	Description: Only up to 60 virtio-net emulated virtual functions are supported if LAG is enabled.
	Keywords: Virtio-net; LAG
	Fixed in version: 3.5.1.11601
2256134	Description: On rare occasions, rebooting the BlueField DPU may result in traffic failure from the x86 host.
	Keywords: Host; Arm
	Fixed in version: 3.5.1.11601
2400121	Description: When emulated PCIe switch is enabled, and more than 8 PFs are enabled, the BIOS boot process might halt.
	Keywords: Emulated PCIe switch
	Fixed in version: 3.5.0.11563
2082985	Description: During boot, the system enters systemctl emergency mode due a corrupt root file system.
	Keywords: Boot
	Fixed in version: 3.5.0.11563
2249187	Description: With the OCP card connecting to multiple hosts, one of the hosts could have the RShim PF exposed and probed by the RShim driver.
	Keywords: RShim; multi-host
	Fixed in version: 3.5.0.11563
2363650	Description: When moving to separate mode on the DPU, the OVS bridge remains and no ping is transmitted between the Arm cores and the remote server.
	Keywords: SmartNIC; operation modes

Ref #	Issue Description
	Fixed in version: 3.5.0.11563
2394226	Description: Pushing the BFB image v3.5 with a WinOF-2 version older than 2.60 can cause a crash on the host side.
	Keywords: Windows; RShim
	Fixed in version: 3.5.0.11563

3 BlueField Software Overview

NVIDIA provides software which enables users to fully utilize the NVIDIA® BlueField® DPU and enjoy the rich feature-set it provides. Using the BlueField software packages, users can:

- Quickly and easily boot an initial Linux image on your development board
- Port existing applications to and develop new applications for BlueField
- Patch, configure, rebuild, update or otherwise customize your image
- Debug, profile, and tune their development system using open-source development tools taking advantage of the diverse and vibrant Arm ecosystem

Coupled with the NVIDIA® ConnectX® interconnect, the BlueField family of DPU devices includes an array of Arm cores according to the following:

- 64-bit Armv8 A72 for BlueField-2 DPUs
- 64-bit Armv8 A78 for BlueField-3 DPUs

Standard Linux distributions run on the Arm cores allowing common open-source development tools to be used. Developers should find the programming environment familiar and intuitive which in turn allows them to design, implement, and verify their control-plane and data-plane applications quickly and efficiently.

BlueField SW ships with the NVIDIA® BlueField® Reference Platform. BlueField SW is a reference Linux distribution based on the Ubuntu Server distribution extended to include the MLNX_OFED stack for Arm and a Linux kernel which supports NVMe-oF. This software distribution can run all customer-based Linux applications seamlessly.

The following are other software elements delivered with BlueField DPU:

- Arm Trusted Firmware (ATF) for BlueField
- UEFI for BlueField
- OpenBMC for BMC (ASPEED 2500) found on development board
- MLNX_OFED stack
- Mellanox MFT

3.1 Debug Tools

BlueField DPU includes hardware support for the Arm DS5 suite as well as CoreSight™ debug. As such, a wide range of commercial off-the-shelf Arm debug tools should work seamlessly with BlueField. The CoreSight debugger interface can be accessed via RShim interface (USB or PCIe if using DPU) as well which could be used for debugging with open-source tools like OpenOCD.

The BlueField DPU also supports the ubiquitous GDB.

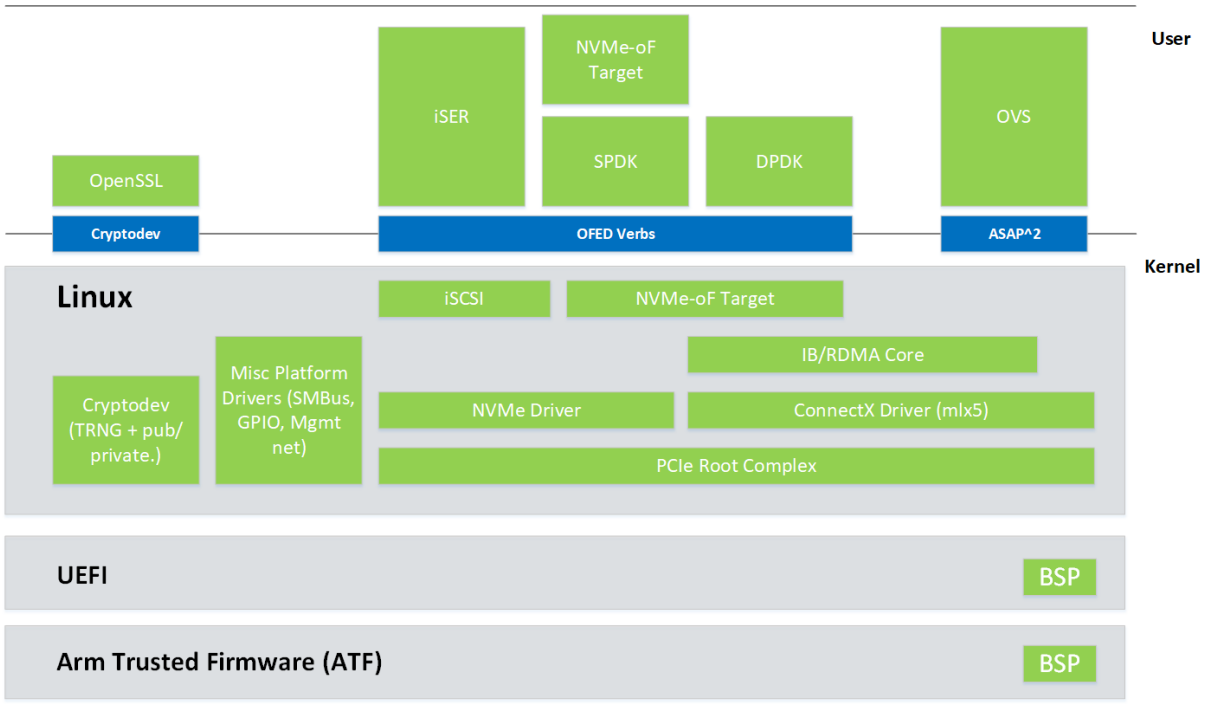
3.2 BlueField-based Storage Appliance

BlueField software provides the foundation for building a JBOF (Just a Bunch of Flash) storage system including NVMe-oF target software, PCIe switch support, NVDIMM-N support, and NVMe disk hot-swap support.

BlueField SW allows enabling ConnectX offload such as RDMA/RoCE, T10 DIF signature offload, erasure coding offload, iSER, Storage Spaces Direct, and more.

3.3 BlueField Architecture

The BlueField architecture is a combination of two preexisting standard off-the-shelf components, Arm AArch64 processors, and ConnectX-6 Dx (for BlueField-2), ConnectX-7 (for BlueField-3), or network controller, each with its own rich software ecosystem. As such, almost any of the programmer-visible software interfaces in BlueField come from existing standard interfaces for the respective components.

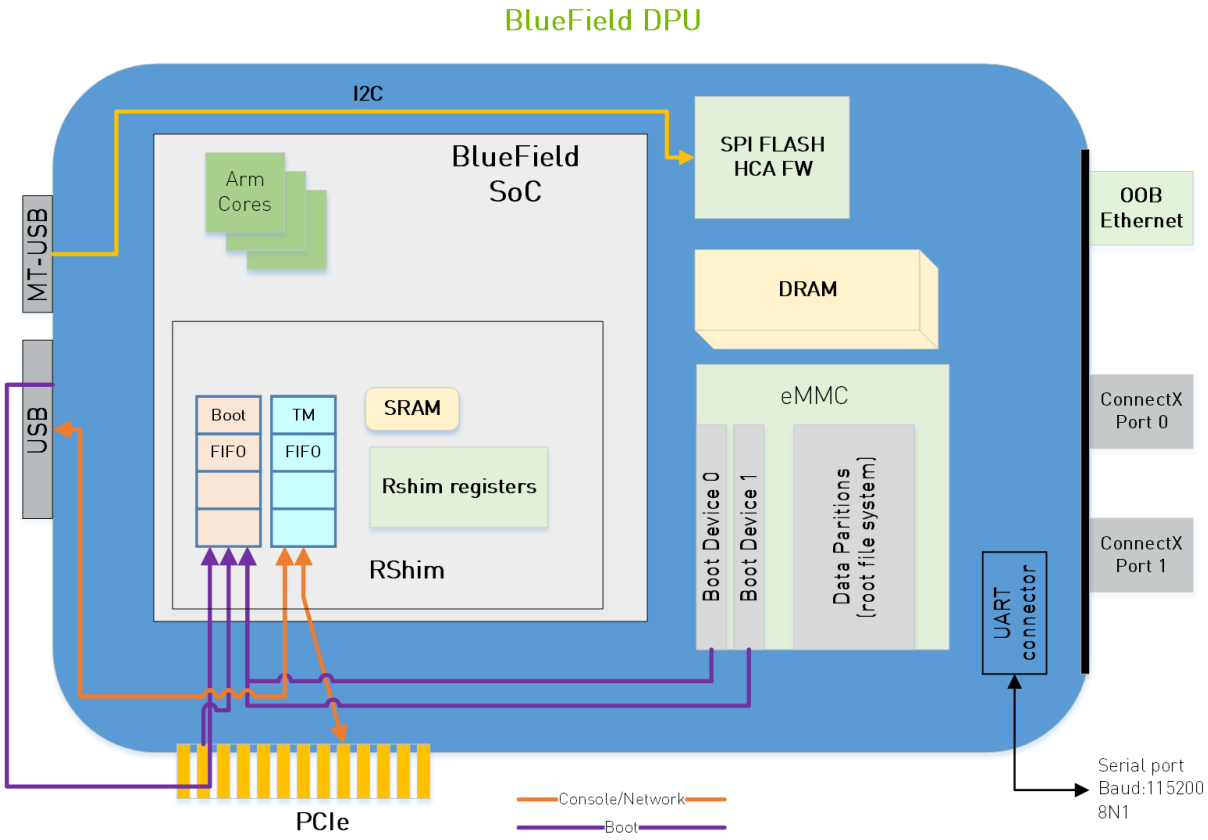


The Arm related interfaces (including those related to the boot process, PCIe connectivity, and cryptographic operation acceleration) are standard Linux on Arm interfaces. These interfaces are enabled by drivers and low-level code provided by NVIDIA as part of the BlueField software delivered and upstreamed to respective open-source projects, such as Linux.

The ConnectX network controller-related interfaces (including those for Ethernet and InfiniBand connectivity, RDMA and RoCE, and storage and network operation acceleration) are identical to the interfaces that support ConnectX standalone network controller cards. These interfaces take advantage of the MLNX_OFED software stack and InfiniBand verbs-based interfaces to support software.

3.4 System Connections

The BlueField DPU has multiple connections (see diagram below). Users can connect to the system via different consoles, network connections, and a JTAG connector.



3.4.1 System Consoles

The BlueField DPU has multiple console interfaces:

- Serial console 0 (`/dev/ttyAMA0` on the Arm cores)
  - Requires cable to NC-SI connector on DPU 25G
  - Requires serial cable to 3-pin connector on DPU 100G
  - Connected to BMC serial port on BF1200 platforms
- Serial console 1 (`/dev/ttyAMA1` on the Arm cores but only for BF1200 reference platform)
  - ttyAMA1 is the console connection on the front panel of the BF1200
- Virtual RShim console (`/dev/hvc0` on the Arm cores) is driven by
  - The RShim PCIe driver (does not require a cable but the system cannot be in isolation mode as isolation mode disables the PCIe device needed)
  - The RShim USB driver (requires USB cable)
  - It is not possible to use both the PCIe and USB RShim interfaces at the same time

3.4.2 Network Interfaces

The DPU has multiple network interfaces.

- ConnectX Ethernet/InfiniBand interfaces

- RShim virtual Ethernet interface (via USB or PCIe)
  The virtual Ethernet interface can be very useful for debugging, installation, or basic management. The name of the interface on the host DPU server depends on the host operating system. The interface name on the Arm cores is normally "tmfifo_net0". The virtual network interface is only capable of roughly 10MB/s operation and should not be considered for production network traffic.
- OOB Ethernet interface
  BlueField-2 based platforms feature an OOB 1GbE management port. This interface provides a 1Gb/s full duplex connection to the Arm cores. The interface name is normally "oob_net0". The interface enables TCP/IP network connectivity to the Arm cores (e.g., for file transfer protocols, SSH, and PXE boot). The OOB port is not a path for the BlueField-2 boot stream (i.e., any attempt to push a BFB to this port will not work).

4 Software Installation and Upgrade

> It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

The NVIDIA® BlueField® DPU is shipped with the BlueField software based on Ubuntu 20.04 pre-installed. The DPU's Arm execution environment has the capability of being functionally isolated from the host server and uses a dedicated network management interface (separate from the host server's management interface). The Arm cores can run the Open vSwitch Database (OVSDB) or other virtual switches to create a secure solution for bare metal provisioning.

The software package also includes support for DPDK as well as applications for accelerated encryption.

The BlueField DPU supports several methods for OS deployment and upgrade:

- Full OS image deployment using a BlueField boot stream file (BFB) via RShim interface
- Full OS deployment using PXE which can be used over different network interfaces available on the BlueField DPU (1GbE mgmt, tmfifo or NVIDIA® ConnectX®)
- Individual packages can be installed or upgraded using standard Linux package management tools (e.g., apt, dpkg, etc.)

4.1 Deploying BlueField Software Using BFB from Host

> It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

> This procedure assumes that a BlueField DPU has already been installed in a server according to the instructions detailed in the DPU's hardware user guide.

The following table lists an overview of the steps required to install Ubuntu BFB on your DPU:

Step	Procedure	Link to Section
1	Uninstall previous DOCA on host (if exists)	Uninstall Previous Software from Host
2	Install RShim on the host	Install RShim on Host
3	Verify that RShim is running on the host	Ensure RShim Running on Host
4	Change the default credentials using `bf.cfg` file (optional)	Changing Default Credentials Using bf.cfg
5	Install the Ubuntu BFB image	BFB Installation
6	Verify installation completed successfully	Verify BFB is Installed
7	Upgrade the firmware on your DPU	Firmware Upgrade

4.1.1 Uninstall Previous Software from Host

If an older DOCA software version is installed on your host, make sure to uninstall it before proceeding with the installation of the new version:

Ubuntu	`host# for f in $( dpkg --list | grep doca | awk '{print $2}' ); do echo $f ; apt remove --purge $f -y ; done` `host# sudo apt-get autoremove`

CentOS/ RHEL	`host# for f in $(rpm -qa |grep -i doca ) ; do yum -y remove $f; done`
	`host# yum autoremove`
	`host# yum makecache`

4.1.2 Install RShim on Host

Before installing the RShim driver, verify that the RShim devices, which will be probed by the driver, are listed under `lsusb` or `lspci`.

```
lspci | grep -i nox
```

Output example:

```
27:00.0 Ethernet controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network controller
27:00.1 Ethernet controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network controller
27:00.2 Non-Volatile memory controller: Mellanox Technologies NVMe SNAP Controller
27:00.3 DMA controller: Mellanox Technologies MT42822 BlueField-2 SoC Management Interface // This is the RShim PF
```

RShim is compiled as part of the `doca-tools` package in the `doca-host-repo-ubuntu<version>_amd64` file (`.deb` or `.rpm`).

To install `doca-tools`:

OS	Procedure

Ubuntu/Debian	1. Download the DOCA Tools host package from the "Installation Files" section in the *NVIDIA DOCA Installation Guide for Linux*.
	2. Unpack the deb repo. Run:
	`host# sudo dpkg -i doca-host-repo-ubuntu<version>_amd64.deb`
	3. Perform apt update. Run:
	`host# sudo apt-get update`
	4. Run `apt install` for DOCA Tools.
	`host# sudo apt install doca-tools`

OS	Procedure

CentOS/RHEL 7.x	1. Download the DOCA Tools host package from the "Installation Files" section in the *NVIDIA DOCA Installation Guide for Linux*.
	2. Unpack the RPM repo. Run:
	`host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`
	3. Enable new yum repos. Run:
	`host# sudo yum makecache`
	4. Run `yum install` to install DOCA Tools.
	`host# sudo yum install doca-tools`

CentOS/RHEL 8.x or Rocky 8.6	1. Download the DOCA Tools host package from the "Installation Files" section in the *NVIDIA DOCA Installation Guide for Linux*.
	2. Unpack the RPM repo. Run:
	`host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`
	3. Enable new dnf repos. Run:
	`host# sudo dnf makecache`
	4. Run `dnf install` to install DOCA Tools.
	`host# sudo dnf install doca-tools`

4.1.3 Ensure RShim Running on Host

1. Verify RShim status. Run:

```
sudo systemctl status rshim
```

Expected output:

```
active (running)
...
Probing pcie-0000:<DPU's PCIe Bus address on host>
create rshim pcie-0000:<DPU's PCIe Bus address on host>
rshim<N> attached
```

Where `<N>` denotes RShim enumeration starting with 0 (then 1, 2, etc.) for every additional DPU installed on the server.

If the text "`another backend already attached`" is displayed, users will not be able to use RShim on the host. Please refer to "RShim Troubleshooting and How-Tos" to troubleshoot RShim issues.

a. If the previous command displays `inactive` or another error, restart RShim service. Run:

```
sudo systemctl restart rshim
```

b. Verify RShim status again. Run:

```
sudo systemctl status rshim
```

If this command does not display "`active (running)`", then refer to "RShim Troubleshooting and How-Tos".

2. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME
DEV_NAME        pcie-0000:04:00.2
```

This output indicates that the RShim service is ready to use.

4.1.4 Installing Ubuntu on BlueField

It is important to know your device name (e.g., `mt41686_pciconf0`).

MST tool is necessary for this purpose which is installed by default on the DPU.

Run:

```
mst status -v
```

Example output:

```
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded
PCI devices:
------------
DEVICE_TYPE             MST                           PCI       RDMA            NET
NUMA
BlueField2(rev:1)       /dev/mst/mt41686_pciconf0.1   3b:00.1   mlx5_1          net-ens1f1
0

BlueField2(rev:1)       /dev/mst/mt41686_pciconf0     3b:00.0   mlx5_0          net-ens1f0
0
```

4.1.4.1 Changing Default Credentials Using bf.cfg

Ubuntu users are prompted to change the default password (ubuntu) for the default user (ubuntu) upon first login. Logging in will not be possible even if the login prompt appears until all services are up ("`DPU is ready`" message appears in `/dev/rshim0/misc`).

Attempting to log in before all services are up prints the following message: "Permission denied, please try again."

Alternatively, Ubuntu users can provide a unique password that will be applied at the end of the BFB installation. This password would need to be defined in a `bf.cfg` configuration file. To set the password for the ubuntu user:

1. Create password hash. Run:

```
# openssl passwd -1
Password:
```

```
Verifying - Password:
$1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1
```

2. Add the password hash in quotes to the `bf.cfg` file:

```
# vim bf.cfg
ubuntu_PASSWORD='$1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1'
```

The `bf.cfg` file is used with the `bfb-install` script in the steps that follow.

> ⚠ Password policy:
> - Minimum password length - 8
> - At least one upper-case letter
> - At least one lower-case letter
> - At least one numerical character

⚠ For a comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

4.1.4.2 GRUB Password Protection

GRUB menu entries are protected by a username and password to prevent unwanted changes to the default boot options or parameters.

The default credentials are as follows:

Username	`admin`
Password	`BlueField`

The password can be changed during BFB installation by providing a new `grub_admin_PASSWORD` parameter in `bf.cfg`:

```
# vim bf.cfg
grub_admin_PASSWORD='
grub.pbkdf2.sha512.10000.5EB1FF92FDD89BDAF3395174282C77430656A6DBEC1F9289D5F5DAD17811AD0E2196D0E49B49EF31C21972669D
180713E265BB2D1D4452B2EA9C7413C3471C53.F533423479EE7465785CC2C79B637BDF77004B5CC16C1DDE806BCEA50BF411DE04DFCCE42279
E2E1F605459F1ABA3A0928CE9271F2C84E7FE7BF575DC22935B1'
```

To get a new encrypted password value use the command `grub-mkpasswd-pbkdf2`.

After the installation, the password can be updated by editing the file `/etc/grub.d/40_custom` and then running the command `update-grub` which updates the file `/boot/grub/grub.cfg`.

4.1.4.3 BFB Installation

> ⚠ **For BlueField-2 DPUs Only**
>
> Check the BFB version installed on your BlueField-2 DPU. If the version is 1.5.0 or lower, please see Known Issue Reference #3600716 under Known Issues section.

Installing the BFB does not update the firmware by default. To install the NIC firmware during BFB upgrade, perform the following offline before sending the BFB file:

1. Generate the `bf.cfg` file and combine it with the BFB file:

```
# echo WITH_NIC_FW_UPDATE=yes > bf.cfg
# cat <path_to_bfb> bf.cfg > new.bfb
```

2. Utilize the newly created BFB file, `new.bfb`, while following the instructions below.

A pre-built BFB of Ubuntu 20.04 with DOCA Runtime and DOCA packages installed is available on the NVIDIA DOCA SDK developer zone page.

All new BlueField-2 devices are secure boot enabled, hence all SW images must be signed by NVIDIA in order to boot. All formally published SW images are signed.

To install Ubuntu BFB, run on the host side:

```
# bfb-install -h
syntax: bfb-install --bfb|-b <BFBFILE> [--config|-c <bf.cfg>] \
  [--rootfs|-f <rootfs.tar.xz>] --rshim|-r <rshimN> [--help|-h]
```

The `bfb-install` utility is installed by the RShim package.

This utility script pushes the BFB image and optional configuration (`bf.cfg` file) to the BlueField side and checks and prints the BFB installation progress. To see the BFB installation progress, please install the `pv` Linux tool.

BFB image installation must complete before restarting the system/BlueField. Doing so may result in the BlueField DPU not operating as expected (e.g., it may not be accessible using SSH). If this happens, re-initiate the update process with `bfb-install` to recover the DPU.

The following is an output example of Ubuntu 20.04 installation with the `bfb-install` script assuming `pv` has been installed.

```
# bfb-install --bfb <BlueField-BSP>.bfb --config bf.cfg --rshim rshim0
Pushing bfb + cfg
1.21GiB 0:01:14 [16.5MiB/s] [ <=> ]
Collecting BlueField booting status. Press Ctrl+C to stop…
INFO[PSC]: PSC BL1 START
INFO[BL2]: start
INFO[BL2]: DDR POST passed
INFO[BL2]: UEFI loaded
INFO[BL31]: start
INFO[BL31]: lifecycle Production
INFO[BL31]: VDD adjustment complete
INFO[BL31]: VDD adjustment complete
INFO[BL31]: power capping disabled
INFO[BL31]: runtime
INFO[UEFI]: eMMC init
INFO[UEFI]: eMMC probed
INFO[UEFI]: UPVS valid
INFO[UEFI]: PMI: updates started
INFO[UEFI]: PMI: boot image update
INFO[UEFI]: PMI: updates completed, status 0
INFO[UEFI]: PCIe enum start
INFO[UEFI]: PCIe enum end
INFO[UEFI]: exit Boot Service
INFO[MISC]: Found bf.cfg
INFO[MISC]: Ubuntu installation started
INFO[MISC]: Installing OS image
INFO[MISC]: Changing the default password for user ubuntu
INFO[MISC]: Running bfb_modify_os from bf.cfg
INFO[MISC]: ==================== bfb_modify_os ====================
```

```
INFO[MISC]: Installation finished
INFO[MISC]: Rebooting...
```

4.1.4.4 Verify BFB is Installed

After installation of the Ubuntu OS is complete, the following note appears in `/dev/rshim0/misc` on first boot:

```
...
INFO[MISC]: Linux up
INFO[MISC]: DPU is ready
```

"DPU is ready" indicates that all the relevant services are up and users can login the system.

After the installation of the Ubuntu 20.04 BFB, the configuration detailed in the following sections is generated.

> ⚠ Make sure all the services (including cloud-init) are started on BlueField and to perform a graceful shutdown before power cycling the host server.

BlueField OS image version is stored under `/etc/mlnx-release` in the DPU.

```
# cat /etc/mlnx-release
DOCA_2.6.0_BSP_4.6.0_Ubuntu_22.04-<version>
```

4.1.4.5 Firmware Upgrade

> ⚠ `mlxfwreset` is not supported in this release. Please perform a graceful shutdown, and power cycle the host where `mlxfwreset` is requested.

To upgrade firmware:

1. Set a temporary static IP on the host. Run:

   ```
   sudo ip addr add 192.168.100.1/24 dev tmfifo_net0
   ```

2. SSH to your DPU via 192.168.100.2 (preconfigured). The default credentials for Ubuntu are as follows.

Username	Password
ubuntu	Set during installation

   For example:

   ```
   ssh ubuntu@192.168.100.2
   Password: <unique-password>
   ```

3. Upgrade the firmware on the DPU. Run:

   ```
   sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl --force-fw-update
   ```

Example output:

```
Device #1:
----------

  Device Type:      BlueField-2
  [...]
  Versions:         Current         Available
     FW             <Old_FW>        <New_FW>
```

Important! To apply NVConfig changes, stop here and follow the steps in section "Updating NVConfig Params".

4. Perform a graceful shutdown and power cycle the host for the changes to take effect.

4.1.4.6 Updating NVConfig Params

1. Reset the `nvconfig` params to their default values:

```
# sudo mlxconfig -d /dev/mst/<device-id> -y reset

Reset configuration for device /dev/mst/<device-name>? (y/n) [n] : y
Applying... Done!
-I- Please reboot machine to load new configurations.
```

To learn the device ID of the DPUs on your setup, run:

```
mst start
mst status -v
```

Example output:

```
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded
PCI devices:
------------
DEVICE_TYPE             MST                           PCI       RDMA            NET
NUMA
BlueField2(rev:1)       /dev/mst/mt41686_pciconf0.1   3b:00.1   mlx5_1          net-ens1f1
0

BlueField2(rev:1)       /dev/mst/mt41686_pciconf0     3b:00.0   mlx5_0          net-ens1f0
0

BlueField3(rev:1)       /dev/mst/mt41692_pciconf0.1   e2:00.1   mlx5_1          net-
ens7f1np1             4

BlueField3(rev:1)       /dev/mst/mt41692_pciconf0     e2:00.0   mlx5_0          net-
ens7f0np0             4
```

The device IDs for the BlueField-2 and BlueField-3 DPUs in this example are `/dev/mst/mt41686_pciconf0` and `/dev/mst/mt41692_pciconf0` respectively.

2. (Optional) Enable NVMe emulation. Run:

```
sudo mlxconfig -d <device-id> s NVME_EMULATION_ENABLE=1
```

3. Skip this step if your BlueField DPU is Ethernet only. Please refer to section "Supported Platforms and Interoperability" under the Release Notes to learn your DPU type.
   If you have a VPI DPU, the default link type of the ports will be configured to IB. If you want to change the link type to Ethernet, please run the following configuration:

```
sudo mlxconfig -d <device-id> s LINK_TYPE_P1=2 LINK_TYPE_P2=2
```

4. Perform a graceful shutdown and power cycle the host for the `mlxconfig` settings to take effect.

4.1.4.7 Customizations During BFB Installation

Using configuration parameters in the `bf.cfg` file, the BlueField's boot options and OS can be further customized. For a full list of the supported parameters to customize your DPU system during BFB installation, refer to section "bf.cfg Parameters". In addition to parameters, the `bf.cfg` file offers control over customization of the BlueField firmware and OS through scripting.

Add any of the following functions to the `bf.cfg` file for them to be called by the `install.sh` script embedded in the BFB:

- `bfb_modify_os` - called after file the system is extracted on the target partitions. It can be used to modify files or create new files on the target file system mounted under `/mnt`. So the file path should look as follows: `/mnt/<expected_path_on_target_OS>`. This can be used to run a specific tool from the target OS (remember to add `/mnt` to the path for the tool).
- `bfb_pre_install` - called before eMMC/SSD partitions format and OS filesystem is extracted
- `bfb_post_install` - called as a last step before reboot. All eMMC/SSD partitions are unmounted at this stage.

For example, the `bf.cfg` script below disables OVS bridge creation upon boot:

```
# cat /root/bf.cfg

bfb_modify_os()
{
        log ==================== bfb_modify_os ====================
        log "Disable OVS bridges creation upon boot"
        sed -i -r -e 's/(CREATE_OVS_BRIDGES=).*/\1"no"/' /mnt/etc/mellanox/mlnx-ovs.conf
}

bfb_pre_install()
{
        log ==================== bfb_pre_install ====================
}

bfb_post_install()
{
        log ==================== bfb_post_install ====================
}
```

> **Warning**
> After modifying files on the BlueField DPU, run the command `sync` to flush file system buffers to eMMC/SSD flash memory to avoid data loss during reboot or power cycle.

4.1.4.8 Default Ports and OVS Configuration

The `/sbin/mlnx_bf_configure` script runs automatically with `ib_umad` kernel module loaded (see `/etc/modprobe.d/mlnx-bf.conf`) and performs the following configurations:

1. Ports are configured with switchdev mode and software steering.
2. RDMA device isolation in network namespace is enabled.

3. Two scalable function (SF) interfaces are created (one per port) if BlueField is configured with Embedded CPU mode (default):

```
# mlnx-sf -a show

SF Index: pci/0000:03:00.0/229408
  Parent PCI dev: 0000:03:00.0
  Representor netdev: en3f0pf0sf0
  Function HWADDR: 02:61:f6:21:32:8c
  Auxiliary device: mlx5_core.sf.2
    netdev: enp3s0f0s0
    RDMA dev: mlx5_2

SF Index: pci/0000:03:00.1/294944
  Parent PCI dev: 0000:03:00.1
  Representor netdev: en3f1pf1sf0
  Function HWADDR: 02:30:13:6a:2d:2c
  Auxiliary device: mlx5_core.sf.3
    netdev: enp3s0f1s0
    RDMA dev: mlx5_3
```

The parameters for these SFs are defined in configuration file `/etc/mellanox/mlnx-sf.conf`.

```
/sbin/mlnx-sf --action create --device 0000:03:00.0 --sfnum 0 --hwaddr 02:61:f6:21:32:8c
/sbin/mlnx-sf --action create --device 0000:03:00.1 --sfnum 0 --hwaddr 02:30:13:6a:2d:2c
```

> ⚠ To avoid repeating a MAC address in the your network, the SF MAC address is set randomly upon BFB installation. You may choose to configure a different MAC address that better suit your network needs.

4. Two OVS bridges are created:

```
# ovs-vsctl show
f08652a8-92bf-4000-ba0b-7996c772aff6
    Bridge ovsbr2
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port p1
            Interface p1
        Port en3f1pf1sf0
            Interface en3f1pf1sf0
        Port pf1hpf
            Interface pf1hpf
    Bridge ovsbr1
        Port p0
            Interface p0
        Port pf0hpf
            Interface pf0hpf
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port en3f0pf0sf0
            Interface en3f0pf0sf0
    ovs_version: "2.14.1"
```

The parameters for these bridges are defined in configuration file `/etc/mellanox/mlnx-ovs.conf`:

```
CREATE_OVS_BRIDGES="yes"
OVS_BRIDGE1="ovsbr1"
OVS_BRIDGE1_PORTS="p0 pf0hpf en3f0pf0sf0"
OVS_BRIDGE2="ovsbr2"
OVS_BRIDGE2_PORTS="p1 pf1hpf en3f1pf1sf0"
OVS_HW_OFFLOAD="yes"
OVS_START_TIMEOUT=30
```

> ⚠ If failures occur in `/sbin/mlnx_bf_configure` or configuration changes happen (e.g. switching to separated host mode) OVS bridges are not created even if `CREATE_OVS_BRIDGES="yes"`.

5. OVS HW offload is configured.

4.1.4.9 Default Network Interface Configuration

Network interfaces are configured using the `netplan` utility:

```
# cat /etc/netplan/50-cloud-init.yaml
# This file is generated from information provided by the datasource.  Changes
# to it will not persist across an instance reboot.  To disable cloud-init's
# network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
    ethernets:
        tmfifo_net0:
            addresses:
            - 192.168.100.2/30
            dhcp4: false
            nameservers:
                addresses:
                - 192.168.100.1
            routes:
            -   metric: 1025
                to: 0.0.0.0/0
                via: 192.168.100.1
        oob_net0:
            dhcp4: true
    renderer: NetworkManager
    version: 2

# cat /etc/netplan/60-mlnx.yaml
network:
  ethernets:
    enp3s0f0s0:
      dhcp4: 'true'
    enp3s0f1s0:
      dhcp4: 'true'
  renderer: networkd
  version: 2
```

BlueField DPUs also have a local IPv6 (LLv6) derived from the MAC address via the STD stack mechanism. For a default MAC, 00:1A:CA:FF:FF:01, the LLv6 address would be fe80::21a:caff:feff:ff01.

For multi-device support, the LLv6 address works with SSH for any number of DPUs in the same host by including the interface name in the SSH command:

```
ssh -6 ubuntu@fe80::21a:caff:feff:ff01%tmfifo_net<n>
```

If `tmfifo_net<n>` on the host does not have an LLv6 address, restart the RShim driver:

```
systemctl restart rshim
```

4.1.5 Ubuntu Boot Time Optimizations

To improve the boot time, the following optimizations were made to Ubuntu OS image:

```
# cat /etc/systemd/system/systemd-networkd-wait-online.service.d/override.conf
[Service]
ExecStart=
ExecStart=/usr/bin/nm-online -s -q --timeout=5

# cat /etc/systemd/system/NetworkManager-wait-online.service.d/override.conf
[Service]
ExecStart=
ExecStart=/usr/lib/systemd/systemd-networkd-wait-online --timeout=5

# cat /etc/systemd/system/networking.service.d/override.conf
[Service]
TimeoutStartSec=5
ExecStop=
```

```
ExecStop=/sbin/ifdown -a --read-environment --exclude=lo --force --ignore-errors
```

This configuration may affect network interface configuration if DHCP is used. If a network device fails to get configuration from the DHCP server, then the timeout value in the two files above must be increased.

Grub Configuration:

Setting the Grub timeout at 2 seconds with `GRUB_TIMEOUT=2` under `/etc/default/grub`. In conjunction with the `GRUB_TIMEOUT_STYLE=countdown` parameter, Grub will show the countdown of 2 seconds in the console before booting Ubuntu. Please note that, with this short timeout, the standard Grub method for entering the Grub menu (i.e., SHIFT or Esc) does not work. Function key F4 can be used to enter the Grub menu.

System Services:

`docker.service` is disabled in the default Ubuntu OS image as it dramatically affects boot time.

The `kexec` utility can be used to reduce the reboot time. Script `/usr/sbin/kexec_reboot` is included in the default Ubuntu 20.04 OS image to run corresponding `kexec` commands.

```
# kexec_reboot
```

4.1.6 DHCP Client Configuration

```
/etc/dhcp/dhclient.conf:
send vendor-class-identifier "NVIDIA/BF/DP";
interface "oob_net0" {
  send vendor-class-identifier "NVIDIA/BF/OOB";
    }
```

4.1.7 Ubuntu Dual Boot Support

BlueField DPU may be installed with support for dual boot. That is, two identical images of the BlueField OS may be installed using BFB.

The following is a proposed SSD partitioning layout for 119.24 GB SSD:

```
Device             Start       End   Sectors  Size Type
/dev/nvme0n1p1      2048    104447    102400   50M EFI System
/dev/nvme0n1p2    104448 114550086 114445639 54.6G Linux filesystem
/dev/nvme0n1p3 114550087 114652486    102400   50M EFI System
/dev/nvme0n1p4 114652487 229098125 114445639 54.6G Linux filesystem
/dev/nvme0n1p5 229098126 250069645  20971520   10G Linux filesystem
```

Where:

- `/dev/nvme0n1p1` - boot EFI partition for the first OS image
- `/dev/nvme0n1p2` - root FS partition for the first OS image
- `/dev/nvme0n1p3` - boot EFI partition for the second OS image
- `/dev/nvme0n1p4` - root FS partition for the second OS image
- `/dev/nvme0n1p5` - common partition for both OS images

For example, the following is a proposed eMMC partitioning layout for a 64GB eMMC:

```
Device             Start       End  Sectors  Size Type
/dev/mmcblk0p1      2048    104447   102400   50M EFI System
/dev/mmcblk0p2    104448  50660334 50555887 24.1G Linux filesystem
/dev/mmcblk0p3  50660335  50762734   102400   50M EFI System
/dev/mmcblk0p4  50762735 101318621 50555887 24.1G Linux filesystem
/dev/mmcblk0p5 101318622 122290141 20971520   10G Linux filesystem
```

Where:

- `/dev/mmcblk0p1` - boot EFI partition for the first OS image
- `/dev/mmcblk0p2` - root FS partition for the first OS image
- `/dev/mmcblk0p3` - boot EFI partition for the second OS image
- `/dev/mmcblk0p4` - root FS partition for the second OS image
- `/dev/mmcblk0p5` - common partition for both OS images

> ⚠ The common partition can be used to store BFB files that will be used for OS image update on the non-active OS partition.

4.1.7.1 Installing Ubuntu OS Image Using Dual Boot

> ⚠ For software upgrade procedure, please refer to section "Upgrading Ubuntu OS Image Using Dual Boot".

Add the values below to the `bf.cfg` configuration file (see section "bf.cfg Parameters" for more information).

```
DUAL_BOOT=yes
```

If EMMC size is ≤16GB, dual boot support is disabled by default, but it can be forced by setting the following parameter in `bf.cfg`:

```
FORCE_DUAL_BOOT=yes
```

To modify the default size of the `/common` partition, add the following parameter:

```
COMMON_SIZE_SECTORS=<number-of-sectors>
```

The number of sectors is the size in bytes divided by the block size (512). For example, for 10GB, the `COMMON_SIZE_SECTORS=$((10*2**30/512))`.

After assigning size for the `/common` partition, what remains is divided equally between the two OS images.

```
# bfb-install --bfb <BFB> --config bf.cfg --rshim rshim0
```

This will result in the Ubuntu OS image to be installed twice on the BlueField DPU.

For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

4.1.7.2 Upgrading Ubuntu OS Image Using Dual Boot

1. Download the new BFB to the BlueField DPU into the `/common` partition. Use `bfb_tool.py` script to install the new BFB on the inactive BlueField DPU partition:

   ```
   /opt/mellanox/mlnx_snap/exec_files/bfb_tool.py --op fw_activate_bfb --bfb <BFB>
   ```

2. Reset BlueField DPU to load the new OS image:

   ```
   /sbin/shutdown -r 0
   ```

   BlueField DPU will now boot into the new OS image.

Use `efibootmgr` utility to manage the boot order if necessary.

- Change the boot order with:

  ```
  # efibootmgr -o
  ```

- Remove stale boot entries with:

  ```
  # efibootmgr -b <E> -B
  ```

  Where `<E>` is the last character of the boot entry (i.e., `Boot000<E>`). You can find that by running:

  ```
  # efibootmgr
  BootCurrent: 0040
  Timeout: 3 seconds
  BootOrder: 0040,0000,0001,0002,0003
  Boot0000* NET-NIC_P0-IPV4
  Boot0001* NET-NIC_P0-IPV6
  Boot0002* NET-NIC_P1-IPV4
  Boot0003* NET-NIC_P1-IPV6
  Boot0040* focal0
  ....2
  ```

Modifying the boot order with `efibootmgr -o` does not remove unused boot options. For example, changing a boot order from 0001,0002, 0003 to just 0001 does not actually remove 0002 and 0003. 0002 and 0003 need to be explicitly removed using `efibootmgr -B`.

4.2 Deploying BlueField Software Using BFB from BMC

It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

⚠ This section assumes that a BlueField DPU has already been installed in a server according to the instructions detailed in the DPU's hardware user guide.

The following table lists an overview of the steps required to install Ubuntu BFB on your DPU:

Step	Procedure	Direct Link
1	Verify that RShim is already running on BMC	Ensure RShim is Running on BMC
2	Change the default credentials using `bf.cfg` file (optional)	Changing Default Credentials Using bf.cfg
3	Install the Ubuntu BFB image	BFB Installation
4	Verify installation completed successfully	Verify BFB is Installed
5	Upgrade the firmware on your DPU	Firmware Upgrade

⚠ It is important to know your device name (e.g., `mt41686_pciconf0`).

MST tool is necessary for this purpose which is installed by default on the DPU.

Run:

```
mst status -v
```

Example output:

```
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded
PCI devices:
------------
DEVICE_TYPE             MST                           PCI       RDMA            NET
NUMA
BlueField2(rev:1)       /dev/mst/mt41686_pciconf0.1   3b:00.1   mlx5_1          net-ens1f1
0

BlueField2(rev:1)       /dev/mst/mt41686_pciconf0     3b:00.0   mlx5_0          net-ens1f0
0
```

4.2.1 Ensure RShim is Running on BMC

Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME
DEV_NAME        usb-1.0
```

This output indicates that the RShim service is ready to use. If you do not receive this output:

1. Restart RShim service:

   ```
   sudo systemctl restart rshim
   ```

2. Verify the current setting again. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME
```

If `DEV_NAME` does not appear, then proceed to "RShim driver not loading on DPU with integrated BMC".

4.2.2 Changing Default Credentials Using bf.cfg

Ubuntu users are prompted to change the default password (ubuntu) for the default user (ubuntu) upon first login. Logging in will not be possible even if the login prompt appears until all services are up ("`DPU is ready`" message appears in `/dev/rshim0/misc`).

> ⚠ Attempting to log in before all services are up prints the following message: "Permission denied, please try again."

Alternatively, Ubuntu users can provide a unique password that will be applied at the end of the BFB installation. This password would need to be defined in a `bf.cfg` configuration file. To set the password for the ubuntu user:

1. Create password hash. Run:

   ```
   # openssl passwd -1
   Password:
   Verifying - Password:
   $1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1
   ```

2. Add the password hash in quotes to the `bf.cfg` file:

   ```
   # vim bf.cfg
   ubuntu_PASSWORD='$1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1'
   ```

   The `bf.cfg` file is used with the `bfb-install` script in the steps that follow.

   > ⚠ Password policy:
   > - Minimum password length - 8
   > - At least one upper-case letter
   > - At least one lower-case letter
   > - At least one numerical character

> ⚠ For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

4.2.3 BFB Installation

To update the software on the NVIDIA® BlueField® device, the BlueField must be booted up without mounting the eMMC flash device. This requires an external boot flow where a BFB (which includes ATF, UEFI, Arm OS, NIC firmware, and initramfs) is pushed from an external host via USB or PCIe. On BlueField devices with an integrated BMC, the USB interface is internally connected to the BMC and

is enabled by default. Therefore, you must verify that the RShim driver is running on the BMC. This provides the ability to push a bootstream over the USB interface to perform an external boot.

The BFB installation procedure consists of the following main stages:

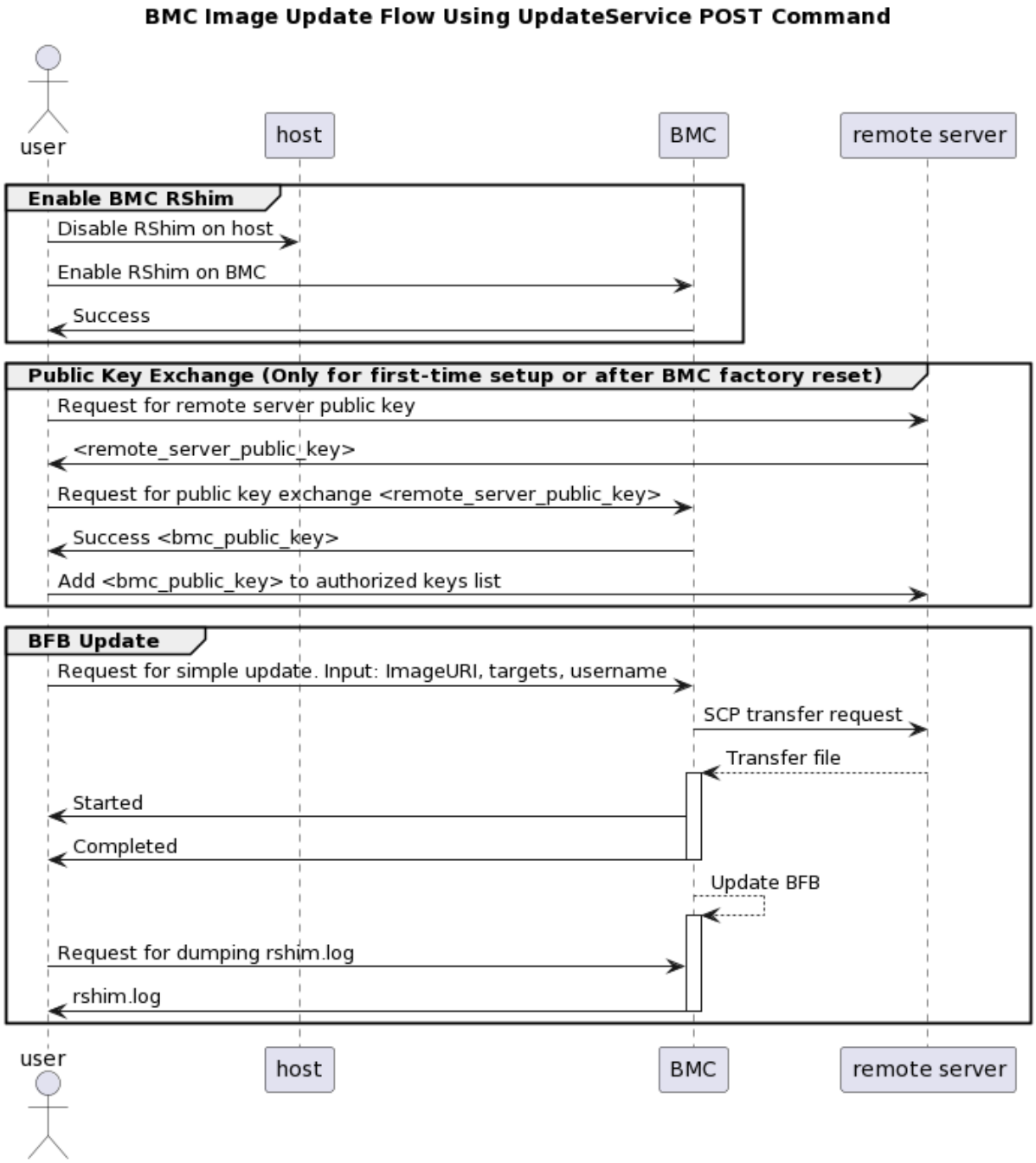
1. Enabling RShim on the BMC. See section "Enable RShim on DPU BMC" for instructions.
2. Initiating the BFB update procedure by transferring the BFB image using one of the following options:
    - Direct SCP
        i. Running an SCP command.
    - Redfish interface
        i. Confirming the identity of the host and BMC—required only during first-time setup or after BMC factory reset.
        ii. Sending a Simple-Update request.

4.2.3.1 Transferring BFB Image

Since the BFB is too large to store on the BMC flash or tmpfs, the image must be written to the RShim device. This can be done by either running SCP directly or using the Redfish interface.

4.2.3.1.1 Redfish Interface

The following is a simple sequence diagram illustrating the flow of the BFB installation process.



The following are detailed instructions outlining each step in the diagram:

1. Confirm the identity of the remote server (i.e., host holding the BFB image) and BMC.

   > Required only during first-time setup or after BMC factory reset.

   a. Run the following on the remote server:

```
ssh-keyscan -t <key_type> <remote_server_ip>
```

Where:

- `key_type` - the type of key associated with the server storing the BFB file (e.g., ed25519)
- `remote_server_ip` - the IP address of the server hosting the BFB file

b. Retrieve the public key of the host holding the BFB image from the response and provide the remote server's credentials to the DPU using the following command:

```
curl -k -u root:'<password>' -H "Content-Type: application/json" -X POST -d
'{"RemoteServerIP":"<remote_server_ip>", "RemoteServerKeyString":"<remote_server_public_key>"}'
https://<bmc_ip>/redfish/v1/UpdateService/Actions/Oem/NvidiaUpdateService.PublicKeyExchange
```

Where:

- `remote_server_ip` - the IP address of the server hosting the BFB file
- `remote_server_public_key` - remote server's public key from the `ssh-keyscan` response, which contains both the type and the public key with a space between the two fields (i.e., "`<type> <public_key>`").
- `bmc_ip` - BMC IP address

c. Extract the BMC public key information (i.e., "`<type> <bmc_public_key> <username>@<hostname>`") from the `PublicKeyExchange` response and append it to the `authorized_keys` file on the host holding the BFB image. This enables passwordless key-based authentication for users.

```
{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "Please add the following public
       key info to ~/.ssh/authorized_keys on the
       remote server",
      "MessageArgs": [
        "<type> <bmc_public_key> root@dpu-bmc"
      ]
    },
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The request completed
       successfully.",
      "MessageArgs": [],
      "MessageId": "Base.1.15.0.Success",
      "MessageSeverity": "OK",
      "Resolution": "None"
        }
    ]
   }
```

d. If the remote server public key must be revoked, use the following command before repeating the previous step:

```
curl -k -u root:'<password>' -H "Content-Type: application/json" -X POST -d
'{"RemoteServerIP":"<remote_server_ip>"}' https://<bmc_ip>/redfish/v1/UpdateService/Actions/Oem/
NvidiaUpdateService.RevokeAllRemoteServerPublicKeys
```

Where:

- `remote_server_ip` - remote server's IP address
- `bmc_ip` - BMC IP address

2. Start BFB image transfer using the following command on the remote server:

```
curl -k -u root:'<password>' -H "Content-Type: application/json" -X POST -d '{"TransferProtocol":"SCP",
"ImageURI":"<image_uri>","Targets":["redfish/v1/UpdateService/FirmwareInventory/DPU_OS"],
"Username":"<username>"}' https://<bmc_ip>/redfish/v1/UpdateService/Actions/UpdateService.SimpleUpdate
```

After the BMC boots, it may take a few seconds (6-8 in NVIDIA® BlueField®-2, and 2 in BlueField-3) until the DPU BSP (`DPU_OS`) is up.

This command uses SCP for the image transfer, initiates a soft reset on the BlueField and then pushes the boot stream. For Ubuntu BFBs, the eMMC is flashed automatically once the bootstream is pushed. On success, a "running" message is received with the current task ID.

Where:

- `image_uri` - the image URI format should be `<remote_server_ip>/<path_to_bfb>`

  `image_uri` path is in reference to the user. For instance, if the BFB image is in `/root/image.bfb` , the user must use `image_uri=<ip>/image.bfb` (not `image_uri=<ip>/root/image.bfb`).

- `username` - username on the remote server
- `bmc_ip` - BMC IP address

  Examples:

  - If RShim is disabled:

```
{
  "error": {
    "@Message.ExtendedInfo": [
      {
        "@odata.type": "#Message.v1_1_1.Message",
        "Message": "The requested resource of type Target named '/dev/rshim0/boot' was not
found.",
        "MessageArgs": [
          "Target",
          "/dev/rshim0/boot"
        ],
        "MessageId": "Base.1.15.0.ResourceNotFound",
        "MessageSeverity": "Critical",
        "Resolution": "Provide a valid resource identifier and resubmit the request."
      }
    ],
    "code": "Base.1.15.0.ResourceNotFound",
    "message": "The requested resource of type Target named '/dev/rshim0/boot' was not
found."
}
```

  - If a username or any other required field is missing:

```
{
  "Username@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The create operation failed because the required property Username was
missing from the request.",
      "MessageArgs": [
        "Username"
      ],
      "MessageId": "Base.1.15.0.CreateFailedMissingReqProperties",
      "MessageSeverity": "Critical",
      "Resolution": "Correct the body to include the required property with a valid value
and resubmit the request if the operation failed."
    }
  ]
}
```

  - If the request is valid and a task is created:

```
{
  "@odata.id":
   "/redfish/v1/TaskService/Tasks/<task_id>",
  "@odata.type": "#Task.v1_4_3.Task",
  "Id": "<task_id>",
  "TaskState": "Running",
```

```
  "TaskStatus": "OK"
}
```

3. Wait 2 seconds and run the following on the host to track image transfer progress:

```
curl -k -u root:'<password>' -X GET https://<bmc_ip>/redfish/v1/TaskService/Tasks/<task_id>
```

> ⚠ The transfer takes ~8 minutes for BlueField-3, and ~40 minutes for BlueField-2. During the transfer, the `PercentComplete` value remains at 0. If no errors occur, the `TaskState` is set to `Running`, and a keep-alive message is generated every 5 minutes with the content "Transfer is still in progress (X minutes elapsed). Please wait". Once the transfer is completed, the `PercentComplete` is set to 100, and the `TaskState` is updated to `Completed`.
>
> Upon failure, a message is generated with the relevant resolution.

Where:

a. `bmc_ip` - BMC IP address

b. `task_id` - task ID

Troubleshooting:

- If host identity is not confirmed or the provided host key is wrong:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Transfer of image '<file_name>' to '/dev/rshim0/boot' failed.",
      "MessageArgs": [
        "<file_name>",
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferFailed",
      "Resolution": " Unknown Host: Please provide server's public key using
PublicKeyExchange ",
      "Severity": "Critical"
    }
…
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Exception",
  "TaskStatus": "Critical"
```

> ℹ In this case, revoke the remote server key (step 1.d.), and repeat steps 1.a. to 1.c.

- If the BMC identity is not confirmed:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Transfer of image '<file_name>' to '/dev/rshim0/boot' failed.",
      "MessageArgs": [
        "<file_name>",
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferFailed",
      "Resolution": "Unauthorized Client: Please use the PublicKeyExchange action to
receive the system's public key and add it as an authorized key on the remote server",
      "Severity": "Critical"
      }
…
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Exception",
  "TaskStatus": "Critical"
```

> In this case, verify that the BMC key has been added correctly to the `authorized_key` file on the remote server.

- If SCP fails:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Transfer of image '<file_name>' to '/dev/rshim0/boot' failed.",
      "MessageArgs": [
        "<file_name>",
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferFailed",
      "Resolution": "Failed to launch SCP",
      "Severity": "Critical"
      }
…
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Exception",
  "TaskStatus": "Critical"
```

- The keep-alive message:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": " <file_name>' is being transferred to '/dev/rshim0/boot'.",
      "MessageArgs": [
        " <file_name>",
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferringToComponent",
      "Resolution": "Transfer is still in progress (5 minutes elapsed): Please wait",
      "Severity": "OK"
    }
…
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Running",
  "TaskStatus": "OK"
```

- Upon completion of transfer of the BFB image to the DPU, the following is received:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Device 'DPU' successfully updated with image '<file_name>'.",
      "MessageArgs": [
        "DPU",
        "<file_name>"
      ],
      "MessageId": "Update.1.0.UpdateSuccessful",
      "Resolution": "None",
      "Severity": "OK"
    },
…
"PercentComplete": 100,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Completed",
  "TaskStatus": "OK"
```

4. Apply the new BFB image:
   BlueField must be restarted to apply the new firmware. To restart BlueField:
   a. Perform a graceful shutdown of the BlueField Arm OS.
   b. Power cycle the server to complete the restart.
   Alternatively, a server reboot may be done instead of power cycle by following these steps:
   a. Graceful shutdown the BlueField Arm OS.

   > Without graceful shutdown of BlueField Arm OS during server reboot, the BlueField Arm side does not undergo a restart process (so only NIC firmware is applied).

b. Wait until completed.
c. Reboot the server (ATF, UEFI, BlueField Arm OS, NIC firmware is applied).

> Server reboot will not restart the BlueField BMC (CEC not applied).

d. Log into BlueField BMC via Redfish and issue a restart (BlueField BMC and CEC is applied).

5. Verify that the new BFB is running by checking its version:

```
curl -k -u root:'<password>' -H "Content-Type: application/json" -X GET https://<bmc_ip>/redfish/v1/
UpdateService/FirmwareInventory/DPU_OS
```

4.2.3.1.2 Direct SCP

```
scp <path_to_bfb> root@<bmc_ip>:/dev/rshim0/boot
```

> For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

4.2.4 Verify BFB is Installed

After installation of the Ubuntu OS is complete, the following note appears in `/dev/rshim0/misc` on first boot:

```
...
INFO[MISC]: Linux up
INFO[MISC]: DPU is ready
```

"DPU is ready" indicates that all the relevant services are up and users can login the system.

After the installation of the Ubuntu 20.04 BFB, the configuration detailed in the following sections is generated.

> Make sure all the services (including cloud-init) are started on BlueField and to perform a graceful shutdown before power cycling the host server.

BlueField OS image version is stored under `/etc/mlnx-release` in the DPU.

```
# cat /etc/mlnx-release
DOCA_2.6.0_BSP_4.6.0_Ubuntu_22.04-<version>
```

4.2.5 Firmware Upgrade

> `mlxfwreset` is not supported in this release. Please perform a graceful shutdown, and power cycle the host where `mlxfwreset` is requested.

To upgrade firmware:

1. Set a temporary static IP on the host. Run:

```
sudo ip addr add 192.168.100.1/24 dev tmfifo_net0
```

2. SSH to your DPU via 192.168.100.2 (preconfigured). The default credentials for Ubuntu are as follows.

Username	Password
ubuntu	Set during installation

   For example:

```
ssh ubuntu@192.168.100.2
Password: <unique-password>
```

3. Upgrade the firmware on the DPU. Run:

```
sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl --force-fw-update
```

   Example output:

```
Device #1:
----------

  Device Type:      BlueField-2
  [...]
  Versions:         Current        Available
     FW             <Old_FW>       <New_FW>
```

   Important! To apply NVConfig changes, stop here and follow the steps in section "Updating NVConfig Params".

4. Perform a graceful shutdown and power cycle the host for the changes to take effect.

4.2.6 Updating NVConfig Params

1. Reset the `nvconfig` params to their default values:

```
# sudo mlxconfig -d /dev/mst/<device-id> -y reset

Reset configuration for device /dev/mst/<device-name>? (y/n) [n] : y
Applying... Done!
-I- Please reboot machine to load new configurations.
```

   To learn the device ID of the DPUs on your setup, run:

```
mst start
mst status -v
```

   Example output:

```
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded
PCI devices:
```

```
------------
DEVICE_TYPE             MST                           PCI       RDMA           NET
NUMA
BlueField2(rev:1)       /dev/mst/mt41686_pciconf0.1   3b:00.1   mlx5_1         net-ens1f1
0

BlueField2(rev:1)       /dev/mst/mt41686_pciconf0     3b:00.0   mlx5_0         net-ens1f0
0

BlueField3(rev:1)       /dev/mst/mt41692_pciconf0.1   e2:00.1   mlx5_1         net-
ens7f1np1               4

BlueField3(rev:1)       /dev/mst/mt41692_pciconf0     e2:00.0   mlx5_0         net-
ens7f0np0               4
```

The device IDs for the BlueField-2 and BlueField-3 DPUs in this example are `/dev/mst/mt41686_pciconf0` and `/dev/mst/mt41692_pciconf0` respectively.

2. (Optional) Enable NVMe emulation. Run:

```
sudo mlxconfig -d <device-id> s NVME_EMULATION_ENABLE=1
```

3. Skip this step if your BlueField DPU is Ethernet only. Please refer to section "Supported Platforms and Interoperability" under the Release Notes to learn your DPU type.
   If you have a VPI DPU, the default link type of the ports will be configured to IB. If you want to change the link type to Ethernet, please run the following configuration:

```
sudo mlxconfig -d <device-id> s LINK_TYPE_P1=2 LINK_TYPE_P2=2
```

4. Perform a graceful shutdown and power cycle the host for the `mlxconfig` settings to take effect.

4.3 Deploying NVIDIA Converged Accelerator

> It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

This section assumes that you have installed the BlueField OS BFB on your NVIDIA® Converged Accelerator using any of the following guides:

- Deploying DPU OS Using BFB from Host
- Deploying BlueField Software Using BFB from BMC
- (4.5.5-LTS) Deploying BlueField Software Using BFB with PXE

NVIDIA® CUDA® (GPU driver) must be installed in order to use the GPU. For information on how to install CUDA on your Converged Accelerator, refer to NVIDIA CUDA Installation Guide for Linux.

4.3.1 Configuring Operation Mode

After installing the BFB, you may now select the mode you want your NVIDIA Converged Accelerator to operate in.

- Standard (default) – the NVIDIA® BlueField® DPU and the GPU operate separately (GPU is owned by the host)
- BlueField-X – the GPU is exposed to the DPU and is no longer visible on the host (GPU is owned by the DPU)

It is important to know your device name (e.g., `mt41686_pciconf0`).

MST tool is necessary for this purpose which is installed by default on the DPU.

Run:

```
mst status -v
```

Example output:

```
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded
PCI devices:
------------
DEVICE_TYPE             MST                           PCI       RDMA           NET
NUMA
BlueField2(rev:1)       /dev/mst/mt41686_pciconf0.1   3b:00.1   mlx5_1         net-ens1f1
0

BlueField2(rev:1)       /dev/mst/mt41686_pciconf0     3b:00.0   mlx5_0         net-ens1f0
0
```

4.3.1.1 BlueField-X Mode

1. Run the following command from the host:

```
mlxconfig -d /dev/mst/<device-name> s PCI_DOWNSTREAM_PORT_OWNER[4]=0xF
```

2. Perform a graceful shutdown and power cycle the host for the configuration to take effect.

4.3.1.2 Standard Mode

To return the DPU from BlueField-X mode to Standard mode:

1. Run the following command from the host:

```
mlxconfig -d /dev/mst/<device-name> s PCI_DOWNSTREAM_PORT_OWNER[4]=0x0
```

2. Perform a graceful shutdown and power cycle the host for the configuration to take effect.

4.3.2 Verifying Configured Operational Mode

Use the following command from host or from DPU:

```
$ sudo mlxconfig -d /dev/mst/<device-name> q PCI_DOWNSTREAM_PORT_OWNER[4]
```

Example of Standard mode output:

```
Device #1:
----------

[...]

Configurations:                                 Next Boot
         PCI_DOWNSTREAM_PORT_OWNER[4]           DEVICE_DEFAULT(0)
```

Example of BlueField-X mode output:

```
Device #1:
----------

[...]

Configurations:                                 Next Boot
         PCI_DOWNSTREAM_PORT_OWNER[4]           EMBEDDED_CPU(15)
```

4.3.3 Verifying GPU Ownership

The following are example outputs for when the DPU is configured to BlueField-X mode.

The GPU is no longer visible from the host:

```
root@host:~# lspci | grep -i nv
None
```

The GPU is now visible from the DPU:

```
ubuntu@dpu:~$ lspci | grep -i nv
06:00.0 3D controller: NVIDIA Corporation GA20B8 (rev a1)
```

4.3.4 CEC and BMC Firmware Operations

Firmware upgrade of BMC and CEC components using BMC can be performed from a remote server using the Redfish interface. The following table presents commands available for performing the upgrade:

No.	Function	Command	Required for BMC/ CEC Update	Description
1	Establish Redfish connection session	`export token=`curl -k -H "Content-Type: application/json" -X POST https://<bmc_ip>/login -d '{"username" : "root", "password" : "<password>"}' | grep token | awk '{print $2;}' | tr -d '"'` Where: • `bmc_ip` - BMC IP address • `password` - Password of root account	BMC CEC	Establish Redfish connection session
2	Trigger a secure firmware update	`curl -k -H "X-Auth-Token: <token>" -H "Content-Type: application/octet-stream" -X POST -T <package_path> https://<bmc_ip>/redfish/v1/UpdateService/update` Where: • `bmc_ip` - BMC IP address • `token` - session token received when establishing connection • `package_path` - firmware update package path	BMC CEC	Triggers the secure update and starts tracking the secure update progress

No.	Function	Command	Required for BMC/ CEC Update	Description
3	Track secure firmware update progress	``` curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/TaskService/Tasks ``` Find the current task ID in the response and use it for checking the progress: ``` curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/TaskService/Tasks/<task_id> | jq -r ' .PercentComplete' ``` Where: • `bmc_ip` - BMC IP address • `token` - session token received when establishing connection • `task_id` - Task ID	BMC CEC	Tracks the firmware update progress
4	Reset/ reboot a BMC	``` curl -k -H "X-Auth-Token: <token>" -H "Content-Type: application/json" -X POST -d '{"ResetType": "GracefulRestart"}' https://<bmc_ip>/redfish/v1/Managers/Bluefield_BMC/Actions/Manager.Reset ``` Where: • `bmc_ip` - BMC IP address • `token` - session token received when establishing connection	BMC	Resets/reboots the BMC
5	Fetch running BMC firmware version	For BlueField-3: ``` curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/UpdateService/FirmwareInventory/BMC_Firmware | jq -r ' .Version' ``` Where: • `bmc_ip` - BMC IP address • `token` - session token received when establishing connection For BlueField-2: ``` curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/UpdateService/FirmwareInventory ``` Fetch the current firmware ID and then perform: ``` curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/UpdateService/FirmwareInventory/<firmware_id>_BMC_Firmware | jq -r ' .Version' ``` Where: • `bmc_ip` - BMC IP address • `token` - session token received when establishing connection	BMC	Fetches the running firmware version from BMC

No.	Function	Command	Required for BMC/CEC Update	Description
6	Fetch running CEC firmware version	`curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/UpdateService/FirmwareInventory/Bluefield_FW_ERoT | jq -r ' .Version'` Where: • `bmc_ip` - BMC IP address • `token` - session token received when establishing connection	CEC	Fetches the running firmware version from CEC

4.3.4.1 BMC Update

Firmware update takes about 12 minutes.

After initiating the BMC secure update with the command #2 to from the previous table, a response similar to the following is received:

```
curl -k -H "X-Auth-Token: <token>" -H "Content-Type: application/octet-stream" -X POST -T <package_path> https://
<bmc_ip>/redfish/v1/UpdateService

{
  "@odata.id": "/redfish/v1/TaskService/Tasks/0",
  "@odata.type": "#Task.v1_4_3.Task",
  "Id": "0",
  "TaskState": "Running"
}
```

Command #3 from the previous table can be used to track secure firmware update progress. For instance:

```
curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/TaskService/Tasks/0 | jq -r
' .PercentComplete'

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current Dload  Upload   Total   Spent
Left  Speed
100  2123  100  2123    0     0  38600      0 --:--:-- --:--:-- --:--:-- 37910
20
```

Command #3 is used to verify the task has completed because during the update procedure the reboot option is disabled. When "PercentComplete" reaches 100, command #4 is used to reboot the BMC. For example:

```
curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/TaskService/Tasks/0 | jq -r
' .PercentComplete'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current Dload  Upload   Total   Spent
Left  Speed
100  3822  100  3822    0     0  81319      0 --:--:-- --:--:-- --:--:-- 81319
100

curl -k -H "X-Auth-Token: <token>" -H "Content-Type: application/octet-stream" -X POST -d '{"ResetType":
"GracefulRestart"}' https://<bmc_ip>/redfish/v1/Managers/Bluefield_BMC/Actions/Manager.Reset
{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The request completed successfully.",
      "MessageArgs": [],
      "MessageId": "Base.1.13.0.Success",
      "MessageSeverity": "OK",
      "Resolution": "None"
    }
  ]
```

```
}
```

Command #5 can be used to verify the current BMC firmware version after reboot:

- For BlueField-3:

```
curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/UpdateService/FirmwareInventory/
BMC_Firmware | jq -r ' .Version'

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current Dload  Upload   Total
Spent    Left  Speed
100   513  100   513    0     0   9679      0 --:--:-- --:--:-- --:--:--  9679
```

- For BlueField-2:

  a. Fetch the firmware ID from `FirmwareInventory`:

```
curl -k -H "X-Auth-Token: <token>" -X GET https:/<bmc_ip>/redfish/v1/UpdateService/
FirmwareInventory/
{
  "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory",
  "@odata.type": "#SoftwareInventoryCollection.SoftwareInventoryCollection",
  "Members": [
    {
      "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/8c8549f3_BMC_Firmware"
…
```

  b. Use command #5 with the fetched firmware ID in the previous step:

```
curl -k -H "X-Auth-Token: <token>" -X GET https:/<bmc_ip>/redfish/v1/UpdateService/
FirmwareInventory/8c8549f3_BMC_Firmware | jq -r ' .Version'

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   471  100   471    0     0    622      0 --:--:-- --:--:-- --:--:--   621
bmc-23.04
```

4.3.4.2 CEC Update

> Firmware update takes about 20 seconds.

After initiating the BMC secure update with the command #2 to from the previous table, a response similar to the following is received:

```
curl -k -H "X-Auth-Token: <token>" -H "Content-Type: application/octet-stream" -X POST -T <package_path> https://
<bmc_ip>/redfish/v1/UpdateService
{
  "@odata.id": "/redfish/v1/TaskService/Tasks/0",
  "@odata.type": "#Task.v1_4_3.Task",
  "Id": "0",
  "TaskState": "Running"
}
```

Command #3 can be used to track the progress of the CEC firmware update. For example:

```
curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/TaskService/Tasks/0 | jq -r
' .PercentComplete'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current Dload  Upload   Total   Spent
Left  Speed
100  2123  100  2123    0     0  38600      0 --:--:-- --:--:-- --:--:-- 37910
100
```

After the CEC secure update operation is complete, perform a graceful shutdown and a power cycle or cold reset of the BlueField-3 DPU must be manually triggered to apply the changes once the update is finished.

Command #6 can be used to verify the current CEC firmware version after reboot:

```
curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/UpdateService/FirmwareInventory/
Bluefield_FW_ERoT | jq -r ' .Version'

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   421  100   421    0     0   1172      0 --:--:-- --:--:-- --:--:--  1172
19-4
```

4.3.4.3 CEC Background Update Status

> This section is relevant only for BlueField-3.

BMC and CEC have an active and inactive copy of the same firmware image on their respective firmware SPI flash. The firmware update updates the inactive copy, and on a successful boot from the newly updated and active image, the inactive image (e.g., the previous active image) is updated with the latest image.

> Firmware update cannot be initiated if the background copy is in progress.

To check the status of the background update:

```
curl -k -H "X-Auth-Token: <token>" -X GET https://<bmc_ip>/redfish/v1/Chassis/Bluefield_ERoT
...
  "Oem": {
    "Nvidia": {
      "@odata.type": "#NvidiaChassis.v1_0_0.NvidiaChassis",
      "AutomaticBackgroundCopyEnabled": true,
      "BackgroundCopyStatus": "Completed",
      "InbandUpdatePolicyEnabled": true
    }
  }
…
```

> The background update initially indicates `InProgress` while the inactive copy of the image is being updated with the copy.

4.3.4.4 Possible Error Codes

> This section is relevant only for BlueField-3.

Fault	Diagnosis and Possible Solution
Connection to BMC breaks during firmware package transfer	• Redfish task URI is not returned by the Redfish server • The Redfish server (if operational) is in `idle` state • After a reboot of BMC, or restart/recovery of the Redfish server, the Redfish server is in `idle` state A new firmware update can be attempted by the Redfish client.

Fault	Diagnosis and Possible Solution

Connection to BMC breaks during firmware update	• Redfish task URI previously returned by the Redfish server is no longer accessible
	• The Redfish server (if operational) is in one of the following states:
	  • In `idle` state, if the firmware update has completed
	  • In `update` state, if the firmware update is still ongoing
	• After a BMC reboot, or the restart/recovery of the Redfish server, the Redfish server is in `idle` state
	A new firmware update can be attempted by the Redfish client.

Two firmware update requests are initiated	The Redfish server blocks the second firmware update request and returns the following: • HTTP code 400 "Bad Request" • Redfish message based on standard registry entry `UpdateInProgress` Check the status of the ongoing firmware update by looking at the `TaskCollection` resource.
Redfish task hangs	• Redfish task URI that previously returned by the Redfish server is no longer accessible • PLDM-based firmware update progresses • After a reboot of BMC, or restart/recovery of the Redfish server, the Redfish server us in `idle` state A new firmware update can be attempted by the Redfish client.
BMC-EROT communication failure during image transfer	The Redfish task monitoring the firmware update indicates a failure: • `TaskState` is set to `Exception` • `TaskStatus` is set to `Warning` • `Messages` array in the task includes an entry based on the standard registry `Update.1.0.0.TransferFailed` indicating the components that failed during image transfer The Redfish client may retry the firmware update.
Firmware update fails	The Redfish task monitoring the firmware update indicates a failure: • `TaskState` is set to `Exception` • `TaskStatus` is set to `Warning` • `Messages` array in the task includes an entry describing the error The Redfish client may retry the firmware update.
ERoT failure (not responding)	The Redfish task monitoring the firmware update indicates a failure: • `TaskState` is set to `Canceled` • `TaskStatus` is set to `Warning` • `Messages` array in the task includes an entry describing the error • The Redfish client reports the error The Redfish client may retry the firmware update.
Firmware image validation failure	The Redfish task monitoring the firmware update indicates a failure: • `TaskState` is set to `Exception` • `TaskStatus` is set to `Warning` • `Messages` array in the task includes an entry based on the standard registry `Update.1.0.0.VerificationFailed` to indicate the component for which verification failed • The Redfish client reports the error The Redfish client might retry the firmware update.
Power loss before activation command is sent	• The Redfish server is in `idle` state A new firmware update can be attempted by the Redfish client.

Fault	Diagnosis and Possible Solution
Firmware activation failure	The Redfish task monitoring the firmware update indicates a failure: • `TaskState` is set to `Exception` • `TaskStatus` is set to `Warning` • `Messages` array in the task includes an entry based on the standard registry `Update.1.0.ActivateFailed` The Redfish client may retry the firmware update.
Push to BMC firmware package greater than 200 MB	• No Redfish task is created • `Messages` array in the task includes an entry based on the standard registry `Base.1.8.1.ResourceExhaustion` and a request to retry the operation is given.

4.3.5 GPU Firmware

4.3.5.1 Get GPU Firmware

```
smbpbi: (See SMBPBI spec)

root@dpu:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x00 0x80 s
root@dpu:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x00 0x5f
root@dpu:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x39 0x32 0x2e 0x30
root@dpu:~#
root@dpu:~#
root@dpu:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x01 0x80 s
root@dpu:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x01 0x5f
root@dpu:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x30 0x2e 0x36 0x42
root@dpu:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x02 0x80 s
root@dpu:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x02 0x5f
root@dpu:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x2e 0x30 0x30 0x2e
root@dpu:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x03 0x80 s
root@dpu:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x03 0x5f
root@dpu:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x30 0x31 0x00 0x00
root@dpu:~#

39 32 2e 30 30 2e 36 42 2e 30 30 2e 30 31 00 00   92.00.6B.00.01
```

4.3.5.2 Updating GPU Firmware

```
root@dpu:~# scp root@10.23.201.227:/<path-to-fw-bin>/1004_0230_891__92006B0001-dbg-ota.bin /tmp/gpu_images/
root@10.23.201.227's password:
1004_0230_891__92006B0001-dbg-ota.bin                                         100%  384KB 384.4KB/s   00:01

root@dpu:~# cat /tmp/gpu_images/progress.txt
TaskState="Running"
TaskStatus="OK"
TaskProgress="50"

root@dpu:~# cat /tmp/gpu_images/progress.txt
TaskState="Running"
TaskStatus="OK"
TaskProgress="50"

root@dpu:~# cat /tmp/gpu_images/progress.txt
TaskState=Frimware update succeeded.
TaskStatus=OK
TaskProgress=100
```

4.4 Installing Repo Package on Host Side

> This section assumes that a BlueField DPU has already been installed in a server according to the instructions detailed in the DPU's hardware user guide.

The following procedure instructs users on upgrading DOCA local repo package for host.

4.4.1 Removing Previously Installed DOCA Runtime Packages

If an older DOCA software version is installed on your host, make sure to uninstall it before proceeding with the installation of the new version:

Ubuntu	`host# for f in $( dpkg --list | grep doca | awk '{print $2}' ); do echo $f ; apt remove --purge $f -y ; done` `host# sudo apt-get autoremove`

CentOS/ RHEL	`host# for f in $(rpm -qa |grep -i doca ) ; do yum -y remove $f; done`
	`host# yum autoremove`
	`host# yum makecache`

4.4.2 Downloading DOCA Runtime Packages

The following table provides links to DOCA Runtime packages depending on the OS running on your host.

OS	Arch	Link
Alinux 3.2	x86	doca-host-repo-alinux32-2.5.2-0.0.6.2.5.2003.1.al8.23.10.3.2.2.0.x86_64.rpm
BCLinux 21.10 SP2	aarch64	doca-host-repo-bclinux2110sp2-2.5.2-0.0.6.23.10.3.2.2.0.oe1.bclinux.aarch64.rpm
	x86	doca-host-repo-bclinux2110sp2-2.5.2-0.0.6.23.10.3.2.2.0.oe1.bclinux.x86_64.rpm
CTyunOS 2.0	aarch64	doca-host-repo-ctyunos20-2.5.2-0.0.6.23.10.3.2.2.0.ctl2.aarch64.rpm
	x86	doca-host-repo-ctyunos20-2.5.2-0.0.6.23.10.3.2.2.0.ctl2.x86_64.rpm
CTyunOS 23.01	aarch64	doca-host-repo-ctyunos2301-2.5.2-0.0.6.23.10.3.2.2.0.ctl3.aarch64.rpm
	x86	doca-host-repo-ctyunos2301-2.5.2-0.0.6.2.5.2003.1.ctl3.23.10.3.2.2.0.x86_64.rpm
Debian 10.13	x86	doca-host-repo-debian1013_2.5.2-0.0.6.2.5.2003.1.23.10.3.2.2.0_amd64.deb
Debian 10.8	x86	doca-host-repo-debian108_2.5.2-0.0.6.2.5.2003.1.23.10.3.2.2.0_amd64.deb
Debian 10.9	x86	doca-host-repo-debian109_2.5.2-0.0.6.23.10.3.2.2.0_amd64.deb
Debian 12.5	aarch64	doca-host-repo-debian125_2.5.2-0.0.6.2.5.2003.1.23.10.3.2.2.0_arm64.deb

OS	Arch	Link
	x86	doca-host-repo-debian125_2.5.2-0.0.6.2.5.2003.1.23.10.3.2.2.0_amd64.deb
Kylin 1.0	aarch64	doca-host-repo-kylin10sp2-2.5.2-0.0.6.23.10.3.2.2.0.ky10.aarch64.rpm
	x86	doca-host-repo-kylin10sp2-2.5.2-0.0.6.23.10.3.2.2.0.ky10.x86_64.rpm
Oracle Linux 7.9	x86	doca-host-repo-ol79-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
Oracle Linux 8.4	x86	doca-host-repo-ol84-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
Oracle Linux 8.6	x86	doca-host-repo-ol86-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
Oracle Linux 8.7	x86	doca-host-repo-ol87-2.5.2-0.0.6.2.5.2003.1.el8.23.10.3.2.2.0.x86_64.rpm
Oracle Linux 9.0	x86	doca-host-repo-ol90-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
openEuler 20.03 SP3	aarch64	doca-host-repo-openeuler2003sp3-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-openeuler2003sp3-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
openEuler 22.03	aarch64	doca-host-repo-openeuler2203-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-openeuler2203-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 7.2	x86	doca-host-repo-rhel72-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 7.4	x86	doca-host-repo-rhel74-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 7.6	aarch64	doca-host-repo-rhel76-2.5.2-0.0.6.2.5.2003.1.el7a.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel76-2.5.2-0.0.6.2.5.2003.1.el7.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 7.7	x86	doca-host-repo-rhel77-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 7.8	x86	doca-host-repo-rhel78-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 7.9	x86	doca-host-repo-rhel79-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 8.0	x86	doca-host-repo-rhel80-2.5.2-0.0.6.2.5.2003.1.el8.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 8.1	aarch64	doca-host-repo-rhel81-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel81-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 8.2	x86	doca-host-repo-rhel82-2.5.2-0.0.6.2.5.2003.1.el8.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 8.3	aarch64	doca-host-repo-rhel83-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel83-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 8.4	aarch64	doca-host-repo-rhel84-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel84-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/CentOS 8.5	aarch64	doca-host-repo-rhel85-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel85-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/Rocky 8.6	aarch64	doca-host-repo-rhel86-2.5.2-0.0.6.2.5.2003.1.el8.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel86-2.5.2-0.0.6.2.5.2003.1.el8.23.10.3.2.2.0.x86_64.rpm

OS	Arch	Link
RHEL/Rocky 8.8	aarch64	doca-host-repo-rhel88-2.5.2-0.0.6.2.5.2003.1.el8.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel88-2.5.2-0.0.6.2.5.2003.1.el8.23.10.3.2.2.0.x86_64.rpm
RHEL/Rocky 8.9	aarch64	doca-host-repo-rhel89-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel89-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/Rocky 8.10	aarch64	doca-host-repo-rhel810-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel810-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/Rocky 9.0	aarch64	doca-host-repo-rhel90-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel90-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/Rocky 9.1	aarch64	doca-host-repo-rhel91-2.5.2-0.0.6.2.5.2003.1.el9.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel91-2.5.2-0.0.6.2.5.2003.1.el9.23.10.3.2.2.0.x86_64.rpm
RHEL/Rocky 9.2	aarch64	doca-host-repo-rhel92-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel92-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/Rocky 9.3	aarch64	doca-host-repo-rhel93-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel93-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
RHEL/Rocky 9.4	aarch64	doca-host-repo-rhel94-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-rhel94-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
SLES 12 SP4	aarch64	doca-host-repo-sles12sp4-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-sles12sp4-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
SLES 12 SP5	aarch64	doca-host-repo-sles12sp5-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-sles12sp5-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
SLES 15 SP2	aarch64	doca-host-repo-sles15sp2-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-sles15sp2-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
SLES 15 SP3	aarch64	doca-host-repo-sles15sp3-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-sles15sp3-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
SLES 15 SP4	aarch64	doca-host-repo-sles15sp4-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-sles15sp4-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
SLES 15 SP5	aarch64	doca-host-repo-sles15sp5-2.5.2-0.0.6.23.10.3.2.2.0.aarch64.rpm
	x86	doca-host-repo-sles15sp5-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm

OS	Arch	Link
SLES 15 SP6	x86	doca-host-repo-sles15sp6-2.5.2-0.0.6.23.10.3.2.2.0.x86_64.rpm
Ubuntu 18.04	x86	doca-host-repo-ubuntu1804_2.5.2-0.0.6.2.5.2003.1.23.10.3.2.2.0_amd64.deb
Ubuntu 20.04	x86	doca-host-repo-ubuntu2004_2.5.2-0.0.6.2.5.2003.1.23.10.3.2.2.0_amd64.deb
Ubuntu 22.04	aarch64	doca-host-repo-ubuntu2204_2.5.2-0.0.6.2.5.2003.1.23.10.3.2.2.0_arm64.deb
	x86	doca-host-repo-ubuntu2204_2.5.2-0.0.6.2.5.2003.1.23.10.3.2.2.0_amd64.deb
UOS 20 1040d	aarch64	doca-host-repo-uos201040_2.5.2-0.0.6.23.10.3.2.2.0_arm64.deb
	x86	doca-host-repo-uos201040_2.5.2-0.0.6.23.10.3.2.2.0_amd64.deb

4.4.3 Installing Local Repo Package for Host Dependencies

1. Install DOCA local repo package for host:

OS	Procedure

Ubuntu	a. Download the DOCA SDK and DOCA Runtime packages from Downloading DOCA Runtime Packages section for the host.
	b. Unpack the deb repo. Run:
	`host# sudo dpkg -i doca-host-repo-ubuntu<version>_amd64.deb`
	c. Perform apt update. Run:
	`host# sudo apt-get update`
	d. Run `apt install` for DOCA runtime, tools, and SDK:
	`host# sudo apt install -y doca-runtime doca-sdk`

CentOS	a. Download the DOCA SDK and DOCA Runtime packages from Downloading DOCA Runtime Packages section for the x86 host.
	b. Install the following software dependencies. Run:
	`host# sudo yum install -y epel-release`
	c. For CentOS 8.2 only, also run:
	`host# yum config-manager --set-enabled PowerTools`
	d. Unpack the RPM repo. Run:
	`host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`
	e. Run `yum install` for DOCA runtime, tools, and SDK.
	`host# sudo yum install -y doca-runtime doca-sdk`

OS	Procedure

RHEL	a. Open a RedHat account.
	i. Log into RedHat website via the developers tab.
	ii. Create a developer user.
	b. Run:
	`host# subscription-manager register --username=<username> --password=PASSWORD`
	To extract pool ID:
	`host# subscription-manager list --available --all`
	`...`
	`Subscription Name: Red Hat Developer Subscription for Individuals`
	`Provides: Red Hat Developer Tools (for RHEL Server for ARM)`
	`...`
	`Red Hat CodeReady Linux Builder for x86_64`
	`...`
	`Pool ID: <pool-id>`
	`...`
	And use the pool ID for the `Subscription Name` and `Provides` that include `Red Hat CodeReady Linux Builder for x86_64`.
	c. Run:
	`host# subscription-manager attach --pool=<pool-id>`
	`host# subscription-manager repos --enable codeready-builder-for-rhel-8-x86_64-rpms`
	`host# yum makecache`
	d. Install the DOCA local repo package for host. Run:
	`host# rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`
	`host# sudo yum install -y doca-runtime doca-sdk`
	e. Sign out from your RHEL account. Run:
	`host# subscription-manager remove --all`
	`host# subscription-manager unregister`

2. Assign a dynamic IP to `tmfifo_net0` interface (RShim host interface).

```
host# ifconfig tmfifo_net0 192.168.100.1 netmask 255.255.255.252 up
```

3. Verify that RShim is active.

```
host# sudo systemctl status rshim
```

This command is expected to display "`active (running)`". If RShim service does not launch automatically, run:

```
host# sudo systemctl enable rshim
host# sudo systemctl start rshim
```

4.5 Installing Popular Linux Distributions on BlueField

4.5.1 Building Your Own BFB Installation Image

Users wishing to build their own customized NVIDIA® BlueField® OS image can use the BFB build environment. See this GitHub webpage for more information.

> ⚠ For any customized BlueField OS image to boot on the UEFI secure-boot-enabled DPU (default DPU secure boot setting), the OS must be either signed with an existing key in the UEFI DB (e.g., the Microsoft key), or UEFI secure boot must be disabled. See "Secure Boot" and its subpages for more details.

4.5.2 Running RedHat on BlueField

In general, running RedHat Enterprise Linux or CentOS on BlueField is similar to setting it up on any other ARM64 server.

A driver disk is required to support the eMMC hardware typically used to install the media onto. The driver disk also supports the tmfifo networking interface that allows creating a network interface over the USB or PCIe connection to an external host. For newer RedHat releases, or if the specific storage or networking drivers mentioned are not needed, you can skip the driver disk.

The way to manage boot flow components with BlueField is through grub boot manager. The installation should create a `/boot/efi` VFAT partition that holds the binaries visible to UEFI for bootup. The standard grub tools then manage the contents of that partition, and the UEFI EEPROM persistent variables, to control the boot.

It is also possible to use the BlueField runtime distribution tools to directly configure UEFI to load the kernel and initramfs from the UEFI VFAT boot partition if desired, but typically using grub is preferred. In particular, you would need to explicitly copy the kernel image to the VFAT partition whenever it is upgraded so that UEFI could access it; normally it is kept on an XFS partition.

4.5.2.1 Provisioning ConnectX Firmware

Prior to installing RedHat, you should ensure that the ConnectX SPI ROM firmware has been provisioned. If the BlueField is connected to an external host via PCIe, and is not running in Secure Boot mode, this is typically done by using MFT on the external host to provision the BlueField. If the BlueField is connected via USB or is configured in Secure Boot mode, you must provision the SPI ROM by booting a dedicated bootstream that allows the SPI ROM to be configured by the MFT running on the BlueField Arm cores.

There are multiple ways to access the RedHat installation media from a BlueField device for installation.

1. You may use the primary ConnectX interfaces on the BlueField to reach the media over the network.
2. You may configure a USB or PCIe connection to the BlueField as a network bridge to reach the media over the network.

> Requires installing and running the RShim drivers on the host side of the USB or PCIe connection.

3. You may connect other network or storage devices to the BlueField via PCIe and use them to connect to or host the RedHat install media.

> This method has not been tested.

In principle, it is possible to perform the installation according to the second method above without first provisioning the ConnectX SPI ROM, but since you need to do that provisioning anyway, it is recommended to perform it first. In particular, the PCIe network interface available via the external host's RShim driver is likely too slow prior to provisioning to be usable for a distribution installation.

4.5.2.2 Managing Driver Disk

NVIDIA provides a number of pre-built driver disks, as well as a documented flow for building one for any particular RedHat version.

Normally a driver disk can be placed on removable media (like a CDROM or USB stick) and is auto-detected by the RedHat installer. However, since BlueField has no removable media slots, you must provide it over the network. Although, if you are installing over the network connection via the PCIe/USB link to an external host, you will not have a network connection either. As a result, the procedure documented is for modifying the default RedHat `images/pxeboot/initrd.img` file to include the driver disk itself.

To create the updated `initrd.img`, you should locate the `image/pxeboot` directory in the RedHat installation media. This will have a kernel image file (vmlinuz) and `initrd.img` (initial RAM disk). The `bluefield_dd/update-initrd.sh` script takes the path to the `initrd.img` as an argument and adds the appropriate BlueField driver disk ISO file to the `initrd.img`.

When booting the installation media, make sure to include `inst.dd=/bluefield_dd.iso` on the kernel command line, which will instruct Anaconda to use that driver disk, enabling the use of the IP over USB/PCIe link (`tmfifo`) and the DesignWare eMMC (`dw_mmc`).

4.5.3 Installing Official CentOS Distributions

Contact NVIDIA Enterprise Support for information on the installation of CentOS distributions.

4.5.4 BlueField Linux Drivers

The following table lists the BlueField drivers which are part of the Linux distribution.

Driver	Description	BlueField	BlueField-2	BlueField-3
`bluefield-edac`	BlueField-specific EDAC driver	✓	✓	✗

Driver	Description	BlueField	BlueField-2	BlueField-3
`gpio-mlxbf`	GPIO driver	✓	✗	✗
`gpio-mlxbf2`	GPIO driver	✗	✓	✗
`gpio-mlxbf3`	GPIO driver	✗	✗	✓
`i2c-mlx`	I2C bus driver (`i2c-mlxbf.c` upstream)	✗	✓	✗
`ipmb-dev-int`	Driver needed to receive IPMB messages from a BMC and send a response back. This driver works with the I2C driver and a user-space program such as OpenIPMI.	✓	✓	✗
`ipmb-host`	Driver needed on the DPU to send IPMB messages to the BMC on the IPMB bus. This driver works with the I2C driver. It only loads successfully if it executes a successful handshake with the BMC.	✓	✓	✗
`mlxbf-gige`	Gigabit Ethernet driver	✗	✓	✓
`mlxbf-livefish`	BlueField HCA firmware burning driver. This driver supports burning firmware for the embedded HCA in the BlueField SoC.	✓	✓	✗
`mlxbf-pka`	BlueField PKA kernel module	✗	✓	✓
`mlxbf-pmc`	Performance monitoring counters. The driver provides access to available performance modules through the `sysfs` interface. The performance modules in BlueField are present in several hardware blocks and each block has a certain set of supported events.	✓	✓	✗
`mlxbf-ptm`	Kernel driver that provides a debufgs interface for the system software to monitor the BlueField device's power and thermal management parameters.	✗	✗	✓
`mlxbf-tmfifo`	TMFIFO driver for BlueField SoC	✓	✓	✓
`mlx-bootctl`	Boot control driver. This driver provides a `sysfs` interface for systems management software to manage reset time actions.	✓	✓	✓
`mlx-cpld`	Device driver for CPLD	✓	✗	✗
`mlx-trio`	TRIO driver for BlueField SoC	✓	✓	✗
`pwr-mlxbf`	Supports reset or low-power mode handling for BlueField.	✗	✓	✓

Driver	Description	BlueField	BlueField-2	BlueField-3
`pinctrl-mlxbf`	Allows multiplexing individual GPIOs to switch from the default hardware mode to software-controlled mode.	✗	✗	✓

4.6 Updating DPU Software Packages Using Standard Linux Tools

This dpu-upgrade procedure enables upgrading DOCA components using standard Linux tools (e.g., `apt update` and `yum update`). This process utilizes native package manager repositories to upgrade DPUs without the need for a full installation, and has the following benefits:

- Only updates components that include modifications
  - Configurable – user can select specific components (e.g., UEFI-ATF, NIC-FW)
- Includes upgrade of:
  - DOCA drivers and libraries
  - DOCA reference applications
  - BSP (UEFI/ATF) upgrade while maintaining the configuration
  - NIC firmware upgrade while maintaining the configuration
- Does not:
  - Impact user binaries
  - Upgrade non-Ubuntu OS kernels
  - Upgrade DPU BMC firmware
- After completion of DPU upgrade:
  - If NIC firmware was not updated, perform DPU Arm reset (software reset / reboot DPU)
  - If NIC firmware was updated, perform firmware reset (`mlxfwreset`) or perform a graceful shutdown and power cycle

OS	Action	Instructions
Ubuntu/Debian	Remove `mlxbf-bootimages` package	`<dpu> $ apt remove --purge mlxbf-bootimages* -y`
	Install the the GPG key	`<dpu> $ apt update` `<dpu> $ apt install gnupg2`

OS	Action	Instructions
	Export the desired distribution	Export `DOCA_REPO` with the relevant URL. The following is an example for Ubuntu 22.04: `<dpu> $ export DOCA_REPO="https://linux.mellanox.com/public/repo/doca/2.5.2/ubuntu22.04/dpu-arm64"` • Ubuntu 22.04 - https://linux.mellanox.com/public/repo/doca/2.5.2/ubuntu22.04/dpu-arm64 • Ubuntu 20.04 - https://linux.mellanox.com/public/repo/doca/2.5.2/ubuntu20.04/dpu-arm64 • Debian 12 - https://linux.mellanox.com/public/repo/doca/2.5.2/debian12/dpu-arm64
	Add GPG key to APT trusted keyring	`<dpu> $ curl $DOCA_REPO/GPG-KEY-Mellanox.pub | gpg --dearmor > /etc/apt/trusted.gpg.d/GPG-KEY-Mellanox.pub`
	Add DOCA online repository	`<dpu> $ echo "deb [signed-by=/etc/apt/trusted.gpg.d/GPG-KEY-Mellanox.pub] $DOCA_REPO ./" > /etc/apt/sources.list.d/doca.list`
	Update index	`<dpu> $ apt update`

	Upgrade UEFI/ATF firmware	Run:
		`<dpu> $ apt install mlxbf-bootimages-signed mlxbf-scripts`
		Then initiate upgrade for UEFI/ATF firmware:
		`<dpu> $ bfrec`

	Upgrade BlueField DPU NIC firmware	Run:
		`<dpu> $ apt install mlnx-fw-updater-signed`
		⚠ This immediately starts NIC firmware upgrade.
		To prevent automatic upgrade, run:
		`<dpu> $ export RUN_FW_UPDATER=no`

	Upgrade system	`<dpu> $ apt upgrade`

OS	Action	Instructions
	Apply the new changes, NIC firmware, and UEFI/ATF	`<dpu> $ mlxfwreset -d /dev/mst/mt*_pciconf0 -y -l 3 --sync 1 r` ⚠ If `mlxfwreset` is not supported, graceful shutdown and host power cycle are required for the NIC firmware upgrade to take effect.

CentOS/ RHEL/ Anolis/ Rocky	Remove `mlxbf-bootimages`, librerswan, and openvswitch-ipsec packages	`<dpu> $ yum -y remove mlxbf-bootimages*`
		`<dpu> $ yum remove libreswan openvswitch-ipsec`
		`<dpu> $ yum makecache`

	Export the desired distribution	Export `DOCA_REPO` with the relevant URL. The following is an example for Rocky Linux 8.6: `<dpu> $ export DOCA_REPO="https://linux.mellanox.com/public/repo/doca/2.5.2/rhel8.6/dpu-arm64/"` • AnolisOS 8.6 - https://linux.mellanox.com/public/repo/doca/2.5.2/anolis8.6/dpu-arm64/ • OpenEuler 20.03 sp1 - https://linux.mellanox.com/public/repo/doca/2.5.2/openeuler20.03sp1/dpu-arm64/ • CentOS 7.6 with 4.19 kernel - https://linux.mellanox.com/public/repo/doca/2.5.2/rhel7.6-4.19/dpu-arm64/ • CentOS 7.6 with 5.10 kernel - https://linux.mellanox.com/public/repo/doca/2.5.2/rhel7.6-5.10/dpu-arm64/ • CentOS 7.6 with 5.4 kernel - https://linux.mellanox.com/public/repo/doca/2.5.2/rhel7.6/dpu-arm64/ • Rocky Linux 8.6 - https://linux.mellanox.com/public/repo/doca/2.5.2/rhel8.6/dpu-arm64/

	Add DOCA online repository	`echo "[doca]`
		`name=DOCA Online Repo`
		`baseurl=$DOCA_REPO`
		`enabled=1`
		`gpgcheck=0`
		`priority=10`
		`cost=10" > /etc/yum.repos.d/doca.repo`
		A file is created under `/etc/yum.repos.d/doca.repo`.

	Update index	`<dpu> $ yum makecache`

	Upgrade UEFI/ATF firmware	Run:
		`<dpu> $ yum install mlxbf-bootimages-signed.aarch64 mlxbf-bfscripts`
		Then initiate the upgrade for UEFI/ATF firmware:
		`<dpu> $ bfrec`

<table>
<tr><th>OS</th><th>Action</th><th>Instructions</th></tr>
<tr><td></td><td>Upgrade BlueField DPU NIC firmware</td><td>The following command updates the firmware package and automatically attempts to flash the firmware to the NIC:

<pre><dpu> $ yum install mlnx-fw-updater-signed.aarch64</pre>
⚠ To prevent automatic flashing of the firmware to the NIC, run the following first:

<pre><dpu> $ export RUN_FW_UPDATER=no</pre>
⚠ This step can be used as a standalone firmware update. In any case, it is performed as part of the upgrade flow.

⚠ Flashing the firmware to the NIC can be performed manually by running the following command, after the firmware package had been updated:

<pre>sudo /opt/mellanox/mlnx-fw-updater/
mlnx_fw_updater.pl --force-fw-update</pre></td></tr>
<tr><td></td><td>Upgrade system</td><td><pre><dpu> $ yum upgrade --nobest</pre></td></tr>
<tr><td></td><td>Apply the new changes, NIC firmware, and UEFI/ATF</td><td><pre><dpu> $ mlxfwreset -d /dev/mst/mt*_pciconf0 -y -l 3 --sync 1 r</pre>
⚠ If <code>mlxfwreset</code> is not supported, a graceful shutdown and host power cycle are required for the NIC firmware upgrade to take effect.</td></tr>
</table>

5 Initial Configuration

The following pages provide instructions regarding general configuration of the BlueField DPU.

5.1 Modes of Operation

The NVIDIA® BlueField® DPU has several modes of operation:

- DPU mode, or embedded function (ECPF) ownership, where the embedded Arm system controls the NIC resources and data path (default)
- Zero-trust mode which is an extension of the ECPF ownership with additional restrictions on the host side
- NIC mode where the DPU behaves exactly like an adapter card from the perspective of the external host

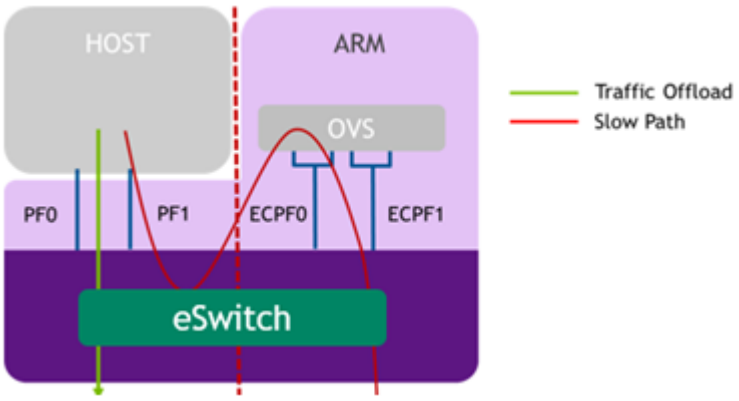
5.1.1 DPU Mode

This mode, known also as embedded CPU function ownership (ECPF) mode, is the default mode for BlueField DPU.

In DPU mode, the NIC resources and functionality are owned and controlled by the embedded Arm subsystem. All network communication to the host flows through a virtual switch control plane hosted on the Arm cores, and only then proceeds to the host. While working in this mode, the DPU is the trusted function managed by the data center and host administrator—to load network drivers, reset an interface, bring an interface up and down, update the firmware, and change the mode of operation on the DPU device.

A network function is still exposed to the host, but it has limited privileges. In particular:

1. The driver on the host side can only be loaded after the driver on the DPU has loaded and completed NIC configuration.
2. All ICM (Interface Configuration Memory) is allocated by the ECPF and resides in the DPU's memory.
3. The ECPF controls and configures the NIC embedded switch which means that traffic to and from the host (DPU) interface always lands on the Arm side.



When the server and DPU are initiated, the networking to the host is blocked until the virtual switch on the DPU is loaded. Once it is loaded, traffic to the host is allowed by default.

There are two ways to pass traffic to the host interface: Either using representors to forward traffic to the host (every packet to/from the host would be handled also by the network interface on the embedded Arm side) or push rules to the embedded switch which allows and offloads this traffic.

In DPU mode, OpenSM must be run from the DPU side (not the host side). Also, management tools (e.g., sminfo, ibdev2netdev, ibnetdiscover) can only be run from the DPU side (not from the host side).

5.1.2 Zero-trust Mode

Zero-trust mode is a specialization of DPU mode which implements an additional layer of security where the host system administrator is prevented from accessing the DPU from the host. Once zero-trust mode is enabled, the data center administrator should control the DPU entirely through the Arm cores and/or BMC connection instead of through the host.

For security and isolation purposes, it is possible to restrict the host from performing operations that can compromise the DPU. The following operations can be restricted individually when changing the DPU host to zero-trust mode:

- Port ownership – the host cannot assign itself as port owner
- Hardware counters – the host does not have access to hardware counters
- Tracer functionality is blocked
- RShim interface is blocked
- Firmware flash is restricted

5.1.2.1 Enabling Zero-trust Mode

To enable host restriction:

1. Start the MST service.
2. Set zero-trust mode. From the Arm side, run:

```
$ sudo mlxprivhost -d /dev/mst/<device> r --disable_rshim --disable_tracer --disable_counter_rd --
disable_port_owner
```

> ⚠ Graceful shutdown and power cycle are required if any `--disable_*` flags are used.

5.1.2.2 Disabling Zero-trust Mode

To disable host restriction, set the mode to privileged. Run:

```
$ sudo mlxprivhost -d /dev/mst/<device> p
```

The configuration takes effect immediately.

> ⚠ Graceful shutdown and power cycle are required when reverting to privileged mode if host restriction has been applied using any `--disable_*` flags.

5.1.3 NIC Mode

In this mode, the DPU behaves exactly like an adapter card from the perspective of the external host.

> ⚠ The following instructions presume the DPU to operate in DPU mode. If the DPU is operating in zero-trust mode, please return to DPU mode before continuing.

5.1.3.1 NIC Mode for BlueField-3

> ⚠ When BlueField-3 is configured to operate in NIC mode, Arm OS will not boot.

NIC mode for BlueField-3 saves power, improves device performance, and improves the host memory footprint.

5.1.3.1.1 Configuring NIC Mode on BlueField-3 from Linux

5.1.3.1.1.1 Enabling NIC Mode from Linux

Before moving to NIC mode, make sure you are operating in DPU mode by running:

```
host/dpu> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 -e q
```

The output should have `INTERNAL_CPU_MODEL= EMBBEDDED_CPU(1)` and `EXP_ROM_UEFI_ARM_ENABLE = True (1)` (default).

To enable NIC mode from DPU mode:

1. Run the following on the host or Arm:

```
host/dpu> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 s INTERNAL_CPU_MODEL=1 INTERNAL_CPU_OFFLOAD_ENGINE=1
```

2. Perform a graceful shutdown and power cycle the host.

5.1.3.1.1.2 Disabling NIC Mode from Linux

To return to DPU mode from NIC mode:

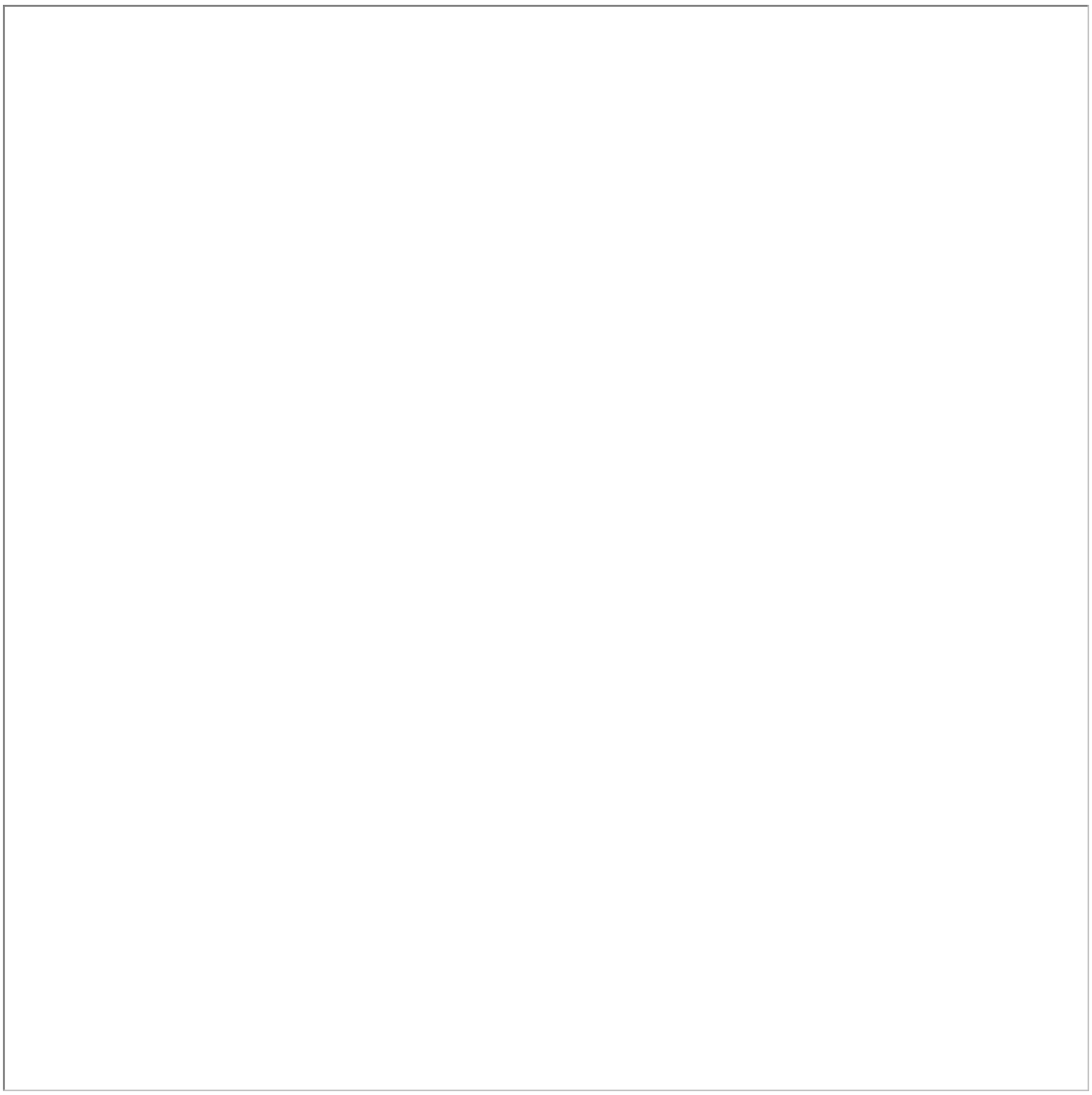
1. Run the following on the host:

```
host> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 s INTERNAL_CPU_MODEL=1 INTERNAL_CPU_OFFLOAD_ENGINE=0
```

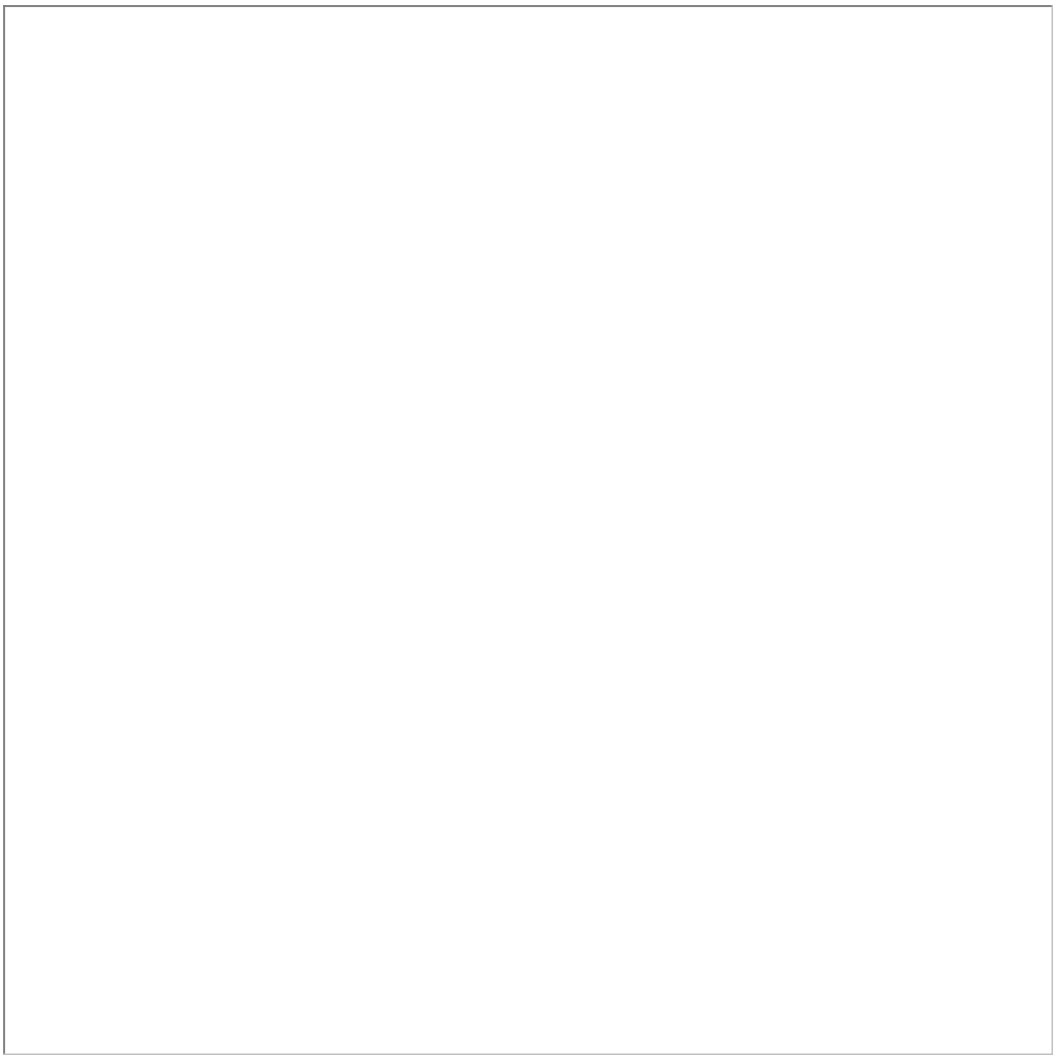
2. Perform a graceful shutdown and power cycle the host.

5.1.3.1.2 Configuring NIC Mode on BlueField-3 from UEFI

1. Access the Arm UEFI menu by pressing the Esc button twice
2. Select "Device Manager".
3. Select "Network Device List".
4. Select the network device that presents the uplink (i.e., select the device with the uplink MAC address).
5. Select "NVIDIA Network adapter - $<uplink-mac>".
6. Select "BlueField Internal Cpu Configuration".



- To enable NIC mode, set "Internal Cpu Offload Engine" to "Disabled".
- To switch back to DPU mode, set "Internal Cpu Offload Engine" to "Enabled".



5.1.3.1.3 Updating ATF and UEFI in BlueField-3 NIC Mode

Once in NIC mode, updating ATF and UFEI can be done using `preboot-install.bfb` by running:

```
# bfb-install --bfb <BlueField-BSP>.bfb --rshim rshim0
```

5.1.3.2 NIC Mode for BlueField-2

In this mode, the ECPFs on the Arm side are not functional but the user is still able to access the Arm system and update `mlxconfig` options.

> ⚠ When NIC mode is enabled, the drivers and services on the Arm are no longer functional.

5.1.3.2.1 Enabling NIC Mode on BlueField-2

To enable NIC mode from DPU mode:

1. Run the following from the x86 host side:

```
$ mst start
$ mlxconfig -d /dev/mst/<device> s INTERNAL_CPU_MODEL=1 \
INTERNAL_CPU_PAGE_SUPPLIER=1 \
INTERNAL_CPU_ESWITCH_MANAGER=1 \
INTERNAL_CPU_IB_VPORT0=1 \
INTERNAL_CPU_OFFLOAD_ENGINE=1
```

> ⚠ To restrict RShim PF (optional), make sure to configure `INTERNAL_CPU_RSHIM=1` as part of the `mlxconfig` command.

2. Perform a graceful shutdown and power cycle the host.

> ⚠ Multi-host is not supported when the DPU is operating in NIC mode.

> ⚠ To obtain firmware BINs for BlueField-2 devices, please refer to the BlueField-2 firmware download page.

5.1.3.2.2 Disabling NIC Mode on BlueField-2

To change from NIC mode back to DPU mode:

1. Install and start the RShim driver on the host.
2. Disable NIC mode. Run:

```
$ mst start
$ mlxconfig -d /dev/mst/<device> s INTERNAL_CPU_MODEL=1 \
INTERNAL_CPU_PAGE_SUPPLIER=0 \
INTERNAL_CPU_ESWITCH_MANAGER=0 \
INTERNAL_CPU_IB_VPORT0=0 \
INTERNAL_CPU_OFFLOAD_ENGINE=0
```

> ⚠ If `INTERNAL_CPU_RSHIM=1`, then make sure to configure `INTERNAL_CPU_RSHIM=0` as part of the `mlxconfig` command.

3. Perform a graceful shutdown and power cycle the host.

5.2 System Configuration and Services

This page provides information on system services and scripts based on the default DPU OS (i.e., Ubuntu).

5.2.1 First Boot After BFB Installation

During the first boot, the cloud-init service configures the system based on the data provided in the following files:

- `/var/lib/cloud/seed/nocloud-net/network-config` - network interface configuration

- `/var/lib/cloud/seed/nocloud-net/user-data` - default users and commands to run on the first boot

5.2.2 RDMA and ConnectX Driver Initialization

RDMA and NVIDIA® ConnectX® drivers are loaded upon boot by the `openibd.service`.

> ⚠ The `mlx5_core` kernel module is loaded automatically by the kernel as a registered device driver.

One of the kernel modules loaded by the `openibd.service`, `ib_umad`, triggers modprobe rule from `/etc/modprobe.d/mlnx-bf.conf` file that runs the `/sbin/mlnx_bf_configure` script. See Default Ports and OVS Configuration for more information.

5.2.3 Firewall Configuration

The Ubuntu BFB image includes the following firewall configuration by default (enabled):

```
$ cat /etc/iptables/rules.v4

*mangle
:PREROUTING ACCEPT [45:3582]
:INPUT ACCEPT [45:3582]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [36:4600]
:POSTROUTING ACCEPT [36:4600]
:KUBE-IPTABLES-HINT - [0:0]
:KUBE-KUBELET-CANARY - [0:0]
COMMIT
*filter
:INPUT ACCEPT [41:3374]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [32:3672]
:DOCKER-USER - [0:0]
:KUBE-FIREWALL - [0:0]
:KUBE-KUBELET-CANARY - [0:0]
:LOGGING - [0:0]
:POSTROUTING - [0:0]
:PREROUTING - [0:0]
-A INPUT -j KUBE-FIREWALL
-A INPUT -p tcp -m tcp --dport 111 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -p udp -m udp --dport 111 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -i lo -m comment --comment MD_IPTABLES -j ACCEPT
-A INPUT -d 127.0.0.0/8 -m mark --mark 0xb -m comment --comment MD_IPTABLES -j DROP
-A INPUT -m mark --mark 0xb -m state --state RELATED,ESTABLISHED -m comment --comment MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp ! --dport 22 ! --tcp-flags FIN,SYN,RST,ACK SYN -m mark --mark 0xb -m state --state NEW -m
comment --comment MD_IPTABLES -j DROP
-A INPUT -f -m mark --mark 0xb -m comment --comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,SYN,RST,PSH,ACK,URG -m mark --mark 0xb -m comment --
comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -m mark --mark 0xb -m comment --comment MD_IPTABLES
-j DROP
-A INPUT -m mark --mark 0xb -m state --state INVALID -m comment --comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m tcp --tcp-flags RST RST -m mark --mark 0xb -m hashlimit --hashlimit-above 2/sec --hashlimit-
burst 2 --hashlimit-mode srcip --hashlimit-name hashlimit_0 --hashlimit-htable-expire 30000 -m comment --comment
MD_IPTABLES -j DROP
-A INPUT -p tcp -m mark --mark 0xb -m state --state NEW -m hashlimit --hashlimit-above 50/sec --hashlimit-burst 50
--hashlimit-mode srcip --hashlimit-name hashlimit_1 --hashlimit-htable-expire 30000
-m comment --comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m mark --mark 0xb -m conntrack --ctstate NEW -m hashlimit --hashlimit-above 60/sec --hashlimit-
burst 20 --hashlimit-mode srcip --hashlimit-name hashlimit_2 --hashlimit-htable-expire 30000 -m comment --comment
MD_IPTABLES -j DROP
-A INPUT -m mark --mark 0xb -m recent --rcheck --seconds 86400 --name portscan --mask 255.255.255.255 --rsource -m
comment --comment MD_IPTABLES -j DROP
-A INPUT -m mark --mark 0xb -m recent --remove --name portscan --mask 255.255.255.255 --rsource -m comment --
comment MD_IPTABLES
-A INPUT -p tcp -m tcp --dport 22 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --set --name DEFAULT --
mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES
-A INPUT -p tcp -m tcp --dport 22 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --update --seconds 60 --
hitcount 50 --name DEFAULT --mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES -j DROP

-A INPUT -p tcp -m tcp --dport 443 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --set --name DEFAULT --
mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES
-A INPUT -p tcp -m tcp --dport 443 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --update --seconds 60 --
hitcount 10 --name DEFAULT --mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES -j DROP
-A INPUT -p udp -m udp --dport 161 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --set --name DEFAULT --
mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES
-A INPUT -p udp -m udp --dport 161 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --update --seconds 60 --
hitcount 100 --name DEFAULT --mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES -j DROP
```

```
-A INPUT -p tcp -m tcp --dport 22 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 443 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 179 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 68 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 122 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 161 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 6306 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 69 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 389 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 389 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 1812:1813 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --
comment MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 49 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 49 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --sport 53 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --sport 53 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 500 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 4500 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 1293 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 1293 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 1707 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 1707 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -i lo -p udp -m udp --dport 3786 -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment MD_IPTABLES
-j ACCEPT
-A INPUT -i lo -p udp -m udp --dport 33000 -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment MD_IPTABLES
-j ACCEPT
-A INPUT -p icmp -m mark --mark 0xb -m comment --comment MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --sport 5353 --dport 5353 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m
comment --comment MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 33434:33523 -m mark --mark 0xb -m comment --comment MD_IPTABLES -j REJECT --reject-
with icmp-port-unreachable
-A INPUT -p udp -m udp --dport 123 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 514 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 67 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 60102 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
"MD_IPTABLES: Feature HA port" -j ACCEPT
-A INPUT -m mark --mark 0xb -m comment --comment MD_IPTABLES -j LOGGING
-A FORWARD -j DOCKER-USER
-A OUTPUT -o oob_net0 -m comment --comment MD_IPTABLES -j ACCEPT
-A DOCKER-USER -j RETURN

-A LOGGING -m mark --mark 0xb -m comment --comment MD_IPTABLES -j NFLOG --nflog-prefix  "IPTables-Dropped: " --
nflog-group 3
-A LOGGING -m mark --mark 0xb -m comment --comment MD_IPTABLES -j DROP
-A PREROUTING -i oob_net0 -m comment --comment MD_IPTABLES -j MARK --set-xmark 0xb/0xffffffff
-A PREROUTING -p tcp -m tcpmss ! --mss 536:65535 -m tcp ! --dport 22 -m mark --mark 0xb -m conntrack --ctstate NEW
-m comment --comment MD_IPTABLES -j DROP
COMMIT
*nat
:PREROUTING ACCEPT [1:320]
:INPUT ACCEPT [1:320]
:OUTPUT ACCEPT [8:556]
:POSTROUTING ACCEPT [8:556]
:KUBE-KUBELET-CANARY - [0:0]
:KUBE-MARK-DROP - [0:0]
:KUBE-MARK-MASQ - [0:0]
:KUBE-POSTROUTING - [0:0]
-A POSTROUTING -m comment --comment "kubernetes postrouting rules" -j KUBE-POSTROUTING
-A KUBE-MARK-DROP -j MARK --set-xmark 0x8000/0x8000
-A KUBE-MARK-MASQ -j MARK --set-xmark 0x4000/0x4000
-A KUBE-POSTROUTING -m mark ! --mark 0x4000/0x4000 -j RETURN
-A KUBE-POSTROUTING -j MARK --set-xmark 0x4000/0x0
-A KUBE-POSTROUTING -m comment --comment "kubernetes service traffic requiring SNAT" -j MASQUERADE --random-fully
COMMIT
```

This configuration is provided by the `bf-release` package and is installed during the first boot of the Ubuntu OS after the BFB installation using the `cloud-init` service and the `/var/lib/cloud/seed/nocloud-net/user-data` configuration file.

To disable this default firewall configuration after OS is UP, run:

```
$ rm -f /etc/iptables/rules.v4
```

```
$ iptables -F
```

To disable this default firewall configuration during the BFB installation, use `bf.cfg` with the following command in the `bfb_modify_os` function:

```
bfb_modify_os()
{
    perl -ni -e "if(/^write_files:/../^users/) {next unless m{^users}; print} else {print}" /mnt/var/lib/cloud/
seed/nocloud-net/user-data
}
```

5.3 Host-side Interface Configuration

The NVIDIA® BlueField® DPU registers on the host OS a "DMA controller" for DPU management over PCIe. This can be verified by running the following:

```
#  lspci -d 15b3: | grep 'SoC Management Interface'
27:00.2 DMA controller: Mellanox Technologies MT42822 BlueField-2 SoC Management Interface (rev 01)
```

A special driver called RShim must be installed and run to expose the various BlueField management interfaces on the host OS. Refer to section "Install RShim on Host" for information on how to obtain and install the host-side RShim driver.

When the RShim driver runs properly on the host side, a sysfs device, `/dev/rshim0/*`, and a virtual Ethernet interface, `tmfifo_net0`, become available. The following is an example for querying the status of the RShim driver on the host side:

```
# systemctl status rshim
● rshim.service - rshim driver for BlueField SoC
     Loaded: loaded (/lib/systemd/system/rshim.service; disabled; vendor preset: enabled)
     Active: active (running) since Tue 2022-05-31 14:57:07 IDT; 1 day 1h ago
       Docs: man:rshim(8)
    Process: 90322 ExecStart=/usr/sbin/rshim $OPTIONS (code=exited, status=0/SUCCESS)
   Main PID: 90323 (rshim)
      Tasks: 11 (limit: 76853)
     Memory: 3.3M
     CGroup: /system.slice/rshim.service
             90323 /usr/sbin/rshim
May 31 14:57:07 …  systemd[1]: Starting rshim driver for BlueField SoC...
May 31 14:57:07  … systemd[1]: Started rshim driver for BlueField SoC.
May 31 14:57:07  … rshim[90323]: Probing pcie-0000:a3:00.2(vfio)
May 31 14:57:07  … rshim[90323]: Create rshim pcie-0000:a3:00.2
May 31 14:57:07  … rshim[90323]: rshim pcie-0000:a3:00.2 enable
May 31 14:57:08  … rshim[90323]: rshim0 attached
```

If the RShim device does not appear, refer to section "RShim Troubleshooting and How-Tos".

5.3.1 Virtual Ethernet Interface

On the host, the RShim driver exposes a virtual Ethernet device called `tmfifo_net0`. This virtual Ethernet can be thought of as a peer-to-peer tunnel connection between the host and the DPU OS. The DPU OS also configures a similar device. The DPU OS's BFB images are customized to configure the DPU side of this connection with a preset IP of 192.168.100.2/30. It is up to the user to configure the host side of this connection. Configuration procedures vary for different OSs.

The following example configures the host side of `tmfifo_net0` with a static IP and enables IPv4-based communication to the DPU OS:

```
#  ip addr add dev tmfifo_net0 192.168.100.1/30
```

> ⚠ For instructions on persistent IP configuration of the tmfifo_net0 interface, refer to step "Assign a static IP to tmfifo_net0" under "Updating Repo Package on Host Side".

Logging in from the host to the DPU OS is now possible over the virtual Ethernet. For example:

```
ssh ubuntu@192.168.100.2
```

5.3.2 RShim Support for Multiple DPUs

Multiple DPUs may connect to the same host machine. When the RShim driver is loaded and operating correctly, each board is expected to have its own device directory on sysfs, `/dev/rshim<N>`, and a virtual Ethernet device, `tmfifo_net<N>`.

The following are some guidelines on how to set up the RShim virtual Ethernet interfaces properly if multiple DPUs are installed in the host system.

There are two methods to manage multiple `tmfifo_net` interfaces on a Linux platform:

- Using a bridge, with all `tmfifo_net<N>` interfaces on the bridge - the bridge device bares a single IP address on the host while each DPU has unique IP in the same subnet as the bridge
- Directly over the individual `tmfifo_net<N>` - each interface has a unique subnet IP and each DPU has a corresponding IP per subnet

Whichever method is selected, the host-side `tmfifo_net` interfaces should have different MAC addresses, which can be:

- Configured using `ifconfig`. For example:

  ```
  $ ifconfig tmfifo_net0 192.168.100.1/24 hw ether 02:02:02:02:02:02
  ```

- Or saved in configuration via the `/udev/rules` as can be seen later in this section.

In addition, each Arm-side `tmfifo_net` interface must have a unique MAC and IP address configuration, as BlueField OS comes uniformly pre-configured with a generic MAC, and 192.168.100.2. The latter must be configured in each DPU manually or by DPU customization scripts during BlueField OS installation.

5.3.2.1 Multi-board Management Example

This example deals with two BlueField DPUs installed on the same server (the process is similar for more DPUs).

This example assumes that the RShim package has been installed on the host server.

5.3.2.1.1 Configuring Management Interface on Host

> ⚠ This example is relevant for CentOS/RHEL operating systems only.

1. Create a `bf_tmfifo` interface under `/etc/sysconfig/network-scripts`. Run:

```
vim /etc/sysconfig/network-scripts/ifcfg-br_tmfifo
```

2. Inside `ifcfg-br_tmfifo`, insert the following content:

```
DEVICE="br_tmfifo"
BOOTPROTO="static"
IPADDR="192.168.100.1"
NETMASK="255.255.255.0"
ONBOOT="yes"
TYPE="Bridge"
```

3. Create a configuration file for the first BlueField DPU, `tmfifo_net0`. Run:

```
vim /etc/sysconfig/network-scripts/ifcfg-tmfifo_net0
```

4. Inside `ifcfg-tmfifo_net0`, insert the following content:

```
DEVICE=tmfifo_net0
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br_tmfifo
```

5. Create a configuration file for the second BlueField DPU, `tmfifo_net1`. Run:

```
DEVICE=tmfifo_net1
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br_tmfifo
```

6. Create the rules for the `tmfifo_net` interfaces. Run:

```
vim /etc/udev/rules.d/91-tmfifo_net.rules
```

7. Restart the network for the changes to take effect. Run:

```
# /etc/init.d/network restart
Restarting network (via systemctl):            [  OK  ]
```

5.3.2.1.2 Configuring BlueField DPU Side

BlueField DPUs arrive with the following factory default configurations for tmfifo_net0.

Address	Value
MAC	`00:1a:ca:ff:ff:01`
IP	`192.168.100.2`

Therefore, if you are working with more than one DPU, you must change the default MAC and IP addresses.

5.3.2.1.2.1 Updating RShim Network MAC Address

This procedure is relevant for Ubuntu/Debian (`sudo` needed), and CentOS BFBs. The procedure only affects the `tmfifo_net0` on the Arm side.

1. Use a Linux console application (e.g. screen or minicom) to log into each BlueField. For example:

```
# sudo screen /dev/rshim<0|1>/console 115200
```

2. Create a configuration file for `tmfifo_net0` MAC address. Run:

```
# sudo vi /etc/bf.cfg
```

3. Inside `bf.cfg`, insert the new MAC:

```
NET_RSHIM_MAC=00:1a:ca:ff:ff:03
```

4. Apply the new MAC address. Run:

```
sudo bfcfg
```

5. Repeat this procedure for the second BlueField DPU (using a different MAC address).

> Arm must be rebooted for this configuration to take effect. It is recommended to update the IP address before you do that to avoid unnecessary reboots.

> For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

5.3.2.1.2.2 Updating IP Address

For Ubuntu:

1. Access the file `50-cloud-init.yaml` and modify the tmfifo_net0 IP address:

```
sudo vim /etc/netplan/50-cloud-init.yaml

               tmfifo_net0:
                    addresses:
                    - 192.168.100.2/30    ===>>>    192.168.100.3/30
```

2. Reboot the Arm. Run:

```
sudo reboot
```

3. Repeat this procedure for the second BlueField DPU (using a different IP address).

> Arm must be rebooted for this configuration to take effect. It is recommended to update the MAC address before you do that to avoid unnecessary reboots.

For CentOS:

1. Access the file `ifcfg-tmfifo_net0`. Run:

```
# vim /etc/sysconfig/network-scripts/ifcfg-tmfifo_net0
```

2. Modify the value for `IPADDR`:

```
IPADDR=192.168.100.3
```

3. Reboot the Arm. Run:

```
reboot
```

Or perform `netplan apply`.

4. Repeat this procedure for the second BlueField DPU (using a different IP address).

> Arm must be rebooted for this configuration to take effect. It is recommended to update the MAC address before you do that to avoid unnecessary reboots.

5.3.3 Permanently Changing Arm-side MAC Address

> It is assumed that the commands in this section are executed with root (or `sudo`) permission.

The default MAC address is `00:1a:ca:ff:ff:01`. It can be changed using `ifconfig` or by updating the UEFI variable as follows:

1. Log into Linux from the Arm console.
2. Run:

```
$ "ls /sys/firmware/efi/efivars".
```

3. If not mounted, run:

```
$ mount -t efivarfs none /sys/firmware/efi/efivars
$ chattr -i /sys/firmware/efi/efivars/RshimMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
$ printf "\x07\x00\x00\x00\x00\x1a\xca\xff\xff\x03" > \
  /sys/firmware/efi/efivars/RshimMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

The `printf` command sets the MAC address to `00:1a:ca:ff:ff:03` (the last six bytes of the `printf` value). Either reboot the device or reload the tmfifo driver for the change to take effect.

The MAC address can also be updated from the server host side while the Arm-side Linux is running:

1. Enable the configuration. Run:

```
# echo "DISPLAY_LEVEL 1" > /dev/rshim0/misc
```

2. Display the current setting. Run:

```
# cat /dev/rshim0/misc
DISPLAY_LEVEL   1 (0:basic, 1:advanced, 2:log)
BOOT_MODE       1 (0:rshim, 1:emmc, 2:emmc-boot-swap)
BOOT_TIMEOUT    300 (seconds)
DROP_MODE       0 (0:normal, 1:drop)
SW_RESET        0 (1: reset)
DEV_NAME        pcie-0000:04:00.2
DEV_INFO        BlueField-2(Rev 1)
PEER_MAC        00:1a:ca:ff:ff:01 (rw)
PXE_ID          0x00000000 (rw)
```

```
VLAN_ID          0 0 (rw)
```

3. Modify the MAC address. Run:

```
$ echo "PEER_MAC  xx:xx:xx:xx:xx:xx" > /dev/rshim0/misc
```

For more information and an example of the script that covers multiple DPU installation and configuration, refer to section "Installing Full DOCA Image on Multiple DPUs" of the *NVIDIA DOCA Installation Guide*.

5.3.4 OOB Ethernet Interface

The OOB interface is a gigabit Ethernet interface which provides TCP/IP network connectivity to the Arm cores. This interface is named `oob_net0` and is intended to be used for management traffic (e.g. file transfer protocols, SSH, etc). The Linux driver that controls this interface is named `mlxbf_gige.ko`, and is automatically loaded upon boot. This interface can be configured and monitored by use of standard tools (e.g. ifconfig, ethtool, etc). The OOB interface is subject to the following design limitations:

- Only supports 1Gb/s full-duplex setting
- Only supports GMII access to external PHY device
- Supports maximum packet size of 2KB (i.e. no support for jumbo frames)

The OOB interface can also be used for PXE boot. This OOB port is not a path for the boot stream. Any attempt to push a BFB to this port will not work. Please refer to How to use the UEFI boot menu for more information about UEFI operations related to the OOB interface.

5.3.4.1 OOB Interface MAC Address

The MAC address to be used for the OOB port is burned into Arm-accessible UPVS EEPROM during the manufacturing process. This EEPROM device is different from the SPI Flash storage device used for the NIC firmware and associated NIC MACs/GUIDs. The value of the OOB MAC address is specific to each platform and is visible on the board-level sticker.

It is not recommended to reconfigure the MAC address from the MAC configured during manufacturing.

If there is a need to re-configure this MAC for any reason, follow these steps to configure a UEFI variable to hold new value for OOB MAC.:

The creation of an OOB MAC address UEFI variable will override the OOB MAC address defined in EEPROM, but the change can be reverted.

1. Log into Linux from the Arm console.
2. Issue the command `ls /sys/firmware/efi/efivars` to show whether efivarfs is mounted. If it is not mounted, run:

```
mount -t efivarfs none /sys/firmware/efi/efivars
```

3. Run:

```
chattr -i /sys/firmware/efi/efivars/OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

4. Set the MAC address to 00:1a:ca:ff:ff:03 (the last six bytes of the printf value).

```
printf "\x07\x00\x00\x00\x00\x1a\xca\xff\xff\x03" > /sys/firmware/efi/efivars/
OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

5. Reboot the device for the change to take effect.

To revert this change and go back to using the MAC as programmed during manufacturing, follow these steps:

1. Log into UEFI from the Arm console, go to "Boot Manager" then "EFI Internal Shell".
2. Delete the OOB MAC UEFI variable. Run:

```
dmpstore -d OobMacAddr
```

3. Reboot the device by running "reset" from UEFI.
4. Log into Linux from the Arm console.
5. Issue the command `ls /sys/firmware/efi/efivars` to show whether efivarfs is mounted. If it is not mounted, run:

```
mount -t efivarfs none /sys/firmware/efi/efivars
```

6. Run:

```
chattr -i /sys/firmware/efi/efivars/OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

7. Reconfigure the original MAC address burned by the manufacturer in the format `aa\bb\cc\dd\ee\ff`. Run:

```
printf "\x07\x00\x00\x00\x00\<original-MAC-address>" > /sys/firmware/efi/efivars/
OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

8. Reboot the device for the change to take effect.

5.3.4.2 Supported ethtool Options for OOB Interface

The Linux driver for the OOB port supports the handling of some basic ethtool requests: get driver info, get/set ring parameters, get registers, and get statistics.

To use the ethtool options available, use the following format:

```
$ ethtool [<option>] <interface>
```

Where `<option>` may be:

- `<no-argument>` - display interface link information
- `-i` - display driver general information
- `-S` - display driver statistics
- `-d` - dump driver register set
- `-g` - display driver ring information
- `-G` - configure driver ring(s)

- `-k` - display driver offload information
- `-a` - query the specified Ethernet device for pause parameter information
- `-r` - restart auto-negotiation on the specified Ethernet device if auto-negotiation is enabled

For example:

```
$ ethtool oob_net0
Settings for oob_net0:
        Supported ports: [ TP ]
        Supported link modes:   1000baseT/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  1000baseT/Full
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Link partner advertised link modes:  1000baseT/Full
        Link partner advertised pause frame use: Symmetric
        Link partner advertised auto-negotiation: Yes
        Link partner advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 3
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: Unknown
        Link detected: yes
```

```
$ ethtool -i oob_net0
driver: mlxbf_gige
version:
firmware-version:
expansion-rom-version:
bus-info: MLNXBF17:00
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: yes
supports-priv-flags: no
```

```
# Display statistics specific to BlueField-2 design (i.e. statistics that are not shown in the output of "ifconfig
oob0_net")
$ ethtool -S oob_net0
NIC statistics:
    hw_access_errors: 0
    tx_invalid_checksums: 0
    tx_small_frames: 1
    tx_index_errors: 0
    sw_config_errors: 0
    sw_access_errors: 0
    rx_truncate_errors: 0
    rx_mac_errors: 0
    rx_din_dropped_pkts: 0
    tx_fifo_full: 0
    rx_filter_passed_pkts: 5549
    rx_filter_discard_pkts: 4
```

5.3.4.3 IP Address Configuration for OOB Interface

The files that control IP interface configuration are specific to the Linux distribution. The udev rules file (`/etc/udev/rules.d/92-oob_net.rules`) that renames the OOB interface to `oob_net0` and is the same for Yocto, CentOS, and Ubuntu:

```
SUBSYSTEM=="net", ACTION=="add", DEVPATH=="/devices/platform/MLNXBF17:00/net/eth[0-9]", NAME="oob_net0"
```

The files that control IP interface configuration are slightly different for CentOS and Ubuntu:

- CentOS configuration of IP interface:
    - Configuration file for `oob_net0` : `/etc/sysconfig/network-scripts/ifcfg-oob_net0`

- For example, use the following to enable DHCP:

```
NAME="oob_net0"
DEVICE="oob_net0"
NM_CONTROLLED="yes"
PEERDNS="yes"
ONBOOT="yes"
BOOTPROTO="dhcp"
TYPE=Ethernet
```

- For example, to configure static IP use the following:

```
NAME="oob_net0"
DEVICE="oob_net0"
IPV6INIT="no"
NM_CONTROLLED="no"
PEERDNS="yes"
ONBOOT="yes"
BOOTPROTO="static"
IPADDR="192.168.200.2"
PREFIX=30
GATEWAY="192.168.200.1"
DNS1="192.168.200.1"
TYPE=Ethernet
```

- For Ubuntu configuration of IP interface, refer to section "Default Network Interface Configuration".

5.4 Secure Boot

These pages provide guidelines on how to operate secured NVIDIA® BlueField® DPUs. They provide UEFI secure boot references for the UEFI portion of the secure boot process.

This section provides directions for illustration purposes, it does not intend to enforce or mandate any procedure about managing keys and/or production guidelines. Platform users are solely responsible of implementing secure strategies and safe approaches to manage their boot images and their associated keys and certificates.

Security aspects such as key generation, key management, key protection, and certificate generation are out of the scope of this section.

Secure boot is a process which verifies each element in the boot process prior to execution, and halts or enters a special state if a verification step fails at any point during the boot. It is based on an unmodifiable ROM code which acts as the root-of-trust (RoT) and uses an off-chip public key, to authenticate the initial code which is loaded from an external non-volatile storage. The off-chip public key integrity is verified by the ROM code against an on-chip public key hash value stored in E-FUSEs. Then the authenticated code and each element in the boot process cryptographically verify the next element prior to passing execution to it. This extends the chain-of-trust (CoT) by verifying elements that have their RoT in hardware. In addition, no external intervention in the authentication process is permitted to prevent unauthorized software and firmware from being loaded. There should be no way to interrupt or bypass the RoT with runtime changes.

5.4.1 Supported BlueField DPUs

Secured BlueField devices have pre-installed software and firmware signed with NVIDIA signing keys. The on-chip public key hash is programmed into E-FUSEs.

To verify whether the DPU in your possession supports secure boot, run the following command:

```
# sudo mst start
# sudo flint -d /dev/mst/mt41686_pciconf0 q full | grep "Life cycle"
Life cycle:            GA SECURED
```

“GA SECURED” indicates that the BlueField device has secure boot enabled.

To verify whether the BlueField Arm has secure boot enabled, run the following command from the BlueField console:

```
ubuntu@localhost:~$ sudo mlxbf-bootctl | grep lifecycle
lifecycle state: GA Secured
```

5.4.2 UEFI Secure Boot

This feature is available in the NVIDIA® BlueField®-2 and above.

UEFI Secure Boot is a feature of the Unified Extensible Firmware Interface (UEFI) specification. The feature defines a new interface between the operating system and firmware/BIOS.

When enabled and fully configured on the DPU, UEFI Secure Boot helps the Arm=based software running on top of UEFI resist attacks and infection from malware. UEFI Secure Boot detects tampering with boot loaders, key operating system files, and unauthorized option ROMs by validating their digital signatures. Malicious actions are blocked from running before they can attack or infect the system.

UEFI Secure Boot works as a security gate. Code signed with valid keys (whose public key/ certificates exist in the DPU) gets through the gate and executes while blocking and rejecting code that has either a bad or no signature.

The DPU enables UEFI secure boot with the Ubuntu OS included in the platform's software.

5.4.2.1 Verifying UEFI Secure Boot on DPU

To verify whether UEFI secure boot is enabled, run the following command from the BlueField console:

```
ubuntu@localhost:~$ sudo mokutil --sb-state
SecureBoot enabled
```

As UEFI secure boot is not specific to BlueField platforms, please refer to the Canonical documentation online for further information on UEFI secure boot:

- https://wiki.ubuntu.com/UEFI/SecureBoot
- https://wiki.ubuntu.com/UEFI/SecureBoot/Signing

5.4.2.2 Main Use Cases for UEFI Secure Boot

UEFI secure boot can be used in 2 main cases for the DPU:

Method	Pros	Cons
Using the default enabled UEFI secure boot (with Ubuntu OS or any Microsoft-signed boot loader) See "Using Default Enabled UEFI Secure Boot" for more.	Relatively easy	Limited flexibility; only allows executing NVIDIA binary files
		Dependency on Microsoft or NVIDIA as signing entities
Enabling UEFI Secure Boot with a custom OS (other than the default Ubuntu) See "Enabling UEFI Secure Boot with Custom OS" for more.	Autonomy, as you control your own keys (no dependency on Microsoft or NVIDIA as signing entities)	You must create your own capsule files to enroll and customize UEFI secure boot

Signing binaries is complex as you must create X.509 certificates and enroll them in UEFI or shim which requires a fair amount of prior knowledge of how secure boot works. For that reason, BlueField secured platforms are shipped with all the needed certificates and signed binaries (which allows working seamlessly with the first use case in the table above).

NVIDIA strongly recommends utilizing UEFI secure boot in any case due the increased security it enables.

5.4.2.2.1 Verifying UEFI Secure Boot on DPU

To verify whether UEFI secure boot is enabled, run the following command from the BlueField console:

```
ubuntu@localhost:~$ sudo mokutil --sb-state
SecureBoot enabled
```

As UEFI secure boot is not specific to BlueField platforms, refer to the Canonical documentation online for further information on UEFI secure boot to familiarize yourself with the UEFI secure boot concept:

- https://wiki.ubuntu.com/UEFI/SecureBoot
- https://wiki.ubuntu.com/UEFI/SecureBoot/Signing

5.4.2.3 Using Default Enabled UEFI Secure Boot

As part of the default settings of the DPU, UEFI secure boot is enabled and requires no special configuration from the user to use it with the bundled Ubuntu OS.

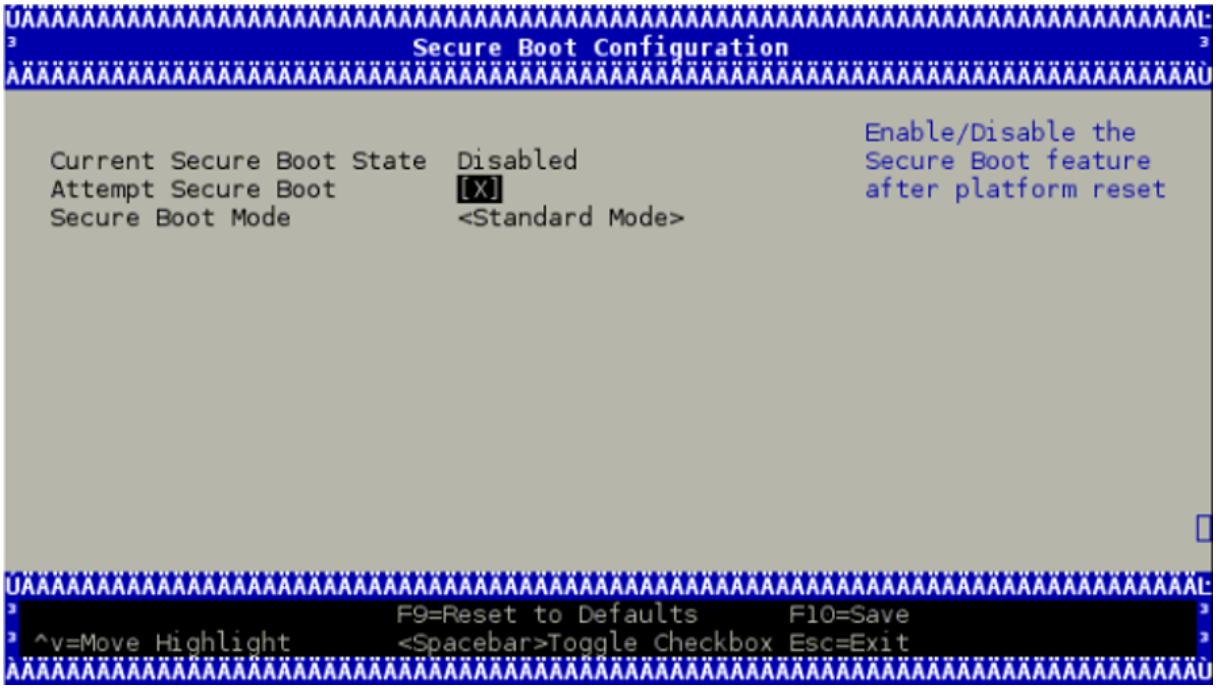
5.4.2.3.1 Disabling UEFI Secure Boot

UEFI secure boot can be disabled per device from the UEFI menu as part of the DPU boot process which requires access to the BlueField console.

To disable UEFI secure boot, reboot the platform and stop at the UEFI menu.

> ⚠ On BlueField devices with UEFI secure boot enabled, the UEFI menu is password-protected to prevent unwanted changes to the UEFI settings. The default password is `bluefield`.

From the UEFI menu screen, select "Device Manager" then "Secure Boot Configuration". If "Attempt Secure Boot" is checked, then uncheck it and reboot.



Disabling secure boot permanently is not recommended in production environments.

It is also possible to disable UEFI secure boot using Redfish API for DPUs with an on-board BMC. For more details, please refer to your NVIDIA sales representative to receive the *NVIDIA BlueField DPU Initial Deployment Guide*.

5.4.2.3.2 Existing DPU Certificates

As part of having UEFI secure boot enabled, the UEFI databases are populated with NVIDIA self-signed X.509 certificates. The Microsoft certificate is also installed into the UEFI database to ensure that the Ubuntu distribution can boot while UEFI secure boot is enabled (and generally any suitable OS loader signed by Microsoft).

The pre-installed certificate files are:

- NVIDIA PK key certificate
- NVIDIA KEK key certificate
- NVIDIA db certificate
- Microsoft db certificate

5.4.2.4 Enabling UEFI Secure Boot with Custom OS

This section lists the required steps to enable using UEFI secure boot with a custom OS (other than the default Ubuntu).

> ⚠ All processes described in the following subsections require some level of testing and knowledge in how operating system boot flows and bootloaders work.

5.4.2.4.1 Options for Enabling UEFI Secure Boot

There are 3 main ways for signing custom binaries and running them on the DPU with UEFI secure boot enabled:

#	Method	Pros	Cons
1	Sign OS loader (e.g., Shim) by Microsoft. See "Signing OS Loader by Microsoft" for more.	Does not require access to the BlueField console	Dependency on Microsoft as signing entity
2	Shim – enroll a machine owner key (MOK) certificate in the shim and use the private part to sign your files. See "Enrolling MOK Key" for more.	Easy	• Limited flexibility: Only allows executing a custom kernel or load a custom module. It does not allow executing UEFI applications, UEFI drivers, or OS loaders. • Dependency on Microsoft or NVIDIA as signing entities • Not scalable: Requires access to BlueField console per device (i.e., UART console required)
3	UEFI – enroll your own key certificate in the UEFI database and use the private part to sign your files. See "Enrolling Your Own Key to UEFI DB" for more.	Autonomy, as you control your keys (not dependent on Microsoft or NVIDIA as signing entities)	• Requires adding your key certificate to database manually • Requires access to BlueField console per device (i.e., UART console required) • Not scalable: Requires access to BlueField console per device (i.e., UART console required)

For generation of custom keys and certificates, see section "Generation of Custom Keys and Certificates".

Signing binaries for UEFI secure boot is complex as you must create X.509 certificates and enroll them in UEFI or shim which requires a fair amount of prior knowledge of how secure boot works. See the processes used to enroll keys and to sign UEFI binaries in the rest of this document.

Secure booting binaries for executing a UEFI application, UEFI driver, OS loader, custom kernel, or loading a custom module depends on the certificates and public keys available in the UEFI database and the shim's MOK list.

5.4.2.4.2 Signing OS Loader by Microsoft

5.4.2.4.2.1 Custom Kernel Images

One option to boot custom binaries on a DPU is to sign the OS loader (shim) by Microsoft following the Microsoft guidelines which are updated and maintained by Microsoft. The certificates/keys must be embedded within the shim OS loader so it may boot, in addition the custom Kernel binary image and the custom Kernel modules must be signed accordingly.

5.4.2.4.2.2 NVIDIA Kernel Modules

In this option, the NVIDIA db certificates should remain enrolled. This is due to the out-of-tree kernel modules and drivers (e.g., OFED) provided by NVIDIA which are signed by NVIDIA and authenticated by this NVIDIA certificate in the UEFI.

> ⚠ Signing binaries with Microsoft is a process the involves lead time which must be taken into consideration. This course of action requires testing to making sure the complied BFB image including the signed Microsoft bootloader works properly.

5.4.2.4.3 Enrolling MOK Key

To boot a custom kernel or load a custom module, you must create a MOK key pair. The newly created MOK key must be an RSA 2048-bit. The private part is used for signing operations and must be kept safe. The public X.509 key certificate in DER format must be enrolled within the shim MOK list.

Once the public key certificate is enrolled within the shim, the MOK key is accepted as a valid signing key.

Note that kernel module signing requires a special configuration. For example, the `extendedKeyUsage` field must show an OID of 1.3.6.1.4.1.2312.16.1.2. That OID informs shim that this is meant to be a module signing certificate.

The following is an example of OpenSSL configuration file for illustration purposes:

```
HOME                    = .
RANDFILE                = $ENV::HOME/.rnd
[ req ]
distinguished_name      = req_distinguished_name
x509_extensions         = v3
string_mask             = utf8only
prompt                  = no

[ req_distinguished_name ]
countryName             = US
stateOrProvinceName     = Westborough
localityName            = Massachusetts
0.organizationName      = CampanyX
commonName              = Secure Boot Signing
emailAddress            = example@example.com

[ v3 ]
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer
basicConstraints        = critical,CA:FALSE
extendedKeyUsage        = codeSigning,1.3.6.1.4.1.311.10.3.6,1.3.6.1.4.1.2312.16.1.2
nsComment               = "OpenSSL Generated Certificate"
```

To enroll the MOK key certificate, download the associated key certificate to the BlueField file system and run the following command:

```
ubuntu@localhost:~$ sudo mokutil --import mok.der
input password:
input password again:
```

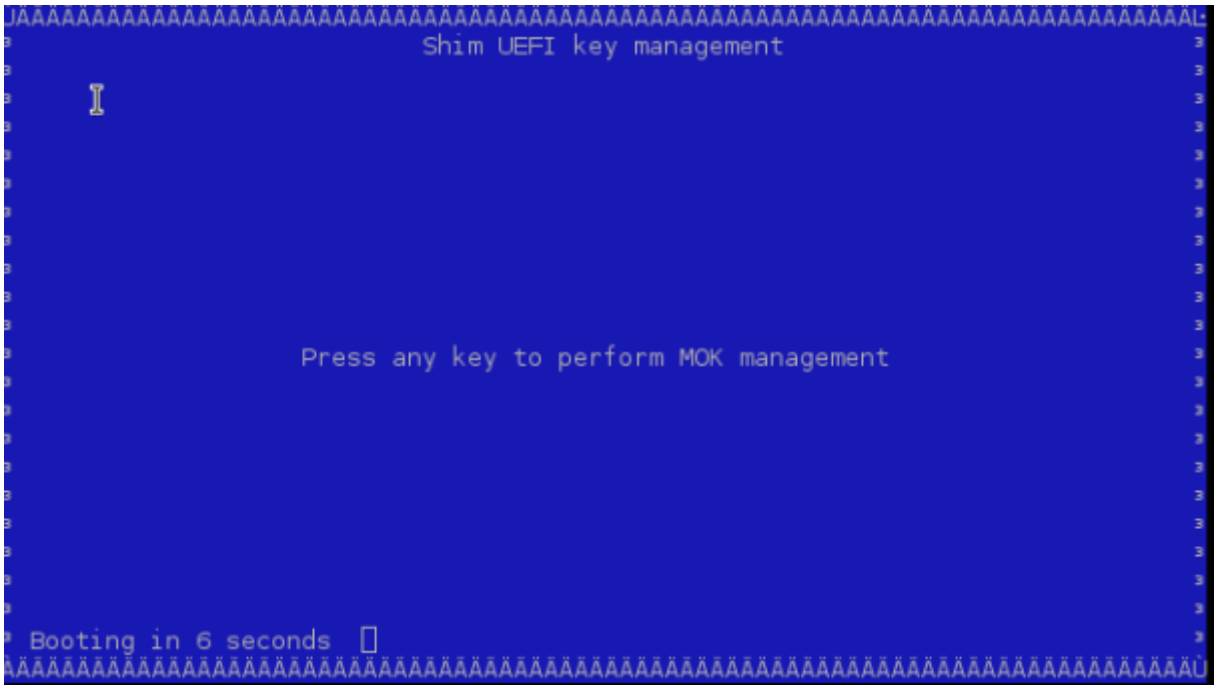
You must follow the prompts to enter a password to be used to make sure you really do want to enroll the key certificate.

Note that the key certificate is not enrolled yet. It will be enrolled by the shim upon the next reboot. To list the imported certificate file to enroll:

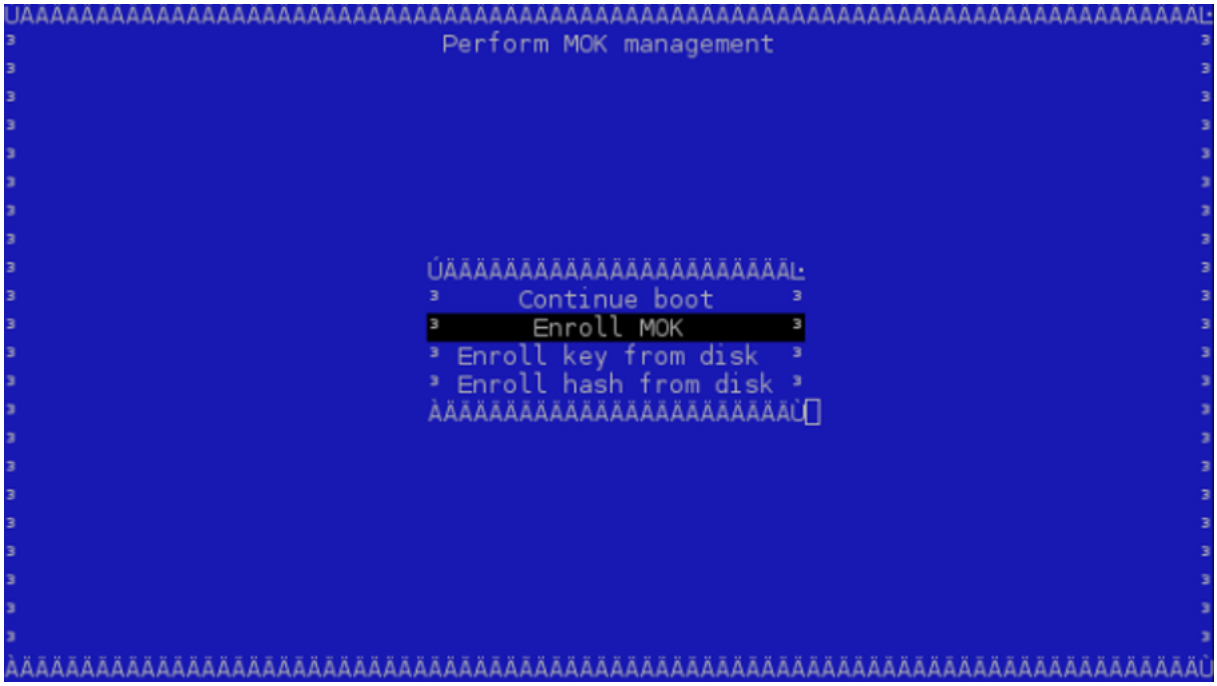
```
ubuntu@localhost:~$ sudo mokutil --list-new
```

A reboot must be performed.

Just before loading GRUB, shim displays a blue screen which is actually another piece of the shim project called "MokManager". You may ignore the blue screen showing the error message. Press "OK" to enter the "Shim UEFI key management" screen.



Select "Enroll MOK" and follow the menus to finish the enrolling process.



You may look at the properties of the key you are adding to make sure it is indeed correct using "View key". MokManager will ask for the same password you typed in earlier when running mokutil before reboot. MokManager will save the key and you will need to reboot again.

To list the enrolled certificate files, run the following command:

```
ubuntu@localhost:~$ sudo mokutil --list-enrolled
```

5.4.2.4.4 Generation of Custom Keys and Certificates

To boot binaries not signed with the existing public keys and certificates in the UEFI database (like the Microsoft certificate and key described in "Signing OS Loader by Microsoft"), create an X.509 certificate (which includes the public key part of the public-private key pair) that can be imported either directly though the UEFI or, more easily, via shim.

Creating a certificate and public key for use in the UEFI secure boot is relatively simple. OpenSSL can do it by running the command `req`.

For illustration purposes only, this example shows how to create a 2048-bit RSA MOK key and its associated certificate file in DER format:

```
$ openssl req -new -x509 -newkey rsa:2048 -nodes -days 36500 -outform DER -keyout "mok.priv" -out "mok.der"
```

An OpenSSL configuration file may be used for key generation. It may be specified using `--config path/to/openssl.cnf`.

⚠ Detailed key and certificate generation are beyond the scope of this document. Any organization should choose the proper way to generate keys and certificates based on their security policy.

The following sections refer to the db private key as `key.priv` and its DER certificate as `cert.der`. Similarly, the MOK private key is referred to as `mok.priv` and its DER certificate as `mok.der`.

5.4.2.4.5 Enrolling Your Own Key to UEFI DB

Some users may need to generate their own keys. For convenience, the processes used to enroll keys into UEFI db as well as to sign UEFI binaries are provided in this document.

To execute your binaries while UEFI secure boot is enabled, you need your own pair of private and public key certificates. The supported keys are RSA 2048-bit and ECDSA 384-bit.

The private part is used for signing operations and must be kept safe. The public part X.509 key certificate in DER format must be enrolled within the UEFI db.

A prerequisite for the following steps is having UEFI secure boot temporarily disabled on the DPU. After temporarily disabling UEFI secure boot per device as in section "Existing DPU Certificates", it is possible to override all the key certificate files of the UEFI database. This allows you to enroll your PK key certificate, KEK key certificate, and db certificates.

The following subsections detail how enrolling can be done.

5.4.2.4.5.1 Using a Capsule

To enroll your key certificates, create a capsule file by way of tools and scripts provided along with the BlueField software.

To create the capsule files, execute the `mlx-mkcap` script. After BlueField software installation, the script can be found under `/lib/firmware/mellanox/boot/capsule/scripts`. This script generates a capsule file to supply the key certificates to UEFI and enables UEFI secure boot:

```
$ ./mlx-mkcap --pk-key pk.cer --kek-key kek.cer --db-key db.cer EnrollYourKeysCap
```

Note that you may specify as many db certificates as needed using the `--db-key` flag. In this example, only a single db certificate is specified.

To set the UEFI password, you may specify the `--uefi-passwd` flag. For example, to set the UEFI password to `bluefield`, run:

```
$ ./mlx-mkcap --pk-key pk.cer --kek-key kek.cer --db-key db.cer --uefi-passwd "bluefield" EnrollYourKeysCap
```

The resulting capsule file, `EnrollYourKeysCap`, can be downloaded to the BlueField file system to initiate the key enrollment process. From the the BlueField console execute the following command then reboot:

```
ubuntu@localhost:~$ bfrec --capsule EnrollYourKeysCap
```

On the next reboot, the capsule file is processed and the UEFI database is populated with the keys extracted from the capsule file.

> ⚠ Enrolling the PK key certificate file enables the UEFI secure boot.

5.4.2.4.5.2 Enroll Certificate into UEFI DB

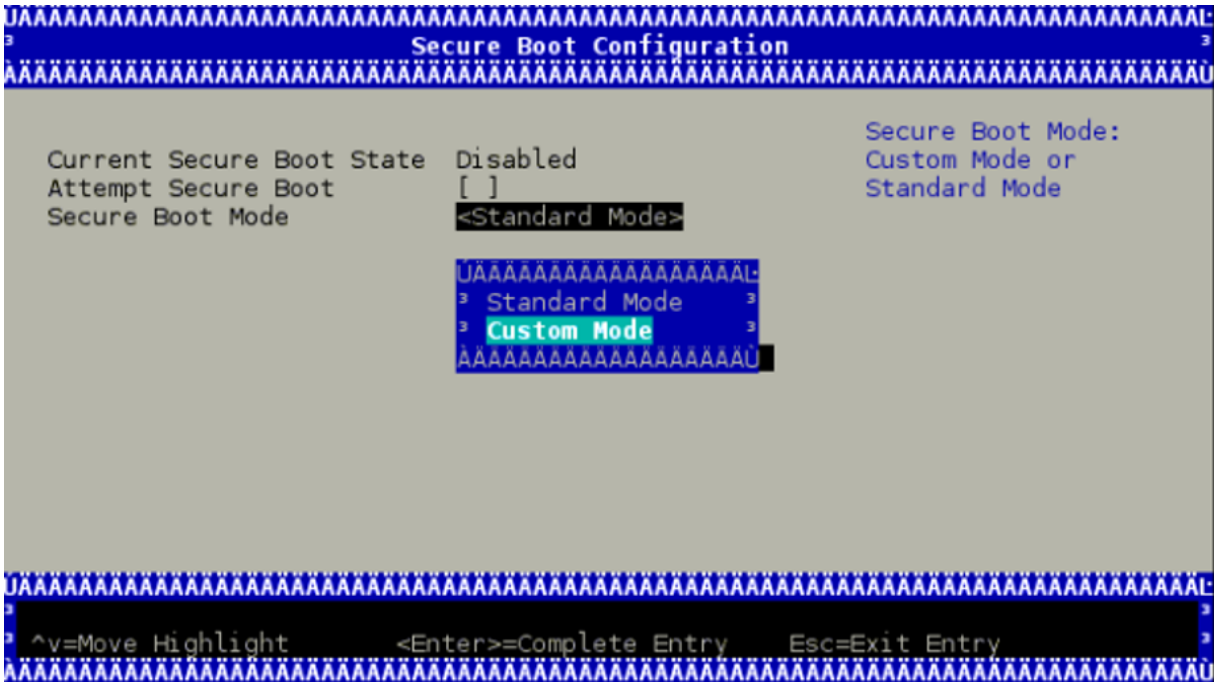
As mentioned, the public part of the X.509 key certificate in DER format must be enrolled within the UEFI db. The X.509 DER certificate file must be installed into the EFI system partition (ESP).

Download the certificate file to BlueField file system and place it into the ESP:

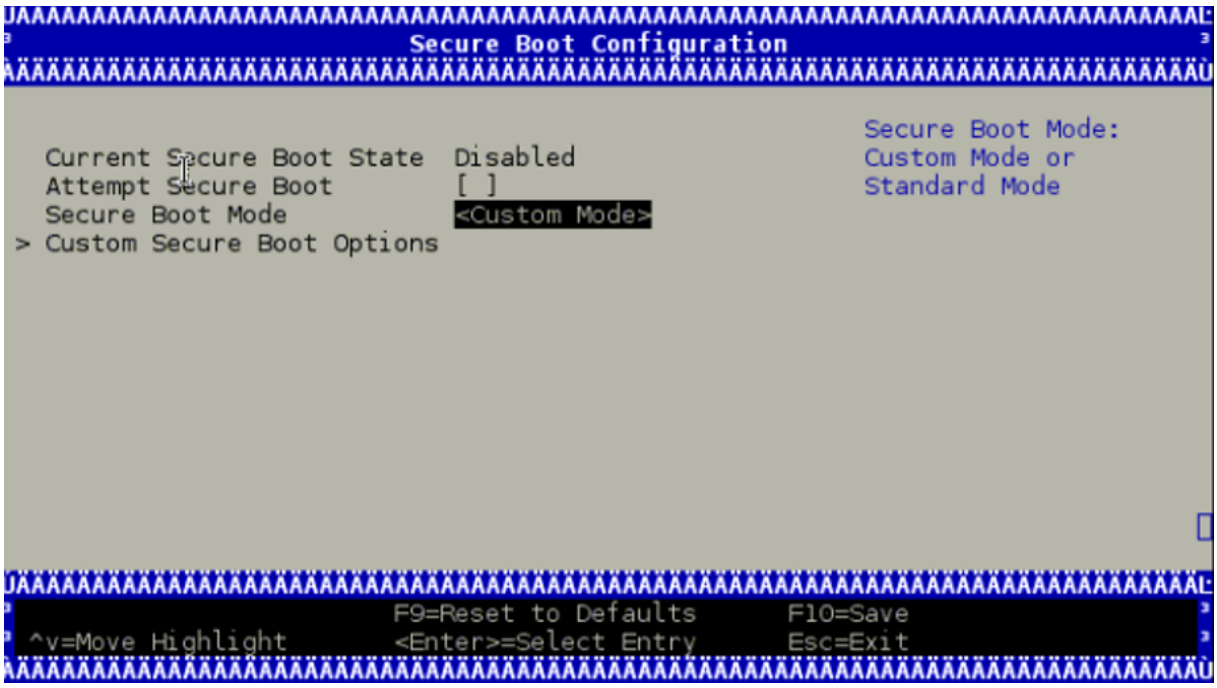
```
ubuntu@localhost:~$ sudo cp path/to/cert.der /boot/efi/
```

To enroll the certificate into the UEFI db, you must to reboot and log in again into the UEFI menu. From the "UEFI menu", select "Device Manager" entry then "Secure Boot Configuration". Navigate to "Secure Boot Mode" and select "Custom Mode" setup.

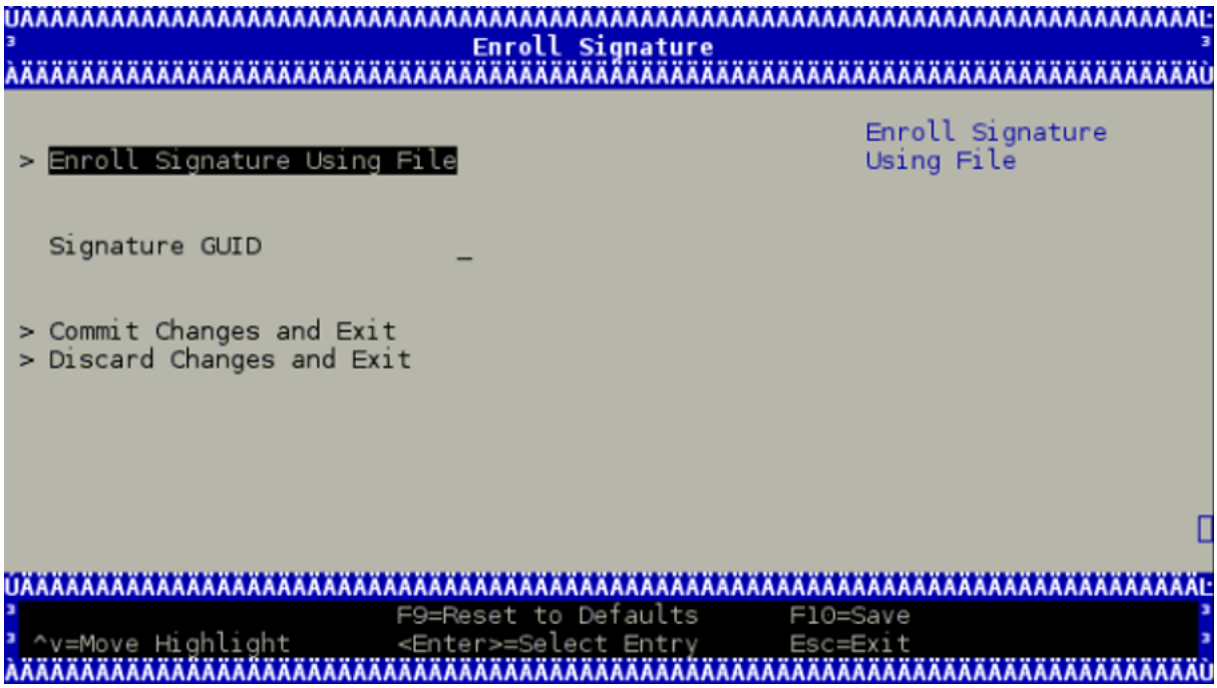
The secure boot "Custom Mode" setup feature allows a physically present user to modify the UEFI database.



Once the platform is in "Custom Mode", a "Custom Secure Boot Options" menu entry appears which allows you to manipulate the UEFI database keys and certificates.



To enroll your DER certificate file, select "DB Options" and enter the "Enroll Signature" menu. Select "Enroll Signature Using File" and navigate within the EFI System Partition (ESP) to the db DER certificate file.



The ESP path is shown below as "system-boot, [VenHw(*)/HD(*)]".

While enrolling the certificate file, you may enter a GUID along with the key certificate file. The GUID is the platform's way of identifying the key. It serves no purpose other than for you to tell which key is which when you delete it (it is not used at all in signature verification).

This value must be in the following format: 11111111-2222-3333-4444-1234567890ab. If nothing is entered, a GUID of 00000000-0000-0000-0000-000000000000 is created.

Finally, commit the changes and exit. You may be asked to reboot.

5.4.2.5 Signing Binaries

5.4.2.5.1 Signing Custom Kernel and UEFI Binaries

To sign a custom kernel or any other EFI binary (UEFI application, UEFI driver or OS loader) you want to have loaded by shim, you need the private part of the key and the certificate in PEM format.

To convert the certificate into PEM, run:

```
$ openssl x509 -in mok.der -inform DER -outform PEM -out mok.pem
```

Now, to sign your EFI binary, run:

```
$ sbsign --key mok.priv --cert mok.pem binary.efi --output binary.efi.signed
```

If you are using your db key, use the private part of the key and its associated certificate converted into PEM format for binary signing.

If the X.509 key certificate is enrolled in UEFI db or by way of shim, the binary should be loaded without an issue.

5.4.2.5.2 Signing Kernel Modules

The X.509 certificate you added must be visible to the kernel. To verify the keys visible to the kernel, run:

```
ubuntu@localhost:~$ sudo cat /proc/keys
```

For a straightforward result, run:

```
ubuntu@localhost:~$ dmesg | grep -i "X.509"
[    1.869521] Loading compiled-in X.509 certificates
[    1.875441] Loaded X.509 cert 'Build time autogenerated kernel key: b1a3fbd0178bdb7190387a4187e8e4b0eb476cdc'
[    1.941752] integrity: Loading X.509 certificate: UEFI:db
[    1.947636] integrity: Loaded X.509 cert 'YourSigningDbKey: a109f01707ba6769c4d546530ba1592c7daedc3b'
[    1.958736] integrity: Loading X.509 certificate: UEFI:db
[    1.964170] integrity: Loaded X.509 cert 'Microsoft Corporation UEFI CA 2011:
13adbf4309bd82709c8cd54f316ed522988a1bd4'
[    2.023740] integrity: Loading X.509 certificate: UEFI:MokListRT
[    2.030090] integrity: Loaded X.509 cert 'YourSingingMokKey: 2012e5122669ffc0cc28827c6134329a6bec0b88'
[    2.040796] integrity: Loading X.509 certificate: UEFI:MokListRT
[    2.046830] integrity: Loaded X.509 cert 'SomeOrg: shim: 331c1c8963538e327d6e39346f4f53b200987015'
[    2.055796] integrity: Loading X.509 certificate: UEFI:MokListRT
[    2.062114] integrity: Loaded X.509 cert 'Canonical Ltd. Master Certificate Authority:
ad91990bc22ab1f517048c23b6655a268e345a63'
```

If the X.509 certificate attributes (`commonName` , etc.) are configured properly, you should see your key certificate information in the result output. In this example, two custom keys are visible to the kernel:

- `YourSigningMokKey` - registered with the shim as a MOK
- `YourSigningDbKey` - registered with UEFI as db

> ⚠ This example is for illustration purposes only. The actual output might differ from the output shown in this example depending on what key was previously enrolled and how it was enrolled.

You may sign kernel modules using either of these approaches:

- `kmodsign` command
- Linux kernel script sign-file

5.4.2.5.2.1 Signing Kernel Modules Using kmodsign

If you are using the `kmodsign` command to sign kernel modules, run:

```
ubuntu@localhost:~$ sudo cat /proc/keys
```

The signature is appended to the kernel module by `kmodsign` .

But if you rather keep the original kernel module unchanged, run:

```
ubuntu@localhost:~$ kmodsign sha512 mok.priv mok.der module.ko module-signed.ko
```

Refer to `kmosign --help` for more information.

5.4.2.5.2.2 Signing Kernel Modules Using Sign File

To sign the kernel module using the Linux kernel script sign-file, please refer to Linux kernel documentation.

If you are using your db key, use the private part of the key and its associated certificate for binary signing.

To validate that the module is signed, check that it includes the string `~Module signature appended~`:

```
ubuntu@localhost:~$ hexdump -Cv module.ko | tail -n 5
00002c20  10 14 08 cd eb 67 a8 3d  ac 82 e1 1d 46 b5 5c 91  |.....g.=....F.\.|
00002c30  9c cb 47 f7 c9 77 00 00  02 00 00 00 00 00 00 00  |..G..w..........|
00002c40  02 9e 7e 4d 6f 64 75 6c  65 20 73 69 67 6e 61 74  |..~Module signat|
00002c50  75 72 65 20 61 70 70 65  6e 64 65 64 7e 0a        |ure appended~.|
00002c5e
```

5.4.2.5.3 Ongoing Updates

5.4.2.5.3.1 Update Key Certificates

> ⚠ This requires UEFI secure boot to have been enabled using your own keys, which means that you own the signing keys.

While UEFI secure boot is enabled, it is possible to update your keys using a capsule file.

To create a capsule intended to update the UEFI secure boot keys, generate a new set of keys and then run:

```
$ ./mlx-mkcap --pk-key new_pk.cer --kek-key new_kek.cer --db-key new_db1.cer --db-key new_db2.cer --db-key
new_db3.cer --signer-key db.key --signer-cert db.pem EnrollYourNewKeysCap
```

Note that `--signer-key` and `--signer-cert` are set so the capsule is signed. When UEFI secure boot is enabled, the capsule is verified using the key certificates previously enrolled in the UEFI database. It is important to use the old signing keys associated with the certificates in the UEFI database to sign the capsule. The new key certificates are intended to replace the existing key certificates after capsule processing. Once the UEFI database is updated, the new keys must be used to sign the newly created capsule files.

To enroll the new set of keys, download the capsule file to the BlueField console and use `bfrec` to initiate the capsule update.

5.4.2.5.3.2 Disable UEFI Secure Boot Using a Capsule

> ⚠ This requires UEFI secure boot to have been enabled using your own keys, which means that you own the signing keys.

It is possible to disable UEFI secure boot through a capsule update. This requires an empty PK key when creating the capsule file.

To create a capsule intended to disable UEFI secure boot:

1. Create a dummy empty PK certificate:

```
$ touch null_pk.cer
```

2. Create the capsule file:

```
$ ./mlx-mkcap --pk-key null_pk.cer --signer-key db.key --signer-cert db.pem DeletePkCap
```

`--signer-key` and `--signer-cert` must be specified with the appropriate private keys and certificates associated with the actual key certificates in the UEFI database.

To enroll the empty PK certificate, download the capsule file to the BlueField console and use `bfrec` to initiate the capsule update.

> Deleting the PK certificate will result in UEFI secure boot to be disabled which is not recommended in a production environment.

5.4.3 Updating Platform Firmware

To update the platform firmware on secured devices, download the latest NVIDIA® BlueField® software images from NVIDIA.com.

5.4.3.1 Updating eMMC Boot Partitions Image

The capsule file `/lib/firmware/mellanox/boot/capsule/MmcBootCap` is used to update the eMMC boot partition and update the Arm pre-boot code (i.e., Arm trusted firmware and UEFI).

The capsule file is signed with NVIDIA keys. If UEFI secure boot is enabled, make sure the NVIDIA certificate files are enrolled into the UEFI database. Please refer to "UEFI Secure Boot" for more information on how to update the UEFI database key certificates.

To initiate the update of the eMMC boot partitions, run the following command:

```
ubuntu@localhost:~$ sudo bfrec --capsule /lib/firmware/mellanox/boot/capsule/MmcBootCap
```

After the command completes, reboot the system to process the capsule file. On the next reboot, UEFI will verify the capsule signature. If verified, UEFI will process the capsule file, extract the pre-boot image and burn it into the eMMC boot partitions.

Note that the pre-boot code is signed with the NVIDIA key. The bootloader images are installed into the eMMC with their associated certificate files. The public key is derived from the certificate file and its integrity is verified by the ROM code against an on-chip public key hash value stored in E-FUSEs. If the verification fails, then the pre-boot code will not be allowed to execute.

5.4.3.1.1 Recovering eMMC Boot Partition

If the system cannot boot from the eMMC boot partitions for any reason, it is recommended to download a valid BFB image and boot it over the BlueField platform.

The recovery path relies on the platform to be configured to boot solely from the RShim interface (either RShim USB or RShim PCIe). With this configuration there must not be a way to interrupt or bypass the RoT when secure booting.

You will need to append a capsule file to the BFB prior to booting. Run:

```
$ mlx-mkbfb --capsule MmcBootCap install.bfb recovery_install.bfb
```

Then boot the `recovery_install.bfb` using the RShim interface. Run:

```
$ cat recovery_install.bfb > /dev/rshim0/boot
```

The capsule file will be processed by UEFI upon boot.

5.4.3.2 Updating SPI Flash FS4 Image

The SPI flash contains the firmware image of the DPU firmware in FS4 format. The firmware image is provided along with the software.

There are two different ways to install the firmware image:

- From the BlueField console, using the following command:

  ```
  ubuntu@localhost:~$ /opt/mellanox/mlnx-fw-updater/firmware/mlxfwmanager_sriov_dis_aarch64_<bf-dev>
  ```

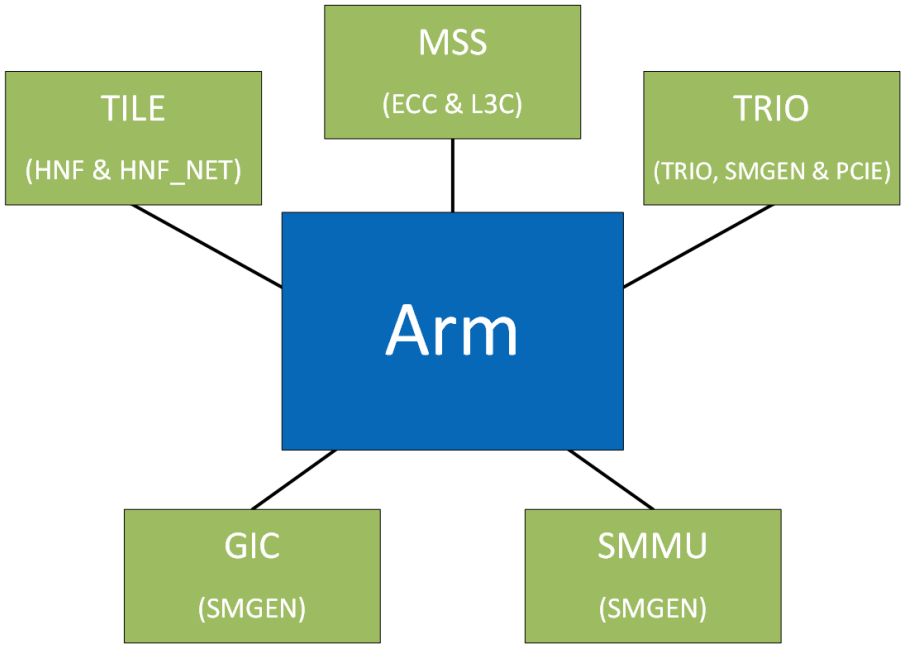
- From the PCIe host console, using the following command:

  ```
  # flint -d /dev/mst/mt<bf-dev>_pciconf0 -i firmware.bin b
  ```

`bf-dev` is 41686 for BlueField-2 or 41692 for BlueField-3.

6 Management

6.1 Performance Monitoring Counters



The performance modules in NVIDIA® BlueField® are present in several hardware blocks and each block has a certain set of supported events.

The `mlx_pmc` driver provides access to all of these performance modules through a sysfs interface. The driver creates a directory under `/sys/class/hwmon` under which each of the blocks explained above has a subdirectory. Please note that all directories under `/sys/class/hwmon` are named as "`hwmon<N>`" where `N` is the `hwmon` device number corresponding to the device. This is assigned by Linux and could change with the addition of more devices to the `hwmon` class. Each `hwmon` directory has a "`name`" node which can be used to identify the correct device. In this case, reading the "`name`" file should return "`bfperf`".

The hardware blocks that include performance modules are:

- Tile (block containing 2 cores and a shared L2 cache) has 2 sets of counters, one set for HNF and HNF_NET events. These are present as "tile" and "tilenet" directories in the sysfs interface of the driver.
- TRIO (PCIe root complex) has 3 sets of counters, one each for TRIO, SMGEN and PCIE TLR events. The sysfs directories for these are called "trio", "triogen" and "pcie" respectively.

- MSS (memory sub-system containing the memory controller and L3 cache)
- GIC and SMMU with one set of counters each for the SMGEN events. These are simply labelled "gic" and "smmu" respectively.

The number of Tile, TRIO and MSS blocks depends on the system. There is a maximum of 8 Tile, 3 TRIO and 2 MSS blocks in BlueField, and this is added as a suffix to the sysfs directory names. For example, this is a list of directories present in a BlueField-2 system:

```
ubuntu@bf:/$ ls /sys/class/hwmon/hwmon0/
device l3cachehalf0 pcie0 smmu0 tile1 tilenet0 tilenet3 triogen0
ecc l3cachehalf1 pcie1 subsystem tile2 tilenet1 trio0 triogen1
gic0 name power  tile0 tile3 tilenet2 trio1 uevent
```

The PCIe TLR statistics for each TRIO are under the "pcie" block.

6.1.1 Performance Data Collection Mechanisms

The performance data of the BlueField hardware is collected using two mechanisms:

1. Programming hardware counters to monitor specific events
2. Reading registers that hold performance/event statistics

All blocks except "ecc" and "pcie" use the mechanism 1.

6.1.1.1 Using Hardware Counters

For blocks that use hardware counters to collect data, each counter present in the block is represented by "`event<N>`" and "`counter<N>`" sysfs files.

For example:

```
ubuntu@bf:/$ ls /sys/class/hwmon/hwmon0/tile0/
counter0 counter1 counter2 counter3 event0 event1 event2 event3 event_list
```

An `event<N>` and `counter<N>` pair can be used to program and monitor events. The "`event_list`" sysfs file displays the list of events supported by that block along with the hexadecimal value corresponding to each event.

Use the `echo` command to write the event number to the `event<N>` file, and use the `cat` command to read the counter value from the corresponding counter (`counter<N>`).

The counters are enabled individually once the event number is written to the corresponding event file. However, the L3 cache performance counters cannot be enabled or disabled individually and can only be triggered or stopped all at the same time.

So in the example provided, all 4 event files may be programmed with the necessary event numbers and then the "enable" file may be used to start the counters. Writing `0` to the enable file stops the counters while `1` starts them.

6.1.1.2 Reading Registers

For "ecc" and "pcie" blocks, the counters cannot be started or stopped by the user, instead the statistics are automatically collected by HW and stored in registers. These register names are exposed within the directory and can be read by the user at any time.

6.1.2 List of Supported Events

6.1.2.1 SMGEN Performance Module

Hex Value	Name	Description
0x0	AW_REQ	Reserved for internal use
0x1	AW_BEATS	Reserved for internal use
0x2	AW_TRANS	Reserved for internal use
0x3	AW_RESP	Reserved for internal use
0x4	AW_STL	Reserved for internal use
0x5	AW_LAT	Reserved for internal use
0x6	AW_REQ_TBU	Reserved for internal use
0x8	AR_REQ	Reserved for internal use
0x9	AR_BEATS	Reserved for internal use
0xa	AR_TRANS	Reserved for internal use
0xb	AR_STL	Reserved for internal use
0xc	AR_LAT	Reserved for internal use
0xd	AR_REQ_TBU	Reserved for internal use
0xe	TBU_MISS	The number of TBU miss
0xf	TX_DAT_AF	Mesh Data channel write FIFO almost Full. This is from the TRIO toward the Arm memory.
0x10	RX_DAT_AF	Mesh Data channel read FIFO almost Full. This is from the Arm memory toward the TRIO.
0x11	RETRYQ_CRED	Reserved for internal use

6.1.2.2 Tile HNF Performance Module

Hex Value	Name	Description
0x45	HNF_REQUESTS	Number of REQs that were processed in HNF
0x46	HNF_REJECTS	Reserved for internal use
0x47	ALL_BUSY	Reserved for internal use
0x48	MAF_BUSY	Reserved for internal use
0x49	MAF_REQUESTS	Reserved for internal use

Hex Value	Name	Description
0x4a	RNF_REQUESTS	Number of REQs sent by the RN-F selected by HNF_PERF_CTL register RNF_SEL field
0x4b	REQUEST_TYPE	Reserved for internal use
0x4c	MEMORY_READS	Number of reads to MSS
0x4d	MEMORY_WRITES	Number of writes to MSS
0x4e	VICTIM_WRITE	Number of victim lines written to memory
0x4f	POC_FULL	Reserved for internal use
0x50	POC_FAIL	Number of times that the POC Monitor sent RespErr Okay status to an Exclusive WriteNoSnp or CleanUnique REQ
0x51	POC_SUCCESS	Number of times that the POC Monitor sent RespErr ExOkay status to an Exclusive WriteNoSnp or CleanUnique REQ
0x52	POC_WRITES	Number of Exclusive WriteNoSnp or CleanUnique REQs processed by POC Monitor
0x53	POC_READS	Number of Exclusive ReadClean/ReadShared REQs processed by POC Monitor
0x54	FORWARD	Reserved for internal use
0x55	RXREQ_HNF	Reserved for internal use
0x56	RXRSP_HNF	Reserved for internal use
0x57	RXDAT_HNF	Reserved for internal use
0x58	TXREQ_HNF	Reserved for internal use
0x59	TXRSP_HNF	Reserved for internal use
0x5a	TXDAT_HNF	Reserved for internal use
0x5b	TXSNP_HNF	Reserved for internal use
0x5c	INDEX_MATCH	Reserved for internal use
0x5d	A72_ACCESS	Access requests (Reads, Writes, CopyBack, CMO, DVM) from A72 clusters
0x5e	IO_ACCESS	Accesses requests (Reads, Writes) from DMA IO devices
0x5f	TSO_WRITE	Total Store Order write Requests from DMA IO devices
0x60	TSO_CONFLICT	Reserved for internal use
0x61	DIR_HIT	Requests that hit in directory
0x62	HNF_ACCEPTS	Reserved for internal use
0x63	REQ_BUF_EMPTY	Number of cycles when request buffer is empty
0x64	REQ_BUF_IDLE_MAF	Reserved for internal use
0x65	TSO_NOARB	Reserved for internal use
0x66	TSO_NOARB_CYCLES	Reserved for internal use
0x67	MSS_NO_CREDIT	Number of cycles that a Request could not be sent to MSS due to lack of credits

Hex Value	Name	Description
0x68	TXDAT_NO_LCRD	Reserved for internal use
0x69	TXSNP_NO_LCRD	Reserved for internal use
0x6a	TXRSP_NO_LCRD	Reserved for internal use
0x6b	TXREQ_NO_LCRD	Reserved for internal use
0x6c	TSO_CL_MATCH	Reserved for internal use
0x6d	MEMORY_READS_BYPASS	Number of reads to MSS that bypass Home Node
0x6e	TSO_NOARB_TIMEOUT	Reserved for internal use
0x6f	ALLOCATE	Number of times that Directory entry was allocated
0x70	VICTIM	Number of times that Directory entry allocation did not find an Invalid way in the set
0x71	A72_WRITE	Write requests from A72 clusters
0x72	A72_Read	Read requests from A72 clusters
0x73	IO_WRITE	Write requests from DMA IO devices
0x74	IO_Reads	Read requests from DMA IO devices
0x75	TSO_Reject	Reserved for internal use
0x80	TXREQ_RN	Reserved for internal use
0x81	TXRSP_RN	Reserved for internal use
0x82	TXDAT_RN	Reserved for internal use
0x83	RXSNP_RN	Reserved for internal use
0x84	RXRSP_RN	Reserved for internal use
0x85	RXDAT_RN	Reserved for internal use

6.1.2.3 TRIO Performance Module

Hex Value	Name	Description
0xa0	TPIO_DATA_BEAT	Data beats from Arm PIO to TRIO
0xa1	TDMA_DATA_BEAT	Data beats from Arm memory to PCI completion
0xa2	MAP_DATA_BEAT	Reserved for internal use
0xa3	TXMSG_DATA_BEAT	Reserved for internal use
0xa4	TPIO_DATA_PACKET	Data packets from Arm PIO to TRIO
0xa5	TDMA_DATA_PACKET	Data packets from Arm memory to PCI completion
0xa6	MAP_DATA_PACKET	Reserved for internal use

Hex Value	Name	Description
0xa7	TXMSG_DATA_PACKET	Reserved for internal use
0xa8	TDMA_RT_AF	The in-flight PCI DMA READ request queue is almost full
0xa9	TDMA_PBUF_MAC_AF	Indicator of the buffer of Arm memory reads is too full awaiting PCIe access
0xaa	TRIO_MAP_WRQ_BUF_EMPTY	PCIe write transaction buffer is empty
0xab	TRIO_MAP_CPL_BUF_EMPTY	Arm PIO request completion queue is empty
0xac	TRIO_MAP_RDQ0_BUF_EMPTY	The buffer of MAC0's read transaction is empty
0xad	TRIO_MAP_RDQ1_BUF_EMPTY	The buffer of MAC1's read transaction is empty
0xae	TRIO_MAP_RDQ2_BUF_EMPTY	The buffer of MAC2's read transaction is empty
0xaf	TRIO_MAP_RDQ3_BUF_EMPTY	The buffer of MAC3's read transaction is empty
0xb0	TRIO_MAP_RDQ4_BUF_EMPTY	The buffer of MAC4's read transaction is empty
0xb1	TRIO_MAP_RDQ5_BUF_EMPTY	The buffer of MAC5's read transaction is empty
0xb2	TRIO_MAP_RDQ6_BUF_EMPTY	The buffer of MAC6's read transaction is empty
0xb3	TRIO_MAP_RDQ7_BUF_EMPTY	The buffer of MAC7's read transaction is empty

6.1.2.4 L3 Cache Performance Module

The L3 cache interfaces with the Arm cores via the SkyMesh. The CDN is used for control data. The NDN is used for responses. The DDN is for the actual data transfer.

Hex Value	Name	Description
0x00	DISABLE	Reserved for internal use
0x01	CYCLES	Timestamp counter
0x02	TOTAL_RD_REQ_IN	Read Transaction control request from the CDN of the SkyMesh
0x03	TOTAL_WR_REQ_IN	Write transaction control request from the CDN of the SkyMesh
0x04	TOTAL_WR_DBID_ACK	Write transaction control responses from the NDN of the SkyMesh
0x05	TOTAL_WR_DATA_IN	Write transaction data from the DDN of the SkyMesh
0x06	TOTAL_WR_COMP	Write completion response from the NDN of the SkyMesh
0x07	TOTAL_RD_DATA_OUT	Read transaction data from the DDN
0x08	TOTAL_CDN_REQ_IN_BANK0	CHI CDN Transactions Bank 0

Hex Value	Name	Description
0x09	TOTAL_CDN_REQ_IN_BANK1	CHI CDN Transactions Bank 1
0x0a	TOTAL_DDN_REQ_IN_BANK0	CHI DDN Transactions Bank 0
0x0b	TOTAL_DDN_REQ_IN_BANK1	CHI DDN Transactions Bank 1
0x0c	TOTAL_EMEM_RD_RES_IN_BANK0	Total EMEM Read Response Bank 0
0x0d	TOTAL_EMEM_RD_RES_IN_BANK1	Total EMEM Read Response Bank 1
0x0e	TOTAL_CACHE_RD_RES_IN_BANK0	Total Cache Read Response Bank 0
0x0f	TOTAL_CACHE_RD_RES_IN_BANK1	Total Cache Read Response Bank 1
0x10	TOTAL_EMEM_RD_REQ_BANK0	Total EMEM Read Request Bank 0
0x11	TOTAL_EMEM_RD_REQ_BANK1	Total EMEM Read Request Bank 1
0x12	TOTAL_EMEM_WR_REQ_BANK0	Total EMEM Write Request Bank 0
0x13	TOTAL_EMEM_WR_REQ_BANK1	Total EMEM Write Request Bank 1
0x14	TOTAL_RD_REQ_OUT	EMEM Read Transactions Out
0x15	TOTAL_WR_REQ_OUT	EMEM Write Transactions Out
0x16	TOTAL_RD_RES_IN	EMEM Read Transactions In
0x17	HITS_BANK0	Number of Hits Bank 0
0x18	HITS_BANK1	Number of Hits Bank 1
0x19	MISSES_BANK0	Number of Misses Bank 0
0x1a	MISSES_BANK1	Number of Misses Bank 1
0x1b	ALLOCATIONS_BANK0	Number of Allocations Bank 0
0x1c	ALLOCATIONS_BANK1	Number of Allocations Bank 1
0x1d	EVICTIONS_BANK0	Number of Evictions Bank 0
0x1e	EVICTIONS_BANK1	Number of Evictions Bank 1
0x1f	DBID_REJECT	Reserved for internal use
0x20	WRDB_REJECT_BANK0	Reserved for internal use
0x21	WRDB_REJECT_BANK1	Reserved for internal use
0x22	CMDQ_REJECT_BANK0	Reserved for internal use
0x23	CMDQ_REJECT_BANK1	Reserved for internal use
0x24	COB_REJECT_BANK0	Reserved for internal use
0x25	COB_REJECT_BANK1	Reserved for internal use
0x26	TRB_REJECT_BANK0	Reserved for internal use
0x27	TRB_REJECT_BANK1	Reserved for internal use
0x28	TAG_REJECT_BANK0	Reserved for internal use
0x29	TAG_REJECT_BANK1	Reserved for internal use
0x2a	ANY_REJECT_BANK0	Reserved for internal use

Hex Value	Name	Description
0x2b	ANY_REJECT_BANK1	Reserved for internal use

6.1.2.5 PCIe TLR Statistics

Hex Value	Name	Description
0x0	PCIE_TLR_IN_P_PKT_CNT	Incoming posted packets
0x10	PCIE_TLR_IN_NP_PKT_CNT	Incoming non-posted packets
0x18	PCIE_TLR_IN_C_PKT_CNT	Incoming completion packets
0x20	PCIE_TLR_OUT_P_PKT_CNT	Outgoing posted packets
0x28	PCIE_TLR_OUT_NP_PKT_CNT	Outgoing non-posted packets
0x30	PCIE_TLR_OUT_C_PKT_CNT	Outgoing completion packets
0x38	PCIE_TLR_IN_P_BYTE_CNT	Incoming posted bytes
0x40	PCIE_TLR_IN_NP_BYTE_CNT	Incoming non-posted bytes
0x48	PCIE_TLR_IN_C_BYTE_CNT	Incoming completion bytes
0x50	PCIE_TLR_OUT_C_BYTE_CNT	Outgoing posted bytes
0x58	PCIE_TLR_OUT_NP_BYTE_CNT	Outgoing non-posted bytes
0x60	PCIE_TLR_OUT_C_BYTE_CNT	Outgoing completion bytes

6.1.2.6 Tile HNFNET Performance Module

Hex Value	Name	Description
0x12	CDN_REQ	The number of CDN requests
0x13	DDN_REQ	The number of DDN requests
0x14	NDN_REQ	The number of NDN requests
0x15	CDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the CDN network
0x16	CDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the CDN network
0x17	CDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the CDN network
0x18	CDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the CDN network
0x19	CDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the CDN network
0x1a	CDN_DIAG_N_EGRESS	Packets sent out from north port in the CDN network
0x1b	CDN_DIAG_S_EGRESS	Packets sent out from south port in the CDN network
0x1c	CDN_DIAG_E_EGRESS	Packets sent out from east port in the CDN network

Hex Value	Name	Description
0x1d	CDN_DIAG_W_EGRESS	Packets sent out from west port in the CDN network
0x1e	CDN_DIAG_C_EGRESS	Packets sent out from core port in the CDN network
0x1f	CDN_DIAG_N_INGRESS	Packets received by north port in the CDN network
0x20	CDN_DIAG_S_INGRESS	Packets received by south port in the CDN network
0x21	CDN_DIAG_E_INGRESS	Packets received by east port in the CDN network
0x22	CDN_DIAG_W_INGRESS	Packets received by west port in the CDN network
0x23	CDN_DIAG_C_INGRESS	Packets received by core port in the CDN network
0x24	CDN_DIAG_CORE_SENT	Packets completed from core port in the CDN network
0x25	DDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the DDN network
0x26	DDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the DDN network
0x27	DDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the DDN network
0x28	DDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the DDN network
0x29	DDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the DDN network
0x2a	DDN_DIAG_N_EGRESS	Packets sent out from north port in the DDN network
0x2b	DDN_DIAG_S_EGRESS	Packets sent out from south port in the DDN network
0x2c	DDN_DIAG_E_EGRESS	Packets sent out from east port in the DDN network
0x2d	DDN_DIAG_W_EGRESS	Packets sent out from west port in the DDN network
0x2e	DDN_DIAG_C_EGRESS	Packets sent out from core port in the DDN network
0x2f	DDN_DIAG_N_INGRESS	Packets received by north port in the DDN network
0x30	DDN_DIAG_S_INGRESS	Packets received by south port in the DDN network
0x31	DDN_DIAG_E_INGRESS	Packets received by east port in the DDN network
0x32	DDN_DIAG_W_INGRESS	Packets received by west port in the DDN network
0x33	DDN_DIAG_C_INGRESS	Packets received by core port in the DDN network
0x34	DDN_DIAG_CORE_SENT	Packets completed from core port in the DDN network
0x35	NDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the NDN network
0x36	NDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the NDN network
0x37	NDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the NDN network
0x38	NDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the NDN network
0x39	NDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the NDN network

Hex Value	Name	Description
0x3a	NDN_DIAG_N_EGRESS	Packets sent out from north port in the NDN network
0x3b	NDN_DIAG_S_EGRESS	Packets sent out from south port in the NDN network
0x3c	NDN_DIAG_E_EGRESS	Packets sent out from east port in the NDN network
0x3d	NDN_DIAG_W_EGRESS	Packets sent out from west port in the NDN network
0x3e	NDN_DIAG_C_EGRESS	Packets sent out from core port in the NDN network
0x3f	NDN_DIAG_N_INGRESS	Packets received by north port in the NDN network
0x40	NDN_DIAG_S_INGRESS	Packets received by south port in the NDN network
0x41	NDN_DIAG_E_INGRESS	Packets received by east port in the NDN network
0x42	NDN_DIAG_W_INGRESS	Packets received by west port in the NDN network
0x43	NDN_DIAG_C_INGRESS	Packets received by core port in the NDN network
0x44	NDN_DIAG_CORE_SENT	Packets completed from core port in the NDN network

6.1.3 Programming Counter to Monitor Events

To program a counter to monitor one of the events from the event list, the event name or number needs to be written to the corresponding event file.

Let us call the `/sys/class/hwmon/hwmon<N>` folder corresponding to this driver as `BFPERF_DIR`.

For example, to monitor the event `HNF_REQUESTS` (`0x45`) on `tile2` using counter 3:

```
$ echo 0x45 > <BFPERF_DIR>/tile2/event3
```

Or:

```
$ echo HNF_REQUESTS > <BFPERF_DIR>/tile2/event3
```

Once this is done, `counter3` resets the counter and starts monitoring the number of `HNF_REQUESTS`.

To read the counter value, run:

```
$ cat <BFPERF_DIR>/tile2/counter3
```

To see what event is currently being monitored by a counter, just read the corresponding event file to get the event name and number.

```
$ cat <BFPERF_DIR>/tile2/event3
```

In this case, reading the `event3` file returns "`0x45: HNF_REQUESTS`".

To clear the counter, write 0 to the counter file.

```
$ echo 0 > <BFPERF_DIR>/tile2/counter3
```

This resets the accumulator and the counter continues monitoring the same event that has previously been programmed, but starts the count from 0 again. Writing non-zero values to the counter files is not allowed.

To stop monitoring an event, write `0xff` to the corresponding event file.

This is slightly different for the l3cache blocks due to the restriction that all counters can only be enabled, disabled, or reset together. So once the event is written to the event file, the counters will have to be enabled to start monitoring their respective events by writing "1" to the "enable" file. Writing "0" to this file will stop all the counters. The most reliable way to get accurate counter values would be by disabling the counters after a certain time period and then proceeding to read the counter values.

> ⚠ Programming a counter to monitor a new event automatically stops all the counters. Also, enabling the counters resets the counters to 0 first.

For blocks that have performance statistics registers (mechanism 2), all of these statistics are directly made available to be read or reset.

For example, to read the number of incoming posted packets to TRIO2:

```
$ cat <BFPERF_DIR>/pcie2/IN_P_PKT_CNT
```

The count can be reset to 0 by writing 0 to the same file. Again, non-zero writes to these files are not allowed.

6.2 Intelligent Platform Management Interface

6.2.1 BMC Retrieving Data from BlueField via IPMB

NVIDIA® BlueField® DPU® software will respond to Intelligent Platform Management Bus (IPMB) commands sent from the BMC via its Arm I^2C bus.

> ⚠ The BlueField `ipmb_dev_int` driver is registered at the 7-bit I^2C address `0x30` by default. The I2C address of the BlueField can be changed in the file `usr/bin/set_emu_param.sh`.
>
> - BlueField Controller cards provide connection from the host server BMC to BlueField Arm I^2C bus
> - BlueField DPUs provide connection from the host server BMC to the BlueField NC-SI port
> - BlueField Reference Platforms provide connection from its on-board BMC to BlueField Arm I^2C bus

6.2.1.1 List of IPMI Supported Sensors

Sensor	ID	Description
`bluefield_temp`	0	Support NIC monitoring of BlueField's temperature
`ddr0_0_temp` [a]	1	Support monitoring of DDR0 temp (on memory controller 0)
`ddr0_1_temp` [a]	2	Support monitoring of DDR1 temp (on memory controller 0)
`ddr1_0_temp` [a]	3	Support monitoring of DDR0 temp (on memory controller 1)
`ddr1_1_temp` [a]	4	Support monitoring of DDR1 temp (on memory controller 1)
`p0_temp`	5	Port 0 temperature
`p1_temp`	6	Port 1 temperature
`p0_link`	7	Port0 link status
`p1_link`	8	Port1 link status

[a] These sensors are not available, and hence are not populated, on BlueField DPUs.

On BlueField-2 based boards, DDR sensors and FRUs are not supported. They will appear as no reading.

6.2.1.2 List of IPMI Supported FRUs

FRU	ID	Description
`update_timer`	0	`set_emu_param.service` is responsible for collecting data on sensors and FRUs every 3 seconds. This regular update is required for sensors but not for FRUs whose content is less susceptible to change. `update_timer` is used to sample the FRUs every hour instead. Users may need this timer in the case where they are issuing several raw IPMItool FRU read commands. This helps in assessing how much time users have to retrieve large FRU data before the next FRU update. `update_timer` is a hexadecimal number.
`fw_info`	1	NVIDIA® ConnectX® firmware information, Arm firmware version, and MLNX_OFED version. The `fw_info` is in ASCII format.
`nic_pci_dev_info`	2	NIC vendor ID, device ID, subsystem vendor ID, and subsystem device ID. The `nic_pci_dev_info` is in ASCII format.
`cpuinfo`	3	CPU information reported in `lscpu` and `/proc/cpuinfo`. The `cpuinfo` is in ASCII format.

FRU	ID	Description
`ddr0_0_spd` [a]	4	FRU for SPD MC0 DIMM 0 (MC = memory controller). The `ddr0_0_spd` is in binary format.
`ddr0_1_spd` [a]	5	FRU for SPD MC0 DIMM1. The `ddr0_1_spd` is in binary format.
`ddr1_0_spd` [a]	6	FRU for SPD MC1 DIMM0. The `ddr1_0_spd` is in binary format.
`ddr1_1_spd` [a]	7	FRU for SPD MC1 DIMM1. The `ddr1_1_spd` is in binary format.
`emmc_info`	8	eMMC size, list of its partitions, and partitions usage (in ASCII format). eMMC CID, CSD, and extended CSD registers (in binary format). The ASCII data is separated from the binary data with "StartBinary" marker.
`qsfp0_eeprom`	9	FRU for QSFP 0 EEPROM page 0 content (256 bytes in binary format)
`qsfp1_eeprom`	10	FRU for QSFP 1 EEPROM page 0 content (256 bytes in binary format)

`ip_addresses`	11	This FRU file can be used to write the BMC port 0 and port 1 IP addresses to the BlueField. It is empty to begin with.
		The file passed through the "`ipmitool fru write 11 <file>`" command must have the following format:
		`BMC: XXX.XXX.XXX.XXX`
		`P0: XXX.XXX.XXX.XXX`
		`P1: XXX.XXX.XXX.XXX`
		The size of the written file should be exactly 61 bytes.

`dimms_ce_ue`	12	FRU reporting the number of correctable and uncorrectable errors in the DIMMs. This FRU is updated once every 3 seconds.
`eth0`	13	Network interface 0 information. Updated once every minute.
`eth1`	14	Network interface 1 information. Updated once every minute.
`bf_uid`	15	BlueField UID

FRU	ID	Description
`eth_hw_counters`	16	List of ConnectX interface hardware counters

⚠ [a] On BlueField-2 based boards, DDR sensors and FRUs are not supported. They will appear as no reading.

6.2.1.3 Supported IPMI Commands

The table below provides a list of supported IPMItool command arguments.

They can be issued from the BMC in the following format:

```
ipmitool -I ipmb <ipmitool_command_argument>
```

BlueField software responds to IPMItool commands issued on BlueField console. IPMItool commands on Bluefield console are supported regardless if a host server BMC is connected to the Arm I^2C bus on BlueField.

The format for these commands is as follows:

```
$ ipmitool -U ADMIN -P ADMIN -p 9001 -H localhost <ipmitool_command_argument>
```

Command Description	IPMItool Command	Relevant IPMI 2.0 Rev 1.1 Spec Section
Get device ID	mc info	20.1
Broadcast “Get Device ID”	Part of "mc info"	20.9
Get BMC global enables	mc getenables	22.2
Get device SDR info	sdr info	35.2
Get device SDR	"sdr get", "sdr list" or "sdr elist"	35.3
Get sensor hysteresis	sdr get <sensor-id>	35.7

Command Description	IPMItool Command	Relevant IPMI 2.0 Rev 1.1 Spec Section
Set sensor threshold	sensor thresh <sensor-id> <threshold> <setting> • sensor-id - name of the sensor for which a threshold is to be set threshold - which threshold to set • ucr - upper critical • unc - upper non-critical • lnc - lower non-critical • lcr - lower critical • setting - the value to set the threshold to To configure all lower thresholds, use: sensor thresh <sensor-id> lower <lnr> <lcr> <lnc> ⚠ The lower non-recoverable <lnr> option is not supported To configure all upper thresholds, use: sensor thresh <sensor-id> upper <unc> <ucr> <unr> ⚠ The upper non-recoverable <unr> option is not supported	35.8
Get sensor threshold	sdr get <sensor-id>	35.9
Get sensor event enable	sdr get <sensor-id>	35.11
Get sensor reading	sensor reading <sensor-id>	35.14
Get sensor type	sdr type <type>	35.16
Read FRU data	fru read <fru-number> <file-to-write-to>	34.2
Get SDR repository info	sdr info	33.9
Get SEL info	"sel" or "sel info"	40.2
Get SEL allocation info	"sel" or "sel info"	40.3
Get SEL entry	"sel list" or "sel elist"	40.5
Add SEL entry	sel add <filename>	40.6
Delete SEL entry	sel delete <id>	40.8
Clear SEL	sel clear	40.9
Get SEL time	sel time get	40.1
Set SEL time	sel time set "MM/DD/YYYY HH:M:SS"	40.11

6.2.2 Loading and Using IPMI on BlueField Running CentOS

1. Load the BlueField CentOS image.

⚠ The following steps are performed from the BlueField CentOS prompt. The BlueField is running CentOS 7.6 with kernel 5.4. The CentOS installation was done using the CentOS everything ISO image.

The following drivers need to be loaded on the BlueField running CentOS:

- jc42.ko
- ee1004.ko
- at24.ko
- eeprom.ko
- i2c-dev.ko

Example of loading ee1004.ko, at24.ko, and eeprom.ko:

```
modprobe ee1004
modprobe at24
modprobe eeprom
```

The i2c-dev module is built into the kernel 5.4.60 on CentOS 7.6.

2. Optional: Update the i2c-mlx driver if the installed version is older than version i2c-mlx-1.0-0.gab579c6.src.rpm.
    a. Re-compile i2c-mlx. Run:

    ```
    $ yum remove -y kmod-i2c-mlx
    $ modprobe -rv i2c-mlx
    ```

    b. Transfer the i2c-mlx RPM from the BlueField software tarball under distro/SRPM onto the Arm. Run:

    ```
    $ rpmbuild --rebuild /root/i2c-mlx-1.0-0.g422740c.src.rpm
    $ yum install -y /root/rpmbuild/RPMS/aarch64/i2c-
    mlx-1.0-0.g422740c_5.4.17_mlnx.9.ga0bea68.aarch64.rpm
    $ ls -l /lib/modules/$(uname -r)/extra/i2c-mlx/i2c-mlx.ko
    ```

    c. Load i2c-mlx. Run:

    ```
    $ modprobe i2c-mlx
    ```

3. Install the following packages:

    ```
    $ yum install ipmitool lm_sensors
    ```

    If the above operation fails for IPMItool, run the following to install it:

    ```
    wget http://sourceforge.net/projects/ipmitool/files/ipmitool/1.8.18/ipmitool-1.8.18.tar.gz
    tar -xvzf ipmitool-1.8.18.tar.gz
    cd ipmitool-1.8.18
    ./bootstrap
    ./configure
    make
    make install DESTDIR=/tmp/package-ipmitool
    ```

4. The i2c-tools package is also required, but the version contained in the CentOS Yum repository is old and does not work with BlueField. Therefore, please download i2c-tools version 4.1, and then build and install it.

    ```
    # Build i2c-tools from a newer source
    wget http://mirrors.edge.kernel.org/pub/software/utils/i2c-tools/i2c-tools-4.1.tar.gz
    tar -xvzf i2c-tools-4.1.tar.gz
    cd i2c-tools-4.1
    make
    make install PREFIX=/usr

    # create a link to the libraries
    ln -sfn /usr/lib/libi2c.so.0.1.1 /lib64/libi2c.so
    ln -sfn /usr/lib/libi2c.so.0.1.1 /lib64/libi2c.so.0
    ```

5. Generate an RPM binary from the BlueField's mlx-OpenIPMI-2.0.25 source RPM.

The following packages might be needed to build the binary RPM depending on which version of CentOS you are using.

```
$ yum install libtool rpm-devel rpmdevtools rpmlint wget ncurses-devel automake
$ rpmbuild --rebuild mlx-OpenIPMI-2.0.25-0.g581ebbb.src.rpm
```

You may obtain this rpm file by means of scp from the server host's Bluefield Distribution folder. For example:

```
$ scp <BF_INST_DIR>/distro/SRPMS/mlx-OpenIPMI-2.0.25-0.g4fdc53d.src.rpm <ip-address>:/
<target_directory>/
```

If there are issues with building the OpenIPMI RPM, verify that the swig package is not installed.

```
$ yum remove -y swig
```

6. Generate a binary RPM from the ipmb-dev-int source RPM and install it. Run:

```
$ rpmbuild --rebuild ipmb-dev-int-1.0-0.g304ea0c.src.rpm
```

7. Generate a binary RPM from the ipmb-host source RPM and install it. Run:

```
$ rpmbuild --rebuild ipmb-host-1.0-0.g304ea0c.src.rpm
```

8. Load OpenIPMI, ipmb-host, and ipmb-dev-int RPM packages. Run:

```
$ yum install -y /root/rpmbuild/RPMS/aarch64/mlx-
OpenIPMI-2.0.25-0.g581ebbb_5.4.0_49.el7a.aarch64.aarch64.rpm
$ yum install -y /root/rpmbuild/RPMS/aarch64/ipmb-dev-int-1.0-0.g304ea0c_5.4.0_49.el7a.aarch64.aarch64.rpm
$ yum install -y /root/rpmbuild/RPMS/aarch64/ipmb-host-1.0-0.g304ea0c_5.4.0_49.el7a.aarch64.aarch64.rpm
```

9. Load the IPMB driver. Run:

```
$ modprobe ipmb-dev-int
```

10. Install and start rasdaemon package. Run:

```
yum install rasdaemon
systemctl enable rasdaemon
systemctl start rasdaemon
```

11. Start the IPMI daemon. Run:

```
$ systemctl enable mlx_ipmid
$ systemctl start mlx_ipmid
$ systemctl enable set_emu_param
$ systemctl start set_emu_param
```

12. Test if the IPMI daemon responds on the BlueField. For example, run:

```
$ ipmitool -U ADMIN -P ADMIN -p 9001 -H localhost mc info
```

13. From the BMC, run:

```
$ ipmitool -I ipmb mc info
```

14. Test that the BlueField can send requests to the BMC. Run:

```
$ ipmitool mc info
```

6.2.3 Retrieving Data from BlueField Via OOB/ConnectX Interfaces

It is possible for the external host to retrieve IPMI data via the OOB interface or the ConnectX interfaces.

To do that, set the network interface address properly in progconf. For example, if the OOB ip address is 192.168.101.2, edit the OOB_IP variable in the `/etc/ipmi/progconf` file as follows:

```
root@localhost:~# cat /etc/ipmi/progconf
SUPPORT_IPMB="NONE"
LOOP_PERIOD=3
BF_FAMILY=$(/usr/bin/bffamily | tr -d '[:space:]')
OOB_IP="192.168.101.2"
```

Then reboot or restart the ipmi service as follows:

```
systemctl restart mlx_ipmid
```

6.2.4 BlueField Retrieving Data From BMC Via IPMB

BlueField has 2 IPMB modes. It can be used as a responder but also as a requester.

- Responder Mode
  When used as a responder, the BlueField receives IPMB request messages from the BMC on SMBus 2. It then, processes the message and sends a response back to the BMC. In this case, the BlueField needs to load the ipmb_dev_int driver.

  ```
  BMC (requester) ----IPMB/SMBus 2----> BlueField (responder)
  ```

- Requester Mode
  When used as a requester, the BlueField sends IPMB request messages to the BMC via SMBus 2. The BMC then, processes the request and sends a message back to the BlueField. So the BlueField needs to load the ipmb_host driver when the BMC is up. If the BMC is not up, ipmb_host will fail to load because it has to execute a handshake with the other end before loading.

  ```
  BlueField (requester) ----IPMB/SMBus 2----> BMC (responder)
  ```

Both modes are enabled automatically at boot time on Yocto.

Once the set_emu_param.service is started, it will try to load the ipmb_host drivers. If the BMC is down or not responsive when BlueField tries to load the ipmb_host driver, the latter will not load successfully. In that case, make sure the BMC is up and operational, and run the following from BlueField's console:

```
echo 0x1011 > /sys/bus/i2c/devices/i2c-2/delete_device
rmmod ipmb_host
```

The set_emu_param.service script will try to load the driver again.

6.2.4.1 BlueField and BMC I²C Addresses on BlueField Reference Platform

6.2.4.1.1 BlueField in Responder Mode

Device	I²C Address
BlueField ipmb_dev_int	0x30
BMC ipmb_host	0x20

6.2.4.1.2 BlueField in Requester Mode

Device	I²C Address
BlueField ipmb_host	0x11
BMC ipmb_dev_int	0x10

6.2.5 Changing I²C Addresses

To use a different BlueField or BMC I²C address, you must make changes to the following files' variables.

Filename Path	Parameter Change
/usr/bin/set_emu_param.sh	The ipmb_dev_int and ipmb_host drivers are registered at the following I²C addresses: IPMB_DEV_INT_ADD=<BlueField I²C Address 1> IPMB_HOST_ADD=<BlueField I²C Address 2> These addresses must be different from one another. Otherwise, one of the drives will fail to register. To change the BMC I²C address: IPMB_HOST_CLIENTADDR=<BMC I²C Address> <I²C address> must be equal to: 0x1000+<7-bit I²C address>

6.3 Logging

6.3.1 RShim Logging

RShim logging uses an internal 1KB HW buffer to track booting progress and record important messages. It is written by the NVIDIA® BlueField® networking platform's (DPU or SuperNIC) Arm cores and is displayed by the RShim driver from the USB/PCIe host machine. Starting in release 2.5.0, ATF has been enhanced to support the RShim logging.

The RShim log messages can be displayed described in the following:

1. Check the `DISPLAY_LEVEL` level in file `/dev/rshim0/misc`.

```
# cat /dev/rshim0/misc
DISPLAY_LEVEL   0 (0:basic, 1:advanced, 2:log)
…
```

2. Set `DISPLAY_LEVEL` to 2.

```
# echo "DISPLAY_LEVEL 2" > /dev/rshim0/misc
```

3. Log messages are displayed in the misc file.

```
# cat /dev/rshim0/misc
...
----------------------------------------
    Log Messages
----------------------------------------
 INFO[BL2]: start
 INFO[BL2]: no DDR on MSS0
 INFO[BL2]: calc DDR freq (clk_ref 53836948)
 INFO[BL2]: DDR POST passed
 INFO[BL2]: UEFI loaded
 INFO[BL31]: start
 INFO[BL31]: runtime
 INFO[UEFI]: eMMC init
 INFO[UEFI]: eMMC probed
 INFO[UEFI]: PCIe enum start
 INFO[UEFI]: PCIe enum end
```

This is an example output for BlueField-2.

The following table details the ATF/UEFI messages for BlueField-2 and BlueField-3:

Message	Explanation	Action
INFO[BL2]: start	BL2 started	Informational
INFO[BL2]: no DDR on MSS<N>	DDR is not detected on memory controller <N>	Informational (depends on device)
INFO[BL2]: calc DDR freq (clk_ref 156M, clk xxx)	DDR frequency is calculated based on reference clock 156M	Informational

Message	Explanation	Action
INFO[BL2]: calc DDR freq (clk_ref 100M, clk xxx)	DDR frequency is calculated based on reference clock 100M	Informational
INFO[BL2]: calc DDR freq (clk_ref xxxx)	DDR frequency is calculated based on reference clock xxxx	Informational
INFO[BL2]: DDR POST passed	BL2 DDR training passed	Informational
INFO[BL2]: UEFI loaded	UEFI image is loaded successfully in BL2	Informational
ERR[BL2]: DDR init fail on MSS<N>	DDR initialization failed on memory controller <N>	Informational (depends on device)
ERR[BL2]: image <N> bad CRC	Image with ID <N> is corrupted which will cause hang	Error message. Reset the device and retry. If problem persists, use a different image to retry it.
ERR[BL2]: DDR BIST failed	DDR BIST failed	Need to retry. Check the ATF booting message whether the detected OPN is correct or not, or whether it is supported by this image. If still fails, contact NVIDIA Support.
ERR[BL2]: DDR BIST Zero Mem failed	DDR BIST failed in the zero-memory operation	Power-cycle and retry. If the problem persists, contact your NVIDIA FAE.
WARN[BL2]: DDR frequency unsupported	DDR training is programmed with unsupported parameters	Check whether official FW is being used. If the problem persists, contact your NVIDIA FAE.
WARN[BL2]: DDR min-sys(unknown)	System type cannot be determined and boot as a minimal system	Check whether the OPN or PSID is supported. If the problem persists, contact your NVIDIA FAE.
WARN[BL2]: DDR min-sys(misconf)	System type misconfigured and boot as a minimal system	Check whether the OPN or PSID is supported. If the problem persists, contact your NVIDIA FAE.
Exception(BL2): syndrome = xxxxxxxx …	Exception in BL2 with syndrome code and register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
PANIC(BL2): PC = xxx …	Panic in BL2 with register dump. System will hung.	Capture the log, analyze the cause, and report to FAE if needed
ERR[BL2]: load/auth failed	Failed to load image (non-existent/ corrupted), or image authentication failed when secure boot is enabled	Try again with the correct and properly signed image
INFO[BL31]: start	BL31 started	Informational
INFO[BL31]: runtime	BL31 enters the runtime state. This is the latest BL31 message in normal booting process.	Informational

Message	Explanation	Action
Exception(BL31): syndrome = xxxxxxxx cptr_el3 xx daif xx …	Exception in BL31 with syndrome code and register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
PANIC(BL31): PC = xxx cptr_el3 xxx daif xxx …	Panic in BL31 with register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
INFO[UEFI]: eMMC init	eMMC driver is initialized	Informational and should always be printed
INFO[UEFI]: eMMC probed	eMMC card is initialized	Informational and should always be printed
ASSERT(UEFI): xxx : line-no	Runtime assert message in UEFI	Contact your NVIDIA FAE with this information. Usually the system is able to continue running.
INFO[UEFI]: PCIe enum start	PCIe enumeration start	Informational
INFO[UEFI]: PCIe enum end	PCIe enumeration end	Informational
ERR[UEFI]: Synchronous Exception at xxxxxx ERR[UEFI]: PC=xxxxxx ERR[UEFI]: PC=xxxxxx …	UEFI Exception with PC value reported	Contact your NVIDIA FAE with this information
ERR[BL2]: FW auth failed	Image authentication error	Wrong image has been used in the current secure lifecycle. Switch to the correct image.
ERR[BL2]: IROT cert sig not found	Failed to load attestation certificates	Contact your NVIDIA FAE with this information
ERR[BL2]: IROT cert sig not found	Failed to load certification update record Only relevant for certain BlueField-3 devices.	Contact your NVIDIA FAE with this information
INFO[BL31]: PSC Turtle Mode detected	PSC enters turtle mode BlueField-3 only.	Informational

Message	Explanation	Action
INFO[BL31]: In Enhanced NIC mode	BlueField-3 enters enhanced NIC mode	Informational
ERR[BL31]: (set_page err | pmbus_lsb err | mfr_vr_mc err | set_vout err)	BlueField-3 power management programming error. Usually happens when the I2C voltage regulator is not accessible.	Contact your NVIDIA FAE with this information
INFO [BL31]: MB8: VDD adjustment complete	BlueField-3 MainBin 8-core board VDD CPU adjustment	Informational
INFO [BL31]: VDD adjustment complete	BlueField-3 (non-8-core board) VDD CPU adjustment	Informational
INFO [BL31]: VDD: xxx mV	BlueField-3 VDD CPU voltage	Informational
ERR[BL31]: cannot access vr0 (or access vr1)	BlueField-3 unable to access voltage regulator (vr0 or vr1) via I2C	Contact your NVIDIA FAE with this information
ERR[BL31]: ATX power not detected!	ATX power is not connected	Contact your NVIDIA FAE with this information
INFO[BL31]: PTMERROR: Unknown OPN	Unable to detect the OPN on this device	Contact your NVIDIA FAE with this information
INFO[BL31]: PTMERROR: VR access error	Unable to access the voltage regulator on this device This also means power capping will be disabled.	Contact your NVIDIA FAE with this information
INFO[BL31]: power capping disabled	BlueField-3 power capping disabled	Informational
INFO[BL2]: boot mode (rshim | emmc | unknown)	Device boot mode (from external RShim or eMMC)	Informational
ERR[BL31]: ECC_SINGLE_ERROR_CNT=xxx	Single ECC error counter report	Contact your NVIDIA FAE with this information
ERR[BL31]: ECC_DOUBLE_ERROR_CNT=xxx	Double ECC error counter report	Contact your NVIDIA FAE with this information

Message	Explanation	Action
ERR[BL31]: mss0|mss1: C0|C1 single-bit ecc, IRQ[%d]	MSS (0 or 1) channel (0 or 1) single-bit ECC error interrupt #	Contact your NVIDIA FAE with this information
ERR[BL31]: mss0|mss1: C0|C1 Double bit ecc, IRQ[%d]	MSS (0 or 1) channel (0 or 1) double-bit ECC error interrupt #	Contact your NVIDIA FAE with this information
ERR[BL31]: Double-bit ECC also detected in same buffer	Single/double ECC error detected in the same buffer	Contact your NVIDIA FAE with this information
ERR[BL31]: l3c: double-bit ecc	L3c double-bit ECC error detected	Contact your NVIDIA FAE with this information
ERR[BL31]: MSS%d DIMM%d single|double bit ECC error detected	MSS DRAM single (or double) bit error detected	Contact your NVIDIA FAE with this information
ERR[BL31]: MSS%d SRAM double bit ECC error detected	MSS SRAM double bit ECC error detected	Contact your NVIDIA FAE with this information

6.3.2 IPMI Logging in UEFI

During UEFI boot, the BlueField sends IPMI SEL messages over IPMB to the BMC in order to track boot progress and report errors. The BMC must be in responder mode to receive the log messages.

6.3.2.1 SEL Record Format

The following table presents standard SEL records (record type = 0x02).

Byte(s)	Field	Description
1 2	Record ID	ID used to access SEL record. Filled in by the BMC. Is initialized to zero when coming from UEFI.
3	Record Type	Record type
4 5 6 7	Timestamp	Time when event was logged. Filled in by BMC. Is initialized to zero when coming from UEFI.
8 9	Generator ID	This value is always 0x0001 when coming from UEFI
10	EvM Rev	Event message format revision which provides the version of the standard a record is using. This value is 0x04 for all records generated by UEFI.
11	Sensor Type	Sensor type code for sensor that generated the event

Byte(s)	Field	Description
12	Sensor Number	Number of the sensor that generated the event. These numbers are arbitrarily chosen by the OEM.
13	Event Dir | Event Type	[7] – 0b0 = Assertion, 0b1 = Deassertion [6:0] – Event type code
14	Event Data 1	[7:6] – Type of data in Event Data 2 • 0b00 = unspecified • 0b10 = OEM code • 0b11 = Standard sensor-specific event extension [5:4] – Type of data in Event Data 3 • 0b00 = unspecified • 0b10 = OEM code • 0b11 = Standard sensor-specific event extension [3:0] – Event Offset; offers more detailed event categories. See *IPMI 2.0 Specification* section 29.7 for more detail.
15	Event Data 2	Data attached to the event. 0xFF for unspecified. Under some circumstances, this may be used to specify more detailed event categories.
16	Event Data 3	Data attached to the event. 0xFF for unspecified.

See *IPMI 2.0 Specification* section 32.1 for more detail.

6.3.2.2 Possible SEL Field Values

BlueField UEFI implements a subset of the IPMI 2.0 SEL standard. Each field may have the following values:

Field	Possible Values	Description of Values
Record Type	0x02	Standard SEL record. All events sent by UEFI are standard SEL records.
Event Dir	0b0	All events sent by UEFI are assertion events
Event Type	0x6F	Sensor-specific discrete events. Events with this type do not deviate from the standard.
Sensor Number	0x06	UEFI boot progress “sensor”. If value is 0x06, the sensor type will always be “System Firmware Progress” (0x0F).

For Sensor Type, Event Offset, and Event Data 1-3 definitions, see next table.

6.3.2.3 Event Definitions

Events are defined by a combination of Record Type, Event Type, Sensor Type, Event Offset (occupies Event Data 1), and sometimes Event Data 2 (referred to as the Event Extension if it defines sub-events).

The following tables list all currently implemented IPMI events (with Record Type = 0x02, Event Type = 0x6F).

⚠ Note that if an Event Data 2 or Event Data 3 value is not specified, it can be assumed to be Unspecified (0xFF).

Sensor Type	Sensor Type Code	Event Offset	Event Description, Actions to Take
System Firmware Progress	0x0F	0x00	System firmware error (POST error). Event Data 2: • 0x06 - Unrecoverable EMMC error. Contact NVIDIA support.
		0x02	System firmware progress: Informational message, no actions needed. Event Data 2: • 0x02 - Hard Disk Initialization. Logged when EMMC is initialized. • 0x04 - User Authentication. Logged when a user enters the correct UEFI password. This event is never logged if there is no UEFI password. • 0x07 - PCI Resource Configuration. Logged when PCI enumeration has started. • 0x0B - SMBus Initialization. This event is logged as soon as IPMB is configured in UEFI. • 0x13 - Starting OS Boot Process. Logged when Linux begins booting.

6.3.2.4 Reading IPMI SEL Log Messages

Log messages may be read from the BMC by issuing it a “Get SEL Entry Command” while it is in responder mode, either from a remote host, or from BlueField itself once it is booted.

```
$ ipmitool sel list
  7b | Pre-Init |0000691604| System Firmwares #0x06 | SMBus initialization | Asserted
  7c | Pre-Init |0000691604| System Firmwares #0x06 | Hard-disk initialization | Asserted
  7d | Pre-Init |0000691654| System Firmwares #0x06 | System boot initiated
$ ipmitool sel get 0x7d
SEL Record ID          : 007d
 Record Type           : 02
 Timestamp             : 01/09/1970 00:07:34
 Generator ID          : 0001
 EvM Revision          : 04
 Sensor Type           : System Firmwares
 Sensor Number         : 06
 Event Type            : Sensor-specific Discrete
 Event Direction       : Assertion Event
 Event Data            : c213ff
 Description           : System boot initiated
$ ipmitool sel clear
Clearing SEL.  Please allow a few seconds to erase.
$ ipmitool sel list
SEL has no entries
```

6.3.3 ACPI BERT Logging

ACPI boot error record table (BERT) is supported to log `last boot error` in Linux. Once Linux `printk` is enabled (e.g., by adding "`kernel.printk=8`" to `/etc/sysctl.conf`), it will try to report the errors automatically for last boot. The following is an example of such error reports:

```
[    2.635539] BERT: Error records from previous boot:
[    2.640434] [Hardware Error]: event severity: fatal
[    2.645331] [Hardware Error]:  Error 0, type: fatal
[    2.650236] [Hardware Error]:   section type: unknown, c6adf9e6-1108-4760-8827-003d059fe2e1
[    2.658606] [Hardware Error]:   section length: 0x35
```

```
[    2.663580] [Hardware Error]:   00000000: 52524520 4645555b 203a5d49 0a0d0a0d   ERR[UEFI]: ....
[    2.672284] [Hardware Error]:   00000010: 636e7953 6e6f7268 2073756f 65637845  Synchronous Exce
[    2.680987] [Hardware Error]:   00000020: 6f697470 7461206e 36783020 37313643  ption at 0x6C617
[    2.689696] [Hardware Error]:   00000030: 34 37 30 0d 0a
...
```

6.4 SoC Management Interface

The SoC management interface, formerly known as RShim, allows an external agent such as the host CPU or BMC to operate the DPU and monitor its operational state. This interface allows provisioning of the DPU, resetting Arm cores, and obtaining logs.

> ⚠ For instructions for Windows support, please refer to page "Windows Support".

6.4.1 Installation and Upgrade

Please refer to section Updating Repo Package on Host Side.

6.4.1.1 Configuration File

The configuration file for the SoC management interface is located at `/etc/rshim.conf` and includes the parameters listed in the table below.

Parameter	Default	Description
`BOOT_TIMEOUT`	150	Timeout value in seconds when pushing BFB while Arm side is not reading the boot stream.
`DROP_MODE`	0	Once set to 1, the RShim driver ignores all RShim writes and returns 0 for RShim read. This is used in cases such as during `FW_RESET` or bypassing the RShim PF to VM.
`PCIE_RESET_DELAY`	10	Delay in seconds for RShim over PCIe, which is added after chip reset and before pushing the boot stream.
`PCIE_INTR_POLL_INTERVAL`	10	Interrupt polling interval in seconds when running RShim over direct memory mapping.
`PCIE_HAS_VFIO`	1	Setting this parameter to 0 disallows RShim memory mapping via VFIO.
`PCIE_HAS_UIO`	1	Setting this parameter to 0 disallows RShim memory mapping via UIO.

> ⚠ Configuring RShim is optional. The default parameters are designed to support out-of-box deployment scenarios including multiple DPUs on a single host.

Users may control which RShim index maps to which device by following this procedure:

```
# Uncomment the 'rshim<N>' line to configure the mapping.
#
# device-name pci-device
rshim0      pcie-0000:21:00.2
rshim1      pcie-0000:81:00.2
```

```
#
# Ignored devices.
# Uncomment the 'none' line to configure the ignored devices.
#
#none         usb-1-1.4
#none         pcie-lf-0000:84:00.0
```

If any of these configurations are changed, then the SoC management interface must be restarted by running:

```
systemctl restart rshim
```

6.4.2 Host-side Interface Configuration

The NVIDIA® BlueField® DPU registers on the host OS a "DMA controller" for DPU management over PCIe. This can be verified by running the following:

```
#  lspci -d 15b3: | grep 'SoC Management Interface'
27:00.2 DMA controller: Mellanox Technologies MT42822 BlueField-2 SoC Management Interface (rev 01)
```

A special SoC management driver must be installed and run on the host OS to expose the various BlueField management interfaces to the OS. Currently, this driver is named RShim and is automatically installed as part of the DOCA installation. Refer to section "Install RShim on Host" for information on how to obtain and install the host-side SoC management interface driver.

When the SoC management interface driver runs properly on the host side, a sysfs device, `/dev/rshim0/*`, and a virtual Ethernet interface, `tmfifo_net0`, become available. The following is an example for querying the status of the SoC management interface driver on the host side:

```
# systemctl status rshim
● rshim.service - rshim driver for BlueField SoC
     Loaded: loaded (/lib/systemd/system/rshim.service; disabled; vendor preset: enabled)
     Active: active (running) since Tue 2022-05-31 14:57:07 IDT; 1 day 1h ago
       Docs: man:rshim(8)
    Process: 90322 ExecStart=/usr/sbin/rshim $OPTIONS (code=exited, status=0/SUCCESS)
   Main PID: 90323 (rshim)
      Tasks: 11 (limit: 76853)
     Memory: 3.3M
     CGroup: /system.slice/rshim.service
             90323 /usr/sbin/rshim
May 31 14:57:07 …  systemd[1]: Starting rshim driver for BlueField SoC...
May 31 14:57:07  … systemd[1]: Started rshim driver for BlueField SoC.
May 31 14:57:07  … rshim[90323]: Probing pcie-0000:a3:00.2(vfio)
May 31 14:57:07  … rshim[90323]: Create rshim pcie-0000:a3:00.2
May 31 14:57:07  … rshim[90323]: rshim pcie-0000:a3:00.2 enable
May 31 14:57:08  … rshim[90323]: rshim0 attached
```

If the SoC management interface driver device does not appear, refer to section "RShim Troubleshooting and How-Tos".

6.4.2.1 Virtual Ethernet Interface

On the host, the SoC management interface driver exposes a virtual Ethernet device called `tmfifo_net0`. This virtual Ethernet can be thought of as a peer-to-peer tunnel connection between the host and the DPU OS. The DPU OS also configures a similar device. The DPU OS's BFB images are customized to configure the DPU side of this connection with a preset IP of 192.168.100.2/30. It is up to the user to configure the host side of this connection. Configuration procedures vary for different OSs.

The following example configures the host side of `tmfifo_net0` with a static IP and enables IPv4-based communication to the DPU OS:

```
#  ip addr add dev tmfifo_net0 192.168.100.1/30
```

> ⚠ For instructions on persistent IP configuration of the tmfifo_net0 interface, refer to step "Assign a static IP to tmfifo_net0" under "Updating Repo Package on Host Side".

Logging in from the host to the DPU OS is now possible over the virtual Ethernet. For example:

```
ssh ubuntu@192.168.100.2
```

6.4.2.2 SoC Management Interface Driver Support for Multiple DPUs

Multiple DPUs may connect to the same host machine. When the SoC management interface driver is loaded and operating correctly, each BlueField device is expected to have its own device directory on sysfs, `/dev/rshim<N>`, and a virtual Ethernet device, `tmfifo_net<N>`.

> ⚠ **Important!**
>
> `<N>` correlates to the number of BlueField DPUs used where the SoC management interfaces of the first DPU is 0, incrementing by 1 for each added BlueField.

The following are some guidelines on how to set up the SoC management virtual Ethernet interfaces properly if multiple DPUs are installed in the host system.

There are two methods to manage multiple `tmfifo_net` interfaces on a Linux platform:

- Using a bridge, with all `tmfifo_net<N>` interfaces on the bridge - the bridge device bares a single IP address on the host while each DPU has unique IP in the same subnet as the bridge
- Directly over the individual `tmfifo_net<N>` - each interface has a unique subnet IP and each DPU has a corresponding IP per subnet

Whichever method is selected, the host-side `tmfifo_net` interfaces should have different MAC addresses, which can be:

- Configured using `ifconfig`. For example:

  ```
  $ ifconfig tmfifo_net0 192.168.100.1/24 hw ether 02:02:02:02:02:02
  ```

- Or saved in configuration via the `/udev/rules` as can be seen later in this section.

In addition, each Arm-side `tmfifo_net` interface must have a unique MAC and IP address configuration, as BlueField OS comes uniformly pre-configured with a generic MAC, and 192.168.100.2. The latter must be configured in each DPU manually or by DPU customization scripts during BlueField OS installation.

6.4.2.2.1 Multi-board Management Example

This example deals with two BlueField DPUs installed on the same server (the process is similar for more DPUs). The example assumes that the RShim package has been installed on the host server.

6.4.2.2.1.1 Configuring Management Interface on Host

> ⚠ This example is relevant for CentOS/RHEL operating systems only.

1. Create a `bf_tmfifo` interface under `/etc/sysconfig/network-scripts`. Run:

```
vim /etc/sysconfig/network-scripts/ifcfg-br_tmfifo
```

2. Inside `ifcfg-br_tmfifo`, insert the following content:

```
DEVICE="br_tmfifo"
BOOTPROTO="static"
IPADDR="192.168.100.1"
NETMASK="255.255.255.0"
ONBOOT="yes"
TYPE="Bridge"
```

3. Create a configuration file for the first BlueField DPU, `tmfifo_net0`. Run:

```
vim /etc/sysconfig/network-scripts/ifcfg-tmfifo_net0
```

4. Inside `ifcfg-tmfifo_net0`, insert the following content:

```
DEVICE=tmfifo_net0
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br_tmfifo
```

5. Create a configuration file for the second BlueField DPU, `tmfifo_net1`. Run:

```
DEVICE=tmfifo_net1
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br_tmfifo
```

6. Create the rules for the `tmfifo_net` interfaces. Run:

```
vim /etc/udev/rules.d/91-tmfifo_net.rules
```

7. Restart the network for the changes to take effect. Run:

```
# /etc/init.d/network restart
Restarting network (via systemctl):            [  OK  ]
```

6.4.2.2.1.2 Configuring BlueField DPU Side

BlueField DPUs arrive with the following factory default configurations for tmfifo_net0.

Address	Value
MAC	`00:1a:ca:ff:ff:01`
IP	`192.168.100.2`

Therefore, if you are working with more than one DPU, you must change the default MAC and IP addresses.

Updating RShim Network MAC Address

> ⚠ This procedure is relevant for Ubuntu/Debian (`sudo` needed), and CentOS BFBs. The procedure only affects the `tmfifo_net0` on the Arm side.

1. Use a Linux console application (e.g. screen or minicom) to log into each BlueField. For example:

   ```
   # sudo screen /dev/rshim<0|1>/console 115200
   ```

2. Create a configuration file for `tmfifo_net0` MAC address. Run:

   ```
   # sudo vi /etc/bf.cfg
   ```

3. Inside `bf.cfg` , insert the new MAC:

   ```
   NET_RSHIM_MAC=00:1a:ca:ff:ff:03
   ```

4. Apply the new MAC address. Run:

   ```
   sudo bfcfg
   ```

5. Repeat this procedure for the second BlueField DPU (using a different MAC address).

   > ℹ Arm must be rebooted for this configuration to take effect. It is recommended to update the IP address before you do that to avoid unnecessary reboots.

> ⚠ For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

Updating IP Address

For Ubuntu:

1. Access the file `50-cloud-init.yaml` and modify the tmfifo_net0 IP address:

   ```
   sudo vim /etc/netplan/50-cloud-init.yaml

                  tmfifo_net0:
                       addresses:
                       - 192.168.100.2/30    ===>>>    192.168.100.3/30
   ```

2. Reboot the Arm. Run:

```
sudo reboot
```

3. Repeat this procedure for the second BlueField DPU (using a different IP address).

> Arm must be rebooted for this configuration to take effect. It is recommended to update the MAC address before you do that to avoid unnecessary reboots.

For CentOS:

1. Access the file `ifcfg-tmfifo_net0`. Run:

```
# vim /etc/sysconfig/network-scripts/ifcfg-tmfifo_net0
```

2. Modify the value for `IPADDR`:

```
IPADDR=192.168.100.3
```

3. Reboot the Arm. Run:

```
reboot
```

   Or perform `netplan apply`.

4. Repeat this procedure for the second BlueField DPU (using a different IP address).

> Arm must be rebooted for this configuration to take effect. It is recommended to update the MAC address before you do that to avoid unnecessary reboots.

6.4.2.3 Permanently Changing Arm-side MAC Address

> It is assumed that the commands in this section are executed with root (or `sudo`) permission.

The default MAC address is `00:1a:ca:ff:ff:01`. It can be changed using `ifconfig` or by updating the UEFI variable as follows:

1. Log into Linux from the Arm console.
2. Run:

```
$ "ls /sys/firmware/efi/efivars".
```

3. If not mounted, run:

```
$ mount -t efivarfs none /sys/firmware/efi/efivars
$ chattr -i /sys/firmware/efi/efivars/RshimMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
$ printf "\x07\x00\x00\x00\x00\x1a\xca\xff\xff\x03" > \
  /sys/firmware/efi/efivars/RshimMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

The `printf` command sets the MAC address to `00:1a:ca:ff:ff:03` (the last six bytes of the `printf` value). Either reboot the device or reload the tmfifo driver for the change to take effect.

The MAC address can also be updated from the server host side while the Arm-side Linux is running:

1. Enable the configuration. Run:

```
# echo "DISPLAY_LEVEL 1" > /dev/rshim0/misc
```

2. Display the current setting. Run:

```
# cat /dev/rshim0/misc
DISPLAY_LEVEL   1 (0:basic, 1:advanced, 2:log)
BOOT_MODE       1 (0:rshim, 1:emmc, 2:emmc-boot-swap)
BOOT_TIMEOUT    300 (seconds)
DROP_MODE       0 (0:normal, 1:drop)
SW_RESET        0 (1: reset)
DEV_NAME        pcie-0000:04:00.2
DEV_INFO        BlueField-2(Rev 1)
PEER_MAC        00:1a:ca:ff:ff:01 (rw)
PXE_ID          0x00000000 (rw)
VLAN_ID         0 0 (rw)
```

3. Modify the MAC address. Run:

```
$ echo "PEER_MAC  xx:xx:xx:xx:xx:xx" > /dev/rshim0/misc
```

For more information and an example of the script that covers multiple DPU installation and configuration, refer to section "Installing Full DOCA Image on Multiple DPUs" of the *NVIDIA DOCA Installation Guide*.

6.4.3 SoC Management Interface Features and Functionality

	Function	Command	Comments
1	Push BFB	`bfb-install -r rshim<N> -b <bfb> [-c bf.cfg]`	Using `bf.cfg` in the command is optional. For more details about `bf.cfg`, refer to section "DPU Configuration File".
2	Open console	`screen /dev/rshim<N>/console 115200` `minicom -D /dev/rshim<N>/console`	The `N` index depends on the number of DPUs in your setup. Use Linux's screen or minicom console applications to access the BlueField console.
3	Configure a virtual network interface	`ip addr add dev tmfifo_net<N> 192.168.100.1/30`	The `N` index depends on the number of DPUs in your setup. Refer to section "SoC Management Interface Driver Support for Multiple DPUs" for more information. The default IP address for the DPU is 192.168.100.2/30. The IP used in the command (192.168.100.1/30) is for example purposes only.
4	Log into the DPU	`ssh -6 user@fe80::21a:caff:feff:ff01%tmfifo_net<N>`	The `N` index depends on the number of DPUs in your setup. Refer to section "SoC Management Interface Driver Support for Multiple DPUs" for more information.
5	PXE boot over RShim	N/A	Please refer to section "Deploying BlueField Software Using BFB with PXE" for more information.

	Function	Command	Comments
6	Issue Arm software reset	`echo "SW_RESERT 1" > /dev/rshim<N>/misc`	
7	Expose log messages	N/A	For more information, please refer to section "Logging".

6.4.4 DPU Configuration File

The `bf.cfg` file contains configuration that can be pushed to customize the installation of the BFB.

Please see section "bf.cfg Parameters" for the `bf.cfg` file contents.

6.5 BlueField OOB Ethernet Interface

The BlueField OOB interface is a gigabit Ethernet interface which provides TCP/IP network connectivity to the Arm cores. This interface is named `oob_net0` and is intended to be used for management traffic (e.g., file transfer protocols, SSH, etc). The Linux driver that controls this interface is named `mlxbf_gige.ko`, and is automatically loaded upon boot. This interface can be configured and monitored using of standard tools (e.g., ifconfig, ethtool, etc). The OOB interface is subject to the following design limitations:

- Only supports 1Gb/s full-duplex setting
- Only supports GMII access to external PHY device
- Supports maximum packet size of 2KB (i.e., no support for jumbo frames)

The OOB interface can also be used for PXE boot. This OOB port is not a path for the BlueField boot stream. Any attempt to push a BFB to this port would not work. Refer to "How to use the UEFI boot menu" for more information about UEFI operations related to the OOB interface.

6.5.1 OOB Interface MAC Address

The MAC address to be used for the OOB port is burned into Arm-accessible UPVS EEPROM during the manufacturing process. This EEPROM device is different from the SPI Flash storage device used for the NIC firmware and associated NIC MACs/GUIDs. The value of the OOB MAC address is specific to each platform and is visible on the board-level sticker.

It is not recommended to reconfigure the MAC address from the MAC configured during manufacturing.

If there is a need to re-configure this MAC for any reason, follow these steps to configure a UEFI variable to hold new value for OOB MAC.:

The creation of an OOB MAC address UEFI variable will override the OOB MAC address defined in EEPROM, but the change can be reverted.

1. Log into Linux from the Arm console.
2. Issue the command `ls /sys/firmware/efi/efivars` to show whether efivarfs is mounted. If it is not mounted, run:

```
mount -t efivarfs none /sys/firmware/efi/efivars
```

3. Run:

```
chattr -i /sys/firmware/efi/efivars/OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

4. Set the MAC address to 00:1a:ca:ff:ff:03 (the last six bytes of the printf value).

```
printf "\x07\x00\x00\x00\x00\x1a\xca\xff\xff\x03" > /sys/firmware/efi/efivars/
OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

5. Reboot the device for the change to take effect.

To revert this change and go back to using the MAC as programmed during manufacturing, follow these steps:

1. Log into UEFI from the Arm console, go to "Boot Manager" then "EFI Internal Shell".
2. Delete the OOB MAC UEFI variable. Run:

```
dmpstore -d OobMacAddr
```

3. Reboot the device by running "reset" from UEFI.
4. Log into Linux from the Arm console.
5. Issue the command `ls /sys/firmware/efi/efivars` to show whether efivarfs is mounted. If it is not mounted, run:

```
mount -t efivarfs none /sys/firmware/efi/efivars
```

6. Run:

```
chattr -i /sys/firmware/efi/efivars/OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

7. Reconfigure the original MAC address burned by the manufacturer in the format `aa\bb\cc\dd\ee\ff`. Run:

```
printf "\x07\x00\x00\x00\x00\<original-MAC-address>" > /sys/firmware/efi/efivars/
OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

8. Reboot the device for the change to take effect.

6.5.2 Supported ethtool Options for OOB Interface

The Linux driver for the OOB port supports the handling of some basic ethtool requests: get driver info, get/set ring parameters, get registers, and get statistics.

To use the ethtool options available, use the following format:

```
$ ethtool [<option>] <interface>
```

Where `<option>` may be:

- `<no-argument>` - display interface link information
- `-i` - display driver general information
- `-S` - display driver statistics
- `-d` - dump driver register set
- `-g` - display driver ring information
- `-G` - configure driver ring(s)
- `-k` - display driver offload information
- `-a` - query the specified Ethernet device for pause parameter information
- `-r` - restart auto-negotiation on the specified Ethernet device if auto-negotiation is enabled

For example:

```
$ ethtool oob_net0
Settings for oob_net0:
        Supported ports: [ TP ]
        Supported link modes:   1000baseT/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  1000baseT/Full
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Link partner advertised link modes:  1000baseT/Full
        Link partner advertised pause frame use: Symmetric
        Link partner advertised auto-negotiation: Yes
        Link partner advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 3
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: Unknown
        Link detected: yes
```

```
$ ethtool -i oob_net0
driver: mlxbf_gige
version:
firmware-version:
expansion-rom-version:
bus-info: MLNXBF17:00
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: yes
supports-priv-flags: no
```

```
# Display statistics specific to BlueField-2 design (i.e. statistics that are not shown in the output of "ifconfig
oob0_net")
$ ethtool -S oob_net0
NIC statistics:
    hw_access_errors: 0
    tx_invalid_checksums: 0
    tx_small_frames: 1
    tx_index_errors: 0
    sw_config_errors: 0
    sw_access_errors: 0
    rx_truncate_errors: 0
    rx_mac_errors: 0
    rx_din_dropped_pkts: 0
    tx_fifo_full: 0
    rx_filter_passed_pkts: 5549
    rx_filter_discard_pkts: 4
```

6.5.3 IP Address Configuration for OOB Interface

The files that control IP interface configuration are specific to the Linux distribution. The udev rules file (`/etc/udev/rules.d/92-oob_net.rules`) that renames the OOB interface to `oob_net0` and is the same for Yocto, CentOS, and Ubuntu:

```
SUBSYSTEM=="net", ACTION=="add", DEVPATH=="/devices/platform/MLNXBF17:00/net/eth[0-9]", NAME="oob_net0"
```

The files that control IP interface configuration are slightly different for CentOS and Ubuntu:

- CentOS configuration of IP interface:
  - Configuration file for `oob_net0`: `/etc/sysconfig/network-scripts/ifcfg-oob_net0`
  - For example, use the following to enable DHCP:

    ```
    NAME="oob_net0"
    DEVICE="oob_net0"
    NM_CONTROLLED="yes"
    PEERDNS="yes"
    ONBOOT="yes"
    BOOTPROTO="dhcp"
    TYPE=Ethernet
    ```

  - For example, to configure static IP use the following:

    ```
    NAME="oob_net0"
    DEVICE="oob_net0"
    IPV6INIT="no"
    NM_CONTROLLED="no"
    PEERDNS="yes"
    ONBOOT="yes"
    BOOTPROTO="static"
    IPADDR="192.168.200.2"
    PREFIX=30
    GATEWAY="192.168.200.1"
    DNS1="192.168.200.1"
    TYPE=Ethernet
    ```

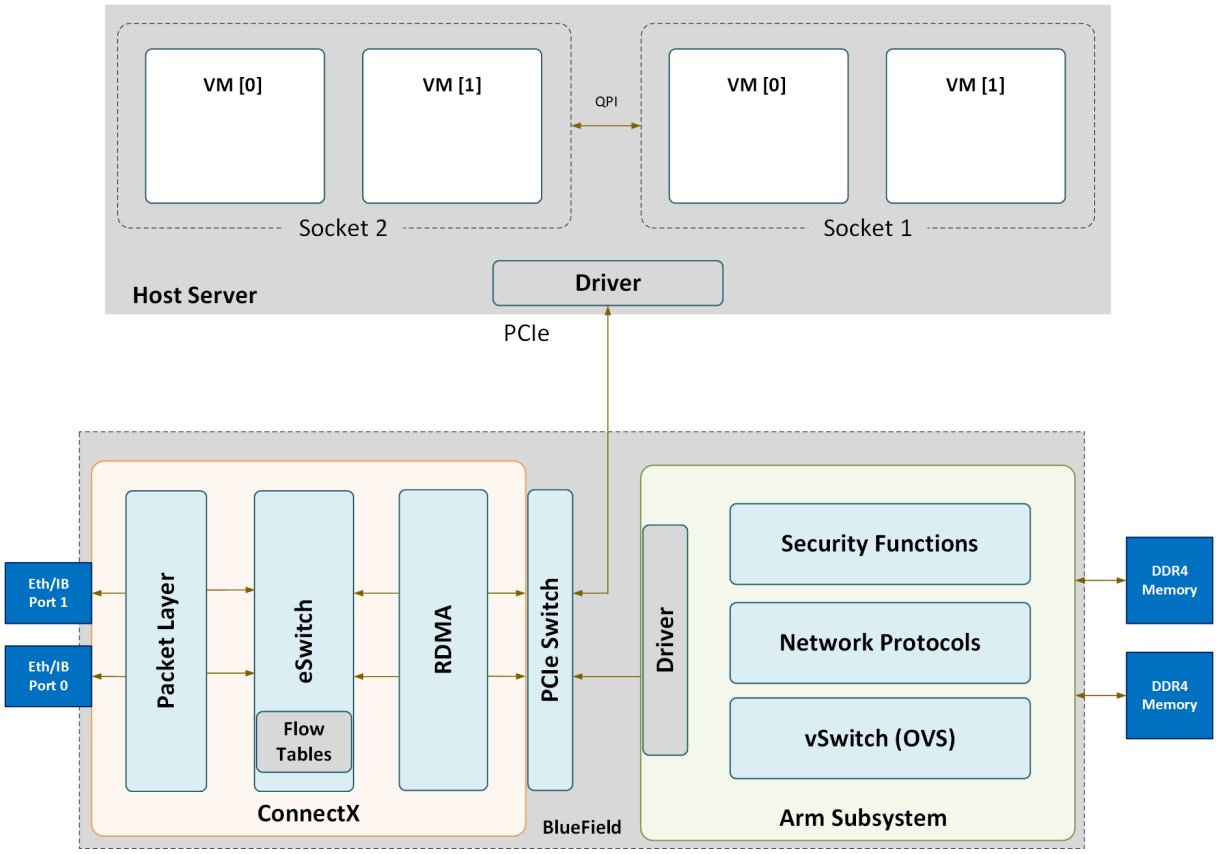
- For Ubuntu configuration of IP interface, please refer to section "Default Network Interface Configuration".

7 DPU Operation

The NVIDIA® BlueField® DPU® family delivers the flexibility to accelerate a range of applications while leveraging ConnectX-based network controllers hardware-based offloads with unmatched scalability, performance, and efficiency.

7.1 Functional Diagram

The following is a functional diagram of the NVIDIA® BlueField®-2 DPU.



For each BlueField DPU network port, there are 2 physical PCIe networking functions exposed:

- To the embedded Arm subsystem
- To the host over PCIe

> ⚠ Different functions have different default grace period values during which functions can recover from/handle a single fatal error:
>
> - ECPFs have a graceful period of 3 minutes
> - PFs have a graceful period of 1 minute
> - VFs/SFs have a graceful period of 30 seconds

The mlx5 drivers and their corresponding software stacks must be loaded on both hosts (Arm and the host server). The OS running on each one of the hosts would probe the drivers. BlueField-2 network interfaces are compatible with NVIDIA® ConnectX®-6 and higher. BlueField-3 network interfaces are compatible with ConnectX-7 and higher.

The same network drivers are used both for BlueField and the ConnectX NIC family.

7.2 Kernel Representors Model

> ⚠ This model is only applicable when the DPU is operating in DPU mode.

BlueField® DPU uses netdev representors to map each one of the host side physical and virtual functions.

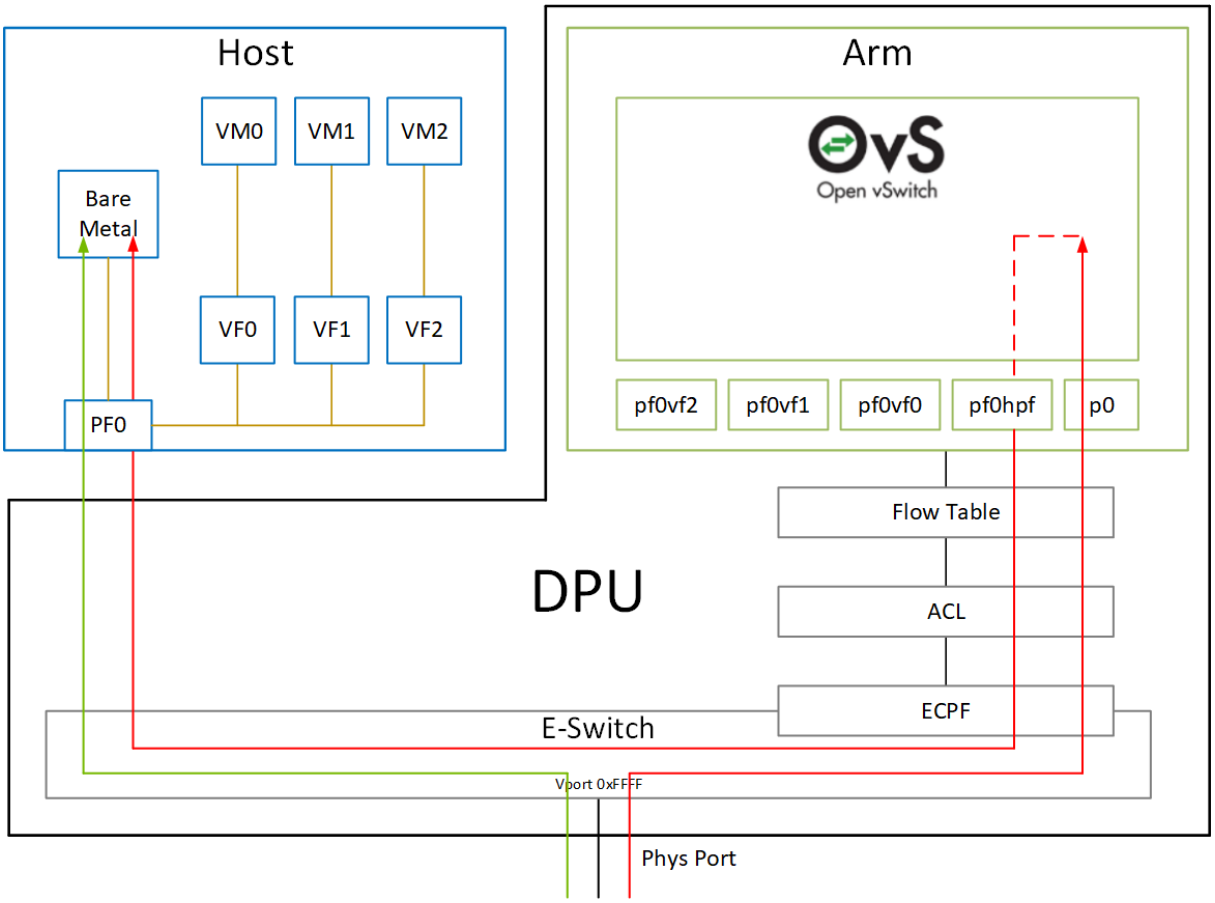
1. Serve as the tunnel to pass traffic for the virtual switch or application running on the Arm cores to the relevant PF or VF on the host side.
2. Serve as the channel to configure the embedded switch with rules to the corresponding represented function.

Those representors are used as the virtual ports being connected to OVS or any other virtual switch running on the Arm cores.

When operating in DPU mode, we see 2 representors for each one of the DPU's network ports: one for the uplink, and another one for the host side PF (the PF representor created even if the PF is not probed on the host side). For each one of the VFs created on the host side a corresponding representor would be created on the Arm side. The naming convention for the representors is as follows:

- Uplink representors: `p<port_number>`
- PF representors: `pf<port_number>hpf`
- VF representors: `pf<port_number>vf<function_number>`

The following diagram shows the mapping of between the PCIe functions exposed on the host side and the representors. For the sake of simplicity, a single port model (duplicated for the second port) is shown.



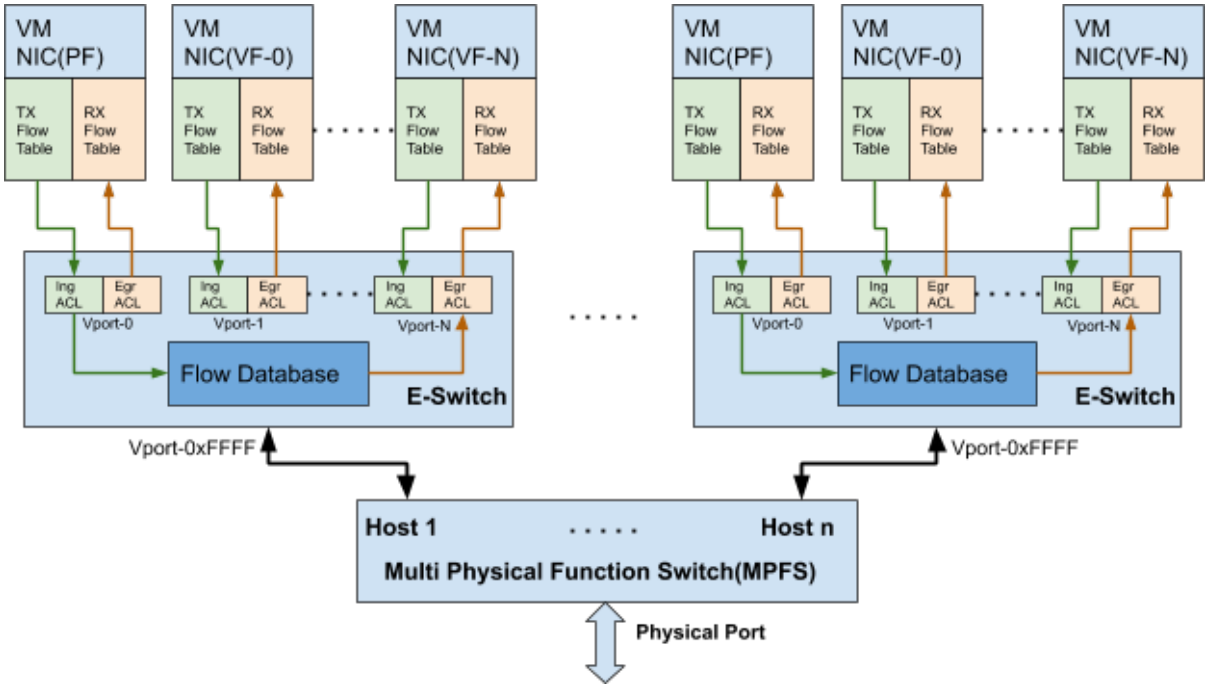
The red arrow demonstrates a packet flow through the representors, while the green arrow demonstrates the packet flow when steering rules are offloaded to the embedded switch. More details on that are available in the switch offload section.

> ⚠ The MTU of host functions (PF/VF) must be smaller than the MTUs of both the uplink and corresponding PF/VF representor. For example, if the host PF MTU is set to 9000, both uplink and PF representor must be set to above 9000.

7.3 Multi-Host

> ⚠ This is only applicable to NVIDIA® BlueField® networking platforms (DPU or SuperNIC) running on multi-host model.

In multi-host mode, each host interface can be divided into up to 4 independent PCIe interfaces. All interfaces would share the same physical port, and are managed by the same multi-physical function switch (MPFS). Each host would have its own e-switch and would control its own traffic.



7.3.1 Representors

Similar to Kernel Representors Model, each host here has an uplink representor, PF representor, and VF representors (if SR-IOV is enabled). There are 8 sets of representors (uplink/PF; see example code). For each host to work with OVS offload, the corresponding representors must be added to the OVS bridge.

```
139: p0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b2 brd ff:ff:ff:ff:ff:ff
140: p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b3 brd ff:ff:ff:ff:ff:ff
141: p2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b4 brd ff:ff:ff:ff:ff:ff
142: p3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b5 brd ff:ff:ff:ff:ff:ff
143: p4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b6 brd ff:ff:ff:ff:ff:ff
144: p5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b7 brd ff:ff:ff:ff:ff:ff
145: p6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b8 brd ff:ff:ff:ff:ff:ff
146: p7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b9 brd ff:ff:ff:ff:ff:ff
147: pf0hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP group default qlen 1000
    link/ether 86:c5:8a:b7:7c:84 brd ff:ff:ff:ff:ff:ff
148: pf1hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 6e:ea:1b:84:88:49 brd ff:ff:ff:ff:ff:ff
149: pf2hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 92:ec:99:cb:d7:23 brd ff:ff:ff:ff:ff:ff
150: pf3hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0e:0d:8e:03:2e:27 brd ff:ff:ff:ff:ff:ff
151: pf4hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 5e:42:af:05:67:93 brd ff:ff:ff:ff:ff:ff
152: pf5hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 96:e4:69:4c:b7:7f brd ff:ff:ff:ff:ff:ff
153: pf6hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 5e:67:33:c0:35:05 brd ff:ff:ff:ff:ff:ff
154: pf7hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 12:29:7d:56:07:3e brd ff:ff:ff:ff:ff:ff
```

The following is an example of adding all representors to OVS:

```
    Bridge armBr-3
```

```
        Port armBr-3
            Interface armBr-3
                type: internal
        Port p3
            Interface p3
        Port pf3hpf
            Interface pf3hpf
    Bridge armBr-2
        Port p2
            Interface p2
        Port pf2hpf
            Interface pf2hpf
        Port armBr-2
            Interface armBr-2
                type: internal
    Bridge armBr-5
        Port p5
            Interface p5
        Port pf5hpf
            Interface pf5hpf
        Port armBr-5
            Interface armBr-5
                type: internal
    Bridge armBr-7
        Port pf7hpf
            Interface pf7hpf
        Port armBr-7
            Interface armBr-7
                type: internal
        Port p7
            Interface p7
    Bridge armBr-0
        Port p0
            Interface p0
        Port armBr-0
            Interface armBr-0
                type: internal
        Port pf0hpf
            Interface pf0hpf
    Bridge armBr-4
        Port p4
            Interface p4
        Port pf4hpf
            Interface pf4hpf
        Port armBr-4
            Interface armBr-4
                type: internal
    Bridge armBr-1
        Port armBr-1
            Interface armBr-1
                type: internal
        Port p1
            Interface p1
        Port pf1hpf
            Interface pf1hpf
    Bridge armBr-6
        Port armBr-6
            Interface armBr-6
                type: internal
        Port p6
            Interface p6
        Port pf6hpf
            Interface pf6hpf
    ovs_version: "2.13.1"
```

For now, users can get the representor-to-host PF mapping by comparing the MAC address queried from host control on the Arm-side and PF MAC on the host-side. In the following example, the user knows p0 is the uplink representor for p6p1 as the MAC address is the same.

From Arm:

```
# cat /sys/class/net/p0/smart_nic/pf/config
MAC        : 0c:42:a1:70:1d:9a
MaxTxRate  : 0
State      : Up
```

From host:

```
# ip addr show p6p1
3: p6p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:9a brd ff:ff:ff:ff:ff:ff
```

The implicit mapping is as follows:

- PF0, PF1 = host controller 1
- PF2, PF3 = host controller 2
- PF4, PF5 = host controller 3

- PF6, PF7 = host controller 4

> ⚠ The maximum SF or VF count across all hosts is limited to 488 in total. The user can divide 488 VFs/SFs to single or multiple controllers as desired.

7.4 Virtual Switch on DPU

> ⚠ For general information on OVS offload using ASAP² direct, please refer to the MLNX_OFED documentation under OVS Offload Using ASAP² Direct.

> ⚠ ASAP^2 is only supported in Embedded (DPU) mode.

NVIDIA® BlueField® supports ASAP^2 technology. It utilizes the representors mentioned in the previous section. BlueField SW package includes OVS installation which already supports ASAP^2. The virtual switch running on the Arm cores allows us to pass all the traffic to and from the host functions through the Arm cores while performing all the operations supported by OVS. ASAP^2 allows us to offload the datapath by programming the NIC embedded switch and avoiding the need to pass every packet through the Arm cores. The control plane remains the same as working with standard OVS.

OVS bridges are created by default upon first boot of the DPU after BFB installation.

If manual configuration of the default settings for the OVS bridge is desired, run:

```
systemctl start openvswitch-switch.service
ovs-vsctl add-port ovsbr1 p0
ovs-vsctl add-port ovsbr1 pf0hpf
ovs-vsctl add-port ovsbr2 p1
ovs-vsctl add-port ovsbr2 pf1hpf
```

To verify successful bridging:

```
$ ovs-vsctl show
9f635bd1-a9fd-4f30-9bdc-b3fa21f8940a
    Bridge ovsbr2
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port p1
            Interface p1
        Port pf1sf0
            Interface en3f1pf1sf0
        Port pf1hpf
            Interface pf1hpf
    Bridge ovsbr1
        Port pf0hpf
            Interface pf0hpf
        Port p0
            Interface p0
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port pf0sf0
            Interface en3f0pf0sf0
    ovs_version: "2.14.1"
```

The host is now connected to the network.

7.4.1 Verifying Host Connection on Linux

When the DPU is connected to another DPU on another machine, manually assign IP addresses with the same subnet to both ends of the connection.

1. Assuming the link is connected to p3p1 on the other host, run:

```
$ ifconfig p3p1 192.168.200.1/24 up
```

2. On the host which the DPU is connected to, run:

```
$ ifconfig p4p2 192.168.200.2/24 up
```

3. Have one ping the other. This is an example of the DPU pinging the host:

```
$ ping 192.168.200.1
```

7.4.2 Verifying Connection from Host to BlueField

There are two SFs configured on the BlueFIeld-2 device, `enp3s0f0s0` and `enp3s0f1s0`, and their representors are part of the built-in bridge. These interfaces will get IP addresses from the DHCP server if it is present. Otherwise it is possible to configure IP address from the host. It is possible to access BlueField via the SF netdev interfaces.

For example:

1. Verify the default OVS configuration. Run:

```
# ovs-vsctl show
5668f9a6-6b93-49cf-a72a-14fd64b4c82b
    Bridge ovsbr1
        Port pf0hpf
            Interface pf0hpf
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port p0
            Interface p0
        Port en3f0pf0sf0
            Interface en3f0pf0sf0
    Bridge ovsbr2
        Port en3f1pf1sf0
            Interface en3f1pf1sf0
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port pf1hpf
            Interface pf1hpf
        Port p1
            Interface p1
    ovs_version: "2.14.1"
```

2. Verify whether the SF netdev received an IP address from the DHCP server. If not, assign a static IP. Run:

```
# ifconfig enp3s0f0s0
enp3s0f0s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.200.125  netmask 255.255.255.0  broadcast 192.168.200.255
        inet6 fe80::8e:bcff:fe36:19bc  prefixlen 64  scopeid 0x20<link>
        ether 02:8e:bc:36:19:bc  txqueuelen 1000  (Ethernet)
        RX packets 3730  bytes 1217558 (1.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 22  bytes 2220 (2.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3. Verify the connection of the configured IP address. Run:

```
# ping 192.168.200.25 -c 5
PING 192.168.200.25 (192.168.200.25) 56(84) bytes of data.
64 bytes from 192.168.200.25: icmp_seq=1 ttl=64 time=0.228 ms
64 bytes from 192.168.200.25: icmp_seq=2 ttl=64 time=0.175 ms
64 bytes from 192.168.200.25: icmp_seq=3 ttl=64 time=0.232 ms
64 bytes from 192.168.200.25: icmp_seq=4 ttl=64 time=0.174 ms
64 bytes from 192.168.200.25: icmp_seq=5 ttl=64 time=0.168 ms

--- 192.168.200.25 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 91ms
rtt min/avg/max/mdev = 0.168/0.195/0.232/0.031 ms
```

7.4.3 Verifying Host Connection on Windows

Set IP address on the Windows side for the RShim or Physical network adapter, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> New-NetIPAddress -InterfaceAlias "Ethernet 16" -IPAddress "192.168.100.1" -PrefixLength
22
```

To get the interface name, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> Get-NetAdapter
```

Output should give us the interface name that matches the description (e.g. NVIDIA BlueField Management Network Adapter).

```
Ethernet 2          NVIDIA ConnectX-4 Lx Ethernet Adapter        6 Not Present  24-8A-07-0D-E8-1D
Ethernet 6          NVIDIA ConnectX-4 Lx Ethernet Ad...#2       23 Not Present  24-8A-07-0D-E8-1C
Ethernet 16         NVIDIA BlueField Management Netw...#2       15 Up           CA-FE-01-CA-FE-02
```

Once IP address is set, Have one ping the other.

```
C:\Windows\system32>ping 192.168.100.2


Pinging 192.168.100.2 with 32 bytes of data:
Reply from 192.168.100.2: bytes=32 time=148ms TTL=64
Reply from 192.168.100.2: bytes=32 time=152ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
```

7.4.4 Enabling OVS HW Offloading

OVS HW offloading is set by default by the `/sbin/mlnx_bf_configure` script upon first boot after installation.

1. Enable TC offload on the relevant interfaces. Run:

   ```
   $ ethtool -K <PF> hw-tc-offload on
   ```

2. Enable the HW offload: run the following commands (after enabling the HW offload):

   ```
   $ ovs-vsctl set Open_vSwitch . Other_config:hw-offload=true
   ```

3. Restarting OVS is required for the configuration to apply:
   - For Ubuntu:

     ```
     $ systemctl restart openvswitch-switch
     ```

- For CentOS/RHEL:

```
$ systemctl restart openvswitch
```

To show OVS configuration:

```
$ ovs-dpctl show
system@ovs-system:
  lookups: hit:0 missed:0 lost:0
  flows: 0
  masks: hit:0 total:0 hit/pkt:0.00
  port 0: ovs-system (internal)
  port 1: armbr1 (internal)
  port 2: p0
  port 3: pf0hpf
  port 4: pf0vf0
  port 5: pf0vf1
  port 6: pf0vf2
```

At this point OVS would automatically try to offload all the rules.

To see all the rules that are added to the OVS datapath:

```
$ ovs-appctl dpctl/dump-flows
```

To see the rules that are offloaded to the HW:

```
$ ovs-appctl dpctl/dump-flows type=offloaded
```

7.4.5 Enabling OVS-DPDK Hardware Offload

1. Remove previously configured OVS bridges. Run:

   ```
   ovs-vsctl del-br <bridge-name>
   ```

   Issue the command `ovs-vsctl show` to see already configured OVS bridges.
2. Enable the Open vSwitch service. Run:

   ```
   systemctl start openvswitch
   ```
3. Configure huge pages:

   ```
   echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```
4. Enable hardware offload (disabled by default). Run:

   ```
   ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true
   ovs-vsctl --no-wait set Open_vSwitch . other_config:hw-offload=true
   ```
5. Configure the DPDK whitelist. Run:

   ```
   ovs-vsctl set Open_vSwitch . other_config:dpdk-extra="-w
   0000:03:00.0,representor=[0,65535],dv_flow_en=1,dv_xmeta_en=1,sys_mem_en=1"
   ```
6. Create OVS-DPDK bridge. Run:

   ```
   ovs-vsctl add-br br0-ovs -- set Bridge br0-ovs datapath_type=netdev -- br-set-external-id br0-ovs bridge-id
   br0-ovs -- set bridge br0-ovs fail-mode=standalone
   ```

7. Add PF to OVS. Run:

```
ovs-vsctl add-port br0-ovs p0 -- set Interface p0  type=dpdk options:dpdk-devargs=0000:03:00.0
```

8. Add representor to OVS. Run:

```
ovs-vsctl add-port br0-ovs pf0vf0 -- set Interface pf0vf0 type=dpdk options:dpdk-
devargs=0000:03:00.0,representor=[0]
ovs-vsctl add-port br0-ovs pf0hpf -- set Interface pf0hpf type=dpdk options:dpdk-
devargs=0000:03:00.0,representor=[65535]
```

9. Restart the Open vSwitch service. This step is required for HW offload changes to take effect.
    - For CentOS, run:

      ```
      systemctl restart openvswitch
      ```

    - For Debian/Ubuntu, run:

      ```
      systemctl restart openvswitch-switch
      ```

For a reference setup configuration for BlueField-2 devices, refer to the article "Configuring OVS-DPDK Offload with BlueField-2".

7.4.6 Configuring DPDK and Running TestPMD

1. Configure hugepages. Run:

```
echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

2. Run testpmd.
    - For Ubuntu/Debian:

      ```
      env LD_LIBRARY_PATH=/opt/mellanox/dpdk/lib/aarch64-linux-gnu /opt/mellanox/dpdk/bin/dpdk-testpmd -a
      03:00.0,representor=[0,65535] --socket-mem=1024 -- --total-num-mbufs=131000 -i
      ```

    - For CentOS:

      ```
      env LD_LIBRARY_PATH=/opt/mellanox/dpdk/lib64/ /opt/mellanox/dpdk/bin/dpdk-testpmd -a
      03:00.0,representor=[0,65535] --socket-mem=1024 -- --total-num-mbufs=131000 -i
      ```

For a detailed procedure with port display, refer to the article "Configuring DPDK and Running testpmd on BlueField-2".

7.4.7 Flow Statistics and Aging

The aging timeout of OVS is given in milliseconds and can be configured by running the following command:

```
$ ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

7.4.8 Connection Tracking Offload

This feature enables tracking connections and storing information about the state of these connections. When used with OVS, the DPU can offload connection tracking, so that traffic of established connections bypasses the kernel and goes directly to hardware.

Both source NAT (SNAT) and destination NAT (DNAT) are supported with connection tracking offload.

7.4.8.1 Configuring Connection Tracking Offload

This section provides an example of configuring OVS to offload all IP connections of host PF0.

1. Enable OVS HW offloading.
2. Create OVS connection tracking bridge. Run:

```
$ ovs-vsctl add-br ctBr
```

3. Add p0 and pf0hpf to the bridge. Run:

```
$ ovs-vsctl add-port ctBr p0
$ ovs-vsctl add-port ctBr pf0hpf
```

4. Configure ARP packets to behave normally. Packets which do not comply are routed to table1. Run:

```
$ ovs-ofctl add-flow ctBr "table=0,arp,action=normal"
$ ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1)"
```

5. Configure RoCEv2 packets to behave normally. RoCEv2 packets follow UDP port 4791 and a different source port in each direction of the connection. RoCE traffic is not supported by CT. In order to run RoCE from the host add the following line before `ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1)"`:

```
$ ovs-ofctl add-flow ctBr table=0,udp,tp_dst=4791,action=normal
```

   This rule allows RoCEv2 UDP packets to skip connection tracking rules.
6. Configure the new established flows to be admitted to the connection tracking bridge and to then behave normally. Run:

```
$ ovs-ofctl add-flow ctBr "table=1,priority=1,ip,ct_state=+trk+new,action=ct(commit),normal"
```

7. Set already established flows to behave normally. Run:

```
$ ovs-ofctl add-flow ctBr "table=1,priority=1,ip,ct_state=+trk+est,action=normal"
```

7.4.8.2 Connection Tracking With NAT

This section provides an example of configuring OVS to offload all IP connections of host PF0, and performing source network address translation (SNAT). The server host sends traffic via source IP from 2.2.2.1 to 1.1.1.2 on another host. Arm performs SNAT and changes the source IP to 1.1.1.16. Note that static ARP or route table must be configured to find that route.

1. Configure untracked IP packets to do nat. Run:

```
ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1,nat)"
```

2. Configure new established flows to do SNAT, and change source IP to 1.1.1.16. Run:

```
ovs-ofctl add-flow ctBr "table=1,in_port=pf0hpf,ip,ct_state=+trk+new,action=ct(commit,nat(src=1.1.1.16)),
p0"
```

3. Configure already established flows act normal. Run:

```
ovs-ofctl add-flow ctBr "table=1,ip,ct_state=+trk+est,action=normal"
```

Conntrack shows the connection with SNAT applied. Run `conntrack -L` for Ubuntu 22.04 kernel or `cat /proc/net/nf_conntrack` for older kernel versions. Example output:

```
ipv4     2 tcp      6 src=2.2.2.1 dst=1.1.1.2 sport=34541 dport=5001 src=1.1.1.2 dst=1.1.1.16 sport=5001
dport=34541 [OFFLOAD] mark=0 zone=1 use=3
```

7.4.8.3 Querying Connection Tracking Offload Status

Start traffic on PF0 from the server host (e.g., iperf) with an external network. Note that only established connections can be offloaded. TCP should have already finished the handshake, UDP should have gotten the reply.

> ⚠ ICMP is not currently supported.

To check if specific connections are offloaded from Arm, run `conntrack -L` for Ubuntu 22.04 kernel or `cat /proc/net/nf_conntrack` for older kernel versions.

The following is example output of offloaded TCP connection:

```
ipv4     2 tcp      6 src=1.1.1.2 dst=1.1.1.3 sport=51888 dport=5001 src=1.1.1.3 dst=1.1.1.2 sport=5001 dport=51888
[HW_OFFLOAD] mark=0 zone=0 use=3
```

7.4.8.4 Performance Tune Based on Traffic Pattern

Offloaded flows (including connection tracking) are added to virtual switch FDB flow tables. FDB tables have a set of flow groups. Each flow group saves the same traffic pattern flows. For example, for connection tracking offloaded flow, TCP and UDP are different traffic patterns which end up in two different flow groups.

A flow group has a limited size to save flow entries. By default, the driver has 4 big FDB flow groups. Each of these big flow groups can save at most 4000000/(4+1)=800k different 5-tuple flow entries. For scenarios with more than 4 traffic patterns, the driver provides a module parameter (`num_of_groups`) to allow customization and performance tune.

> ⚠ The size of each big flow groups can be calculated according to formula: size = 4000000/(num_of_groups+1)

To change the number of big FDB flow groups, run:

```
$ echo <num_of_groups> > /sys/module/mlx5_core/parameters/num_of_groups
```

The change takes effect immediately if there is no flow inside the FDB table (no traffic running and all offloaded flows are aged out), and it can be dynamically changed without reloading the driver.

If there are residual offloaded flows when changing this parameter, then the new configuration only takes effect after all flows age out.

7.4.8.5 Connection Tracking Aging

Aside from the aging of OVS, connection tracking offload has its own aging mechanism with a default aging time of 30 seconds.

7.4.8.6 Maximum Tracked Connections

⚠ The maximum number for tracked offloaded connections is limited to 1M by default.

The OS has a default setting of maximum tracked connections which may be configured by running:

```
$ /sbin/sysctl -w net.netfilter.nf_conntrack_max=1000000
```

This changes the maximum tracked connections (both offloaded and non-offloaded) setting to 1 million.

The following option specifies the limit on the number of offloaded connections. For example:

```
# devlink dev param set pci/${pci_dev} name ct_max_offloaded_conns value $max cmode runtime
```

This value is set to 1 million by default from BlueFiled. Users may choose a different number by using the `devlink` command.

⚠ Make sure `net.netfilter.nf_conntrack_tcp_be_liberal=1` when using connection tracking.

7.4.9 Offloading VLANs

OVS enables VF traffic to be tagged by the virtual switch.

For the BlueField DPU, the OVS can add VLAN tag (VLAN push) to all the packets sent by a network interface running on the host (either PF or VF) and strip the VLAN tag (VLAN pop) from the traffic going from the wire to that interface. Here we operate in Virtual Switch Tagging (VST) mode. This means that the host/VM interface is unaware of the VLAN tagging. Those rules can also be offloaded to the HW embedded switch.

To configure OVS to push/pop VLAN you need to add the tag=$TAG section for the OVS command line that adds the representor ports. So if you want to tag all the traffic of VF0 with VLAN ID 52, you should use the following command when adding its representor to the bridge:

```
$ ovs-vsctl add-port armbr1 pf0vf0 tag=52
```

If the virtual port is already connected to the bridge prior to configuring VLAN, you would need to remove it first:

```
$ ovs-vsctl del-port pf0vf0
```

In this scenario all the traffic being sent by VF 0 will have the same VLAN tag. We could set a VLAN tag by flow when using the TC interface, this is explained in section "Using TC Interface to Configure Offload Rules".

7.4.10 VXLAN Tunneling Offload

VXLAN tunnels are created on the Arm side and attached to the OVS. VXLAN decapsulation/encapsulation behavior is similar to normal VXLAN behavior, including over `hw_offload=true`.

To allow VXLAN encapsulation, the uplink representor (`p0`) should have an MTU value at least 50 bytes greater than that of the host PF/VF. Please refer to "Configuring Uplink MTU" for more information.

7.4.10.1 Configuring VXLAN Tunnel

1. Consider `p0` to be the local VXLAN tunnel interface (or VTEP).

   To be consistent with the examples below, it is assumed that `p0` is configured with a 1.1.1.1 IPv4 address.

2. Remove `p0` from any OVS bridge.
3. Build a VXLAN tunnel over OVS arm-ovs. Run:

   ```
   ovs-vsctl add-br arm-ovs -- add-port arm-ovs vxlan11 -- set interface vxlan11 type=vxlan
   options:local_ip=1.1.1.1 options:remote_ip=1.1.1.2 options:key=100
   options:dst_port=4789
   ```

4. Connect any host representor (e.g., `pf0hpf`) for which VXLAN is desired to the same arm-ovs bridge.
5. Configure the MTU of the VTEP (`p0`) used by VXLAN to at least 50 bytes larger than the host representor's MTU.

At this point, the host is unaware of any VXLAN operations done by the DPU's OVS. If the remote end of the VXLAN tunnel is properly set, any network traffic traversing arm-ovs undergoes VXLAN encap/decap.

7.4.10.2 Querying OVS VXLAN hw_offload Rules

Run the following:

```
ovs-appctl dpctl/dump-flows type=offloaded
in_port(2),eth(src=ae:fd:f3:31:7e:7b,dst=a2:fb:09:85:84:48),eth_type(0x0800), packets:1, bytes:98, used:0.900s,
actions:set(tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,tp_dst=4789,flags(key))),3
```

```
tunnel(tun_id=0x64,src=1.1.1.2,dst=1.1.1.1,tp_dst=4789,flags(+key)),in_port(3),eth(src=a2:fb:09:85:84:48,dst=ae:fd:
f3:31:7e:7b),eth_type(0x0800), packets:75, bytes:7350, used:0.900s, actions:2
```

⚠ For the host PF, in order for VXLAN to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm (see section "Zero-trust Mode"). Run:

   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```

2. The MTU of the end points (`pf0hpf` in the example above) of the VXLAN tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the VXLAN headers. For example, you can set the MTU of P0 to 2000.

7.4.11 GRE Tunneling Offload

GRE tunnels are created on the Arm side and attached to the OVS. GRE decapsulation/encapsulation behavior is similar to normal GRE behavior, including over `hw_offload=true`.

To allow GRE encapsulation, the uplink representor (`p0`) should have an MTU value at least 50 bytes greater than that of the host PF/VF.

Please refer to "Configuring Uplink MTU" for more information.

7.4.11.1 Configuring GRE Tunnel

1. Consider `p0` to be the local GRE tunnel interface. `p0` should not be attached to any OVS bridge.

   ⚠ To be consistent with the examples below, it is assumed that `p0` is configured with a 1.1.1.1 IPv4 address and that the remote end of the tunnel is 1.1.1.2.

2. Create an OVS bridge, `br0`, with a GRE tunnel interface, `gre0`. Run:

   ```
   ovs-vsctl add-port br0 gre0 -- set interface gre0 type=gre options:local_ip=1.1.1.1
   options:remote_ip=1.1.1.2 options:key=100
   ```

3. Add `pf0hpf` to `br0`.

   ```
   ovs-vsctl add-port br0 pf0hpf
   ```

4. At this point, any network traffic sent or received by the host's PF0 undergoes GRE processing inside the BlueField OS.

7.4.11.2 Querying OVS GRE hw_offload Rules

Run the following:

```
ovs-appctl dpctl/dump-flows type=offloaded
recirc_id(0),in_port(3),eth(src=50:6b:4b:2f:0b:74,dst=de:d0:a3:63:0b:30),eth_type(0x0800),ipv4(frag=no),
packets:878, bytes:122802, used:0.440s,
actions:set(tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,ttl=64,flags(key))),2
```

```
tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,flags(+key)),recirc_id(0),in_port(2),eth(src=de:d0:a3:63:0b:30,dst=50:6b
:4b:2f:0b:74),eth_type(0x0800),ipv4(frag=no), packets:995, bytes:97510, used:0.440s, actions:3
```

For the host PF, in order for GRE to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm (see section "Zero-trust Mode"). Run:

   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```

2. The MTU of the end points (`pf0hpf` in the example above) of the GRE tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the GRE headers. For example, you can set the MTU of P0 to 2000.

7.4.12 GENEVE Tunneling Offload

GENEVE tunnels are created on the Arm side and attached to the OVS. GENEVE decapsulation/ encapsulation behavior is similar to normal GENEVE behavior, including over `hw_offload=true` .

To allow GENEVE encapsulation, the uplink representor (`p0`) must have an MTU value at least 50 bytes greater than that of the host PF/VF.

Please refer to "Configuring Uplink MTU" for more information.

7.4.12.1 Configuring GENEVE Tunnel

1. Consider `p0` to be the local GENEVE tunnel interface. `p0` should not be attached to any OVS bridge.
2. Create an OVS bridge, `br0` , with a GENEVE tunnel interface, `gnv0` . Run:

   ```
   ovs-vsctl add-port br0 gnv0 -- set interface gnv0 type=geneve options:local_ip=1.1.1.1
   options:remote_ip=1.1.1.2 options:key=100
   ```

3. Add `pf0hpf` to `br0` .

   ```
   ovs-vsctl add-port br0 pf0hpf
   ```

4. At this point, any network traffic sent or received by the host's PF0 undergoes GENEVE processing inside the BlueField OS.

Options are supported for GENEVE. For example, you may add option `0xea55` to tunnel metadata, run:

```
ovs-ofctl add-tlv-map geneve_br "{class=0xffff,type=0x0,len=4}->tun_metadata0"
ovs-ofctl add-flow geneve_br ip,actions="set_field:0xea55->tun_metadata0",normal
```

For the host PF, in order for GENEVE to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm (see section "Zero-trust Mode"). Run:

```
$ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
```

2. The MTU of the end points (`pf0hpf` in the example above) of the GENEVE tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the GENEVE headers. For example, you can set the MTU of P0 to 2000.

7.4.13 Using TC Interface to Configure Offload Rules

Offloading rules can also be added directly, and not just through OVS, using the tc utility. To enable TC ingress on all the representors (i.e., uplink, PF, and VF).

```
$ tc qdisc add dev p0 ingress
$ tc qdisc add dev pf0hpf ingress
$ tc qdisc add dev pf0vf0 ingress
```

7.4.13.1 L2 Rules Example

The rule below drops all packets matching the given source and destination MAC addresses.

```
$ tc filter add dev pf0hpf protocol ip parent ffff: \
                               flower \
                                           skip_sw \
                                           dst_mac e4:11:22:11:4a:51 \
                                           src_mac e4:11:22:11:4a:50 \
                                action drop
```

7.4.13.2 VLAN Rules Example

The following rules push VLAN ID 100 to packets sent from VF0 to the wire (and forward it through the uplink representor) and strip the VLAN when sending the packet to the VF.

```
$ tc filter add dev pf0vf0 protocol 802.1Q parent ffff: \
                                   flower \
                                             skip_sw \
                                             dst_mac e4:11:22:11:4a:51 \
                                             src_mac e4:11:22:11:4a:50 \
                                   action vlan push id 100 \
                                   action mirred egress redirect dev p0

$ tc filter add dev p0 protocol 802.1Q parent ffff: \
                                   flower \
                                             skip_sw \
                                             dst_mac e4:11:22:11:4a:51 \
                                             src_mac e4:11:22:11:4a:50 \
                                             vlan_ethtype 0x800 \
                                             vlan_id 100 \
                                             vlan_prio 0 \
                                 action vlan pop \
                                 action mirred egress redirect dev pf0vf0
```

7.4.13.3 VXLAN Encap/Decap Example

```
$ tc filter add dev pf0vf0 protocol 0x806 parent ffff: \
                                   flower \
                                             skip_sw \
                                             dst_mac e4:11:22:11:4a:51 \
                                             src_mac e4:11:22:11:4a:50 \
                                   action tunnel_key set \
                                   src_ip 20.1.12.1 \
                                   dst_ip 20.1.11.1 \
                                   id 100 \
                                   action mirred egress redirect dev vxlan100

$ tc filter add dev vxlan100 protocol 0x806 parent ffff: \
                                   flower \
```

```
                                skip_sw \
                                dst_mac e4:11:22:11:4a:51 \
                                src_mac e4:11:22:11:4a:50 \
                                enc_src_ip 20.1.11.1 \
                                enc_dst_ip 20.1.12.1 \
                                enc_key_id 100 \
                                enc_dst_port 4789 \
                        action tunnel_key unset \
                        action mirred egress redirect dev pf0vf0
```

7.4.14 VirtIO Acceleration Through Hardware vDPA

For configuration procedure, please refer to the MLNX_OFED documentation under OVS Offload Using ASAP² Direct > VirtIO Acceleration through Hardware vDPA.

7.5 Configuring Uplink MTU

To configure the port MTU while operating in DPU mode, users must restrict the external host port ownership by issuing the following command on the BlueField:

```
mlxprivhost -d /dev/mst/<pciconf0 device> r --disable_port_owner
```

Server cold reboot is required for this restriction to take effect.

Once the host is restricted, the port MTU is configured by changing the MTU of the uplink representor (`p0` or `p1`).

7.6 Link Aggregation

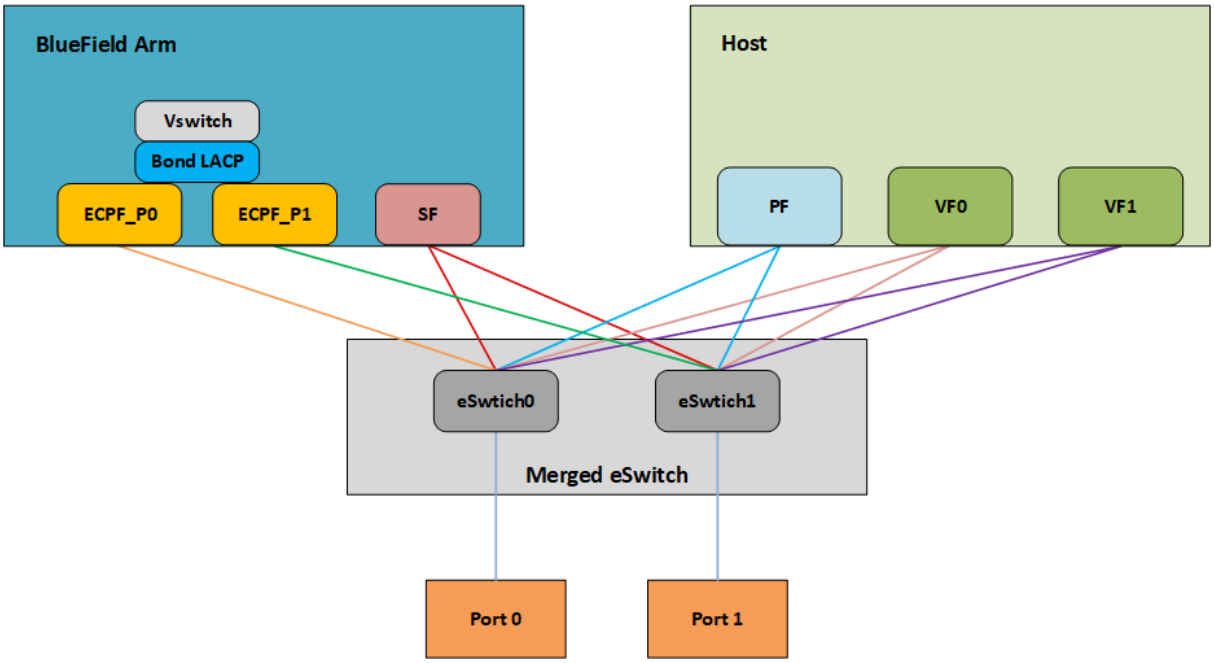
Network bonding enables combining two or more network interfaces into a single interface. It increases the network throughput, bandwidth and provides redundancy if one of the interfaces fails.

NVIDIA® BlueField® DPU has an option to configure network bonding on the Arm side in a manner transparent to the host. Under such configuration, the host would only see a single PF.

⚠ This functionality is supported when the DPU is set in embedded function ownership mode for both ports.

⚠ While LAG is being configured (starting with step 2 under section "LAG Configuration"), traffic cannot pass through the physical ports.

The diagram below describes this configuration:



7.6.1 LAG Modes

Two LAG modes are supported on BlueField:

- Queue Affinity mode
- Hash mode

7.6.1.1 Queue Affinity Mode

In this mode, packets are distributed according to the QPs.

1. To enable this mode, run:

   ```
   $ mlxconfig -d /dev/mst/<device-name> s LAG_RESOURCE_ALLOCATION=0
   ```

   Example device name: `mt41686_pciconf0`.

2. Add/edit the following field from `/etc/mellanox/mlnx-bf.conf` as follows:

   ```
   LAG_HASH_MODE="no"
   ```

3. Perform a graceful shutdown and system power cycle.

7.6.1.2 Hash Mode

In this mode, packets are distributed to ports according to the hash on packet headers.

⚠ For this mode, prerequisite steps 3 and 4 are not required.

1. To enable this mode, run:

```
$ mlxconfig -d /dev/mst/<device-name> s LAG_RESOURCE_ALLOCATION=1
```

   Example device name: `mt41686_pciconf0`.
2. Add/edit the following field from `/etc/mellanox/mlnx-bf.conf` as follows:

```
LAG_HASH_MODE="yes"
```

3. Perform a graceful shutdown and system power cycle.

7.6.2 Prerequisites

1. Set the LAG mode to work with.
2. (Optional) Hide the second PF on the host. Run:

```
$ mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=True NUM_OF_PF=1
```

   Example device name: `mt41686_pciconf0`.

   > ⚠ This step necessitates a system power cycle. If not performed, the second physical interface will still be visible to the host, but it will not be functional. This step has no effect on LAG configuration or functionality on the Arm side.

3. Delete any installed Scalable Functions (SFs) on the Arm side.
4. Stop the driver on the host side. Run:

```
$ systemctl stop openibd
```

5. The uplink interfaces (`p0` and `p1`) on the Arm side must be disconnected from any OVS bridge.

7.6.3 LAG Configuration

1. Create the bond interface. Run:

```
$ ip link add bond0 type bond
$ ip link set bond0 down
$ ip link set bond0 type bond miimon 100 mode 4 xmit_hash_policy layer3+4
```

   > ⚠ While LAG is being configured (starting with the next step), traffic cannot pass through the physical ports.

2. Subordinate both the uplink representors to the bond interface. Run:

```
$ ip link set p0 down
$ ip link set p1 down
$ ip link set p0 master bond0
$ ip link set p1 master bond0
```

3. Bring the interfaces up. Run:

```
$ ip link set p0 up
$ ip link set p1 up
$ ip link set bond0 up
```

The following is an example of LAG configuration in Ubuntu:

```
# cat /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source /etc/network/interfaces.d/*
auto lo
iface lo inet loopback
#p0
auto p0
iface p0 inet manual
        bond-master bond1
#
#p1
auto p1
iface p1 inet manual
        bond-master bond1
#bond1
auto bond1
iface bond1 inet static
        address 192.168.1.1
        netmask 255.255.0.0
        mtu 1500
        bond-mode 2
        bond-slaves p0 p1
        bond-miimon 100
        pre-up (sleep 2 && ifup p0) &
        pre-up (sleep 2 && ifup p1) &
```

As a result, only the first PF of the DPU would be available to the host side for networking and SR-IOV.

When in shared RQ mode (enabled by default), the uplink interfaces (`p0` and `p1`) must always stay enabled. Disabling them will break LAG support and VF-to-VF communication on same host.

For OVS configuration, the bond interface is the one that needs to be added to the OVS bridge (interfaces `p0` and `p1` should not be added). The PF representor for the first port (`pf0hpf`) of the LAG must be added to the OVS bridge. The PF representor for the second port (`pf1hpf`) would still be visible, but it should not be added to OVS bridge. Consider the following examples:

```
ovs-vsctl add-br bf-lag
ovs-vsctl add-port bf-lag bond0
ovs-vsctl add-port bf-lag pf0hpf
```

Trying to change bonding configuration in Queue Affinity mode (including bringing the subordinated interface up/down) while the host driver is loaded would cause FW syndrome and failure of the operation. Make sure to unload the host driver before altering DPU bonding configuration to avoid this.

When performing driver reload (`openibd restart`) or reboot, it is required to remove bond configuration and to reapply the configurations after the driver is fully up. Refer to steps 1-4 of "Removing LAG Configuration".

7.6.4 Removing LAG Configuration

1. If Queue Affinity mode LAG is configured (i.e., `LAG_RESOURCE_ALLOCATION=0`):

a. Delete any installed Scalable Functions (SFs) on the Arm side.
b. Stop driver (openibd) on the host side. Run:

```
systemctl stop openibd
```

2. Delete the LAG OVS bridge on the Arm side. Run:

```
ovs-vsctl del-br bf-lag
```

This allows for later restoration of OVS configuration for non-LAG networking.

3. Stop OVS service. Run:

```
systemctl stop openvswitch-switch.service
```

4. Run:

```
ip link set bond0 down
modprobe -rv bonding
```

As a result, both of the DPU's network interfaces would be available to the host side for networking and SR-IOV.

5. For the host to be able to use the DPU ports, make sure to attach the ECPF and host representor in an OVS bridge on the Arm side. Refer to "Virtual Switch on DPU" for instructions on how to perform this.
6. Revert from `HIDE_PORT2_PF`, on the Arm side. Run:

```
mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=False NUM_OF_PF=2
```

7. Restore default LAG settings in the DPU's firmware. Run:

```
mlxconfig -d /dev/mst/<device-name> s LAG_RESOURCE_ALLOCATION=DEVICE_DEFAULT
```

8. Delete the following line from `/etc/mellanox/mlnx-bf.conf` on the Arm side:

```
LAG_HASH_MODE=...
```

9. Perform a graceful shutdown and system power cycle.

7.6.5 LAG on Multi-host

Only LAG hash mode is supported with BlueField multi-host.

7.6.5.1 LAG Multi-host Prerequisites

1. Enable LAG hash mode.
2. Hide the second PF on the host. Run:

```
$ mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=True NUM_OF_PF=1
```

3. Make sure NVME emulation is disabled:

```
$ mlxconfig -d /dev/mst/<device-name> s NVME_EMULATION_ENABLE=0
```

Example device name: `mt41686_pciconf0`.

4. The uplink interfaces (`p0` and `p4`) on the Arm side, representing port0 and port1, must be disconnected from any OVS bridge. As a result, only the first PF of the DPU would be available to the host side for networking and SR-IOV.

7.6.5.2 LAG Configuration on Multi-host

1. Create the bond interface. Run:

```
$ ip link add bond0 type bond
$ ip link set bond0 down
$ ip link set bond0 type bond miimon 100 mode 4 xmit_hash_policy layer3+4
```

2. Subordinate both the uplink representors to the bond interface. Run:

```
$ ip link set p0 down
$ ip link set p4 down
$ ip link set p0 master bond0
$ ip link set p4 master bond0
```

3. Bring the interfaces up. Run:

```
$ ip link set p0 up
$ ip link set p4 up
$ ip link set bond0 up
```

4. For OVS configuration, the bond interface is the one that must be added to the OVS bridge (interfaces p0 and p4 should not be added). The PF representor, `pf0hpf`, must be added to the OVS bridge with the bond interface. The rest of the uplink representors must be added to another OVS bridge along with their PF representors. Consider the following examples:

```
ovs-vsctl add-br br-lag
ovs-vsctl add-port br-lag bond0
ovs-vsctl add-port br-lag pf0hpf
ovs-vsctl add-br br1
ovs-vsctl add-port br1 p1
ovs-vsctl add-port br1 pf1hpf
ovs-vsctl add-br br2
ovs-vsctl add-port br2 p2
ovs-vsctl add-port br2 pf2hpf
ovs-vsctl add-br br3
ovs-vsctl add-port br3 p3
ovs-vsctl add-port br3 pf3hpf
```

When performing driver reload (`openibd restart`) or reboot, you must remove bond configuration from NetworkManager, and to reapply the configurations after the driver is fully up.

7.6.5.3 Removing LAG Configuration on Multi-host

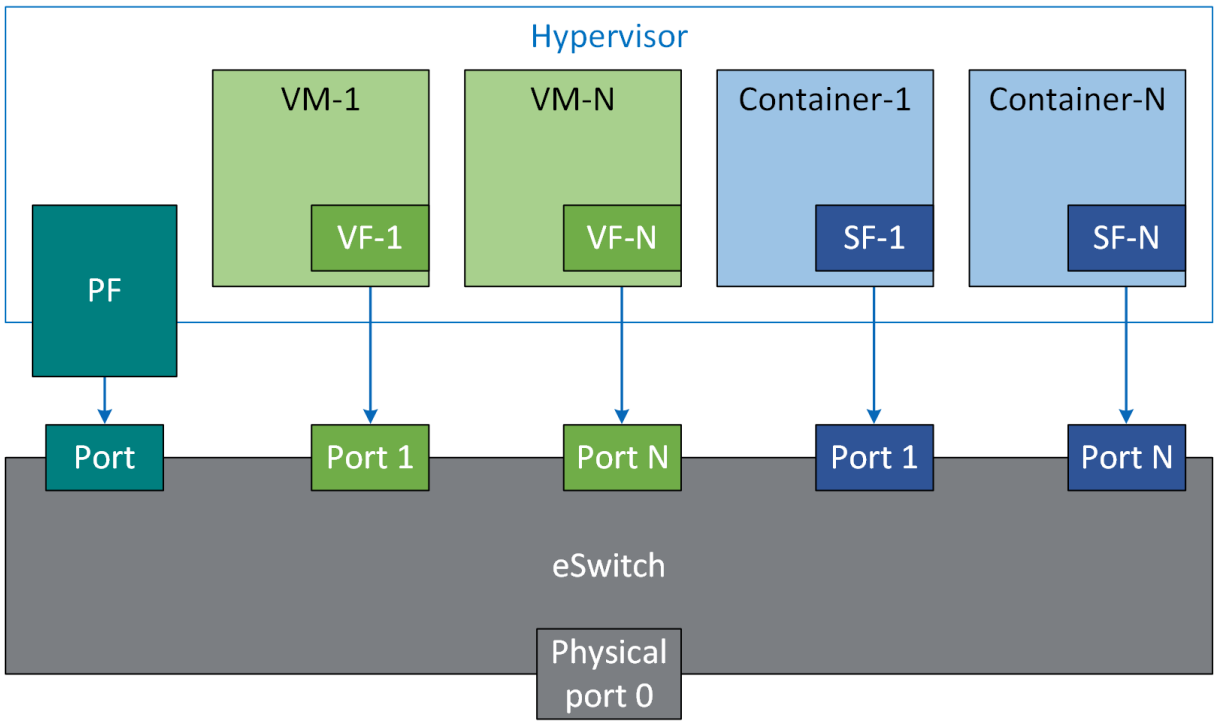
Refer to section "Removing LAG Configuration".

7.7 Scalable Functions

A scalable function (SF) is a lightweight function that has a parent PCIe function on which it is deployed. An mlx5 SF has its own function capabilities and its own resources. This means that an SF

has its own dedicated queues (txq, rxq, cq, eq) which are neither shared nor stolen from the parent PCIe function.

No special support is needed from system BIOS to use SFs. SFs co-exist with PCIe SR-IOV virtual functions. SFs do not require enabling PCIe SR-IOV.



7.7.1 Scalable Function Configuration

The following procedure offers a guide on using scalable functions with upstream Linux kernel.

7.7.1.1 Device Configuration

1. Make sure your firmware version supports SFs (20.30.1004 and above).
2. Enable SF support in device. Run:

```
$ mlxconfig -d 0000:03:00.0 s PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1 PF_TOTAL_SF=236 PF_SF_BAR_SIZE=10
```

3. Cold reboot the system for the configuration to take effect.

7.7.1.2 Mandatory Kernel Configuration on Host

Support for Linux kernel mlx5 SFs must be enabled as it is disabled by default.

The following two Kconfig flags must be enabled.

- MLX5_ESWITCH
- MLX5_SF

7.7.1.3 Software Control and Commands

SFs use a 4-step process as follows:

- Create
- Configure
- Deploy
- Use

SFs are managed using mlxdevm tool. It is located under directory `/opt/mellanox/iproute2/sbin/mlxdevm`.

1. Display the physical (i.e. uplink) port of the PF. Run:

```
$ devlink port show
pci/0000:03:00.0/65535: type eth netdev p0 flavour physical port 0 splittable false
```

2. Add an SF. Run:

```
$ mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum 0 sfnum 88
pci/0000:03:00.0/229409: type eth netdev eth0 flavour pcisf controller 0 pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached trust off
```

⚠ An added SF is still not usable for the end-user application. It can only be used after configuration and activation.

⚠ SF number ≥1000 is reserved for the virtio-net controller.

When an SF is added on the external controller (e.g. DPU) users must supply the controller number. In a single host DPU case, there is only one controller starting with controller number 1.
Example of adding an SF for PF0 of external controller 1:

```
$ mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum 0 sfnum 88 controller 1
pci/0000:03:00.0/32768: type eth netdev eth6 flavour pcisf controller 1 pfnum 0 sfnum 88 splittable false
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached
```

3. Show the newly added devlink port by its port index or its representor device.

```
$ mlxdevm port show en3f0pf0sf88
pci/0000:03:00.0/229409: type eth netdev en3f0pf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached trust off
```

Or:

```
$ mlxdevm port show pci/0000:03:00.0/229409
pci/0000:03:00.0/229409: type eth netdev en3f0pf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached trust off
```

4. Set the MAC address of the SF. Run:

```
$ mlxdevm port function set pci/0000:03:00.0/229409 hw_addr 00:00:00:00:88:88
```

5. Set SF as trusted (optional). Run:

```
$ mlxdevm port function set pci/0000:03:00.0/229409 trust on
pci/0000:03:00.0/229409: type eth netdev en3f0pf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:88:88 state inactive opstate detached trust on
```

A trusted function has additional privileges like the ability to update steering database.

6. Configure OVS. Run:

```
$ systemctl start openvswitch
$ ovs-vsctl add-br network1
$ ovs-vsctl add-port network1 ens3f0npf0sf88
$ ip link set dev ens3f0npf0sf88 up
```

7. Activate the SF. Run:

```
$ mlxdevm port function set pci/0000:03:00.0/229409 state active
```

Activating the SF results in creating an auxiliary device and initiating driver load sequence for netdevice, RDMA, and VDPA devices. Once the operational state is marked as attached, a driver is attached to this SF and device loading begins.

An application interested in using the SF netdevice and rdma device must monitor the RDMA and netdevices either through udev monitor or poll the sysfs hierarchy of the SF's auxiliary device.

8. By default, SF is attached to the configuration driver mlx5_core.sf_cfg. Users must unbind an SF from the configuration and bind it to the mlx5_core.sf driver to make use of it. Run:

```
 $ echo mlx5_core.sf.4 > /sys/bus/auxiliary/devices/mlx5_core.sf.4/driver/unbind
 $ echo mlx5_core.sf.4 > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind
```

9. View the new state of the SF. Run:

```
$ mlxdevm port show en3f0pf0sf88 -jp
{
    "port": {
        "pci/0000:03:00.0/229409": {
            "type": "eth",
            "netdev": "en3f0pf0sf88",
            "flavour": "pcisf",
            "controller": 0,
            "pfnum": 0,
            "sfnum": 88,
            "function": {
                "hw_addr": "00:00:00:00:88:88",
                "state": "active",
                "opstate": "detached",
                "trust": "on"
            }
        }
    }
}
```

10. View the auxiliary device of the SF. Run:

```
$ cat /sys/bus/auxiliary/devices/mlx5_core.sf.4/sfnum
88
```

There can be hundreds of auxiliary SF devices on the auxiliary bus. Each SF's auxiliary device contains a unique sfnum and PCI information.

11. View the parent PCI device of the SF. Run:

```
$ readlink /sys/bus/auxiliary/devices/mlx5_core.sf.1
../../../devices/pci0000:00/0000:00:00.0/0000:01:00.0/0000:02:00.0/0000:03:00.0/mlx5_core.sf.1
```

12. View the devlink instance of the SF device. Run:

```
$ devlink dev show
$ devlink dev show auxiliary/mlx5_core.sf.4
```

13. View the port and netdevice associated with the SF. Run:

```
$ devlink port show auxiliary/mlx5_core.sf.4/1
auxiliary/mlx5_core.sf.4/1: type eth netdev enp3s0f0s88 flavour virtual port 0 splittable false
```

14. View the RDMA device for the SF. Run:

```
$ rdma dev show
$ ls /sys/bus/auxiliary/devices/mlx5_core.sf.4/infiniband/
```

15. Deactivate SF. Run:

```
$ mlxdevm port function set pci/0000:03:00.0/229409 state inactive
```

Deactivating the SF triggers driver unload in the host system. Once SF is deactivated, its operational state changes to "detached". An orchestration system should poll for the operational state to be changed to "detached" before deleting the SF. This ensures a graceful hot-unplug.

16. Delete SF. Run:

```
$ mlxdevm port del pci/0000:03:00.0/229409
```

Finally, once the state is "inactive" and the operational state is "detached" the user can safely delete the SF. For faster provisioning, a user can reconfigure and active the SF again without deletion.

7.8 RDMA Stack Support on Host and Arm System

Full RDMA stack is pre-installed on the Arm Linux system. RDMA, whether RoCE or InfiniBand, is supported on NVIDIA® BlueField® networking platforms (DPUs or SuperNICs) in the configurations listed below.

7.8.1 Separate Host Mode

RoCE is supported from both the host and Arm system.

InfiniBand is supported on the host system.

7.8.2 Embedded CPU Mode

7.8.2.1 RDMA Support on Host

To use RoCE on a host system's PCIe PF, OVS hardware offloads must be enabled on the Arm system.

RoCE is not supported by connection tracking offload. Please refer to "Configuring Connection Tracking Offload" for a workaround for it.

7.8.2.2 RDMA Support on Arm

RoCE is unsupported on the Arm system on the PCIe PF. However, RoCE is fully supported using scalable function as explained under "Scalable Functions". Scalable functions are created by default, allowing RoCE traffic without further configuration.

InfiniBand is supported on the Arm system on the PCIe PF in this mode.

7.9 Controlling Host PF and VF Parameters

NVIDIA® BlueField® allows control over some of the networking parameters of the PFs and VFs running on the host side.

7.9.1 Setting Host PF and VF Default MAC Address

From the Arm, users may configure the MAC address of the physical function in the host. After sending the command, users must reload the NVIDIA driver in the host to see the newly configured MAC address. The MAC address goes back to the default value in the FW after system reboot.

Example:

```
$ echo "c4:8a:07:a5:29:59" > /sys/class/net/p0/smart_nic/pf/mac
$ echo "c4:8a:07:a5:29:61" > /sys/class/net/p0/smart_nic/vf0/mac
```

7.9.2 Setting Host PF and VF Link State

vPort state can be configured to Up, Down, or Follow. For example:

```
$ echo "Follow" > /sys/class/net/p0/smart_nic/pf/vport_state
```

7.9.3 Querying Configuration

To query the current configuration, run:

```
$ cat /sys/class/net/p0/smart_nic/pf/config
MAC         : e4:8b:01:a5:79:5e
MaxTxRate   : 0
State       : Follow
```

Zero signifies that the rate limit is unlimited.

7.9.4 Disabling Host Networking PFs

It is possible to not expose ConnectX networking functions to the host for users interested in using storage or VirtIO functions only. When this feature is enabled, the host PF representors (i.e. `pf0hpf` and `pf1hpf`) will not be seen on the Arm.

- Without a PF on the host, it is not possible to enable SR-IOV, so VF representors will not be seen on the Arm either
- Without PFs on the host, there can be no SFs on it

To disable host networking PFs, run:

```
mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=0
```

To reactivate host networking PFs:

- For single-port DPUs, run:

  ```
  mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=1
  ```

- For dual-port DPUs, run:

  ```
  mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=2
  ```

⚠ When there are no networking functions exposed on the host, the reactivation command must be run from the Arm.

⚠ Graceful shutdown and power cycle are required to apply configuration changes.

7.10 DPDK on BlueField DPU

Please refer to "Mellanox BlueField Board Support Package" in the DPDK documentation.

7.11 BlueField SNAP on DPU

NVIDIA® BlueField® SNAP (Software-defined Network Accelerated Processing) technology enables hardware-accelerated virtualization of NVMe storage. BlueField SNAP presents networked storage as a local NVMe SSD, emulating an NVMe drive on the PCIe bus. The host OS/Hypervisor makes use of its standard NVMe-driver unaware that the communication is terminated, not by a physical drive, but by the BlueField SNAP. Any logic may be applied to the data via the BlueField SNAP framework and transmitted over the network, on either Ethernet or InfiniBand protocol, to a storage target.

BlueField SNAP combines unique hardware-accelerated storage virtualization with the advanced networking and programmability capabilities of the DPU. BlueField SNAP together with the DPU enable a world of applications addressing storage and networking efficiency and performance.

To enable BlueField SNAP on your DPU, please contact NVIDIA Support.

7.12 Compression Acceleration

NVIDIA® BlueField® networking platforms (DPUs or SuperNIC) support high-speed compression acceleration. This feature allows the host to offload multiple compression/decompression jobs to BlueField.

Compress-class operations are supported in parallel to the net, vDPA, and RegEx class operations.

7.12.1 Configuring Compression Acceleration

The compression application can run either from the host or Arm.

For more information, please refer to:

- The DPDK community documentation about compression
- The mlx5 support documentation

7.13 Public Key Acceleration

NVIDIA® BlueField® networking platforms (DPUs or SuperNICs) incorporates several public key acceleration (PKA) engines to offload the processor of the Arm host, providing high-performance computation of PK algorithms. BlueField's PKA is useful for a wide range of security applications. It can assist with SSL acceleration, or a secure high-performance PK signature generator/checker and certificate related operations.

BlueField's PKA software libraries implement a simple, complete framework for crypto public key infrastructure (PKI) acceleration. It provides direct access to hardware resources from the user space and makes available a number of arithmetic operations—some basic (e.g., addition and multiplication), and some complex (e.g., modular exponentiation and modular inversion)—and high-level operations such as RSA, Diffie-Hallman, Elliptic Curve Cryptography, and the Federal Digital Signature Algorithm (DSA as documented in FIPS-186) public-private key systems.

7.13.1 PKA Prerequisites

- The BlueField PKA software is intended for BlueField products with HW accelerated crypto capabilities. To verify whether your BlueField chip has crypto capabilities, look for CPU flags `aes`, `sha1`, and `sha2` in the BlueField OS. For example:

```
# lscpu
...
Flags: fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
```

- BlueField bootloader must enable SMMU support to benefit from the full hardware and software capabilities. SMMU support may be enabled in UEFI menu through system configuration options.

7.13.2 PKA Use Cases

Some of the use cases for the BlueField PKA involve integrating OpenSSL software applications with BlueField's PKA hardware. The BlueField PKA dynamic engine for OpenSSL allows applications

integrated with OpenSSL (e.g., StrongSwan) to accomplish a variety of security-related goals and to accelerate the cryptographic processing with the BlueField PKA hardware. OpenSSL versions ≥1.0.0, ≤1.1.1, and 3.0.2 are supported.

> ⚠ With CentOS 7.6, only OpenSSL 1.1 (not 1.0) works with PKA engine and keygen.
> Use `openssl11` with PKA engine and keygen.

The engine supports the following operations:

- RSA
- DH
- DSA
- ECDSA
- ECDH
- Random number generation that is cryptographically secure.

Up to 4096-bit keys for RSA, DH, and DSA operations are supported. Elliptic Curve Cryptography support of (nist) prime curves for 160, 192, 224, 256, 384 and 521 bits.

For example, to sign a file using BlueField's PKA engine:

```
$ openssl dgst -engine pka -sha256 -sign <privatekey> -out <signature> <filename>
```

To verify the signature, execute:

```
$ openssl dgst -engine pka -sha256 -verify <publickey> -signature <signature> <filename>
```

For further details on BlueField PKA, please refer to "PKA Driver Design and Implementation Architecture Document" and/or "PKA Programming Guide". Directions and instructions on how to integrate the BlueField PKA software libraries are provided in the README files on the Mellanox PKA GitHub.

7.14 IPsec Functionality

7.14.1 Transparent IPsec Encryption and Decryption

BlueField DPU can offload IPsec operations transparently from the host CPU. This means that the host does not need to be aware that network traffic is encrypted before hitting the wire or decrypted after coming off the wire. IPsec operations can be run on the DPU in software on the Arm cores or in the accelerator block.

7.14.2 IPsec Hardware Offload: Crypto Offload

IPsec hardware crypto offload, also known as IPsec inline offload or IPsec aware offload, enables the user to offload IPsec crypto encryption and decryption operations to the hardware, leaving the encapsulation/decapsulation task to the software.

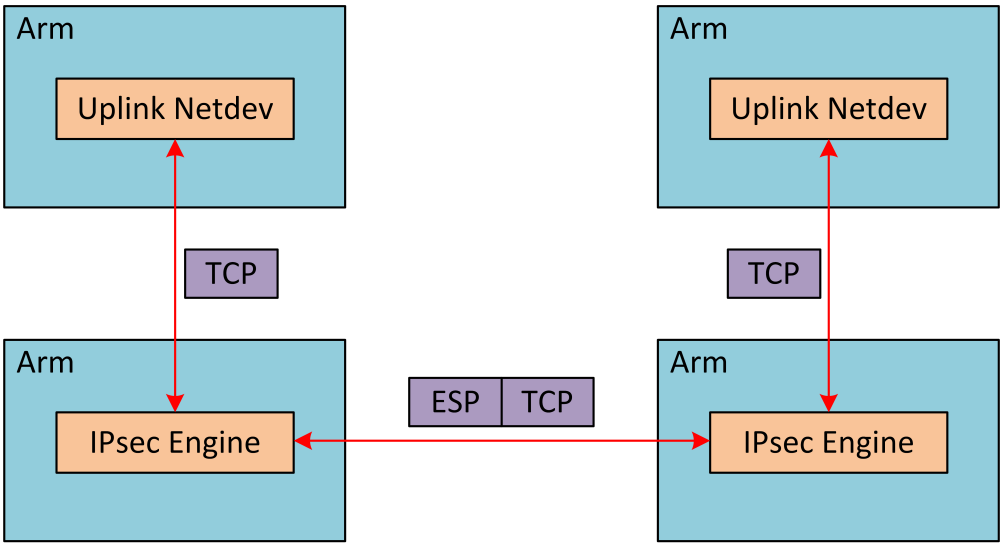
Please refer to the MLNX_OFED documentation under Features Overview and Configuration > Ethernet Network > IPsec Crypto Offload for more information on enabling and configuring this feature.

Please note that to use IPsec crypto offload with OVS, you must disable hardware offloads.

7.14.3 IPsec Hardware Offload: Packet Offload

> ⚠ IPSec packet offload is only supported on Ubuntu BlueField kernel 5.15

IPsec packet offload offloads both IPsec crypto and IPsec encapsulation to the hardware. IPsec packet offload is configured on the Arm via the uplink netdev. The following figure illustrates IPsec packet offload operation in hardware.



7.14.3.1 Enabling IPsec Packet Offload

Explicitly enable IPsec packet offload on the Arm cores before setting up offload-aware IPsec tunnels .

> ⚠ If an OVS VXLAN tunnel configuration already exists, stop `openvswitch` service prior to performing the steps below and restart the service afterwards.

Explicitly enable IPsec full offload on the Arm cores.

1. Set `IPSEC_FULL_OFFLOAD="yes"` in `/etc/mellanox/mlnx-bf.conf`.
2. Restart IB driver (rebooting also works). Run:

```
/etc/init.d/openibd restart
```

If `mlx-regex` is running:

a. Disable `mlx-regex` :

```
systemctl stop mlx-regex
```

b. Restart IB driver according to the command above.

c. Re-enable `mlx-regex` after the restart has finished:

```
systemctl restart mlx-regex
```

To revert IPsec full offload mode, redo the procedure from step 1, only difference is to set `IPSEC_FULL_OFFLOAD="no"` in `/etc/mellanox/mlnx-bf.conf` .

To use IPsec packet packet with strongSwan, refer to section "IPsec Packet Offload strongSwan Support".

To configure IPsec rules, please follow the instructions in MLNX_OFED documentation under Features Overview and Configuration > Ethernet Network > IPsec Crypto Offload > Configuring Security Associations for IPsec Offloads but, use "offload packet" to achieve IPsec Packet offload.

7.14.3.2 Configuring IPsec Rules with iproute2

If you are working directly with the `ip xfrm` tool, you must use the `/opt/mellanox/iproute2/sbin/ip` to benefit from IPsec packet offload support.

The following example configures IPsec packet offload rules with local address 192.168.1.64 and remote address 192.168.1.65:

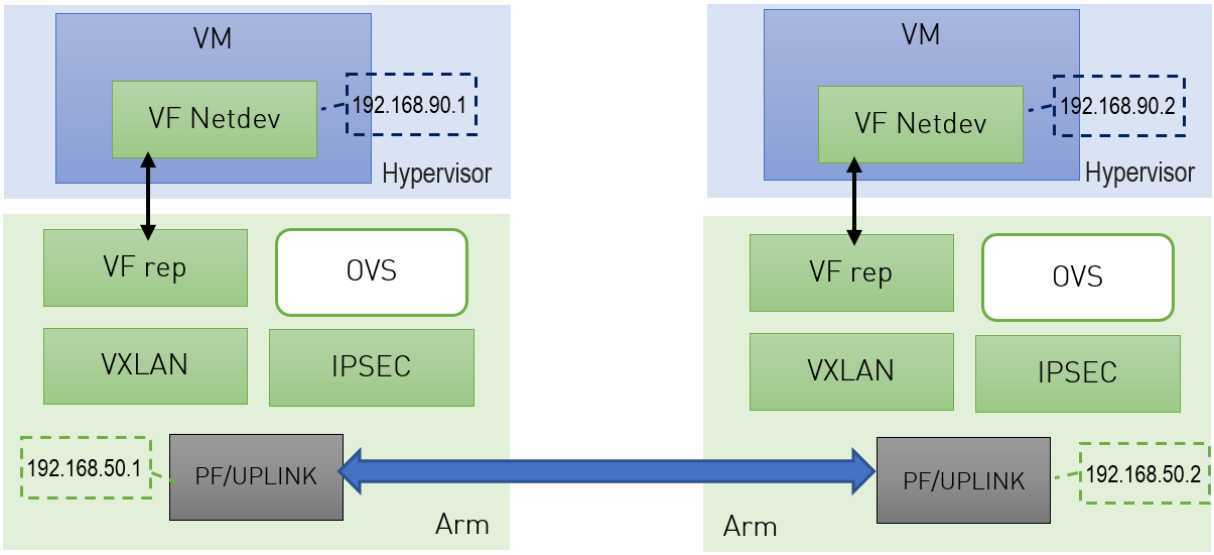
```
ip xfrm state add src 192.168.1.64/24 dst 192.168.1.65/24 proto esp spi 0x4834535d reqid 0x4834535d mode transport
aead 'rfc4106(gcm(aes))' 0xc57f6f084ebf8c6a71dd9a053c2e03b94c658a9bf00dd25780e73948931d10d08058a27c 128 offload
packet dev p0 dir out sel src 192.168.1.64 dst 192.168.1.65
ip xfrm state add src 192.168.1.65/24 dst 192.168.1.64/24 proto esp spi 0x2be60844 reqid 0x2be60844 mode transport
aead 'rfc4106(gcm(aes))' 0xacca06b66489011d3c1c21f1a36d925cf7449d3aeaa6fe534446c3a8f8bd5f5fdc266589 128 offload
packet dev p0 dir in sel src 192.168.1.65 dst 192.168.1.64
sudo ip xfrm policy add src 192.168.1.64 dst 192.168.1.65 dir out tmpl src 192.168.1.64/24 dst 192.168.1.65/24
proto esp reqid 0x4834535d mode transport
sudo ip xfrm policy add src 192.168.1.65 dst 192.168.1.64 dir in tmpl src 192.168.1.65/24 dst 192.168.1.64/24 proto
esp reqid 0x2be60844 mode transport
```

The numbers used by the `spi` , `reqid` , or `aead` algorithms are random. These same numbers are also used in the configuration of peer Arm. Do not confuse these numbers with source and destination IPs. The connection may fail if they are not consistent.

7.14.3.3 IPsec Packet Offload strongSwan Support

BlueField DPU supports configuring IPsec rules using strongSwan 5.9.10—appears as 5.9.10bf in the BFB which is based on upstream 5.9.10 version—which supports new fields in the `swanctl.conf` file.

The following figure illustrates an example with two BlueField DPUs, Left and Right, operating with a secured VXLAN channel.



Support for strongSwan IPsec packet HW offload requires using VXLAN together with IPSec as shown here.

1. Follow the procedure under section "Enabling IPsec Packet Offload".
2. Follow the procedure under section "VXLAN Tunneling Offload" to configure VXLAN on Arm.

   > ⚠ Make sure the MTU of the PF used by VXLAN is at least 50 bytes larger than VXLAN-REP MTU.

3. Enable tc offloading. Run:

   ```
   ethtool -K <PF> hw-tc-offload on
   ```

   > ⚠ Do not add the PF itself using "ovs-vsctl add-port" to the OVS.

7.14.3.3.1 Setting IPSec Packet Offload Using strongSwan

strongSwan configures IPSec HW packet offload using a new value added to its configuration file `swanctl.conf` (as of strongSwan version 5.9.10).

The file should be placed under "sysconfdir" which by default can be found at `/etc/swanctl/swanctl.conf`.

The terms Left (BFL) and Right (BFR) are used to identify the two nodes that communicate (corresponding with the figure under section "IPsec Packet Offload strongSwan Support").

In this example, 192.168.50.1 is used for the left PF uplink and 192.168.50.2 for the right PF uplink.

```
connections {
   BFL-BFR {
      local_addrs  = 192.168.50.1
      remote_addrs = 192.168.50.2

      local {
        auth = psk
        id = host1
      }
      remote {
        auth = psk
        id = host2
      }
        children {
         bf-out {
            local_ts = 192.168.50.1/24 [udp]
            remote_ts = 192.168.50.2/24 [udp/4789]
            esp_proposals = aes128gcm128-x25519-esn
            mode = transport
            policies_fwd_out = yes
            hw_offload = packet
         }
         bf-in {
            local_ts = 192.168.50.1/24 [udp/4789]
            remote_ts = 192.168.50.2/24 [udp]
            esp_proposals = aes128gcm128-x25519-esn
            mode = transport
            policies_fwd_out = yes
            hw_offload = packet
         }
      }
      version = 2
      mobike = no
      reauth_time = 0
      proposals = aes128-sha256-x25519
   }
}

secrets {
   ike-BF {
      id-host1 = host1
      id-host2 = host2
      secret = 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
   }
}
```

BFB installation will place two example swanctl.conf files for both Left and Right nodes (BFL.swanctl.conf and BFR.swanctl.conf respectively) in the strongSwan conf.d directory. Please move one of them manually to the other machine and edit it according to your configuration.

Note that:

- "`hw_offload = packet`" is responsible for configuring IPsec packet offload
- Packet offload support has been added to the existing `hw_offload` field and preserves backward compatibility.
  For your reference:

Value	Description
`no`	Do not configure HW offload
`crypto`	Configure crypto HW offload if supported by the kernel and hardware, fail if not supported
`yes`	Same as crypto (considered legacy)
`packet`	Configure packet HW offload if supported by the kernel and hardware, fail if not supported

Value	Description
`auto`	Configure packet HW offload if supported by the kernel and hardware, do not fail (perform fallback to crypto or no as necessary)

Whenever the value of `hw_offload` is changed, strongSwan configuration must be reloaded.

- `[udp/4789]` is crucial for instructing strongSwan to IPSec only VXLAN communication

Packet HW offload can only be done on what is streamed over VXLAN.

Mind the following limitations:

Field	Limitation
reauth_time	Ignored if set
rekey_time	Do not use. Ignored if set.
rekey_bytes	Do not use. Not supported and will fail if it is set.
rekey_packets	Use for rekeying

7.14.3.3.2 Running strongSwan Example

Notes:

- IPsec daemons are started by systemd `strongswan.service`, users must avoid using `strongswan-starter.service` as it is a legacy service and using both services at the same time leads to anomalous behavior
- Use `systemctl [start | stop | restart]` to control IPsec daemons through `strongswan.service`. For example, to restart, the command `systemctl restart strongswan.service` will effectively do the same thing as `ipsec restart`.

Do not use `ipsec` script to restart/stop/start.

If you are using the `ipsec` script, then, in order to restart or start the daemons, `openssl.cnf.orig` must be copied to openssl.cnf before performing `ipsec restart` or `ipsec start`. Then `openssl.cnf.mlnx` can be copied to `openssl.cnf` after restart or start. Failing to do so can result in errors since `openssl.cnf.mlnx` allows IPsec PK and RNG hardware offload via the OpenSSL plugin.

- On Ubuntu/Debian/Yocto, `openssl.cnf*` can be found under `/etc/ssl/`
- On CentOS, `openssl.cnf*` can be found under `/etc/pki/tls/`

- The strongSwan package installs `openssl.cnf` config files to enable hardware offload of PK and RNG operations via the OpenSSL plugin
- The OpenSSL dynamic engine is used to carry out the offload to hardware. OpenSSL dynamic engine ID is "pka".

Procedure:

1. Perform the following on Left and Right devices (corresponding with the figure under section "IPsec Packet Offload strongSwan Support").

```
# systemctl start strongswan.service
# swanctl --load-all
```

The following should appear.

```
Starting strongSwan 5.9.10bf IPsec [starter]...
no files found matching '/etc/ipsec.d/*.conf'
# deprecated keyword 'plutodebug' in config setup
# deprecated keyword 'virtual_private' in config setup
loaded ike secret 'ike-BF'
no authorities found, 0 unloaded
no pools found, 0 unloaded
loaded connection 'BFL-BFR'
successfully loaded 1 connections, 0 unloaded
```

2. Perform the actual connection on one side only (client, Left in this case).

```
# swanctl -i --child bf-in bf-out
```

The following should appear.

```
[IKE] initiating IKE_SA BFL-BFR[1] to 192.168.50.2
[ENC] generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N(HASH_ALG)
N(REDIR_SUP) ]
[NET] sending packet: from 192.168.50.1[500] to 192.168.50.2[500] (240 bytes)
[NET] received packet: from 192.168.50.2[500] to 192.168.50.1[500] (273 bytes)
[ENC] parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) CERTREQ N(FRAG_SUP) N(HASH_ALG)
N(CHDLESS_SUP) N(MULT_AUTH) ]
[CFG] selected proposal: IKE:AES_CBC_128/HMAC_SHA2_256_128/PRF_HMAC_SHA2_256/CURVE_25519
[IKE] received 1 cert requests for an unknown ca
[IKE] authentication of 'host1' (myself) with pre-shared key
[IKE] establishing CHILD_SA bf{1}
[ENC] generating IKE_AUTH request 1 [ IDi N(INIT_CONTACT) IDr AUTH N(USE_TRANSP) SA TSi TSr N(MULT_AUTH)
N(EAP_ONLY) N(MSG_ID_SYN_SUP) ]
[NET] sending packet: from 192.168.50.1[500] to 192.168.50.2[500] (256 bytes)
[NET] received packet: from 192.168.50.2[500] to 192.168.50.1[500] (224 bytes)
[ENC] parsed IKE_AUTH response 1 [ IDr AUTH N(USE_TRANSP) SA TSi TSr N(AUTH_LFT) ]
[IKE] authentication of 'host2' with pre-shared key successful
[IKE] IKE_SA BFL-BFR[1] established between 192.168.50.1[host1]...192.168.50.2[host2]
[IKE] scheduling reauthentication in 10027s
[IKE] maximum IKE_SA lifetime 11107s
[CFG] selected proposal: ESP:AES_GCM_16_128/NO_EXT_SEQ
[IKE] CHILD_SA bf{1} established with SPIs ce543905_i c60e98a2_o and TS 192.168.50.1/32 === 192.168.50.2/32
initiate completed successfully
```

You may now send encrypted data over the HOST VF interface (192.168.70.[1|2]) configured for VXLAN.

7.14.3.3.3 Building strongSwan

Do this only if you want to build your own BFB and would like to rebuild strongSwan.

1. Install dependencies mentioned here. libgmp-dev is missing from that list, so make sure to install that as well.
2. Git clone https://github.com/Mellanox/strongswan.git.
3. Git checkout BF-5.9.10. This branch is based on the official strongSwan 5.9.10 branch with added packaging and support for DOCA IPsec plugin (check the NVIDIA DOCA IPsec Security Gateway Application Guide for more information regarding the strongSwan DOCA plugin).

4. Run `autogen.sh` within the strongSwan repo.
5. Run the following:

```
configure --enable-openssl --disable-random --prefix=/usr/local --sysconfdir=/etc --enable-systemd
make
make install
```

Note:

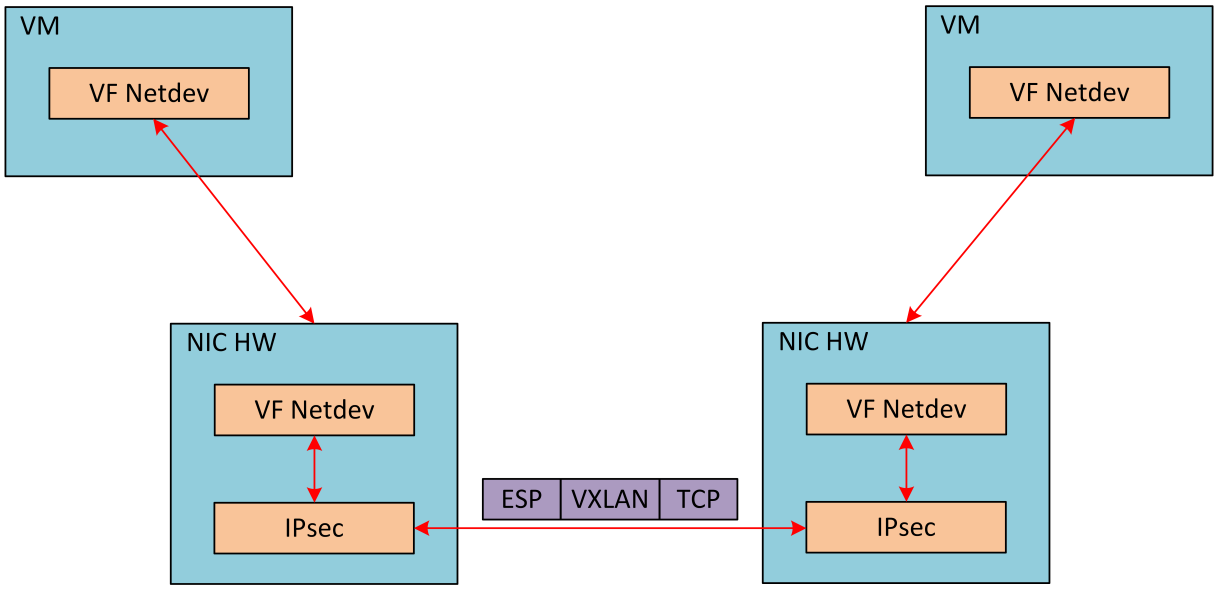
- `--enable-systemd` enables the systemd service for strongSwan present inside the GitHub repo (see step 3) at `init/systemd-starter/strongswan.service.in`.
- When building strongSwan on your own, the `openssl.cnf.mlnx` file, required for PK and RNG HW offload via OpenSSL plugin, is not installed. It must be copied over manually from github repo inside the openssl-conf directory. See section "Running Strongswan Example" for important notes.

> ⚠ The `openssl.cnf.mlnx` file references PKA engine shared objects. libpka (version 1.3 or later) and openssl (version 1.1.1) must be installed for this to work.

7.14.3.4 IPsec Packet Offload and OVS Offload

IPsec packet offload configuration works with and is transparent to OVS offload. This means all packets from OVS offload are encrypted by IPsec rules.

The following figure illustrates the interaction between IPsec packet offload and OVS VXLAN offload.



> ⚠ OVS offload and IPsec IPv6 do not work together.

7.14.4 OVS IPsec

To start the service, run:

```
systemctl start openvswitch-ipsec.service
```

Refer to section "Enabling IPsec Packet Offload" for information to prepare the IPsec packet offload environment.

7.14.4.1 Configuring IPsec Tunnel

For the sake of example, if you want to build an IPsec tunnel between two hosts with the following external IP addresses:

- `host1` - 1.1.1.1
- `host2` - 1.1.1.2

You have to first make sure `host1` and `host2` can ping each other via these external IPs.

This example will set up some variables on both hosts, set `ip1` and `ip2`:

```
# ip1=1.1.1.1
# ip2=1.1.1.2
REP=eth5
PF=p0
```

1. Set up OVS bridges in both hosts.
   a. On `Arm_1`:

   ```
   ovs-vsctl add-br ovs-br
   ovs-vsctl add-port ovs-br $REP
   ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
   ```

   b. On `Arm_2`:

   ```
   ovs-vsctl add-br ovs-br
   ovs-vsctl add-port ovs-br $REP
   ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
   ```

   ⚠ `Configuring other_config:hw-offload=true` sets IPsec packet offload. Setting it to `false` sets software IPsec. Make sure that IPsec devlink's mode is set back to `none` for software IPsec.

2. Set up IPsec tunnel. Three authentication methods are possible. Follow the steps relevant for the method that works best for your environment.

   ⚠ Do not try to use more than 1 authentication method.

   ⚠ After the IPsec tunnel is set up, strongSwan configuration will be automatically done.

3. Make sure the MTU of the PF used by tunnel is at least 50 bytes larger than VXLAN-REP MTU.

a. Disable host PF as the port owner from Arm (see section "Zero-trust Mode"). Run:

```
$ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
```

b. The MTU of the end points (`pf0hpf` in the example above) of the tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the tunnel headers. For example, you can set the MTU of P0 to 2000.

7.14.4.1.1 Authentication Methods

7.14.4.1.1.1 Using Pre-shared Key

⚠ The following example uses `tun type=gre` and `dst_port=1723` . Depending on your configuration, `tun type` can be `vxlan` or `geneve` with `dst_port` 4789 or 6081 respectively.

⚠ The following example uses `ovs-br` as the bridge name. However, this value can be any string you have chosen to create the bridge previously.

1. On `Arm_1` , run:

```
# ovs-vsctl add-port ovs-br tun -- \
            set interface tun type=gre \
                          options:local_ip=$ip1 \
                          options:remote_ip=$ip2 \
                          options:key=100 \
                          options:dst_port=1723 \
                          options:psk=swordfish
```

2. On `Arm_2` , run:

```
# ovs-vsctl add-port ovs-br tun -- \
            set interface tun type=gre \
                          options:local_ip=$ip2 \
                          options:remote_ip=$ip1 \
                          options:key=100 \
                          options:dst_port=1723 \
                          options:psk=swordfish
```

7.14.4.1.1.2 Using Self-signed Certificate

1. Generate self-signed certificates in both `host1` and `host2` , then copy the certificate of `host1` to `host2` , and the certificate of `host2` to `host1` .
2. Move both `host1-cert.pem` and `host2-cert.pem` to `/etc/swanctl/x509/` , if on Ubuntu, or `/etc/strongswan/swanctl/x509/` , if on CentOS.
3. Move the local private key to `/etc/swanctl/private` , if on Ubuntu, or `/etc/strongswan/swanctl/private` , if on CentOS. For example, for `host1` :

```
mv host1-privkey.pem /etc/swanctl/private
```

4. Set up OVS `other_config` on both sides.
    a. On `Arm_1` :

```
# ovs-vsctl set Open_vSwitch . other_config:certificate=/etc/swanctl/x509/host1-cert.pem \
  other_config:private_key=/etc/swanctl/private/host1-privkey.pem
```

b. On `Arm_2`:

```
# ovs-vsctl set Open_vSwitch . other_config:certificate=/etc/swanctl/x509/host2-cert.pem \
  other_config:private_key=/etc/swanctl/private/host2-privkey.pem
```

5. Set up the tunnel.
   a. On `Arm_1`:

```
# ovs-vsctl add-port ovs-br vxlanp0 -- set interface vxlanp0 type=vxlan options:local_ip=$ip1 \
  options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \
  options:remote_cert=/etc/swanctl/x509/host2-cert.pem
# service openvswitch-switch restart
```

   b. On `Arm_2`:

```
# ovs-vsctl add-port ovs-br vxlanp0 -- set interface vxlanp0 type=vxlan options:local_ip=$ip2 \
  options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \
  options:remote_cert=/etc/swanctl/x509/host1-cert.pem
# service openvswitch-switch restart
```

7.14.4.1.1.3 Using CA-signed Certificate

1. For this method, you need all the certificates and the requests to be in the same directory during the certificate generating and signing. This example refers to this directory as `certsworkspace`.
   a. On `Arm_1`:

```
# ovs-pki init --force
# cp /var/lib/openvswitch/pki/controllerca/cacert.pem <path_to>/certsworkspace
# ovs-pki req -u host1
# ovs-pki sign host1 switch
```

   b. On `Arm_2`:

```
# ovs-pki init --force
# cp /var/lib/openvswitch/pki/controllerca/cacert.pem <path_to>/certsworkspace
# ovs-pki req -u host2
# ovs-pki sign host2 switch
```

2. Move both `host1-cert.pem` and `host2-cert.pem` to `/etc/ swanctl/x509/`, if on Ubuntu, or `/etc/strongswan/swanctl/x509/`, if on CentOS.
3. Move the local private key to `/etc/swanctl/private`, if on Ubuntu, or `/etc/strongswan/swanctl/private`, if on CentOS. For example, for `host1`:

```
mv host1-privkey.pem /etc/swanctl/private
```

4. Copy `cacert.pem` to the `x509ca` directory under `/etc/swanctl/x509ca/`, if on Ubuntu, or `/etc/strongswan/swanctl/x509ca/`, if on CentOS.
5. Set up OVS `other_config` on both sides.
   a. On `Arm_1`:

```
# ovs-vsctl set Open_vSwitch . \
        other_config:certificate=/etc/strongswan/swanctl/x509/host1.pem \
        other_config:private_key=/etc/strongswan/swanctl/private/host1-privkey.pem \
        other_config:ca_cert=/etc/strongswan/swanctl/x509ca/cacert.pem
```

b. On `Arm_2`:

```
# ovs-vsctl set Open_vSwitch . \
        other_config:certificate=/etc/strongswan/swanctl/x509/host2.pem \
        other_config:private_key=/etc/strongswan/swanctl/private/host2-privkey.pem \
        other_config:ca_cert=/etc/strongswan/swanctl/x509ca/cacert.pem
```

6. Set up the tunnel:

a. On `Arm_1`:

```
# ovs-vsctl add-port ovs-br vxlanp0 -- set interface vxlanp0 type=vxlan options:local_ip=$ip1 \
options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \ options:remote_name=host2
 #service openvswitch-switch restart
```

b. On `Arm_2`:

```
# ovs-vsctl add-port ovs-br vxlanp0 -- set interface vxlanp0 type=vxlan options:local_ip=$ip2 \
options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \ options:remote_name=host1
#service openvswitch-switch restart
```

7.14.4.2 Ensuring IPsec is Configured

Use `/opt/mellanox/iproute2/sbin/ip xfrm state show`. You should be able to see IPsec states with the keyword `in mode packet`.

7.14.4.3 Troubleshooting

For troubleshooting information, refer to Open vSwitch's official documentation.

7.15 fTPM over OP-TEE

Security Disclaimer

The fTPM trusted application is signed with a development key intended solely for testing purposes and is not securely signed. This feature is strictly for testing and should not be used in any operational environment.

fTMP over OP-TEE is supported on BlueField-3 only at beta level.

The Trusted Computing Group (TCG) is responsible for the specifications governing the trusted platform module (TPM). In many systems, the TPM provides integrity measurements, health checks and authentication services.

Attributes of a TPM:

- Support for bulk (symmetric) encryption in the platform
- High quality random numbers
- Cryptographic services
- Protected persistent store for small amounts of data, sticky bits, monotonic counters, and extendible registers
- Protected pseudo-persistent store for unlimited amounts of keys and data

- Extensive choice of authorization methods to access protected keys and data
- Platform identities
- Support for platform privacy
- Signing and verifying digital signatures
- Certifying the properties of keys and data
- Auditing the usage of keys and data

With TPM 2.0., the TCG creates a library specification describing all the commands or features that could be implemented and may be necessary in servers, laptops, or embedded systems. Each platform can select the features needed and the level of security or assurance required. This flexibility allows the newest TPMs to be applied to many embedded applications.

Firmware TPM (fTPM) is implemented in protected software. The code runs on the main CPU so that a separate chip is not required. While running like any other program, the code is in a protected execution environment called a trusted execution environment (TEE) which is separate from the rest of the programs running on the CPU. By doing this, secrets (e.g., private keys perhaps needed by the TPM but should not be accessed by others) can be kept in the TEE creating a more secure environment.

> fTPM provides similar functionality to a chip-based TPM, but does not require extra hardware. It complies with the official TCG reference implementation of the TPM 2.0 specification. The source code of this implementation is located here.

> fTPM fully supports TPM2 Tools and the TCG TPM2 Software Stack (TSS).

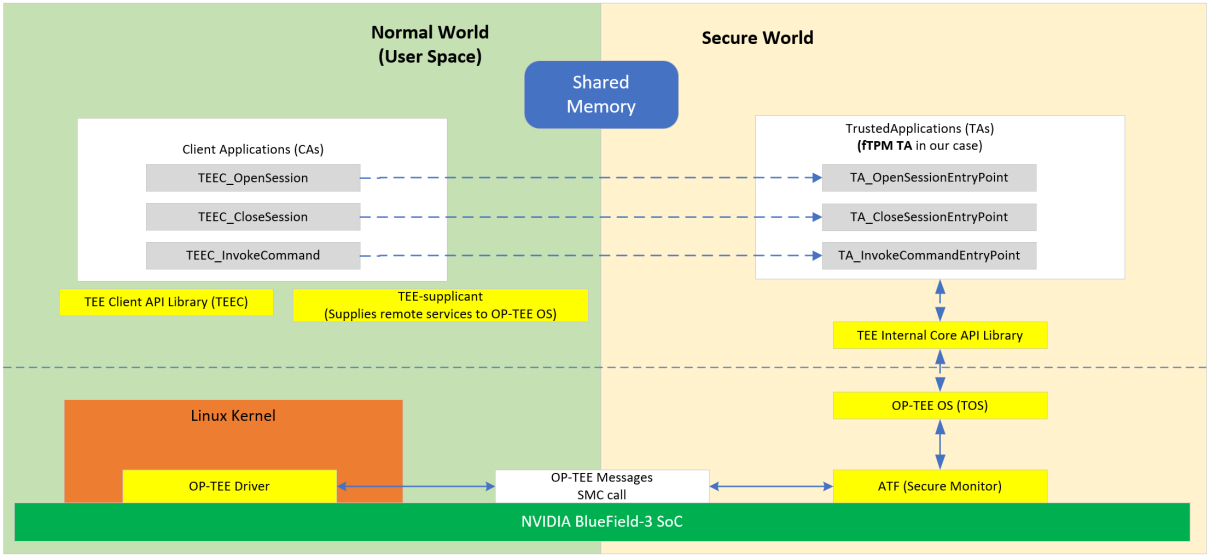
Characteristics of an fTPM:

- Emulated TPM using an isolated hardware environment
- Executes in an open-source trusted execution environment (OP-TEE)
- fTPM trusted application (TA) is part of the OP-TEE binary. This allows early access on bootup, runs only in secure DRAM.

  > Currently, the only TA supported is fTPM.

- fTPM is not a task waiting to be woken up. It only executes when TPM primitives are forwarded to it from the user space. It is guaranteed shielded execution via the TEE OS and, when invoked via the TEE Dispatcher, runs to completion.

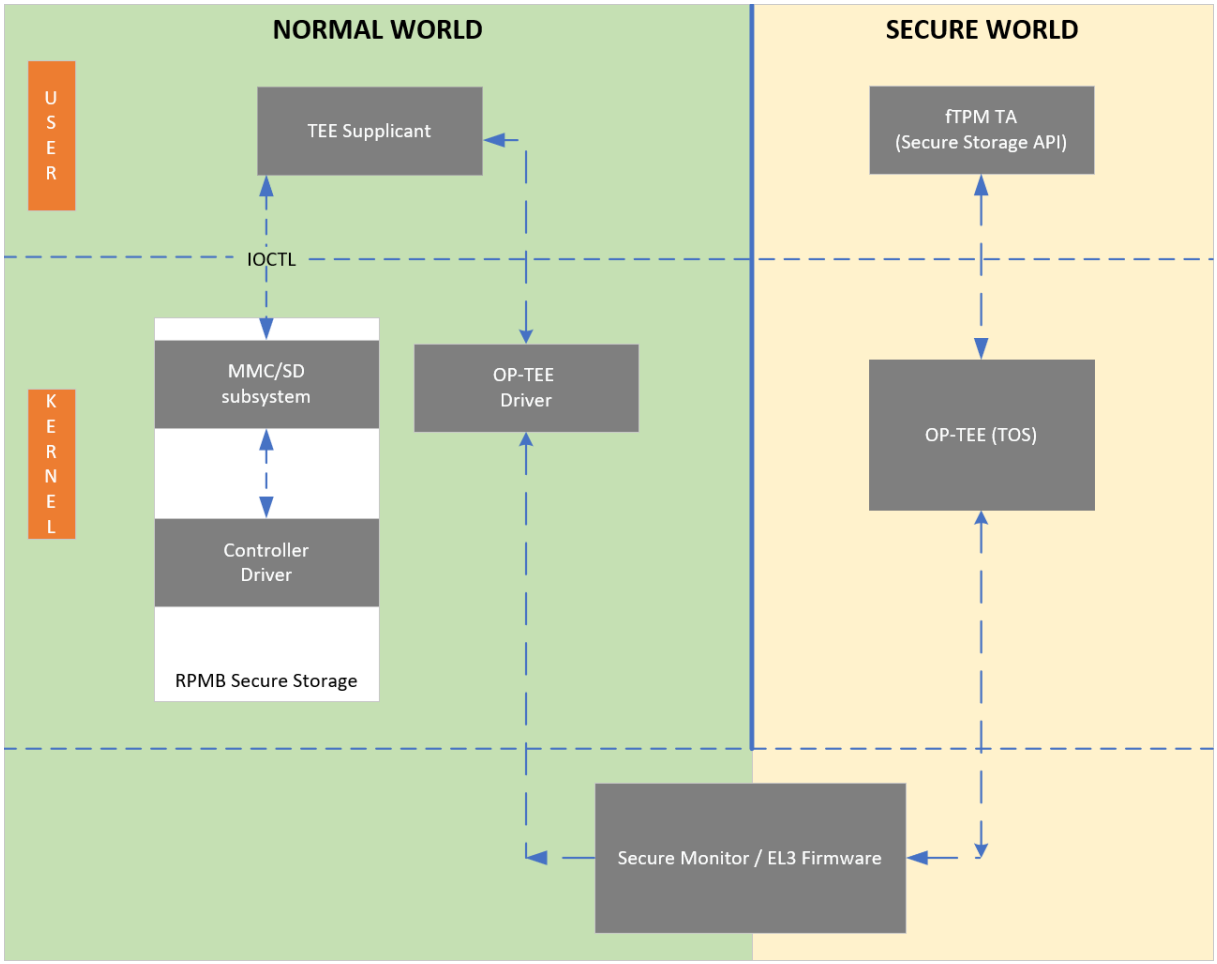
The fTPM TA is the only TA NVIDIA® BlueField®-3 currently supports. Any TA loaded by OP-TEE must be signed (signing done externally) and then authenticated by OP-TEE before being allowed to load and execute.



A replay-protected memory block (RPMB) is provided as a means for a system to store data to the specific memory area in an authenticated and replay-protected manner, making it readable and writable only after a successful authentication read/write accesses. The RPMB is a dedicated partition available on the eMMC, which makes it possible to store and retrieve data with integrity and authenticity support. A signed access to an RPMB is supported by first programming authentication key information to the eMMC memory (shared secret). The RPMB authentication key is programmed into the DPU at manufacturing time.

RPMB features a 4MB partition secure storage for BlueField-3.

There is no eMMC controller driver in OP-TEE. All device operations have to go through the normal world via the TEE-supplicant daemon, which relies on the Linux kernel's ioctl interface to access the device. All writes to the RPMB are atomic, authenticated, and encrypted. The RPMB partition stores data in an authenticated, replay-protected manner, making it a perfect complement to fTPM for storing and protecting data.



7.15.1 Enabling OP-TEE on BlueField-3

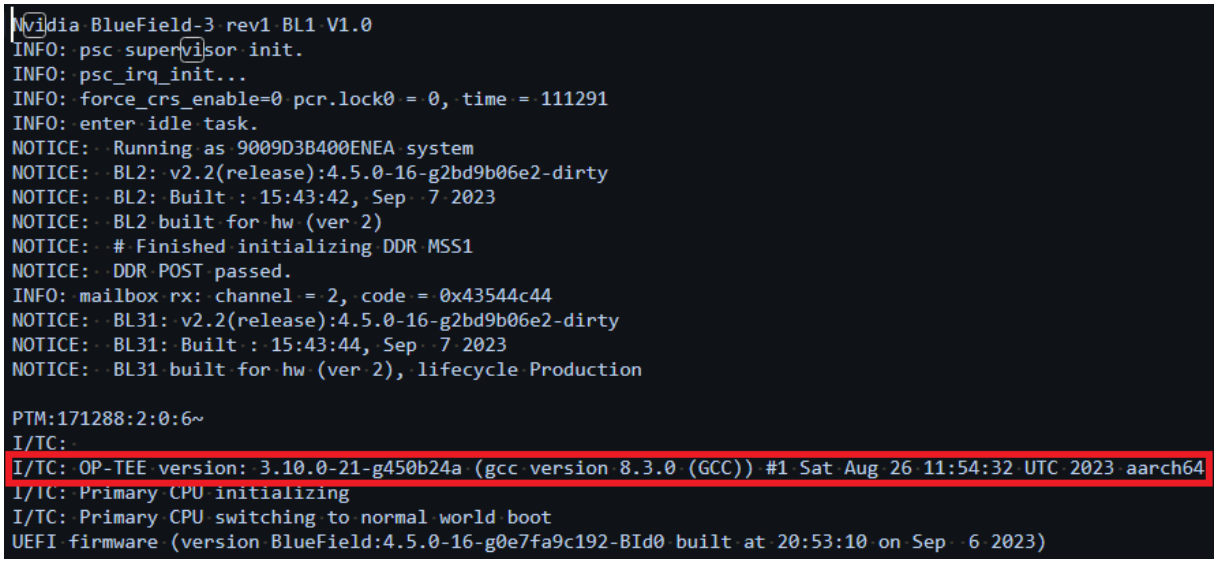
Enable OP-TEE in the UEFI menu:

1. ESC into the UEFI on DPU boot.
2. Navigate to Device Manager > System Configuration.
3. Check "Enable OP-TEE".
4. Save the change and reset/reboot.
5. Upon reboot OP-TEE is enabled.

> ⚠ OP-TEE is essentially dormant (does not have an OS scheduler) and reacts to external inputs.

7.15.2 Verifying BlueField-3 is Running OP-TEE

Users can see the OP-TEE version during BlueField-3 DPU boot:



The following indicators should all be present if fTPM over OP-TEE is enabled:

- Check "dmesg" for the OP-TEE driver initializing

```
root@localhost ~]# dmesg | grep tee
[    5.646578] optee: probing for conduit method.
[    5.653282] optee: revision 3.10 (450b24ac)
[    5.653991] optee: initialized driver
```

- Verify that the following kernel modules are loaded (running):

```
[root@localhost ~]# lsmod | grep tee
tpm_ftpm_tee           16384  0
optee                  49152  1
tee                    49152  3 optee,tpm_ftpm_tee
```

- Verify that the proper devices are created/available (4 in total):

```
[root@localhost ~]# ls -l /dev/tee*
crw------- 1 root root 234,  0 Sep  8 18:24 /dev/tee0
crw------- 1 root root 234, 16 Sep  8 18:24 /dev/teepriv0

[root@localhost ~]# ls -l /dev/tpm*
crw-rw---- 1 tss root  10,   224 Sep  8 18:24 /dev/tpm0
crw-rw---- 1 tss tss  252, 65536 Sep  8 18:24 /dev/tpmrm0
```

- Verify that the required processes are running (3 in total):

```
[root@localhost ~]# ps axu | grep tee
root         707  0.0  0.0  76208  1372 ?        Ssl  14:42   0:00 /usr/sbin/tee-supplicant
root         715  0.0  0.0      0     0 ?        I<   14:42   0:00 [optee_bus_scan]

[root@localhost ~]# ps axu | grep tpm
root         124  0.0  0.0      0     0 ?        I<   18:24   0:00 [tpm_dev_wq]
```

7.16 QoS Configuration

> ⚠ To learn more about port QoS configuration, refer to this community post.

When working in Embedded Host mode, using `mlnx_qos` on both the host and Arm will result with undefined behavior. Users must only use `mlnx_qos` from the Arm. After changing the QoS settings from Arm, users must restart the mlx5 driver on host.

When configuring QoS using DCBX, the `lldpad` service from the DPU side must be disabled if the configurations are not done using tools other than `lldpad`.

This section explains how to configure QoS group and settings using devlink located under `/opt/mellanox/iproute2/sbin/`. It is applicable to host PF/VF and Arm side SFs. The following uses VF as example.

The settings of a QoS group include creating/deleting a QoS group and modifying its `tx_max` and `tx_share` values. The settings of VF QoS include modifying its `tx_max` and `tx_share` values, assigning a VF to a QoS group, and unassigning a VF from a QoS group. This section focuses on the configuration syntax.

Please refer to section "Limit and Bandwidth Share Per VF" in the MLNX_OFED User Manual for detailed explanation on vPort QoS behaviors.

7.16.1 devlink port function rate add

	devlink port function rate add <DEV>/<GROUP_NAME> Adds a QoS group.	
Syntax Description	DEV/GROUP_NAME	Specifies group name in string format
Example	This command adds a new QoS group named "12_group" under device "pci/0000:03:00.0": `devlink port function rate add pci/0000:03:00.0/12_group`	
Notes		

7.16.2 devlink port function rate del

	devlink port function rate del <DEV>/<GROUP_NAME> Deletes a QoS group.	
Syntax Description	DEV/GROUP_NAME	Specifies group name in string format
Example	This command deletes QoS group "12_group" from device "pci/0000:03:00.0": `devlink port function rate del pci/0000:03:00.0/12_group`	
Notes		

7.16.3 devlink port function rate set tx_max tx_share

	devlink port function rate set {<DEV>/<GROUP_NAME> | <DEV>/<PORT_INDEX>} tx_max <TX_MAX> [tx_share <TX_SHARE>] Sets `tx_max` and `tx_share` for QoS group or devlink port.	
Syntax Description	DEV/GROUP_NAME	Specifies the group name to operate on
	DEV/PORT_INDEX	Specifies the devlink port to operate on
	TX_MAX	`tx_max` bandwidth in Mb/s
	TX_SHARE	tx_share bandwidth in Mb/s

Example	This command sets `tx_max` to 2000Mb/s and `tx_share` to 500Mb/s for the "12_group" QoS group:
	`devlink port function rate set pci/0000:03:00.0/12_group tx_max 2000Mbps tx_share 500Mbps`
	This command sets `tx_max` to 2000Mb/s and `tx_share` to 500Mb/s for the VF represented by port index 196609:
	`devlink port function rate set pci/0000:03:00.0/196609 tx_max 200Mbps tx_share 50Mbps`
	This command displays a mapping between VF devlink ports and netdev names:
	`$ devlink port`
	In the output of this command, VFs are indicated by `flavour pcivf`.	

Notes		

7.16.4 devlink port function rate set parent

	devlink port function rate set <DEV>/<PORT_INDEX> {parent <PARENT_GROUP_NAME>} Assigns devlink port to a QoS group.	
Syntax Description	DEV/PORT_INDEX	Specifies the devlink port to operate on
	PARENT_GROUP_NAME	parent group name in string format
Example	This command assigns this function to the QoS group "12_group": `devlink port function rate set pci/0000:03:00.0/196609 parent 12_group`	
Notes		

7.16.4.1 devlink port function rate set noparent

	devlink port function rate set <DEV>/<PORT_INDEX> noparent Ungroups a devlink port.	
Syntax Description	DEV/PORT_INDEX	Specifies the devlink port to operate on

Example	This command ungroups this function: `devlink port function rate set pci/0000:03:00.0/196609 noparent`
Notes	

7.16.5 devlink port function rate show

	devlink port function rate show [<DEV>/<GROUP_NAME> | <DEV>/<PORT_INDEX>] Displays QoS information QoS group or devlink port.	
Syntax Description	DEV/GROUP_NAME	Specifies the group name to display
	DEV/PORT_INDEX	Specifies the devlink port to display

Example	This command displays the QoS info of all QoS groups and devlink ports on the system:
	`devlink port function rate show`
	`pci/0000:03:00.0/12_group type node tx_max 2000Mbps tx_share 500Mbps`
	`pci/0000:03:00.0/196609 type leaf tx_max 200Mbps tx_share 50Mbps parent 12_group`
	This command displays QoS info of 12_group:
	`devlink port function rate show pci/0000:03:00.0/12_group`
	`pci/0000:03:00.0/12_group type node tx_max 2000Mbps tx_share 500Mbps`	

Notes	If a QoS group name or devlink port are not specified, all QoS groups and devlink ports are displayed.	

7.17 VirtIO-net Emulated Devices

Virtio-net device emulation enables users to create VirtIO-net emulated PCIe devices in the system where the NVIDIA® BlueField® DPU is connected. This is done by the virtio-net-controller software module present in the DPU. Virtio-net emulated devices allow users to hot plug up to 31 virtio-net PCIe PF Ethernet NIC devices or 504 virtio-net PCIe VF Ethernet NIC devices in the host system where the DPU is plugged in.

⚠ Currently, VirtIO specification v1.0 is supported.

DPU software also enables users to create virtio block PCIe PF and SR-IOV PCIe VF devices. This is covered in the *NVIDIA BlueField SNAP and virtio-blk SNAP Documentation*.

7.17.1 VirtIO-net Controller

Virtio-net-controller is a systemd service running on the DPU, with a user interface frontend to communicate with the background service. An SF representor is created for each virtio-net device created on the host. Virtio-net controller only uses an SF number ≥1000. Refer to section "Scalable Functions" for more information.

⚠ SF representor name is determined by udev rules. The default name is in the format of `<prefix><pf_num><sf_num>`. For example: `en3f0pf0sf1001`.

Each virtio-net PF/VF requires a dedicated SF and it should be reserved from mlxconfig (see section "VirtIO-net PF Device Configuration"). However, since an SF is a shared resource on the system, there may be other application-created SFs as well. In that case, `PF_TOTAL_SF` must be updated to consider those SFs. Otherwise, virtio-net is not able to create enough configured PF/VF.

⚠ Since the controller provides hardware resources and acknowledges (ACKs) the request from the host's virtio driver, it is mandatory to reboot the host OS first and the DPU second. This also applies to reconfiguring a controller from the DPU (e.g., reconfiguring LAG); unloading the virtio-net driver from guest side is recommended.

7.17.1.1 SystemD Service

Controller systemd service is enabled by default and runs automatically if `VIRTIO_NET_EMULATION_ENABLE` is true from mlxconfig.

1. To check controller service status, run:

```
$ systemctl status virtio-net-controller.service
```

2. To reload the service, make sure to unload virtio-net/virtio-pcie drivers on host. Then run:

```
$ systemctl restart virtio-net-controller.service
```

3. To monitor log output of the controller service, run:

```
$ journalctl -u virtio-net-controller -f
```

The controller service has an optional configuration file which allows users to customize several parameters. The configuration file should be defined on the DPU at the following path `/opt/mellanox/mlnx_virtnet/virtnet.conf`.

This file is read every time the controller starts. Dynamic change of `virtnet.conf` is not supported. It is defined as a JSON format configuration file. The currently supported options are:

- `ib_dev_p0` - RDMA device (e.g., `mlx5_0`) used to create SF on port 0. This port is the EMU manager when `is_lag` is 0. Default value is `mlx5_0`.
- `ib_dev_p1` - RDMA device (e.g., `mlx5_1`) used to create SF on port 1. Default value is `mlx5_1`.
- `ib_dev_lag` - RDMA LAG device (e.g., `mlx5_bond_0`) used to create SF on LAG. Default value is `mlx5_bond_0`. This port is EMU manager when `is_lag` is 1. `ib_dev_lag` and `ib_dev_p0` / `ib_dev_p1` cannot be configured simultaneously.
- `ib_dev_for_static_pf` - the RDMA device (e.g., `mlx5_0`) which the static virtio PF is created on

- `is_lag` - specifies whether LAG is used. Note that if LAG is used, make sure to use the correct IB dev for static PF.
- `static_pf` -
  - `mac_base` - base MAC address for static PFs. MACs are automatically assigned with the following pattern: `pf_mac` → `pf_0`, `pf_mac` +1 → `pf_1`, etc.

    > ⚠ Note that the controller does not validate the MAC address (other than its length). The user must ensure MAC is valid and unique.

  - `features` - virtio spec-defined feature bits for static PFs. If unsure, leave `features` out of the JSON file and a default value is automatically assigned.
- `vf` -
  - `mac_base` - base MAC address for static PFs. MACs are automatically assigned with the following pattern: `pf_mac` → `pf_0`, `pf_mac` +1 → `pf_1`, etc.
  - `features` - virtio spec-defined feature bits for static VFs. If unsure, leave `features` out of the JSON file and a default value is automatically assigned.
  - `vfs_per_pf` - number of VFs to create on each PF. This is mandatory if `mac_base` is specified.

    > ⚠ This value does not equal `VIRTIO_NET_EMULATION_NUM_VF` in mlxconfig. `vfs_per_pf` ≤ `VIRTIO_NET_EMULATION_NUM_VF`.

  - `qp_num` - number of QPs for each VF. If not specified, then the QP number assigned is taken from its parent PF.
- `recovery` - specifies whether recovery is enabled. If unspecified, recovery is enabled by default. To disable it, set `recovery` to 0.
- `sf_pool_percent` - determines the initial SF pool size as the percentage of `PF_TOTAL_SF` of `mlxconfig`. Valid range: [0, 100]. For instance, if the value is 5, it means an SF pool with 5% of `PF_TOTAL_SF` is created. 0 means no SF pool is reserved beforehand (default).

  > ⚠ `PF_TOTAL_SF` is shared by all applications. User must ensure the percent request is guaranteed or else the controller will not be able to reserve the requested SFs resulting in failure.

- `sf_pool_force_destroy` - specifies whether to destroy the SF pool. When set to 1, the controller destroys the SF pool when stopped/restarted (and the SF pool is recreated if `sf_pool_percent` is not 0 when starting), otherwise it does not. Default value is 0.

For example, the following definition has all static PFs using mlx5_0 (port 0) as the data path device in a non-lag configuration:

```
{
  "ib_dev_p0": "mlx5_0",
  "ib_dev_p1": "mlx5_1",
  "ib_dev_for_static_pf": "mlx5_0",
  "is_lag": 0,
  "recovery": 1,
  "sf_pool_percent": 0,
  "sf_pool_force_destroy": 0,
  "static_pf": {
```

```
    "mac_base": "11:22:33:44:55:66",
    "features": "0x230047082b"
  },
  "vf": {
    "mac_base": "CC:48:15:FF:00:00",
    "features": "0x230047082b",
    "vfs_per_pf": 100,
    "qp_num": 4
  }
}
```

The following is an example for LAG configuration:

```
{
  "ib_dev_lag": "mlx5_bond_0",
  "ib_dev_for_static_pf": "mlx5_bond_0",
  "is_lag": 1,
  "recovery": 1,
  "sf_pool_percent": 0,
  "sf_pool_force_destroy": 0
}
```

7.17.1.2 User Frontend

To communicate with the service, a user frontend program (virtnet) is installed on the DPU. Run the following command to check its usage:

```
# virtnet -h
usage: virtnet [-h] [-v] {hotplug,unplug,list,query,modify,log} ...

Nvidia virtio-net-controller command line interface v1.0.9

positional arguments:
  {hotplug,unplug,list,query,modify,log}
                        ** Use -h for sub-command usage
    hotplug             hotplug virtnet device
    unplug              unplug virtnet device
    list                list all virtnet devices
    query               query all or individual virtnet device(s)
    modify              modify virtnet device
    log                 set log level

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
```

Note that each positional argument has its own help menu as well. For example:

```
# virtnet log -h
usage: virtnet log [-h] -l {info,err,debug}
optional arguments:
  -h, --help            show this help message and exit
  -l {info,err,debug}, --level {info,err,debug}
                        log level: info/err/debug
```

To operate a particular device, either the VUID or device index can be used to locate the device. Both attributes can be fetched from command "virtnet list". For example, to modify the MAC of a specific VF, you may run either of the following commands:

```
# virtnet modify -p 0 -v 0 device -m 0C:C4:7A:FF:22:98
```

Or:

```
# virtnet modify -u <VUID-string> device -m 0C:C4:7A:FF:22:98
```

The following `modify` options require unbinding the virtio device from virtio-net driver in the guest OS:

- MAC

- MTU
- Features
- Msix_num
- max_queue_size

For example:

- On the guest OS:

```
$ echo "bdf of virtio-dev" > /sys/bus/pci/drivers/virtio-pci/unbind
```

- On the Arm side:

```
$ virtnet modify ...
```

- On the guest OS:

```
$ echo "bdf of virtio-dev" > /sys/bus/pci/drivers/virtio-pci/bind
```

7.17.1.3 Controller Recovery

It is possible to recover the control and data planes if communications are interrupted so the original traffic can resume.

Recovery depends on the JSON files stored in `/opt/mellanox/mlnx_virtnet/recovery` where there is a file that corresponds to each device (either PF or VF). The following is an example of the data stored in these files:

```
{
  "port_ib_dev": "mlx5_0",
  "pf_id": 0,
  "function_type": "pf",
  "bdf_raw": 26624,
  "device_type": "hotplug",
  "mac": "0c:c4:7a:ff:22:93",
  "pf_num": 0,
  "sf_num": 2000,
  "mq": 1
}
```

These files should not be modified under normal circumstances. However, if necessary, advanced users may tune settings to meet their requirements. Users are responsible for the validity of the recovery files and should only perform this when the controller is not running.

Controller recovery is enabled by default and does not need user configuration or intervention unless a system reset is needed or BlueField configuration is changed (i.e., any of the mlxconfig options `PCI_SWITCH_EMULATION_NUM_PORT`, `VIRTIO_NET_EMULATION_NUM_VF`, or `VIRTIO_NET_EMULATION_NUM_PF`). To this end, the files under `/opt/mellanox/mlnx_virtnet/recovery` must be deleted.

The first time LAG is configured with a controller, recover files must be cleaned up to ensure the controller does not try to recover devices with the previous IB parent device.

7.17.1.4 Controller Live Update

Live update minimizes network interface down time by performing online upgrade of the virtio-net controller without necessitating a full restart.

To perform a live update, you must install a newer version of the controller either using the `rpm` or `deb` package (depending on the OS distro used). Run:

For Ubuntu/Debian	`dpkg --force-all -i virtio-net-controller-x.y.z-1.mlnx.aarch64.deb`
For CentOS/RedHat	`rpm -Uvh virtio-net-controller-x.y.z-1.mlnx.aarch64.rpm --force`

It is recommended to use the following command to verify the versions of the controller currently running and the one just installed:

```
virtnet version
```

If the versions that are correct, issue the following command to start the live update process:

```
virtnet update --start
virtnet update -s
```

> ⚠ If an error appears regarding the "update" command not being supported, this implies that the controller version you are trying to install is too old. Reinstalling the proper version will resolve this issue.

During the update process, the following command may be used to check the update status:

```
virtnet update status
virtnet update -t
```

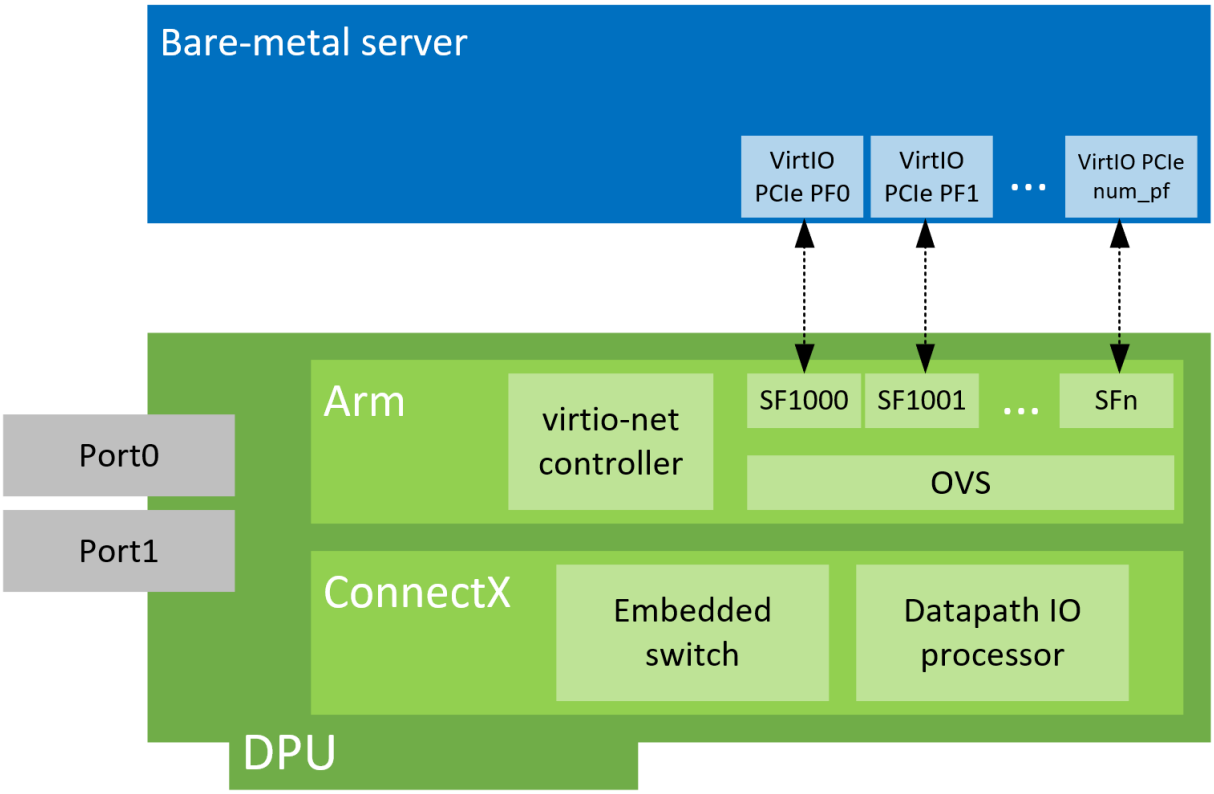
During the update, all existing `virtnet` commands (e.g., `list`, `query`, `modify`) are still supported. VF creation/deletion works as well.

When the update process completes successfully, the command `virtnet update status` will reflect the status accordingly.

> ⚠ If a device is actively migrating, the existing `virtnet` commands will appear as "migrating" for that specific device so that user can retry later.

7.17.2 VirtIO-net PF Devices

This section covers managing virtio-net PCIe PF devices using virtio-net controller.



7.17.2.1 VirtIO-net PF Device Configuration

1. Run the following command on the DPU:

```
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
```

2. Add the following kernel boot parameters to the Linux boot arguments:

```
pci=realloc
```

3. Cold reboot the host system.
4. Apply the following configuration on the DPU:

```
$ mst start
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s \
PCI_SWITCH_EMULATION_ENABLE=1 \
PCI_SWITCH_EMULATION_NUM_PORT=16 \
VIRTIO_NET_EMULATION_ENABLE=1 \
VIRTIO_NET_EMULATION_NUM_VF=0 \
VIRTIO_NET_EMULATION_NUM_PF=0 \
VIRTIO_NET_EMULATION_NUM_MSIX=10 \
SRIOV_EN=0 \
PF_SF_BAR_SIZE=10 \
PF_TOTAL_SF=64
$ mlxconfig -d /dev/mst/mt41686_pciconf0.1 s \
PF_SF_BAR_SIZE=10 \
PF_TOTAL_SF=64
```

5. Cold reboot the host system a second time.

7.17.2.2 Creating Modern Hotplug VirtIO-net PF Device

Virtio emulated network PCIe devices are created and destroyed using virtio-net-controller application console. When this application is terminated, all created virtio-net emulated devices are hot unplugged.

1. Create a hotplug virtio-net device. Run:

```
$ virtnet hotplug -i mlx5_0 -f 0x0 -m 0C:C4:7A:FF:22:93 -t 1500 -n 3 -s 1024
```

> ⚠ The maximum number of virtio-net queues is bound by the minimum of the following numbers:
>
> - `VIRTIO_NET_EMULATION_NUM_MSIX` from the command `mlxconfig -d <mst_dev> q`
> - `max_virtq` from the command `virtnet list`

   This creates one hotplug virtio-net device with MAC address 0C:C4:7A:FF:22:93, MTU 1500, and 3 virtio queues with a depth of 1024 entries. This device is uniquely identified by its index. This index is used to query and update device attributes. If the device is created successfully, an output appears similar to the following:

```
{
  "bdf": "85:00.0",
  "vuid": "VNETS1D0F0",
  "id": 3,
  "sf_rep_net_device": "en3f0pf0sf2000",
  "mac": "0C:C4:7A:FF:22:93"
}
```

2. Add the representor port of the device to the OVS bridge and bring it up. Run:

```
$ ovs-vsctl add-port <bridge> en3f0pf0sf2000
$ ip link set dev en3f0pf0sf2000 up
```

   Once steps 1-3 are completed, virtio-net device should be available in the host system.
3. To query all the device configurations of virtio-net device that you created, run:

```
$ virtnet query -p 0
```

4. To list all the virtio-net devices, run:

```
$ virtnet list
```

5. To modify device attributes, for example, changing its MAC address, run:

```
$ virtnet modify -p 0 device -m 0C:C4:7A:FF:22:98
```

6. Once usage is complete, to hot-unplug a virtio-net device, run:

```
$ virtnet unplug -p 0
```

7.17.2.3 Creating Transitional Hotplug VirtIO-net PF Device

A transitional device is a virtio device which supports drivers conforming to virtio specification 1.x and legacy drivers operating under virtio specification 0.95 (i.e., legacy mode) so that servers with old Linux kernels can still utilize virtio-based technology.

1. Run the following command on the DPU:

```
$ mst start
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s \
  VIRTIO_NET_EMULATION_PF_PCI_LAYOUT=1 \
  VIRTIO_EMULATION_HOTPLUG_TRANS=1
```

2. Add the following parameters to the Linux boot arguments on the guest OS (host OS or VM) side:

```
virtio_pci.force_legacy=1 intel_iommu=off
```

   Refer to the known limitations below.
3. Cold reboot the host system.
4. If `virtio_pci` is a kernel module rather than built-in from the guest OS, run the following command after both the host and DPU OSes are up:

```
modprobe -rv virtio_pci
modprobe -v virtio_pci force_legacy=1
```

5. To create a transitional hotplug virtio-net device. Run the following command on the DPU (with additional `-l` / `--legacy`):

```
$ virtnet hotplug -i mlx5_0 -f 0x0 -m 0C:C4:7A:FF:22:93 -t 1500 -n 3 -s 1024 -l
```

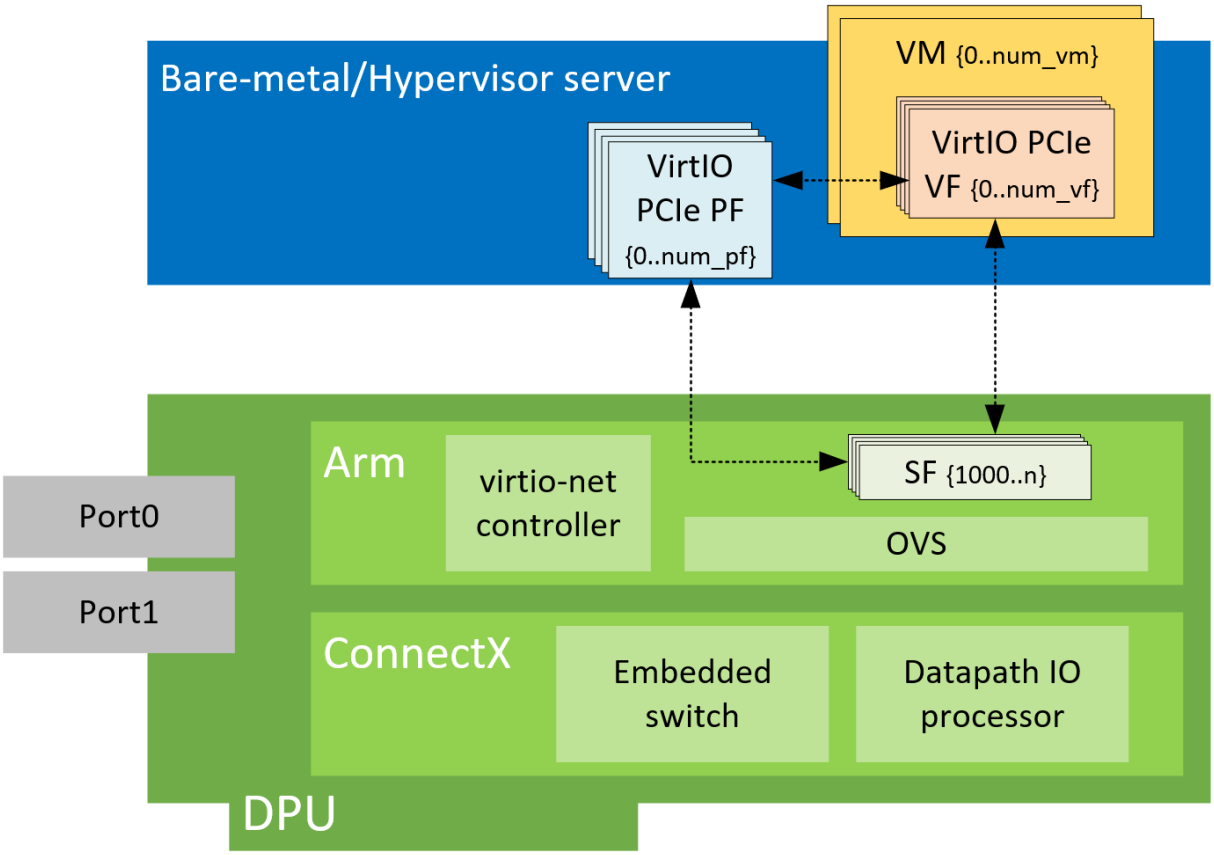
6. Proceed from step 2 of section "Creating Modern Hotplug VirtIO-net PF Device" for the rest of configuration.

Known limitations:

- AMD CPU is not supported.
- Only kernel versions 3.10 and above are supported. `intel_iommu=off` is not required for kernel 5.1 and above.
- An x86-64 system has only 64K I/O port space which is shared by all peripherals. The virtio transitional device uses I/O BAR. The hotplug device is under one PCIe bridge which is at the emulated PCIe switch downstream port. According to the PCIe specification, the granularity for the bridge I/O window is 4K bytes. If the system cannot satisfy the I/O resource demands by the emulated PCIe switch (depending on the port number of the PCIe switch), the I/O BAR allocation will fail. One hot-plug device requires one emulated PCIe switch port. Each emulated PCIe switch port takes 4K bytes of I/O space if the transitional virtio device is supported. Use `cat /proc/ioports` to check how many I/O port resources are allocated for the host bridge which contains the NIC. The number of supported hotplug transitional virtio device equals: (allocated I/O port space - 4k) / 4k.

7.17.3 Virtio-net SR-IOV VF Devices

This section covers managing virtio-net PCIe SR-IOV VF devices using virtio-net-controller.



7.17.3.1 Virtio-net SR-IOV VF Device Configuration

> ⚠ Virtio-net SR-IOV VF is only supported with statically configured PF, hot-plugged PF is not currently supported.

1. On the DPU, make sure virtio-net-controller service is enabled so that it starts automatically. Run:

   ```
   systemctl status virtio-net-controller.service
   ```

2. On the host, enable SR-IOV. Please refer to MLNX_OFED documentation under Features Overview and Configuration > Virtualization > Single Root IO Virtualization (SR-IOV) > Setting Up SR-IOV for instructions on how to do that. Make sure the parameters "`intel_iommu=on iommu=pt pci=realloc`" exist in `grub.conf` file.

3. It is recommended to add `pci=assign-busses` to the boot command line when creating more than 127 VFs. Without this option, the following errors might appear from host and the virtio driver will not probe these devices.

```
pci 0000:84:00.0: [1af4:1041] type 7f class 0xffffff
pci 0000:84:00.0: unknown header type 7f, ignoring device
```

4. Run the following command on the DPU:

```
mst start && mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
```

5. Add the following kernel boot parameters to the Linux boot arguments:

```
intel_iommu=on iommu=pt pci=realloc
```

6. Cold reboot the host system.
7. Apply the following configuration on the DPU in three steps to support up to 125 VFs per PF (500 VFs in total).

   a.
   ```
   $ mst start && mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1
   ```

   b.
   ```
   $ mlxconfig -d /dev/mst/mt41686_pciconf0 s \
   PCI_SWITCH_EMULATION_ENABLE=0 \
   PCI_SWITCH_EMULATION_NUM_PORT=0 \
   VIRTIO_NET_EMULATION_ENABLE=1 \
   VIRTIO_NET_EMULATION_NUM_VF=126 \
   VIRTIO_NET_EMULATION_NUM_PF=4 \
   VIRTIO_NET_EMULATION_NUM_MSIX=4 \
   SRIOV_EN=1 \
   PF_SF_BAR_SIZE=8 \
   PF_TOTAL_SF=508 \
   NUM_OF_VFS=0
   ```

   c.
   ```
   $ mlxconfig -d /dev/mst/mt41686_pciconf0.1 s PF_TOTAL_SF=1 PF_SF_BAR_SIZE=8
   ```

8. Cold reboot the host system.

7.17.3.2 Creating Virtio-net SR-IOV VF Devices

1. On the host, make sure the static virtio network device presents. Run:

```
# lspci | grep -i virtio
85:00.3 Network controller: Red Hat, Inc. Virtio network device
```

2. On the host, make sure virtio_pci and virtio_net are loaded. Run:

```
# lsmod | grep virtio
```

The net device should be created:

```
# ethtool -i p7p3
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: 0000:85:00.3
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
```

3. To create SR-IOV VF devices on the host, run:

```
# echo 2 > /sys/bus/pci/drivers/virtio-pci/0000\:85\:00.3/sriov_numvfs
```

> When the number of VFs created is high, SR-IOV enablement may take several minutes.

2 VFs should be created from the host:

```
# lspci | grep -i virt
85:00.3 Network controller: Red Hat, Inc. Virtio network device
85:04.5 Network controller: Red Hat, Inc. Virtio network device
85:04.6 Network controller: Red Hat, Inc. Virtio network device
```

4. From the DPU virtio-net controller, run the following command to get VF information.

```
# virtnet list
{
    "vf_id": 0,
    "parent_pf_id": 0,
    "function_type": "VF",
    "vuid": "VNETS0D0F2VF1",
    "bdf": "83:00.6",
    "sf_num": 3000,
    "sf_parent_device": "mlx5_0",
    "sf_rep_net_device": "en3f0pf0sf3000",
    "sf_rep_net_ifindex": 19,
    "sf_rdma_device": "mlx5_7",
    "sf_vhca_id": "0x192",
    "msix_config_vector": "0x0",
    "num_msix": 10,
    "max_queues": 4,
    "max_queues_size": 256,
    "net_mac": "5A:94:07:04:F6:1C",
    "net_mtu": 1500
},
```

You may use the pci-bdf to match the PF/VF on the host to the information showing on DPU.
To query all the device configurations of the virtio-net device of that VF, run:

```
$ virtnet query -p 0 -v 0
```

Add the corresponding SF representor to the OVS bridge and bring it up. Run:

```
# ovs-vsctl add-port <bridge> en3f0pf0sf1004
# ip link set dev en3f0pf0sf1004 up
```

Now the VF is functional.

> When port MTU (p0/p1 of the DPU) is changed after the controller is started, you must restart controller service. It is not recommended to use jumbo MTUs because that may lead to performance degradation.

5. To destroy SR-IOV VF devices on the host, run:

```
# echo 0 > /sys/bus/pci/drivers/virtio-pci/0000\:85\:00.3/sriov_numvfs
```

> When the command returns from the host OS, it does not necessarily mean the controller finished its operations. Look at controller log from the DPU and make sure you see a log like below before removing virtio kernel modules or recreate VFs.

```
# virtio-net-controller[3544]: [INFO]  virtnet.c:617:virtnet_device_vfs_unload: PF(0): Unload (4)
VFs finished
```

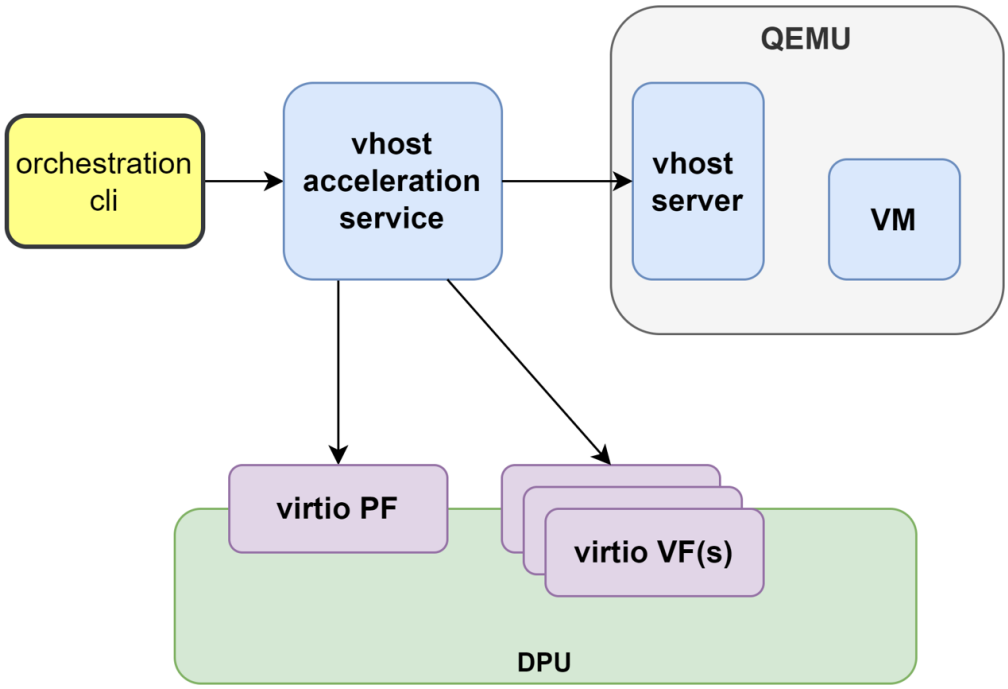
Once VFs are destroyed, created SFs from the DPU side are not destroyed but are saved into the SF pool to be reused later.

7.17.3.3 Transitional VirtIO-net VF Device Support

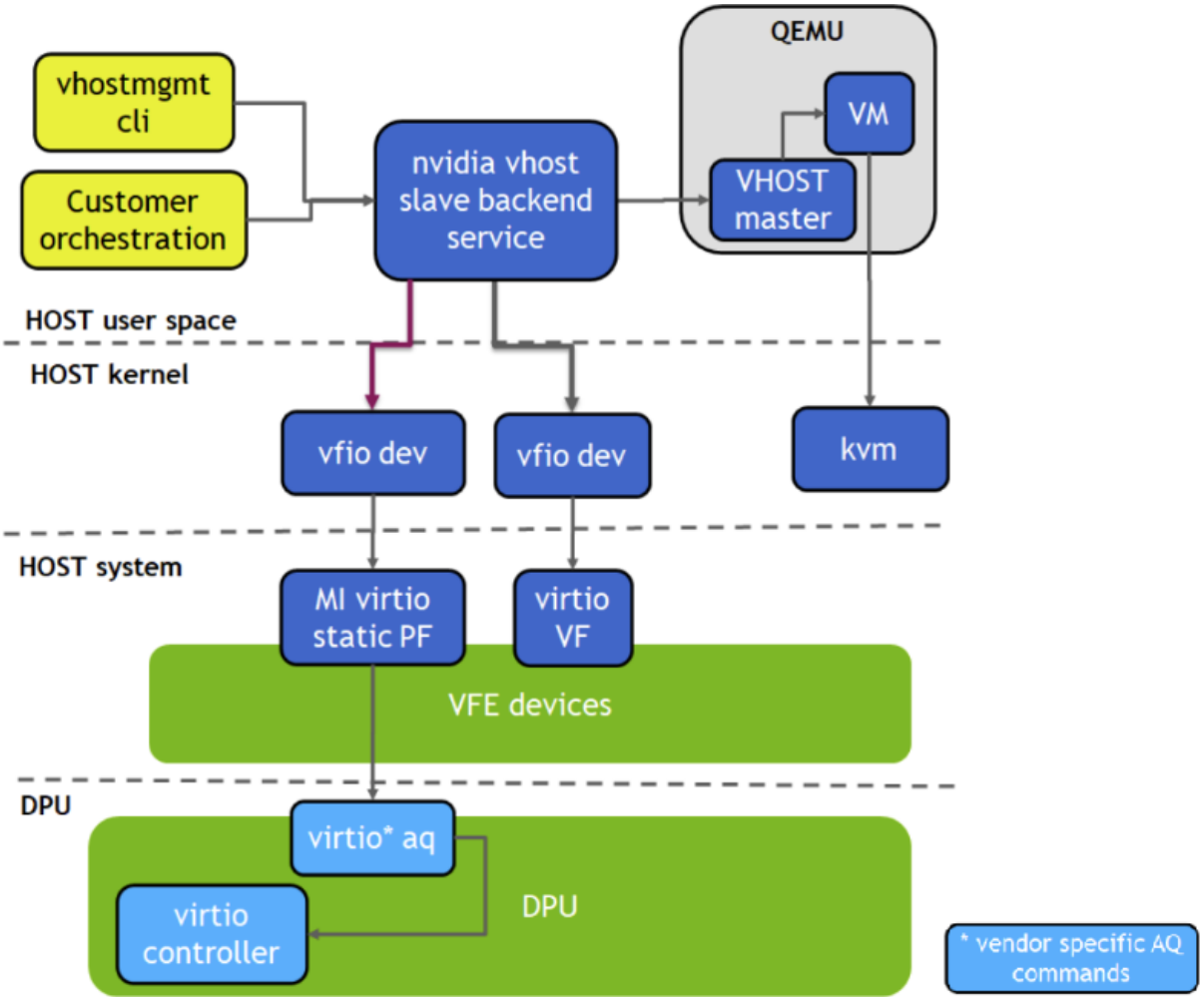
Transitional virtio-net VF devices are not currently supported.

7.17.4 Virtio VF PCIe Devices for vHost Acceleration

Virtio VF PCIe devices can be attached to the guest VM using vhost acceleration software stack. This enables performing live migration of guest VMs.



This section describes the steps to enable VM live migration using virtio VF PCIe devices along with vhost acceleration software.



7.17.4.1 Prerequisites

- Minimum hypervisor kernel version – Linux kernel 5.7 (for VFIO SR-IOV support)

7.17.4.2 Install vHost Acceleration Software Stack

Vhost acceleration software stack is built using open-source BSD licensed DPDK.

To install vhost acceleration software:

1. Clone the software source code.

```
git clone https://github.com/Mellanox/dpdk-vhost-vfe
```

Latest release tag is `vfe-1.0`.

2. Build software:

```
apt-get install libev-dev
yum install -y numactl-devel libev-devel
meson build -Dexamples=vdpa
ninja -C build
```

To install QEMU:

> Upstream QEMU later then 8.1 can be used or the following QEMU.

1. Clone QEMU sources.

```
git clone https://github.com/Mellanox/qemu -b stable-8.1-presetup
```

> Latest release tag is `vfe-0.4`.

2. Build QEMU.

```
mkdir bin
cd bin
../configure --target-list=x86_64-softmmu --enable-kvm
make -j24
```

7.17.4.3 Configure vHost and DPU System

1. Set the DPU nvconfig.

```
mlxconfig -d /dev/mst/mt41686_pciconf0 s \
VIRTIO_NET_EMULATION_ENABLE=1 VIRTIO_NET_EMULATION_NUM_PF=1 VIRTIO_NET_EMULATION_NUM_VF=16 \
VIRTIO_BLK_EMULATION_ENABLE=1 VIRTIO_BLK_EMULATION_NUM_PF=1 VIRTIO_BLK_EMULATION_NUM_VF=16 \
VIRTIO_NET_EMULATION_NUM_MSIX=64 VIRTIO_BLK_EMULATION_NUM_MSIX=64 NUM_VF_MSIX=64
```

2. Cold reboot the system after above configuration.
3. Setup the hypervisor system:
    a. Configure hugepages and libvirt VM XML (see OVS Hardware Offloads Configuration for information on doing that).
    b. Add a virtio-net interface and a virtio-blk interface in VM XML.

```
<qemu:commandline>
<qemu:arg value='-chardev'/>
<qemu:arg value='socket,id=char0,path=/tmp/vfe-net0,server=on'/>
<qemu:arg value='-netdev'/>
<qemu:arg value='type=vhost-user,id=vdpa,chardev=char0,queues=4'/>
<qemu:arg value='-device'/>
<qemu:arg value='virtio-net-pci,netdev=vdpa,mac=00:00:00:00:33:00,page-per-
vq=on,rx_queue_size=1024,tx_queue_size=1024,mq=on'/>
<qemu:arg value='-chardev'/>
<qemu:arg value='socket,id=char1,path=/tmp/vfe-blk0,server=on'/>
<qemu:arg value='-device'/>
<qemu:arg value='vhost-user-blk-pci,chardev=char1,page-per-vq=on,num-queues=4,disable-
legacy=on,disable-modern=off'/>
</qemu:commandline>
```

4. Create block device on the DPU:

```
spdk_rpc.py bdev_null_create Null0 1024 512
snap_rpc.py controller_virtio_blk_create --pf_id 0 --bdev_type spdk mlx5_0 --bdev Null0 --num_queues 1 --
admin_q --force_in_order
```

5. On BlueField-3 SNAP:

```
spdk_rpc.py bdev_null_create Null0 1024 512
snap_rpc.py virtio_blk_controller_create --pf_id 0 --bdev Null0 --num_queues 1 --admin_q --force_in_order
```

7.17.4.4 Run vHost Acceleration Service

1. Bind the virtio PF devices to `vfio-pci` driver:

```
modprobe vfio vfio_pci
echo 1 > /sys/module/vfio_pci/parameters/enable_sriov

echo 0x1af4 0x1041 > /sys/bus/pci/drivers/vfio-pci/new_id
echo 0x1af4 0x1042 > /sys/bus/pci/drivers/vfio-pci/new_id

echo 0000:af:00.2 > /sys/bus/pci/drivers/vfio-pci/bind
echo 0000:af:00.3 > /sys/bus/pci/drivers/vfio-pci/bind

lspci -vvv -s 0000:af:00.3 | grep "Kernel driver"
Kernel driver in use: vfio-pci
lspci -vvv -s 0000:af:00.2 | grep "Kernel driver"
Kernel driver in use: vfio-pci
```

2. Enable SR-IOV and create a VF(s):

```
echo 1 > /sys/bus/pci/devices/0000:af:00.2/sriov_numvfs
echo 1 > /sys/bus/pci/devices/0000:af:00.3/sriov_numvfs

lspci | grep Virtio
af:00.2 Ethernet controller: Red Hat, Inc. Virtio network device
af:00.3 Non-Volatile memory controller: Red Hat, Inc. Virtio block device
af:04.5 Ethernet controller: Red Hat, Inc. Virtio network device
af:05.1 Non-Volatile memory controller: Red Hat, Inc. Virtio block device
```

3. Add a VF representor to the OVS bridge on the DPU:

```
virtnet query -p 0 -v 0 | grep sf_rep_net_device
"sf_rep_net_device": "en3f0pf0sf3000",

 ovs-vsctl add-port ovsbr1 en3f0pf0sf3000
```

4. Run the vhost acceleration software service:

```
cd dpdk-vhost-vfe
sudo ./build/app/dpdk-vfe-vdpa -a 0000:00:00.0 --log-level=.,8 --vfio-vf-token=cdc786f0-59d4-41d9-b554-
fed36ff5e89f -- --client
```

5. Provision the virtio-net PF and VF.

```
cd dpdk-vhost-vfe

python ./app/vfe-vdpa/vhostmgmt mgmtpf -a 0000:af:00.2
# Wait on virtio-net-controller finishing handle PF FLR

# On DPU, change VF MAC address or other device options
virtnet modify -p 0 -v 0 device -m 00:00:00:00:33:00
python ./app/vfe-vdpa/vhostmgmt vf -a 0000:af:04.5 -v /tmp/vfe-net0
```

6. Provision the virtio-blk PF and VF.

```
cd dpdk-vhost-vfe

python ./app/vfe-vdpa/vhostmgmt mgmtpf -a 0000:af:00.3
# Wait on SNAP controller to finish handling PF FLR

# On DPU, the user must create a VF device controller before adding the VF device to the
# vhostmgmt upon pf or vf device delete from vhostmgmt, or vhostmgmt restart:
#       For BlueField-3, the VF controller is automatically recreated
#       For BlueField-2, the VF controller must be manually recreated
# Use snap_rpc.py controller_list to check for controller exsistence and create controller if it's not
there
snap_rpc.py controller_virtio_blk_create mlx5_0 --pf_id 0 --vf_id 0 --bdev_type spdk --bdev Null0 --
force_in_order
python ./app/vfe-vdpa/vhostmgmt vf -a 0000:af:05.1 -v /tmp/vfe-blk0
```

> ⚠ If the SR-IOV is disabled and reenabled, the user must re-provision the VFs.

7.17.4.5 Start the VM

```
virsh start <domain-name>
```

7.17.4.6 Simple Live Migration

Prepare two identical hosts and perform the provisioning of the virtio device to DPDK on both.

Boot the VM on one server:

```
virsh migrate --verbose --live --persistent gen-l-vrt-440-162-CentOS-7.4 qemu+ssh://gen-l-vrt-439/system --unsafe
```

7.17.4.7 Remove Device

When finished with using the virtio device, use following commands to remove them from DPDK:

```
python ./app/vfe-vdpa/vhostmgmt vf -r 0000:af:04.5
python ./app/vfe-vdpa/vhostmgmt mgmtpf -r 0000:af:00.2

python ./app/vfe-vdpa/vhostmgmt vf -r 0000:af:05.1
python ./app/vfe-vdpa/vhostmgmt mgmtpf -r 0000:af:00.3
```

7.18 Shared RQ Mode

When creating 1 send queue (SQ) and 1 receive queue (RQ), each representor consumes ~3MB memory per single channel. Scaling this to the desired 1024 representors (SFs and/or VFs) would require ~3GB worth of memory for single channel. A major chunk of the 3MB is contributed by RQ allocation (receive buffers and SKBs). Therefore, to make efficient use of memory, shared RQ mode is implemented so PF/VF/SF representors share receive queues owned by the uplink representor.

The feature is enabled by default. To disable it:

1. Edit the field `ALLOW_SHARED_RQ` in `/etc/mellanox/mlnx-bf.conf` as follows:

   ```
   ALLOW_SHARED_RQ="no"
   ```

2. Restart the driver. Run:

   ```
   /etc/init.d/openibd restart
   ```

To connect from the host to BlueField in shared RQ mode, please refer to section Verifying Connection from Host to BlueField.

> ⚠ PF/VF representor to PF/VF communication on the host is not possible.

The following behavior is observed in shared RQ mode:

- It is expected to see a `0` in the `rx_bytes` and `rx_packets` and valid `vport_rx_packets` and `vport_rx_bytes` after running traffic. Example output:

```
# ethtool -S pf0hpf
NIC statistics:
     rx_packets: 0
     rx_bytes: 0
     tx_packets: 66946
     tx_bytes: 8786869
     vport_rx_packets: 546093
     vport_rx_bytes: 321100036
     vport_tx_packets: 549449
     vport_tx_bytes: 321679548
```

- Ethtool usage - in this mode, it is not possible to change/set the ring or coalesce parameters for the RX side using ethtool. Changing channels also only affects the TX side.

7.19 RegEx Acceleration

NVIDIA® BlueField® DPU supports high-speed RegEx acceleration. This allows the host to offload multiple RegEx jobs to the DPU. This feature can be used from the host or from the Arm side.

An application using this feature typically loads a compiled rule set to the BlueField RegEx engines and sends jobs for processing. For each job, the RegEx engine will return a list of matches (e.g. matching rule, offset, length).

An example and standard API for loading the rules and sending RegEx jobs is available through DPDK.

For more details on RegEx Acceleration, please refer to DOCA documentation which can be accessed through this webpage.

For a RegEx compiler, please contact NVIDIA Support.

7.19.1 Configuring RegEx Acceleration

The RegEx application can run either from the host or Arm side. Before running the application, users must perform the following:

```
host$ sudo /etc/init.d/openibd stop
# Enable host access to the RegEx engine as root user
dpu$ echo 1 > /sys/bus/pci/devices/0000\:03\:00.0/regex/pf/regex_en
```

7.20 DPU Bring-Up and Driver Installation

It is recommended to upgrade your BlueField product to the latest software and firmware versions in order to enjoy the latest features and bug fixes. It is important that both the BlueField Arm host and the server host have the same MLNX_OFED version installed.

This section is made up of the following pages:

- MLNX_OFED Installation
- eMMC Backup and Restore
- Network Bonding Configuration

7.20.1 MLNX_OFED Installation

Unknown macro: 'easy-heading-free'

7.20.1.1 Installing MLNX_OFED on Arm Cores

7.20.1.1.1 Prerequisite Packages for Installing MLNX_OFED

- MLNX_OFED installation requires some prerequisite packages to be installed on the system Currently, CentOS installed on the DPU has a private network to the host via the RShim connection, and it can be used to Secure Copy Protocol (SCP) all the required packages. However, it is recommended for the DPU to have a direct access to the network to use "yum install" to install all the required packages. For direct access to the network, set up the routing on the host via:

```
$ iptables -t nat -o em1 -A POSTROUTING -j MASQUERADE
$ echo 1 > /proc/sys/net/ipv4/ip_forward
$ systemctl restart dhcpd
```

> "em1" is the outgoing network interface on the host. Change this according to your system requirements.

> These commands are not saved in Linux startup script, and might be needed again after host machine reboots.

- Reset the DPU network for Internet connection (access to the web) as long as the host is connected:

```
[root@localhost ~]# ifdown eth0; ifup eth0
[root@localhost ~]# ping google.com
PING google.com (172.217.10.142) 56(84) bytes of data.
64 bytes from lga34s16-in-f14.1e100.net (172.217.10.142): icmp_seq=1 ttl=53 time=19.2 ms
64 bytes from lga34s16-in-f14.1e100.net (172.217.10.142): icmp_seq=2 ttl=53 time=17.7 ms
64 bytes from lga34s16-in-f14.1e100.net (172.217.10.142): icmp_seq=3 ttl=53 time=15.8 ms
```

- Run "yum install" to install all the required MLNX_OFED packages:

```
$ yum install rpm-build
$ yum group install "Development Tools"
$ yum install kernel-devel-`uname -r`
$ yum install valgrind-devel libnl3-devel python-devel
$ yum install tcl tk
```

Note that this is not needed if you installed CentOS 7 with the kickstart ("-k") option.

7.20.1.1.2 Removing Pre-installed Kernel Module

There are cases where the kernel is shipped with an earlier version of the mlx5_core driver taken from the upstream Linux code. This version does not support the BlueField® Arm, but is loaded before the MLNX_OFED driver, and therefore, needs to be removed.
To remove the kernel module from the initramfs, run the following command:

```
$ mkdir /boot/tmp
$ cd /boot/tmp
$ gunzip  <  ../initramfs-4*64.img  |  cpio  -i
$ rm -f lib/modules/4*/updates/mlx5_core.ko
$ rm -f lib/modules/4*/updates/tmfifo*.ko
$ cp ../initramfs-4*64.img ../initramfs-4.11.0-22.el7a.aarch64.img-bak
$ find | cpio -H newc -o | gzip -9 > ../initramfs-4*64.img
$ rpm -e mlx5_core
$ depmod -a
```

7.20.1.2 Updating DPU Firmware

To burn the firmware which comes with OFED after OFED is installed, run:

- For CentOS and Ubuntu:

```
$ /opt/mellanox/mlnx-fw-updater/firmware/mlxfwmanager_sriov_dis_aarch64_<device_id> -force
```

- For Yocto:

```
$ /lib/firmware/mellanox/mlfwmanager_sriov_dis_aarch64_<device_id>
```

7.20.1.3 Installing MLNX_OFED on DPU

These instructions provide an example of MLNX_OFED_LINUX installation on RHEL7.4 ALT or CentOS 7.4ALT where the in-box kernel is 4.11.0-22.el7a.aarch64. For a different CentOS or RHEL version, the kernel version 4.11.0-22.el7a.aarch64 should be replaced by the corresponding in-box kernel version.

1. Transfer the MLNX_OFED image over to the BlueField. This can be done over the 1G OOB interface or RShim. The latter is used in this procedure. The MLNX_OFED images should be provided in the software drop:

```
$ scp MLNX_OFED_LINUX-4.2-X.X.X.X-rhel7.4alternate-aarch64.tgz root@192.168.100.2:/tmp
```

2. Install MLNX_OFED.

   If the date is not set correctly while installing MLNX_OFED, first, set the date (e.g date -s 'Mon Feb 5 15:02:10 EST 2018'), then run the installation.

   If the kernel on the BlueField is 4.11.0-22.el7a.aarch64, run:

```
$ cd /tmp
$ tar xzf MLNX_OFED_LINUX-4.2-X.X.X.X-rhel7.4alternate-aarch64.tgz
$ ./mlnxofedinstall --bluefield
```

   If the kernel is different than 4.11.0-22.el7a.aarch64, run:

```
$ cd /tmp/MLNX_OFED_LINUX-4.2-X.X.X.X-rhel7.4alternate-aarch64
$ ./mlnxofedinstall --add-kernel-support --skip-repo --bluefield
```

   Alternatively, the following command may be run regardless of whether in-box or customized kernel is used:

```
$ cd /mnt
```

```
$ ./mlnxofedinstall --bluefield --auto-add-kernel-support
```

This step might take longer than expected to be completed. If you are using a different package than the required one, run "yum install".

> ⚠ To get MLNX_OFED_LINUX installation with upstream rdma-core package (required for DPDK, SPDK, nvme-snap, etc.) add the parameters "--upstream-libs" and "--dpdk" to the mlnxofedinstall command.

3. Disable rshim-getty service. Run:

```
$ systemctl disable rshim-getty
```

4. Disable NetworkManager. Run:

```
$ systemctl disable NetworkManager.service
$ systemctl disable NetworkManager-wait-online.service
```

5. To bring up network interfaces when the NetworkManager is disabled, run:

```
$ sed -i -e "\$s@\$@\, RUN+=\"/sbin/ip link set dev '%k' up\"\, RUN+=\"/sbin/ethtool -L '%k' combined
4\"@" /etc/udev/rules.d/82-net-setup-link.rules
$ echo "SUBSYSTEM==\"net\", ACTION==\"add\", RUN+=\"/sbin/ip link set dev '%k' up\"" >> /etc/udev/rules.d/
82-net-setup-link.rules
```

6. Make sure that mlnx_snap.service is down. Run:

```
$ systemctl stop mlnx_snap.service
```

7. Restart openibd:

```
$ /etc/init.d/openibd restart
```

7.20.1.4 Installing MLNX_OFED on Host

MLNX_OFED should be installed on any host using the DPU. This includes the host used to provision the DPU as well as the final system where the DPU is attached to.
To install MLNX_OFED on the host:

```
$ mount MLNX_OFED_LINUX-4.2-X.X.X.X-rhel7.4-x86_64.iso /mnt
$ cd /mnt
$ ./mlnxofedinstall
```

> ⚠ The last step of installing MLNX_OFED is to check and update the firmware. If it is possible to flash the firmware, flash it back according to the instructions in 39267337.

Manually load the mlx5_core driver on the BlueField Arm before loading it on the host, as the BlueField Arm is responsible for managing the memory. Manually blacklist the mlx5_core driver on the host and load it only after the BlueField Arm loading process is complete. To blacklist the driver, run:

```
$ echo "blacklist mlx5_core" > /etc/modprobe.d/blacklist-mlx5_core.conf
```

To prevent the Linux kernel from loading the mlx5_core driver included inside of the initramfs, open /boot/grub/grub.conf and append the following to the vmlinux line:

```
$ rdblacklist=mlx5_core
```

Also, change to "ONBOOT=no" in /etc/infiniband/openib.conf.
Once the BlueField Arm driver is loaded, manually load the driver via:

```
$ modprobe mlx5_core
```

⚠ When rebooting CentOS on the Arm-side, the host-side driver should be unloaded first. This is done with "rmmod mlx5_ib mlx5_core ib_core mlx_compat mlxfw". Reload the host driver after the Arm driver is loaded.

7.20.2 eMMC Backup and Restore

Unknown macro: 'easy-heading-free'

The complete setup process can be time-consuming. Fortunately, the filesystem installed on one BlueField® can be directly used on another BlueField system. Therefore, the fastest, most efficient way to install CentOS onto a BlueField system is to restore the eMMC image backup from another BlueField image.

7.20.2.1 Backing Up the eMMC Image

Before backing up the eMMC, all of its partitions need to be unmounted to avoid data corruption. Unmounting the root partition of the CentOS is impractical, therefore using the initial Yocto running entirely on memory is a good option.

1. If the DPU is currently running CentOS, issue a shutdown command so that the kernel unmounts the entire filesystem:

```
[root@localhost ~]# shutdown -h now
[  OK  ] Started Show Plymouth Power Off Screen.
[  OK  ] Stopped Dynamic System Tuning Daemon.
[  OK  ] Stopped target Network.
         Stopping LSB: Bring up/down networking...
[  OK  ] Stopped LSB: Bring up/down networking.
         Stopping Network Manager...
[  OK  ] Stopped Network Manager.
……
         Unmounting /run/user/0...
         Unmounting /mnt...
         Unmounting /boot/efi...
[  OK  ] Deactivated swap /dev/disk/by-path/platform-PRP0001:00-part3.
[  OK  ] Deactivated swap /dev/disk/by-partu...4fa-7c6a-4fd4-a795-84415d19f840.
[  OK  ] Deactivated swap /dev/disk/by-id/mmc-R1J56L_0x353c1019-part3.
[  OK  ] Deactivated swap /dev/mmcblk0p3.
[  OK  ] Deactivated swap /dev/disk/by-uuid/...291-1ad6-4e3a-b5b4-9087950a3296.
[  OK  ] Unmounted /run/user/0.
[  OK  ] Unmounted /boot/efi.
         Unmounting /boot...
[  OK  ] Unmounted /mnt.
[14370.028599] XFS (mmcblk0p2): Unmounting Filesystem
[  OK  ] Unmounted /boot.
[  OK  ] Reached target Unmount All Filesystems.
[  OK  ] Stopped target Local File Systems (Pre).
[  OK  ] Stopped Remount Root and Kernel File Systems.
……
[14370.787777] reboot: Power down
ERROR:   System Off: operation not handled.
```

```
PANIC at PC : 0x0000000000459b9c
```

2. On the host, push the install.bfb through the RShim boot device for the BlueField to boot up running the Yocto mini system entirely on memory:

```
[root@bu-lab02 ~]# cat /root/BlueField-<version>/sample/install.bfb > /dev/rshim0/boot
```

3. Once the mini-Yocto has finished booting, bring up the interface which is selected to copy over the eMMC image to the host. Any working network interface can be used, in this example the representor interface is used as it offers a faster transfer speed (using the RShim network interface is also a good option).
   On the BlueField side:

```
root@bluefield:~# ifconfig rep0-0 192.168.200.2 up
```

   On the host side:

```
[root@bu-lab02 ~]# ifconfig p4p1 192.168.200.1/24 up
[root@bu-lab02 ~]# ping 192.168.200.2
PING 192.168.200.2 (192.168.200.2) 56(84) bytes of data.
64 bytes from 192.168.200.2: icmp_seq=1 ttl=64 time=0.281 ms
64 bytes from 192.168.200.2: icmp_seq=2 ttl=64 time=0.073 ms
^C
--- 192.168.200.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.073/0.177/0.281/0.104 ms
```

4. Check if netcat is working properly. This is the tool that is used to pipe data across networks.
   a. Set up a netcat server on the host to listen to port 12345, and let it send the message “Hello from host” to the client:

```
[root@bu-lab02 ~]# echo "Hello from host" | nc -l 12345
```

   b. On BlueField, send the message “Hello from BlueField” to the server:

```
root@bluefield:~# echo "Hello from BlueField" | nc 192.168.200.1 12345
Hello from host
```

   c. On the host, the nc command completes and prints out “Hello from BlueField”:

```
[root@bu-lab02 ~]# echo "Hello from host" | nc -l 12345
Hello from BlueField
```

   d. This may fail since the iptables forbid listening to the port. If that happens, flush the rules by running:

```
iptables -F
```

      Forcing nc to use IPv4 addresses might resolve the issue:

```
[root@bu-lab02 ~]# nc -4 -l 12345
```

   e. Back up the eMMC image from the BlueField to the host. Set up the host to listen on port 12345, compress what it receives and store it to a file:

```
[root@bu-lab02 ~]# nc -l 12345 | pv | gzip -1 > /backup.dd.gz
```

> ⚠ The “pv” command is optional. It is used to monitor the progress of the backup. The backup should finish when the total data consumed is 13.8G, which is approximately 6 minutes if using the representor port.

5. On BlueField, read the entire eMMC boot partition with the “dd” command and pass it to the host:

```
root@bluefield:~# dd if=/dev/mmcblk0 bs=64M | nc 192.168.200.1 12345
```

6. If the “pv” command is used, it will start showing the transfer speed and data:

```
[root@bu-lab02 ~]# nc -l 12345 | pv | gzip -1 > / backup.dd.gz
7.69GiB 0:04:44 [64.4MiB/s] [        <=>                                  ]
```

7. Once this is complete, the generated backip.dd.gz file on the host is the compressed eMMC image of the BlueField system.

> ⚠ Note that the backup does not include the eMMC boot partitions, as they are actually physically separate partitions on the eMMC device.

8. If nc is not usable for any reason, the same thing can be accomplished via the “ssh” command. It will take longer due to the encryption/decryption overhead of SSH. On the host, to get the same results, run:

```
ssh 192.168.200.2 "dd if=/dev/mmcblk0 bs=64M" | pv | gzip -1 > /backup.dd.gz
```

7.20.2.2 Restoring the eMMC Image

To restore the eMMC, the BlueField system cannot be using the eMMC when recovering it, thus the mini Yocto running entirely on memory is the solution.

1. Push the install.bfb for it to boot from memory and set up the network interface that is going to be used. This step is the same as when backing up the eMMC image. Instead of transferring the image from the BlueField to the host, it is done the other way around.
2. Set up the host to extract the image and set up a netcat server to send the image:

```
[root@bu-lab02 ~]# zcat /backup.dd.gz | pv | nc -l 12345
```

3. On the BlueField side, retrieve the image using netcat and write it to the eMMC:

```
root@bluefield:~# nc 192.168.200.1 12345 | dd of=/dev/mmcblk0 bs=64M
```

   This can also be done with SSH. To have the same effect, on the host, run:

```
zcat /backup.dd.gz | pv | ssh 192.168.200.2 dd of=/dev/mmcblk0 bs=64M
```

4. When this is complete, the UEFI persistent variable needs to be set up so that UEFI knows where to boot grub from. This can be done using the efibootmgr tool included in the mini Yocto:

```
root@bluefield:~# mount -t efivarfs none /sys/firmware/efi/efivars
root@bluefield:~# efibootmgr -c -d /dev/mmcblk0 -p 1 -l "\EFI\centos\grubaa64.efi" -L "CentOS 7.4"
```

Alternatively, if this is not done, at boot time UEFI would stop at the boot menu and you would have to go to the UEFI console and use the UEFI console bcfg command to achieve the same affect:

```
Shell> bcfg boot add 0 FS0:\EFI\centos\shim.efi "CentOS 7.4"
```

5. Exit the shell and select “continue booting”, and UEFI resumes the next stage of the booting process.
   As mentioned before, this does not update the eMMC boot partitions. Therefore, if this process is used to deploy a new DPU, the boot partitions should also be updated by running the bfrec script from within the mini Yocto:

   ```
   root@bluefield:~# /opt/mlnx/scripts/bfrec
   ```

6. In addition, if OFED is already installed on the restored system, update the firmware of the BlueField to the matching one. This can be done when entering into the restored image via:

   ```
   [root@localhost ~]# /opt/mellanox/mlnx-fw-updater/firmware/mlxfwmanager_sriov_dis_aarch64
   ```

7.20.3 Network Bonding Configuration

Network bonding enables combining two or more network interfaces into a single interface. It increases the network throughput, bandwidth and provides redundancy if one of the interfaces fail.

Before you configure network bonding, make sure that the port is connected to a switch configured for LACP bonding. Then, on the Arm, run:

```
bindnmcli connection add con-name bond1 type bond ifname bond1 miimon 100 mode 4 ip4 192.168.200.19/24
nmcli connection add con-name slave-rep0-ffff type bond-slave ifname rep0-ffff master bond1
nmcli connection add con-name slave-rep1-ffff type bond-slave ifname rep1-ffff master bond1
nmcli connection up slave-rep0-ffff
nmcli connection up slave-rep1-ffff
nmcli connection up bond1
```

7.21 Transparent IPsec Encryption and Decryption

BlueField SmartNIC can be used for offloading IPsec operations from the host CPU and executing them transparently, utilizing the Arm cores of the SmartNIC. Please refer to the community post "BlueField IP Forwarding and IPSEC SPI RSS" for configuration and tuning instructions for optimal performance results.

7.22 Mediated Devices

NVIDIA mediated devices deliver flexibility in allowing to create accelerated devices without SR-IOV on the BlueField® system. These mediated devices support NIC and RDMA and offer the same level of ASAP2 offloads as SR-IOV VFs. Mediates devices are supported using mlx5 sub-function acceleration technology.

Two sub-function devices are created on the BlueField device upon boot (one per port if the port is in switchdev mode) using commands from "/etc/mellanox/mlnx-sf.conf":

```
/sbin/mlnx-sf -a create -d 0000:03:00.0 -u 61a59715-aeec-42d5-be83-f8f42ba8b049 --mac 12:11:11:11:11:11
/sbin/mlnx-sf -a create -d 0000:03:00.1 -u 5b198182-1901-4c29-97a0-6623f3d02065 --mac 12:11:11:11:11:12
```

The help menu for `mlnx-sf` is presented below:

```
Usage: mlnx-sf [ OPTIONS ]
OPTIONS:
-a, -action,    --action <action>               Perform action
    action:     { enable | create | configure | remove | show | set_max_mdevs | query_mdevs_num }

-d, -device,    --device <device>               PCI device <domain>:<bus>:<device>.<func> (E.g.: 0000:03:00.0)
-m, -max_mdevs, --max_mdevs <max mdevs number>  Set maximum number of MDEVs
-u, -uuid,      --uuid <uuid>                   UUID to create SF with
-M, -mac,       --mac <MAC>                     MAC to create SF with
-p, -permanent, --permanent [<conf file>]       Store configuration to be used after reboot and/or driver restart.
Default (/etc/mellanox/mlnx-sf.conf).
-V, -version,   --version                       Display script version and exit
-D, -dryrun,    --dryrun                        Display commands only
-v, -verbose,   --verbose                       Run script in verbose mode (print out every step of execution)
-h, -help,      --help                          Display help

"/etc/mellanox/mlnx-sf.conf" can be updated manually or using "mlnx-sf" tool with "-p" parameter.
```

7.22.1 Related Configuration

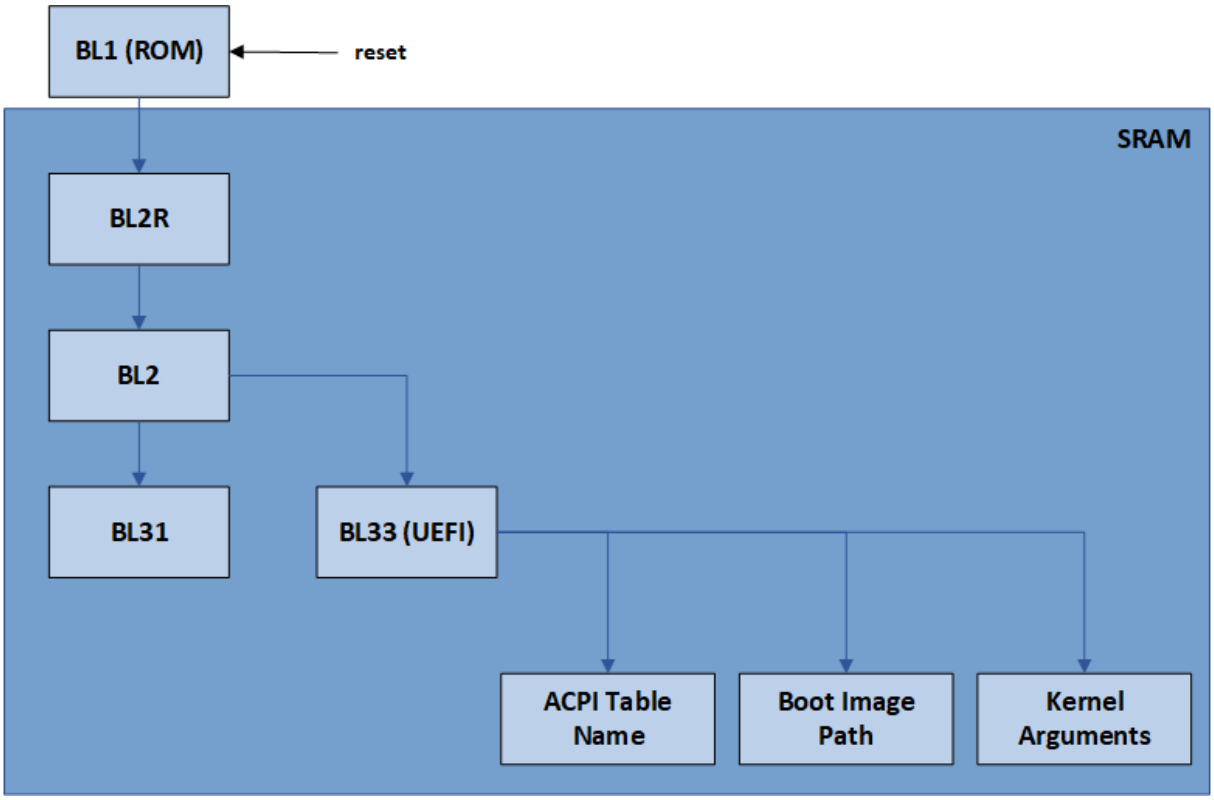
Interface names are configured using the UDEV rule under: `/etc/udev/rules.d/82-net-setup-link.rules`.

```
SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}!="", ATTR{phys_port_name}!="", \
        IMPORT{program}="/etc/infiniband/vf-net-link-name.sh $attr{phys_switch_id} $attr{phys_port_name}" \
        NAME="$env{NAME}", RUN+="/sbin/ethtool -L $env{NAME} combined 4"
# MDEV network interfaces
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/
pci0000:00/0000:00:00.0/0000:01:00.0/0000:02:02.0/0000:03:00.0/61a59715-aeec-42d5-be83-f8f42ba8b049/net/eth[0-9]",
NAME="p0m0"
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/
pci0000:00/0000:00:00.0/0000:01:00.0/0000:02:02.0/0000:03:00.1/5b198182-1901-4c29-97a0-6623f3d02065/net/eth[0-9]",
NAME="p1m0"
```

NVMe SNAP uses `p0m0` as its default interface. See `/etc/nvme_snap/sf1.conf`.

8 Upgrading Boot Software

This section describes how to use the BlueField alternate boot partition support feature to safely upgrade the boot software. We give the requirements that motivate the feature and explain the software interfaces that are used to configure it.



8.1 BFB File Overview

The default BlueField bootstream (BFB) shown above (located at `/lib/firmware/mellanox/boot/default.bfb`) is assumed to be loaded from the eMMC. In it, there is a hard-coded boot path pointing to a GUID partition table (GPT) on the eMMC device. Once loaded, as a side effect, this path would be also stored in the UPVS (UEFI Persistent Variable Store) EEPROM. That is, if you use the `bfrec` tools provided in the `mlxbf-bfscripts` package to write this BFB to the eMMC boot partition (see bfrec man for more information), then during boot, the DPU would load this from the boot FIFO, and the UEFI would assume to boot off the eMMC.

BFB files can be useful for many things such as installing new software on a BlueField DPU. For example, the installation BFB for BlueField platforms normally contains an initramfs file in the BFB chain. Using the initramfs (and Linux kernel Image also found in the BFB) you can do things like set the boot partition on the eMMC using `mlx-bootctl` or flash new HCA firmware using MFT utilities. You can also install a full root file system on the eMMC while running out of the initramfs.

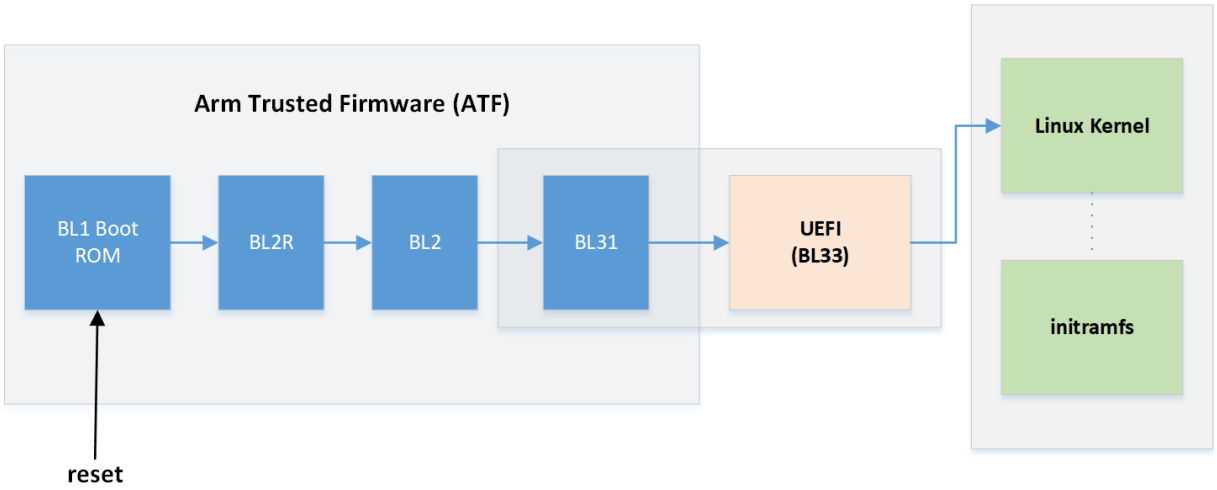
The following table presents the types of files possible in a BFB.

Filename	Description	ID	Read By
Bl2r-cert	Secure Firmware BL2R (RIoT Core) certificate	33	BL1
Bl2r	Secure Firmware BL2R (RIoT Core)	28	BL1
bl2-cert	Trusted Boot Firmware BL2 certificate	6	BL1/BL2R[(a)]
bl2	Trusted Boot Firmware BL2	1	BL1/BL2R[(a)]
trusted-key-cert	Trusted key certificate	7	BL2
bl31-key-cert	EL3 Runtime Firmware BL3-1 key certificate	9	BL2
bl31-cert	EL3 Runtime Firmware BL3-1 certificate	13	BL2
bl31	EL3 Runtime Firmware BL3-1	3	BL2
bl32-key-cert	Secure Payload BL3-2 (Trusted OS) key certificate	10	BL2
bl32-cert	Secure Payload BL3-2 (Trusted OS) certificate	14	BL2
bl32	Secure Payload BL3-2 (Trusted OS)	4	BL2
bl33-key-cert	Non-Trusted Firmware BL3-3 key certificate	11	BL2
bl33-cert	Non-Trusted Firmware BL3-3 certificate	15	BL2
bl33	Non-Trusted Firmware BL3-3	5	BL2
boot-acpi	Name of the ACPI table	55	UEFI
boot-dtb	Name of the DTB file	56	UEFI
boot-desc	Default boot menu item description	57	UEFI
boot-path	Boot image path	58	UEFI
boot-args	Arguments for boot image	59	UEFI
boot-timeout	Boot menu timeout	60	UEFI
image	Boot image	62	UEFI
initramfs	In-memory filesystem	63	UEFI

[(a)] When BL2R is booted in BlueField-2 devices, both the BL2 image and the BL2 certificate are read by BL2R. Thus, the BL2 image and certificate are read by BL1. BL2R is not booted in BlueField-1 devices.

Before explaining the implementation of the solution, the BlueField boot process needs to be expanded upon.

8.2 BlueField Boot Process



The BlueField boot flow is comprised of 4 main phases:

- Hardware loads Arm Trusted Firmware (ATF)
- ATF loads UEFI—together ATF and UEFI make up the booter software
- UEFI loads the operating system, such as the Linux kernel
- The operating system loads applications and user data

When booting from eMMC, these stages make use of two different types of storage within the eMMC part:

- ATF and UEFI are loaded from a special area known as the eMMC boot partition. Data from a boot partition is automatically streamed from the eMMC device to the eMMC controller under hardware control during the initial boot-up. Each eMMC device has two boot partitions, and the partition which is used to stream the boot data is chosen by a non-volatile configuration register in the eMMC.
- The operating system, applications, and user data come from the remainder of the chip, known as the user area. This area is accessed via block-size reads and writes, done by a device driver or similar software routine.

8.3 Upgrading Bootloader

In most deployments, the Arm cores of BlueField are expected to obtain their bootloader from an on-board eMMC device. Even in environments where the final OS kernel is not kept on eMMC—for instance, systems which boot over a network—the initial booter code still comes from the eMMC.

Most software stacks need to be modified or upgraded in their lifetime. Ideally, the user can to install the new software version on their BlueField system, test it, and then fall back to an older version if the new one does not work. In some environments, it is important that this fallback operation happen automatically since there may be no physical access to the system. In others, there may be an external agent, such as a service processor, which could manage the process.

To satisfy the requests listed above, the following must be performed:

1. Provision two software partitions on the eMMC, 0 and 1. At any given time, one area must be designated the primary partition, and the other the backup partition. The primary partition is the one booted on the next reboot or reset.
2. Allow software running on the Arm cores to declare that the primary partition is now the backup partition, and vice versa. (For the remainder of this section, this operation is referred to as "swapping the partitions" even though only the pointer is modified, and the data on the partitions does not move.)
3. Allow an external agent, such as a service processor, to swap the primary and backup partitions.
4. Allow software running on the Arm cores to reboot the system, while activating an upgrade watchdog timer. If the upgrade watchdog expires (due to the new image being broken, invalid, or corrupt), the system automatically reboots after swapping the primary and backup partitions.

8.4 Updating Boot Partition

The Bluefield software distribution provides a boot file that can be used to update the eMMC boot partitions. The BlueField boot file (BFB) is located in the boot directory `<BF_INST_DIR>/boot/` and contains all the necessary boot loader images (i.e. ATF binary file images and UEFI binary image).

The table below presents the pre-built boot images included within the BlueField software release:

Filename	Description
bl1.bin	The trusted firmware bootloader stage 1 (BL1) image, already stored into the on-chip boot ROM. It is executed when the device is reset.
bl2r.bin	The secure firmware (RIoT core) image. This image provides support for crypto operation and calculating measurements for security attestation and is relevant to BlueField-2 devices only.
bl2.bin	The trusted firmware bootloader stage 2 (BL2) image
bl31.bin	The trusted firmware bootloader stage 3-1 (BL31) image
BLUEFIELD_EFI.fd	The UEFI firmware image. It is also referred to as the non-trusted firmware bootloader stage 3-3 (BL33) image.
default.bfb	The BlueField boot file (BFB) which encapsulates all bootloader components such as bl2r.bin, bl2.bin, bl31.bin, and BLUEFIELD_EFI.fd. This file may be used to boot the BlueField devices from the RShim interface. It also could be installed into the eMMC boot partition.

It is also possible to build bootloader images from sources and create the BlueField boot file (BFB). Please refer to the sections below for more details.

The software image includes various tools and utilities to update the eMMC boot partitions. It also embeds a boot file in `/lib/firmware/mellanox/boot/default.bfb`. To update the eMMC boot partitions using the embedded boot file, execute the following command from the BlueField console:

```
$ /opt/mellanox/scripts/bfrec
```

⚠ `bfrec` is also available under `/usr/bin`.

The boot partitions update is initiated by the `bfrec` tool at runtime. With no options specified, the "bfrec" uses the default boot file `/lib/firmware/mellanox/boot/default.bfb` to update the boot partitions of device `/dev/mmcblk0`. This might be done directly in an OS using the "mlxbf-bootctl" utility, or at a later stage after reset using the capsule interface.

The syntax of `bfrec` is as follows:

```
Syntax: bfrec [--help]
              [--bootctl [<FILE>]]
              [--capsule [<FILE>]]


Description:
--help                  : print help
--bootctl [<FILE>]      : update the boot partition via the kernel path. If no FILE is specified, then default is
used.
--capsule [<FILE>]      : update the boot partition via the capsule path. If no FILE is specified, then default is
used.
--policy POLICY         : determines the update policy. May be: single - updates the secondary partition and swaps
to it, dual - updates both boot partitions, does not swap. If this flag is not specified, 'single' policy is
assumed.
```

When `bfrec` is called with the option `--bootctl`, the tool uses the boot file FILE, if given, rather than the default `/lib/firmware/mellanox/boot/default.bfb` in order to update the boot partitions. The command line usage is as follows:

```
$ bfrec --bootctl
$ bfrec --bootctl FILE
```

Where FILE represents the BlueField boot file encapsulating the new bootloader images to be written to the eMMC boot partitions.

For example, if the new bootstream file which we would like to install and validate is called `newdefault.bfb`, download the file to the BlueField and update the eMMC boot partitions by executing the following commands from the BlueField console:

```
# /opt/mellanox/scripts/bfrec --bootctl newdefault.bfb
```

The `--capsule` option updates the boot partition via the capsule interface. The capsule update image is reported in UEFI, so that at a later point the bootloader consumes the capsule file and performs the boot partition update. This option might be executed with or without additional arguments. The command line usage is as follows:

```
$ bfrec --capsule
$ bfrec --capsule FILE
```

Where FILE represents the capsule update image file encapsulating the new boot image to be written to the eMMC boot partitions.

For example, if the new bootstream file which we want to install and validate is called "`newdefault.bfb`", download the file to the BlueField and update the eMMC boot partitions by executing the following commands from the BlueField console:

```
$ /opt/mellanox/scripts/bfrec --capsule newdefault.bfb $ reboot
```

For more information about the capsule updates, please refer to `<BF_INST_DIR>/Documentation/HOWTO-capsule`.

After reset, the BlueField platform boots from the newly updated boot partition. To verify the version of ATF and UEFI, execute the following command:

```
$ /opt/mellanox/scripts/bfver
```

8.4.1 mlxbf-bootctl

It is also possible to update the eMMC boot partitions directly with the `mlxbf-bootctl` tool. The tool is shipped as part of the software image (under `/sbin`) and the sources are shipped in the `src` directory in the BlueField Runtime Distribution. A simple `make` command builds the utility.

The syntax of `mlxbf-bootctl` is as follows:

```
syntax: mlxbf-bootctl [--help | -h] [--swap | -s]
                      [--device | -d MMCFILE]
                      [--output | -o OUTPUT] [--read | -r INPUT]
                      [--bootstream | -b BFBFILE]
                      [--overwrite-current]
                      [--watchdog-swap interval | --nowatchdog-swap]
```

Where:

- `--device` - use a device other than the default `/dev/mmcblk0`
- `--bootstream` - write the specified bootstream to the alternate partition of the device. This queries the base device (e.g. `/dev/mmcblk0`) for the alternate partition, and uses that information to open the appropriate boot partition device (e.g. `/dev/mmcblk0boot0`).
- `--overwrite-current` (used with "`--bootstream`") - overwrite the current boot partition instead of the alternate one

  Not recommended as there is no easy way to recover if the new bootloader code does not bring the system up. Use `--swap` instead.

- `--output` (used with "`--bootstream`") - specify a file to which to write the boot partition data (creating it if necessary), rather than using an existing master device and deriving the boot partition device
- `--watchdog-swap` - arrange to start the Arm watchdog timer with a countdown of the specified number of seconds until it triggers; also, set the boot software so that it swaps the primary and alternate partitions at the next reset
- `--nowatchdog-swap` - ensure that after the next reset, no watchdog is started, and no swapping of boot partitions occurs

To update the boot partitions, execute the following command:

```
$ mlxbf-bootctl --swap --device /dev/mmcblk0 --bootstream default.bfb
```

This writes the new bootstream to the alternate boot partition, swaps alternate and primary so that the new bootstream is used on the next reboot.

It is recommended to enable the watchdog when calling `mlxbf-bootctl` in order to ensure that the Arm bootloader can perform alternate boot in case of a nonfunctional bootloader code within the primary boot partition. If something goes wrong on the next reboot and the system does not come up properly, it will reboot and return to the original configuration. To do so, the user may run:

```
$ mlxbf-bootctl --bootstream bootstream.new --swap --watchdog-swap 60
```

This reboots the system, and if it hangs for 60 seconds or more, the watchdog fires and resets the chip, the bootloader swaps the partitions back again to the way they were before, and the system reboots back with the original boot partition data. Similarly, if the system comes up but panics and resets, the bootloader will again swap the boot partition back to the way it was before.

The user must ensure that Linux after the reboot is configured to boot up with the `sbsa_gwdt` driver enabled. This is the Server Base System Architecture (SBSA) Generic WatchDog Timer. As soon as the driver is loaded, it begins refreshing the watchdog and preventing it from firing, which allows the system to finish booting up safely. In the example above, 60 seconds are allowed from system reset until the Linux watchdog kernel driver is loaded. At that point, the user's application may open /dev/watchdog explicitly, and the application would then become responsible for refreshing the watchdog frequently enough to keep the system from rebooting.

For documentation on the Linux watchdog subsystem, see Linux watchdog documentation.

To disable the watchdog completely, run:

```
$ echo V > /dev/watchdog
```

The user may select to incorporate other features of the Arm generic watchdog into their application code using the programming API as well.

Once the system has booted up, in addition to disabling or reconfiguring the watchdog itself if the user desires, they must also clear the "swap on next reset" functionality from the bootloader by running:

```
$ mlxbf-bootctl --nowatchdog-swap
```

Otherwise, next time the system is reset (via reboot, external reset, etc.) it assumes a failure or watchdog reset occurred and swaps the eMMC boot partition automatically.

8.4.2 LVFS and fwupd

Officially released bootloaders (ATF and UEFI) may be alternatively installed from the LVFS (Linux Vendor Firmware Service). LVFS is a free service operated by the Linux Foundation, which allows vendors to host stable firmware images for easy download and installation.

The DPU must have a functioning connection to the Internet.

Interaction with LVFS is carried out through a standard open-source tool called fwupd. fwupd is an updater daemon that runs in the background, waiting for commands from a management

application. fwupd and the command line manager, fwupdmgr, comes pre-installed on the BlueField Ubuntu image.

To verify bootloader support for a fwupd update, run the following command:

```
$ fwupdmgr get-devices
```

If "UEFI Device Firmware" device appears, then your currently installed bootloader supports the update process. Other devices may appear depending on your distribution of choice. Version numbers similar to 0.0.0.1 may appear if you are using an older version of the bootloader.

1. Before updating, a fresh list of release metadata must be obtained. Run:

   ```
   $ fwupdmgr refresh
   ```

2. Optionally, to confirm if a new release is available, run:

   ```
   $ fwupdmgr get-releases
   ```

3. Update your system bootloader, run "upgrade" with the GUID of the UEFI device. Run:

   ```
   $ fwupdmgr upgrade 39342586-4e0e-4833-b4ba-1256b0ffb471
   ```

   This will upgrade the ATF and UEFI to the latest available stable version of the bootloader through a UEFI capsule update, without upgrading the root file system. If your system is already at the latest available version, this upgrade command will do nothing.
4. Reboot the DPU to complete the upgrade.

> ⚠ Installing boot firmware directly through mlxbf-bootctl may cause fwupdmgr to detect an incorrect version string. If your workflow depends on fwupd, try to update the bootloader through capsule update (i.e. bfrec --capsule) or fwupdmgr only.

For more information about LVFS and fwupd, please refer to the official website of LVFS.

8.4.3 Updating Boot Partitions with BMC

The Arm cores notify the BMC prior to the reboot that an upgrade is about to happen. Software running on the BMC can then be implemented to watch the Arm cores after reboot. If after some time the BMC does not detect the Arm cores come up properly, it can use its USB debug connection to the Arm cores to properly reset the Arm cores. It first sets a suitable mode bit that the Arm bootloader responds to by switching the primary and alternating boot partitions as part of resetting into its original state.

8.5 Creating BlueField Boot File

The BlueField software distribution provides tools to format and to package the bootloader images into a single bootable file.

To create the BlueField boot file, use the `mlx-mkbfb` tool with the appropriate images. The bootloader images are embedded within the BSD under `<BF_INST_DIR>/boot/`. It is also possible to build the binary images from sources. Please refer to the following sections for further details.

1. First, set the PATH variable:

```
$ export PATH=$PATH:<BF_INST_DIR>/bin
```

2. Then, generate the boot file by using the `mlx-mkbfb` command:

```
$ mlx-mkbfb \ --bl2 bl2.bin \ --bl31 bl31.bin \ --bl33 BLUEFIELD_EFI.fd \ --boot-acpi "=default" \
default.bfb
```

   This command creates the `default.bfb` from `bl2.bin`, `bl31.bin`, and `BLUEFILED_EFI.fd`. The generated file might be used to update the eMMC boot partitions.

To verify the content of the boot file, run:

```
$ mlx-mkbfb -d default.bfb
```

To extract the bootloader images from the boot file, run:

```
$ mlx-mkbfb -x default.bfb
```

To obtain further details about the tool options, run the tool with `-h` or `--help`.

8.6 UEFI Boot Management

The UEFI firmware provides boot management function that can be configured by modifying architecturally defined global variables which are stored in the UPVS EEPROM. The boot manager will attempt to load and boot the OS in an order defined by the persistent variables.

The UEFI boot manager can be configured; boot entries may be added or removed from the boot menu. The UEFI firmware can also effectively generate entries in this boot menu, according to the available network interfaces and possibly the disks attached to the system.

8.6.1 Boot Option

The boot option is a unique identifier for a UEFI boot entry. This identifier is assigned when the boot entry is created, and it does not change. It also represents the boot option in several lists, including the BootOrder array, and it is the name of the directory on disk in which the system stores data related to the boot entry, including backup copies of the boot entry. A UEFI boot entry ID has the format "`Bootxxxx`" where `xxxx` is a hexadecimal number that reflects the order in which the boot entries are created.

Besides the boot entry ID, the UEFI boot entry has the following fields:

- Description (e.g. Yocto, CentOS, Linux from RShim)
- Device Path (e.g. VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)/Image)
- Boot arguments (e.g. console=ttyAMA0 earlycon=pl011,0x01000000 initrd=initramfs)

8.6.2 List UEFI Boot Options

To display the boot option already installed in the NVIDIA® BlueField® system, reboot and go to the UEFI menu screen. To get to the UEFI menu, hit Esc when prompted (in the RShim or UART console) before the countdown timer runs out.

```
Press <ESC> twice to enter UEFI menu
3 seconds remaining
2 seconds remaining
1 seconds remaining
```

Boot options are listed as soon as you select the "Boot Manager" entry.

```
                                Boot Manager
                                                        Device Path :
   Boot Option Menu                                     HD(1,GPT,B55B6B71-964E
                                                        -714B-AAF8-7AE8D768372
   focal0                                               7,0x800,0x19000)/\EFI\
   ubuntu                                               ubuntu\shimaa64.efi
   Linux from rshim
   Linux from mmc0
   EFI Internal Shell
   EFI Misc Device
   EFI Network
   EFI Network 1
   EFI Network 2
   EFI Network 3
   EFI Network 4
   EFI Network 5
                                                  v
 ^v=Move Highlight      <Enter>=Select Entry      Esc=Exit
```

It is also possible to retrieve more details about the boot entries. To do so, select "EFI Internal Shell" entry from the Boot Manager screen.

```
UEFI Interactive Shell v2.1
EDK II
UEFI v2.50 (EDK II, 0x00010000)
Mapping table
      FS1: Alias(s):F1:
          VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)
      FS0: Alias(s):HD0b:;BLK1:
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)/HD(1,GPT,3DCADB7E-BCCC-4897-A766-3C070EDD)
     BLK0: Alias(s):
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)
     BLK2: Alias(s):
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)/HD(2,GPT,9E61E8B5-EC9C-4299-8A0B-1B42E3DB)


Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
Shell>
```

From the UEFI shell, you may run the following command to display the option list:

```
Shell> bcfg boot dump -v
```

Here `-v` displays the option list with extra info including boot parameters. The following is an output example:

```
Option: 00. Variable: Boot0000
  Desc    - Linux from rshim
  DevPath - VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)/Image
  Optional- Y
  00000000: 63 00 6F 00 6E 00 73 00-6F 00 6C 00 65 00 3D 00  *c.o.n.s.o.l.e.=.*
  00000010: 74 00 74 00 79 00 41 00-4D 00 41 00 30 00 20 00  *t.t.y.A.M.A.0. .*
  00000020: 65 00 61 00 72 00 6C 00-79 00 63 00 6F 00 6E 00  *e.a.r.l.y.c.o.n.*
  00000030: 3D 00 70 00 6C 00 30 00-31 00 31 00 2C 00 30 00  *=.p.l.0.1.1.,.0.*
  00000040: 78 00 30 00 31 00 30 00-30 00 30 00 30 00 30 00  *x.0.1.0.0.0.0.0.*
  00000050: 30 00 20 00 20 00 69 00-6E 00 69 00 74 00 72 00  *0. . .i.n.i.t.r.*
```

```
   00000060: 64 00 3D 00 69 00 6E 00-69 00 74 00 72 00 61 00  *d.=.i.n.i.t.r.a.*
   00000070: 6D 00 66 00 73 00 00 00-                         *m.f.s...*
Option: 01. Variable: Boot0002
  Desc    - Yocto Poky
  DevPath - HD(1,GPT,3DCADB7E-BCCC-4897-A766-3C070EDD7C25,0x800,0xAE800)/Image
  Optional- Y
  00000000: 63 00 6F 00 6E 00 73 00-6F 00 6C 00 65 00 3D 00  *c.o.n.s.o.l.e.=.*
  00000010: 74 00 74 00 79 00 41 00-4D 00 41 00 30 00 20 00 *t.t.y.A.M.A.0. .*
  00000020: 65 00 61 00 72 00 6C 00-79 00 63 00 6F 00 6E 00  *e.a.r.l.y.c.o.n.*
  00000030: 3D 00 70 00 6C 00 30 00-31 00 31 00 2C 00 30 00  *=.p.l.0.1.1.,.0.*
  00000040: 78 00 30 00 31 00 30 00-30 00 30 00 30 00 30 00  *x.0.1.0.0.0.0.0.*
  00000050: 30 00 20 00 72 00 6F 00-6F 00 74 00 3D 00 2F 00  *0. .r.o.o.t.=./.*
  00000060: 64 00 65 00 76 00 2F 00-6D 00 6D 00 63 00 62 00  *d.e.v./.m.m.c.b.*
  00000070: 6C 00 6B 00 30 00 70 00-32 00 20 00 72 00 6F 00  *l.k.0.p.2. .r.o.*
  00000080: 6F 00 74 00 77 00 61 00-69 00 74 00              *o.t.w.a.i.t.*
    Option: 02. Variable: Boot0003
      Desc    - EFI Misc Device
      DevPath - VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)
      Optional- N
    Option: 03. Variable: Boot0004
      Desc    - EFI Network
      DevPath - MAC(001ACAFFFF01,0x1)
      Optional- N
    Option: 04. Variable: Boot0005
      Desc    - EFI Network 1
      DevPath - MAC(001ACAFFFF01,0x1)/IPv4(0.0.0.0)
      Optional- N
    Option: 05. Variable: Boot0006
      Desc    - EFI Network 2
      DevPath - MAC(001ACAFFFF01,0x1)/IPv6(0000:0000:0000:0000:0000:0000:0000:0000)
      Optional- N
    Option: 06. Variable: Boot0007
      Desc    - EFI Network 3
      DevPath - MAC(001ACAFFFF01,0x1)/IPv4(0.0.0.0)/Uri()
      Optional- N
    Option: 07. Variable: Boot0008
      Desc    - EFI Internal Shell
      DevPath - MemoryMapped(0xB,0xFE5FE000,0xFEAE357F)/FvFile(7C04A583-9E3E-4F1C-AD65-E05268D0B4D1)
      Optional- N
```

Boot arguments are printed in Hex mode, but you may recognize the boot parameters printed on the side in ASCII format.

8.6.3 UEFI System Configuration

UEFI System Configuration menu can be accessed under UEFI menu → Device Manager → System Configuration.

The following options are supported:

- Set Password - set a password for UEFI. Default: No password.
- Select SPCR UART - choose UART for Port Console Redirection. Default: Disabled.
- Enable SMMU - enable SMMU in ACPI. Default: Disabled.
- Disable SPMI - disable/enable ACPI SPMI Table. Default: Enabled.
- Enable 2nd eMMC - this option is relevant only for some BlueField Reference Platform boards. Default: Disabled.
- Boot Partition Protection - enable eMMC boot partition so it can be updated by the UEFI capsule only
- Disable PCIe - disable PCIe in ACPI. Default: Enabled.
- Disable ForcePXERetry - if ForcePXE is enabled from the BMC, the boot process keeps retrying PXE boot if it fails unless this option is enabled. If ForcePXERetry is disabled, the boot process only attempts PXE boot once, then it retries the normal boot flow if all PXE boot entries fail.
- Reset EFI Variables - clears all EFI variables to factory default state and disables SMMU and wipes the BOOT option variables and secure boot keys
- Reset MFG Info - clears the manufacturing information

⚠ All the above options, except for password and the two reset options, are also programmatically configurable via the BlueField Linux `/etc/bf.cfg`. Refer to section "bf.cfg Parameters" for further information.

9 Troubleshooting and How-Tos

9.1 RShim Troubleshooting and How-Tos

9.1.1 Another backend already attached

Several generations of NVIDIA® BlueField® networking platforms (DPUs or SuperNICs) are equipped with a USB interface in which RShim can be routed, via USB cable, to an external host running Linux and the RShim driver.

In this case, typically following a system reboot, the RShim over USB prevails and the BlueField host reports RShim status as "`another backend already attached`". This is correct behavior, since there can only be one RShim backend active at any given time. However, this means that the BlueField host does not own RShim access.

To reclaim RShim ownership safely:

1. Stop the RShim driver on the remote Linux. Run:

```
systemctl stop rshim
systemctl disable rshim
```

2. Restart RShim on the BlueField host. Run:

```
systemctl enable rshim
systemctl start rshim
```

The "`another backend already attached`" scenario can also be attributed to the RShim backend being owned by the BMC in BlueField devices with integrated BMC. This is elaborated on further down on this page.

9.1.2 RShim driver not loading

Verify whether your BlueField features an integrated BMC or not. Run:

```
# sudo sudo lspci -s $(sudo lspci -d 15b3: | head -1 | awk '{print $1}') -vvv | grep "Product Name"
```

Example output for BlueField with an integrated BMC:

```
Product Name: BlueField-2 DPU 25GbE Dual-Port SFP56, integrated BMC, Crypto and Secure Boot Enabled, 16GB on-board
DDR, 1GbE OOB management, Tall Bracket, FHHL
```

If your BlueField has an integrated BMC, refer to RShim driver not loading on host with integrated BMC.

If your BlueField does not have an integrated BMC, refer to RShim driver not loading on host on BlueField without integrated BMC.

9.1.2.1 RShim driver not loading on BlueField with integrated BMC

9.1.2.1.1 RShim driver not loading on host

1. Access the BMC via the RJ45 management port of BlueField.
2. Delete RShim on the BMC:

   ```
   systemctl stop rshim
   systemctl disable rshim
   ```

3. Enable RShim on the host:

   ```
   systemctl enable rshim
   systemctl start rshim
   ```

4. Restart RShim service. Run:

   ```
   sudo systemctl restart rshim
   ```

   If RShim service does not launch automatically, run:

   ```
   sudo systemctl status rshim
   ```

   This command is expected to display "`active (running)`".
5. Display the current setting. Run:

   ```
   # cat /dev/rshim<N>/misc | grep DEV_NAME
   DEV_NAME        pcie-04:00.2 (ro)
   ```

   This output indicates that the RShim service is ready to use.

9.1.2.1.2 RShim driver not loading on BMC

1. Verify that the RShim service is not running on host. Run:

   ```
   systemctl status rshim
   ```

   If the output is `active`, then it may be presumed that the host has ownership of the RShim.
2. Delete RShim on the host. Run:

   ```
   systemctl stop rshim
   systemctl disable rshim
   ```

3. Enable RShim on the BMC. Run:

   ```
   systemctl enable rshim
   ```

```
systemctl start rshim
```

4. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME
DEV_NAME        usb-1.0
```

This output indicates that the RShim service is ready to use.

9.1.2.2 RShim driver not loading on host on BlueField without integrated BMC

1. Download the suitable DEB/RPM for RShim (management interface for BlueField from the host) driver.
2. Reinstall RShim package on the host.
   - For Ubuntu/Debian, run:

     ```
     sudo dpkg --force-all -i rshim-<version>.deb
     ```

   - For RHEL/CentOS, run:

     ```
     sudo rpm -Uhv rshim-<version>.rpm
     ```

3. Restart RShim service. Run:

   ```
   sudo systemctl restart rshim
   ```

   If RShim service does not launch automatically, run:

   ```
   sudo systemctl status rshim
   ```

   This command is expected to display "`active (running)`".
4. Display the current setting. Run:

   ```
   # cat /dev/rshim<N>/misc | grep DEV_NAME
   DEV_NAME        pcie-04:00.2 (ro)
   ```

   This output indicates that the RShim service is ready to use.

9.1.3 Change ownership of RShim from NIC BMC to host

1. Verify that your card has BMC. Run the following on the host:

   ```
   # sudo sudo lspci -s $(sudo lspci -d 15b3: | head -1 | awk '{print $1}') -vvv |grep "Product Name"
   Product Name: BlueField-2 DPU 25GbE Dual-Port SFP56, integrated BMC, Crypto and Secure Boot Enabled, 16GB
   on-board DDR, 1GbE OOB management, Tall Bracket, FHHL
   ```

   The product name is supposed to show "integrated BMC" .
2. Access the BMC via the RJ45 management port of BlueField.
3. Delete RShim on the BMC:

   ```
   systemctl stop rshim
   systemctl disable rshim
   ```

4. Enable RShim on the host:

```
systemctl enable rshim
systemctl start rshim
```

5. Restart RShim service. Run:

```
sudo systemctl restart rshim
```

If RShim service does not launch automatically, run:

```
sudo systemctl status rshim
```

This command is expected to display "`active (running)`".

6. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME
DEV_NAME        pcie-04:00.2 (ro)
```

This output indicates that the RShim service is ready to use.

9.1.4 How to support multiple BlueField devices on the host

For more information, refer to section "RShim Multiple Board Support".

9.1.5 BFB installation monitoring

The BFB installation flow can be traced using various interfaces:

- From the host:
  - RShim console (`/dev/rshim0/console`)
  - RShim log buffer (`/dev/rshim0/misc`); also included in bfb-install's output
  - UART console (`/dev/ttyUSB0`)
- From the BMC console:
  - SSH to the BMC and run `obmc-console-client`

    Additional information about BMC interfaces is available in BMC software documentation
- From the BlueField:
  - `/root/<OS>.installation.log` available on the BlueField Arm OS after installation

9.2 Connectivity Troubleshooting

9.2.1 Connection (ssh, screen console) to the BlueField is lost

The UART cable in the Accessories Kit (OPN: MBF20-DKIT) can be used to connect to the DPU console and identify the stage at which BlueField is hanging.

Follow this procedure:

1. Connect the UART cable to a USB socket, and find it in your USB devices.

```
sudo lsusb
Bus 002 Device 003: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial (UART) IC
```

For more information on the UART connectivity, please refer to the DPU's hardware user guide under Supported Interfaces > Interfaces Detailed Description > NC-SI Management Interface.

It is good practice to connect the other end of the NC-SI cable to a different host than the one on which the BlueField DPU is installed.

2. Install the minicom application.
    - For CentOS/RHEL:

      ```
      sudo yum install minicom -y
      ```

    - For Ubuntu/Debian:

      ```
      sudo apt-get install minicom
      ```

3. Open the minicom application.

   ```
   sudo minicom -s -c on
   ```

4. Go to "Serial port setup"
5. Enter "F" to change "Hardware Flow control" to NO
6. Enter "A" and change to `/dev/ttyUSB0` and press Enter
7. Press ESC.
8. Type on "Save setup as dfl"
9. Exit minicom by pressing Ctrl + a + z.

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0                              |
|                                                                       |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

9.2.2 Driver not loading in host server

What this looks like in dmsg:

```
[275604.216789] mlx5_core 0000:af:00.1: 63.008 Gb/s available PCIe bandwidth, limited by 8 GT/s x8 link at
0000:ae:00.0 (capable of 126.024 Gb/s with 16 GT/s x8 link)
[275624.187596] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
100s
[275644.152994] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
79s
[275664.118404] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
59s
[275684.083806] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
39s
[275704.049211] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
19s
[275723.954752] mlx5_core 0000:af:00.1: mlx5_function_setup:1237:(pid 943): Firmware over 120000 MS in pre-
initializing state, aborting
```

```
[275723.968261] mlx5_core 0000:af:00.1: init_one:1813:(pid 943): mlx5_load_one failed with error code -16
[275723.978578] mlx5_core: probe of 0000:af:00.1 failed with error -16
```

The driver on the host server is dependent on the Arm side. If the driver on Arm is up, then the driver on the host server will also be up.

Please verify that:

- The driver is loaded in the BlueField (Arm)
- The Arm is booted into OS
- The Arm is not in UEFI Boot Menu
- The Arm is not hanged

Then:

1. Perform a graceful shutdown and a power cycle on the host server.
2. If the problem persists, reset nvconfig (`sudo mlxconfig -d /dev/mst/<device> -y reset`), perform a graceful shutdown, then power cycle the host.

   > **Warning**
   >
   > If your DPU is VPI capable, please be aware that this configuration will reset the link type on the network ports to IB. To change the network port's link type to Ethernet, run:
   >
   > ```
   > sudo mlxconfig -d <device> s LINK_TYPE_P1=2 LINK_TYPE_P2=2
   > ```

   Perform a graceful shutdown and systema graceful shutdown andPerform a graceful shutdown and system
3. If this problem still persists, please make sure to install the latest bfb image and then restart the driver in host server. Please refer to "Upgrading NVIDIA BlueField DPU Software" for more information.

9.2.3 No connectivity between network interfaces of source host to destination device

Verify that the bridge is configured properly on the Arm side.

The following is an example for default configuration:

```
$ sudo ovs-vsctl show
f6740bfb-0312-4cd8-88c0-a9680430924f
    Bridge ovsbr1
        Port pf0sf0
            Interface pf0sf0
        Port p0
            Interface p0
        Port pf0hpf
            Interface pf0hpf
        Port ovsbr1
            Interface ovsbr1
                type: internal
    Bridge ovsbr2
        Port p1
            Interface p1
        Port pf1sf0
            Interface pf1sf0
        Port pf1hpf
            Interface pf1hpf
        Port ovsbr2
            Interface ovsbr2
                type: internal
    ovs_version: "2.14.1"
```

If no bridge configuration exists, please refer to "Virtual Switch on BlueField DPU".

9.2.4 Uplink in Arm down while uplink in host server up

Please check that the cables are connected properly into the network ports of the DPU and the peer device.

9.3 Performance Troubleshooting

9.3.1 Degradation in performance

Degradation in performance indicates that openvswitch may not be offloaded.

Verify offload state. Run:

```
# ovs-vsctl get Open_vSwitch . other_config:hw-offload
```

- If `hw-offload = true` - Fast Pass is configured (desired result)
- If `hw-offload = false` - Slow Pass is configured

If `hw-offload = false`:

- For RHEL/CentOS, run:

```
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
# systemctl restart openvswitch;
# systemctl enable openvswitch;
```

- Ubuntu/Debian:

```
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
# /etc/init.d/openvswitch-switch restart
```

9.4 PCIe Troubleshooting and How-Tos

9.4.1 Insufficient power on the PCIe slot error

If the error "insufficient power on the PCIe slot" is printed in dmsg, please refer to the Specifications section of your hardware user guide and make sure that you are providing your DPU the correct amount of power.

To verify how much power is supported on your host's PCIe slots, run the command `lspci -vvv | grep PowerLimit`. For example:

```
# lspci -vvv | grep PowerLimit
                        Slot #6, PowerLimit 75.000W; Interlock- NoCompl-
                        Slot #1, PowerLimit 75.000W; Interlock- NoCompl-
                        Slot #4, PowerLimit 75.000W; Interlock- NoCompl-
```

⚠ Be aware that this command is not supported by all host vendors/types.

9.4.2 HowTo update PCIe device description

lspci may not present the full description for the NVIDIA PCIe devices connected to your host. For example:

```
# lspci | grep -i Mellanox
a3:00.0 Infiniband controller: Mellanox Technologies Device a2d6 (rev 01)
a3:00.1 Infiniband controller: Mellanox Technologies Device a2d6 (rev 01)
a3:00.2 DMA controller: Mellanox Technologies Device c2d3 (rev 01)
```

Please run the following command:

```
# update-pciids
```

Now you should be able to see the full description for those devices. For example:

```
# lspci | grep -i Mellanox
a3:00.0 Infiniband controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network
controller (rev 01)
a3:00.1 Infiniband controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network
controller (rev 01)
a3:00.2 DMA controller: Mellanox Technologies MT42822 BlueField-2 SoC Management Interface (rev 01)
```

9.4.3 HowTo handle two BlueField DPU devices in the same server

Please refer to section "Multi-board Management Example".

9.5 SR-IOV Troubleshooting

9.5.1 Unable to create VFs

1. Please make sure that SR-IOV is enabled in BIOS.
2. Verify `SRIOV_EN` is true and `NUM_OF_VFS` bigger than 1. Run:

   ```
   # mlxconfig -d /dev/mst/mt41686_pciconf0 -e q |grep -i "SRIOV_EN\|num_of_vf"
   Configurations:          Default         Current         Next Boot
   *        NUM_OF_VFS      16              16              16
   *        SRIOV_EN        True(1)         True(1)         True(1)
   ```

3. Verify that `GRUB_CMDLINE_LINUX="iommu=pt intel_iommu=on pci=assign-busses"`.

9.5.2 No traffic between VF to external host

1. Please verify creation of representors for VFs inside the Bluefield DPU. Run:

   ```
   # /opt/mellanox/iproute2/sbin/rdma link |grep -i up
   ...
   link mlx5_0/2 state ACTIVE physical_state LINK_UP netdev pf0vf0
   ...
   ```

2. Make sure the representors of the VFs are added to the bridge. Run:

   ```
   # ovs-vsctl add-port <bridage_name> pf0vf0
   ```

3. Verify VF configuration. Run:

```
$ ovs-vsctl show
bb993992-7930-4dd2-bc14-73514854b024
    Bridge ovsbr1
        Port pf0vf0
            Interface pf0vf0
                type: internal
        Port pf0hpf
            Interface pf0hpf
        Port pf0sf0
            Interface pf0sf0
        Port p0
            Interface p0
    Bridge ovsbr2
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port pf1sf0
            Interface pf1sf0
        Port p1
            Interface p1
        Port pf1hpf
            Interface pf1hpf
    ovs_version: "2.14.1"
```

9.6 eSwitch Troubleshooting

9.6.1 Unable to configure legacy mode

To set devlink to "Legacy" mode in BlueField, run:

```
# devlink dev eswitch set pci/0000:03:00.0 mode legacy
# devlink dev eswitch set pci/0000:03:00.1 mode legacy
```

Please verify that:

- No virtual functions are open. To verify if VFs are configured, run:

  ```
  # /opt/mellanox/iproute2/sbin/rdma link | grep -i up
  link mlx5_0/2 state ACTIVE physical_state LINK_UP netdev pf0vf0
  link mlx5_1/2 state ACTIVE physical_state LINK_UP netdev pf1vf0
  ```

  If any VFs are configured, destroy them by running:

  ```
  # echo 0 > /sys/class/infiniband/mlx5_0/device/mlx5_num_vfs
  # echo 0 > /sys/class/infiniband/mlx5_1/device/mlx5_num_vfs
  ```

- If any SFs are configured, delete them by running:

  ```
  /sbin/mlnx-sf -a delete --sfindex <SF Index>
  ```

  ⚠ You may retrieve the `<SF Index>` of the currently installed SFs by running:

  ```
  # mlnx-sf -a show

  SF Index: pci/0000:03:00.0/229408
    Parent PCI dev: 0000:03:00.0
    Representor netdev: en3f0pf0sf0
    Function HWADDR: 02:61:f6:21:32:8c
    Auxiliary device: mlx5_core.sf.2
      netdev: enp3s0f0s0
      RDMA dev: mlx5_2

  SF Index: pci/0000:03:00.1/294944
    Parent PCI dev: 0000:03:00.1
    Representor netdev: en3f1pf1sf0
    Function HWADDR: 02:30:13:6a:2d:2c
    Auxiliary device: mlx5_core.sf.3
      netdev: enp3s0f1s0
  ```

```
    RDMA dev: mlx5_3
```

Pay attention to the SF Index values. For example:

```
/sbin/mlnx-sf -a delete --sfindex pci/0000:03:00.0/229408
/sbin/mlnx-sf -a delete --sfindex pci/0000:03:00.1/294944
```

If the error "`Error: mlx5_core: Can't change mode when flows are configured`" is encountered while trying to configure legacy mode, please make sure that

1. Any configured SFs are deleted (see above for commands).
2. Shut down the links of all interfaces, delete any ip xfrm rules, delete any configured OVS flows, and stop openvswitch service. Run:

```
ip link set dev p0 down
ip link set dev p1 down
ip link set dev pf0hpf down
ip link set dev pf1hpf down
ip link set dev vxlan_sys_4789 down

ip x s f ;
ip x p f ;

tc filter del dev p0 ingress
tc filter del dev p1 ingress
tc qdisc show dev p0
tc qdisc show dev p1
tc qdisc del dev p0 ingress
tc qdisc del dev p1 ingress
tc qdisc show dev p0
tc qdisc show dev p1

systemctl stop openvswitch-switch
```

9.6.2 Arm appears as two interfaces

What this looks like:

```
# sudo /opt/mellanox/iproute2/sbin/rdma link
link mlx5_0/1 state ACTIVE physical_state LINK_UP netdev p0
link mlx5_1/1 state ACTIVE physical_state LINK_UP netdev p1
```

- Check if you are working in legacy mode.

```
# devlink dev eswitch show pci/0000:03:00.<0|1>
```

If the following line is printed, this means that you are working in legacy mode:

```
pci/0000:03:00.<0|1>: mode legacy inline-mode none encap enable
```

Please configure the DPU to work in switchdev mode. Run:

```
devlink dev eswitch set pci/0000:03:00.<0|1> mode switchdev
```

- Check if you are working in separated mode:

```
# mlxconfig -d /dev/mst/mt41686_pciconf0 q | grep -i cpu
* INTERNAL_CPU_MODEL SEPERATED_HOST(0)
```

Please configure the DPU to work in embedded mode. Run:

```
devlink dev eswitch set pci/0000:03:00.<0|1> mode switchdev
```

9.7 Isolated Mode Troubleshooting and How-Tos

9.7.1 Unable to burn FW from host server

Please verify that you are not in running in isolated mode. Run:

```
$ sudo mlxprivhost -d /dev/mst/mt41686_pciconf0 q
Current device configurations:
------------------------------
level                          : PRIVILEGED
...
```

By default, BlueField operates in privileged mode. Please refer to "Modes of Operation" for more information.

9.8 General Troubleshooting

9.8.1 Server unable to find the DPU

- Ensure that the DPU is placed correctly
- Make sure the DPU slot and the DPU are compatible
- Install the DPU in a different PCI Express slot
- Use the drivers that came with the DPU or download the latest
- Make sure your motherboard has the latest BIOS
- Perform a graceful shutdown then power cycle the server

9.8.2 DPU no longer works

- Reseat the DPU in its slot or a different slot, if necessary
- Try using another cable
- Reinstall the drivers for the network driver files may be damaged or deleted
- Perform a graceful shutdown then power cycle the server

9.8.3 DPU stopped working after installing another BFB

- Try removing and reinstalling all DPUs
- Check that cables are connected properly
- Make sure your motherboard has the latest BIOS

9.8.4 Link indicator light is off

- Try another port on the switch
- Make sure the cable is securely attached
- Check you are using the proper cables that do not exceed the recommended lengths
- Verify that your switch and DPU port are compatible

9.8.5 Link light is on but no communication is established

- Check that the latest driver is loaded
- Check that both the DPU and its link are set to the same speed and duplex settings

9.9 Installation Troubleshooting and How-Tos

9.9.1 bf.cfg Parameters

The following is a comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation:

```
################################################################################
# Configuration which can also be set in
#    UEFI->Device Manager->System Configuration
################################################################################
# Enable SMMU in ACPI.
#SYS_ENABLE_SMMU = TRUE

# Enable I2C0 in ACPI.
#SYS_ENABLE_I2C0 = FALSE

# Disable SPMI in ACPI.
#SYS_DISABLE_SPMI = FALSE

# Enable the second eMMC card which is only available on the BlueField Reference Platform.
#SYS_ENABLE_2ND_EMMC = FALSE

# Enable eMMC boot partition protection.
#SYS_BOOT_PROTECT = FALSE

# Enable SPCR table in ACPI.
#SYS_ENABLE_SPCR = FALSE

# Disable PCIe in ACPI.
#SYS_DISABLE_PCIE = FALSE

# Enable OP-TEE in ACPI.
#SYS_ENABLE_OPTEE = FALSE

################################################################################
# Boot Order configuration
# Each entry BOOT<N> could have the following format:
# PXE:
#    BOOT<N> = NET-<NIC_P0 | NIC_P1 | OOB | RSHIM>-<IPV4 | IPV6>
# PXE over VLAN (vlan-id in decimal):
#    BOOT<N> = NET-<NIC_P0 | NIC_P1 | OOB | RSHIM>[.<vlan-id>]-<IPV4 | IPV6>
# UEFI Shell:
#    BOOT<N> = UEFI_SHELL
# DISK: boot entries created during OS installation.
#    BOOT<N> = DISK
################################################################################
# This example configures PXE boot over the 2nd ConnectX port.
# If fails, it continues to boot from disk with boot entries created during OS
# installation.
#BOOT0 = NET-NIC_P1-IPV4
#BOOT1 = DISK

################################################################################
# Other misc configuration
################################################################################

# MAC address of the rshim network interface (tmfifo_net0).
#NET_RSHIM_MAC = 00:1a:ca:ff:ff:01

# DHCP class identifier for PXE (arbitrary string up to 32 characters)
#PXE_DHCP_CLASS_ID = NVIDIA/BF/PXE

# Create dual boot partition scheme (Ubuntu only)
# DUAL_BOOT=yes

# Upgrade NIC firmware
# WITH_NIC_FW_UPDATE=yes

# Target storage device for the DPU OS (Default SSD: /dev/nvme0n1)
device=/dev/nvme0n1

# bfb_modify_os - SHELL function called after file the system is extracted on the target partitions.
# It can be used to modify files or create new files on the target file system mounted under
# /mnt. So the file path should look as follows: /mnt/<expected_path_on_target_OS>. This
# can be used to run a specific tool from the target OS (remember to add /mnt to the path for
# the tool).

# bfb_pre_install - SHELL function called before EMMC partitions format
# and OS filesystem is extracted
```

```
# bfb_post_install - SHELL function called as a last step before reboot.
# All EMMC partitions are unmounted at this stage.
```

9.9.2 BlueField target is stuck inside UEFI menu

Upgrade to the latest stable boot partition images, see "How to upgrade the boot partition (ATF & UEFI) without re-installation".

9.9.3 BFB does not recognize the BlueField board type

If the .bfb file cannot recognize the BlueField board type, it reverts to low core operation. The following message will be printed on your screen:

```
***System type can't be determined***
***Booting as a minimal system***
```

Please contact NVIDIA Support if this occurs.

9.9.4 Unable to load BL2, BL2R, or PSC image

The following errors appear in console if images are corrupted or not signed properly:

Device	Error
BlueField	`ERROR: Failed to load BL2 firmware`
BlueField-2	`ERROR: Failed to load BL2R firmware`
BlueField-3	`Failed to load PSC-BL1` or `PSC VERIFY_BCT timeout`

9.9.5 CentOS fails into "dracut" mode during installation

This is most likely configuration related.

- If installing through the RShim interface, check whether /var/pxe/centos7 is mounted or not. If not, either manually mount it or re-run the setup.sh script.
- Check the Linux boot message to see whether eMMC is found or not. If not, the BlueField driver patch is missing. For local installation via RShim, run the setup.sh script with the absolute path and check if there are any errors. For a corporate PXE server, make sure the BlueField and ConnectX driver disk are patched into the initrd image.

9.9.6 How to find the software versions of the running system

Run the following:

```
/opt/mellanox/scripts/bfvcheck:
root@bluefield:/usr/bin/bfvcheck# ./bfvcheck
Beginning version check...
-RECOMMENDED VERSIONS-
ATF: v1.5(release):BL2.0-1-gf9f7cdd
UEFI: 2.0-6004a6b
FW: 18.25.1010
-INSTALLED VERSIONS-
ATF: v1.5(release):BL2.0-1-gf9f7cdd
UEFI: 2.0-6004a6b
```

```
FW: 18.25.1010
Version checked
```

Also, the version information is printed to the console.

For ATF, a version string is printed as the system boots.

```
"NOTICE:  BL2: v1.3(release):v1.3-554-ga622cde"
```

For UEFI, a version string is printed as the system boots.

```
"UEFI firmware (version 0.99-18d57e3 built at 00:55:30 on Apr 13 2018)"
```

For Yocto, run:

```
$ cat /etc/bluefield_version
2.0.0.10817
```

9.9.7 How to upgrade the host RShim driver

See the readme at `<BF_INST_DIR>/src/drivers/rshim/README`.

9.9.8 How to upgrade the boot partition (ATF & UEFI) without re-installation

1. Boot the target through the RShim interface from a host machine:

   ```
   $ cat <BF_INST_DIR>/sample/install.bfb > /dev/rshim<N>/boot
   ```

2. Log into the BlueField target:

   ```
   $ /opt/mlnx/scripts/bfrec
   ```

9.9.9 How to upgrade ConnectX firmware from Arm side

The `mst`, `mlxburn`, and `flint` tools can be used to update firmware.

For Ubuntu, CentOS and Debian, run the following command from the Arm side:

```
sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl
```

9.9.10 How to configure ConnectX firmware

Configuring ConnectX firmware can be done using the mlxconfig tool.

It is possible to configure privileges of both the internal (Arm) and the external host (for DPUs) from a privileged host. According to the configured privilege, a host may or may not perform certain operations related to the NIC (e.g. determine if a certain host is allowed to read port counters).

For more information and examples please refer to the MFT User Manual which can be found at the following link.

9.9.11 How to use the UEFI boot menu

Press the "Esc" key when prompted after booting (before the countdown timer runs out) to enter the UEFI boot menu and use the arrows to select the menu option.

It could take 1-2 minutes to enter the Boot Manager depending on how many devices are installed or whether the EXPROM is programmed or not.

Once in the boot manager:

- "EFI Network xxx" entries with device path "PciRoot..." are ConnectX interface
- "EFI Network xxx" entries with device path "MAC(..." are for the RShim interface and the BlueField OOB Ethernet interface

Select the interface and press ENTER will start PXE boot.

The following are several useful commands under UEFI shell:

```
Shell> ls FS0:                                      # display file
Shell> ls FS0:\EFI                                  # display file
Shell> cls                                          # clear screen
Shell> ifconfig -l                                  # show interfaces
Shell> ifconfig -s eth0 dhcp                        # request DHCP
Shell> ifconfig -l eth0                             # show one interface
Shell> tftp 192.168.100.1 grub.cfg FS0:\grub.cfg    # tftp download a file
Shell> bcfg boot dump                               # dump boot variables
Shell> bcfg boot add 0 FS0:\EFI\centos\shim.efi "CentOS" # create an entry
```

9.9.12 How to Use the Kernel Debugger (KGDB)

The default Yocto kernel has `CONFIG_KGDB` and `CONFIG_KGDB_SERIAL_CONSOLE` enabled. This allows the Linux kernel on BlueField to be debugged over the serial port. A single serial port cannot be used both as a console and by KGDB at the same time. It is recommended to use the RShim for console access (`/dev/rshim0/console`) and the UART port (`/dev/ttyAMA0 or /dev/ttyAMA1`) for KGDB. Kernel GDB over console (KGDBOC) does not work over the RShim console. If the RShim console is not available, there are open-source packages such as KGDB demux and agent-proxy which allow a single serial port to be shared.

There are two ways to configure KGDBOC. If the OS is already booted, then write the name of the serial device to the KGDBOC module parameter. For example:

```
$ echo ttyAMA1 > /sys/module/kgdboc/parameters/kgdboc
```

To attach GDB to the kernel, it must be stopped first. One way to do that is to send a "g" to `/proc/sysrq-trigger` .

```
$ echo g > /proc/sysrq-trigger
```

To debug incidents that occur at boot time, kernel boot parameters must be configured. Add " `kgdboc=ttyAMA1,115200 kgdwait` " to the boot arguments to use UART1 for debugging and force it to wait for GDB to attach before booting.

Once the KGDBOC module is configured and the kernel stopped, run the Arm64 GDB on the host machine connected to the serial port, then set the remote target to the serial device on the host side.

```
<BF_INST_DIR>/sdk/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb <BF_INST_DIR>/
sample/vmlinux

(gdb) target remote /dev/ttyUSB3
Remote debugging using /dev/ttyUSB3
arch_kgdb_breakpoint () at /labhome/dwoods/src/bf/linux/arch/arm64/include/asm/kgdb.h:32
32              asm ("brk %0" : : "I" (KGDB_COMPILED_DBG_BRK_IMM));
(gdb)
```

`<BF_INST_DIR>` is the directory where the BlueField software is installed. It is assumed that the SDK has been unpacked in the same directory.

9.9.13 How to enable/disable SMMU

SMMU could affect performance for certain applications. It is disabled by default and can be modified in different ways.

- Enable/disable SMMU in the UEFI System Configuration
- Set it in `bf.cfg` and push it together with the `install.bfb` (see section "Installing Popular Linux Distributions on BlueField")
- In BlueField Linux, create a file with one line with `SYS_ENABLE_SMMU=TRUE`, then run `bfcfg`.

The configuration change will take effect after reboot. The configuration value is stored in a persistent UEFI variable. It is not modified by OS installation.

See section "UEFI System Configuration" for information on how to access the UEFI System Configuration menu.

9.9.14 How to change the default console of the install image

On UART0:

```
$ echo "console=ttyAMA0 earlycon=pl011,0x01000000 initrd=initramfs" > bootarg
$ <BF_INST_DIR>/bin/mlx-mkbfb --boot-args bootarg \
      <BF_INST_DIR>/sample/ install.bfb
```

On UART1:

```
$ echo "console=ttyAMA1 earlycon=pl011,0x01000000 initrd=initramfs" > bootarg
$ <BF_INST_DIR>/bin/mlx-mkbfb --boot-args bootarg \
      <BF_INST_DIR>/sample/install.bfb
```

On RShim:

```
$ echo "console=hvc0 initrd=initramfs" > bootarg
$ <BF_INST_DIR>/bin/mlx-mkbfb --boot-args bootarg \
      <BF_INST_DIR>/sample/install.bfb
```

9.9.15 How to change the default network configuration during BFB installation

On Ubuntu OS, the default network configuration for `tmfifo_net0` and `oob_net0` interfaces is set by the cloud-init service upon first boot after BFB installation.

The default content of `/var/lib/cloud/seed/nocloud-net/network-config` as follows:

```
# cat /var/lib/cloud/seed/nocloud-net/network-config
version: 2
renderer: NetworkManager
ethernets:
 tmfifo_net0:
  dhcp4: false
  addresses:
   - 192.168.100.2/30
  nameservers:
   addresses: [ 192.168.100.1 ]
  routes:
  - to: 0.0.0.0/0
   via: 192.168.100.1
   metric: 1025
 oob_net0:
  dhcp4: true
```

This content can be modified during BFB installation using `bf.cfg`. For example:

```
# cat bf.cfg
bfb_modify_os()
{
   sed -i -e '/oob_net0/,+1d' /mnt/var/lib/cloud/seed/nocloud-net/network-config
cat >> /mnt/var/lib/cloud/seed/nocloud-net/network-config << EOF
 oob_net0:
  dhcp4: false
  addresses:
   - 10.0.0.1/24
EOF
}

# bfb-install -c bf.cfg -r rshim0 -b <BFB>
```

⚠ Using the same technique, any configuration file on the BlueField DPU side can be updated during the BFB installation process.

9.9.16 Sanitizing DPU eMMC and SSD Storage

During the BFB installation process, DPU storage can be securely sanitized either using the `shred` or the `mmc` and `nvme` utilities in the `bf.cfg` configuration file as illustrated in the following subsections.

⚠ By default, only the installation target storage is formatted using the Linux `mkfs` utility.

9.9.16.1 Using shred Utility

```
# cat bf.cfg
SANITIZE_DONE=${SANITIZE_DONE:-0}
export SANITIZE_DONE
if [ $SANITIZE_DONE -eq 0 ]; then
        sleep 3m
        /sbin/modprobe nvme

        if [ -e /dev/mmcblk0 ]; then
```

```
                echo Sanitizing /dev/mmcblk0 | tee /dev/kmsg
                echo Sanitizing /dev/mmcblk0 > /tmp/sanitize.emmc.log
                mmc sanitize /dev/mmcblk0 >>  /tmp/sanitize.emmc.log 2>&1
        fi
        if [ -e /dev/nvme0n1 ]; then
                echo Sanitizing /dev/nvme0n1 | tee /dev/kmsg
                echo Sanitizing /dev/nvme0n1 > /tmp/sanitize.ssd.log
                nvme sanitize /dev/nvme0n1 -a 2 >> /tmp/sanitize.ssd.log 2>&1
                nvme sanitize-log /dev/nvme0n1 >> /tmp/sanitize.ssd.log 2>&1
        fi
        SANITIZE_DONE=1
        echo ===================== sanitize.log ===================== | tee /dev/kmsg
        cat /tmp/sanitize.*.log | tee /dev/kmsg
        sync
fi
bfb_modify_os()
{
        echo ===================== bfb_modify_os ===================== | tee /dev/kmsg
        if ( /bin/ls -1 /tmp/sanitize.*.log > /dev/null 2>&1 ); then
                cat /tmp/sanitize.*.log > /mnt/root/sanitize.log
        fi
}
```

9.9.16.2 Using mmc and nvme Utilities

```
# cat bf.cfg
SANITIZE_DONE=${SANITIZE_DONE:-0}
export SANITIZE_DONE
if [ $SANITIZE_DONE -eq 0 ]; then
        sleep 3m
        /sbin/modprobe nvme

        if [ -e /dev/mmcblk0 ]; then
                echo Sanitizing /dev/mmcblk0 | tee /dev/kmsg
                echo Sanitizing /dev/mmcblk0 > /tmp/sanitize.emmc.log
                mmc sanitize /dev/mmcblk0 >>  /tmp/sanitize.emmc.log 2>&1
        fi
        if [ -e /dev/nvme0n1 ]; then
                echo Sanitizing /dev/nvme0n1 | tee /dev/kmsg
                echo Sanitizing /dev/nvme0n1 > /tmp/sanitize.ssd.log
                nvme sanitize /dev/nvme0n1 -a 2 >> /tmp/sanitize.ssd.log 2>&1
                nvme sanitize-log /dev/nvme0n1 >> /tmp/sanitize.ssd.log 2>&1
        fi
        SANITIZE_DONE=1
        echo ===================== sanitize.log ===================== | tee /dev/kmsg
        cat /tmp/sanitize.*.log | tee /dev/kmsg
        sync
fi
bfb_modify_os()
{
        echo ===================== bfb_modify_os ===================== | tee /dev/kmsg
        if ( /bin/ls -1 /tmp/sanitize.*.log > /dev/null 2>&1 ); then
                cat /tmp/sanitize.*.log > /mnt/root/sanitize.log
        fi
}
```

9.9.17 How to perform graceful shutdown

Before powering off or power cycling the DPU, it is strongly recommended to perform graceful shutdown of the DPU Arm OS.

Graceful shutdown of the Arm OS ensures that data within the eMMC/NVMe cache is properly written to storage, and helps prevent filesystem inconsistencies and file corruption.

There are several ways to gracefully shutdown the DPU Arm OS:

- Log into the DPU Arm OS and perform a shutdown command prior to power cycling the host server. For example:

  ```
  sudo shutdown -h now
  ```

- Assuming the DPU BMC can issue NC-SI OEM commands to the DPU:
  a. Issue the Shutdown Smart NIC OS NC-SI OEM command.
  b. After DPU Arm OS shutdown, it is recommended to issue DPU Arm OS state query which indicates whether DPU Arm OS shutdown has completed (`standby` indication). This can be done by issuing the Get Smart NIC OS State NC-SI OEM command.

10 Windows Support

10.1 Network Drivers

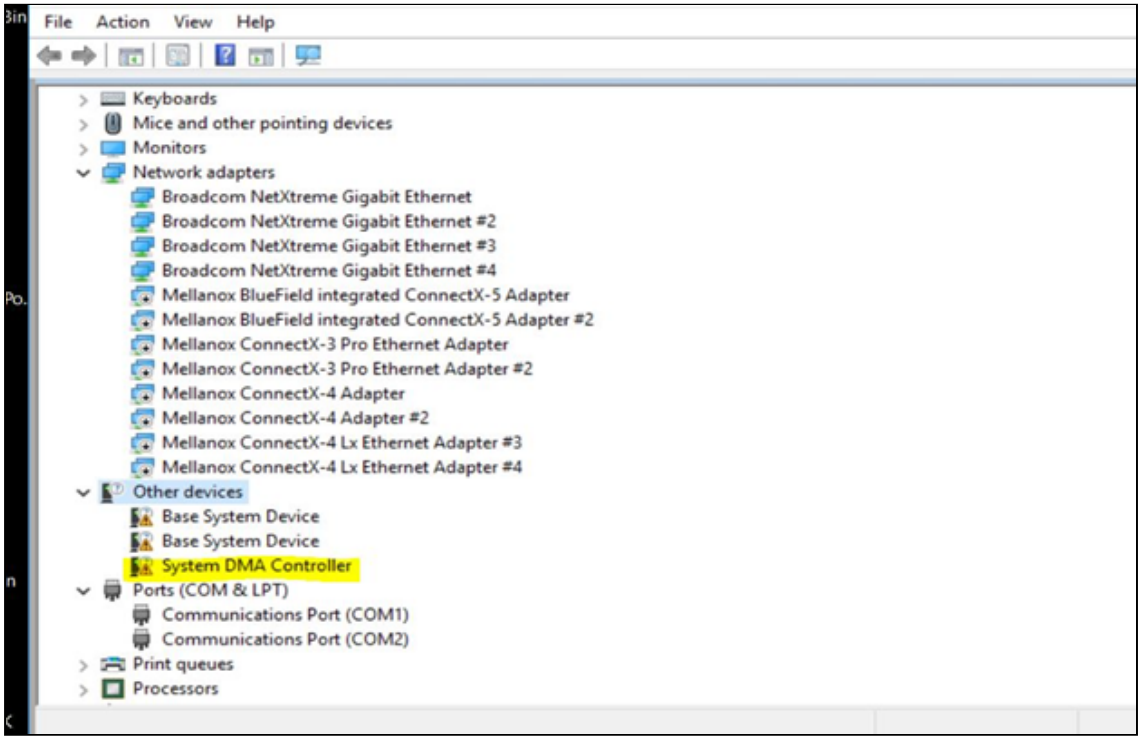
BlueField Windows support from the host-side is facilitated by the WinOF-2 driver. For more information on WinOF-2 (including installation), please refer to the WinOF-2 Documentation.

10.2 RShim Drivers

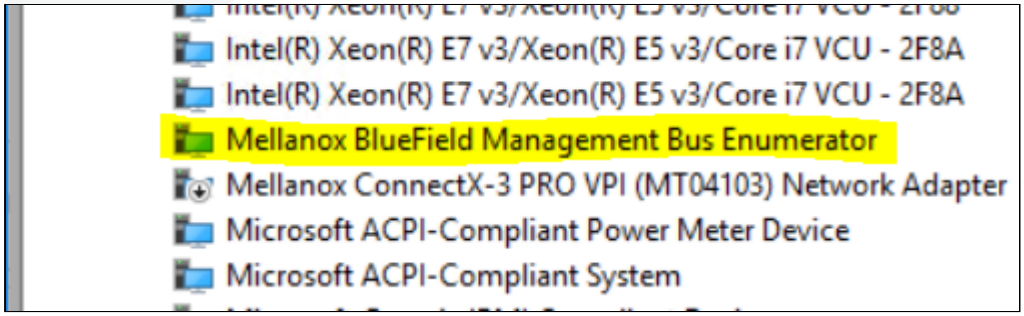
RShim drivers provide functionalities like resetting the Arm cores, pushing a bootstream image, as well as some networking and console functionalities.

10.3 Verifying RShim Drivers Installation

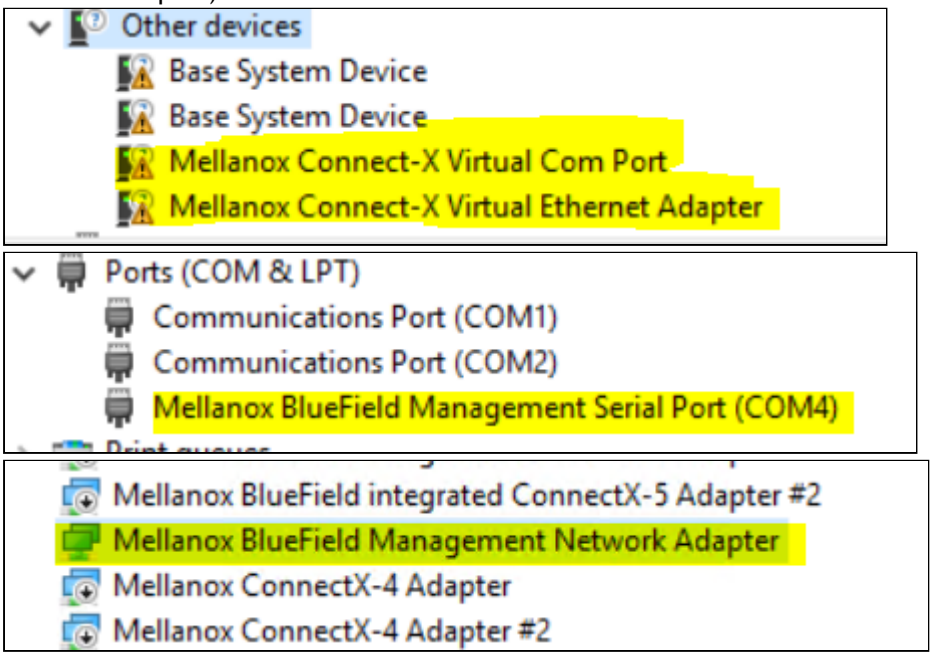
1. Open the Device Manager when no drivers are installed to make sure a new PCIe device is available as below.



2. Run the installer to install all 3 drivers (`MlxRshimBus.sys` , `MlxRshimCom.sys` , and `MlxRshimEth.sys`).



3. Make sure the Bus driver created 2 child devices after the installation (Com port and the Ethernet adapter).



At this time, PuTTY application or any other network utility can be used to communicate with BlueField via Virtual Com Port or Virtual Ethernet Adapter (ssh). The Com Port can be used using the 9600 baud-rate and default settings.

> ⚠ RShim drivers can be connect via PCIe (the drivers we are providing) or via USB (external connection) but not both at the same time. So when the bus driver detects that an external USB is already attached, it will not create the child virtual devices for data access. Access via PCIe is available once the USB connection is removed.

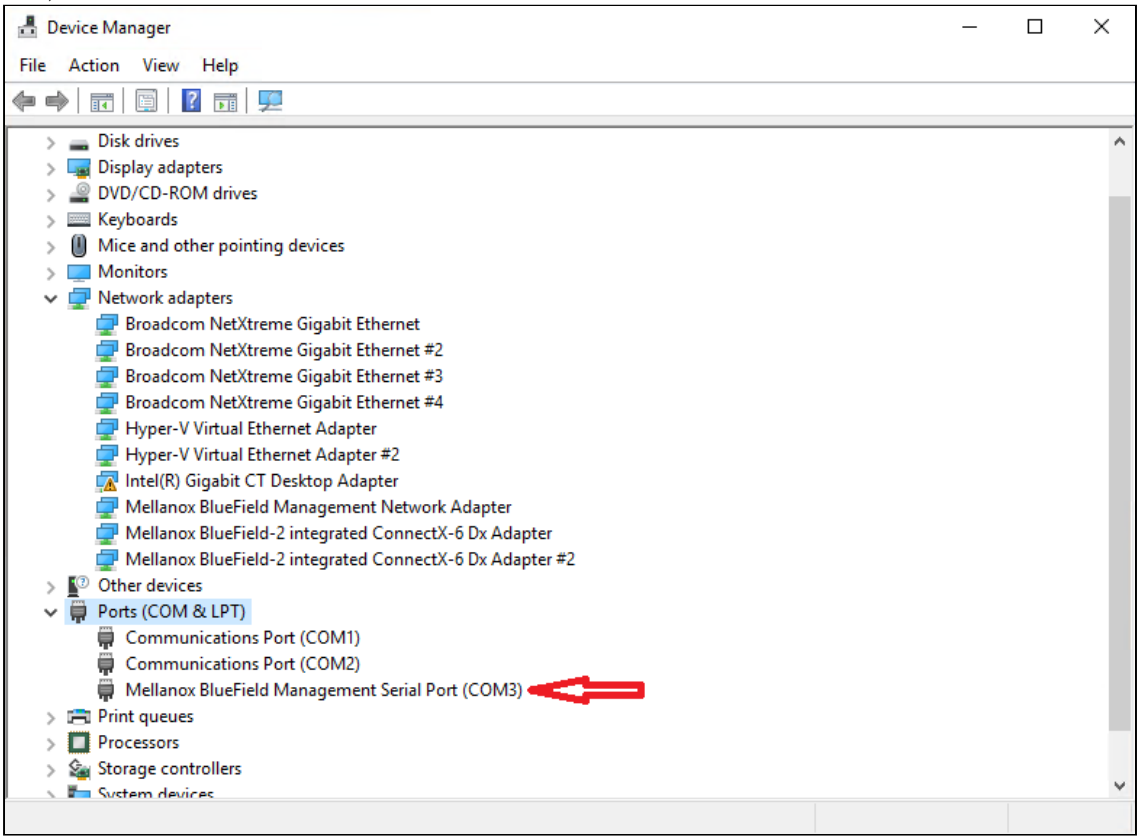
10.4 Accessing BlueField From Host

BlueField can be accessed via PuTTY or any other network utility application to communicate via virtual COM or virtual Ethernet adapter. To use COM:

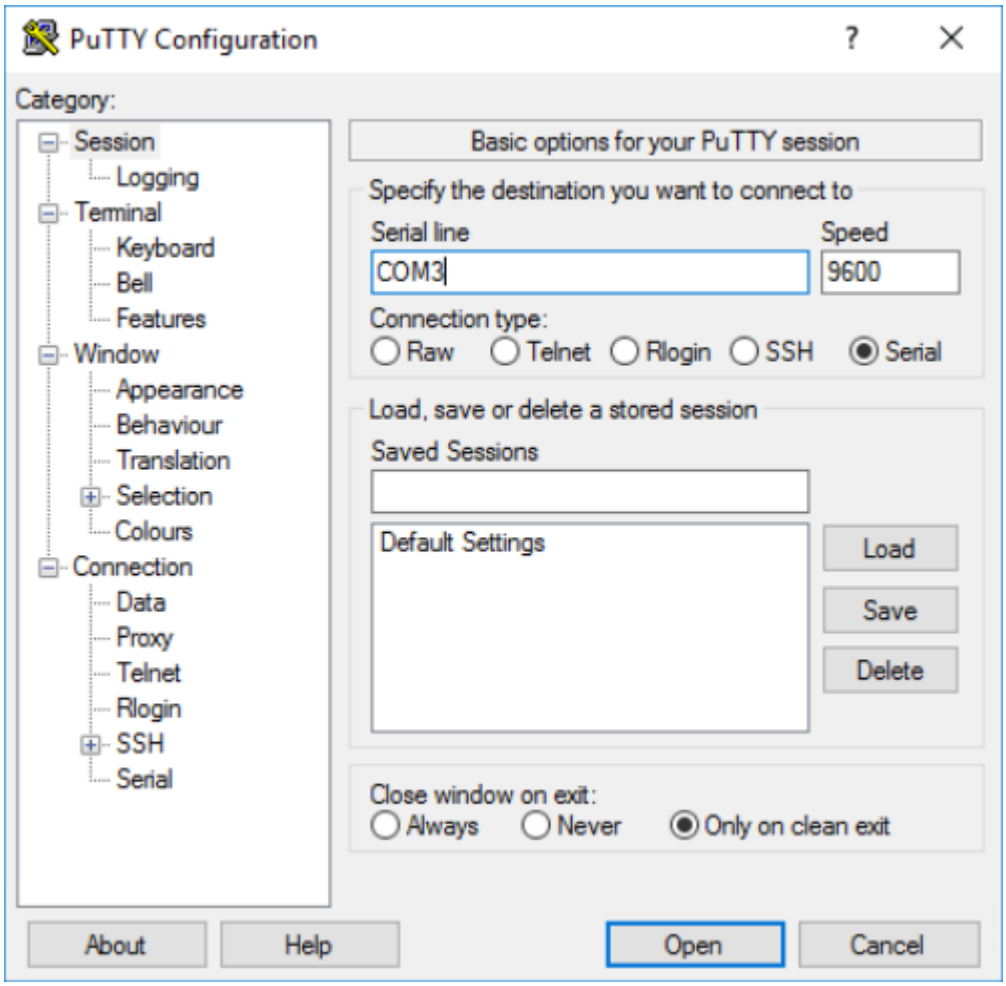
1. Open Putty.
2. Change connection type to Serial.
3. Run the following command in order to know what to set the "Serial line" field to:

```
C:\Users\username\Desktop> reg query HKLM\HARDWARE\DEVICEMAP\SERIALCOMM | findstr MlxRshim
    \MlxRshim\COM3      REG-SZ      COM3
```

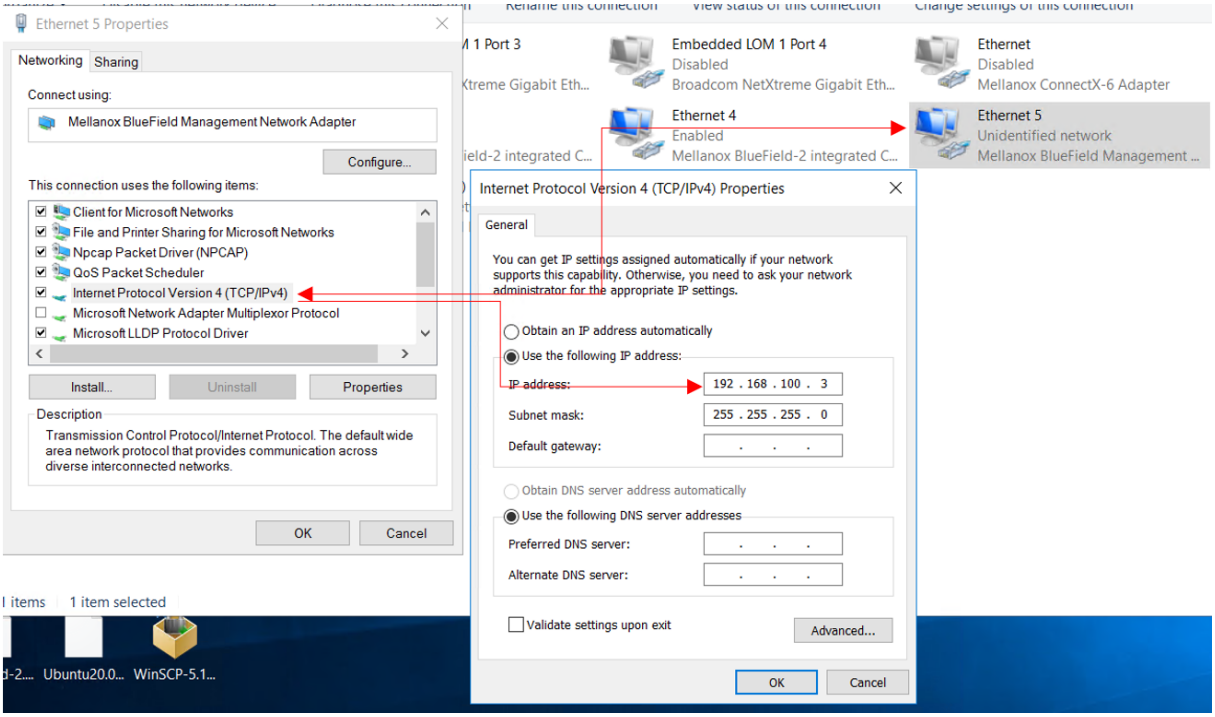
In this case use COM3. This name can also be found via Device Manager under "Ports (Com & LPT)".



4. Press Open and hit Enter.



To access via BlueField management network adapter, configure an IP address as shown in the example below and run a ping test to confirm configuration.



10.5 RShim Ethernet Driver

The device does not support any type of stateful or stateless offloads. This is indicated to the Operating System accordingly when the driver loads. The MAC address is a pre-defined MAC address (CA-FE-01-CA-FE-02). The following registry keys can be used to change basic settings such as MAC address.

Registry Name	Description	Valid Values
HKLM\SYSTEM\CurrentControlSet\Control\Class\{4d36e972-e325-11ce-bfc1-08002be10318}\<nn>*JumboPacket	The size, in bytes, of the largest supported Jumbo Packet (an Ethernet frame that is greater than 1514 bytes) that the hardware can support.	1514 (default) - 2048
HKLM\SYSTEM\CurrentControlSet\Control\Class\{4d36e972-e325-11ce-bfc1-08002be10318}\<nn>*NetworkAddress	The network address of the device. The format for a MAC address is: XX-XX-XX-XX-XX-XX.	CA-FE-01-CA-FE-02 (default)
HKLM\SYSTEM\CurrentControlSet\Control\Class\{4d36e972-e325-11ce-bfc1-08002be10318}\<nn>\ReceiveBuffers	The number of receive descriptors used by the miniport adapter.	16 – 64 (Default)

Update the MAC address manually using registry key if there are more than one BlueField in the system.

For instructions on how to find interface index in the registry (nn), please refer to section "Finding the Index Value of the Network Interface" in the WinOF-2 User Manual under Features Overview and Configuration > Configuring the Driver Registry Keys.

10.6 MlxRshimBus Driver

This driver does all the read/write work to the hardware registers. User space application can send down IOCTL's to restart the system on chip or to push a new BlueField boot stream image.

10.7 RshimCmd Tool

RshimCmd is a command line tool that enables the user to:

- Restart BlueField.
- Push a boot stream file (`.bfb`). A BFB file is a generated BlueField boot stream file that contains Linux operating system image that runs on BlueField. BFB files can be downloaded from the NVIDIA DOCA SDK webpage.

Usage	`RshimCmd -RestartSmartNic <Option> -BusNum <BusNum>`

Example	`RshimCmd -EnumDevices`
	`RshimCmd -PushImage c:\bin\MlnxBootImage.bfb -BusNum 11`
	`RshimCmd -RestartSmartNic 1 -BusNum 11`

Detailed Usage	`RshimCmd -h`

The BFB image can be either CentOS or Ubuntu. Ubuntu credentials are: `ubuntu` / `ubuntu` and for Centos credentials are: root/centos, IP address of RShim Ethernet component (called tmfifo_net0) on the BlueField side is `192.168.100.2/30` by default. Please set IP address on the Windows side accordingly to be able to communicate via SSH.

10.8 BlueField UEFI System Boot Customizations during Installation

Bluefield's UEFI system boot options and more can be customized during the BFB Installation through the use of configuration parameters in the `bf.cfg` file. For further information on the `bf.cfg` file, refer to the BlueField Documentation.

To include the `bf.cfg` file into the BFB installation, append the file to BFB file as described below:

1. Copy the BFB file to a local folder. For example:

```
Copy <path>\DOCA_1.4.0_BSP_3.9.2_Ubuntu_20.04-5.20220707.bfb c:\bf\MlnxBootImage.bfb
```

2. Append the bf.cfg file into the BFB file.

```
Cd c:\bf
Copy /b MlnxBootImage.bfb + bf.cfg MlnxBootImage_with_bf_cfg.bfb
```

3. Download the BFB image.

```
RshimCmd -PushImage c:\bf\MlnxBootImage_with_bf_cfg.bfb -BusNum 11
```

As the `bf.cfg` is intended for Linux OSes, it should be created according to Linux rules. For example, the lines of this text file should end in LF and not in CR/LF as accepted in Windows.

All the syntax should be as the accepted by the OS expects. For example, there should be no spaces in the middle of "set" statements: `NET_RSHIM_MAC=00:1a:ca:ff:ff:05`.

10.9 EventLogs and Driver Logging

All driver logging is part of the Mellanox-WinOF2-Kernel trace session that comes with the network drivers installation. The default location to the trace is at `%SystemRoot%\system32\LogFiles\Mlnx\Mellanox-WinOF2-System.etl`.

The following are the Event logs RShim drivers generate:

10.9.1 MlxRShimBus Driver

Event ID	Severity	Message
2	Informational	RShim Bus driver loaded successfully
3	Informational	Device successfully stopped
4	Error	The SmartNIC seems to be stuck as the boot FIFO data is not being drained.
5	Error	Driver startup failed due to failure in creation of the child device.
6	Error	SmartNIC is in a bad state. Please restart SmartNIC and reload bus drivers. Please refer to user manual on how to restart SmartNIC.
7	Warning	SmartNIC is in LiveFish mode
8	Warning	Failed creating child virtual devices as a backend USB device is attached and accessing RShim FIFO. Please refer to user manual for more details.

10.9.2 MlxRShim Serial Driver

Event ID	Severity	Message
2	Informational	RShim serial driver loaded successfully
3	Informational	device successfully stopped

10.9.3 MlxRShim Ethernet Driver

Event ID	Severity	Message
2	Error	MAC address read from registry is not supported. Please set valid unicast address.
3	Informational	Device is successfully stopped
4	Warning	Value read from registry is invalid. Therefore use the default value.
5	Error	SmartNIC seems stuck as transmit packets are not being drained.
6	Informational	RShim Ethernet driver loaded successfully

11 Document Revision History

11.1 Rev 4.5.1 – March 01, 2024

N/A

11.2 Rev 4.5.0 – December 12, 2023

Added:

- Section "Updating Software Using Redfish"
- Section "Sanitizing DPU eMMC and SSD Storage"
- Section "How to perform graceful shutdown"
- Section "BFB installation monitoring"

Updated:

- Page "Updating DPU Software Packages Using Standard Linux Tools"
- Section "RShim Logging"
- Section "NIC Mode"
- Section "Enabling OVS-DPDK Hardware Offload"
- Section "Enabling IPsec Packet Offload"
- Section "Setting IPSec Packet Offload Using strongSwan"
- Section "Running strongSwan Example"
- Section "Building strongSwan"
- Section "IPsec Packet Offload and OVS Offload"

11.3 Rev 4.2.2 – October 24, 2023

Updated:

- Section "NIC Mode"

11.4 Rev 4.2.0 – August 10, 2023

Updated:

- Step 3 under section "PXE Server Preparations"
- Section "Removing Previously Installed DOCA Runtime Packages"
- Section "NIC Mode"
- Sections "Connection Tracking With NAT" and "Querying Connection Tracking Offload Status" with conntack command for Ubuntu 22.04 kernels
- Section "LAG Configuration"
- Section "SystemD Service"
- Page "QoS Configuration"
- Section "bf.cfg Parameters"

11.5 Rev 4.0.2 – May 08, 2023

Added:

- Page "SoC Management Interface"
- Page "Legal Notices and 3rd Party Licenses"
- Section "Unable to load BL2, BL2R, or PSC image"

Updated:

- Section "Default Ports and OVS Configuration" with new step 2
- Section "BlueField Linux Drivers" with `gpio-mlxbf3`, `mlxbf-ptm`, `pwr-mlxbf`, and `pinctrl-mlxbf`
- Page "Updating DPU Software Packages Using Standard Linux Tools"
- Page "UEFI Secure Boot"
- Section "IPsec Hardware Offload: Full Offload" with Canonical note
- Section "How to upgrade ConnectX firmware from Arm side"
- Section "VirtIO-net PF Device Configuration" by removing `ECPF_ESWITCH_MANAGER` and `ECPF_PAGE_SUPPLIER` from step 4
- Section "Virtio-net SR-IOV VF Device Configuration" by removing `ECPF_ESWITCH_MANAGER` and `ECPF_PAGE_SUPPLIER` from step 7.b
- Section "vDPA over VirtIO Full Emulation"

11.6 Rev 3.9.3 – November 02, 2022

Added:

- Section "DHCP Client Configuration"
- Section "Updating DPU Software Packages Using Standard Linux Tools"
- Section "Creating Transitional Hotplug VirtIO-net PF Device"
- Section "Transitional VirtIO-net VF Device Support"

Updated:

- Section "Upgrading Boot Software" by specifying that the "Reset EFI Variables" action also wipes the BOOT option variables and secure boot keys
- Section "BlueField Linux Drivers"
- Section "Configuring Uplink MTU"
- Section "Disabling Host Networking PFs" by adding instructions for reactivating host networking for single-port DPUs
- Section "Configuring RegEx Acceleration on BlueField-2"
- Section "Virtio-net SR-IOV VF Device Configuration"
- `PXE_DHCP_CLASS_ID` in section "bf.cfg Parameters"

Removed:

- Step 7 in section "Configuring Host Server Side"
- Separated Mode from "Modes of Operation"

11.7 Rev 3.9.2 – August 02, 2022

Added:

- Section "Updating NVConfig Params"
- Page "System Configuration and Services"
- Section "Enrolling New NVIDIA Certificates"
- Section "bf.cfg Parameters"
- Support for OpenSSL version 3.0.2 in section "PKA Use Cases"
- Section "How to change the default network configuration during BFB installation"

Updated:

- Section "Firmware Upgrade"
- Section "Customizations During BFB Installation"
- Section "UEFI System Configuration"
- Page "Host-side Interface Configuration"
- Section "Enrolling Certificates Using Capsule"
- Section "NIC Mode" with supported MLNX_OFED versions
- Section "PKA Use Cases" with support for OpenSSL version 3.0.2

11.8 Rev 3.9 – May 03, 2022

Added:

- Section "GRUB Password Protection"
- New note under step 2 in section "Default Ports and OVS Configuration"
- Section "BlueField Linux Drivers"
- Canonical db certificate to section "Existing DPU Certificates"
- New note under section "Enrolling Certificates Using Capsule"
- New power cycle note under section "Enabling Host Restriction"
- New power cycle note under section "Disabling Host Restriction"
- Section "NIC Mode"
- Section "LAG on Multi-host"
- New power cycle note under section "Disabling Host Networking PFs"
- Section "PKA Prerequisites"
- Section "OVS IPsec"
- Section "Rate Limiting VF Group"
- Note to section "User Frontend"
- Section "Controller Live Update"

Updated:

- Code block in section "Customizations During BFB Installation"
- Section "Building Your Own BFB Installation Image"
- Section "Configuring VXLAN Tunnel"
- Step 2 in section "Prerequisites"
- Section "Enabling IPsec Full Offload"
- Code block under step 1 in section "LAG Configuration"

11.9 Rev 3.8.5 – January 19, 2022

Added:

- Section "Another backend already attached"

Updated:

- Section "Ensure RShim Running on Host"

12 Legal Notices and 3rd Party Licenses

DPU Software Components	Version	3rd Party Components and Licenses
DOCA SDK	2.5.1	Link
DOCA SDK 3rd Party Notice		Link
DOCA SDK 3rd Party Unify Notice		Link
SoC OS Linux Ubuntu 22.04 Distro	5.15.0-1032-bluefield	Link
SoC OS Linux Ubuntu 20.04 Distro	5.4.0-1076-bluefield	Link
BSP – ATF	4.5.1	Link
BSP – ATF 3rd Party Notice		Link
BSP – ATF 3rd Party Unify Notice		Link
BSP – UEFI (EDK2)	4.5.1	Link
BlueField UEFI (EDK2) 3rd Party Notice		Link
BlueField UEFI (EDK2) 3rd Party Unify Notice		Link
BlueField BMC	23.10-7	Link
BlueField BMC 3rd Party Notice		Link
BlueField BMC 3rd Party Unify Notice		Link
Virtio Network Controller	1.7.13	Link
Virtio Network Controller 3rd Party Notice		Link
Virtio Network Controller 3rd Party Unify Notice		Link
MLNX LibSnap and virtio-blk	1.6.0-1	Link
MLNX LibSnap and virtio-blk 3rd Party Notice		Link
MLNX SNAP and SPDK	3.8.0-1	Link
MLNX SNAP and SPDK 3rd Party Notice		Link
NVIDIA MLNX_OFED License	23.10-4	Link
NVIDIA MLNX_OFED 3rd Party Unify Notice		Link
NVIDIA MFT License	4.27.0	Link
NVIDIA MFT 3rd Party Notice		Link
NVIDIA MLNX_DPDK	22.11.2310.2.0	Link
NVIDIA MLNX_DPDK 3rd Party Notice		Link
NVIDIA MLNX_DPDK 3rd Party Unify Notice		Link

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. Neither NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: “NVIDIA”) make any representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA