



NVIDIA BLUEFIELD DPU FAMILY SOFTWARE V3.5.1.11601 DOCUMENTATION

Software version v3.5.1.11601

Table of Contents

| | |
|---|-----------|
| Welcome to NVIDIA BlueField DPU Family Software documentation! | 9 |
| Intended Audience | 9 |
| Software Download | 9 |
| Technical Support | 9 |
| Glossary | 9 |
| Related Documentation | 12 |
| Release Notes | 14 |
| Changes and New Features | 14 |
| Changes and New Features in 3.5.1.11601 | 14 |
| Supported Platforms and Interoperability | 14 |
| Supported NVIDIA BlueField-2 DPU Platforms | 14 |
| Supported NVIDIA BlueField DPU Platforms | 16 |
| Embedded Software | 17 |
| Tested SSDs | 18 |
| Supported DPU Linux Distributions (aarch64) | 18 |
| Supported DPU Host OS Distributions | 19 |
| Supported Open vSwitch | 19 |
| Known Issues | 19 |
| Bug Fixes In This Version | 24 |
| BlueField Software Overview | 25 |
| Debug Tools | 25 |
| BlueField Adapter/SmartNIC | 25 |
| BlueField-based Storage Appliance | 26 |
| BlueField Architecture | 26 |
| Installation and Initialization | 27 |
| Unpacking BlueField Software Distribution With Yocto | 27 |
| System Connections | 28 |
| RShim Host Driver | 29 |
| Upgrading Boot Software | 36 |
| Building UEFI (EDK2) | 46 |
| Building Poky Initramfs | 48 |
| Using Yocto as a Cross-compilation SDK and Root Filesystem Generator | 50 |

| | |
|--|-----------|
| BlueField-2 OOB Ethernet Interface | 50 |
| OpenOCD on BlueField | 54 |
| UEFI Boot Option Management | 54 |
| Boot Option | 55 |
| List UEFI Boot Options | 55 |
| Creating, Deleting, and Modifying UEFI Boot Option | 56 |
| UEFI System Configuration | 57 |
| Installing Popular Linux Distributions on BlueField | 57 |
| Building Your Own BFB Installation Image..... | 57 |
| Installing Official CentOS 7.x Distribution | 59 |
| Running RedHat on BlueField | 71 |
| Installing Reference Yocto Distribution..... | 72 |
| Installing Ubuntu on BlueField | 74 |
| Installing Debian on BlueField | 75 |
| Upgrading NVIDIA BlueField DPU Software | 75 |
| Hardware Prerequisites | 75 |
| Software Prerequisites | 76 |
| Image Installation..... | 78 |
| Post-installation Procedure..... | 78 |
| Windows Support..... | 79 |
| BlueField Management | 81 |
| Performance Monitoring Counters..... | 81 |
| Programming Counter to Monitor Event | 81 |
| List of Supported Events | 83 |
| SNMP Subagent | 89 |
| Intelligent Platform Management Interface (IPMI) | 91 |
| BMC Retrieving Data from BlueField via IPMB | 91 |
| Loading and Using IPMI on BlueField Running CentOS..... | 95 |
| Retrieving Data from BlueField Via OOB/ConnectX Interfaces | 97 |
| BlueField Retrieving Data From BMC Via IPMB..... | 97 |
| Changing I2C Addresses | 99 |
| RShim Logging | 99 |
| IPMI Logging in UEFI | 103 |
| SEL Record Format..... | 103 |

| | |
|---|------------|
| Possible SEL Field Values..... | 104 |
| Event Definitions..... | 104 |
| Reading IPMI SEL Log Messages | 105 |
| DPU Operation..... | 107 |
| DPU Bring-Up and Driver Installation | 107 |
| Installing Linux on DPU..... | 107 |
| MLNX_OFED Installation..... | 117 |
| eMMC Backup and Restore..... | 121 |
| Local Yum Repository Setup and Usage..... | 124 |
| Functional Diagram | 126 |
| Modes of Operation..... | 127 |
| Separated Host | 128 |
| Embedded CPU Function Ownership Mode | 128 |
| Restricting DPU Host | 129 |
| Kernel Representors Model | 130 |
| Multi-Host | 131 |
| Representors..... | 132 |
| Virtual Switch on BlueField DPU..... | 134 |
| Verifying Host Connection on Linux | 134 |
| Verifying Host Connection on Windows..... | 135 |
| Enabling OVS HW Offloading | 135 |
| Enabling OVS-DPDK Hardware Offload..... | 136 |
| Configuring DPDK and Running TestPMD | 136 |
| Flow Statistics and Aging | 136 |
| Connection Tracking Offload | 136 |
| Offloading VLANs..... | 139 |
| VXLAN Tunneling Offload | 139 |
| GRE Tunneling Offload | 140 |
| Geneve Tunneling Offload | 141 |
| Using TC Interface to Configure Offload Rules | 142 |
| VirtIO Acceleration Through Hardware vDPA | 143 |
| Configuring Uplink MTU | 143 |
| BlueField Link Aggregation | 143 |
| Prerequisites..... | 144 |

| | |
|---|------------|
| LAG Configuration | 144 |
| Mediated Devices | 145 |
| Related Configuration | 145 |
| RDMA Stack Support on Host and Arm System | 145 |
| Separate Host Mode | 145 |
| Embedded CPU Mode | 146 |
| Controlling Host PF and VF Parameters | 146 |
| Setting Host PF and VF Default MAC Address | 146 |
| Setting Host PF and VF Link State | 146 |
| Query Configuration | 146 |
| DPDK on BlueField SmartNIC | 147 |
| NVMe SNAP on BlueField SmartNIC | 147 |
| RegEx Acceleration | 147 |
| Configuring RegEx Acceleration on BlueField-2 | 147 |
| Public Key Acceleration | 148 |
| IPsec Functionality | 148 |
| Transparent IPsec Encryption and Decryption | 148 |
| IPsec Hardware Offload: Crypto Offload | 149 |
| IPsec Hardware Offload: Full Offload | 149 |
| QoS Configuration | 156 |
| Rate Limiting Host PF and VF | 156 |
| VirtIO-net Emulated Devices | 156 |
| VirtIO-net PF Devices | 156 |
| Virtio-net SR-IOV VF Devices | 158 |
| Deep Packet Inspection | 160 |
| Controller Card Operation | 161 |
| Bring-Up and Driver Installation | 161 |
| BlueField Software | 161 |
| Software On eMMC | 162 |
| PXE Server Configuration on Host Side | 162 |
| Installing Linux on BlueField Controller Card | 163 |
| MLNX_OFED Installation | 172 |
| Windows Support | 176 |
| Network Drivers | 176 |

| | |
|--|------------|
| RShim Drivers | 176 |
| Accessing BlueField DPU From Host..... | 176 |
| RShim Ethernet Driver | 179 |
| Troubleshooting | 180 |
| RShim Troubleshooting | 180 |
| RShim driver is not loading..... | 180 |
| HowTo upgrade the host RShim driver..... | 180 |
| Connectivity Troubleshooting | 181 |
| Connection (ssh, screen console) to the BlueField is lost | 181 |
| Driver not loading in host server | 182 |
| No connectivity between network interfaces of source host to destination device..... | 182 |
| Uplink in Arm down while uplink in host server up | 183 |
| Performance Troubleshooting | 183 |
| Degradation in performance | 183 |
| PCIe Troubleshooting | 184 |
| Insufficient power on the PCIe slot error | 184 |
| HowTo update PCIe device description..... | 184 |
| HowTo handle two BlueField DPU devices in the same server | 184 |
| SR-IOV Troubleshooting | 184 |
| Unable to create VFs | 184 |
| No traffic between VF to external host | 185 |
| eSwitch Troubleshooting | 185 |
| Unable to configure legacy mode | 185 |
| Arm appears as two interfaces | 186 |
| Isolated Mode Troubleshooting..... | 186 |
| Unable to burn FW from host server | 186 |
| HowTo upgrade ConnectX firmware from Arm side..... | 187 |
| General Troubleshooting..... | 187 |
| Server unable to find the DPU | 187 |
| DPU no longer works | 187 |
| DPU stopped working after installing another BFB | 187 |
| Link indicator light is off | 187 |
| Link light is on but no communication is established | 187 |

| | |
|--|------------|
| Installation Troubleshooting..... | 188 |
| BlueField target is stuck inside UEFI menu | 188 |
| BFB does not recognize the BlueField board type | 188 |
| CentOS fails into "dracut" mode during installation | 188 |
| How to find the software versions of the running system | 188 |
| How to upgrade the host RShim driver | 189 |
| How to upgrade the boot partition (ATF & UEFI) without re-installation | 189 |
| How to upgrade ConnectX firmware from Arm side | 189 |
| How to configure ConnectX firmware | 189 |
| How to use the UEFI boot menu | 190 |
| How to get installation images from the BlueField release tarball | 190 |
| How to install Yocto | 190 |
| How to install CentOS | 190 |
| How to Use the Kernel Debugger (KGDB)..... | 191 |
| How to enable/disable SMMU | 191 |
| How to change the default console of the install image | 192 |
| Document Revision History | 193 |
| Rev 3.5.1 - February 05, 2021 | 193 |
| Rev 3.5 - January 25, 2021 | 193 |
| Rev 3.1.0 - September 17, 2020 | 193 |
| Rev 2.5.1 - April 22, 2020..... | 194 |
| Rev 2.5.0 - March 03, 2020 | 194 |
| Rev 2.4.0 - December 13, 2019 | 195 |
| Rev 2.2.0 - September 29, 2019 | 195 |
| Rev 2.1.0 - July 29, 2019 | 195 |
| Release Notes Change Log History | 197 |
| Changes and New Features in 3.5.0.11563 | 197 |
| Changes and New Features in 3.1.0.11424 | 197 |
| General Changes | 197 |
| SmartNIC Changes | 197 |
| Changes and New Features in 2.5.1.11213 | 198 |
| SmartNIC Changes | 198 |
| Changes and New Features in 2.5.0.11176 | 198 |
| General Changes | 198 |

| | |
|---|------------|
| SmartNIC Changes | 198 |
| Changes and New Features in 2.4.0.11082 | 198 |
| General Changes | 198 |
| SmartNIC Changes | 198 |
| Changes and New Features in 2.2.0.11000 | 199 |
| General Changes | 199 |
| SmartNIC Changes | 199 |
| Changes and New Features in 2.1.0.10924 | 199 |
| General Changes | 199 |
| SmartNIC Changes | 199 |
| Changes and New Features in 2.0.1.10841 | 199 |
| General Changes | 199 |
| SmartNIC Changes | 200 |
| Changes and New Features in 1.2.0.10639 | 200 |
| General Changes | 200 |
| BlueField Reference Platform Changes..... | 200 |
| SmartNIC Changes | 201 |
| Bug Fixes History..... | 202 |

Welcome to NVIDIA BlueField DPU Family Software documentation!

NVIDIA® BlueField® DPU family software is a reference Linux distribution based on the Ubuntu Server and the Yocto Poky distribution extended to include DOCA runtime libraries, the Mellanox OFED stack for Arm and a Linux kernel that supports various accelerations for storage, networking, and security. As such, customers can run any Linux-based applications in the BlueField software environment seamlessly.

These pages provide product release notes as well as information that explains the BlueField Software Distribution (BSD) and how to develop and/or customize applications, system software, and file system images for the BlueField platform.

Intended Audience

This product 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.

Software Download

To download product software, please refer to the [DOCA SDK](#) developer zone.

Technical Support

Customers who purchased products directly from NVIDIA are invited to contact us on the [NVIDIA Support](#) website.

Glossary

| Abbreviation / Acronym | Whole Word / 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 |
| BFB | BlueField bootstream |
| BSD | BlueField Software Distribution |
| BUF | Buffer |
| CHI | Coherent Hub Interface; Arm® protocol used over the BlueField Skymesh specification |
| CL | Cache line |

| Abbreviation / Acronym | Whole Word / Description |
|------------------------|---|
| CMDQ | Command queue |
| CMO | Cache maintenance operation |
| COB | Collision buffer |
| DAT | Data |
| DMA | Direct memory access |
| DOCA | DPU SDK |
| DPI | Deep packet inspection |
| DPU | Data Processing Unit, the third pillar of the data center with CPU and GPU |
| DVM | Distributed virtual memory |
| ECPF | Embedded CPU Physical Function |
| 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 |
| FS | File system |
| FW | Firmware |
| GDB | GNU debugger |
| GPT | GUID partition table |
| HNF | Home node interface |
| Host | <p>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".</p> <ul style="list-style-type: none"> • 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 |
| IB | InfiniBand |
| ICM | Interface Configuration Memory |
| IPMB | Intelligent Platform Management Bus |
| IPMI | Intelligent Platform Management Interface |
| KGDB | Kernel debugger |
| KGDBOC | Kernel debugger over console |
| LAT | Latency |
| LCRD | Link credit |
| MDEV | Mediated device |
| MSS | Memory subsystem |
| MST | Mellanox Software Tools |

| Abbreviation / Acronym | Whole Word / Description |
|------------------------|---|
| NAT | Network address translation |
| NIC | Network interface card |
| OCD | On-chip debugger |
| OOB | Out-of-band |
| OVS | Open vSwitch |
| PCIe | PCI Express; Peripheral Component Interconnect Express |
| PF | Physical function |
| PK | Public key |
| PKA | Public key accelerator |
| POC | Point of coherence |
| RD | Read |
| RegEx | Regular expression |
| REQ | Request |
| RES | Response |
| 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 |
| RShim | Random Shim |
| RX | Receive |
| SBSA | Server Base System Architecture |
| SDK | Software development kit |
| SF | Sub-function |
| SNP | Snooping |
| SR-IOV | Single Root IO Virtualization |
| STL | Stall |
| TBU | Translation Buffer Unit |
| TRB | Trail buffer |
| TSO | Total store order |
| TX | Transmit |
| UEFI | Unified Extensible Firmware Interface |
| UPVS | UEFI Persistent Variable Store |
| VF | Virtual function |
| VM | Virtual machine |
| VPI | Virtual Protocol Interconnect |

| Abbreviation / Acronym | Whole Word / Description |
|------------------------|--------------------------|
| VST | Virtual Switch Tagging |
| WR | Write |
| WRDB | Write data buffer |

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 Mellanox Firmware Tools Release Notes and User Manual |
| NVIDIA Mellanox 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 Mellanox WinOF Release Notes and User Manual |
| NVIDIA Mellanox BlueField DPU Software Quick Start Guide | This document provides procedure to get started with your NVIDIA BlueField DPU |
| NVIDIA BlueField-2 Ethernet 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 the BlueField-2 DPU |
| NVIDIA BlueField Ethernet DPU User Manual | This document provides details as to the interfaces of the board, specifications, required software and firmware for operating the BlueField SmartNIC, hardware installation, driver installation and bring-up instructions |
| 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 Mellanox NVMe 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. |


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.

- [Changes and New Features](#)
- [Supported Platforms and Interoperability](#)
- [Known Issues](#)
- [Bug Fixes In This Version](#)

Changes and New Features

 For an archive of changes and features from previous releases, please refer to [Release Notes Change Log History](#).

 Currently, NVIDIA® BlueField®-2 DPU supports configuring network ports as either Ethernet only or InfiniBand only.

Changes and New Features in 3.5.1.11601

- [Bug fixes](#)

Supported Platforms and Interoperability

Supported NVIDIA BlueField-2 DPU Platforms


| Model Number | Marketing Description |
|-----------------|--|
| MBF2H322A-AEEOT | NVIDIA® BlueField®-2 P-Series DPU 25GbE Dual-Port SFP56, PCIe Gen4 x8, Crypto Enabled, 8GB on-board DDR, 1GbE OOB management, Tall Bracket, HHHL |
| MBF2H322A-AENOT | BlueField-2 P-Series DPU 25GbE Dual-Port SFP56, PCIe Gen4 x8, Crypto Disabled, 8GB on-board DDR, 1GbE OOB management, Tall Bracket, HHHL |
| MBF2H332A-AEEOT | BlueField-2 P-Series DPU 25GbE Dual-Port SFP56, PCIe Gen3/4 x8, Crypto Enabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, HHHL |
| MBF2H332A-AENOT | BlueField-2 P-Series DPU 25GbE Dual-Port SFP56, PCIe Gen3/4 x8, Crypto Disabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, HHHL |
| MBF2H516A-CEEOT | BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56, PCIe Gen4 x16, Crypto Enabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, FHHL |

| Model Number | Marketing Description |
|-----------------|--|
| MBF2H516A-CENOT | BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56, PCIe Gen4 x16, Crypto Disabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, FHHL |
| MBF2H516A-EEEOT | BlueField-2 P-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56, PCIe Gen4 x16, Crypto Enabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, FHHL |
| MBF2H516A-EENOT | BlueField-2 P-Series DPU 100GbE/EDR VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL |
| MBF2H516B-CENOT | BlueField-2 P-Series BF2500 DPU Controller, 100GbE Dual-Port QSFP56, PCIe Gen4 x16, Crypto Disabled, 16GB on-board DDR, 1GbE OOB Management, Tall Bracket, FHHL |
| MBF2H516B-EENOT | BlueField-2 P-Series BF2500 DPU Controller, 100GbE/EDR/HDR100 VPI Dual-Port QSFP56, PCIe Gen4 x16, Crypto Disabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, FHHL |
| MBF2M322A-AEEOT | BlueField-2 E-Series DPU 25GbE Dual-Port SFP56, PCIe Gen3/4 x8, Crypto, 8GB on-board DDR, 1GbE OOB management, Tall Bracket, HHHL |
| MBF2M322A-AENOT | BlueField-2 E-Series DPU 25GbE Dual-Port SFP56, PCIe Gen3/4 x8, Crypto Disabled, 8GB on-board DDR, 1GbE OOB management, Tall Bracket, HHHL |
| MBF2M332A-AEEOT | BlueField-2 E-Series DPU 25GbE Dual-Port SFP56, PCIe Gen4 x8, Crypto, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, HHHL |
| MBF2M332A-AENOT | BlueField-2 E-Series DPU 25GbE Dual-Port SFP56, PCIe Gen4 x8, Crypto Disabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, HHHL |
| MBF2M516A-CEEOT | BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL |
| MBF2M516A-CENOT | BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56, PCIe Gen4 x16, Crypto Disabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, FHHL |
| MBF2M516A-EEEOT | BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56, PCIe Gen4 x16, Crypto Enabled, 16GB on-board DDR, 1GbE OOB management, Tall Bracket, FHHL |
| MBF2M516A-EENOT | BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL |

Supported NVIDIA BlueField DPU Platforms

| Model Number | Marketing Description |
|------------------|--|
| MBF1L516A-CSNAT | BlueField DPU 100GbE dual-port QSFP28; PCIe Gen3.0/4.0 x16; BlueField G-Series 16 Cores; Crypto disabled; 8GB on-board DDR; tall bracket; FHHL |
| MBE1100A-BC1 | 1U BlueField Reference Platform with Crypto enable; A network appliance development platform |
| MBE1100A-BN1 | 1U BlueField Reference Platform; A network appliance development platform |
| MBE1200A-BC1 | 2U BlueField Reference Platform with Crypto enable; A storage controller development platform with up to 16 SSDs (SSD are not included) |
| MBE1201A-BC1 | 2U BlueField Reference Platform; BlueField P-Series; Crypto enabled; A storage controller platform with option for up to 16 SSDs (SSDs are not included) |
| MBE1201A-BN1 | 2U BlueField Reference Platform; BlueField P-Series; Crypto disabled; A storage controller platform with option for up to 16 SSDs (SSDs are not included) |
| MBE1200A-BN1 | 2U BlueField Reference Platform; A storage controller development platform with option for up to 16 SSDs (SSDs are not included) |
| MBF1M332A-AENAT | BlueField DPU 25GbE dual-port SFP28; PCIe Gen3.0/4.0 x8; BlueField G-Series 8 Cores; Crypto disabled; 16GB on-board DDR; tall bracket; HHHL; ROHS R6 |
| MBF1M332A-AECAT | Bluefield Network Adapter; 8 cores; dual-port SFP28; PCIe4.0 x8; Crypto; 16GB on-board DDR; ROHS R6 |
| MBF1M332A-ASNAT | BlueField DPU 25GbE dual-port SFP28; PCIe Gen3.0/4.0 x8; BlueField G-Series 16 Cores; Crypto disabled; 16GB on-board DDR; HHHL; ROHS R6 |
| MBF1M332A-ASCAT | BlueField DPU 25GbE dual-port SFP28; PCIe Gen3.0/4.0 x8; BlueField G-Series 16 Cores; Crypto enabled; 16GB on-board DDR; HHHL; ROHS R6 |
| MBF1L516A-ESNAT | BlueField DPU VPI EDR IB (100Gb/s) and 100GbE; Dual Port QSFP28; PCIe Gen4.0 x16; BlueField G-Series 16 cores; Crypto disabled; 16GB on-board DDR; FHHL; Single Slot; Tall Bracket |
| MBF1L516A-ESCAT | BlueField DPU VPI EDR IB (100Gb/s) and 100GbE; Dual Port QSFP28; PCIe Gen4.0 x16; BlueField G-Series 16 cores; Crypto enabled; 16GB on-board DDR; FHHL; Single Slot; Tall Bracket |
| MBF1L516A-CSNAT | BlueField DPU 100GbE; Dual Port QSFP28; PCIe Gen4.0 x16; BlueField G-Series 16 cores; Crypto disabled; 16GB on-board DDR; FHHL; Single Slot; Tall Bracket |
| MBF1L516A-CSCAT | BlueField DPU 100GbE; Dual Port QSFP28; PCIe Gen4.0 x16; BlueField G-Series 16 cores; Crypto enabled; 16GB on-board DDR; FHHL; Single Slot; Tall Bracket |
| MBF1L516B-CSNAT; | BlueField Controller card; Dual Port 100GbE QSFP28; BlueField G-Series 16 cores; PCIe Gen4.0 x16; Crypto disabled; 16GB on-board DDR; FHHL; Single Slot |

| Model Number | Marketing Description |
|---|---|
| MBF1M616A-CSNAT; MBF1M606A-CSNAT; MBF1M646A-CSNAT | BlueField Controller card; Dual Port 100Gb/s QSFP28; PCIe Gen4.0 x16; BlueField E-Series 16 cores; Crypto disabled; No Memory DIMM; FH3/4L; Single Slot; ROHS R6; Tall Bracket |
| MBF1M656A-CSNAT; MBF1M626A-CSNAT; MBF1M636A-CSNAT | BlueField Controller card; Dual Port 100Gb/s QSFP28; BlueField E-Series 16 cores; PCIe Gen3.0 x16; Auxiliary Card PCIe Gen3.0 x16; 2 x I-PEX 350mm Cable; Crypto disabled; No Memory DIMM; FH3/4L; Dual Slot; ROHS R6 |
| MBF1M332A-ASNST | BlueField NVMe SNAP DPU 25GbE dual-port SFP28; PCIe Gen3.0/4.0 x8; BlueField G-Series 16 Cores; Crypto disabled; 16GB on-board DDR; HHHL |
| MBF1M322A-ASNAT | BlueField DPU 25GbE dual-port SFP28; PCIe Gen3.0/4.0 x8; BlueField G-Series 16 Cores; Crypto disabled; 8GB on-board DDR; HHHL |
| MBF1M332A-AECAT | BlueField Network Adapter; 8 cores; dual-port SFP28; PCIe4.0 x8; Crypto; 16GB on-board DDR; ROHS R6 |
| MBF1M332A-ASCAT | BlueField DPU 25GbE dual-port SFP28; PCIe Gen3.0/4.0 x8; BlueField G-Series 16 Cores; Crypto enabled; 16GB on-board DDR; HHHL; ROHS R6 |
| MBF1M332A-AENAT | BlueField DPU 25GbE dual-port SFP28; PCIe Gen3.0/4.0 x8; BlueField G-Series 8 Cores; Crypto disabled; 16GB on-board DDR; tall bracket; HHHL; ROHS R6 |
| MBF1M332A-ASNAT | BlueField DPU 25GbE dual-port SFP28; PCIe Gen3.0/4.0 x8; BlueField G-Series 16 Cores; Crypto disabled; 16GB on-board DDR; HHHL; ROHS R6 |
| MBF1M656A-ESNAT | BlueField Controller card; Dual Port VPI 100Gb/s QSFP28; BlueField E-Series 16 cores; PCIe Gen3.0 x16; Auxiliary Card PCIe Gen3.0 x16; 2 x I-PEX 350mm Cable; Crypto disabled; 2x 8GB Memory DIMM; FH3/4L; Single Slot; |

 FDR speed (56Gb/s) protocol is currently not supported on the following card OPNs:

- MBF1L516A-ESNAT
- MBF1L516A-ESCAT

Embedded Software

 Ubuntu 20.04 is the default OS on the BlueField-2 platforms. If another OS is desired, please contact NVIDIA Support.

- BlueField HCA firmware version 18.29.2002
- BlueField-2 HCA firmware version 24.29.2002

- MLNX_OFED 5.2-2.2.0.0
- Installation BFB 3.5.1.11601
- Yocto/Poky reference root file system 3.5.1.11601
- Yocto/Poky reference initramfs 3.5.1.11601
- BlueField Software Distribution 3.5.1.11601
 - BlueField ATF 3.5.1-2-g33ffaa6
 - BlueField UEFI 3.5.1-1-g4078432
 - Yocto meta-bluefield recipes
 - BlueField Linux drivers
 - CentOS installation driver disk (dd)
 - BlueField utilities
- BlueField cross-build development SDK
- NVMe SNAP: 3.1.0-70
- MFT: 4.16.1-9
- SPDK: 20.10-7
- DPDK: 20.11.0-1
- PXE: 3.6.0204
- RShim: 2.0.5-9.g5d9e1b3

Tested SSDs

| Vendor | Model |
|-----------------|---|
| Realtek | RTS5 763DL |
| | RTS5 762 |
| Marvell | 88NR |
| Western Digital | Ultrastar DC M.2 |
| | Ultrastar DC U.2 |
| Toshiba | XG6 |
| CNEX | CNX-2670 |
| SK Hynix | PI6011 |
| Intel | DC P3500 |
| | Intel Corporation Optane SSD 900 |
| | Intel SSD DC P4500 |
| | SSD DC P3520 Series 450 GB model: SSDPE2MX450G7 |
| | SSD OPTANE 900p 280 GB model: SSDPE21D280GAX1 |
| Micron | Micron 9200 SSD |
| | Micron 9300 SSD |
| Samsung | 172Xa |

Supported DPU Linux Distributions (aarch64)

- Ubuntu 20.04
- Yocto 3.0
- CentOS 7.6 (drivers only)

- CentOS 8.2 (drivers only)
- Debian 10 (drivers only)

Supported DPU Host OS Distributions

- RHEL/CentOS 7.4
- RHEL/CentOS 7.5
- RHEL/CentOS 7.6
- RHEL/CentOS 8.0
- RHEL/CentOS 8.1
- RHEL/CentOS 8.2
- Ubuntu 18.04
- Ubuntu 20.04
- Debian 9.11
- Windows Server 2012R2
- Windows Server 2016
- Windows Server 2019

Supported Open vSwitch

- 2.14.1

Known Issues

| Ref # | Issue |
|---------|---|
| 2411542 | Description: Multi-APP QoS is not supported when LAG is configured. |
| | Workaround: N/A |
| | Keywords: Multi-APP QoS, LAG |
| | Discovered in version: 3.5.1.11601 |
| 2406401 | Description: Address Translation Services is not supported in BlueField-2 step A1 devices. Enabling ATS can cause server hang. |
| | Workaround: Configure "ATS_ENABLED=0" using mlxconfig. |
| | Keywords: ATS |
| | Discovered in version: 3.5.1.11601 |
| 2456947 | Description: All NVMe emulation counters (Ctrl, SQ, Namespace) return "0" when queried. |
| | Workaround: N/A |
| | Keywords: Emulated devices; NVMe |
| | Discovered in version: 3.5.1.11601 |
| 2458040 | <p>Description: When using OpenSSL on BlueField platforms where Crypto support is disabled, the following errors may be encountered:</p> <p>PKA_ENGINE: PKA instance is invalid</p> <p>PKA_ENGINE: failed to retrieve valid instance</p> <p>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.</p> |

| Ref # | Issue |
|-------------|---|
| | <p>Workaround: Unlink PKA hardware from OpenSSL configuration. This can be done by running the following commands:</p> <pre># cp /etc/ssl/openssl.cnf.orig /etc/ssl/openssl.cnf # cp /etc/ssl/openssl.cnf.orig /etc/ssl/openssl.cnf.mlnx</pre> <p>Keywords: PKA; Crypto</p> <p>Discovered in version: 3.5.1.11601</p> |
| 24549 25 | <p>Description: When working with a very large number of VFs, resource allocation must be considered. If the required resources are too large, some controllers may not get IO queues, and, therefore, will not be able to expose disks (namespaces) to the host.</p> <p>Workaround: When creating NVMe controllers on VFs using "snap_rpc.py controller_nvme_create", use the flag "--nr_io_queues=1".</p> <p>Keywords: Virtual functions; virtual machine; SNAP; VM; SR-IOV; NVMe</p> <p>Discovered in version: 3.5.1.11601</p> |
| 24400 60 | <p>Description: When a VirtIO-net controller is already running, starting another controller on a different emulation manager will return an error.</p> <p>Workaround: N/A</p> <p>Keywords: VirtIO-net; emulated devices</p> <p>Discovered in version: 3.5.1.11601</p> |
| 24452 89 | <p>Description: If secure boot is enabled, MFT cannot be installed on the BlueField DPU independently from BlueField drivers (MLNX_OFED).</p> <p>Workaround: N/A</p> <p>Keywords: MFT; secure boot</p> <p>Discovered in version: 3.5.1.11601</p> |
| 25814 08 | <p>Description: On a BlueField device operating in Embedded CPU mode, PXE driver will fail to boot as long as the Arm side is not fully loaded and the OVS bridge is not configured.</p> <p>Workaround: Run warm reboot on the host side and boot again via the device when Arm is up and the OVS bridge is configured.</p> <p>Keywords: Embedded CPU; PXE; UEFI; Arm</p> <p>Discovered in version: 3.5.0.11601</p> |
| 24025 31 | <p>Description: PHYless reset on BlueField-2 devices may cause the device to disappear.</p> <p>Workaround: Perform host power reset.</p> <p>Keywords: PHY; firmware reset</p> <p>Discovered in version: 3.5.0.11563</p> |
| 17552 86 | <p>Description: Port speed may change to SDR spontaneously.</p> <p>Workaround: Keep the keep_ib_link_up bit at 0 in NVconfig to make sure the port is raised with the correct speed.</p> <p>Keywords: SDR; port speed</p> <p>Discovered in version: 3.5.0.11563</p> |


| Ref # | Issue |
|---------|---|
| 2264269 | Description: If the configured BAR2 size results in a number of SFs that the device cannot support, then that number is reduced to an allowed value. |
| | Workaround: N/A |
| | Keywords: SFs; sub-functions; BAR2 |
| | Discovered in version: 3.5.0.11563 |
| 2371060 | Description: When emulated PCIe switch is enabled, and the OS performs resource reallocation, the OS boot process might halt. |
| | Workaround: N/A |
| | Keywords: Emulated PCIe switch |
| | Discovered in version: 3.5.0.11563 |
| 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. |
| | Workaround: Restart the drivers on the Arm cores, and then the host. Run: |
| | <pre>/etc/init.d/openibd restart</pre> |
| | Keywords: Ubuntu; reboot; representors; switchdev |
| 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. |
| | Workaround: Add efivars_pstore_disable=1 into the kernel command line in grub config, which will prevent the generation of such files. If the problem already happened and you are not able to recover from Linux, try to enter the UEFI menu > Device Manager > System Configuration > Reset UEFI variables. Afterwards, the CentOS/Ubuntu boot entry will be lost. This can be manually added back in the UEFI shell by running: |
| | <pre>bcfg boot add 0 FS0:\EFI\centos\grubaa64.efi CentOS</pre> |
| | If this does not work, please contact NVIDIA Support. |
| 2400381 | Description: When working with strongSwan 5.9.0bf, running <code>ip xfrm state show</code> returns partial information as to the offload parameters, not showing "mode full". |
| | Workaround: Please use <code>/opt/mellanox/iproute2/sbin/ip xfrm state show</code> instead. |
| | Keywords: strongSwan; ip xfrm; IPsec |
| | Discovered in version: 3.5.0.11563 |
| 2394130 | Description: When creating a large number of VirtIO VFs, hung task call traces may be seen in the dmesg. |
| | Workaround: Disregard the call trace. Once all VirtIO VFs are created, the request behind the hung task will proceed. |

| Ref # | Issue |
|-------------|---|
| | Keywords: VirtIO; call traces; hang |
| | Discovered in version: 3.5.0.11563 |
| 23979 32 | 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 <code>ip x s f && ip x p f</code> . |
| | Workaround: N/A |
| | Keywords: IPsec; SR-IOV; driver |
| | Discovered in version: 3.5.0.11563 |
| 23950 68 | Description: When issuing firmware reset, the Arm filesystems are not cleanly unmounted. This requires filesystem checks that delay the boot process. This may cause the host system to time out after performing <code>mlxfwreset</code> . |
| | Workaround: Reload the mlx5 drivers in the external host system after <code>mlxfwreset</code> is completed. |
| | Keywords: Firmware reset; Arm |
| | Discovered in version: 3.5.0.11563 |
| 23926 04 | Description: Server crashes after configuring <code>PCI_SWITCH_EMULATION_NUM_PORT</code> to a value higher than the number of PCIe lanes the server supports. |
| | Workaround: N/A |
| | Keywords: Server; hang |
| | Discovered in version: 3.5.0.11563 |
| 23770 21 | Description: Executing "sudo poweroff" on the Arm side causes the system to hang. |
| | Workaround: Reboot your BlueField device or power cycle the server. |
| | Keywords: Hang; reboot |
| | Discovered in version: 3.5.0.11563 |
| 23501 32 | Description: Boot process hangs at BIOS (version 1.2.11) stage when power cycling a server (model Dell PowerEdge R7525) after configuring " <code>PCI_SWITCH_EMULATION_NUM_PORT</code> " > 27. |
| | Workaround: N/A |
| | Keywords: Server; hang; power cycle |
| | Discovered in version: 3.5.0.11563 |
| 22937 91 | Description: Loading/reloading NVMe after enabling VirtIO fails with a PCI bar memory mapping error. |
| | Workaround: Load NVMe before enabling VirtIO. |
| | Keywords: VirtIO; NVMe |
| | Discovered in version: 3.1.0.11424 |
| 22695 32 | Description: Currently, configuring 4 external hosts with 64 VFs per host is not supported. |
| | Workaround: N/A |
| | Keywords: Firmware; MSI-X; host |
| | Discovered in version: 3.1.0.11424 |
| 22788 33 | Description: Creating a bond via NetworkManager and restarting the driver (<code>openibd restart</code>) results in no <code>pf0hpf</code> and bond creation failure. |

| Ref # | Issue |
|---------|--|
| | <p>Workaround: Before restarting driver, destroy bond configuration from NetworkManager. Or, instead of using NetworkManager to create bond, use "ip" commands.</p> <p>Keywords: Bond; LAG; network manager; driver reload</p> <p>Discovered in version: 3.1.0.11424</p> |
| 2286596 | <p>Description: Only up to 62 host virtual functions are currently supported.</p> <p>Workaround: N/A</p> <p>Keywords: SmartNIC, SR-IOV</p> <p>Discovered in version: 3.1.0.11424</p> |
| 2245983 | <p>Description: When working with OVS in the kernel and using Connection Tracking, up to 500,000 flows may be offloaded.</p> <p>Workaround: N/A</p> <p>Keywords: SmartNIC; Connection Tracking</p> <p>Discovered in version: 3.0 beta</p> |
| 2581408 | <p>Description: On a BlueField device operating in Embedded CPU mode, PXE driver will fail to boot as long as the Arm side is not fully loaded and the OVS bridge is not configured.</p> <p>Workaround: Run warm reboot on the host side and boot again via the device when Arm is up and the OVS bridge is configured.</p> <p>Keywords: Embedded CPU; PXE; UEFI; Arm</p> <p>Discovered in version: 2.5.0.11176</p> |
| 2096091 | <p>Description: Restarting NetworkManager disables OVS offloads. As a result, all traffic would pass through OVS instead of being offloaded to the embedded switch.</p> <p>Workaround: After restarting NetworkManager, OVS should be restarted as well.</p> <p>Keywords: Open vSwitch offload; network manager</p> <p>Discovered in version: 2.5.0.11176</p> |
| 2082985 | <p>Description: During boot, the system enters systemctl emergency mode due a corrupt root file system.</p> <p>Workaround:</p> <ol style="list-style-type: none"> 1. Do not power off the system without syncing the disks first. 2. In order to fsck the the root file system, it cannot be mounted. Put the install BFB file via the rshim interface and run "fsck /dev/mmcblk0p2" to recover. <p>Keywords: Boot</p> <p>Discovered in version: 2.4.0.11082</p> |
| 1945513 | <p>Description: If the Linux OS running on the host connected to the BlueField SmartNIC has a kernel version lower then 4.14, MLNX_OFED package should be installed on the host.</p> <p>Workaround: N/A</p> <p>Keywords: Host OS</p> <p>Discovered in version: 2.4.0.11082</p> |
| 1859322 | <p>Description: On some setups, SmartNIC does not power on following server cold boot when UART cable is attached to the same server.</p> |

| Ref # | Issue |
|-------------|--|
| | 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: SmartNIC; Arm; Cold Boot |
| | Discovered in version: 2.4.0.11082 |
| 18999 21 | Description: Driver restart fails when SNAP service are 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 |
| | Discovered in version: 2.2.0.11000 |
| 19002 03 | 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 |
| | Discovered in version: 2.2.0.11000 |
| 19116 18 | 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 |
| | Discovered in version: 2.2.0.11000 |

Bug Fixes In This Version

 For an archive of bug fixes from previous releases, please see "[Bug Fixes History](#)".

| Ref # | Issue Description |
|---------|---|
| 2398050 | Description: Only up to 60 virtio-net emulated virtual functions are supported if LAG is enabled. |
| | Keywords: Virtio-net; LAG |
| | Discovered in version: 3.5.0.11563 |
| 2256134 | Description: On rare occasions, rebooting the BlueField DPU may result in traffic failure from the x86 host. |
| | Keywords: Host; Arm |
| | Discovered in version: 3.5.0.11563 |
| 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 |
| | Discovered in version: 3.5.0.11563 |

BlueField Software Overview

 It is recommended to upgrade your BlueField product to the latest software and firmware versions available in order to enjoy the latest features and bug fixes.

NVIDIA Mellanox 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 are able to:

- 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

The BlueField family of DPU devices combines an array of 64-bit Armv8 A72 cores coupled with the ConnectX® interconnect. 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 quickly and efficiently design, implement and verify their control-plane and data-plane applications.

BlueField SW ships with the Mellanox BlueField Reference Platform. Bluefield SW is a reference Linux distribution based on Ubuntu Server or Yocto Poky distribution and extended to include the Mellanox OFED stack for Arm and a Linux kernel which supports NVMe-oF. This SW distribution is capable of running all customer-based Linux applications seamlessly. Yocto also provides an SDK which contains an extremely flexible cross-build environment allowing software targeted for the BlueField DPU to build on virtually any host server running any Linux distribution.

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
- Hardware Diagnostics
- Mellanox OFED stack
- Mellanox MFT

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 in case of SmartNIC) as well which could be used for debugging with open source tools like OpenOCD.

The BlueField DPU also supports the ubiquitous GDB.

BlueField Adapter/SmartNIC

The BlueField SmartNIC is shipped with the BlueField Software Distribution (BSD) pre-installed. The BlueField adapter Arm execution environment has the capability of being fully isolated (Air Gap) 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.

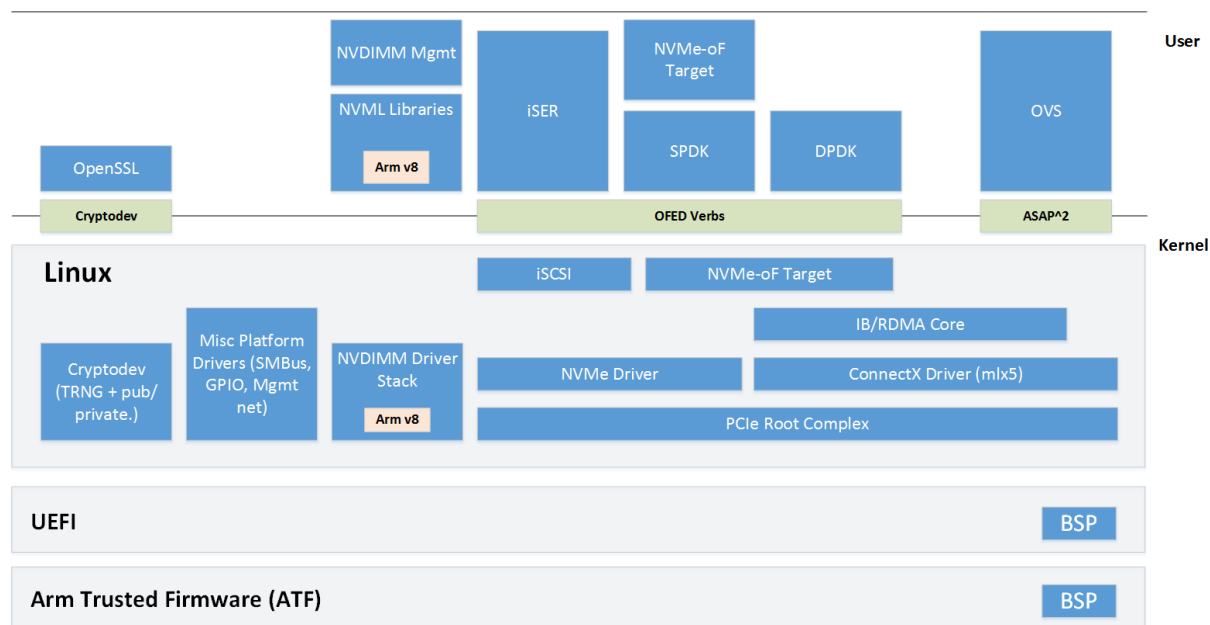
BlueField-based Storage Appliance

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 NVIDIA® Mellanox® ConnectX® offload such as RDMA/RoCE, T10 DIF signature offload, erasure coding offload, iSER, Storage Spaces Direct, and more.

BlueField Architecture


The BlueField architecture is a combination of two preexisting standard off-the-shelf components, Arm AArch64 processors, and ConnectX®-5 (for BlueField), ConnectX-6 Dx (for BlueField-2), 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 Mellanox 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 Mellanox OFED software stack and InfiniBand verbs-based interfaces to support software.

Installation and Initialization

 Please consult the README files in the BlueField Software Distribution (BSD) for the most updated content.

The BSD consists of the following images:

- BlueField-<version>_install.bfb - installation BFB file
- BlueField-<bluefield_version>.tar.xz - base BSD tarball which contains all the BlueField specific source code as well as sample binary images
- core-image-full-BlueField-<bluefield_version>.<yocto_version>.tar.xz - base reference BlueField full root filesystem tar archive
- core-image-initramfs-BlueField-<bluefield_version>.<yocto_version>.cpio.xz - base reference BlueField initramfs cpio image
- poky-glibc-x86_64-core-image-full-sdk-aarch64-toolchain-BlueField-<bluefield_version>.<yocto_version>.sh - Yocto-produced SDK in a self-installing script. This contains all cross-build tools and utilities to allow building an image targeted for the BlueField platform on an x86 server running Linux.

Unpacking BlueField Software Distribution With Yocto

To unpack the Yocto BSD, run:

```
$ tar xvf BlueField-<bluefield_version>.tar.xz
```

This unpacks to a BlueField-<bluefield_version>/ subdirectory (referred to in this document as <BF_INST_DIR>) containing the following top-level hierarchy:

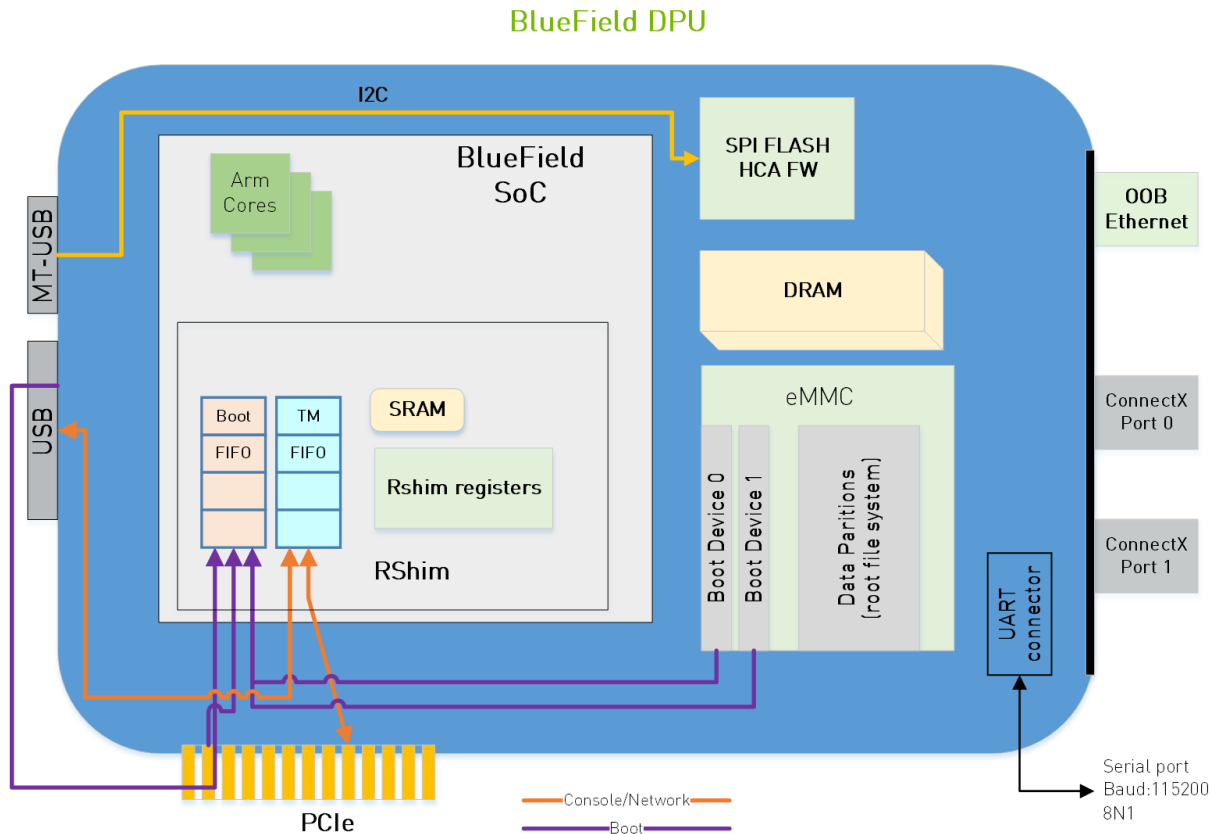
- bin - contains tools to manage the process of installing runtime software. For example, the mlx-mkbfbs tool to generate the BlueField boot stream files used to provide initial boot images.
- boot - contains the boot loader binaries built from the provided sources for each of the BlueField devices. The "bl*" files in a device's dedicated folder are taken from the Arm Trusted Firmware (ATF) and each represents a different boot phase; note that the file bl1.bin corresponds to the boot ROM burned into the DPU itself. The *.fd file is the Unified Extensible Firmware Interface (UEFI) boot image. The *.bfb file is a generated BlueField boot stream file which includes all the above boot loader components.
- distro - contains information pertinent to different Linux distributions. For example, the distro/yocto directory contains the "meta-bluefield" layer used to build a BlueField-targeted version of the standard Yocto/Poky meta-distribution.
- sample - contains sample images which can be used to boot up a BlueField chip to a Linux bash prompt, to either validate that hardware is working correctly, or for experimentation. See the README and README.install files in that directory for more information.
- src - contains patches for various components (e.g. ATF, UEFI, and Linux), as well as complete sources for Linux drivers and user-space components authored by Mellanox® and not yet upstreamed
- Documentation - contains HOWTO READMEs which may also be consulted for a number of useful procedures

The BSD contains numerous README files in the aforementioned directories which provide more information. The following README files must be consulted particularly when upgrading your BSD release as they contain important release-specific information:

- <BF_INST_DIR>/sample/README.install
- <BF_INST_DIR>/sample/README
- <BF_INST_DIR>/distro/yocto/README-bluefield
- <BF_INST_DIR>/src/atf/README
- <BF_INST_DIR>/src/atf/README-bfb
- <BF_INST_DIR>/distro/rhel/pxeboot/README

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.



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 SmartNIC 25G
 - Requires serial cable to 3-pin connector on SmartNIC 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

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 smartNIC 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 only)
BlueField-2 based platforms feature an OOB 1 gigabit Ethernet 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).

RShim Host Driver

Check RShim Device

Before installing the RShim driver, check the following RShim devices which will be probed by the driver.

For BlueField DPU:

- RShim device over USB

```
# lsusb
Bus xxx Device xxx: ID 22dc:0004
```

- RShim device over PCIe (with NIC FW)

```
# lspci -n | grep 15b3
xx:xx.2 xxxx: 15b3:c2d2
```

- RShim device over PCIe (without NIC FW)

```
# lspci -n | grep 15b3
xx:00.0 xxxx: 15b3:0211
```

For BlueField-2 DPU:

- RShim device over USB

```
# lsusb
Bus xxx Device xxx: ID 22dc:0214
```

- RShim device over PCIe (with NIC FW)

```
# lspci -n | grep 15b3
xx:xx.2 xxxx: 15b3:c2d3
```

- RShim device over PCIe (without NIC FW)

```
# lspci -n | grep 15b3
xx:00.0 xxxx: 15b3:0214
```

⚠ For RShim over PCIe in multi-host configuration, host[0] which owns PCIe lane 0 will have RShim access by default. Other hosts will not see the RShim function.

Building and Installing RShim Host Driver

⚠ For more RShim options and explanation, please refer to <BF_INST_DIR>/Documentation/HOWTO-rshim.

RShim driver can be installed in several ways on the server host as elaborated on for different Linux OS in the following subsections.

For CentOS, Fedora, and RedHat

Users need root privileges in order build and install the RShim driver. To get the packages required to perform this, run:

```
yum install pciutils-devel libusb-devel fuse-devel
```

For CentOS/RedHat 8.0 and Fedora, also run:

```
yum install kernel-modules-extra-$(uname -r)
```

Assuming that the tarball is uncompressed under the <BF_INST_DIR> directory, the RShim source RPM may be taken from <BF_INST_DIR>/distro/SRPMS/rshim-xxx.src.rpm.

Copy the RShim source RPM to the host you want to install it on and execute the following (your RPM file name will be different):

```
$ rpmbuild --rebuild rshim-<version-number>.src.rpm
```

Assuming you ran the rpmbuild command from the root home directory, cd to "/root/rpmbuild/RPMS/x86_64" and execute the following (your RPM file name will be different):

```
$ rpm -Uvh rshim-<version-number>.x86_64.rpm -force
```

To continue installation, follow the instructions under [Finalizing Setup](#).

For Ubuntu

Users need root privileges in order build and install the RShim driver. To get the packages required to perform this, run:

```
apt install autoconf pkg-config build-essential devscripts fakeroot libpci-dev libusb-1.0-0-dev libfuse-dev
```

Then copy the source code into your directory and compile it:

```
cp -ar <BF_INST_DIR> /src/drivers/rshim/ .
rm rshim*.deb; cd rshim; debuild -us -uc
apt install ../rshim*.deb
```

To continue installation, follow the instructions under [Finalizing Setup](#).

For SuSE

Users need root privileges in order build and install the RShim driver. To get the packages required to perform this, run:

```
zypper install autoconf automake
zypper install -t pattern devel_basis
zypper install pciutils-devel libusb-1_0-devel
```

Then follow the same [RPM build steps](#) for CentOS, Fedora, and RedHat.

To continue installation, follow the instructions under [Finalizing Setup](#).

For FreeBSD and Other

For FreeBSD 12.0+ only, users need root privileges in order build and install the RShim driver. To get the packages required to perform this, run:

```
pkg install autoconf automake gmake libpoll-shim libpciaccess libpci pkgconf
```

Then copy the source code into your directory and compile it:

```
cp -ar <BF_INST_DIR> /src/drivers/rshim/ .
cd <rshim>
./bootstrap.sh
./configure
make && make install
```

To continue installation, follow the instructions under [Finalizing Setup](#).

Finalizing Setup

1. Start the Driver. Unload the kernel modules if the old version is still running.

```
rmmod rshim_net
rmmod rshim_pcie
rmmod rshim_usb
rmmod rshim
```

2. Start the RShim service:

```
systemctl start rshim
```

Device Files

Each RShim backend creates a directory named according to the format `/dev/rshim<N>/` with the following files (<N> is the device ID, which could be 0, 1, etc):

- `/dev/rshim<N>/boot`
Boot device file used to send boot stream to the Arm side. For example:

```
$ cat install.bfb > /dev/rshim<N>/boot
```

- `/dev/rshim<N>/console`

Console device, which can be used by console tools to connect to the Arm side. For example:

```
$ screen /dev/rshim<N>/console
```

- `/dev/rshim<N>/rshim`

Device file used to access RShim register space. When reading/writing to this file, encode the offset as `"((rshim_channel << 16) | register_offset)"`.

- `/dev/rshim<N>/misc`

Key/value pairs used to read/write miscellaneous data. For example:

```
# Dump the content.
cat /dev/rshim0/misc
DISPLAY_LEVEL 0 (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 (ro)
DEV_INFO      BlueField-2 (Rev 0)
# Initiate a SW reset.
echo "SW_RESET 1" > /dev/rshim<N>/misc

BOOT_MODE supports the following values:
0: boot from RShim. This is done automatically by the RShim driver when pushing boot stream.
1: boot from eMMC (default)
2: swap eMMC boot partition. Once set, the boot ROM switches the eMMC boot partition to the other one
during the next warm reset.
```

Multiple Board Support

Multiple boards could connect to the same host machine. Each board has its own device directory (`/dev/rshim<N>`). The following are some guidelines how to set up RShim networking properly in this case:

- Each target should load only one backend (usb, pcie or pcie_lf)
- The host RShim network interface should have different MAC and IP addresses, which can be configured with `ifconfig` as shown below or saved in configuration:

```
$ ifconfig tmfifo_net0 192.168.100.2/24 hw ether 02:02:02:02:02:02
```

- The Arm side TMFIFO interface should have unique MAC and IP addresses as well, which can be configured in the console

For an example for multi-board management, please refer to section "[Multi-board Management Example](#)".

Permanently Changing Arm Side MAC Address

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-04:00.2 (ro)
DEV_INFO BlueField-2 (Rev 0)
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
```

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.

Configuring Host Server Side

 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"
NM_CONTROLLED="no"
```

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` interfaces. Run:

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

7. Inside `91-tmfifo_net.rules`, insert the following content:

```
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="00:1a:ca:ff:ff:02", ATTR{type}=="1", NAME="tmfifo_net0",
RUN+="/bin/sh -c '/usr/sbin/ifup tmfifo_net0 2>/dev/null || /sbin/ip link set dev tmfifo_net0 up'"
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="00:1a:ca:ff:ff:04", ATTR{type}=="1", NAME="tmfifo_net1",
RUN+="/bin/sh -c '/usr/sbin/ifup tmfifo_net1 2>/dev/null || /sbin/ip link set dev tmfifo_net1 up'"
```

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

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


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 MAC Address

 This procedure is relevant for Ubuntu/Debian (sudo needed) and CentOS BFBs.

1. Open two console screen tabs per BlueField DPU. Run:

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

 Screen console RPM must be installed.

```
# yum install screen
```

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

```
# 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:

```
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.

Updating IP Address

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
```

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.

For Ubuntu:

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

```
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:

```
reboot
```

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

i 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.

BFB Installation Utility Script

With RShim driver 2.0.5-5 and later, a utility script is available for BFB installation called "bfb-install".

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

This utility script pushes the BFB image and optional configuration to the BlueField side and checks and prints the BFB installation progress. To see the BFB installation progress, please install the "pv" tool.

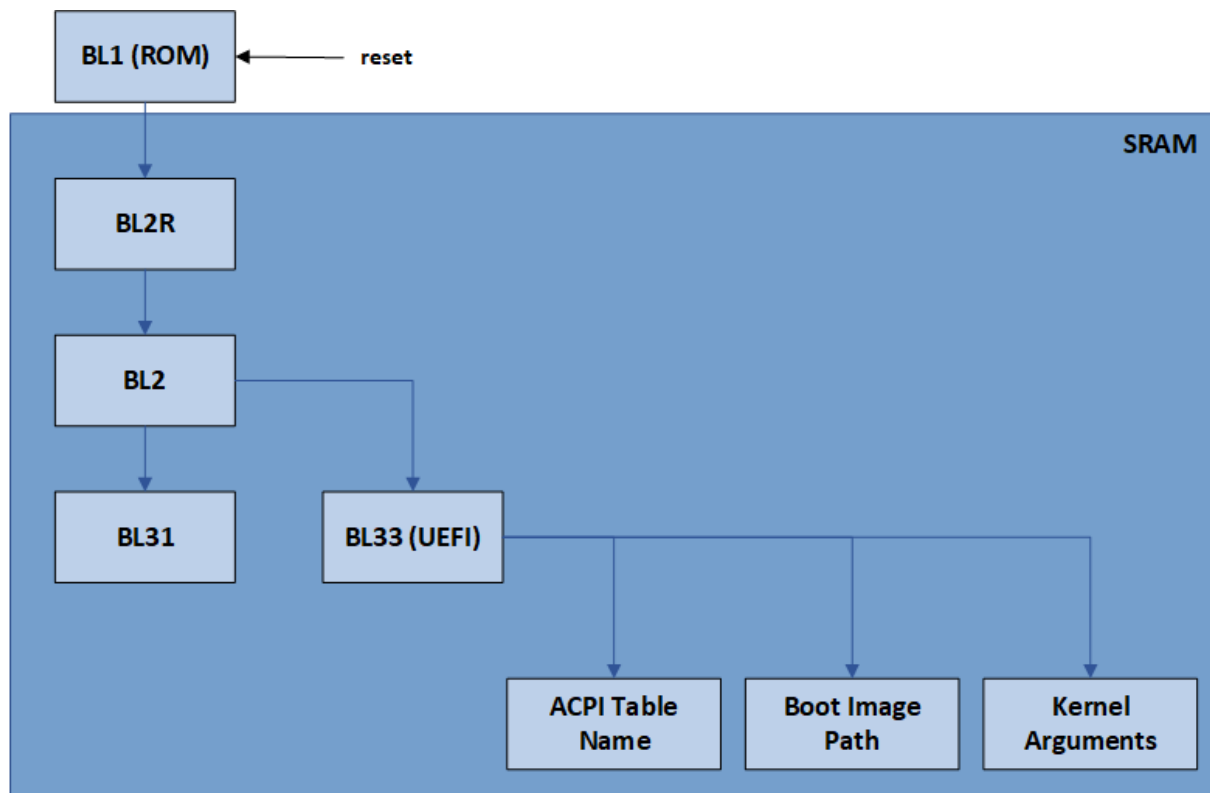
The following is an output example of Yocto installation with the bfb-install script assuming the "pv" tool has been installed.

```
# bfb-install --rshim rshim0 --bfb install.bfb --rootfs core-image-full- bluefield.tar.xz --config bf.cfg
Pushing bfb + cfg + rootfs
896MiB 0:09:12 [1.62MiB/s] [ <=> ]
Collecting BlueField booting status. Press Ctrl+C to stop...
INFO[BL2]: start
INFO[BL2]: no DDR on MSS0
INFO[BL2]: DDR POST passed
INFO[BL2]: UEFI loaded
INFO[BL31]: start
INFO[BL31]: runtime
INFO[UEFI]: eMMC init
INFO[UEFI]: eMMC probed
INFO[MISC]: Found bf.cfg
INFO[MISC]: Found tarball
INFO[MISC]: Start decomp
INFO[MISC]: Start install
INFO[MISC]: Rebooting
```

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.

BFB File Overview




The default BlueField bootstream (BFB) shown above (located at 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 `mlxbf-bootctl` utility to write this BFB to the eMMC boot partition, 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 table below presents the types of files possible in a BFB.

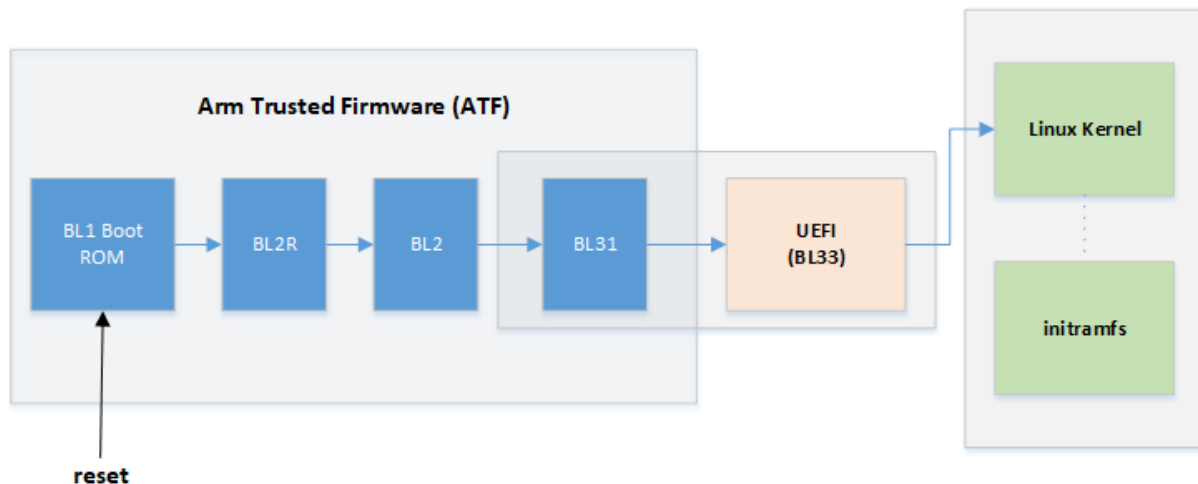
| Filename | Description | ID | Read By |
|------------------|--|----|-----------|
| BL2r-cert | Secure Firmware BL2R (RIoT Core) certificate | 33 | BL1 |
| BL2r | Secure Firmware BL2R (RIoT Core) | 28 | BL1 |
| bl2-cert | Trusted Boot Firmware BL2 certificate | 6 | BL1/BL2R* |
| bl2 | Trusted Boot Firmware BL2 | 1 | BL1/BL2R* |
| 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 |

| Filename | Description | ID | Read By |
|---------------|---|----|---------|
| 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 |

 * 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.

BlueField Boot Process

Before explaining the implementation of the solution, the BlueField boot process needs to be expanded upon.



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.

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 is able to install the new software version on their BlueField system, test it, and then fall back to a previous 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.

In order 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.

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 Yocto 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/mmcbk0. 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.
```

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
```

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 Yocto 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.

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.

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-mkbf" 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.

First, set the PATH variable:

```
$ export PATH=$PATH:<BF_INST_DIR>/bin
```

Then, generate the boot file by using the mlx-mkbf command:

```
$ mlx-mkbf \
  --b12 b12.bin \
  --b131 b131.bin \
  --b133 BLUEFIELD_EFI.fd \
  --boot-acpi "=default" \
  default.bfb
```

This command creates the "default.bfb" from b12.bin, b131.bin, and BLUEFIELD_EFI.fd. The generated file might be used to update the eMMC boot partitions.

To verify the content of the boot file, run:

```
$ mlx-mkbf -d default.bfb
```

To extract the bootloader images from the boot file, run:

```
$ mlx-mkbf -x default.bfb
```

To obtain further details about the tool options, run the tool with -h or --help.

Changing Linux Kernel or Root File System

The solutions above simply update the boot partition to hold new boot loader software (ATF and UEFI). If the user wants to also provide a new kernel image and/or modify the root file system, one way to do that is to place the new kernel image in the image partition of the eMMC, and then modify the root file system directly. Then, in order to boot the new kernel image, simply change the kernel image reference to the new image.

The kernel image reference can be put in one of two place: The UEFI boot manager, or the grub boot loader.

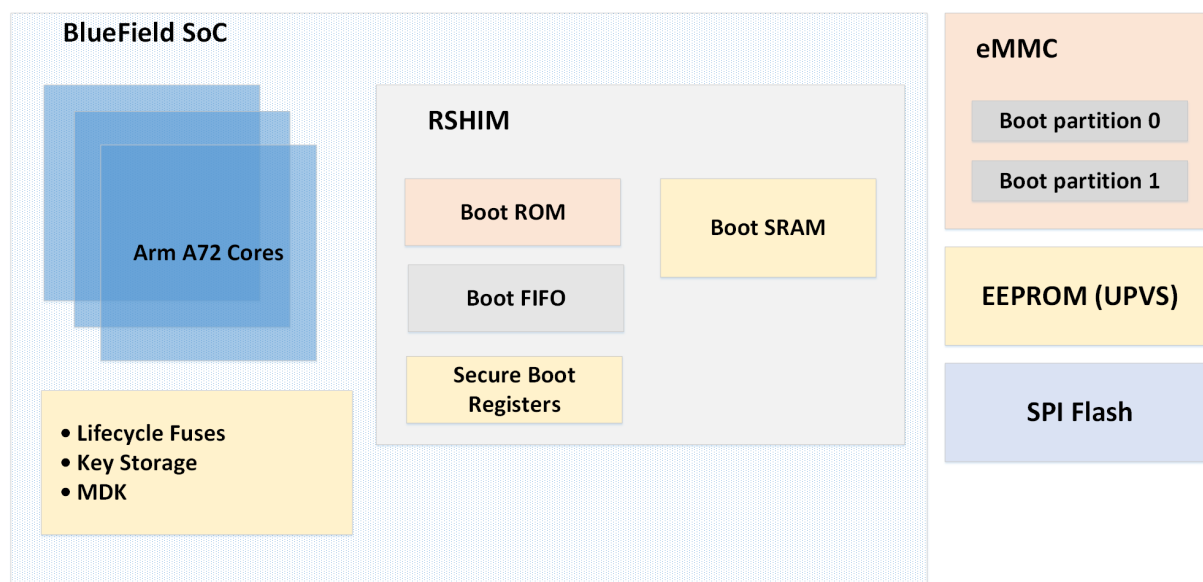
The details on how to perform this depend on the specifics of how and what needs to be upgraded for the specific application. But, in principle, any component of the system can be safely upgraded using this type of approach.

For more information, please refer to [EDK2 user documentation on Github](#).

Building Arm Trusted Firmware

⚠ While descriptions of Arm Trusted Firmware (ATF) are provided related to the BlueField platform; for general knowledge of what ATF is and how it works, please refer to ATF documents from Arm. The "Arm Trusted Firmware User Guide" located at "docs/user-guide.rst" in the ATF sources is a good place to start.


ATF is used in Armv8 systems for booting the chip and then providing secure interfaces. It implements various Arm interface standards like PSCI (Power State Coordination Interface), SMC (Secure Monitor Call) and TBBR (Trusted Board Boot Requirements). ATF is used as the primary bootloader to load UEFI (Unified Extensible Firmware Interface) on the BlueField platform.



ATF has various bootloader stages when loading:

- BL1 - BL1 is stored in the on-chip boot ROM; it is executed once the primary core exits reset. Its main functionality is to do several architectural and platform initialization to the point where it can load the BL2 image. It then loads BL2 and switches execution over to it.


- BL2R (BlueField-2 only) - BL2R is loaded and then executed on the on-chip boot RAM. Its main functionality is to implement crypto operation and calculates measurements for security attestation. It is loaded by BL1 on BlueField-2 platform. BL2R then loads BL2 image and traps itself back to BL1 via an SMC, which then switches execution over to BL2.
- BL2 - BL2 is loaded and then executed on the on-chip boot RAM. Its main functionality is to perform the rest of the low-level architectural and platform initialization (e.g. initializing DRAM, setting up the System Address Mapping and calculating the Physical Memory Regions). It then loads the rest of the boot images (BL31, BL33). Afterwards, it traps itself back to BL1 via an SMC, which in turn switches execution to BL31.
- BL31 - BL31 is known as the EL3 Runtime Software. It is loaded to the boot RAM. Its main functionality is to provide low-level runtime service support. After it finishes all its runtime software initialization, it passes control to BL33.
- BL33 - BL33 is known as the Non-trusted Firmware. For this case we are using EDK2 (Tianocore) UEFI. It is in charge of loading and passing control to the OS. For more detail on this, please refer to the EDK2 source.

 Some users may wish to use the GRUB2 bootloader for various reasons. In that case, UEFI would be configured to load GRUB2 instead of the Linux kernel.

Building ATF Images

To get the source code, directly execute the `atf-d48f19.patch` file found in the directory `<BF_INST_DIR>/src/atf/` on the x86. It downloads the ATF sources from GitHub and patches it with BlueField platform specific code.


Since BL1 is permanently burned into the BlueField on-chip boot ROM, the only real bootloader images which might need to be built are BL2 and BL31 (please refer to the EDK2 documentation of how to build EDK2 to use as BL33). Therefore there is a need to build the "bl2" and "bl31" targets.

 The BL2R target can be built for BlueField-2 platforms only.

Before doing any build, the environment variable `CROSS_COMPILE` should point to the Arm cross-compiler being used. For example:

```
$ export CROSS_COMPILE=/path/to/cross/compiler/aarch64-poky-linux-
```

To build for the BlueField platform, you need `PLAT` set to the platform target when invoking "make" (i.e. "PLAT=bluefield2" for BlueField-2, and "PLAT=bluefield1" for BlueField). You must also set the `MBEDTLS_DIR` to the `mbedtls` directory and enable the trusted board boot (i.e. `TRUSTED_BOARD_BOOT=1`). To obtain the `mbedtls` repos execute the `mbedtls-2.10.0.patch` file.

 If ATF is being built in an environment where the Yocto/Poky SDK script has been run (`environment-setup-aarch64-poky-linux`), you must unset the `LDFLAGS` to `NULL`. That is, execute:

```
export LDFLAGS=
```

So to perform a basic ATF build, run:

```
make \
  BUILD_BASE=/path/to/build \
  MBEDTLS_DIR=/path/to/mbedtls \
  PLAT=bluefield1 \
  bl2 bl31 \
  TRUSTED_BOARD_BOOT=1
```

For BlueField based devices, after the build finishes you can find the needed bl2.bin and bl31.bin under \$BUILD_BASE/bluefield1/release/.

```
make \
  BUILD_BASE=/path/to/build \
  MBEDTLS_DIR=/path/to/mbedtls \
  PLAT=bluefield2 \
  bl2 bl31 bl2r \
  TRUSTED_BOARD_BOOT=1
```

For BlueField-2 based devices, after the build finishes you can find the needed bl2r.bin, bl2.bin and bl31.bin in \$BUILD_BASE/bluefield2/release/.

Trusted Board Boot


The other two files in the directory (mbedtls-2.10.0.patch and gen_fuse_info.py) are related to building ATF with trusted board boot enabled.

For more information of how to perform trusted board boot, please refer to the Secure Boot document.

Building UEFI (EDK2)

After running the "edk2-*.patch" command in the directory <BF_INST_DIR>/src/edk2/ to set up a source tree for UEFI, cd into it and run "make -f <BF_INST_DIR>/src/edk2/Makefile".

Customizations you may need or want to make are expanded on further below.

 EDK2 requires building in the source tree. Also, the EDK2 build system fails with parallel build, so you must build with -j1.

The image built is BLUEFIELD_EFI.fd and/or BLUEFIELD_EFI_SEC.fd in the Build/BlueField/RELEASE_GCC49/FV directory.

Customizable Build Options

The UEFI build options presented in this section are the customizable.

The mode in which to build EDK2: DEBUG or RELEASE:

```
EDK2_MODE = RELEASE
```

Any particular "defines" to use when building EDK2:

```
EDK2_DEFINES = \
  -DSECURE_BOOT_ENABLE=TRUE \
  -DFIRMWARE_VER=0.99 \
```

Path to OpenSSL tarball: Set it to an already-downloaded location for the tarball, or else this makefile will download it into the source tree:

```
OPENSSL_TARBALL = CryptoPkg/Library/OpensslLib/openssl-1.0.2d.tar.gz
```

The compiler toolchain prefix to use for the tools: This can include the full path and prefix if the tools are not in \$(PATH):

```
GCC49_AARCH64_PREFIX = aarch64-poky-linux-
```



GCC 4.9 or later will work. Here Yocto Poky gcc 6.3 or later is used.

If "iasl" is not in your path, specify its directory here. We use <https://github.com/acpica/acpica.git> at commit ed0389cb or later.

```
IASL_PREFIX =
```

If "dtc" is not in your path, specify its directory here. You can typically find it in a Linux build tree in scripts/dtc. If you are using a Yocto SDK you can use the DTC contained within it.

```
DTC_PREFIX=/opt/poky/2.3.1/sysroots/x86_64-pokysdk-linux/usr/bin/
```

Make sure the ARCH environment variable is NULL (unset).

```
DTC_PREFIX =
```

Device tree source files:

```
DTS_FILES = bf-full.dts
DTS_DIR = ../dts
```

It is important to note that when you actually build EDK2 here make sure you are NOT in an environment/bash shell where environment-setup-aarch64-poky-linux was sourced. Simply point GCC49_AARCH64_PREFIX to:

```
$ /opt/poky/2.3.1/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-
```

Running the SDK environment-setup-aarch64-poky-linux script confuses the UEFI make environment. It just needs a pointer to the tools.

Exporting Variables

```
# Export variables that need to be set in the environment.
export IASL_PREFIX GCC49_AARCH64_PREFIX

FD_FILE = Build/BlueField/${EDK2_MODE}_GCC49/FV/BLUEFIELD_EFI.fd

all: $(FD_FILE)

DTB_FILES = $(addprefix dtb/, $(DTS_FILES:.dts=.dtb))

$(FD_FILE): FORCE CryptoPkg/Library/OpensslLib/openssl-1.0.2d $(DTB_FILES)
$(MAKE) -C BaseTools/Source/C
set --; . ./edksetup.sh; \
  build -n 6 -t GCC49 -a AARCH64 -p MlxPlatformPkg/BlueField.dsc \
    -b $(EDK2_MODE) -DDTB_DIR=$(PWD)/dtb $(EDK2_DEFINES)

FORCE:

# Build device tree blobs for specified device tree source(s)

$(DTB_FILES): dtb/%.dtb: $(DTS_DIR)/%.dts
  mkdir -p $(@D)
  cpp -P -x assembler-with-cpp -o- $< | \
    $(DTC_PREFIX)dtc -b 0 -O dtb -o $@.tmp -I dts -
  mv -f $@.tmp $@

# These steps are documented in CryptoPkg/Library/OpensslLib/Patch-HOWTO.txt
CryptoPkg/Library/OpensslLib/openssl-1.0.2d: $(OPENSSL_TARBALL)
  tar -C $(@D) -xf $<
  cd $@ && patch -p0 < ../EDKII_openssl-1.0.2d.patch
  cd $(@D) && ./Install.sh

$(OPENSSL_TARBALL):
  curl https://www.openssl.org/source/openssl-1.0.2d.tar.gz > $@.tmp
  mv -f $@.tmp $@
```

Building Poky Initramfs

Basic Quick Start to Build Poky Initramfs

1. Run the script `scripts-bluefield/yocto_dependencies` from the `/distro/yocto` directory.
2. From the same directory, run the following:

```
. ./meta-bluefield/bluefield-init-build-env
```

Note: Please do not exclude the leading "." in this command.

You should now be in the build directory. Your conf files are in the conf directory.

3. Customize your `bblayers.conf` file. To reproduce the shipped images you may use the `bblayers.conf` that is installed by the `bluefield-init-build-env` script.
4. Customize your `local.conf` file. To reproduce the shipped images you can use the `local.conf` that is installed by the `bluefield-init-build-env` script.
5. Then run:

```
bitbake core-image-initramfs
```

Note that the file system created is located in `"/tmp/deploy/"`.

Common BlueField bitbake targets are:

- bitbake core-image-initramfs
- bitbake core-image-initramfs-netboot
- bitbake core-image-full
- bitbake core-image-full-dev
- bitbake core-image-full-sdk -c populate_sdk
- bitbake core-image-full-sdk -c populate_sdk_ext

You can build Yocto/Poky on most major Linux distributions. Mellanox currently runs tests using CentOS 7.4, however, other distributions such as Ubuntu would also work but may require small modifications to the Yocto config files and/or recipes.

If you are not using CentOS and having difficulties, you may want to try running CentOS in a container or on a VM first in order to get a successful build with which to compare results.

For more information, please refer to the [Yocto Project Mega Manual](#).

Variables

Certain OFED recipes require that the source RPM or tarball already be downloaded to the build systems. These files are included in the BlueField Runtime Distribution. You should place these files anywhere you like and then set the appropriate variable in local.conf. For example, set:

```
MLNX_OFED_PATH=<your_local_path>/distro/mlnx_ofed
```

There are various variables that can be set in local.conf which add files created outside of Yocto to be copied into the root file systems:

- MLNX_OFED_PATH - location of local OFED packages. Look in distro/mlnx_ofed for specific directories.
- MLNX_OFED_VERSION - version of MLNX OFED (e.g. "4.4-2.4.6.0"). This is used with MLNX_OFED_PATH to find files.
- MLNX_OFED_BASE_OS - base OS version of MLNX OFED (e.g. "rhel7.4alternate"). This is used with MLNX_OFED_PATH to find files.
- MLNX_BLUEFIELD_VERSION_PATH - if there is a "bluefield_version" file in this location it gets copied to /etc in the root file systems created by Yocto. See the "update_rootfs_bluefield" function in the meta-bluefield image recipes.
- MLNX_BLUEFIELD_FW_PATH - if this directory exists, the image recipes in meta-bluefield copy the contents of this directory into /lib/firmware/mellanox on the generated root file systems
- MLNX_BLUEFIELD_EXTRA_DEV_PATH - any files in the directory specified by this variable are copied into /opt/mlnx/extra on the full root dev file system.

Downloading Upstream Yocto and Building SDK

The meta-bluefield layer supports the Mellanox BlueField DPU.

To use it, edit your conf/bblayers.conf file to include this directory on the list of directories in BBLAYERS. Similarly, you should also add the layers meta-oe, meta-python, and meta-networking from meta-openembedded, since packages in those layers are used in some of the images included in meta-bluefield/recipes-bsp/images.

You should edit your conf/local.conf file to set MACHINE to "bluefield". To be able to build the same distro configurations used in the Mellanox images (including using the same kernel version shipped by Mellanox), you should also add:

```
$ include conf/bluefield.conf
```

 Mellanox is uses Yocto Rocko 2.4 for this release.

Using Yocto as a Cross-compilation SDK and Root Filesystem Generator

You may download the Yocto/Poky SDK file from the same source from which you acquired the BlueField Runtime Distribution. Typically:

```
$ poky-glibc-x86_64-core-image-full-sdk-aarch64-toolchain-2.4.1.sh
```

Unpacking this file into an SDK directory allows cross-compiling files which are going to run on the BlueField DPU. This directory may be located anywhere you want.

Alternatively, you may download the upstream Yocto and build your own SDK. For more information, see section "[Downloading Upstream Yocto and Building SDK](#)".

To use the SDK cross-compilation tools, you should "source" the top-level "environment-setup-aarch64-poky-linux" script to set various environment variables, including \$PATH, \$CC, \$CROSS_COMPILE, etc. The cross-compilation tools (compiler, assembler, linker, etc.) are located in sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux; many other useful tools are in the directories usr/bin, usr/sbin, bin, and sbin beneath sysroots/x86_64-pokysdk-linux. The sysroots/aarch64-poky-linux hierarchy contains a copy of a root filesystem for Arm64 so the cross-compilation tools can find headers and libraries in it.

To compile your code you should use aarch64-poky-linux-gcc, and, if necessary, the other standard aarch64-poky-linux- tools. In general, you should take advantage of the various environment variables in your makefiles rather than relying on any specific name for the tools. Several of the tools (notably gcc) require a "--sysroot" argument which specifies the aarch64-poky-linux path—the \$ (CC) variable handles this for you.

Note also that for "configure" based software, the top-level environment setup script also sets a \$CONFIG_SITE environment variable pointing to the top-level site-config-aarch64-poky-linux file which includes autoconf definitions for all the known configure variables, to simplify cross-configuration.

Yocto Extensible SDK

The extensible SDK provides a cross-development toolchain and libraries associated with your custom Yocto image. Developers could use the eSDK to develop a cross-build environment supplemented with devtool commands specifically supporting the yocto/bitbake environment.

See [this page](#) for further details.

BlueField-2 OOB Ethernet Interface


The BlueField-2 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 BlueField-2 boot stream. Any attempt to push a BFB to this port will not work. Please refer to [UEFI Boot Option Management](#) for more information about UEFI operations related to the OOB interface.

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 recommended not to reconfigure the MAC address from that 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. The following "printf" command sets 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.

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.

The full list of supported ethtool options are:

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

Where <options> 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
- -K - configure driver offload(s)
- -a - queries the specified Ethernet device for pause parameter information
- -r - restarts 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_net0")
$ 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
```

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 across all three major distributions (Yocto, CentOS, 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 across all three major distributions (Yocto, CentOS, Ubuntu):

- Yocto configuration of IP interface:
 - Configuration file for "ifup" and "ifdown": /etc/network/interfaces
 - For example, use the following to enable DHCP:

```
auto oob_net0
iface oob_net0 inet dhcp
```

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

```
auto oob_net0
iface oob_net0 inet static
    address 192.168.200.2
    netmask 255.255.255.0
    network 192.168.200.0
```

- CentOS configuration of IP interface:
 - Configuration file for "oob_net0" interface: /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
```

- Ubuntu configuration of IP interface:

- Configuration file for "oob_net0" interface: /etc/network/interfaces.d/oob_net0
- For example, use the following to enable DHCP configuration:

```
auto oob_net0
iface oob_net0 inet dhcp
```

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

```
auto oob_net0
iface oob_net0 inet static
    address 192.168.200.2
    netmask 255.255.255.0
    network 192.168.200.0
```

OpenOCD on BlueField

To run OpenOCD (On-chip debugger) for BlueField:

1. Start host-side RShim drivers if not already started (assuming they have already been installed). Run:

```
$ sudo systemctl start rshim
```

Find the RShim device—it is usually located at /dev/rshim0/rshim.

2. Run OpenOCD:

```
$ sudo <BF_INST_DIR>/bin/mlx-openocd
```

Once started, OpenOCD runs a gdb-server in the background to accept commands from a GDB client. This script will start OpenOCD over /dev/rshim0/rshim by default. In order to use a different RShim, modify <BF_INST_DIR>/lib/openocd/target/bluefield.cfg and set the "rshim device xxx" line before running the mlx-openocd script.

To start the GDB client:

1. Set up the cross-compiler toolchain environment. For example:

```
$ . <SDK_DIR>/environment-setup-aarch64-poky-linux
```

2. Run the GDB client:

```
$ aarch64-poky-linux-gdb [optional_elf_image]
(gdb) target remote :3333 # Or <IP>:3333 if running from different machine
(gdb) bt
(gdb) <...normal gdb commands...>
```

For more information, please refer to the <BF_INST_DIR>/Documentation/HOWTO-openocd README.

UEFI Boot Option 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.

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)

List UEFI Boot Options

To display the boot option already installed in the BlueField® system, reboot and go to the UEFI menu screen. To get to the UEFI menu, just hit any key when the screen rolls up after printing the UEFI firmware version.

```
UEFI firmware (version 0.99-e2bbe24 built at 18:38:55 on Apr  5 2018)
```

Boot options are listed as soon as you select the “Boot Manager” entry.

```
Boot Option Menu                                     Device Path :
Linux from rshim                                     VenHw(F019E406-8C9C-11
Yocto Poky                                           E5-8797-001ACA00BFC4) /
EFI Misc Device                                     Image
EFI Network
EFI Network 1
EFI Network 2
EFI Network 3
EFI Internal Shell
```

And to change option, ENTER to select an option, ESC to 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.

Below an example of output:

```
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.O. .*
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.O. .*
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.

Creating, Deleting, and Modifying UEFI Boot Option

The file system supported by EFI is based on the FAT file system. An “EFI system partition” (or ESP) is any partition formatted with one of the UEFI spec-defined variants of FAT and given a specific GPT partition type to help the firmware read it.

Usually, The ESP is located in “FS0:”. To create a new boot entry, run:

```
Shell> bcfg boot add <option#> <file-path> "<description>"
```

The parameter “add” is used to add an option. The “option#” is the option number to add in hexadecimal. The “file-path” is the path of the UEFI binary for the option. The quoted parameter is the description of the option being added.

For example, to create a boot entry to boot “Yocto Poky” as a default option, run:

```
Shell> bcfg boot add 0 FS0:\Image "Yocto Poky"
```

Where “Image” is the actual kernel image to boot from the MMC partition.

To create a boot entry for CentOS, assuming the distro is already installed and the ESP formatted properly, run:


```
Shell> bcfg boot add 2 FS0:\EFI\centos\shim.efi "CentOS 7.4"
```

The boot entry here is installed as a third boot option (option number starts from 0). “shim.efi” is a trivial EFI application that, when run, attempts to open and execute another application (e.g. GRUB bootloader).

To add booting parameters to the boot options, you need to create a file, and then append it to the boot option:

```
Shell> edit FS0:\options.txt
```

Add a single line boot arguments to the file in “FS0:\options.txt”, save the file (UCS-2). Finally append the arguments:

```
Shell> bcfg boot -opt 0 FS0:\options.txt
```

Boot arguments here are appended to boot option #0. Do not run this command several times. You have to remove and re-add the entry before you can change the parameters.

To modify the boot option order, for example, to move boot option #2 to boot option #0, simply run:

```
Shell> bcfg boot mv 2 0
```

The first numeric parameter is the option to move. The second numeric parameter is the new option number.

Finally, to remove a boot option, you may run:

```
Shell> bcfg boot rm 0
```

The numeric parameter refers to the option number to remove.

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.
- Disable PCIe: Disable PCIe in ACPI. Default: Enabled.
- Reset EFI Variables: Clears all EFI variables to factory default state.
- Reset MFG Info: Clears the manufacturing information.

Installing Popular Linux Distributions on BlueField

Building Your Own BFB Installation Image

Users wishing to customize their own kernel or root file system may do so using docker images supplied by NVIDIA. BlueField docker images can be obtained at: <https://hub.docker.com/repository/docker/mellanox/bluefield>.

Docker images give users the ability to customize their own Linux distribution and create a BFB. That BFB file can then be installed on any BlueField device. Instructions are provided on the BlueField docker page as well as in the docker image itself.

Users can build all BlueField software, the Linux kernel, and OFED using the docker image and "native" or "cross" compile:


- Build using QEMU emulation on an x86 server
- Build on an Arm system (such as a BF1200 or an AWS Arm EC2 instance)
- The same docker image is used regardless of if you build native on an Arm server or use emulation on an x86 server. The only difference is how long the build takes.

Installing Linux Distribution With BFB Installation Image

Select an image according to the usage model and push it from the host machine via the RShim interface (USB or PCIe).

```
$ cat <Distro.bfb> > /dev/rshim0/boot
```

Wait for about 10-15 minutes. At the end of the installation BlueField will be rebooted and ready to use.

 Pre-built BFBs of CentOS 7.6, Ubuntu 20.04, and Debian 10 with MLNX_OFED_LINUX installed are available. Please contact NVIDIA Support to obtain these BFBs.

CentOS Partitioning on BlueField Device

The CentOS BFBs released by NVIDIA support two partitioning schemes:

- SCHEME_A (default partitioning for CentOS):

```
/dev/mmcblk0p1 200M /boot/efi  
/dev/mmcblk0p2 1014M /boot  
/dev/mmcblk0p3 13G /
```

- SCHEME_B:

```
/dev/mmcblk0p1 200M /boot/efi  
/dev/mmcblk0p2 4.0G /  
/dev/mmcblk0p3 8.0G /home  
/dev/mmcblk0p4 1.6G /var
```

To choose one of these schemes create the file "bf.cfg" on the host with the following line:

```
PART_SCHEME=<name>
```

Where "name" is either SCHEME_A or SCHEME_B.

Then run:


```
cat <CentOS.bfb> bf.cfg > /dev/rshim0/boot
```

Installing Official CentOS 7.x Distribution

This section provides instructions on how to set up a local PXE boot environment to install the official Arm-based CentOS 7.x distribution on a BlueField system.

Requirements

- Host machine running CentOS 7.

 CentOS 6.2+ would need slight modification in the "setup.sh" script to set up the tftp/dhcpd services.

- BlueField prebuilt packages installed under the directory BF_INST_DIR. If they are not installed yet, get the tarball file BlueField-1.2.0.xxxxxx.yyyyy.tar.xz and run:

```
$ tar Jxvf BlueField-1.2.0.xxxxxx.yyyyy.tar.xz -C <PATH>.
```

The "<BF_INST_DIR>" could then be found under the directory "<PATH> / BlueField-1.2.0.xxxxxx.yyyyy".

Host Machine Setup

1. Download the CentOS installation ISO file from the following URL: <http://archive.kernel.org/centos-vault/altarch/7.4.1708/isos/aarch64/CentOS-7-aarch64-Everything.iso>.


If ConnectX interfaces are expected during the installation (rather than installing OFED later), download the "mlnx-ofed" file from the [Mellanox Flexboot](#) webpage by selecting "DUD & kIOS Download" → Version (e.g. 4.2-1.4.10.0) → RHEL/CentOS 7.4 → aarch64 → dd-rhel7.4-mlnx-ofed-4.2-1.4.10.0-aarch64.iso.gz. Download the file and decompress it. (This step is needed if you are performing PXE boot over the ConnectX interface.)


2. Navigate to "pxeboot" directory:

```
$ cd <BF_INST_DIR>/distro/rhel/pxeboot
```

3. Run the setup script:

```
$ ./setup.sh -d <BF_INST_DIR> -i <centos-installation.iso> [-c <ttyAMA0 | ttyAMA1 | rshim>] [-o ofed-dud.iso] [-k]
```

 The option "-k" enables automatic installation according to the kickstart file ks.cfg. This option is recommended. When specified, the sample centos.ks is installed under /var/pxe/ks/ which is used for CentOS installation.

 UART1 (ttyAMA1) is used by default. To specify a different console use "-c xxx". SmartNIC uses UART0, so it takes "-c ttyAMA0". For SmartNICs without UART connections, please use "-c rshim".

⚠ A nonpxe.bfb is also generated which can be used to boot the device via the RShim interface. It starts CentOS installation by skipping the UEFI PXE process which should be faster. The nonpxe.bfb is generated on the same directory where the ./setup.sh is run. So you must have write permissions on this directory.

⚠ SELinux is used on CentOS/RedHat to orchestrate what programs are allowed to do, especially with regards to network access. The "setup.sh" script relies on setting up the server host as a network server and providing DHCP/PXE/NAT functionalities to the BlueField. SELinux may not allow this behavior. For example, if pinging the server host machine from the BlueField does not work, it could be that SELinux is disallowing the host. Perform "setenforce 0" on the server host in order to temporarily disable SELinux to verify whether this is the case.

Basic Yocto Installation

1. Connect the UART console.
Find the device file and connect to it using minicom or screen. For example:

```
$ screen /dev/ttyUSB0 115200
```

Use "yum install screen" or "yum install minicom" to install minicom/screen if not found.
For minicom, set:

- Bps/Par/Bits - 115200 8N1
- Hardware Flow Control - No
- Software Flow Control - No

2. Power cycle the board.
3. Select an image according to the board type and push it from the host machine via the RShim interface (USB or PCIe).

```
$ cat <BF_INST_DIR>/sample/install.bfb > /dev/rshim0/boot
```

4. Log into Linux from the UART console (root with no password). Run the following script to flash the default image and wait until it is done. The board will boot into Linux.

```
$ /opt/mlnx/scripts/bfinst --minifs
```

This step is needed to update the boot partition images.

RShim PXE Boot

1. Reboot the board. Once the "UEFI firmware ..." message appears on the UART console, press the "Esc" key several times to enter the UEFI boot menu.
2. Restart the dhcpd/tftp-server services. Run:

```
$ systemctl restart dhcpd; systemctl restart xinetd
```

3. Select the "Boot Manager" in the UART console and press Enter. Then select "EFI Network" in the Boot Manager and press Enter to start the PXE boot.
4. Check the Rx/Tx statistics on host side. Run:

```
$ ifconfig tmfifo_net0
```

5. After some time, a list of OS appears. Select "Install centos/7.4 AArch64 - BlueField" and press Enter to start the CentOS installation.

⚠ It takes time to fetch the Linux kernel image and initrd. So please be patient and check the Rx/Tx packet counters. The installation starts when the counters reach ~50K.

OOB PXE Boot

⚠ The OOB interface is available only on BlueField-2 based devices.

1. Set up the PXE boot environment by running the setup.sh script on the server host:

```
/root/BlueField-3.0.alphaX.XXXXX/distro/rhel/pxeboot/setup.sh \  
-d /root/BlueField-3.0.alphaX.XXXXX/ \  
-i /root/CentOS-7-aarch64-Everything.iso \  
-o /root/dd-rhel7.4-mlnx-ofed-4.2-1.4.10.0-aarch64.iso \  
-c ttyAMA0 \  
-p yocto-full-image-path \  
-k \  
-s <oob-interface> \  
-m <oob-mac>
```

Where:

- -d points to the location from which the tar file has been extracted. The script uses this directory to find all the source code it needs.
- -i points to the OS installation disk. This is the image that is accessed via PXE boot to install the OS on BlueField-2.
- -o points to the MLNX_OFED driver disk for Arm. Download and extract it from the [Mellanox FlexBoot](#) page.
- -c specifies the default UART port for the OS to use since the BlueField-2 DPU has two Arm UARTs. Use "ttyAMA0" for BlueField-2, which is UART0.
- -p (optional) points to the Yocto full image path.
- -k (optional) kickstarts auto-installation based on a default kickstart file which is installed as /var/pxe/ks/oob_centos.ks or /var/pxe/ks/oob_yocto.ks.
- -s points to the name of the OOB network interface on the server host. It can be retrieved from the host via "ifconfig -a".

⚠ This parameter needs to be passed along with -m <oob-mac>.

- -m points to the BlueField-2 OOB MAC address, which can be retrieved from the BlueField prompt via "ifconfig oob_net0". The format should be xx:xx:xx:xx:xx:xx.

⚠ This parameter needs to be passed along with -s <oob-interface>.

2. Reboot the board. Once the "UEFI firmware ..." message appears on the UART console, press the "Esc" key several times to enter the UEFI boot menu.
3. Select the "Boot Manager" in the UART console and press Enter.
4. Then from the Boot Manager, select "EFI Network" which corresponds to the IPv4 OOB network interface, and press Enter to start the PXE boot.
5. Check the Rx/Tx statistics on host side. Run:

```
$ ifconfig <oob-interface>
```

6. After some time, a list of OS appears. Select "Install centos/7 AArch64 via OOB - BlueField" to boot via OOB. Then press Enter to start the CentOS installation.

CentOS Installation

Follow the installation wizard.

Post-installation

1. Enable "yum install" or external network access after CentOS installation.
 - a. On the host side, run:

```
$ systemctl restart dhcpd
$ echo 1 > /proc/sys/net/ipv4/ip_forward
$ iptables -t nat -A POSTROUTING -o <out_intf> -j MASQUERADE
'<out_intf>' is the outgoing network interface to the network.
```

- b. Check the IP address for tmfifonet0 on the BlueField device side. Run:

```
$ ifconfig tmfifonet0
tmfifonet0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.100.2 netmask 255.255.255.0 broadcast 192.168.100.255
inet6 fe80::21a:caff:feff:ff01 prefixlen 64 scopeid 0x20<link>
ether 00:1a:ca:ff:ff:01 txqueuelen 1000 (Ethernet)
RX packets 95 bytes 14106 (13.7 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 104 bytes 13219 (12.9 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

If an IP address is not assigned, run:

```
$ ifdown tmfifonet0; ifup tmfifonet0
```

2. Install driver RPMs from source.

 This step is not needed if OFED is installed.

- If "rpmbuild" is not available, run:

```
$ yum install rpm-build
```

- If development tools are not available, run:

```
$ yum group install "Development Tools"
```


- If kernel-devel is not installed, run:

```
$ yum install kernel-devel-`uname -r`
```


All driver source RPMs are located at <BF_INST_DIR>/distro/SRPMS. Upload them to the target (e.g. under /opt).

The following is an example which exhibits how to install i2c-mlnx-1.0-0.g6af3317.src.rpm.

```
$ cd /opt
$ rpmbuild --rebuild i2c-mlnx-1.0-0.g6af3317.src.rpm
$ cd ~/rpmbuild/RPMS/aarch64/
$ rpm -ivh i2c-mlnx-1.0-0.g6af3317_4.11.0_22.e17a.aarch64.rpm
```

 The tmfifo driver is also included in the initramfs. Remove it from initramfs as instructed (3.a) below when upgrading the tmfifo driver.

3. Install OFED (optional) as instructed in section "[Installing MLNX_OFED on DPU](#)".

 If MLNX_OFED_LINUX is installed with "--add-kernel-support", run "dracut -f" to update drivers in initramfs after MLNX_OFED_LINUX installation.

Building a New bluefield_dd ISO Image

To build an updated driver disk for a different version of RHEL, you may use the sources located in the "bluefield_dd" directory. Run the build-dd.sh script there and provide it as an argument the path to the kernel-devel RPM, and it then builds a matching driver disk.

Typically, you would do this as a cross-build when initially booting up a BlueField system, so the cross-build environment must be configured prior to running the script.

Note that for the Yocto SDK, you must "unset LDFLAGS" before running the script, since the kernel build uses raw LDFLAGS via "ld" rather than via "gcc" as the Yocto SDK assumes.

So, for example:

```
source /path/to/SDK/environment-setup-aarch64-poky-linux
unset LDFLAGS
./build-dd.sh /path/to/kernel-devel-4.11.0-22.el7a.aarch64.rpm
```

This generates a suitable .iso file in the current directory.

PXE Boot Flow

UEFI allows booting over PXE in the same way that is familiar with other operating system installations.


If the BlueField eMMC is already provisioned with a bootstream containing ATF and UEFI, it should power on and boot up with that bootstream. If the eMMC also contains a bootable kernel, you would need to interrupt the boot by hitting "Esc" quickly once UEFI starts up. This takes you to the UEFI main menu on the serial console. If the eMMC is provisioned with a bootstream but does not contain a bootable kernel, you would enter the UEFI main menu on the serial console automatically at power on.

If the eMMC is not yet provisioned with a bootstream, you would need to boot it externally using USB or PCIe. In that case, providing a bootstream containing only ATF and UEFI would boot the chip and you automatically enter the UEFI main menu on the serial console.

From this point, you may navigate to the network boot option and select the primary ConnectX network Interface; or the OOB network interface; or select the RShim network interface, which would bridge to the external host over USB or PCIe, and use its network interface instead.

At this stage, you should be able to boot the CentOS installation media from a PXE server in the normal manner, loading Grub and then selecting your kernel of choice from the media.

RShim PXE Boot Over VLAN

 This function is available only on BlueField-2 DPU.

This section describes how to perform Yocto/CentOS PXE boot and installation over VLAN interface. For simplicity, this procedure sets up the PXE server on the server host machine which has the RShim (USB or PCIe) connection to the BlueField board, and assumes that the BlueField release tarball has been installed under directory <BF_INSTALL>.

1. Set up the PXE server on the server host as root.

```
$ cd <BF_INSTALL>/distro/rhel/pxeboot/  
$ ./setup.sh -i <centos-iso> -d <BF_INSTALL> -c <ttyAMA0 | ttyAMA1 | rshim> -k -p <yocto-full-image-path>
```

The parameter `-p <yocto-full-image-path>` is not needed for CentOS. Once specified, however, the script sets up PXE server environment for Yocto as well. Type `./setup.sh` to see more help information.

The default grub configuration file is `/var/lib/tftpboot/grub.cfg`. It can be customized it as needed, such as adding VLAN support (see step 4 below).

The default kickstart file is `/var/pxe/ks/yocto.ks` for Yocto or `/var/pxe/ks/centos.ks` for CentOS. They can be customized as well.

Note that this script also copies the patched `grubaa64.efi` file from the directory `<BF_INST_DIR>/distro/rhel/pxeboot/patches/` to `/var/lib/tftpboot/`. This file is needed since the default `grubaa64.efi` from the CentOS ISO does not support PXE over VLAN. See section "[Patch to Support PXE Boot Over VLAN](#)" for more details about this patch.

Once done, reset BlueField and enter the UEFI Boot Manager to pick the network interface for PXE boot.

2. Configure VLAN in UEFI.

In order to PXE boot over VLAN, VLAN needs to be created in UEFI and on the server host. VLANs can be created inside the UEFI boot menu. Once created, they are stored in the UEFI persistent variables. It is possible to write the UEFI variables directly if needed without having to enter the UEFI boot menu. Press "ESC" key to enter UEFI boot menu, then follow the steps below.


```

2.1) Select "Device Manager".
=====
Continue
Select Language          <Standard English>      This selection will
Boot Manager              take you to the
> Device Manager          Device Manager
Boot Maintenance Manager

2.2) Select "Network Device List".
=====
Devices List              Select the network
Secure Boot Configuration device according the
iSCSI Configuration       MAC address
> Network Device List
Driver Health
    The platform is healthy

2.3) Select "MAC:00:1A:CA:FF:FF:01".
=====
Network Device List      Network Device
> MAC:00:1A:CA:FF:FF:01
MAC:50:6B:4B:08:CE:EB
MAC:50:6B:4B:08:CE:EC
Driver Health
    The platform is healthy

2.4) Select "VLAN configuration".
=====
Network Device           VLAN Configuration
> VLAN Configuration     (MAC:001ACAFFFF01)
Driver Health
    The platform is healthy

2.5) Select "Enter Configuration Menu".
=====
> Enter Configuration Menu      Press ENTER to enter
                                configuration menu
                                for VLAN configuration.

2.6) Fill in the VLAN ID value, then move down to "Add VLAN" and press enter.
=====
Create new VLAN           VLAN ID of new VLAN
VLAN ID                   [100 ]      or existing VLAN,
Priority                   [0]        valid value is 0~4094
Add VLAN

Configured VLAN List
Remove VLAN

2.7) Repeat step 2.6 to create all the VLANs. They will have an output like the
one presented below. Select the VLAN ID and run the action "Remove VLAN" to
delete the selected VLAN.
=====
Create new VLAN           Create a new VLAN or
VLAN ID                   [0]        update existing VLAN
Priority                   [0]
Add VLAN

Configured VLAN List
VLAN ID: 100, Priority:0 [ ]
VLAN ID: 200, Priority:0 [ ]
Remove VLAN

2.8) Reboot or Press 'ESC' keys to go back to the "Boot Manager".
VLAN interfaces will be show up in the EFI Network list. Select it and press
enter to start PXE boot over this VLAN.
=====
Boot Option Menu          Device Path :
                           MAC(001ACAFFFF01,0x1)/
                           Vlan(100)/IPv4(0.0.0.0)

EFI Network
> EFI Network 2
EFI Network 3
CentOS 7.4
EFI Network 6
EFI Network 7
EFI Internal Shell
Linux from mmc0
EFI Misc Device
...

```

If needed, the boot order can be modified in the UEFI shell to boot from VLAN interface by default as follows:

```
Shell> bcfg boot mv 4 0
```

This moves boot entry 4 to 0.

3. Create VLAN interface on the server host.

The VLAN interface needs to be configured on the server host side, such as modifying `/etc/sysconfig/network-scripts/ifcfg-tmfifo_net0` by changing `DEVICE=tmfiffo_net0` to

DEVICE=tmfifonet0.100 and adding line VLAN=yes.

Then restart the RShim service (systemctl restart rshim) to make it take effect.

4. Configure VLAN in /var/lib/tftpboot/grub.cfg on the server host. The examples below enable VLAN 100 and 200 on the RShim network interface tmfifonet0.

- For Yocto:

```
menuentry 'Install Yocto AArch64 - BlueField' --class red --class gnu-linux --class gnu --class os
{
    linux /yocto/vmlinuz ro ip=dhcp console=ttyAMA0 bfks=http://192.168.100.1/ks/yocto.ks
    bfnet="tmfifonet0.100:dhcp" bfnet="tmfifonet0.200:dhcp" initrd /yocto/initrd.img
}
```

- For CentOS:

```
menuentry 'Install centos/7 AArch64 - BlueField' --class red --class gnu-linux --class gnu --class os
{
    linux (tftp)/centos/7/vmlinuz ro ip=dhcp repo=http://192.168.100.1/centos7 inst.dd=/
    bluefield_dd.iso vlan=tmfifonet0.100:tmfifonet0 vlan=tmfifonet0.200:tmfifonet0 console=ttyAMA1
    initrd (tftp)/centos/7/initrd.img
}
```

In order to perform PXE over VLAN for CentOS, /var/pxe/ks/centos.ks needs to be updated as well by adding --vlanid=100 --vlanid=200 to the line of network --bootproto... to create VLAN 100 and 200.

Patch to Support PXE Boot Over VLAN

The directory <BF_INST_DIR>/distro/rhel/pxeboot/patches/ contains patches to the default grub from CentOS to support PXE boot over VLAN, and a prebuilt grubaa64.efi binary in order to simplify matters. The patch is based on the <https://git.centos.org/r/rpms/grub2.git> branch remotes/origin/c7-alt. The following is the top commit of this branch after applying all the existing patches according to the RPM SPEC file:

```
commit 66d2044d7fb6c3bd0b170a9b0828d9a0b5dce582
Author: CentOS Sources <bugs@centos.org>
Date: Tue Oct 30 01:04:32 2018 -0400

debrand grub2-2.02-0.76.el7
```

Copy the patches to the SOURCE/ directory and add them into SOURCE/grub.patches as provided below. Then rebuild the RPM SPEC file to generate the grubaa64.efi.

```
diff --git a/SOURCES/grub.patches b/SOURCES/grub.patches
index 26b949d..38f7b03 100644
--- a/SOURCES/grub.patches
+++ b/SOURCES/grub.patches
@@ -286,3 +286,6 @@ Patch0285: 0285-editenv-handle-relative-symlinks.patch
Patch0286: 0286-efinet-also-use-the-firmware-acceleration-for-http.patch
Patch0287: 0287-Make-root_url-reflect-the-protocol-hostname-of-our-b.patch
Patch0289: 0288-efi-uga-Fix-PCIe-LER-when-GRUB2-accesses-non-enabled.patch
+Patch0290: 0290-efinet-Add-EFI-boot-path-support-for-VLAN.patch
+Patch0291: 0291-Fix-potential-IP-fragment-issue.patch
+Patch0292: 0292-Increase-tftp-blksize-to-boost-performance.patch
```

This VLAN issue and patches have been submitted to CentOS Bug Tracker <https://bugs.centos.org/view.php?id=15307>.

OoB PXE Boot Over VLAN



The OoB interface is available only on BlueField-2 based devices.

This section describes how to perform Yocto/CentOS PXE boot and installation over VLAN interface. For simplicity, this procedure sets up the PXE server on the server host machine which has the OoB

connection to the BlueField board, and assumes that the BlueField release tarball has been installed under directory <BF_INSTALL>.

1. Please follow the steps in "[OOB PXE Boot](#)"
2. Configure VLAN in UEFI.

In order to PXE boot over VLAN, VLAN needs to be created in UEFI and on the server host. VLANs can be created inside the UEFI boot menu. Once created, they are stored in the UEFI persistent variables. It is possible to write the UEFI variables directly if needed without having to enter the UEFI boot menu. Press "ESC" key to enter UEFI boot menu, then follow the steps below.

```

2.1) Select "Device Manager".
=====
Continue                                     This selection will
Select Language          <Standard English> take you to the
Boot Manager              Device Manager
> Device Manager
  Boot Maintenance Manager

2.2) Select "Network Device List".
=====
Devices List                               Select the network
Secure Boot Configuration                  device according the
iSCSI Configuration                       MAC address
> Network Device List
  Driver Health
    The platform is healthy

2.3) Select the OOB MAC Address
=====
Network Device List                         Network Device
> MAC:00:1A:CA:FF:FF:01
  MAC:50:6B:4B:08:CE:EB
  MAC:50:6B:4B:08:CE:EC
  Driver Health
    The platform is healthy

2.4) Select "VLAN configuration".
=====
Network Device                             VLAN Configuration
> VLAN Configuration                       (MAC:******)
  Driver Health
    The platform is healthy

2.5) Select "Enter Configuration Menu".
=====
> Enter Configuration Menu                 Press ENTER to enter
                                           configuration menu
                                           for VLAN configuration.

2.6) Fill in the VLAN ID value, then move down to "Add VLAN" and press enter.
=====
Create new VLAN                               VLAN ID of new VLAN
VLAN ID                                     or existing VLAN,
Priority                                     valid value is 0~4094
Add VLAN

Configured VLAN List
Remove VLAN

2.7) Repeat step 2.6 to create all the VLANs. They will have an output like the
one presented below. Select the VLAN ID and run the action "Remove VLAN" to
delete the selected VLAN.
=====
Create new VLAN                               Create a new VLAN or
VLAN ID                                     update existing VLAN
Priority                                     [0]
Add VLAN                                     [0]

Configured VLAN List
VLAN ID: 100, Priority:0 [ ]
VLAN ID: 200, Priority:0 [ ]
Remove VLAN

2.8) Reboot or Press 'ESC' keys to go back to the "Boot Manager".
VLAN interfaces will be show up in the EFI Network list. Select it and press
enter to start PXE boot over this VLAN.
=====
Boot Option Menu                             Device Path :
                                           MAC(*****,0x1)/
                                           Vlan(100)/IPv4(0.0.0.0)
> EFI Network
  EFI Network 2
  EFI Network 3
  CentOS 7.4
  EFI Network 6
  EFI Network 7
  EFI Internal Shell
  Linux from mmc0
  EFI Misc Device
  ...

```

If needed, the boot order can be modified in the UEFI shell to boot from VLAN interface by default as follows:

```
Shell> bcfg boot mv 4 0
```

This moves boot entry 4 to 0.

3. Create OOB VLAN interface on the server host.

The VLAN interface needs to be configured on the server host side, by:

```
cp /etc/sysconfig/network-scripts/ifcfg-<oob-interface> /etc/sysconfig/network-scripts/ifcfg-<oob-interface>.100
```

Change `DEVICE=<oob-interface>` to `DEVICE=<oob-interface>.100` and add line `VLAN=yes`.

```
vi /etc/sysconfig/network-scripts/ifcfg-em2  
#comment out this line "IPADDR=192.168.101.1"
```

4. Restart the networking service in order for the changes to take effect. As root issue the following command:

```
systemctl restart network
```

5. Configure VLAN in `/var/lib/tftpboot/grub.cfg` on the server host. The examples below enable VLAN 100 and 200 on the OOB network interface.

- For Yocto:

```
menuentry 'Install Yocto AArch64 - BlueField' --class red --class gnu-linux --class gnu --class os  
{  
  linux /yocto/vmlinuz ro ip=dhcp console=ttyAMA0 bfks=http://192.168.101.1/ks/yocto.ks bfnet=<oob-interface>.100:dhcp" bfnet=<oob-interface>.200:dhcp" initrd /yocto/initrd.img  
}
```

- For CentOS:

```
menuentry 'Install centos/7 AArch64 - BlueField' --class red --class gnu-linux --class gnu --class os {  
  linux (tftp)/centos/7/vmlinuz ro ip=dhcp repo=http://192.168.101.1/centos7 inst.dd=/bluefield_dd.iso vlan=<oob-interface>.100:<oob-interface> vlan=<oob-interface>.200:<oob-interface> console=ttyAMA1 initrd (tftp)/centos/7/initrd.img  
}
```

In order to perform PXE over VLAN for CentOS, `/var/pxe/ks/oob_centos.ks` needs to be updated as well by adding `--vlanid=100 --vlanid=200` to the line of `"network --bootproto..."` to create VLAN 100 and 200.

Non-PXE Boot Flow

It is possible to explicitly boot the PXE boot components of CentOS directly over USB or PCIe to avoid the requirement of a PXE server being available on the network. This is somewhat equivalent to booting a local bootable CD and then using another media source to find all the packages to install.

The root of the CentOS install image, for example `$ROOT`, corresponds to the root of the ISO image file; that is, the directory which contains EFI, EULA, GPL, LiveOS, Packages, etc. You should create a bootstream which includes the pxeboot kernel and initramfs and use that to boot the image. This is assuming the `initrd.img` file in this directory has already been updated to include `bluefield_dd.iso`, as described in the previous section.

You must also determine from where to load the ISO image for the installation. In this example, that destination is `http://1.2.3.4/rhel`.

After the kernel image is uncompressed, it must be placed, along with the `initrd.img`, in a BlueField bootstream. To do so, `cd` to the `samples` directory of the BlueField Runtime Distribution, make sure that the "bin" directory is on your `$PATH`, and run:

```
gunzip < $ROOT/images/pxeboot/vmlinuz > /tmp/Image
build-bfb \
--bfb ../boot/default.bfb \
--kernel Image \
--initramfs $ROOT/images/pxeboot/initrd.img \
--bootarg "inst.dd=/dw_mmc.iso inst.repo=http://1.2.3.4/rhel" \
--no-gpt -i rshim pxeboot.bfb
```

This bootstream can then be used to boot the BlueField in the normal way. It comes up in text mode on the console; you may also select to run the installer from a VNC client. When running the `setup.sh` script, a `nonpxe.bfb` is generated in the same way.

Installation Troubleshooting and FAQ

How to reset the board or NIC via the RShim interface from host side


Run the following:

```
$ echo "SW_RESET 1" > /dev/rshim<N>/misc
```

Make sure to replace "rshim<N>" with the actual name (e.g. `rshim0`).

How to upgrade existing CentOS to some BlueField release without reinstallation

1. Follow Step 3 of section "[Basic Yocto Installation](#)" to push the installation image via RShim (USB or PCIe).

 Do not run the `bfinst` script, or else a new installation will start.

2. Run `/opt/mlnx/scripts/bfrec` to upgrade the boot partitions. This step upgrades the ATF & UEFI images to this release.
3. Reboot into CentOS. Follow Step 2 of section "[Post-installation](#)" to install/upgrade the `tmfifo` driver and other drivers as needed from the source RPM.

To re-run the "setup.sh" script after host reboot before installing CentOS

Either re-run the script, or mount `/var/pxe/centos7` to the CentOS ISO file.

How to change the MAC address of the `tmfifo` network interface (Arm side)

See section "[Permanently Changing the MAC Address of the Arm Side](#)".

Why CentOS (Arm side) did not get DHCP address on `tmfifo` interface (`eth0`) after re-boot

The host-side DHCP daemon must be restarted after board reboot in order to provide DHCP service. A configuration file could accomplish this automatically.

```
$ Create /sbin/ifup-local or add to it.
INTF=$1
if [ "$INTF" = "tmfifo_net0" ]; then
    systemctl restart dhcpd
fi
```

How to kickstart auto-installation

Run `setup.sh` with the `"-k"` option. The default kickstart file is installed as `/var/pxe/ks/ks.cfg`. It should have all packages needed for OFED. Add more packages if needed.

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 bootflow 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.

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.

Note that, 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.

Managing Driver Disk

As discussed previously, you likely need a driver disk for RedHat installations. NVIDIA provides a number of pre-built driver disks, as well as a documented flow for building one for any particular RedHat version. See section "[Building a New bluefield_dd ISO Image](#)" for details on how to do that.

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 typically 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).

Installing Reference Yocto Distribution

⚠ These steps are also documented in sample/README-install.

It is assumed that you have [installed and loaded the RShim driver](#) on your host and have a console connection to the Arm cores.

Run the following on the host:

```
$ cat <BF_INST_DIR>/sample/install.bfb bf.cfg core-image-full-dev-bluefield.tar.xz > /dev/rshim0/boot
```

The file `core-image-full-dev-bluefield.tar.xz` is the Yocto rootfs tarball. Other tarball could also be used, such as `core-image-full-bluefield.tar.xz`.

The BFB image may also be installed using the "[bfb-install](#)" utility script.

Maintaining User Data After Software Upgrade

By default, when installing Yocto for the first time, a special partition is created and mounted to `/data`. This is the persistent partition.

Any files written to this partition are treated specially by the installer, and are not overwritten on subsequent installs of Yocto.

To forcefully format the `/data` partition on install, run `bfinst` with the `--fmtpersist` option during the install process.

This section describes the persistence features of the BlueField Yocto distribution.

Persistent Configuration

Yocto offers a persistent configuration feature that uses the persistent partition.

To enable it, create an empty file called "persistconfig" at the root of the data partition, and a folder called "etc", then reboot.

```
$ mkdir /data/etc
$ touch /data/persistconfig
$ reboot
```

Upon startup, `/etc` is mounted as a special overlay filesystem. Any changes made to `/etc` in this state persist after software upgrades.

❗ This feature depends on a file called `/etc.img`; do not delete it.

Do not modify `/data/etc` while `persistconfig` is enabled. To disable `persistconfig`, just delete the file in `/data`:

```
$ rm /data/persistconfig
$ reboot
```

Upon reboot, `/etc` is mounted from the root filesystem, and any changed files are reverted to what they were before `persistconfig` was enabled. Any changes made during `persistconfig` mode will be remembered in `/data/etc`.

❗ Any `/etc` files updated during a software upgrade are masked if any user-made changes are in persistent `/etc` mode. For example, if you modified `/etc/foo`, and a software upgrade updates `/etc/foo`, the vendor-made changes will not be visible, and manual intervention will be required.

Understanding Persistent Data Partition Feature Implementation

The persistent partition `/data` is created at install time with a known GUID. Upon subsequent installs, the `bfinst` script will check for a partition with this GUID, and if it exists, will leave it intact and only format the other partitions.

To implement persistent configuration, an overlayfs has been used (see [Documentation/filesystems/overlayfs.txt](#) in the Linux source tree for more information). When persistence is turned on, after the data partition is mounted, the script checks that `/data/persistconfig` exists. If it does, `/etc` is mounted as an overlayfs. The lower filesystem is `/etc.img`, a read-only etc image created at install time, and the upper filesystem is `/data/etc`, residing on the persistent partition.

When the persistent config is first turned on, all read accesses to `/etc` will go to the lower filesystem. But, when something is written to `/etc`, the file being written is copied from the lower to the upper filesystem, and then opened in write mode on the upper filesystem, redirecting all writes. Any further reads and writes on that file will now go to the copy residing on the upper

filesystem. The overlayfs effectively "merges" the two folders and presents a new filesystem composed of both on `/etc`.

Since all changes to `/etc` are automatically written to `/data/etc`, all configuration changes survive software upgrades.

The `/data` and `/etc` mounts are the first initialization tasks that happen on the system, preempting even `systemd`. This is accomplished with the script `/sbin/preinit`, which performs the mounts and then execs into `/sbin/init`. A kernel argument is passed by grub to force the kernel to use `/sbin/preinit` as its initialization process.

The reason this is done is to get around an issue involving `systemd`, and to ensure any initialization processes open files in the `/etc` overlayfs, not the rootfs. The `systemd` software reads its configuration from two locations: `/etc/systemd` and `/lib/systemd`. It is also normally responsible for all mounts in the system. When the user adds their own services to `/etc/systemd`, it is possible to get `systemd` to pick up the new changes when `/etc` is mounted. Unfortunately, it is not possible to execute newly acquired services during the boot process. To fix this natively with `systemd` is very complex and unreliable, so we just go under `systemd` entirely to simplify the solution.

Installing Ubuntu on BlueField

Ubuntu Hardware and Software Requirements

- Host system with [RShim driver installed](#)
- BlueField NIC inserted in a host PCIe slot
- USB cable connecting the NIC card and the host

Installation Procedure for Canonical's Ubuntu BFB

1. If Ubuntu 18.04 is installed on the host, make sure to have the HWE kernel also. Run:

```
$ sudo apt install linux-signed-generic-hwe-18.04
$ sudo reboot
```

2. Install the image by pushing the downloaded image to the NIC. Run:

```
$ sudo sh -c 'xzcat server-2004-ubuntu-20.04-alpha2-20200725-66.bfb.xz > /dev/rshim0/boot'
```

This will start an Ubuntu kernel that automatically installs an image contained in the initramfs included inside the bfb. After flashing this image, the installation script reboots, and eventually a login prompt appears in the serial console.

3. The BFB image may also be installed using the ["bfb-install" utility script](#). You would have to uncompress the `.bfb.xz` file to `.bfb` for it to work.

Ubuntu With MLNX_OFED Installation

A pre-built BFB of Ubuntu 18.04 with `MLNX_OFED_LINUX` installed is available. Please contact NVIDIA Support to get this BFB.

To install Ubuntu BFB, run on the host side:

```
$ cat <ubuntu.bfb> > /dev/rshim0/boot  
Username: ubuntu  
Password: ubuntu
```

Installing Debian on BlueField

Debian Hardware and Software Requirements

- Host system with [RShim driver installed](#)
- BlueField NIC inserted in a host PCIe slot

Debian with MLNX_OFED Installation

A pre-built BFB of Debian10 with MLNX_OFED_LINUX installed is available. Please contact NVIDIA Support to get this BFB.

To install Debian BFB, run the following on the host side:

```
$ cat <debian.bfb> > /dev/rshim0/boot  
Username: root  
Password: debian
```

The BFB image may also be installed using the ["bfb-install" utility script](#).


Upgrading NVIDIA BlueField DPU Software


Hardware Prerequisites

This quick start guide assumes that an NVIDIA® BlueField®-2 DPU has been installed in a server according to the instructions detailed in your [DPU's hardware user guide](#).

BlueField DPU Cable Connections

Connect the cables provided with your DPU package to the NC-SI and USB interfaces of the DPU. Please refer to the Supported Interfaces section of your [hardware user guide](#) for an illustration of the location of those interfaces.

-  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.

| USB RShim connectivity cable | NC-SI UART connectivity cable |
|------------------------------|--|
| |  |

Software Prerequisites

1. Please download the following packages from the [NVIDIA DOCA SDK page](#) from the downloader at the bottom of the page:
 - BFB package (e.g. Ubuntu 20.04 image for BlueField) under the BlueField Software tab
 - BlueField MLNX_OFED for the host under the BlueField Drivers tab

i Alternatively, you may run this command prior to the MLNX_OFED installation process (step 2):

```
wget http://www.mellanox.com/downloads/ofed/MLNX_OFED-<version>/MLNX_OFED_LINUX-<version>-<distribution>-<arch>.tgz
```

For example:

```
wget http://www.mellanox.com/downloads/ofed/MLNX_OFED-5.0-2.1.8.0/MLNX_OFED_LINUX-5.0-2.1.8.0-ubuntu18.04-x86_64.tgz
```

⚠ Please download and install the latest BlueField MLNX_OFED on your server for best user experience.

- If you choose not to install MLNX_OFED (step 2), please download the [suitable DEB/RPM](#) for RShim (management interface for BlueField from the host) driver
2. Install MLNX_OFED. Run:

```
sudo tar -xvf MLNX_OFED_LINUX-<version>-x86_64.tgz
sudo ./mlnxofedinstall --auto-add-kernel-support --ovs-dpdk
```

This script also installs the firmware version associated with the MLNX_OFED version. The script print-out will display the Current and Available firmware versions as shown in the following example:

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

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

The upgrade takes effect only after power cycle which is performed in later steps.

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

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

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

4. (Optional) Enable NVMe emulation. Run:

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

5. Skip this step if your BlueField DPU is Ethernet only. Please refer to [Supported Platforms and Interoperability](#) 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> s LINK_TYPE_P1=2 LINK_TYPE_P2=2
```

6. (Optional) If you choose not to install `MLNX_OFED`, you must install `RShim`.

- For Ubuntu/Debian, run:

```
sudo dpkg --force-all -i rshim-<version>.deb
```

- For RHEL/CentOS, run:

```
sudo rpm -Uvh rshim-<version>.rpm
```

7. Assign a static IP to `tmfifo_net0` (RShim host interface).

- For Ubuntu, edit the file `/etc/netplan/01-netcfg.yaml` by adding the following lines:

```
tmfifo_net0:
  addresses: [192.168.100.1/24]
  dhcp4: false
```

Example:

```
sudo cat /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enol:
      dhcp4: yes
    tmfifo_net0:
      addresses: [192.168.100.1/24]
      dhcp4: no
```

- For Debian:

- Create the file `/etc/network/interfaces.d/tmfifo`.
- Set the following lines:

```
auto tmfifo_net0
iface tmfifo_net0 inet static
address 192.168.100.1/24
```

- For RHEL/CentOS:
 - Create the file `/etc/sysconfig/network-scripts/ifcfg-tmfifo_net0`.
 - Set the following lines:

```
DEVICE=tmfifo_net0
BOOTPROTO=none
ONBOOT=yes
PREFIX=24
IPADDR=192.168.100.1
NM_CONTROLLED=no
```

- Power cycle your server.
- Verify that RShim is active.

```
sudo systemctl status rshim
```


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


```
sudo systemctl enable rshim
sudo systemctl start rshim
```

Image Installation

The following is an example of Ubuntu installation assuming the "pv" tool has been installed (to view the installation progress).

```
sudo bfb-install --rshim <rshimN> --bfb <image_path.bfb>
Pushing bfb
1.08GiB 0:00:57 [19.5MiB/s] [    <=>    ]
Collecting BlueField booting status. Press Ctrl+C to stop...
INFO[BL2]: start
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
INFO[MISC]: Ubuntu installation started
INFO[MISC]: Installation finished
INFO[MISC]: Rebooting...
```

 This installation sets up the OVS bridge.

 For a more detailed explanation, please refer to the [Installation and Initialization](#).

Post-installation Procedure

- Restart MLNX_OFED. Run:

```
sudo /etc/init.d/openibd restart
Unloading HCA driver:                [ OK ]
Loading HCA driver and Access Layer: [ OK ]
```

- Configure the physical function (PF) interfaces.

```
sudo ifconfig <interface-1> <network-1/mask> up
sudo ifconfig <interface-2> <network-2/mask> up
```

For example:

```
sudo ifconfig p2p1 192.168.200.32/24 up
sudo ifconfig p2p2 192.168.201.32/24 up
```

Pings between the source and destination should now be operational.

Windows Support

⚠ For more information on Windows support on BlueField DPU, please refer to [BlueField Windows Support](#).

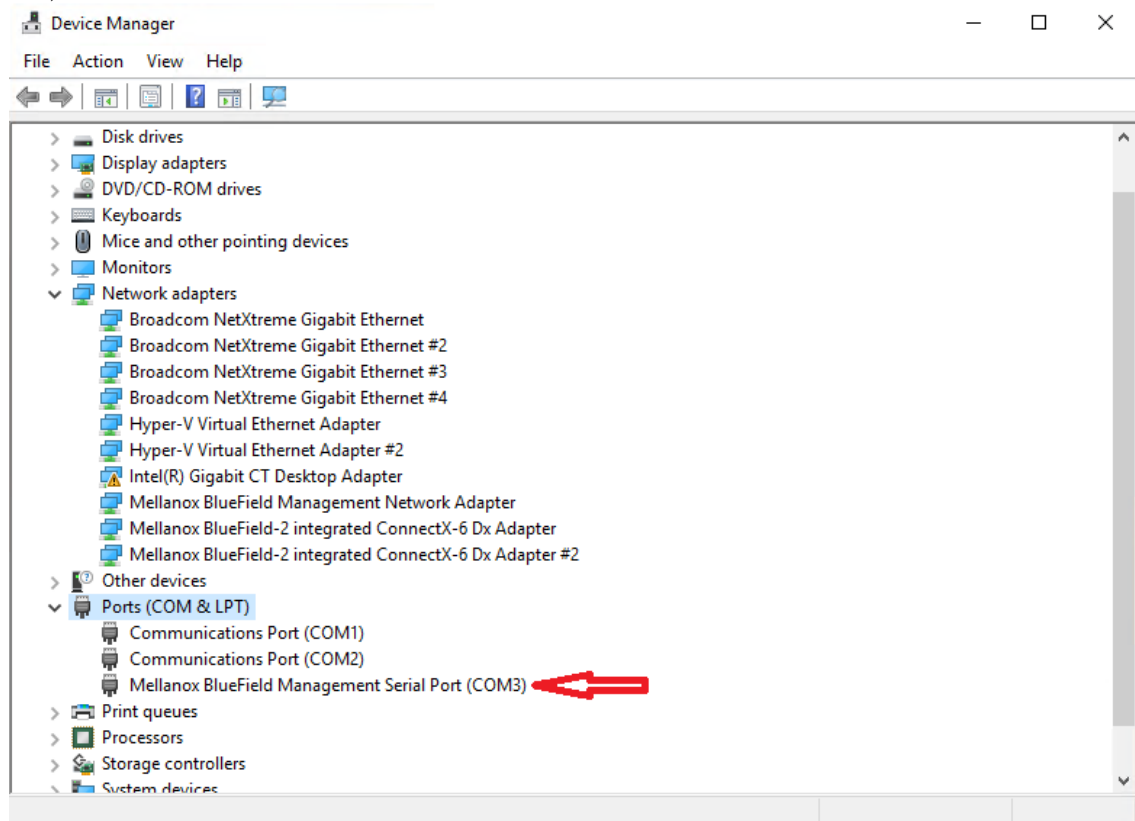
Accessing BlueField DPU From Host

The BlueField DPU can be accessed via PuTTY or any other network utility application to communicate via virtual COM or virtual Ethernet adapter. To use COM:

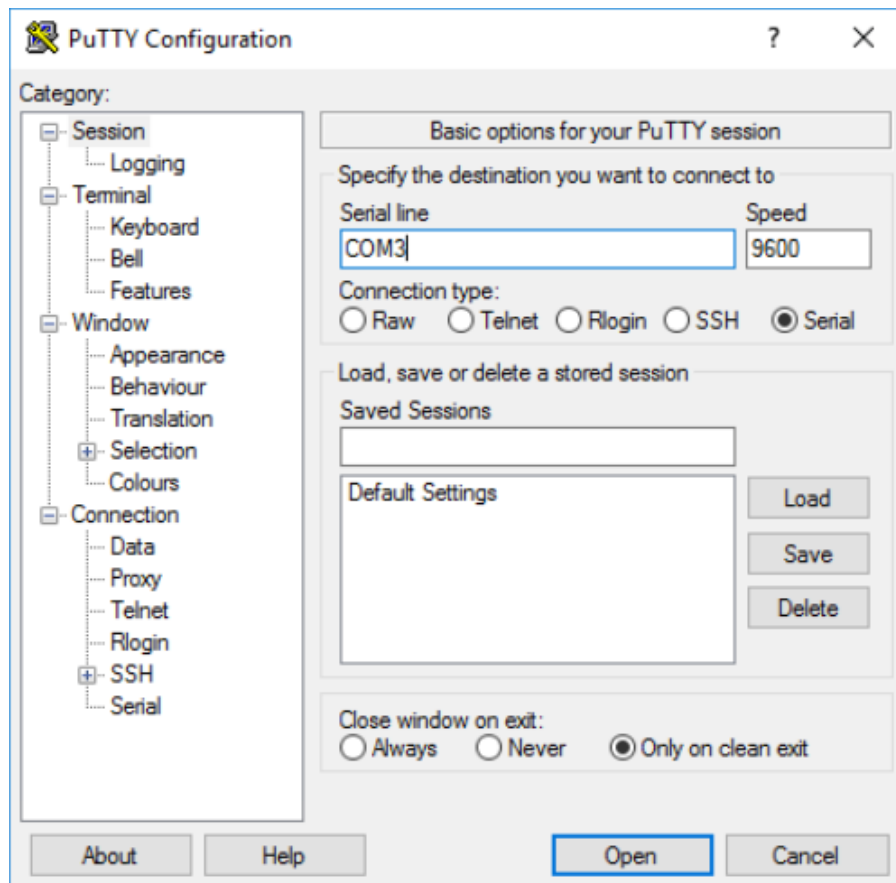
1. Open Putty.
2. Change connection type to Serial.
3. Run the following command in order to know what to set the "Serial line" field to:

```
C:\Users\username\Desktop> reg query HKLM\HARDWARE\DEVICEMAP\SERIALCOMM | findstr MlxRshim
\MlxRshim\COM3          REG-SZ          COM3
```

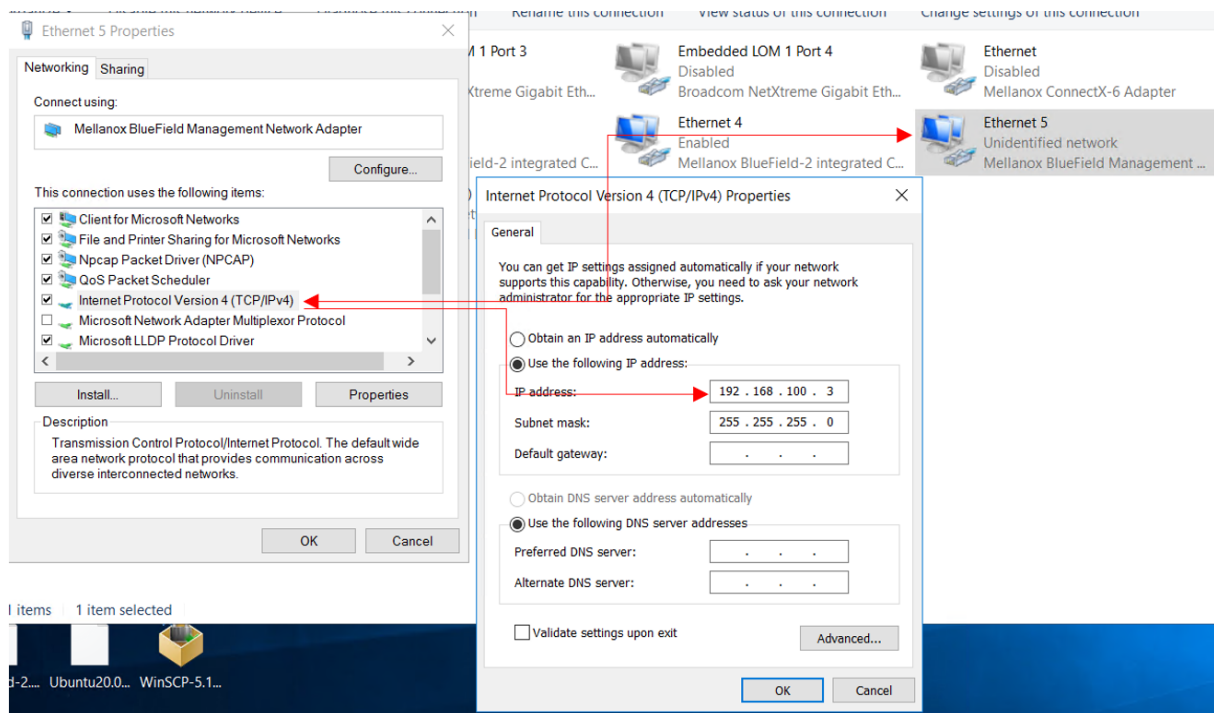
In this case use COM3. This name can also be found via Device Manager under "Ports (Com & LPT)".



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.



BlueField Management

- [Performance Monitoring Counters](#)
- [Intelligent Platform Management Interface \(IPMI\)](#)
- [RShim Logging](#)

Performance Monitoring Counters

The performance data of the BlueField® hardware is collected using two mechanisms:

- Programming hardware counters to monitor specific events
- Reading registers that hold performance/event statistics

The performance modules in BlueField are present in several hardware blocks and each block has a certain set of supported events. The hardware blocks that include a performance module are:

- Tile (block containing 2 cores and a shared L2 cache)
- TRIO (PCIe root complex)
- MSS (Memory Sub-system containing the Memory Controller and L3 cache)

The number of tiles depends on the system. There are a maximum of 8 Tiles, 3 TRIOs and 2 MSS blocks in BlueField.

The performance module in each Tile has 4 hardware counters that can be simultaneously programmed to monitor specific events. There are 2 performance modules in each TRIO and are referred to by the driver as “trio” and “triogen” performance modules. The TRIO performance module (trio) has a single counter while the generic module (triogen) has 4 counters.

The MSS blocks have a generic performance module (mss) with 4 hardware counters along with an “ecc” block that exposes a set of registers that hold ECC statistics.

The PCIe TLR statistics for each trio are under the “pcie” block.

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 “hwmonN” 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”.

For blocks that use hardware counters (mechanism 1) to collect this info, each counter present in the block is represented by “eventN” and “counter” `sysfs` files. An eventN and counterN pair can be used to program and monitor events.

There is also an “event_list” `sysfs` file that displays the list of events supported by that block along with the event numbers.

Programming Counter to Monitor Event

In order 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/hwmonN` folder corresponding to this driver as “BFPERF_DIR”.

For example, in order 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, just run:

```
$ cat <BFPERF_DIR>/tile2/counter3
```

In order 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”.

In order 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, in order 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.

List of Supported Events

TRIO Generic Performance Module

| 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 |

Tile Performance Module

| 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 |

| Value | Name | Description |
|-------|---------------|---|
| 0x48 | MAF_BUSY | Reserved for internal use |
| 0x49 | MAF_REQUESTS | Reserved for internal use |
| 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 |

| Value | Name | Description |
|-------|---------------------|--|
| 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 |
| 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 |

| Value | Name | Description |
|-------|----------|---------------------------|
| 0x84 | RXRSP_RN | Reserved for internal use |
| 0x85 | RXDAT_RN | Reserved for internal use |

TRIO Performance Module

| 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 |
| 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 |

MSS Generic Performance Module

| Value | Name | Description |
|-------|-----------|--------------------------------|
| 0xc0 | RXREQ_MSS | CHI requests sent into memory |
| 0xc1 | RXDAT_MSS | CHI data sent into memory |
| 0xc2 | TXRSP_MSS | CHI responses sent from memory |
| 0xc3 | TXDAT_MSS | CHI data sent from memory |

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.

| 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 |
| 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 |

| Value | Name | Description |
|-------|-------------------------|---------------------------------|
| 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 |
| 0x2b | ANY_REJECT_BANK1 | Reserved for internal use |

PCIe TLR Statistics

| 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 |

| Value | Name | Description |
|-------|--------------------------|-----------------------------|
| 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 |

SNMP Subagent

Simple Network Management Protocol is a UDP-based protocol which allows remote monitoring and configuration of certain system statistics and parameters. SNMP is used by agents and managers to send and retrieve information. An agent is a software process that responds to SNMP queries to provide status and statistics about a network node. A manager is an application that manages SNMP agents on a network by issuing requests, getting responses, and listening for and processing agent-issued traps.

System statistics and parameters are collectively known in SNMP terminology as variables. Each variable is uniquely identified by an object identifier (OID). A variable may be either scalar or columnar. A scalar variable is a singleton. A columnar variable is part of a two-dimensional collection of variables known as a table. In a table a row represents one record or instance of an object.

SNMP uses a collection of text files, called MIB (Management Information Base), to describe variables and traps that a server may offer and maps OIDs to human-readable names. All MIBs fit into a universal OID hierarchy. There is a point in the tree under which all vendor MIBs are rooted: .1.3.6.1.4.1 (.iso.org.dod.internet.private.enterprises). Each vendor should have an enterprise number registered with IANA (Internet Assigned Numbers Authority). For Mellanox, this number is 33049.

Performance Monitoring via SNMP Subagent

Depending on a counter's type, there are two types of tables to display the hardware counters:

1. 4-column (block ID, counter ID, event, and counter value) table. The number of rows for each table is determined according to the following simple equation: $\langle \text{num_of_blocks} \rangle * \langle \text{num_of_counters} \rangle$. For example, the MSS module has 2 blocks and 4 hardware counters, so the total amount of rows in table should be $2 * 4 = 8$ rows. In order to program a counter, the block ID, counter ID and event name or number should be set using SNMP SET commands, for example:

```
$ snmpset -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcMssBlockIdSet.0 u 0
$ snmpset -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcMssCounterIdSet.0 u 3
$ snmpset -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcMssEventIdSet.0 s 0xc0
```

Or using a single SNMP SET format:

```
$ snmpset -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcMssBlockIdSet.0 u 0 MLNX-PMC-MIB::mlnxPmcMssCounterIdSet.0 u 0 MLNX-PMC-MIB::mlnxPmcMssEventIdSet.0 s 0xc0
```

This command starts monitoring the event 0xc0:RXREQ_MSS 0xc0 in the counter /sys/class/hwmon/hwmon0/mss0/counter3.

To stop monitoring, set 0x0 into corresponding event, for example:

```
$ snmpset -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcMssBlockIdSet.0 u 0
$ snmpset -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcMssCounterIdSet.0 u 3
$ snmpset -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcMssEventIdSet.0 s 0x0
```

Here is an output example for the MSS module:

```
$ snmptable -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcMssTable
SNMP table: MLNX-PMC-MIB::mlnxPmcMssTable
mlnxPmcMssBlockId mlnxPmcMssCounterId mlnxPmcMssEvent mlnxPmcMssCounters
0 0 0xc3 9485515546
0 1 0xc2 3718241246
0 2 0xc1 3718118806
0 3 0xc0 6601319766
1 0 0xc0 5675145425
1 1 0xc1 3001880054
1 2 0xc2 3001805288
1 3 0xc3 8347484949
```

2. 3-column (block ID, event, counter value) table. The number of rows for each table is determined according to the following simple equation: $\text{<num_of_blocks> * <num_of_events>}$. For example, PCIe module has 3 blocks and 15 events, so the total amount of rows in the table should be $3 * 15 = 45$ rows.

Here is an output example for PCIe module:

```
$ snmptable -v2c -c mlnx 192.168.100.2 MLNX-PMC-MIB::mlnxPmcPcieTable
SNMP table: MLNX-PMC-MIB::mlnxPmcPcieTable
mlnxPmcPcieBlockId mlnxPmcPcieEvent mlnxPmcPcieCounters
0 IN_P_BYTE_CNT 0
0 OUT_C_BYTE_CNT 0
0 IN_NP_PKT_CNT 0
0 OUT_P_BYTE_CNT 125312
0 OUT_NP_BYTE_CNT 0
0 IN_C_PKT_CNT 17260
0 IN_NP_BYTE_CNT 0
0 IN_P_PKT_CNT 0
0 OUT_C_PKT_CNT 0
0 OUT_NP_PKT_CNT 17260
0 OUT_P_PKT_CNT 1958
0 IN_C_BYTE_CNT 0
1 IN_P_BYTE_CNT 0
1 OUT_C_BYTE_CNT 0
1 IN_NP_PKT_CNT 0
1 OUT_P_BYTE_CNT 0
1 OUT_NP_BYTE_CNT 0
1 IN_C_PKT_CNT 1
1 IN_NP_BYTE_CNT 0
1 IN_P_PKT_CNT 0
1 OUT_C_PKT_CNT 0
1 OUT_NP_PKT_CNT 1
1 OUT_P_PKT_CNT 0
1 IN_C_BYTE_CNT 0
2 IN_P_BYTE_CNT 95284696
2 OUT_C_BYTE_CNT 264050560
2 IN_NP_PKT_CNT 4026124
2 OUT_P_BYTE_CNT 116872
2 OUT_NP_BYTE_CNT 200243467
2 IN_C_BYTE_CNT 366859
2 IN_NP_BYTE_CNT 0
2 IN_P_PKT_CNT 1575547
2 OUT_C_PKT_CNT 4065099
2 OUT_NP_PKT_CNT 366859
2 OUT_P_PKT_CNT 28984
2 IN_C_BYTE_CNT 1362368
```

Intelligent Platform Management Interface (IPMI)

❗ For CentOS and Ubuntu, IPMI services are disabled by default. To enable them, run:

```
systemctl start set_emu_param  
systemctl start mlx_ipmid
```


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 I²C bus.

- ⚠ The BlueField ipmb_dev_int driver is registered at the 7-bit I²C address 0x30 by default. The I²C address of the BlueField can be changed in the file /usr/bin/set_emu_param.sh.
- Mellanox BlueField Controller cards provide connection from the host server BMC to BlueField Arm I²C bus.
- Mellanox BlueField SmartNICs provide connection from the host server BMC to the BlueField NC-SI port.
- Mellanox BlueField Reference Platforms provide connection from its on-board BMC to BlueField Arm I²C bus.

List of IPMI Supported Sensors

| Sensor | ID | Description |
|----------------|----|--|
| bluefield_temp | 0 | Support NIC monitoring of BlueField's temperature |
| ddr0_0_temp* | 1 | Support monitoring of DDR0 temp (on memory controller 0) |
| ddr0_1_temp* | 2 | Support monitoring of DDR1 temp (on memory controller 0) |
| ddr1_0_temp* | 3 | Support monitoring of DDR0 temp (on memory controller 1) |
| ddr1_1_temp* | 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 |


 *These sensors are not available, and hence are not populated, on BlueField SmartNICs.

 *On BlueField-2 based boards, DDR sensors and FRUs are not supported. They will appear as no reading.

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 | 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 |
| ddr0_0_spd* | 4 | FRU for SPD MC0 DIMM 0 (MC = memory controller) The ddr0_0_spd is in binary format |
| ddr0_1_spd* | 5 | FRU for SPD MC0 DIMM1 The ddr0_1_spd is in binary format |
| ddr1_0_spd* | 6 | FRU for SPD MC1 DIMM0 The ddr1_0_spd is in binary format |
| ddr1_1_spd* | 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) |

| FRU | ID | Description |
|-----------------|----|--|
| ip_addresses | 11 | <p>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:</p> <pre>BMC: XXX.XXX.XXX.XXX P0: XXX.XXX.XXX.XXX P1: XXX.XXX.XXX.XXX</pre> <p>The size of the written file should be exactly 61 bytes.</p> |
| dimms_ce_ue | 12 | <p>FRU reporting the number of correctable and uncorrectable errors in the DIMMs.</p> <p>This FRU is updated once every 3 seconds.</p> |
| 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 |
| eth_hw_counters | 16 | List of ConnectX interface hardware counters |

 *On BlueField-2 based boards, DDR sensors and FRUs are not supported. They will appear as no reading.

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 <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 I²C 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 |

| Command Description | IPMITool Command | Relevant IPMI 2.0 Rev 1.1 Spec Section |
|-------------------------|--|--|
| 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 |
| Set sensor threshold | <p>sensor thresh <sensor-id> <threshold> <setting></p> <ul style="list-style-type: none"> • sensor-id - name of the sensor for which a threshold is to be set • threshold - which threshold to set <ul style="list-style-type: none"> • ucr - upper critical • unc - upper non-critical • lnc - lower non-critical • lcr - lower critical • setting - the value to set the threshold to <p>To configure all lower thresholds, use: sensor thresh <sensor-id> lower <lnc> <lcr> <lnc></p> <div>  The lower non-recoverable <lnc> option is not supported </div> <p>To configure all upper thresholds, use: sensor thresh <sensor-id> upper <unc> <ucr> <unr></p> <div>  The upper non-recoverable <unr> option is not supported </div> | 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 |

| Command Description | IPMITool Command | Relevant IPMI 2.0 Rev 1.1 Spec Section |
|---------------------|-----------------------------------|--|
| 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 |

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

2. Install the required drivers.

Example of building and loading jc42.ko:

```
# Compile jc42.c
mkdir -p /root/drivers/hwmon/
wget -O /root/drivers/hwmon/jc42.c https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/plain/drivers/hwmon/jc42.c?h=linux-5.4.y
echo "obj-m := jc42.o" > /root/drivers/hwmon/Kbuild
make -C /usr/src/kernels/$(uname -r) M=/root/drivers/hwmon modules
ls -l /root/drivers/hwmon/jc42.ko
make -C /usr/src/kernels/$(uname -r) M=/root/drivers/hwmon modules_install
modprobe jc42
```

Example of building and loading ee1004.ko and at24.ko:

```
# Compile ee1004.c and at24.c
mkdir -p /root/drivers/misc/eeprom
wget -O /root/drivers/misc/eeprom/ee1004.c https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/plain/drivers/misc/eeprom/ee1004.c?h=linux-5.4.y
wget -O /root/drivers/misc/eeprom/at24.c https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/plain/drivers/misc/eeprom/at24.c?h=linux-5.4.y
echo "obj-m := ee1004.o at24.o" > /root/drivers/misc/eeprom/Kbuild
make -C /usr/src/kernels/$(uname -r) M=/root/drivers/misc/eeprom modules
ls -l /root/drivers/misc/eeprom/ee1004.ko
ls -l /root/drivers/misc/eeprom/at24.ko
make -C /usr/src/kernels/$(uname -r) M=/root/drivers/misc/eeprom modules_install
modprobe ee1004
modprobe at24
```

The EEPROM and i2c-dev modules are included in CentOS 7.6.

```
# Install the i2c-dev and eeprom drivers
modprobe i2c-dev
modprobe eeprom
```

3. 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
```

4. 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
```


5. 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 -sf /usr/lib/libi2c.so.0.1.1 /lib64/libi2c.so
ln -sf /usr/lib/libi2c.so.0.1.1 /lib64/libi2c.so.0
```

6. 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
```

7. 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
```

8. Generate a binary RPM from the ipmb-host source RPM and install it. Run:

```
$ rpmbuild --rebuild ipmb-host-1.0-0.g304ea0c.src.rpm
```

9. 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
```

10. Load the IPMB driver. Run:

```
$ modprobe ipmb-dev-int
```

11. Start the IPMI daemon. Run:

```
$ systemctl enable set_emu_param  
$ systemctl start set_emu_param  
$ systemctl enable mlx_ipmid  
$ systemctl start mlx_ipmid
```

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 mc info
```

14. Test that the BlueField can send requests to the BMC:

```
ipmitool mc info
```

Retrieving Data from BlueField Via OOB/ConnectX Interfaces

It is possible to for the external host to retrieve IPMI data via the OOB interface (for BlueField-2 only) 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"
```

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.

BlueField and BMC I²C Addresses on BlueField Reference Platform

BlueField in Responder Mode

| Device | I ² C Address |
|------------------------|--------------------------|
| BlueField ipmb_dev_int | 0x30 |
| BMC ipmb_host | 0x20 |

BlueField in Requester Mode

| Device | I ² C Address |
|---------------------|--------------------------|
| BlueField ipmb_host | 0x11 |
| BMC ipmb_dev_int | 0x10 |

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 | <p>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.</p> <p>To change the BMC I²C address: IPMB_HOST_CLIENTADDR=<BMC I²C Address> <I²C address> must be equal to: 0x1000+<7-bit I²C address></p> |

RShim Logging

RShim Logging uses an internal 1KB HW buffer to track booting progress and record important messages. It is written by the BlueFieldArm 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 the DISPLAY_LEVEL to 2.

```
# echo "DISPLAY_LEVEL 2" > /dev/rshim0/misc
```

3. Log messages are displayed in the misc file. Here is an example output:

```
# 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
```

4. Log messages are displayed in the misc file. Here is an example output:

```
# 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
```

The following table details the ATF/UEFI messages added in this release.

| 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 |
| 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 Mellanox 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 Mellanox 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 Mellanox 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 Mellanox FAE. |

| Message | Explanation | Action |
|---|---|---|
| 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 Mellanox 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 |
| 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 |
| 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 | Report to FAE. Usually the system is able to continue running. |

| 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 |
| 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 |

| Message | Explanation | Action |
|---|---|--|
| 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 Mellanox 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 Mellanox 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 Mellanox 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 Mellanox 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 Mellanox 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 |
| 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 |
| 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 |

| Message | Explanation | Action |
|-----------------------------|--------------------------------|--|
| ASSERT(UEFI): xxx : line-no | Runtime assert message in UEFI | Report to FAE. 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 |

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.

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 |
| 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 |

| Byte(s) | Field | Description |
|---------|--------------|--|
| 14 | Event Data 1 | <p>[7:6] - Type of data in Event Data 2</p> <ul style="list-style-type: none"> • 0b00 = unspecified • 0b10 = OEM code • 0b11 = Standard sensor-specific event extension <p>[5:4] - Type of data in Event Data 3</p> <ul style="list-style-type: none"> • 0b00 = unspecified • 0b10 = OEM code • 0b11 = Standard sensor-specific event extension <p>[3:0] - Event Offset; offers more detailed event categories.</p> <p>See <i>IPMI 2.0 Specification</i> section 29.7 for more detail.</p> |
| 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.

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.

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: <ul style="list-style-type: none"> • 0x06 - Unrecoverable EMMC error. Contact Mellanox support. |
| | | 0x02 | System firmware progress: Informational message, no actions needed. Event Data 2: <ul style="list-style-type: none"> • 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. |

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 the BlueField DPU 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


```

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.

- [DPU Bring-Up and Driver Installation](#)
- [Functional Diagram](#)
- [Modes of Operation](#)
- [Kernel Representors Model](#)
- [Multi-Host](#)
- [Virtual Switch on BlueField DPU](#)
- [Configuring Uplink MTU](#)
- [BlueField Link Aggregation](#)
- [Mediated Devices](#)
- [RDMA Stack Support on Host and Arm System](#)
- [Controlling Host PF and VF Parameters](#)
- [DPDK on BlueField SmartNIC](#)
- [NVMe SNAP on BlueField SmartNIC](#)
- [RegEx Acceleration](#)
- [Public Key Acceleration](#)
- [IPsec Functionality](#)
- [QoS Configuration](#)
- [VirtIO-net Emulated Devices](#)
- [Deep Packet Inspection](#)

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:


- [Installing Linux on DPU](#)
- [MLNX_OFED Installation](#)
- [eMMC Backup and Restore](#)
- [Local Yum Repository Setup and Usage](#)

Installing Linux on DPU

This page demonstrates CentOS 7.4 installation on the DPU. Other operating systems work similarly with the PXE boot installation process.

Software Requirements

| Requirement | Description |
|-------------|-------------|
|-------------|-------------|

| | |
|------------------------------------|---|
| CentOS 7.4 Linux OS ¹ | <p>To get CentOS 7.4 image, run:</p> <pre>\$ wget http://archive.kernel.org/centos-vault/altarch/7.4.1708/iso8/aarch64/CentOS-7-aarch64-Everything.iso</pre> <div>  Some required drivers do not compile and load if running CentOS 5.x or earlier. </div> |
| Access to the latest DPU SW bundle | <p>Mellanox uses box.com to distribute BlueField® software. Contact your sales/support representative for a custom link to download BlueField software releases.</p> |
| Tarball | <p>In this document, we assume the tarball BlueField-<version>.tar.gz is extracted at /root, to do this, run the following command:</p> <pre>\$ tar -xvf BlueField-<version>.tar.xz -C /root</pre> |

Flashing DPU Bootloader Code

Push the initial install bootstream to the DPU:

Before installing an OS, flash the bootloader code first. The DPU is shipped with an obsolete bootloader code, and should be according to the following instructions.

Opening Terminal Connection to DPU

To open a console window to the DPU, a terminal application is required. The application “minicom” is used in this flow, however, any standard terminal application can work (e.g. “screen”).

 Install minicom by running “yum install minicom” or “apt-get install minicom”.

1. On the host, type “minicom” to open minicom on the current terminal, use “minicom -s” to set it up.
2. Go to the settings menu by pressing “Ctrl-a + o” (the setting menu opens by default when launching with the “-s” option). Navigate to the “Serial port setup” submenu and set the “Serial Device” to the one connected (should be one of the /dev/ttyUSBx if using the serial-UART cable).
3. Change the baud rate to 115200 8N1, and ensure that the hardware and software flow control are set to “No”.

```

+-----+
| 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?              |
+-----+
| Screen and keyboard |
| Save setup as dfl   |
| Save setup as..    |
| Exit                |
+-----+

```

4. Select “Save setup as dfl” in order not to have to set it again in the future.

Using Initial Install Bootstream


1. On the host side, ensure that the RShim driver is running:

```
$ systemctl status rshim
```

An RShim device is located under the /dev directory, if you only have one, it should be “rshim0”:

```
$ [root@bu-lab02 ~]# ls /dev/rshim0/
boot      console  net      rshim
```

You can boot a DPU by pushing a bootstream to it, which is done by writing a bootstream file to the /dev/rshimX/boot device. (See step 2 below.)

 The /dev/rshimX/console device can be used as a console instead of the serial-USB console. The primary bootloader does not support this device, however, UEFI and Linux support it. In cases where the special UART adapter board is unavailable, this can be used instead.

2. Push the initial install bootstream to the DPU:

```
$ cat /root/BlueField-<version>/sample/install.bfb > /dev/rshim0/boot
```

On the terminal, various boot messages appear until Linux is loaded. This is the Yocto embedded Linux running off the kernel initramfs pushed in the bootstream.

3. At login prompt, login as root without password.

```

done.
Starting OpenBSD Secure Shell server: sshd
done.
Starting rpcbind daemon...done.
starting statd: done
exportfs: can't open /etc/exports for reading
NFS daemon support not enabled in kernel
Starting syslogd/klogd: done
Poky (Yocto Project Reference Distro) 2.3.1 bluefield /dev/ttyAMA0

bluefield login: root
root@bluefield:~#
CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.5.2 | VT102 | Offline


```

4. After Linux is loaded, in the terminal, run the /opt/mlnx/scripts/bfrec script to update the bootloader.

Preparing the Host-Side Environment

 Some required drivers do not compile and load if running CentOS 5.x or earlier.

Before installing the preferred OS on the BlueField DPU, the host must be set up for it to be capable of provisioning the DPU. Use the RShim PCIe driver through the server host to push the initial bootloader and supply the OS image for PXE boot through the PCIe connection.

 This process only needs to be done on the host machine which is provisioning the DPU, it is not required on the end machine.

Setup Procedure With Installation Script

If the host is running CentOS 7 (or equivalent) on the host, you may run a script to complete all the steps detailed in the [next section](#).

```
$ /root/BlueField-<version>/distro/rhel/pxeboot/setup.sh \  
-d /root/BlueField-<version>/ \  
-i /root/CentOS-7-aarch64-Everything.iso \  
-o /root/dd-rhel7.4-mlnx-ofed-4.2-X.X.X.X-aarch64.iso \  
-c ttyAMA0 \  
-k
```

Where:

- -d points to where the tar file has been extracted from. The script uses this directory to find all the source code it needs.
- -i points to the OS installation disk. This is the image that is accessed via PXE boot to install the OS on the DPU.
- -o points to the Mellanox OFED driver disk for Arm. Download and extract it from the [Mellanox FlexBoot](#) page.
- -c specifies the default UART port for the OS to use since the BlueField DPU has two Arm UARTs. For the DPU, “ttyAMA0” is used, which is UART0.
- -t, if specified and given the argument of what platform is set (DPU in this case), generates a “nonpxe.bfb” file containing the install kernel and rootfs. If this file is pushed to the RShim boot device, it automatically runs the installation process and skips the initial UEFI PXE boot operations.
- -k (optional) kickstarts auto-installation based on a default kickstart file which is installed as /var/pxe/ks/ks.cfg.

Note that there should be no firewall blocking the IP communication between the DPU and the server host machine. If a firewall exists, disable it with the following commands:

```
iptables -F  
iptables -t  
nat -F
```

Setup Procedure Without Installation Script

If the host is running CentOS 7 or equivalent, please refer to the [previous section](#) which describes a simpler way to perform the installation using an installation script.

The following sections demonstrate CentOS 7 installation, however, installation in other environments should be relatively similar.

Step 1: Set up the RShim Interface

The RShim driver communicates with the RShim device on the BlueField DPU. The RShim is in charge of many miscellaneous functions of the DPU, including resetting the Arm cores, providing the initial bootstream, and using the TMFIFO and the RShim network, to exchange network and console data with the host.

The RShim can be reached by the host via the USB connector and the PCIe slot. It is preferable, however, to use the PCIe connection.

⚠ To enable access to the RShim via the PCIe slot, a link must be open through firmware. This is done by adding “multi_function.rshim_pf_en = 0x1” to the [fw_boot_config] section in the firmware’s ini file.

Step 2: Install RShim Drivers

To install the kernel modules, please follow the instruction in section [RShim Host Driver](#).

Step 3: Configure RShim Net Interface

To use the RShim net interface, create an interface config file.

To create the RShim interface config file, run:

```
$ cat >/etc/sysconfig/network-scripts/ifcfg-tmfifo_net0 <<EOF
DEVICE=tmfifo_net0
BOOTPROTO=none
ONBOOT=yes
PREFIX=24
IPADDR=192.168.100.1
EOF
```

Step 4: Configure the TFTP Server

The host should be configured to act as a TFTP server to the DPU via the PCIe RShim network. This server provides the required files by the DPU to perform the PXE boot for installing the preferred OS.

⚠ Configuring the TFTP server requires a TFTP package. If it is not installed, install it via “yum install tftp” or “apt-get tftp”, depending on your Linux distribution. On some versions, the TFTP package cannot be found. In that case, install “xinetd”.

1. Extract the OS image and copy the required PXE boot components:

```
$ mount -t iso9660 -o loop CentOS-7-aarch64-Everything.iso /mnt
$ mkdir -p /var/lib/tftpboot/centos/7.4
$ cp /mnt/EFI/BOOT/BOOTAA64.EFI /var/lib/tftpboot/
$ cp /mnt/EFI/BOOT/grubaa64.efi /var/lib/tftpboot/
$ cp /mnt/images/pxeboot/vmlinuz /var/lib/tftpboot/centos/7.4
$ cp /mnt/images/pxeboot/initrd.img /var/lib/tftpboot/centos/7.4/initrd-orig.img
```

2. Patch the initrd with the eMMC driver and TMFIFO (RShim network) driver:

```
$ mkdir -p /tmp/.bfcentos
$ mkdir -p /tmp/.bfinstdd
$ cd /tmp/.bfcentos
$ xzcat /var/lib/tftpboot/centos/7.4/initrd-orig.img | cpio -idm
$ mount /root/BlueField-<version>/distro/rhel/bluefield_dd/bluefield_dd-4.11.0-22.el7a.aarch64.iso /tmp/.bfinstdd
$ mkdir -p usr/lib/modules/4.11.0-22.el7a.aarch64/updates
$ cp /tmp/.bfinstdd/lib/modules/4.11.0-22.el7a.aarch64/updates/dw_mmc*.ko usr/lib/modules/4.11.0-22.el7a.aarch64/updates/
$ cp /tmp/.bfinstdd/lib/modules/4.11.0-22.el7a.aarch64/updates/tmfifo.ko usr/lib/modules/4.11.0-22.el7a.aarch64/updates/
$ cp /root/BlueField-<version>/distro/rhel/bluefield_dd/bluefield_dd-4.11.0-22.el7a.aarch64.iso ./bluefield_dd.iso
$ umount /tmp/.bfinstdd; rmdir /tmp/.bfinstdd
$ chown root:root * -R
$ depmod -b /tmp/.bfcentos 4.11.0-22.el7a.aarch64
$ find . | cpio -oc | xz --check=crc32 --lzma2=dict=32MiB > /var/lib/tftpboot/centos/7.4/initrd.img
```

⚠ These commands assume that you are using kernel version “4.11.0-22.el7a.aarch64”. If you are using a different version, utilize the corresponding bluefield_dd.iso. If none is found, compile one by running the following:

```
$ source /path/to/SDK/environment-setup-aarch64-poky-linux; unset LDFLAGS; ./build-dd.sh /path/to/kernel-devel-4.11.0-44.el7a.aarch64.rpm
```

3. Change the grub configuration to PXE boot over the right location:

```
$ cat >/var/lib/tftpboot/grub.cfg <<EOF
menuentry 'Install centos/7.4 AArch64 - BlueField' --class red --class gnu-linux
--class gnu --class os {
    linux (tftp)/centos/7.4/vmlinuz ro ip= method=http://192.168.100.1/centos7 inst.dd=/bluefield_dd.iso
    console=hvc0
    initrd (tftp)/centos/7.4/initrd.img
}
EOF
```

4. Start the TFTP server:

```
$ systemctl restart tftp
```

⚠ Based on the system, the user may need to use “system TFTP restart” instead. Also, if required, the user might need to switch use “xinetd” instead of “TFTP”.

Step 5: Set Up the Server

⚠ Ensure that the DHCP server is only servicing the RShim network and not any external network. If not configured properly, the additional DHCP server may throttle the external network traffic.

DHCP server set up on the host is required for DPU to get a private IP from the host for PXE boot process completion. Configure the correct server names and domain names so that the DPU can connect to the network via the host later on.

1. Get the server/domain names on the host:

```
bash-4.2$ cat /etc/resolv.conf
# Generated by NetworkManager
search example.domain.com example2.com
nameserver 192.168.1.2
nameserver 192.168.1.3
```

This example shows that the domains are example.domain.com and example2.com, and the servers names are 192.168.1.2 and 192.168.1.3.

2. Set up the DHCP config file accordingly:


```
cat >/etc/dhcp/dhcpd.conf <<EOF
allow booting;
allow bootp;

subnet 192.168.100.0 netmask 255.255.255.0 {
    range 192.168.100.10 192.168.100.20;
    option broadcast-address 192.168.100.255;
    option routers 192.168.100.1;
    option domain-name-servers 192.168.1.2 192.168.1.3;
    # Set the domain search according to the network configuration
    option domain-search "example.domain.com" "example2.com";
    next-server 192.168.100.1;
    filename "/BOOTAA64.EFI";
}
# Specify the IP address for this client.
host pxe_client {
    hardware ethernet 00:1a:ca:ff:ff:01;
    fixed-address 192.168.100.2;
}
EOF
```

 It is recommended to back up the previous dhcpd.conf file before overwriting it.

Step 6: Set Up the HTTP Server


The TFTP server allows the PXE boot to load the initrd and kernel. The DPU obtains all the other required sources through the network, thus, making it necessary to set up an HTTP.

 Setting up the HTTP server requires the HTTP package. If it is not installed, please install it via “yum install httpd” or “apt-get httpd”, depending on your Linux distribution.

To configure the http server to serve the contents of the installation disk, run the following:

```
$ cat >/etc/httpd/conf.d/pxeboot.conf <<EOF
Alias /centos7 /mnt
<Directory /mnt>
    Options Indexes FollowSymLinks
    Require ip 127.0.0.1 192.168.100.0/24
</Directory>
EOF
$ systemctl enable httpd
$ systemctl restart httpd
```

Installing CentOS 7 on BlueField DPU

 If the error “no root is found” appears in the installation process, check or disable the firewall as needed on the server host machine.

Full PXE Boot Installation

1. Get to the UEFI boot menu.
 - a. Reboot the DPU by typing “reboot” on the console. A “UEFI firmware...” message should appear and the screen clears.
 - b. Press ESC several times until you enter the UEFI boot menu.

| | | |
|--------------------------|----------------------|----------------------|
| Continue | | This selection will |
| Select Language | <Standard English> | direct the system to |
| Boot Manager | | continue to booting |
| Boot Maintenance Manager | | process |
| | | |
| =Move Highlight | <Enter>=Select Entry | |

2. On the host, restart the DHCP and TFTP service:

```
$ systemctl restart dhcpd
$ systemctl restart tftp #might be xinetd
```

3. Navigate to the Boot Manager.

```

*****
*                                     *
*                               Boot Manager                                *
*                               *****                                   *
*                                                                           *
* Device Path :                                                            *
* VenHw(F019E406-8C9C-11                                                *
* E5-8797-001ACA00BFC4)/                                              *
* Image                                                                    *
*                                                                           *
* Boot Option Menu                                                         *
*                                                                           *
* Linux from rshim                                                        *
* Linux from mmc0                                                         *
* CentOS 7.4                                                              *
* EFI Misc Device                                                         *
* EFI Network                                                             *
* EFI Internal Shell                                                       *
*                                                                           *
* and to change option, ENTER to select an                              *
* option, ESC to exit                                                      *
*                                                                           *
*****
*                               *
* =Move Highlight      <Enter>=Select Entry      Esc=Exit            *
*****

```

4. Select **EFI Network**, it will then use the **TFTP** service on the host to discover all available **PXE** boot options. Shortly after, a “**..Fetching Netboot Image**” message will appear enabling **CentOS** installation.

```

Install centos/7.4 AArch64 - BlueField

Use the  and  keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.


```

Use the `↑` and `↓` keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.

5. Select CentOS download.

⚠ This process may take few minutes as it fetches data over the RShim/virtual network. Running “ifconfig” on the host and monitoring the RX/TX packets on the “tmfif0_net0” network indicates that the fetching data process is not complete.

6. Follow the installation instructions in the configuration menu. Recommended settings are included.

 These configuration inputs are not needed when the kickstart option “-k” is specified when running the setup.sh script.

```

=====
VNC

Text mode provides a limited set of installation options. It does not offer
custom partitioning for full control over the disk layout. Would you like to use
VNC mode instead?

1) Start VNC
2) Use text mode

Please make your choice from above ['q' to quit | 'c' to continue |
'r' to refresh]: 2
=====
Installation:main* 2:shell 3:log 4:storage-lo> Switch tab: Alt+Tab | Help: F1
=====
1) [x] Language settings                2) [!] Time settings
    (English (United States))           (Timezone is not set.)
3) [!] Installation source              4) [!] Software selection
    (Processing...)                     (Processing...)
5) [!] Installation Destination          6) [x] Kdump
    (No disks selected)                 (Kdump is enabled)
7) [x] Network configuration             8) [!] Root password
    (Wired (eth0) connected)            (Password is not set.)
9) [!] User creation
    (No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin
installation | 'r' to refresh]: 2
=====
Time settings

Timezone: not set

NTP servers: not configured

1) Set timezone
2) Configure NTP servers
Please make your choice from above ['q' to quit | 'c' to continue |
'r' to refresh]: 1
=====
Timezone settings

Available regions
1) Europe                6) Pacific                10) Arctic
2) Asia                  7) Australia              11) US
3) America               8) Atlantic               12) Etc
4) Africa                9) Indian
5) Antarctica
Please select the timezone.
Use numbers or type names directly [b to region list, q to quit]: 11
=====
Timezone settings

Available timezones in region US
1) Alaska                4) Eastern                6) Mountain
2) Arizona               5) Hawaii                 7) Pacific
3) Central
Please select the timezone.
Use numbers or type names directly [b to region list, q to quit]: 4
=====
Installation

1) [x] Language settings                2) [x] Time settings
    (English (United States))           (US/Eastern timezone)
3) [x] Installation source              4) [x] Software selection
    (http://192.168.100.1/centos7)       (Minimal Install)
5) [!] Installation Destination          6) [x] Kdump
    (No disks selected)                 (Kdump is enabled)
7) [x] Network configuration             8) [!] Root password
    (Wired (eth0) connected)            (Password is not set.)
9) [!] User creation
    (No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin
installation | 'r' to refresh]: 4
=====
Base environment
Software selection

Base environment

1) [x] Minimal Install                6) [ ] Server with GUI
2) [ ] Compute Node                  7) [ ] GNOME Desktop
3) [ ] Infrastructure Server           8) [ ] KDE Plasma Workspaces
4) [ ] File and Print Server           9) [ ] Development and Creative
5) [ ] Basic Web Server                Workstation
Please make your choice from above ['q' to quit | 'c' to continue |
'r' to refresh]: 9
=====
Base environment
Software selection

1) [ ] Minimal Install                6) [ ] Server with GUI
2) [ ] Compute Node                  7) [ ] GNOME Desktop
3) [ ] Infrastructure Server           8) [ ] KDE Plasma Workspaces
4) [ ] File and Print Server           9) [x] Development and Creative

```

```

5) [ ] Basic Web Server                                     Workstation
Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c
=====
Installation

1) [x] Language settings                                2) [x] Time settings
   (English (United States))                          (US/Eastern timezone)
3) [!] Installation source                              4) [!] Software selection
   (Processing...)                                     (Processing...)
5) [!] Installation Destination                        6) [x] Kdump
   (No disks selected)                                (Kdump is enabled)
7) [x] Network configuration                          8) [!] Root password
   (Wired (eth0) connected)                          (Password is not set.)
9) [!] User creation
   (No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin installation | 'r' to refresh]: 5
=====
Probing storage...
Installation Destination

[x] 1) : 13.75 GiB (mmcb1k0)

1 disk selected; 13.75 GiB capacity; 1007.5 KiB free ...

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c
=====
Autopartitioning Options

[ ] 1) Replace Existing Linux system(s)

[x] 2) Use All Space

[ ] 3) Use Free Space

Installation requires partitioning of your hard drive. Select what space to use for the install target.

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c
=====
Partition Scheme Options

[ ] 1) Standard Partition

[ ] 2) Btrfs

[x] 3) LVM

[ ] 4) LVM Thin Provisioning

Select a partition scheme configuration.

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: 1
=====
Partition Scheme Options

[x] 1) Standard Partition

[ ] 2) Btrfs

[ ] 3) LVM

[ ] 4) LVM Thin Provisioning

Select a partition scheme configuration.

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c
Generating updated storage configuration
Checking storage configuration...
=====
Installation

1) [x] Language settings                                2) [x] Time settings
   (English (United States))                          (US/Eastern timezone)
3) [x] Installation source                              4) [x] Software selection
   (http://192.168.100.1/centos7)                     (Development and Creative
                                                    Workstation)
5) [x] Installation Destination                        6) [x] Kdump
   (Automatic partitioning                            (Kdump is enabled)
   selected)                                           (Kdump is enabled)
7) [x] Network configuration                          8) [!] Root password
   (Wired (eth0) connected)                          (Password is not set.)
9) [!] User creation
   (No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin installation |
'r' to refresh]: 8
=====
Please select new root password. You will have to type it twice.

Password:
Password (confirm):
=====
Question

The password you have provided is weak: The password fails the dictionary check
- it is based on a dictionary word.
Would you like to use it anyway?

```

```

Please respond 'yes' or 'no': yes
=====
Installation

1) [x] Language settings                2) [x] Time settings
   (English (United States))           (US/Eastern timezone)
3) [x] Installation source              4) [x] Software selection
   (http://192.168.100.1/centos7)       (Development and Creative
5) [x] Installation Destination         Workstation)
   (Automatic partitioning              6) [x] Kdump
   selected)                           (Kdump is enabled)
7) [x] Network configuration            8) [x] Root password
   (Wired (eth0) connected)             (Password is set.)
9) [ ] User creation
   (No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin
installation | 'r' to refresh]: b

```

7. Enter “b” and press “Enter” to initiate the installation process.
8. Press “Enter” to reboot into CentOS.

```

Installing iwl6000-firmware (1347/1348)
Installing words (1348/1348)
Performing post-installation setup tasks
Installing boot loader
.
Performing post-installation setup tasks
.
Configuring installed system
.
Writing network configuration
.
Creating users
.
Configuring addons
.
Generating initramfs
.
Running post-installation scripts
.
    Use of this product is subject to the license agreement found at /usr/sh
    Installation complete. Press return to quit
1: main* 2:shell 3:log 4:storage-lo> Switch tab: Alt+Tab | Help: F1

```

Non-PXE Boot Installation

When the setup script is run with the “-t” option, it generates a nonpxe.bfb file at the directory where the script is run. The directory contains the install kernel and rootfs which are usually loaded by UEFI during the initial PXE boot stage. Thus, if pushing this file, the host TFTP server no longer needs to be used and UEFI would automatically load the install kernel and rootfs from the boot FIFO. Together with the “-k” kickstart option, the host can be configured to initiate non-PXE boot and automatic CentOS installation, as long as the host HTTP and DHCP servers are working. To kick off the installation process, run the following command on the host:

```
$ cat nonpxe.bfb > /dev/rshim0/boot; sleep 2; systemctl restart dhcpd
```

MLNX_OFED Installation

Installing MLNX_OFED on Arm Cores

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.

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
```

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>
```

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
```

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 [Installing MLNX_OFED on the DPU](#).

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.

eMMC Backup and Restore

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.

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-lad6-4e3a-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 : 0x00000000000459b9c
```

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 backup.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
```

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
```

Local Yum Repository Setup and Usage

In many cases, bring up environments do not have access to external networks, and thus "yum install" cannot access its default repositories to download the packages. Also, manually installing RPMs is also exhaustive due to having to resolve all the dependency packages manually, which can lead to loads of extra RPMs being downloaded. To address this, a local yum repository can be set up, so that "yum install" can still be used even on machines with no external network access, leveraging its ability to automatically resolve all the dependencies.

Setting Up a Yum Repository

A yum repository contains a number of RPMs and a "repoinfo" directory which the "createrepo" command has generated to store the metadata of the present RPMs. Follow the below instructions to generate the required metadata (create the "repoinfo" directory and files).

1. Copy all the needed RPMs to a single directory. For example:


```
$ mkdir -p /root/localrepo  
$ cp *.rpm /root/localrepo
```

2. Run the createrepo command to make the directory a repository:

```
$ createrepo /root/localrepo
```

Once completed, a "repoinfo" directory is created that can be used as a repository. However, the createrepo package itself is not on the CentOS default installation. Run the following command to enable its usage:

```
$ yum install -y createrepo yum-utils
```

 This command needs to be run before it can be used, which defeats the purpose of using it to create the repository. Rather than building a repository from scratch, the optimal way is to use an already built repository. The best available repository is the CentOS installation disk. So make sure you have the image called "everything" and not "netinstall", "minimal" or "dvd".

3. Mount the CentOS-7-<arch>-everything.iso, apart from other directories. The "repoinfo" directory makes it a yum repository and a "Package" directory which includes all the RPMs it contains:

```
[root@bu-lab02 ~]# mount /root/CentOS-7-x86_64-Everything-1708.iso /mnt/x86/  
[root@bu-lab02 ~]# ls /mnt/x86/  
CentOS_BuildTag  EULA images LiveOS repodata RPM-GPG-KEY-CentOS-Testing-7  
EFI GPL isolinux Packages RPM-GPG-KEY-CentOS-7 TRANS.TBL  
[root@bu-lab02 ~]# mount /root/CentOS-7-aarch64-Everything.iso /mnt/aarch64/  
[root@bu-lab02 ~]# ls /mnt/aarch64/  
boot.catalog EULA images Packages RPM-GPG-KEY-CentOS-7 TRANS.TBL  
EFI GPL LiveOS repodata RPM-GPG-KEY-CentOS-7-aarch64
```

4. Once completed, the mount point is ready to act as a repository.

Yum Repository Usage

To use the created repository and not the default one, update the locations in which CentOS looks for yum repositories. This data is stored at `/etc/yum.repo.d/`.

1. Remove the existing repository data to avoid access failure when a network connection is unavailable:

```
$ mkdir /etc/backup
$ mv /etc/yum.repo.d/* /etc/backup
```

2. Create a file where the "baseurl" variable points to the repository to use. Turn off "gpgcheck" here for simplicity:

```
$ mkdir /etc/backup
$ mv /etc/yum.repo.d/* /etc/backup
$ cat > /etc/yum.repos.d/myyum.repo<<end
[myyum]
name=myyumsource
baseurl=file:///mnt/x86/
gpgcheck=0
end
```

3. Flush the cache information so that the system can pick up the new repo data:

```
$ yum clean all
```

4. When completed, the system should be able to see the yum repository:

```
$ yum repolist
```

5. The command "yum install" should work from this point, as long as the package is not a third party package which is not included in the CentOS base repository.

Arm CentOS Using Repository from Connected x86 Host

The SmartNIC eMMC size is 16GB, and the CentOS-7-aarch64-Everything.iso image is 7GB. Therefore, it is impractical to scp the image to the eMMC and mount it there. To address this, the aarch64 CentOS image is mounted on the connected x86 host, and the host uses an HTTP service to serve the content of the image to the CentOS running on the Arm cores.

This step is already done if you recently used the setup script as it automatically mounts the image on the host and starts the HTTP service. To verify that it works, try downloading something on the SmartNIC:

```
[root@localhost ~]# wget http://192.168.100.1/centos7/EULA
--2018-04-06 12:10:39-- http://192.168.100.1/centos7/EULA
Connecting to 192.168.100.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 215 [text/plain]
Saving to: 'EULA'

100%[=====] 215 --.-K/s in 0s

2018-04-06 12:10:39 (7.11 MB/s) - 'EULA' saved [215/215]
```

If this works, skip the following steps and go directly to the end to use the yum repository. After the `setup.sh` script is run, it sets up the host to do the HTTP service. However, this setup is lost if the host machine is rebooted. The set up script can be run again for set up and it can be done manually.

1. Ensure that your HTTP configuration file contains the following lines:

```
$ cat /etc/httpd/conf.d/pxeboot.conf
Alias /centos7 /var/pxe/centos7
<Directory /var/pxe/centos7>
    Options Indexes FollowSymLinks
    Require ip 127.0.0.1 192.168.100.0/24
</Directory>
```

This should have been added previously by the setup script.

2. Mount the aarch64 CentOS installation disk at the point specified by the conf file:

```
$ mount /root/CentOS-7-aarch64-Everything.iso /var/pxe/centos7
```

3. Start the HTTP service:

```
$ systemctl start httpd
```

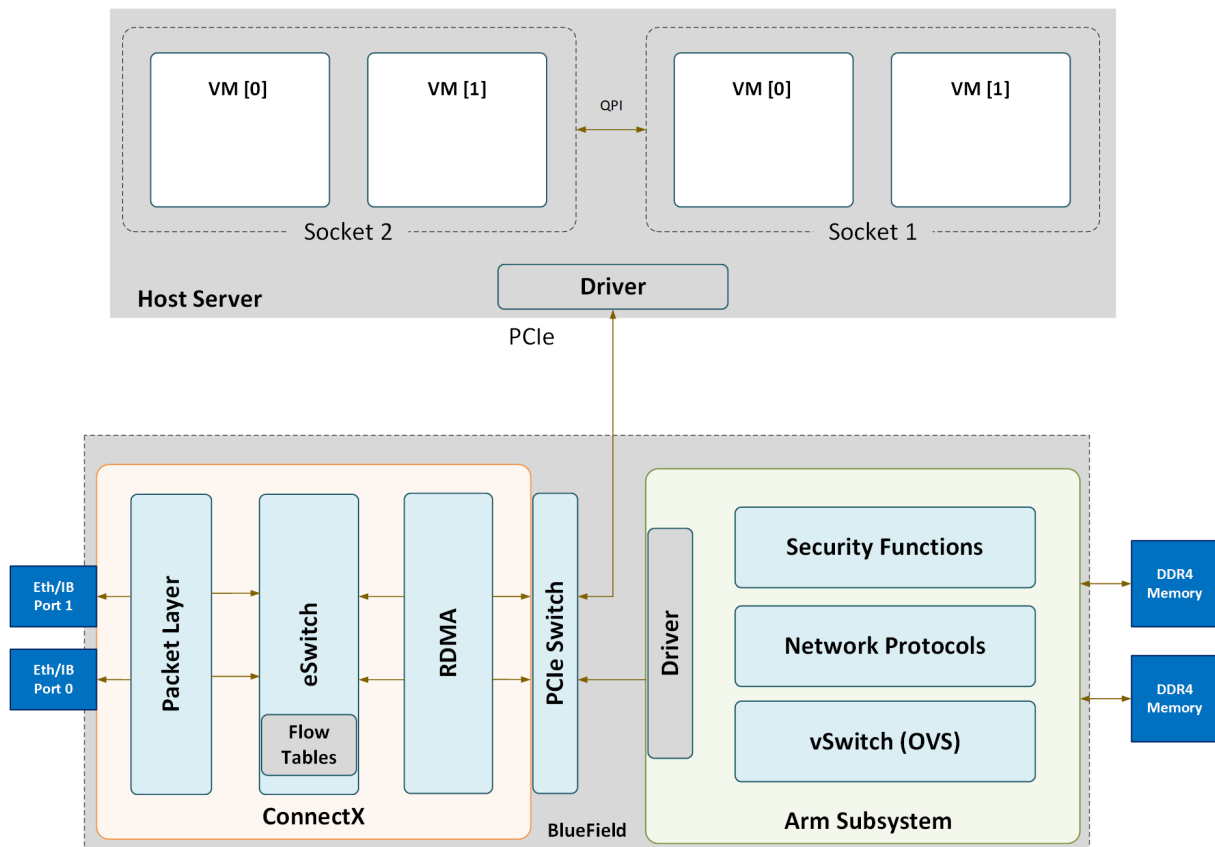
4. Ensure the RShim network is present between the host and the SmartNIC, and that there are no firewall rules on the host to prevent access to it.
5. Once completed, the "wget" command should work.
6. To configure the CentOS on the SmartNIC to use the mounted repository, add the repo file in /etc/yum.repos.d/ pointing to it:

```
$ mkdir /etc/backup
$ mv /etc/yum.repos.d/* /etc/backup/
$ cat > /etc/yum.repos.d/myyum.repo<<end
[myyum]
name=myyumsource
baseurl=http://192.168.100.1/centos7/
gpgcheck=0
end
$ yum clean all
```

7. After completion, the CentOS on the SmartNIC can use the local yum repo mounted on the host via "yum install".

Functional Diagram

The following is a functional diagram of the BlueField® DPU.



For each one of the BlueField DPU network ports, there are 2 physical PCIe networking functions exposed:

1. To the embedded Arm subsystem
2. To the host over PCIe

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 assuming they are installed. The BlueField network interface is compatible with ConnectX-5 and higher. The same network drivers are used both for BlueField and the ConnectX NIC family.

Modes of Operation

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

- [Separated host mode](#) (symmetric model)
- [Embedded function](#) (ECPF) ownership where the embedded Arm system controls the NIC resources and data path

⚠ This is the default mode of operation as of BlueField software 3.5 (MLNX_OFED 5.2).

- [Restricted mode](#) which is an extension of the ECPF ownership with additional restrictions on the host side

Separated Host

In this mode, the ECPF and the function exposed to the host are both symmetric. Each one of those functions has its own MAC address and is able to send and receive Ethernet and RDMA over Converged Ethernet (RoCE) traffic.

There is no dependency between the two functions. They can operate simultaneously or separately. The host can communicate with the embedded function as two separate hosts, each with its own MAC and IP addresses (configured as a standard interface). RDMA connection between the 2 interfaces is supported as well.

There is an equal bandwidth share between the two functions.

The limitations of this mode are as follows:

- Switchdev (virtual switch offload) mode is not supported on either of the functions
- SR-IOV is only supported on the host side

Configuring Separated Host Mode from ECPF Mode

On the server host, follow these steps:

1. Enable separated host mode. Run:

```
$ mst start  
$ mlxconfig -d /dev/mst/mt41682_pciconf0 s INTERNAL_CPU_MODEL=0
```

2. Power cycle.
3. Verify configuration. Run:

```
$ mst start  
$ mlxconfig -d /dev/mst/mt41682_pciconf0 q | grep -i model
```

4. Remove OVS bridges configuration from the Arm-side. Run:

```
$ ovs-vsctl del-br ovsbr1  
$ ovs-vsctl del-br ovsbr2
```

⚠ Make sure `CREATE_OVS_BRIDGES="no"` in `/etc/mellanox/mlnx-ovs.conf` so the bridges are not created automatically on next boot.

Embedded CPU Function Ownership Mode

This mode, known also as ECPF or DPU mode, is the default mode for BlueField DPU.

In ECPF mode, the NIC resources and functionality are owned and controlled by the embedded Arm subsystem. 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 embedded side has loaded and completed NIC configuration.
2. All ICM (Interface Configuration Memory) is allocated by the ECPF and resides in the embedded host memory.

3. The ECPF controls and configures the NIC embedded switch which means that traffic to and from the host interface always lands on the Arm side.

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.

Configuring ECPF Mode from Separated Host Mode

To enable this mode:

1. Start MST (Mellanox Software Tools) driver set service:

```
$ mst start
```

2. Identify the MST device:

```
$ mst status -v
```


Output example:

```
MST modules:
-----
MST PCI module is not loaded
MST PCI configuration module loaded
PCI devices:
-----
DEVICE_TYPE      MST                      PCI      RDMA      NET
-----
NUMA
BlueField(rev:0)  /dev/mst/mt41682_pciconf0.1  37:00.1  mlx5_1    net-ens1f1      0
BlueField(rev:0)  /dev/mst/mt41682_pciconf0     37:00.0  mlx5_0    net-ens1f0      0
```

3. Run the following commands on the Arm:

```
$ mlxconfig -d /dev/mst/mt41682_pciconf0 s INTERNAL_CPU_MODEL=1
$ mlxconfig -d /dev/mst/mt41682_pciconf0.1 s INTERNAL_CPU_MODEL=1
```

4. Power cycle the server.

 If OVS bridges `ovsbr1` and `ovsbr2` are not created (`ovs-vsctl show`) make sure `CREATE_OVS_BRIDGES="yes"` in `/etc/mellanox/mlnx-ovs.conf`.

Restricting DPU Host

By default, the host server has the same permissions as the Arm cores.

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 restricted 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
- FW flash is restricted

Enabling Host Restriction

1. Start the MST service.
2. Set restricted mode. From the Arm side, run:

```
$ mlxprivhost -d /dev/mst/mt41682_pciconf0 r --disable_rshim --disable_tracer --disable_counter_rd --disable_port_owner
```

If RShim is disabled, power cycle is required.


Disabling Host Restriction

Set back to privileged mode:

```
$ mlxprivhost -d /dev/mst/mt41682_pciconf0 p
```

The configuration takes effect immediately. System reboot is not required.

Kernel Representors Model

 This model is only applicable when the DPU is operating ECPF ownership mode.

BlueField® DPU uses netdev representors to map each one of the host side physical and virtual functions:

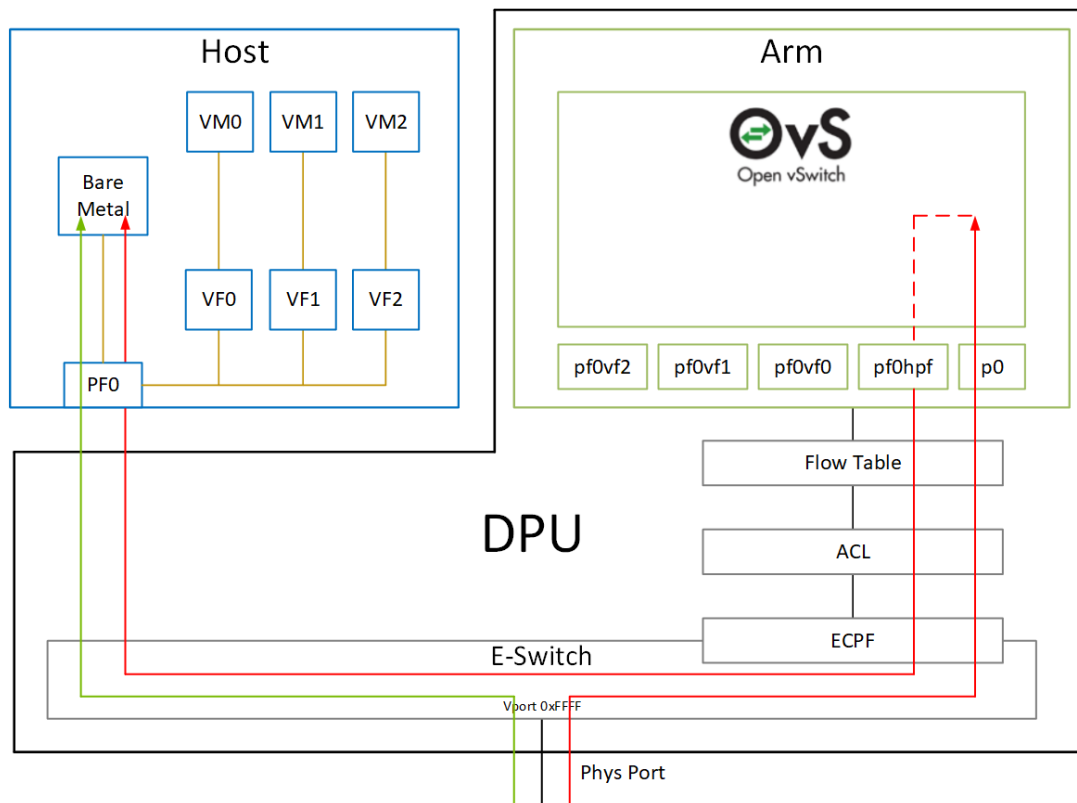
1. Serve as the tunnel to pass traffic for the virtual switch or application running on the Arm cores to the relevant PF or VF on the Arm side.
2. Serve as the channel to configure the embedded switch with rules to the corresponding represented function.

Those representors are used as the virtual ports being connected to OVS or any other virtual switch running on the Arm cores.

When in ECPF ownership 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 diagram below shows the mapping of between the PCI functions exposed on the host side and the representors. For the sake of simplicity, we show a single port model (duplicated for the second port).

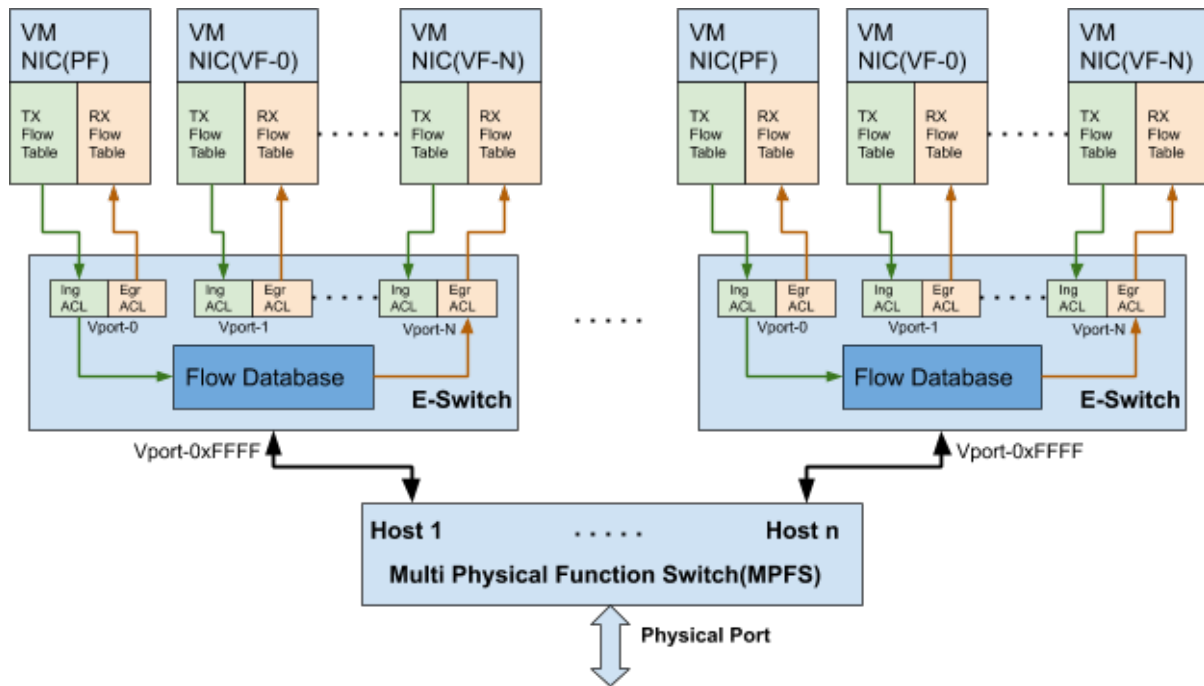


The red arrow demonstrates a packet flow through the representors, while the green arrow demonstrates the packet flow when steering rules are offloaded to the embedded switch. More details on that are available in the switch offload section.

Multi-Host

⚠ This is only applicable to DPUs 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.



Representors

Similar to [Kernel Representors Model](#), each host here has uplink representor, PF representor, and VF representors (if SR-IOV is enabled). There are 8 sets of representors (uplink/PF) (see example below). For each host to work with OVS offload, the corresponding representors must be added to 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

```

Virtual Switch on BlueField DPU

⚠ For general information on OVS offload using ASAP² direct, please refer to the [MLNX_OFED documentation](#) under OVS Offload Using ASAP² Direct.

⚠ ASAP² is only supported in Embedded (DPU) mode.

BlueField® supports [ASAP² technology](#). It utilizes the representors mentioned in the previous section. BlueField SW package includes OVS installation which already supports ASAP². 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. ASAP² 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 the first boot after the installation.

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 pf1sf0
  Port pf1hpf
    Interface pf1hpf
Bridge ovsbr1
  Port pf0hpf
    Interface pf0hpf
  Port p0
    Interface p0
  Port ovsbr1
    Interface ovsbr1
      type: internal
  Port pf0sf0
    Interface pf0sf0
ovs_version: "2.14.1"
```

The host is now connected to the network.

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
```

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. Mellanox BlueField Management Network Adapter).

| | | | | | |
|-------------|---|----|-----|---------|-------------------|
| Ethernet 2 | Mellanox ConnectX-4 Lx Ethernet Adapter | 6 | Not | Present | 24-8A-07-0D-E8-1D |
| Ethernet 6 | Mellanox ConnectX-4 Lx Ethernet Ad...#2 | 23 | Not | Present | 24-8A-07-0D-E8-1C |
| Ethernet 16 | Mellanox 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
```

Enabling OVS HW Offloading

⚠ Please make sure to have SR-IOV enabled prior to following this procedure. Please refer to [MLNX OFED documentation](#) under Features Overview and Configuration > Virtualization > Single Root IO Virtualization (SR-IOV) for instructions on how to do that.

Enable TC offload on the relevant interfaces. Run:

```
$ ethtool -K <PF> hw-tc-offload on
```

To enable the HW offload run the following commands (restarting OVS is required after enabling the HW offload):

```
$ ovs-vsctl set Open_vSwitch . Other_config:hw-offload=true
$ 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: pf0vfv0
port 5: pf0vfv1
port 6: pf0vfv2
```

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
```

Enabling OVS-DPDK Hardware Offload

For configuration procedure, please refer to the [MLNX OFED documentation](#) under Features Overview and Configuration > OVS Offload Using ASAP² Direct > OVS-DPDK Hardware Offloads Configuration. The procedure for BlueField begins at step 4.

For a reference setup configuration for BlueField-2 devices, please refer to the Mellanox Community post "[Configuring OVS-DPDK Offload with BlueField-2](#)".

Configuring DPDK and Running TestPMD

For a detailed procedure, please refer to the Mellanox Community post "[Configuring DPDK and Running testpmd on BlueField-2](#)".

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
```

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.

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"
```

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:

```
$ cat /proc/net/nf_conntrack
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
```

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:

```
$ cat /proc/net/nf_conntrack
```


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
```

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: $\text{size} = 4000000/(\text{num_of_groups}+1)$

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.


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.

```
$ /sbin/sysctl -w net.netfilter.nf_conntrack_max=1000000
```

Note that the OS has a default setting of maximum tracked connections. That can be configured by running:

This changes the maximum tracked connections setting to 1 million.

 Make sure `net.netfilter.nf_conntrack_tcp_be_liberal = 1` when using connection tracking.

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](#)".

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.

Configuring VXLAN Tunnel

1. Consider p0 to be the local VXLAN tunnel interface.
2. Build a VXLAN tunnel over OVS arm-ovs. Run:

```
ovs-vsctl 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
```

3. Connect pf0hpf to the same arm-ovs.
4. Run traffic over PF0 on x86 (the one connected to pf0hpf) to the host the DPU connected.
5. Configure the MTU of the PF used by VXLAN to at least 50 bytes larger than VXLAN-REP MTU.

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 "[Restricting DPU Host](#)"). 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.

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.

Configuring GRE Tunnel

1. Consider p0 to be the local GRE tunnel interface.
2. Build a GRE tunnel over OVS arm-ovs. Run:

```
ovs-vsctl add-port gre_br gre0 -- set interface gre0 type=gre options:local_ip=1.1.1.1
options:remote_ip=1.1.1.2 options:key=100
```

3. Connect pf0hpf to the same arm-ovs.
4. Run traffic over PF0 on x86 (the one connected to pf0hpf) to the host the DPU connected.

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,t1=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 "[Restricting DPU Host](#)"). 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.

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.

Configuring Geneve Tunnel

1. Consider p0 to be the local Geneve tunnel interface.
2. Build a Geneve tunnel over OVS arm-ovs. Run:

```
ovs-vsctl add-port geneve_br gv0 -- set interface gv0 type=geneve options:local_ip=1.1.1.1  
options:remote_ip=1.1.1.2 options:key=100
```

3. Connect pf0hpf to the same arm-ovs.
4. Run traffic over PF0 on x86 (the one connected to pf0hpf) to the host the DPU connected.

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 "[Restricting DPU Host](#)"). 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.

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
```

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
```

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 mirrored 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_ethertype 0x800 \
        vlan_id 100 \
        vlan_prio 0 \
    action vlan pop \
    action mirrored egress redirect dev pf0vf0
```

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 mirrored 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 mirrored egress redirect dev pf0vf0
```

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.

Configuring Uplink MTU

In order to configure the port MTU while operating in [SmartNIC Mode](#), you must restrict the BlueField® SmartNIC host as instructed in section "[Restricting SmartNIC Host](#)".

Once the host is restricted, the port MTU is configured by changing the MTU of the uplink representor (p0, or p1).

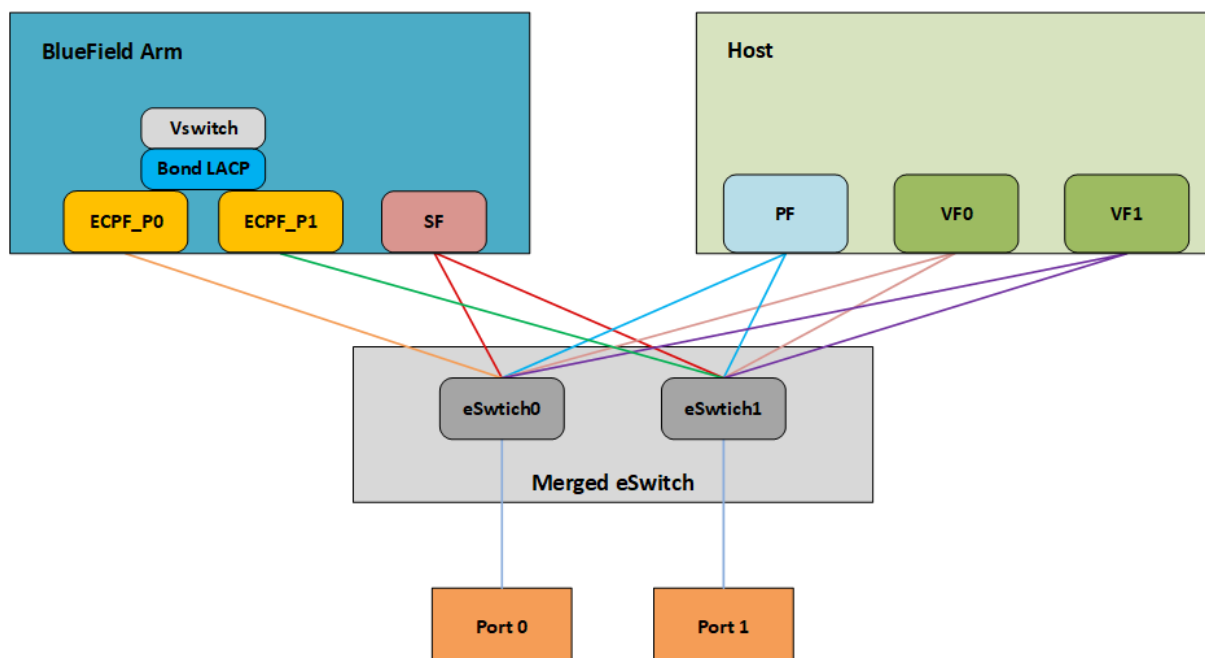
BlueField Link Aggregation

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 fail.

BlueField 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 SmartNIC is set in embedded function ownership mode for both ports.

The diagram below describes this configuration:



Prerequisites

- The mlx5 drivers are not loaded on the server host OS side. Run:

```
$ /etc/init.d/openibd force-stop
```

- All the representors on the Arm side are disconnected from the OVS bridge.

LAG Configuration

1. Create the bond interface:

```
$ nmcli connection add con-name bond1 type bond ifname bond1 miimon 100 mode 4
```

2. EnSubordinate both the uplink representors to the bond interface:


```
$ nmcli connection add con-name sub-p0 type bond-sub ifname p0 master bond1  
$ nmcli connection add con-name sub-p1 type bond-sub ifname p1 master bond1
```

3. Bring the interfaces up:


```
$ nmcli connection up sub-p0  
$ nmcli connection up sub-p1  
$ nmcli connection up bond1
```


4. As a result, only a single Physical function would be available to the host side.
5. To hide the second PF on the host, run:

```
$ mlxconfig -d /dev/mst/mt41682_pciconf0 s HIDE_PORT2_PF=True
```

 If you do not perform this, the function will still be visible, but it will not receive any traffic.

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 second port (pf1hpf) would still be visible, but it should not be added to OVS bridge.

 Trying to change bonding configuration (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 host driver before altering SmartNIC bonding configuration in order to avoid this.

 When performing driver reload (openibd restart) or reboot, it is required to remove bond configuration from NetworkManager, and to reapply the configurations after the driver is fully up.

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. Mediated 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.
```

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`.

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 BlueField® SmartNIC in the configurations listed below.

Separate Host Mode

RoCE is supported from both the host and Arm system.

InfiniBand is supported on the host system.

Embedded CPU Mode

RDMA Support on Host

To use RoCE on a host system's PCI 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.

InfiniBand is not supported on the host side in this mode.

RDMA Support on Arm

RoCE is unsupported on the Arm system on the PCI PF. However, RoCE is fully supported using mediated devices as explained under "[Mediated Devices](#)". Mediated devices are created by default, allowing RoCE traffic without further configuration.

InfiniBand is supported on the Arm system on the PCI PF in this mode.

Controlling Host PF and VF Parameters

BlueField® allows control over some of the networking parameters of the PFs and VFs running on the host side.

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 need to reload the Mellanox 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
```

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
```

Query 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.

DPDK on BlueField SmartNIC

Please refer to "[Mellanox BlueField Board Support Package](#)" in the DPDK documentation.

NVMe SNAP on BlueField SmartNIC

Mellanox NVMe SNAP™ (Software-defined Network Accelerated Processing) technology enables hardware-accelerated virtualization of NVMe storage. NVMe 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 NVMe SNAP. Any logic may be applied to the data via the NVMe SNAP framework and transmitted over the network, on either Ethernet or InfiniBand protocol, to a storage target.

NVMe SNAP combines unique hardware-accelerated storage virtualization with the advanced networking and programmability capabilities of the BlueField SmartNIC. NVMe SNAP together with the BlueField SmartNIC enable a world of applications addressing storage and networking efficiency and performance.

To enable NVMe SNAP on your SmartNIC, please contact Mellanox Support.

RegEx Acceleration

BlueField-2 DPU supports high-speed RegEx acceleration. This allows the host to offload multiple RegEx jobs to the SmartNIC. 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 a RegEx compiler, please contact NVIDIA Support.

Configuring RegEx Acceleration on BlueField-2

The RegEx application can run either from the host or Arm.

Before running application from the host side, user must perform the following:

```
host$ sudo /etc/init.d/openibd stop
bluefield$ echo 1 > /sys/class/net/p0/smart_nic/pf/regex_en
bluefield$ cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
400
// make sure to allocate 200 additional hugepages
bluefield$ echo 600 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
bluefield$ systemctl start mlx-regex // To verify the service is properly running, use "systemctl
status mlx-regex"
host$ sudo /etc/init.d/openibd start
```

Public Key Acceleration

NVIDIA BlueField DPU 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-Hellman, Elliptic Curve Cryptography, and the Federal Digital Signature Algorithm (DSA as documented in FIPS-186) public-private key systems.

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 $\geq 1.0.0$ and $\leq 1.1.1$ are supported. 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](#).

IPsec Functionality

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.

Please refer to the community post "[BlueField IP Forwarding and IPsec SPI RSS](#)" for configuration and tuning instructions for optimal performance results.

IPsec Hardware Offload: Crypto Offload

⚠ This feature is supported only on BlueField-2 based platforms.

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.

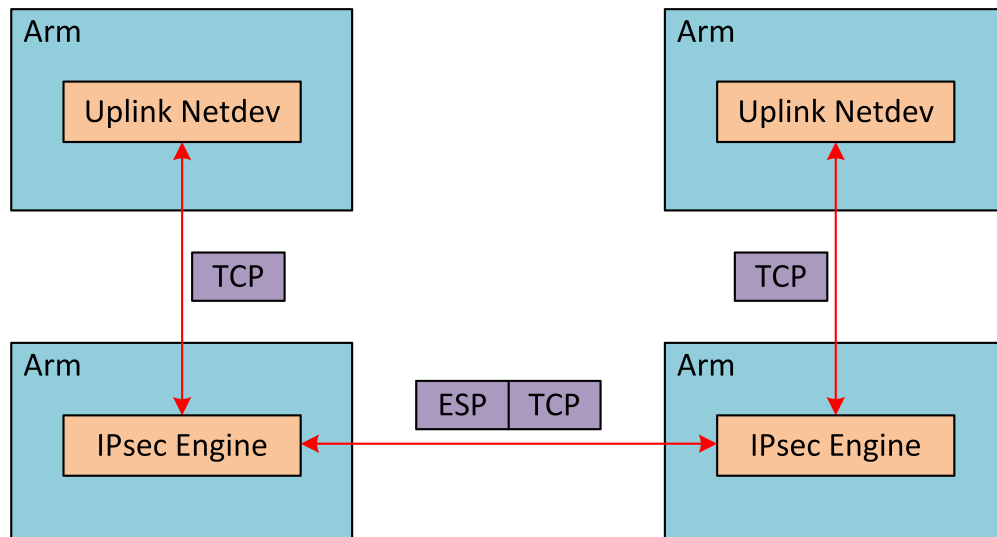
Please refer to the [MLNX_OFED documentation](#) under Features Overview and Configuration > Ethernet Network > IPsec Crypto Offload for more information on enabling and configuring this feature.

Please note that to use IPsec crypto offload with OVS, you must disable hardware offloads.

IPsec Hardware Offload: Full Offload

⚠ This feature is supported only on BlueField-2 based platforms.

IPsec full offload offloads both IPsec crypto and IPsec encapsulation to the hardware. IPsec full offload is configured on the Arm via the uplink netdev. The following figure illustrates IPsec full offload operation in hardware.




Configuring IPsec Full Offload

⚠ If you are working directly with the `ip xfrm` tool, you must use the `/opt/mellanox/iproute2/sbin/ip` to benefit from IPsec full offload support.

Explicitly enable IPsec full offload on the Arm cores before full offload rule is configured.

1. Delete the two SFs created on the BlueField device upon boot (one per port if the port is in switchdev mode). Run:

```
# sudo mlnx-sf -a remove --uuid <UUID1>
# sudo mlnx-sf -a remove --uuid <UUID2>
```

 You may retrieve the UUID (unique device ID) information by running the following command:

```
# /opt/mellanox/iproute2/sbin/devlink dev show
pci/0000:03:00.0
pci/0000:03:00.1
mdev/0214613e-60a3-4437-8de6-efbb41d8436e
mdev/d3116441-04fb-4ea1-a2b3-63fb739e0f4a
```

For example:

```
# sudo mlnx-sf -a remove --uuid 0214613e-60a3-4437-8de6-efbb41d8436e
# sudo mlnx-sf -a remove --uuid d3116441-04fb-4ea1-a2b3-63fb739e0f4a
```

2. Move the SR-IOV mode to legacy. Run on Arm:

```
devlink dev eswitch set pci/0000:03:00.0 mode legacy
```

3. Set the IPsec to full offload. Run on Arm:

```
echo full > /sys/class/net/p0/compat/devlink/ipsec_mode
```


4. Enable firmware steering. Run on Arm:

```
echo dmfs > /sys/bus/pci/devices/0000\:03\:00.0/net/p0/compat/devlink/steering_mode
```

5. Enable the switchdev SR-IOV mode. Run on Arm:

```
devlink dev eswitch set pci/0000:03:00.0 mode switchdev
```

6. To enable IPsec full offload on the second port, please perform steps 2-5 on pci/0000:03:00.1.

 To use IPsec full offload with strongSwan, please refer to section "[IPsec Full 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, instead of the parameter "offload", use "full_offload" to achieve IPsec full offload.

Configuring IPsec Rules with iproute2

The following example configures IPsec full offload rules with local address 192.168.1.64 and remote address 192.168.1.65:

```

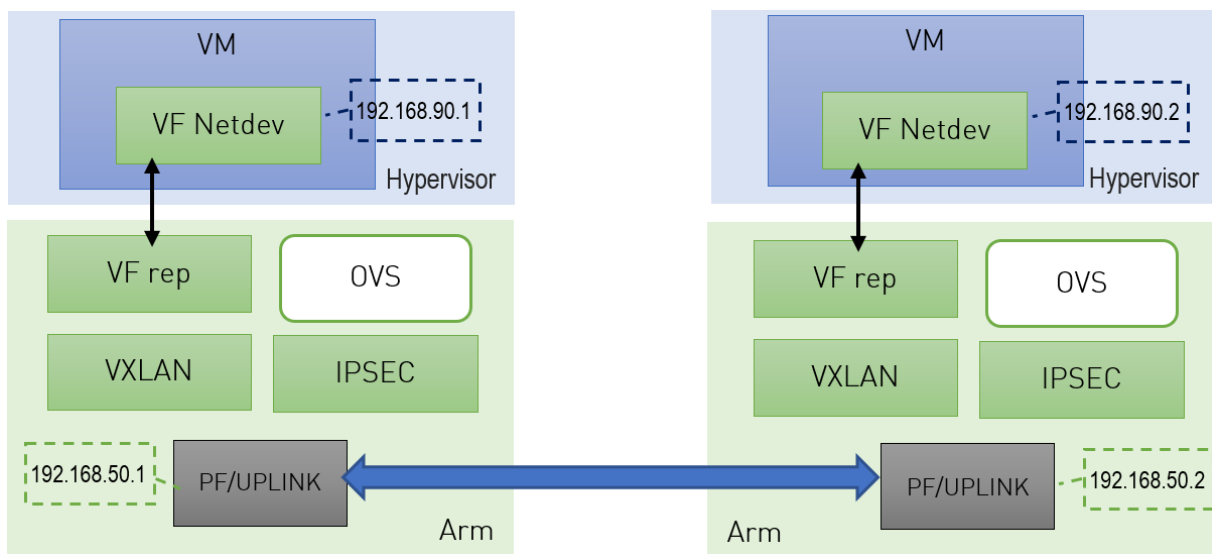
ip xfrm state add src 192.168.1.64/24 dst 192.168.1.65/24 proto esp spi 0x4834535d reqid 0x4834535d mode transport
aead 'rfc4106(gcm(aes))' 0xc57f6f084ebf8c6a71dd9a053c2e03b94c658a9bf00dd25780e73948931d10d08058a27c 128
full_offload dev p0 dir out sel src 192.168.1.64 dst 192.168.1.65
ip xfrm state add 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
full_offload 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
sudo ip xfrm policy add src 192.168.1.65 dst 192.168.1.64 dir fwd tmpl src 192.168.1.65/24 dst 192.168.1.64/24
proto esp reqid 0x2be60844 mode transport

```

IPsec Full Offload strongSwan Support

BlueField DPU supports configuring IPsec rules using strongSwan 5.9.0bf (yet to be upstreamed) which supports new fields in 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 full HW offload requires using VXLAN together with IPsec as shown here.

1. Follow the procedure under section "[Configuring IPsec Full 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.

Setting IPsec Full Offload Using strongSwan

strongSwan configures IPsec HW full offload using a new value added to its configuration file `swanctl.conf`.

The file should be placed under "sysconffdir" 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 Full 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 {
        local_ts = 192.168.50.1/24 [udp/4789]
        remote_ts = 192.168.50.2/24 [udp/4789]
        esp_proposals = aes128gcm128-x25519-esn
        mode = transport
        policies_fwd_out = yes
        hw_offload = full
      }
    }
    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 = full" is responsible for configuring IPsec HW full offload
Full offload support has been added to the existing `hw_offload` field and preserves backward compatibility

| Value | Description | Note |
|-------|---|----------|
| no | Do not configure HW offload, fail if not supported | Existing |
| yes | Configure crypto HW offload if supported by the kernel, fail if not supported | Existing |
| auto | Configure crypto HW offload if supported by the kernel, do not fail | Existing |

| Value | Description | Note |
|-------|---|------|
| full | Configure full HW offload if supported by the kernel, fail if not supported | New |

⚠ Whenever the value of `hw_offload` is changed, `strongSwan` configuration must be reloaded.

⚠ Switching to crypto HW offload requires setting up `devlink/ipsec_mode` to "none" beforehand.

⚠ Switching to full HW offload requires setting up `devlink/ipsec_mode` to "full" beforehand

- "[udp/4789]" is crucial for instructing `strongSwan` to IPsec only VXLAN communication

⚠ Full HW offload can only be done on what is streamed over VXLAN.

Mind the following limitations:

| Field | Limitation |
|----------------------------|---|
| <code>reauth_time</code> | Ignored if set |
| <code>rekey_time</code> | Do not use. Ignored if set. |
| <code>rekey_bytes</code> | Do not use. Not supported and will fail if it is set. |
| <code>rekey_packets</code> | Use for rekeying |

Running strongSwan Example

Notes:

- IPsec daemons are started by `systemd strongswan-starter.service`
- Use `systemctl [start | stop | restart]` to control IPsec daemons through `strongswan-starter.service`. For example, to restart, the command `systemctl restart strongswan-starter.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 Full Offload strongSwan Support](#)").

```
# systemctl start strongswan-starter.service
# swanctl --load-all
```

The following should appear.

```
Starting strongSwan 5.9.0bf 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
```

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_TRANS) 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_TRANS) 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.

Building strongSwan

Do this only if you want to build your own BFB and would like to rebuild strongSwan.

1. strongSwan IPsec full version can be found [here](#) (tag: 5.9.0bf).
2. Install dependencies mentioned [here](#). libgmp-dev is missing from that list, so make sure to install that as well.
3. Git clone <https://github.com/Mellanox/strongswan.git>.
4. Git checkout BF-5.9.0.
5. Run "autogen.sh" within the strongSwan repo.
6. Run the following:

```
configure --enable-openssl --disable-random --prefix=/usr/local --sysconfdir=/etc --enable-systemd
make
make install
```

Note:

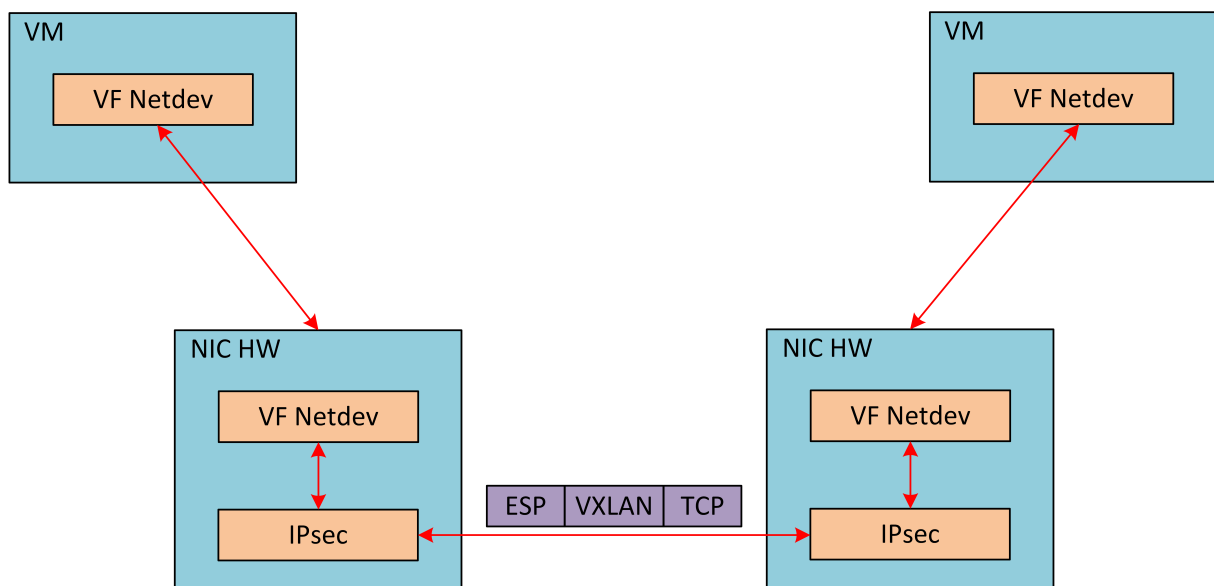
- `--enable-systemd` enables the `systemd` service for strongSwan service present inside github repo (see step 3) at `init/systemd-starter/strongswan-starter.service.in`. This service file is meant for Ubuntu, Debian and Yocto distributions. For CentOS, the contents of the above file must be replaced by the one present in `systemd-conf/strongswan-starter.service.in.centos` (inside the github repo) before running the configure script above.
- 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.

IPsec Full Offload and OVS Offload

IPsec full 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 full offload and OVS VXLAN offload.



⚠ OVS offload and IPsec IPv6 do not work together.

QoS Configuration

⚠ To learn more about port QoS configuration, please refer to [this](#) Mellanox Community post.

Rate Limiting Host PF and VF

From the Arm, users may limit the transmit rate of the PF in the host. For example, these commands limit the transmit rate of the PF of ECPF's p0 to 1000 Mbps and the virtual function #0 to 500 Mbps:

```
$ echo 1000 > /sys/class/net/p0/smart_nic/pf/max_tx_rate
$ echo 500 > /sys/class/net/p0/smart_nic/vf0/max_tx_rate
```

VirtIO-net Emulated Devices

⚠ This feature is supported at beta level. Please contact NVIDIA Support for enablement.

This feature enables users to create VirtIO-net emulated PCIe devices in the system where the NVIDIA® BlueField®-2 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 127 virtio-net PCIe PF Ethernet NIC devices and virtio-net PCI VF Ethernet NIC devices in the host system where the DPU is plugged in.

DPU software also enables users to create virtio block PCI PF and SR-IOV PCI VF devices. This is covered in the *NVIDIA Mellanox NVMe SNAP and virtio-blk SNAP Documentation*.

VirtIO-net PF Devices

This section covers managing virtio-net PCI PF devices using virtio-net-controller.

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. Cold reboot the host system.
3. Apply the following configuration on the DPU in three steps:

```

$ 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=16 \
VIRTIO_NET_EMULATION_VENDOR_ID=0x1af4 \
VIRTIO_NET_EMULATION_DEVICE_ID=0x1041 \
VIRTIO_NET_EMULATION_CLASS_CODE=0x028000 \
ECPF_ESWITCH_MANAGER=1 \
ECPF_PAGE_SUPPLIER=1 \
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

```

4. Cold reboot the host system a second time.

Creating 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. Start virtio-net controller on the DPU. Run:

```
$ virtio_net_controller -i mlx5_0
```

2. Create a hotplug virtio-net device. Run:

```
$ create virtio-net dev 0x0 0C:C4:7A:FF:22:93 1500 3 1024
```

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. This device is uniquely identified by its index. This index is used to query and update device attributes.

3. Set the link state up of the newly create device. Run:

```
$ change virtio-net dev 0 link 1
```

4. Change the SF netdevice's queue parameters. The SF's netdevice is printed by the dump command of the virtio-net-controller. It is also printed during the create command. Run:

```

$ ethtool -L <sf_netdev> combined 1
$ ethtool -G <sf_netdev> rx 16
$ ethtool -G <sf_netdev> tx 128

```

5. Bring up the representor port of the device. Run:

```

$ ip link set dev pf0sf1 up
$ ovs-vsctl add-port <bridge> pf0sf1

```

Once steps 1-5 are completed, virtio-net device should be available in the host system.

6. To show all the device configurations of virtio-net device that you created, run:

```
$ show virtio-net dev 0
```

7. To list all the virtio-net devices, run:

```
$ list virtio-net devices
```

8. Once usage is complete, to hot-unplug a VirtIO net device, run:

```
$ destroy virtio-net dev 0
```

Virtio-net SR-IOV VF Devices

This section covers managing virtio-net PCI SR-IOV VF devices using virtio-net-controller.

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 x86 host, blacklist kernel modules virtio_pci and virtio_net by adding them to /etc/modprobe.d/blacklist.conf:

```
blacklist virtio_pci
blacklist virtio_net
```

2. On the x86 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. Reboot x86 host, make sure those modules are not loaded.

```
# lsmod | grep -i virtio
```

- ⚠** Please make sure those modules are blacklisted before proceeding. If those modules are built into kernel image, they are not blacklistable. In that case, users must have a way to start the virtio_net_controller from Arm using SSH or serial/oob interfaces. This can be achieved by connecting the RShim/serial cable to another host, or setting up the OOB interface.

4. Run the following command on the DPU:

```
mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
```

5. Cold reboot the host system.
6. Apply the following configuration on the DPU in three steps.

- a.

```
$ 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=1 \  
PCI_SWITCH_EMULATION_NUM_PORT=16 \  
VIRTIO_NET_EMULATION_ENABLE=1 \  
VIRTIO_NET_EMULATION_NUM_VF=127 \  
VIRTIO_NET_EMULATION_NUM_PF=1 \  
VIRTIO_NET_EMULATION_NUM_MSIX=4 \  
VIRTIO_NET_EMULATION_VENDOR_ID=0x1af4 \  
VIRTIO_NET_EMULATION_DEVICE_ID=0x1041 \  
VIRTIO_NET_EMULATION_CLASS_CODE=0x028000 \  
ECPF_ESWITCH_MANAGER=1 \  
ECPF_PAGE_SUPPLIER=1 \  
SRIOV_EN=1 \  
PF_SF_BAR_SIZE=8 \  
PF_TOTAL_SF=170
```

c.

```
$ mlxconfig -d /dev/mst/mt41686_pciconf0.1 s PF_TOTAL_SF=170 PF_SF_BAR_SIZE=8
```

7. Cold reboot the host system.

Creating Virtio-net SR-IOV VF Devices

The virtio-net-controller application console must be kept alive to maintain the functionality of the static PF and its VFs.

1. Start virtio-net controller on the SmartNIC. Run:

```
# virtio_net_controller -i mlx5_0
```

2. On the x86 host, make sure the static virtio network device presents.

```
# lspci | grep -i virtio  
85:00.3 Network controller: Red Hat, Inc. Virtio network device
```

3. On x86 host, run:

```
# modprobe -v virtio_pci  
# modprobe -v virtio_net
```

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
```

4. To create SR-IOV VF devices on the x86 host, run:

```
# echo 2 > /sys/bus/pci/drivers/virtio-pci/0000\:85\:00.3/sriov_numvfs
```

2 VFs should be created from x86 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
```

5. From the DPU virtio-net controller, run:


```
virtio-net-controller> list virtio-net devices
PF VF BDF      SF-Rep  Fw Open Stat Features  Queues  Msix MAC              MTU
0   85:0.3 pf0sf1 Y     Y  0x0f 0x2300470028 8      0  0cc47aff2292 1500
0   85:4.5 pf0sf2 Y     Y  0x0f 0x300470028 8      0  c6237b326745 1500
1   85:4.6 pf0sf3 Y     Y  0x0f 0x300470028 8      0  724833666998 1500
```

Bring up corresponding SF and add to OVS bridge:

```
# ip link set dev pf0sf2 up
# ovs-vsctl add-port <bridge> pf0sf2
```

Now the VF is functional.

Deep Packet Inspection

 This feature is supported at alpha level.

Deep packet inspection (DPI) is a method of examining the full content of data packets as they traverse a monitored network checkpoint. DPI is part of [DOCA](#) software solution for NVIDIA® BlueField®-2 DPU.


DPI provides a more robust mechanism for enforcing network packet filtering as it can be used to identify and block a range of complex threats hiding in network datastreams, such as:

- Malicious applications
- Malware data exfiltration attempts
- Content policy violations
- Application recognition
- Load balancing

To access DPI software and documentation, please contact NVIDIA Support at: networking-support@nvidia.com.

Controller Card Operation


The [Mellanox BlueField Controller Card Family](#) is the perfect solution for managing backend NVMe storage, All Flash Arrays (AFA), compute and storage disaggregation, and hyperconverged systems.

-  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:

- [Bring-Up and Driver Installation](#)

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.

BlueField Software

Mellanox provides software that enables users to fully utilize the BlueField® DPU and enjoy the rich feature-set it provides. Using BlueField software packages, users are able to:

- 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.

The BlueField family of DPU devices combines an array of 64-bit Armv8 A72 cores coupled with the ConnectX® interconnect. 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 quickly and efficiently design, implement and verify their control-plane and data-plane applications.

BlueField SW ships with the Mellanox BlueField Controller Cards. BlueField SW is a reference Linux distribution based on the Yocto Poky distribution and extended to include the Mellanox OFED stack for Arm and a Linux kernel which supports NVMe-oF. This SW distribution is capable of running all customer-based Linux applications seamlessly. Yocto also provides an SDK that contains an extremely flexible cross-build environment allowing software targeted for the BlueField DPU to build on virtually any x86 server running any Linux distribution.

The following are other software elements delivered with BlueField DPU:

- Arm Trusted Firmware (ATF) for BlueField
- UEFI for BlueField
- Hardware Diagnostics
- Mellanox OFED stack
- Mellanox MFT

Software On eMMC

The BlueField Controller Card boots off eMMC upon power-up. The image flashed on the eMMC from the factory is the Yocto Linux.

Run the following command to discover the BlueField Software version:

```
cat /etc/bluefield_version
```

Yocto Distribution Installation

The BlueField tarball comes with pre-built Yocto images that can be installed.

The core-image-full image is a full root filesystem image that is appropriate for imaging on the rootfs partition of the eMMC.

1. Prepare the host environment. For more information, refer to [Preparing the Host-Side Environment](#).
2. Boot the BlueField Controller Card over USB using the samples/install.bfb image.
3. Refer to samples/README.install for instructions.
4. To boot the BlueField Controller Card over USB from the server host, run:

```
cat install.bfb > /dev/rshim0/boot
```

5. To prepare the eMMC for Yocto installation from the Arm, run:

```
/opt/mlnx/scripts/bfinst --fullfs /tmp/core-image-full-bluefield.tar.xz
```

6. After the installation is done, execute lowercase reboot on the Arm.

```
shutdown -r now
```

7. Verify the version via:


```
cat /etc/bluefield_version
```

PXE Server Configuration on Host Side

Before installing CentOS 7 on the BlueField Controller Card, you need to configure the PXE server on the host side (x86) to allow the deployment of the CentOS image over the BlueField Controller Card.


Download the CentOS installation iso file from the following link:

```
# Download the centos installation iso file from http://mirror.centos.org/altarch/7/isos/aarch64/CentOS-7-aarch64-Everything.iso
# cd <BF_INST_DIR>/distro/rhel/pxeboot
# ./setup.sh -d <BF_INST_DIR> -i <centos-installation.iso> [-c <ttyAMA0 >]
```

 UART0 (ttyAMA0) is used by default, or you can use "-c ttyAMA0" to manually specify UART0.

Installing Linux on BlueField Controller Card


This section demonstrates CentOS 7.4 installation on the BlueField Controller Card. Other OSs work similarly with the PXE boot installation process.


-  Before installing the preferred OS on the BlueField Controller Card, make sure you install the BlueField Controller Card in a JBOF System. Installing it in a host system may damage the card.

Software Requirements

- CentOS 7.4 Linux OS. To get CentOS 7.4 image, run:

```
wget http://archive.kernel.org/centos-vault/altarch/7.4.1708/isos/aarch64/CentOS-7-aarch64-Everything.iso
```

 Some required drivers do not compile and load if running CentOS 5.x or earlier.


 Please note that CentOS 7.5 is not supported.

- Access to the latest BlueField Controller Card SW bundle: Mellanox uses [bnx.com](https://www.bnx.com) to distribute BlueField software. Contact your sales/support representative for a custom link to download BlueField software releases.
- In this document, we assume the tarball BlueField-1.0.alphaX.XXXXX.tar.gz is extracted at / root, to do this, run the following command:

```
tar -xvf BlueField-1.0.alphaX.XXXXX.tar.xz -C /root
```

Preparing Host-Side Environment

Before installing the preferred OS on the BlueField Controller Card, the host must be set up for it to be capable of provisioning the BlueField Controller Card. The RShim USB driver is installed on the host to communicate with the RShim device on the BlueField DPU. The RShim USB driver must be installed so that it can push the initial bootloader and supply the OS image for PXE boot through the USB connection.

-  This process only needs to be done on the host machine which is provisioning the BlueField Controller Card, it is not required on the end machine.

Setup Procedure With Installation Script

If the host is running CentOS 7 (or equivalent) on the host, you may run a script to complete all the steps detailed in [Preparing the Host-Side Environment](#).

```
/root/BlueField-1.0.alphaX.XXXXX/distro/rhel/pxeboot/setup.sh \
-d /root/BlueField-1.0.alphaX.XXXXX/ \
-i /root/CentOS-7-aarch64-Everything.iso \
-o /root/dd-rhel7.4-mlnx-ofed-4.2-1.4.10.0-aarch64.iso \
-c ttyAMA0\
-k
```

Note that there should be no firewall blocking the IP communication between the BlueField Controller Card and the server host machine. If a firewall exists, disable it with the following commands:

```
iptables -F
iptables -t
nat -F
```

- The “-d” flag points to where the tar file has been extracted from, the script uses this directory to find all the source code it needs.
- The “-i” flag points to the OS installation disk. This is the image that is accessed via PXE boot to install the OS on the BlueField Controller Card.
- The “-o” flag points to the Mellanox OFED driver disk for Arm. Download and extract it from http://www.mellanox.com/page/products_dyn?product_family=34.
- The “-c” flag specifies the default UART port for the OS to use since the BlueField DPU has two Arm UARTs. For the BlueField Controller Card, “ttyAMA0” is used, which is UART0.
- The “-t” flag is optional and needed for nonpxe boot. When specified and given the argument of what Controller card is set (BlueField Controller Card in this case), it generates a “nonpxe.bfb” file which contains the install kernel and rootfs. If this file is pushed to the RShim boot device, it automatically runs the installation process and skips the initial UEFI PXE boot operations. (the -t flag). Please refer to distro/rhel/pxe/README.
- The optional “-k” flag kickstarts auto-installation based on a default kickstart file which is installed as /var/pxe/ks/ks.cfg (optional).

Setup Procedure Without Installation Script


If the host is running CentOS 7 or equivalent, please refer to [Preparing the Host-Side Environment](#) for a simpler way to perform the installation using an installation script.

The following sections demonstrate CentOS 7 installation, however, installation in other environments should be relatively similar.

Step 1: Set up RShim Interface

The RShim driver communicates with the RShim device on the BlueField DPU. The RShim is in charge of many miscellaneous functions of the DPU, including resetting the Arm cores, providing the initial bootstream, and using the TMFIFO and the RShim network, to exchange network and console data with the host.

The RShim can be reached by the host via the USB connector and the PCIe slot. It is preferable, however, to use the USB connection.

 To enable access to the RShim via the PCIe slot, a link must be open through firmware. This is done by adding “multi_function.rshim_pf_en = 0x1” to the [fw_boot_config] section in the firmware's ini file.

Step 2: Install RShim Drivers

To install the kernel modules, please follow the instruction in section [RShim Host Driver](#).

Step 3: Configure TFTP Server

The host should be configured to act as a TFTP server to the BlueField Controller Card via the USB RShim network. This server provides the required files by the BlueField Controller Card to perform the PXE boot for installing the preferred OS.

- ⚠** Configuring the TFTP server requires a TFTP package. If it is not installed, install it via “yum install tftp” or “apt-get tftp”, depending on your Linux distribution.
- Note: On some versions, the TFTP package cannot be found. In such cases, install “xinetd”.

1. Extract the OS image and copy the required PXE boot components:

```
mount -t iso9660 -o loop CentOS-7-aarch64-Everything.iso /mnt
mkdir -p /var/lib/tftpboot/centos/7.4
cp /mnt/EFI/BOOT/BOOTAA64.EFI /var/lib/tftpboot/ cp /mnt/EFI/BOOT/grubaa64.efi /var/lib/tftpboot/
cp /mnt/images/pxeboot/vmlinuz /var/lib/tftpboot/centos/7.4
cp /mnt/images/pxeboot/initrd.img /var/lib/tftpboot/centos/7.4/initrd-orig.img
```

2. Patch the initrd with the eMMC driver and TMFIFO (RShim network) driver:

```
mkdir -p /tmp/.bfcenos mkdir -p $(/tmp/.bfinstdd cd /tmp/.bfcenos
xzcat /var/lib/tftpboot/centos/7.4/initrd-orig.img | cpio -idm mount
/root/BlueField-1.0.alpha3.10409/distro/rhel/bluefield_dd/bluefield_dd-4.11.0- 22.el7a.aarch64.iso
/tmp/.bfinstddmkdir -p usr/lib/modules/4.11.0-22.el7a.aarch64/updates/cp
/tmp/.bfinstdd/lib/modules/4.11.0-22.el7a.aarch64/updates/dw_mmc*.ko usr/lib/mod- ules/
4.11.0-22.el7a.aarch64/updates/cp
/tmp/.bfinstdd/lib/modules/4.11.0-22.el7a.aarch64/updates/tmfifo.ko usr/lib/mod- ules/
4.11.0-22.el7a.aarch64/updates/cp
/root/BlueField-1.0.alpha3.10409/distro/rhel/bluefield_dd/bluefield_dd-4.11.0- 22.el7a.aarch64.iso ./
bluefield_dd.iso
umount /tmp/.bfinstdd; rmdir /tmp/.bfinstdd chown root:root
* -R
depmod -b
/tmp/.bfcenos 4.11.0-22.el7a.aarch64
find . | cpio
-oc | xz --check=crc32 --lzma2=dict=32MiB >
/var/lib/tftpboot/centos/7.4/ initrd.img
```

- ⚠** These commands assume that you are using kernel version “4.11.0-22.el7a.aarch64”. If you are using a different version, utilize the corresponding bluefield_dd.iso. If none is found, compile one by running the following: source /path/to/SDK/environment-setup-aarch64-poky-linux; unset LDFLAGS; ./build-dd.sh /path/to/kernel-devel-4.11.0-44.el7a.aarch64.rpm

3. Change the grub configuration to PXE boot over the right location:

```
cat >/var/lib/tftpboot/grub.cfg <<EOF
menuentry 'Install centos/7.4 AArch64 - BlueField'
--class red --class
gnu-linux
--class gnu --class os {
linux (tftp)/centos/7.4/vmlinuz ro ip=dhcp method=http://192.168.100.1/centos7 inst.dd=/bluefield_dd.iso
console=ttyAMA0
initrd (tftp)/centos/7.4/initrd.img
} EOF
```

4. Start the TFTP server:

```
systemctl restart tftp
```

- ⚠** Based on the system, the user may need to use “system TFTP restart” instead. Also, if required, the user might need to switch use “xinetd” instead of “TFTP”.

Step 4: Set Up the DHCP Server

DHCP server set up on the host is required for BlueField Controller Card to get a private IP from the host for PXE boot process completion. Configure the correct server names and domain names so that the BlueField Controller Card can connect to the network via the host later on.

1. Get the server/domain names on the host:

```
bash-4.2$ cat /etc/resolv.conf
# Generated by NetworkManager search internal.mlnx.com labs.mlnx
nameserver 10.15.2.29
nameserver 10.15.2.16
```

This example shows that the domains are [internal.mlnx.com](#) and labs.mlnx, and the names of the servers are 10.15.2.29 and 10.15.2.16.


2. Set up the DHCP config file accordingly:

```
cat >/etc/dhcp/dhcpd.conf <<EOF
allow booting;
allow bootp;
subnet 192.168.100.0 netmask 255.255.255.0 {
    range 192.168.100.10 192.168.100.20;
    option broadcast-address 192.168.100.255;
    option routers 192.168.100.1;
    option domain-name-servers 10.15.2.29 10.15.2.16;
    # Set the domain search according to the network configuration option domain-search "internal.tilera.com"
    "mtbu.labs.mlnx"; next-server 192.168.100.1;
    filename "/BOOTAA64.EFI";
}
# Specify
the IP address for
this client. host pxe_client {
    hardware ethernet 00:1a:ca:ff:ff:01; fixed-address 192.168.100.2;
} EOF
```

 It is recommended to back up the previous dhcpd.conf file before overwriting it.

Step 5: Set Up the HTTP Server

The TFTP server allows the PXE boot to load the initrd and kernel. The BlueField Controller Card obtains all the other required sources through the network, thus, making it necessary to set up an HTTP.

 Setting up the HTTP server requires the HTTP package. If it is not installed, please install it via “yum install httpd” or “apt-get httpd”, depending on your Linux distribution.

To configure the http server to serve the contents of the installation disk, run the following command:

```
cat >/etc/httpd/conf.d/pxeboot.conf <<EOF Alias /centos7
/mnt
<Directory /mnt>
    Options Indexes FollowSymLinks Require ip 127.0.0.1
    192.168.100.0/24
</Directory>
EOF
systemctl enable httpd systemctl restart
httpd
```

Flashing BlueField Controller Card Bootloader Code

Before installing an OS, flash the bootloader code first. The BlueField Controller Card is shipped with an initial bootloader code, and should be updated with the following instructions.

Opening Terminal Connection to BlueField Controller Card

To open a console window to the BlueField Controller Card, a terminal application is required. The application “minicom” is used for the flow, however, any standard terminal application can work, e.g. “screen”.

 Install minicom by running “yum install minicom” or “apt-get install minicom”.

1. On the host, type “minicom” to open minicom on the current terminal, use “minicom -s” to set it up.
2. Go to the settings menu by pressing “Ctrl-a + o” (the setting menu opens by default when launching with the “-s” option). Navigate to the “Serial port setup” submenu and set the “Serial Device” to the one connected (should be one of the /dev/ttyUSBx if using the serial-UART cable).
3. Change the baud rate to 115200 8N1, and ensure that the hardware and software flow control are set to “No”.

Minicom Settings - Example

```
A - Serial Device      : /dev/ttyUSB1
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?

Screen and keyboard
Save setup as dfl
Save setup as..
Exit
Exit from Minicom
```

4. Select “Save setup as dfl” in order not to have to set it again in the future.

Using Initial Install Bootstream


1. On the host side, ensure that the RShim driver is running:

```
$ systemctl status rshim
```

An RShim device is located under the /dev directory, if you only have one, it should be “rshim0”:

```
[root@bu-lab02 ~]# ls /dev/rshim0/
boot      console net      rshim
```

The boot device is used to push the bootstream to the BlueField Controller Card. Upon writing to it, it automatically resets the Arm cores so that it is booted using the pushed bootstream.

 The console device can be used as a console instead of the serial-USB console. The primary bootloader does not support this device, however, UEFI and Linux support it. In cases where the special UART adapter board is unavailable, this can be used instead.

2. Push the initial install bootstream to the BlueField Controller Card:

```
cat /root/BlueField-1.0.alphaX.XXXXX/sample/install.bfb > \  
/dev/rshim0/boot
```

On the terminal, various boot messages appear until Linux is loaded. This is the Yocto embedded Linux running off the kernel initramfs pushed in the bootstream.


- a. When prompted, type in “root” to get to the command prompt without any password.

Yocto Log

```
[ 18.684305] NET: Registered protocol family 10  
[ 18.694768] Segment Routing with IPv6  
[ 18.926117] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready  
[ 19.082124] NOHZ: local_softirq_pending 08  
  
Poky (Yocto Project Reference Distro) 2.7.4 localhost ttyAMA0  
localhost login:  
root@localhost:~#
```

3. After Linux is loaded, in the terminal, run the /opt/mlnx/scripts/bfrec script to update the bootloader.

Installing CentOS 7.4 on BlueField Controller Card

 If the error “no root is found” appears in the installation process, check or disable the firewall as needed on the server host machine.

Full PXE Boot Installation

1. Get to the UEFI boot menu.
 - a. Reboot the BlueField Controller Card by typing "reboot" on the console. A “UEFI firmware...” message should appear and the screen clears.
 - b. Press ESC several times until you enter the UEFI boot menu.

UEFI Boot Menu

```
Continue  
Select Language          <Standard English>      This selection will  
> Boot Manager                                     direct the system to  
> Device Manager                                     continue to booting  
> Boot Maintenance Manager                             process  
  
^v=Move Highlight      <Enter>=Select Entry
```

- c. On the host, restart the DHCP and TFTP service:

```
systemctl restart dhcpd  
systemctl restart tftp #might be xinetd
```

- d. Navigate to the Boot Manager.
UEFI Boot Manager


```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X                                     Boot Manager                                     X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Boot Option Menu                                Device Path :
CentOS 7.4                                HD(1,GPT,03E5CAD3-CE36
Linux from mmc0                            -4ACE-A999-710E70651E0
EFI Internal Shell                        6,0x800,0x64000)/\EFI\
EFI Misc Device                          centos7.4\grubaa64.efi
EFI Network
EFI Network 1
EFI Network 2
EFI Network 3
EFI Network 4
EFI Network 5

^ and v to change option, ENTER to select an

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X                                     X
X ^v=Move Highlight      <Enter>=Select Entry      Esc=Exit      X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

- e. Select EFI Network, it will then use the TFTP service on the host to discover all available PXE boot options. Shortly after, a “..Fetching Netboot Image” message will appear enabling CentOS installation.
Option to Install CentOS


```

Install centos/7.4 AArch64 - BlueField


Use the ^ and v keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.

```

- f. Select CentOS download.

 This process may take few minutes as it fetches data over the USB network. Running “ifconfig” on the host and monitoring the RX/TX packets on the “tmfifo_net0” network indicates that the fetching data process is not complete.

- g. Follow the installation instructions in the configuration menu. Recommended settings are included.

 These configuration inputs are not needed when the kickstart option “-k” is specified when running the setup.sh script.

```

=====
===== VNC
Text mode provides a limited set of installation options. It does not offer custom partitioning for
full control over the disk layout. Would you like to use VNC mode instead?

1) Start VNC
2) Use text mode

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: 2
=====
Installation:main* 2:shell 3:log 4:storage-l0> Switch tab: Alt+Tab | Help: F1
1) [x] Language settings      2) [!] Time settings
   (English (United States))   (Timezone is not set.)
3) [!] Installation source    4) [!] Software selection (Processing...)      (Processing...)
5) [!] Installation Destination 6) [x] Kdump
   (No disks selected) (Kdump is enabled)
7) [x] Network configuration   8) [!] Root password
   (Wired (eth0) connected)   (Password is not set.)
9) [!] User creation
   (No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin installation | 'r' to refresh]: 2
=====
Time settings Timezone: not set
NTP servers: not configured

1) Set timezone
2) Configure NTP servers
Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: 1
=====
Timezone settings Available regions
Available regions
1) Europe 6) Pacific 10) Arctic
2) Asia 7) Australia 11) US
3) America 8) Atlantic 12) Etc
4) Africa 9) Indian
5) Antarctica

Please select the timezone.
Use numbers or type names directly [b to region list, q to quit]: 11
=====
Timezone settings

Available timezones in region US
1) Alaska 4) Eastern 6) Mountain
2) Arizona 5) Hawaii 7) Pacific
3) Central

Please select the timezone.
Use numbers or type names directly [b to region list, q to quit]: 4
=====
Installation

1) [x] Language settings      2) [x] Time settings (English (United States))      (US/Eastern
   timezone)
3) [x] Installation source    4) [x] Software selection (http://192.168.100.1/centos7)
   (Minimal Install)
5) [!] Installation Destination 6) [x] Kdump
   (No disks selected) (Kdump is enabled)
7) [x] Network configuration   8) [!] Root password
   (Wired (eth0) connected)   (Password is not set.)
9) [!] User creation
   (No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin installation | 'r' to refresh]: 4
=====
Base environment Software selection

Base environment

1) [x] Minimal Install 6) [ ] Server with GUI
2) [ ] Compute Node 7) [ ] GNOME Desktop
3) [ ] Infrastructure Server 8) [ ] KDE Plasma Workspaces
4) [ ] File and Print Server 9) [ ] Development and Creative
5) [ ] Basic Web Server Workstation
Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: 9
=====
Base environment Software selection

1) [ ] Minimal Install 6) [ ] Server with GUI
2) [ ] Compute Node 7) [ ] GNOME Desktop
3) [ ] Infrastructure Server 8) [ ] KDE Plasma Workspaces
4) [ ] File and Print Server 9) [x] Development and Creative
5) [ ] Basic Web Server Workstation
Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c
=====
Installation

1) [x] Language settings      2) [x] Time settings (English (United States))      (US/Eastern
   timezone)
3) [!] Installation source    4) [!] Software selection (Processing...)      (Processing...)
5) [!] Installation Destination 6) [x] Kdump
   (No disks selected) (Kdump is enabled)
7) [x] Network configuration   8) [!] Root password
   (Wired (eth0) connected)   (Password is not set.)

```

```

9) [!] User creation
(No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin installation | 'r' to refresh]: 5
=====
Probing storage...
Installation Destination

[x] 1) : 13.75 GiB (mmcb1k0)

1 disk selected; 13.75 GiB capacity; 1007.5 KiB free ...

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c
=====
Autopartitioning Options

[ ] 1) Replace Existing Linux system(s) [x] 2) Use All Space
[ ] 3) Use Free Space

Installation requires partitioning of your hard drive. Select what space to use for the install
target.

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c
=====
Partition Scheme Options [ ] 1) Standard Partition [ ] 2) Btrfs
[x] 3) LVM
[ ] 4) LVM Thin Provisioning

Select a partition scheme configuration.

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: 1
=====
Partition Scheme Options [x] 1) Standard Partition [ ] 2) Btrfs
[ ] 3) LVM
[ ] 4) LVM Thin Provisioning

Select a partition scheme configuration.

Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c Generating
updated storage configuration
Checking storage configuration...

=====
Installation

1) [x] Language settings 2) [x] Time settings (English (United States)) (US/Eastern
timezone)
3) [x] Installation source 4) [x] Software selection (http://192.168.100.1/centos7)
(Development and Creative
5) [x] Installation Destination Workstation) (Automatic partitioning 6) [x] Kdump
selected) (Kdump is enabled)
7) [x] Network configuration 8) [!] Root password
(Wired (eth0) connected) (Password is not set.)
9) [!] User creation
(No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin installation | 'r' to refresh]: 8
=====
Please select new root password. You will have to type it twice.

Password:
Password (confirm):
=====
Question

The password you have provided is weak: The password fails the dictionary check
- it is based on a dictionary word. Would you like to use it anyway?

Please respond 'yes' or 'no': yes
=====
Installation

1) [x] Language settings 2) [x] Time settings (English (United States)) (US/Eastern
timezone)
3) [x] Installation source 4) [x] Software selection (http://192.168.100.1/centos7)
(Development and Creative
5) [x] Installation Destination Workstation) (Automatic partitioning 6) [x] Kdump
selected) (Kdump is enabled)
7) [x] Network configuration 8) [x] Root password (Wired (eth0) connected) (Password is
set.)
9) [ ] User creation
(No user will be created)
Please make your choice from above ['q' to quit | 'b' to begin installation | 'r' to refresh]: b

```

h. Enter “b” and press “enter” to initiate the installation process.

i. Press “Enter” to reboot into CentOS.

CentOS Installation Completion Screen

```

Installing iwl6000-firmware (1347/1348)
Installing words (1348/1348)
Performing post-installation setup tasks
Installing boot loader
.
Performing post-installation setup tasks
.
Configuring installed system
Writing network configuration
Creating users
Configuring addons
Generating initramfs
Running post-installation scripts
.
    Use of this product is subject to the license agreement found at /usr/sh
Installation complete. Press return to quit
[anaconda] 1:main* 2:shell 3:log 4:storage-l0> Switch tab: Alt+Tab | Help: F1


```

Non-PXE Boot Installation

When the setup script is run with the “-t” option, it generates a nonpxe.bfb file at the directory where the script is run. The directory contains the install kernel and rootfs which are usually loaded by UEFI during the initial PXE boot stage. Thus, if pushing this file, the host TFTP server no longer needs to be used and UEFI would automatically load the install kernel and rootfs from the boot FIFO. Together with the “-k” kickstart option, the host can be configured to initiate non-PXE boot and automatic CentOS installation, as long as the host HTTP and DHCP servers are working. To kick off the installation process, run the following command on the host:

```
cat nonpxe.bfb > /dev/rshim0/boot; sleep 2; systemctl restart dhcpd
```

MLNX_OFED Installation


 This section is relevant to non-Yocto Operating Systems only.

Installing MLNX_OFED on Arm Cores

Prerequisite Packages for Installing MLNX_OFED

- MLNX_OFED installation requires some prerequisite packages to be installed on the system. Currently, CentOS installed on the BlueField Controller Card has a private network to the host via the USB connection, and it can be used to Secure Copy Protocol (SCP) all the required packages. However, it is recommended for the BlueField Controller Card 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 eth0 -A POSTROUTING -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
systemctl restart dhcpd
```

 “eth0” 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 to be applied again after host machine reboots.

- Reset the BlueField Controller Card 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.
/auto/sw_mc_soc_project/distro/rhel/kernel-devel-4.11.0-22.el7a.aarch64.rpm

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
```

Installing MLNX_OFED on BlueField Controller Card

1. Copy the MLNX_OFED image to the BlueField Controller Card via the USB network. The MLNX_OFED images should be provided in the software drop:

```
scp MLNX_OFED_LINUX-4.2-1.4.8.0-rhel7.4alternate-aarch64.iso \
root@192.168.100.2:/root
```

2. Mount the image on the BlueField Controller Card:

```
mount /root/MLNX_OFED_LINUX-4.2-1.4.8.0-rhel7.4alternate-aarch64.iso /mnt
```

3. Install MLNX_OFED.
If the kernel on the BlueField is 4.11.0-22.el7a.aarch64, run:

```
cd /mnt
# ./mlnxofedinstall --bluefield
```

If the kernel is different than 4.11.0-22.el7a.aarch64, run:

```
cd /mnt
# ./mlnxofedinstall --add-kernel-support --skip-repo
```

⚠ For OFED to support DPDK, use the arguments “--upstream-libs” and “--dpdk”.

```
./mlnxofedinstall --distro rhel7.4alternate --add-kernel-support --upstream-libs
--dpdk
```

This step might take longer than expected to be completed. If you are using a different package than the required one, run “yum install”.

⚠ 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.

4. Make sure that mlnx_snap.service is down. Run:

```
$ systemctl stop mlnx_snap.service
```

5. Restart openibd:

```
/etc/init.d/openibd restart
```

Updating BlueField Controller Card Firmware

⚠ The below commands apply to MBF1M616A-CSNAT. The commands vary per OPN.

The below steps demonstrate how to manually update the firmware if the automatic process fails. The firmware image can be found in the BlueField Software package.

1. Copy the firmware image to the BlueField Arm:

```
scp fw-BlueField-rel--XX_XX_XXXX-MBF1M6X6A-CSNA_Ax.ini.bin \
root@192.168.100.2:/root
[root@localhost ~]# mst start
Starting MST (Mellanox Software Tools) driver set Loading MST PCI module - Success
Loading MST PCI configuration module - Success Create devices
Unloading MST PCI module (unused) - Success [root@localhost ~]# mst status
MST modules:
-----
MST PCI module is not loaded
MST PCI configuration module loaded
MST devices:
-----
/dev/mst/mt41682_pciconf0
- PCI configuration cycles access.
domain:bus:dev.fn=0000:04:00.0 addr.reg=88 data.reg=92
Chip revision is: 00
```

2. The output indicates that the device is “/dev/mst/mt41682_pciconf0”. To update the firmware:

```
flint -d /dev/mst/mt41682_pciconf0 b \
-i /root/ fw-BlueField-rel-XX_XX_XXXX-MBF1M6X6A-CSNA_Ax.ini.bin
```

When using the mlx and ini files, use the following command instead:


```
mlxburn -d /dev/mst/mt41682_pciconf0 -fw fw-BlueField.mlx -c bf.ini
```

To burn the firmware which comes with OFED after OFED is installed, run:

```
/opt/mellanox/mlnx-fw-updater/firmware/mlxfwmanager_sriov_dis -force
```

3. Power cycle the BlueField Controller Card for the new firmware to take effect.

```
root@bluefield:~# flint -i /opt/fw-Bluefield-red-18_99_4608_MBF1M6X6A-CSNA_Ax-  
Flexboot-3.5.404_UEFI-14.15.20.bin -d /dev/mst/mt41682_pciconf0 b  
  
Current FW version flash: 18.24.0013  
New FW version: 18.99.4608  
  
Burning FW image without signatures - 55%  
Burning FW image without signatures - OK  
Restoring signature  
-I- To load new FW run mlxfwreset or reboot machine.  
root@bluefield:~#
```

 After MLNX_OFED is installed on the Arm cores, use the mlx5_core driver to use the two Ethernet ports on the BlueField Controller Card. If the Ethernet ports on the BlueField Controller Card are connected to the network, there is no need to bridge the host via RShim net to access the network.

To install the kernel modules, please follow the instruction in section [RShim Host Driver](#).

Windows Support

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](#).

RShim Drivers

RShim drivers provide functionalities like resetting the Arm cores, pushing a bootstream image, as well as some networking and console functionalities. For more information on RShim driver usage, please refer to [WinOF-2 Documentation](#) > Features Overview and Configuration > RShim Drivers and Usage.

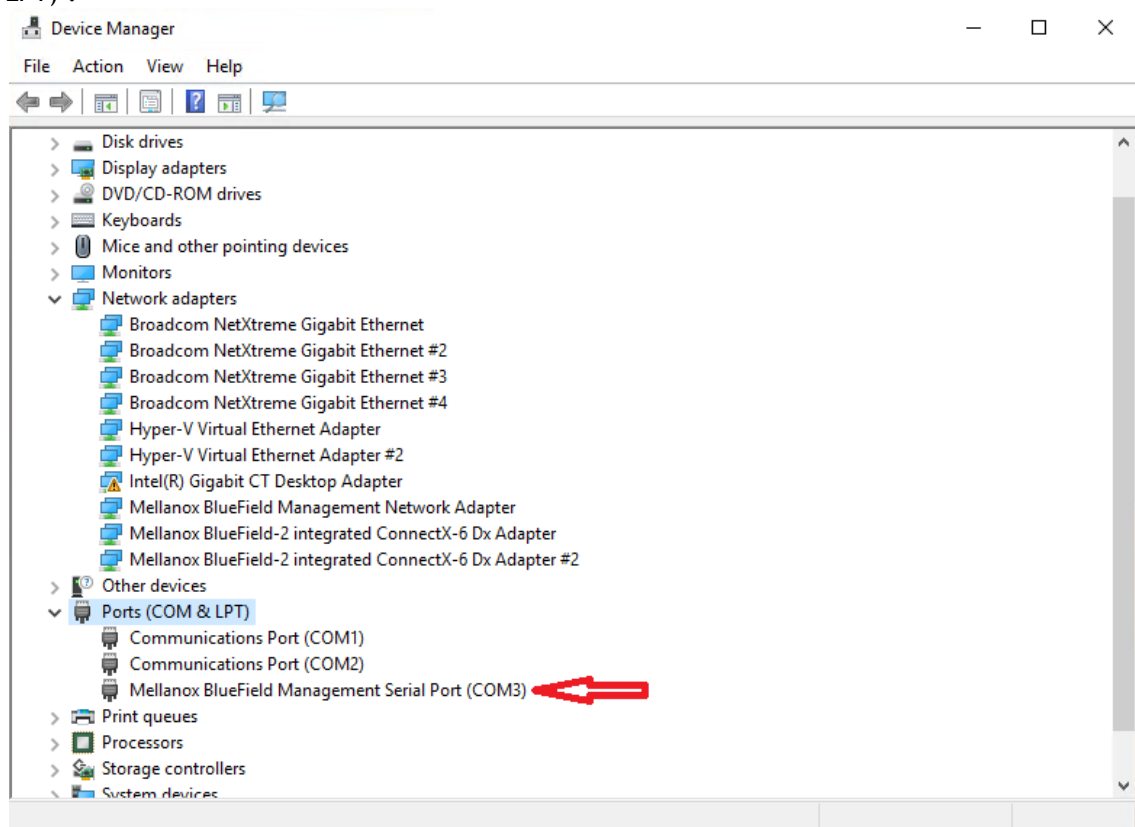
Accessing BlueField DPU From Host

The BlueField DPU can be accessed via PuTTY or any other network utility application to communicate via virtual COM or virtual Ethernet adapter. To use COM:

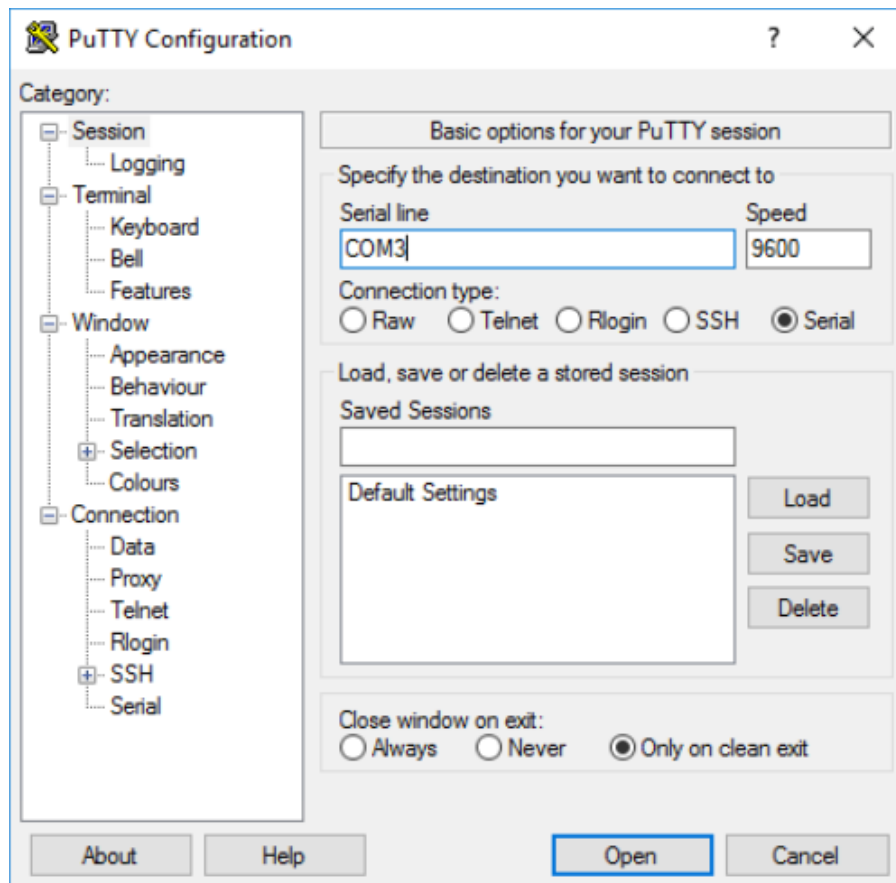
1. Open Putty.
2. Change connection type to Serial.
3. Run the following command in order to know what to set the "Serial line" field to:

```
C:\Users\username\Desktop> reg query HKLM\HARDWARE\DEVICEMAP\SERIALCOMM | findstr MlxRshim
\MlxRshim\COM3          REG-SZ          COM3
```

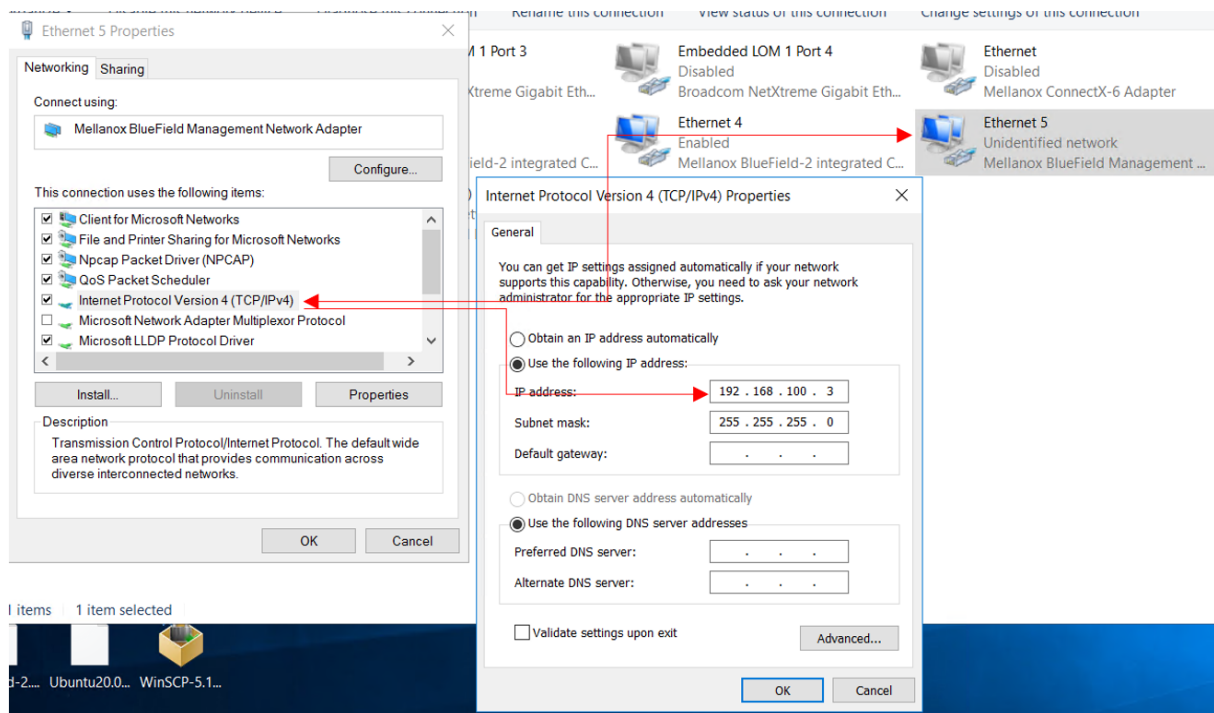

In this case use COM3. This name can also be found via Device Manager under "Ports (Com & LPT)".



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.



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) |

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.

Troubleshooting

- [RShim Troubleshooting](#)
- [Connectivity Troubleshooting](#)
- [Performance Troubleshooting](#)
- [PCIe Troubleshooting](#)
- [SR-IOV Troubleshooting](#)
- [eSwitch Troubleshooting](#)
- [Isolated Mode Troubleshooting](#)
- [General Troubleshooting](#)
- [Installation Troubleshooting](#)

RShim Troubleshooting

RShim driver is not loading

1. Download the [suitable DEB/RPM](#) for RShim (management interface for BlueField from the host) driver.
2. Reinstall RShim packaeg.
 - For Ubuntu/Debian, run:

```
sudo dpkg --force-all -i rshim-<version>.deb
```

- For RHEL/CentOS, run:

```
sudo rpm -Uvh rshim-<version>.rpm
```

3. Restart RShim service. Run:

```
sudo systemctl restart rshim
```

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

```
sudo systemctl status rshim
```

4. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc
DISPLAY_LEVEL 0 (0:basic, 1:advanced, 2:log)
BOOT_MODE      1 (0:rshim, 1:emmc, 2:emmc-boot-swap)
BOOT_TIMEOUT   300 (seconds)
SW_RESET       0 (1: reset)
DEV_NAME       pcie-04:00.2 (ro)
PEER_MAC       00:1a:ca:ff:ff:01 (rw)
PXE_ID         0x00000000 (rw)
VLAN_ID        0 0 (rw)
```

For more information, please refer to section "[Building and Installing RShim Host Driver](#)".

HowTo upgrade the host RShim driver

For more information, please refer to section "[RShim Host Driver](#)".

Connectivity Troubleshooting


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?          |
+-----+
```

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. Power cycle on the host server.
2. If the problem persists, please reset nvconfig (`sudo mlxconfig -d /dev/mst/<device> -y reset`), and then power cycle the host.

 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
```

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.

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 pflsf0
            Interface pflsf0
        Port pflhpf
            Interface pflhpf
        Port ovsbr2
            Interface ovsbr2
                type: internal
    ovs_version: "2.14.1"
```

If no bridge configuration exists, please refer to "[Virtual Switch on BlueField DPU](#)".

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.

Performance Troubleshooting

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
```


PCIe Troubleshooting

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.

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)
```

HowTo handle two BlueField DPU devices in the same server

Please refer to section "[Multi-board Management Example](#)".

SR-IOV Troubleshooting

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)
```

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 <bridge_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 pflsf0
    Interface pflsf0
  Port p1
    Interface p1
  Port pflhpf
    Interface pflhpf
ovs_version: "2.14.1"
```

eSwitch Troubleshooting

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 pflvf0
```

If VFs are present, 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
```

- No mdev/devices are configured. Delete the two SFs created on the BlueField device [upon boot](#) (one per port if the port is in switchdev mode). Run:

```
# sudo mlnx-sf -a remove --uuid <UUID1>
# sudo mlnx-sf -a remove --uuid <UUID2>
```

⚠ You may retrieve the UUID (unique device ID) information by running the following command:

```
# /opt/mellanox/iproute2/sbin/devlink dev show
pci/0000:03:00.0
pci/0000:03:00.1
mdev/0214613e-60a3-4437-8de6-efbb41d8436e
mdev/d3116441-04fb-4ea1-a2b3-63fb739e0f4a
```

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
```

Isolated Mode Troubleshooting

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.

HowTo upgrade ConnectX firmware from Arm side

Please refer to "[HowTo upgrade ConnectX firmware from Arm side](#)".

General Troubleshooting

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
- Power cycle the server

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
- Power cycle the server

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

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

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

Installation Troubleshooting

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](#)".

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.

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.

How to find the software versions of the running system

Run the following:

```
/opt/mellanox/scripts/bfvcheck:  
root@bluefield:/opt/mellanox/scripts# ./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
```

How to upgrade the host RShim driver

See section "[RShim Host Driver](#)" or <BF_INST_DIR>/src/drivers/rshim/README.

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
```


How to upgrade ConnectX firmware from Arm side

The `mst`, `mlxburn`, and `flint` tools can be used to update firmware.

For Yocto

The default firmware images are under `/lib/firmware/mellanox/`. Run the following command from the Arm side:


```
/lib/firmware/mellanox/mlxfwmanager_sriov_dis_aarch64_4168<6|2>
```

 The file `mlxfwmanager_sriov_dis_aarch64_41686` is intended for BlueField-2.
The file `mlxfwmanager_sriov_dis_aarch64_41682` is intended for BlueField.

For Ubuntu, CentOS and Debian

Upgrade BlueField DPU's firmware. Run the following command from the Arm side:

```
sudo /opt/mellanox/mlnx-fw-updater/firmware/mlxfwmanager_sriov_dis_aarch64_4168<6|2>
```

 The file `mlxfwmanager_sriov_dis_aarch64_41686` is intended for BlueField-2.
The file `mlxfwmanager_sriov_dis_aarch64_41682` is intended for BlueField.

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 SmartNICs) 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](#).

How to use the UEFI boot menu

Press the "ESC" key after booting 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-2 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
```

How to get installation images from the BlueField release tarball

The BlueField software tarball is released as BlueField-<ver>.<num>.tar.xz (e.g. BlueField-1.2.0.10639.tar.xz). In this FAQ section, it is assumed that the tarball is uncompressed under directory "<BF_INST_DIR>".

How to install Yocto

See section "[Installing Reference Yocto Distribution](#)" or <BF_INST_DIR>/sample/README.install.

How to install CentOS

See section "[Installing Official CentOS 7.x Distribution](#)" or <BF_INST_DIR>/distro/rhel/pxeboot/README.

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
```

In order 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
```

If you want to debug incidents that occur at boot time, that has to be configured through the kernel boot parameters. 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.

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 Reference Yocto Distribution](#)")
- In BlueField Linux, create a file with one line like "SYS_ENABLE_SMMU=FALSE", 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. In order to modify the configuration, you may either set it specifically or use the "Reset EFI Variables" option in the [UEFI System Configuration](#).

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-mkbbfb --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-mkbbfb --boot-args bootarg \
  <BF_INST_DIR>/sample/install.bfb
```

On RShim:

```
$ echo "console=hvc0 initrd=initramfs" > bootarg
$ <BF_INST_DIR>/bin/mlx-mkbbfb --boot-args bootarg \
  <BF_INST_DIR>/sample/install.bfb
```

Document Revision History

Rev 3.5.1 - February 05, 2021

Updated:

- Page "[Troubleshooting](#)"

Rev 3.5 - January 25, 2021

Added:

- Section "[Retrieving Data from BlueField Via OOB/ConnectX Interfaces](#)"
- Section "[Multi-board Management Example](#)"
- Section "[BFB Installation Utility Script](#)"
- Section "[Upgrading NVIDIA BlueField DPU Software](#)"
- Warning regarding default IPMI behavior under "[Intelligent Platform Management Interface \(IPMI\)](#)"
- Section "[IPsec Full Offload strongSwan Support](#)"
- Section "[Destroying IPsec Configuration](#)"
- Page "[Public Key Acceleration](#)"
- Page "[Troubleshooting](#)"

Updated:

- List of "[Related Documentation](#)"
- Section "[Installing Reference Yocto Distribution](#)"
- Section "[Installing Reference Yocto Distribution](#)"
- Section "[Installing Debian on BlueField](#)"
- Section "[BlueField Retrieving Data From BMC Via IPMB](#)"
- Section "[List of IPMI Supported Sensors](#)"
- Section "[Enabling OVS HW Offloading](#)"
- Section "[Configuring VXLAN Tunnel](#)"
- Page "[VirtIO-net Emulated Devices](#)"

Rev 3.1.0 - September 17, 2020

Added:

- Section "[BlueField-2 OOB Ethernet Interface](#)"
- Section "[OOB PXE Boot](#)"
- Section "[OOB PXE Boot Over VLAN](#)"
- Page "[Multi-host](#)"
- Section "[Enabling OVS-DPDK Hardware Offload](#)"
- Section "[Configuring DPDK and Running TestPMD](#)"
- Page "[RegEx Acceleration](#)"
- Page "[IPsec Functionality](#)"
- Page "[VirtIO-net Emulated Devices](#)"
- Section "[Check RShim Device](#)"
- [Example for enabling SMMU](#) under "[Installing Reference Yocto Distribution](#)"
- Section "[Installing Debian on BlueField](#)"
- Added note at the end of "[BlueField Link Aggregation](#)" page

Updated:

- Section "[OpenOCD on BlueField](#)"
- Section "[Building Your Own BFB Installation Image](#)"
- First code box under "[Mediated Devices](#)"
- Section "[Ubuntu Hardware and Software Requirements](#)"
- Section "[Installation Procedure for Canonical's Ubuntu BFB](#)"
- Section "[Separate Host Mode](#)"
- Section "[Network Interfaces](#)"
- Section "[Installing Reference Yocto Distribution](#)"
- Section "[BFB File Overview](#)"
- Section "[Updating Boot Partition](#)"
- Section "[Building and Installing RShim Host Driver](#)"
- Section "[Device Files](#)"
- Step 1.a. under section "[Post-installation](#)"
- Step 3 under section "[PXE Boot Over VLAN](#)"
- Step 2 under section "[Installing Reference Yocto Distribution](#)"
- Section "[Installation Procedure for Canonical's Ubuntu BFB](#)"
- Section "[RShim Logging](#)"
- Step 2 and 3 under section "[Setup Procedure Without Installation Script](#)"
- Section "[Using Initial Install Bootstream](#)"
- Section "[Connection Tracking Offload](#)"
- Section "[Connection Tracking Aging](#)"
- First two lines in code block under "[Mediated Devices](#)"

Rev 2.5.1 - April 22, 2020

Updated:

- Figure in section "[System Connections](#)"
- Section "[List of IPMI Supported FRUs](#)"
- Section "[Loading and Using IPMI on BlueField Running CentOS](#)"
- Section "[BlueField Retrieving Data From BMC Via IPMB](#)"
- Section "[Changing I²C Addresses](#)"
- Section "[Separated Host](#)"
- Bullet `/dev/rshim<N>/misc`
- Step 1 and 2 under [updating MAC address from the server host side while Linux is running](#)

Rev 2.5.0 - March 03, 2020

Added:

- Section "[CentOS Partitioning on BlueField Device](#)"
- Page "[RShim Logging](#)"
- New step for using the "`mlnx-sf`" tool under "[Configuring Mediated Device](#)"
- Section "[Connection Tracking With NAT](#)"

Updated:

- Page "[Mediated Devices](#)"
- Section "[System Consoles](#)"
- Section "[BMC Retrieving Data from BlueField via IPMB](#)"
- Section "[List of IPMI Supported FRUs](#)"
- Section "[Supported IPMI Commands](#)"

- Section "[Loading and Using IPMI on BlueField Running CentOS](#)"
- Section "[BlueField Retrieving Data From BMC Via IPMB](#)"
- Section "[Updating SmartNIC Firmware](#)"
- Section "[Geneve Tunneling Offload](#)"

Rev 2.4.0 - December 13, 2019

Added:

- Note under step 4 of section "[Installing the Reference Yocto Distribution](#)"
- Step under section "[Configuring Connection Tracking Offload](#)"
- Section "[Performance Tune Based on Traffic Pattern](#)"
- Section "[GRE Tunneling Offload](#)"
- Page "[QoS Configuration](#)"

Updated:

- Step 3 and step 4 note under "[Installing the Reference Yocto Distribution](#)"
- Note under Step 1 under "[Loading and Using IPMI on BlueField Running CentOS](#)"
- Step 10 under "[Loading and Using IPMI on BlueField Running CentOS](#)"
- Steps 4 and 5 under "[Configuring Connection Tracking Offload](#)"
- Section "[RDMA Support on Host](#)"

Rev 2.2.0 - September 29, 2019

Added:

- Section "[Retrieving Data from BMC by BlueField](#)"
- Note under section "[Connection Tracking Aging](#)"

Updated:

- Section "[Updating Boot Partition](#)"
- [Note](#) under section "Retrieving Data from BlueField by BMC"
- Section "[List of IPMI Supported FRUs](#)"
- Section "[Loading and Using IPMI on BlueField Running CentOS](#)"
- Step 1 of section "[Configuring Mediated Device](#)"
- Page "[RDMA Stack Support on Host and Arm System](#)"

Restructured page "[Intelligent Platform Management Interface \(IPMI\)](#)"

Rev 2.1.0 - July 29, 2019

Added:

- Section "[Installing Ubuntu on BlueField](#)"
- Section "[Loading and Using IPMI on BlueField Running CentOS](#)"
- Section "[Connection Tracking Offload](#)"
- Page "[Configuring Uplink MTU](#)"
- Step under "[BlueField LAG Configuration](#)"
- Page "[Mediated Devices](#)"
- Section "[Setting Host PF and VF Link State](#)"
- Section "[DPDK on BlueField SmartNIC](#)"
- Section "[NVMe SNAP on BlueField SmartNIC](#)"

Updated:

- Section "[Updating Boot Partition](#)"
- Section "[mlxbf-bootctl](#)"
- Section "[Changing the Linux Kernel or Root File System](#)"
- Note in section "[Installing CentOS With BFB Installation Image](#)"
- Section "[Host Machine Setup](#)"
- Step 3.c. under section "[Post-installation](#)"
- Section "[Intelligent Platform Management Interfaces \(IPMI\)](#)"
- Command "Read FRU data" in section "[Supported IPMI Commands](#)"
- Command "Set sensor threshold" in section "[Supported IPMI Commands](#)"
- Section "[Updating SmartNIC Firmware](#)"
- Section "[Installing MLNX_OFED on the SmartNIC](#)"
- Section "[SmartNIC Modes of Operation](#)"
- Section "[VXLAN Tunneling Offload](#)"
- Section "[Querying OVS VXLAN hw_offload Rules](#)"
- Section "[Query Configuration](#)"

Release Notes Change Log History

Changes and New Features in 3.5.0.11563

- Changed default SmartNIC configuration to [embedded function mode](#)
 - OVS is configured to allow traffic to and from the host by default
- [Running RDMA applications is enabled by default](#) both from the DPU and the host
 - [UCX](#) is included as part of the DPU SW image
- Improved image [installation script](#)
- Added support for [Geneve tunnel HW offload](#) for OVS-DPDK
- Added GA-level support for [IPsec full offload](#)
 - Integration of [StrongSwan](#) with [IPSec HW offload](#)
 - Integration of StrongSwan with [with OpenSSL and HW Public Key Acceleration engine](#)
- Added support for [VirtIO-net device emulation](#)
 - HotPlug support for VirtIO-blk and VirtIO net devices
 - Up to 127 VirtIO virtual functions support
- Added beta-level support for [RegEx acceleration](#)
- Added alpha-level support for [deep packet inspection](#)
- Added support for CentOS 8.2 and Debian 10 as BlueField DPU OS

Changes and New Features in 3.1.0.11424

General Changes

- Added GA software support for [platforms](#) based on [BlueField-2 DPU](#)
- Added a new [out-of-band 1G interface](#)
- Added a new [host-side RShim user space interface](#)
- Ubuntu 20.04 as default OS
- Kernel version 5.4 based release
- Debian 10 support

SmartNIC Changes

- Added support for [OVS-DPDK hardware offload](#)
- Added support for [VirtIO acceleration through hardware vDPA](#)
- Added beta support for [VirtIO full hardware emulation](#)
- Added support for [IPSec crypto offload](#)
- Added support for IPsec offload for RDMA traffic
- Added support for [IPsec Full Offload](#)
- Added support for [Regular Expression Acceleration](#)
- Added SmartNIC support for [multi-host configuration](#)
- Integrated DPDK 20.08

Changes and New Features in 2.5.1.11213

SmartNIC Changes

- Added ability to create [one sub-function per port](#) on the BlueField device upon boot
- Improved offload rules insertion rate for OVS
- Increased number of offloaded rules for OVS offload

Changes and New Features in 2.5.0.11176

General Changes

- Added support for [burning Device Unique Key \(MDK\) EFUSE bits](#) to enable Secure Boot
- Added the option to [control disk partitioning](#) for CentOS images
- Added NVMe SNAP support for Ubuntu-based images
- Added PCIe Gen4 support for [select boards](#)
- BMC error logging using IPMI

SmartNIC Changes

- Included openvswitch-2.12.1 as part of SmartNIC installation
- Added beta-level support for [OVS-DPDK with HW offload](#), including connection tracking offload
- Reduced memory consumption by SPDK
- Added support for [Geneve tunneling offload](#), including connection tracking offload

Added ability to create [one sub-function per port](#) on the BlueField device upon boot

Changes and New Features in 2.4.0.11082

General Changes

- Added NVMe SNAP support on Yocto OS

SmartNIC Changes

- Updated support for GRE tunnel encapsulation/decapsulation
- Merged both e-switches: All functions now have the same `phys_switch_id`. Going forward, individual ports must be identified by `phys_switch_id` and `phys_port_name`.
- Added support for DPDK 19.11 with HW offloads

Changes and New Features in 2.2.0.11000

General Changes

- Added support for Ubuntu 18.04 image to run on the SoC

SmartNIC Changes

- NVMe SNAP is GA on supported devices (requires kernel v4.20 to be installed on the BlueField SoC)
- Added support for interoperability of NVMe SNAP with OVS offload and hardware LAG
- Added support for DPDK 19.08
- Added support for connection tracking offload for VXLAN traffic
- Added support for connection tracking offload for dual-port devices
- Added support for connection tracking offload for dual-port devices using LAG

Changes and New Features in 2.1.0.10924

General Changes

- Added driver support for CentOS-7.6 running on the the SoC

SmartNIC Changes

- Added Beta-level support for [Connection Tracking Offload](#)
- Added support for DPDK 19.05 with HW offloads
 - The DPDK code is upstream and should be downloaded from [DPDK.org](https://dpdk.org)
 - The HW offloads are described in the mlx5 section of [DPDK.org](https://dpdk.org)
- Added Beta-level support for NVME SNAP
- Added support for RDMA in [embedded CPU function ownership mode \(SmartNIC mode\)](#) using [mediated devices](#)

Changes and New Features in 2.0.1.10841

General Changes

- The Yocto/Poky SDK has been upgraded to version 2.5.1 (“sumo”)
- Secure boot can now be enabled by users. Contact your sales/support team for assistance and documentation.
- New [persistent partition](#) (“/data”) is now created when installing using the provided Yocto BlueField® install.bfb image. This partition is preserved across software updates and users may also enable an overlay FS using /data to configure system specific data in /etc that will also be preserved across software updates. See HOWTO-persistence in the Documentation directory.

- The eMMC boot partitions can now be updated from UEFI using capsule support. See HOWTO-capsule in the Documentation directory.
- A new source RPM is now provided to allow customers to use [OpenIPMI BlueField \(IPMB\)](#) support in CentOS

SmartNIC Changes

- Added support for [configuring default MAC](#) for host side physical and virtual functions
- Added support for [rate limiting](#) for host side rate physical and virtual functions
- The uplink and host PF and VF representors have been renamed from:
 - rep[0,1]-ffff → p[0,1], rep[0,1]-0 → pf[0,1]hpf
 - rep[0,1]-[1..numvfs - 1] → pf[0,1]vf[0..numvfs -1]
- The non-function ECPF netdevices (ensp3f<i>)</i> are no longer created
- Included openvswitch-2.11.1 as part of SmartNIC installation
- DPDK based forwarding to the host. Please contact Mellanox Support for DPDK instructions.

Changes and New Features in 1.2.0.10639

General Changes

- The BlueField Yocto build instructions have been updated. A new script bluefield-init-build-env has been added to simplify the build process. See the updated README-bluefield file for details.
- Added SNMP support for a BlueField specific MIB (MLNX-PMC-MIB.txt). Low level BlueField performance registers can be queried using this feature. (Not supported in Beta1 but will be in GA.)

BlueField Reference Platform Changes

- A new PMC driver (mlx-pmc) allows for monitoring BlueField specific performance monitoring counters via linux sysfs
- Added support for the Yocto extensible SDK
- Added support for a new root file system (initramfs-netboot) for BlueField Yocto. This is a small root file system intended to be used to NFS mount a disk during boot and then boot from that NFS mounted disk. Dracut support has also been added to Yocto.
- Added support for PXE boot/install over a VLAN interface
- Added ability for BlueField UEFI to IPMI boot override via the SMBus
- ATF changes of interest to users developing ATF code for their proprietary systems. These changes are only visible to developers, not users of ATF.
- ATF LOAD_IMAGE_V2 is now the default for BlueField ATF
- GICv3 is now used as default as well
- For users not running Yocto, added new source RPMs for the mlx-l3cache and mlx-trio drivers
- During system boot BlueField now verifies that the HCA firmware, ATF, and UEFI software versions match the current BlueField OFED software version. If the versions do not match a warning message is issued on the console.
- The install BFB procedure has changed. Previously, there was a separate install BFB for each platform. As of now, users should use “install.bfb” for all BlueField systems. The install BFB process now reads the PSID of the system and configures the hardware based on that value.

SmartNIC Changes

- Added ability to set the NIC PF MAC address by running the following command from the Arm side (on the corresponding port interface):

```
echo "aa:bb:cc:dd:ee:ff" > /sys/class/net/eth0/smart_nic/pf/mac
```

- Setting host PF rate limit from Arm side (by running “echo 1000 > /sys/class/net/eth0/smart_nic/pf/max_tx_rate”) now limits the transmission rate of the PF to 1000Mbps
- Added support for 64 VFs on the host

Bug Fixes History

| Ref # | Issue Description |
|---------|---|
| 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 |
| 2297780 | Description: Performing mlxfwreset with PCIe and USB RShim connected to the same BlueField SmartNIC may lead the PCIe host server to crash. |
| | Keywords: RShim; mlxfwreset |
| | 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 |
| | 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 |
| 2249341 | Description: If VirtIO drivers (virtio, virtio_net, virtio_pci) are loaded, hot plugin of a virtio_net emulation device hangs the kernel. |
| | Keywords: VirtIO emulation |
| | Fixed in version: 3.1.0.11424 |
| 2246826 | Description: Openibd restart returns error in multi-host SmartNIC. This error may be safely ignored. |
| | Keywords: Openibd; SmartNIC error |
| | Fixed in version: 3.1.0.11424 |
| 2004375 | Description: If an emulated NVMe device is being passed through to a virtual machine, multiple consecutive VM reboots could lead to the emulated device malfunctioning. |
| | Keywords: NVMe SNAP |
| | Fixed in version: 2.5.0.11176 |
| 1921748 | Description: Entering the SmartNIC UEFI boot menu while mlx5 drivers are being loaded on the host might cause the host to hang upon the next reload of the mlx5 drivers. A server power cycle is required to recover from this situation. |
| | Keywords: SmartNIC; UEFI; Boot menu; mlx5; hang |
| | Fixed in version: 2.5.0.11176 |
| 1919704 | Description: The targetcli recipe found in the Yocto meta-bluefield layer fails because it cannot find the source RPM file specified by the SRC_URI variable. |

| Ref # | Issue Description |
|---------|---|
| | Keywords: Yocto; Linux |
| | Fixed in version: 2.5.0.11176 |
| 1980929 | Description: Offloading header rewrite operations while connection tracking offload is enabled could result in kernel panic. |
| | Keywords: SmartNIC; connection tracking |
| | Fixed in version: 2.4.0.11082 |
| 1973809 | Description: DCB is not enabled on SmartNIC kernel, blocking quality of service configuration. |
| | Keywords: QoS; RoCE |
| | Fixed in version: 2.4.0.11082 |
| 1946660 | Description: High rate of insertion and removal of offloaded rules could result in flow counters allocation failure. |
| | Keywords: Flow counters |
| | Fixed in version: 2.4.0.11082 |
| 1919435 | Description: PXE boot issue occurs when in priv limited mode. |
| | Keywords: SmartNIC; PXE boot |
| | Fixed in version: 2.4.0.11082 |
| 1912887 | Description: CREATE_FLOW_GROUP failed errors are seen in dmaesg during Arm boot up. |
| | Keywords: SmartNIC; switch offload |
| | Fixed in version: 2.4.0.11082 |
| 1929882 | Description: Setting disable_mc_local_lb in vPort context is not functioning properly. |
| | Keywords: SmartNIC |
| | Fixed in version: 2.4.0.11082 |
| 1844417 | Description: Crash occurs while running connection tracking offload under heavy stress. |
| | Keywords: SmartNIC; connection tracking |
| | Fixed in version: 2.2.0.10981 |
| 1813664 | Description: HW LAG is not supported when NVMe SNAP is enabled. |
| | Keywords: LAG; NVMe |
| | Fixed in version: 2.2.0.10981 |
| 1816404 | Description: Rebooting the SmartNIC while the mlx5 drivers are running on the Arm cores could cause the SoC to hang. Cold boot is required in order to recover. |
| | Keywords: SmartNIC; driver |
| | Fixed in version: 2.2.0.10981 |
| 1835026 | Description: The command "yum install mlnx-ofed-all" fails on CentOS7.xALT due to conflicts. |
| | Keywords: Installation |
| | Fixed in version: 2.2.0.10981 |
| 1753155 | Description: RShim Livefish driver could crash when using "Ctrl+c" to stop the bfb push. |
| | Keywords: RShim |

| Ref # | Issue Description |
|---------|--|
| | Fixed in version: 2.2.0.10981 |
| 1727855 | Description: Building the Poky reference image using bitbake is not functional. MFT images are missing. |
| | Keywords: Yocto; Poky; MFT; bitbake |
| | Fixed in version: 2.1.0.10924 |
| 1342296 | Description: It is not possible to boot CentOS Linux after exiting EFI shell; system hangs or an exception is seen. |
| | Keywords: Boot; EFI Shell; CentOS |
| | Fixed in version: 2.1.0.10924 |
| 1753084 | Description: After booting install.bfb, running the bfinst script and rebooting, BlueField® cannot boot and enters UEFI menu. |
| | Keywords: UEFI; bfrec |
| | Fixed in version: 2.0.1.10841 |
| 1555799 | Description: The SmartNIC may fail to upgrade FW in Livefish mode from the x86 host. |
| | Keywords: SmartNIC; Livefish; FW upgrade; x86 host |
| | Fixed in version: 2.0.1.10841 |
| 1486730 | Description: You cannot offload more than 54,000 rules with the same user priority simultaneously. |
| | Keywords: vSwitch offload |
| | Fixed in version: 2.0.1.10841 |
| 1507451 | Description: On the SmartNIC, the maximum supported VFs per port is 64. |
| | Keywords: SmartNIC; Virtual Function |
| | Fixed in version: 2.0.1.10841 |
| 1262255 | Description: The “mlxfwreset” utility is currently not functional. |
| 1517550 | Keyword: MFT |
| | Fixed in version: 2.0.1.10841 |
| 1472859 | Description: When OVS is running on the Arm side, after performing a power cycle or “mlxfwreset”, PXE boot from the host fails. |
| | Keywords: Host; Open vSwitch; PXE |
| | Fixed in version: 2.0.1.10841 |
| 1525939 | Description: When KEEP_LINK_STATE_UP_P1/P2 is true, the netdevice will be up, and ibstat shows the physical state as LinkUp. KEEP_LINK_STATE_UP is true by default on BlueField. |
| | Keywords: Link; Interfaces |
| | Fixed in version: 2.0.1.10841 |
| --- | Description: A patch for Arm errata 859971 is now enabled in the BlueField ATF code. |
| | Keywords: Arm |
| | Fixed in version: 1.2.0.10639 |

| Ref # | Issue Description |
|---------|--|
| 1485765 | Description: Linux CMA size is non-zero by default. |
| | Keywords: Linux |
| | Fixed in version: 1.2.0.10639 |
| 1504789 | Description: RShim PCIe driver intermittently fails. |
| | Keywords: RShim; PCIe |
| | Fixed in version: 1.2.0.10639 |
| 1505453 | Description: The driver on the SmartNIC Arm must be loaded before the driver on the host side. |
| | Keywords: Driver; InfiniBand; SmartNIC |
| | Fixed in version: 1.2.0.10639 |
| 1516216 | Description: The kexec utility does not work on BlueField. |
| | Keywords: Linux |
| | Fixed in version: 1.2.0.10639 |
| 1518482 | Description: Yocto system-networkd does not support IPoB. |
| | Keywords: InfiniBand; Boot; udev |
| | Fixed in version: 1.2.0.10639 |
| 1522054 | Description: Newer GCC Armv8 A72 optimization options are not enabled. |
| | Keywords: Arm |
| | Fixed in version: 1.2.0.10639 |
| 1523606 | Description: Sporadic failures observed when decode-dimms attempts to read SPD information. |
| | Keywords: I2C; SPD |
| | Fixed in version: 1.2.0.10639 |
| 1530541 | Description: Kernel module rshim_pcie_lf hung on SmartNIC setup when in LiveFish mode. |
| | Keywords: LiveFish; SmartNIC |
| | Fixed in version: 1.2.0.10639 |
| 1526880 | Description: UEFI hangs during boot a low percentage of the time |
| | Keywords: UEFI |
| | Fixed in version: 1.2.0.10639 |

| Ref # | Issue Description |
|---------|--|
| 1537870 | Description: L3 cache hit rate seen close to 0%. |
| | Keywords: L3 |
| | Fixed in version: 1.2.0.10639 |
| 1547521 | Description: TMFIFO console output slow |
| | Keywords: General |
| | Fixed in version: 1.2.0.10639 |
| 1553409 | Description: System crash occurs when loading IPMI ssif driver. |
| | Keywords: Linux |
| | Fixed in version: 1.2.0.10639 |
| 1564052 | Description: Error seen repeatedly. A TRIO counter wraps causing a spurious error message to be seen on the console. |
| | Keywords: Linux |
| | Fixed in version: 1.2.0.10639 |

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 NON-INFRINGEMENT, 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, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of NVIDIA Corporation and/or Mellanox Technologies Ltd. in the U.S. and in other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2021 NVIDIA Corporation & affiliates. All Rights Reserved.

