Mellanox BlueField Software User Manual

Rev 1.1

Version 1.0.0.10521

NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT ("PRODUCT(S)") AND ITS RELATED DOCUMENTATION ARE PROVIDED BY MELLANOX TECHNOLOGIES "AS-IS" WITH ALL FAULTS OF ANY KIND AND SOLELY FOR THE PURPOSE OF AIDING THE CUSTOMER IN TESTING APPLICATIONS THAT USE THE PRODUCTS IN DESIGNATED SOLUTIONS. THE CUSTOMER'S MANUFACTURING TEST ENVIRONMENT HAS NOT MET THE STANDARDS SET BY MELLANOX TECHNOLOGIES TO FULLY QUALIFY THE PRODUCT(S) AND/OR THE SYSTEM USING IT. THEREFORE, MELLANOX TECHNOLOGIES CANNOT AND DOES NOT GUARANTEE OR WARRANT THAT THE PRODUCTS WILL OPERATE WITH THE HIGHEST QUALITY. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL MELLANOX BE LIABLE TO CUSTOMER OR ANY THIRD PARTIES FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES OF ANY KIND (INCLUDING, BUT NOT LIMITED TO, PAYMENT FOR PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY FROM THE USE OF THE PRODUCT(S) AND RELATED DOCUMENTATION EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Mellanox Technologies
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Table of Contents

List of Figures

Document Revision History

Rev 1.1 – September 04, 2018

Added section "2.5 Building Poky Initramfs" and moved other subsections underneath it

Rev 1.0 – August 10, 2018

First release

About this Manual

Welcome to the BlueField™ SW User Manual. This document provides 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.

Audience

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

Document Conventions

The following lists conventions used in this document.

NOTE: Identifies important information that contains helpful suggestions.

CAUTION: Alerts you to the risk of personal injury, system damage, or loss of data.

WARNING: Warns you that failure to take or avoid a specific action might result in personal injury or a malfunction of the hardware or software. Be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents before you work on any equipment.

WARNING: Warns you that failure to take or avoid a specific action might result in personal injury or a malfunction of the hardware or software. Be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents before you work on any equipment.

Common Abbreviations and Acronyms

Abbreviation / Acronym	Whole Word / Description
ATF	Arm Trusted Firmware
BFB	BlueField™ bootstream
BSD	BlueField Software Distribution
eMMC	Embedded Multi-Media Card
ESP	EFI system partition
FS	File system

Abbreviation / Acronym	Whole Word / Description
FW	Firmware
GDB	GNU Debugger
GPT	GUID partition
HW	Hardware
IB	InfiniBand
KGDB	Kernel debugger
KGDBOC	Kernel debugger over console
NIC	Network interface card
OCD	On-chip debugger
OVS	Open vSwitch
PCIe	PCI Express or Peripheral Component Interconnect Express
SoC	System on chip
SW	Software
UEFI	Unified Extensible Firmware Interface
UPVS	UEFI Persistent Variable Store
VPI	Virtual Protocol Interconnect

Related Documentation

For additional information, see the following documents:

Document Name	Description
InfiniBand Architecture Specification, Vol. 1, Release 1.2.1	The InfiniBand Architecture Specification that is provided by IBTA.
Firmware Release Notes for Mellanox adapter devices	See the Release Notes PDF file relevant to your adapter device under the docs/ folder of installed package.
MFT User Manual	Mellanox Firmware Tools User's Manual. See under the docs/ folder of installed package.
MFT Release Notes	Release Notes for the Mellanox Firmware Tools. See under the docs/ folder of installed package.
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 User Manual	Mellanox WinOF User Manual describes installation, configuration and operation of Mellanox WinOF driver.
VMA User Manual	Mellanox VMA User Manual describes installation, configuration and operation of Mellanox VMA driver.
BlueField 2U Reference Platform Hardware User Manual	Provides details as to the interfaces of the reference platform, specifications and hardware installation instructions.

Document Name	Description
Mellanox BlueField Reference Platforms Bring Up Guide	This document describes a step-by-step procedure of how to bring up the BlueField Reference Platform.
Mellanox BlueField SmartNIC Installation and Bring Up Guide	This document describes a step-by-step procedure of how to bring up the BlueField SmartNIC.

1 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.

Mellanox® provides software which enables users to fully utilize the BlueField™ SoC 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 SoC 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 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 which contains an extremely flexible cross-build environment allowing software targeted for the BlueField SoC to build on virtually any x86 server running any Linux distribution.

The following are other software elements delivered with BlueField SoC:

- 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

1.1 Debug Tools

BlueField SoC includes hardware support for the Arm DS5 suite as well as CoreSight™ debug & trace. As such, a wide range of commercial off-the-shelf Arm debug tools should work seamlessly with BlueField.

The BlueField SoC also supports the ubiquitous GDB.

1.2 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 from the x86 host and uses a dedicated network management interface (separate from the x86 host'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 encryption.

1.3 BlueField-based Storage Appliance

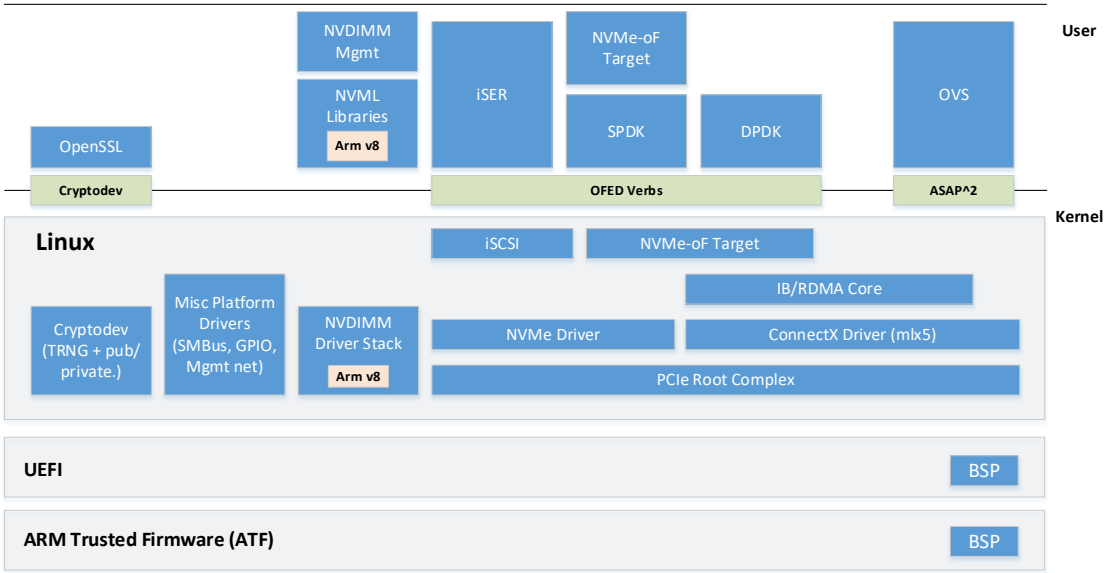
Mellanox® BlueField™ Software provides the foundation for building a JBOF (Just a Bunch of Flash) storage system including NVMe-oF target software, PCIe switch support, NVDIMM-N support, and NVMe disk hot-swap support.

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

1.4 BlueField Architecture

The BlueField architecture is a combination of two preexisting standard off-the-shelf components, Arm AArch64 processors, and Mellanox ConnectX-5 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.

Figure 1 - Interfaces on BlueField



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-5 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-5 standalone network controller cards. These interfaces take advantage of the Mellanox OFED software stack and InfiniBand verbs-based interfaces to support software.

2 Installation and Initialization

Disclaimer: This section is preliminary and subject to change. Please consult the README files in the BlueField™ Software Distribution (BSD) for the most updated content.

The BSD consists of the following images:

- BlueField-<bluefield_version>_install-bluewhale.bfb – installation BFB file for the BlueWhale development board
- BlueField-<bluefield_version>_install-smartnic_MBF1M332A.bfb – installation BFB file for the 8-core version of the BlueField SmartNIC
- BlueField-<bluefield_version>_install-smartnic_MBF1L332A.bfb – installation BFB file for the 4-core version of the BlueField SmartNIC
- 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.

2.1 Unpacking BlueField Software Distribution

To unpack the BSD, run:

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

This unpacks to a BlueField-<bluefield_version>/ subdirectory containing the following top-level hierarchy:

- bin – contains tools to manage the process of installing runtime software. For example, the mlx-mkbfb 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 Mellanox®'s 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 SoC 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.

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

The following README files in particular are important to consult for possible release specific information:

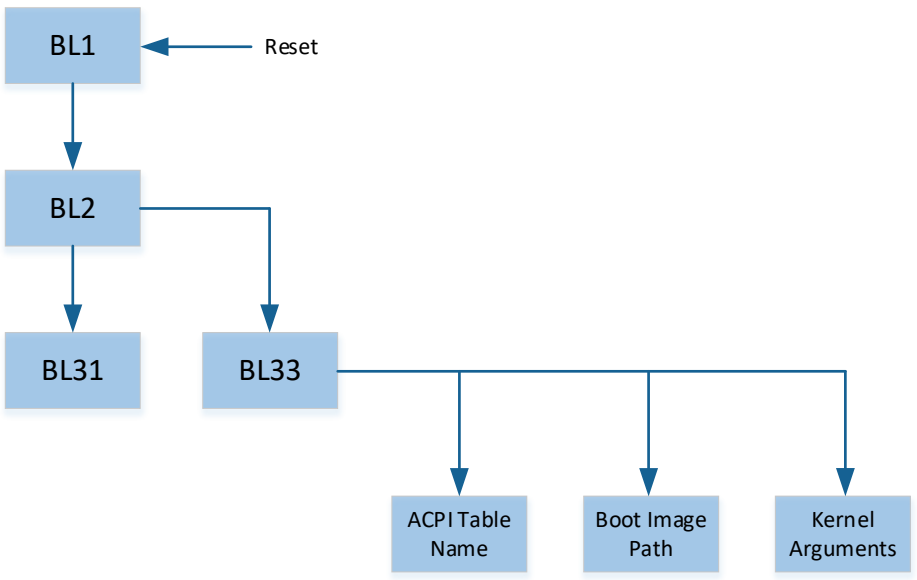
- sample/README.install
- sample/README
- distro/yocto/README-bluefield
- src/atf/README
- src/atf/README-bfb
- distro/rhel/pxeboot/README

2.2 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.

2.2.1 BFB File Overview

Figure 2 - BlueField Bootstream



The default BlueField bootstream (BFB) shown above is a standard boot BFB that is stored on the embedded Multi-Media Card (eMMC) as can be seen by the boot path that points to a GUID partition (GPT) on the eMMC device. That path is a normal UEFI boot path and it will be stored in the UPVS (UEFI Persistent Variable Store) EEPROM as a side effect of booting with this BFB. That is, if you use the mlxbf-bootctl utility to write this BFB to the eMMC boot partition, the SoC chip will read it via the boot FIFO on the RShim device by default on the next reboot.

BFB files can be useful for many things such as installing new software on a BlueField™ SoC. 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 types of files possible in a BFB are listed below.

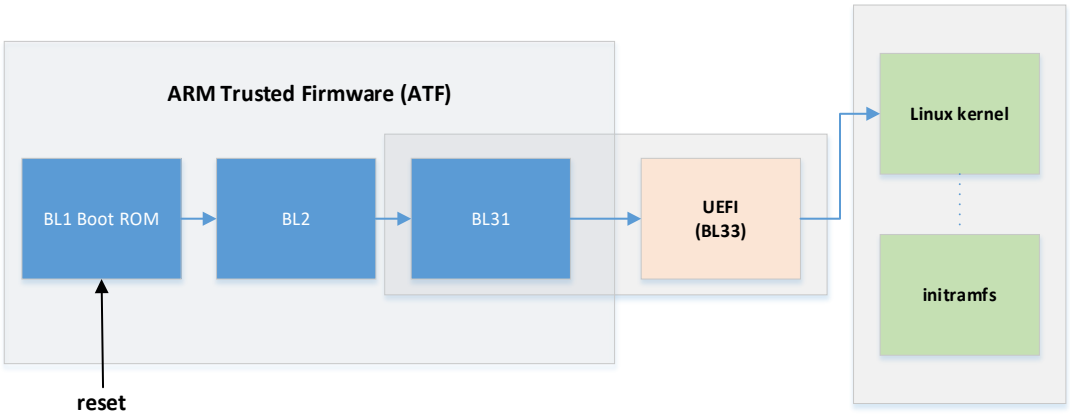
Table 1 - Types of BlueField Bootstream Files

Filename	Description	ID	Read By
bl2-cert	Trusted Boot Firmware BL2 certificate	6	BL1
bl2	Trusted Boot Firmware BL2	1	BL1
trusted-key-cert	Trusted key certificate	7	BL2
bl31-key-cert	EL3 Runtime Firmware BL3-1 key certificate	9	BL2
bl31-cert	EL3 Runtime Firmware BL3-1 certificate	13	BL2
bl31	EL3 Runtime Firmware BL3-1	3	BL2
bl32-key-cert	Secure Payload BL3-2 (Trusted OS) key certificate	10	BL2
bl32-cert	Secure Payload BL3-2 (Trusted OS) certificate	14	BL2
bl32	Secure Payload BL3-2 (Trusted OS)	4	BL2
bl33-key-cert	Non-Trusted Firmware BL3-3 key certificate	11	BL2
bl33-cert	Non-Trusted Firmware BL3-3 certificate	15	BL2
bl33	Non-Trusted Firmware BL3-3	5	BL2
boot-acpi	Name of the ACPI table	55	UEFI
boot-dtb	Name of the DTB file	56	UEFI
boot-desc	Default boot menu item description	57	UEFI
boot-path	Boot image path	58	UEFI
boot-args	Arguments for boot image	59	UEFI
boot-timeout	Boot menu timeout	60	UEFI
uefi-tests	Specify what UEFI tests to run	61	UEFI
image	Boot image	62	UEFI
initramfs	In-memory filesystem	63	UEFI

2.2.2 BlueField Boot Process

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

Figure 3 - Basic BlueField Boot Flow



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 an 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.

2.2.3 The mlxbf-bootctl Program

Access to all the boot partition management is done via a program packaged with the BlueField software called “bootctl”. The binary 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 bootctl is as follows:

```
syntax: mlxbf-bootctl [--help|-h] [--swap|-s] [--device|-d MMCFILE]
                      [--output|-o OUTPUT]
                      [--bootstream|-b BFBFILE] [--overwrite-current]
                      [--watchdog-swap interval | --nowatchdog-swap]
```

Flag functionality breakdown:

- --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!)
- --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.

2.2.4 Upgrading the Bootloader

In most deployments, the Arm cores of BlueField™ are expected to obtain their software stack 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.

2.2.5 Updating the Boot Partition

To update the boot partition on the Arm cores, let us assume to have a new bootstream file called “bootstream.new” which we would like to install and validate. To update to the bootstream, run:

```
# mlxbf-bootctl --bootstream bootstream.new --swap
# reboot
```

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, and then reboots to use it. (You may also use “--overwrite-current” instead of “--swap”, which just overwrites the current boot partition. But this is not recommended as there is no easy way to recover if the new booter code does not bring the system up.)

2.2.6 Safely Updating with a 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 booter responds to by switching the primary and alternating boot partitions as part of resetting into its original state.

2.2.7 Safely Updating Boot Software from the Arm Cores

Without a BMC, the Arm watchdog may be used to achieve similar results. 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. In this case, the user may run:

```
# mlxbf-bootctl --bootstream bootstream.new --swap --watchdog-swap 60
# reboot
```

With these commands, the user reboots the system, and, if it hangs for 60 seconds or more, the watchdog fires and resets the chip, the booter 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 booter 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 the Linux watchdog documentation (e.g. https://www.kernel.org/doc/Documentation/watchdog/watchdog-api.txt).

To disable the watchdog completely, for example, 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 booter 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.

The aforementioned steps can be done manually, or can be done automatically by software running in the newly-booted system.

2.2.8 Changing the 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, the user should partition their eMMC into multiple partitions appropriately.

For example, the user may have a single FAT partition from which UEFI can read the kernel image file, but the new bootstream contains a UEFI bootpath pointing to an updated kernel image. Similarly, the user may have two Linux partitions, and their upgrade procedure would write a new filesystem into the “idle” Linux partition, then reboot with the bootstream holding kernel boot arguments which direct it to boot from the previously idle partition.

The details on how exactly to do 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.

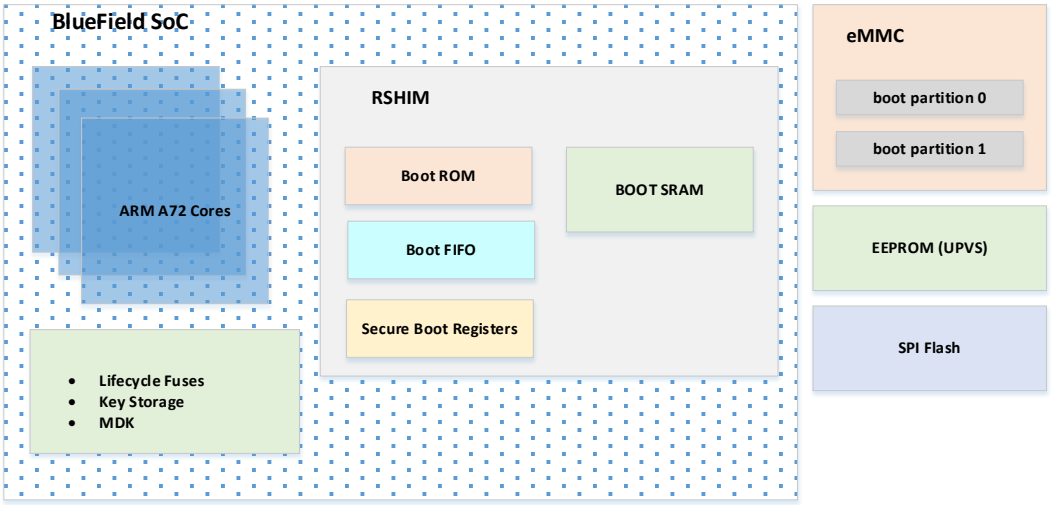
For more information, please refer to EDK2 user documentation on Github at: https://github.com/tianocore/tianocore.github.io/wiki/EDK-II-User-Documentation.

2.3 Building Arm Trusted Firmware

NOTE: 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.

Figure 4 - BlueField High Level Hardware View



ATF has various bootloader stages when loading:

- BL1 – BL1 is stored in the on-chip boot ROM; it is executed when the primary core is reset. Its main functionality is to do some initial architectural and platform initialization to the point where it can load the BL2 image, then it loads BL2 and switches execution to it.
- BL2 – BL2 is loaded and then executed on the on-chip boot SRAM. 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). After loading the images, 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 see the EDK2 source.

NOTE: 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.

2.3.1 Building ATF Images

To get the source code, directly execute the atf-56036e.patch file found in the directory /src/atf. 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 boot loader images which might need to be built are BL2 and BL31 (refer to the EDK2 documentation of how to build EDK2 to use as BL33). Thus we are building the “bl2” and “bl31” targets.

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

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

To build for the BlueField platform, we need PLAT set to “bluefield” when invoking “make”. You also need to set the TARGET_SYSTEM according to the specific system for which you are building (e.g. “bluewhale” if building for the BlueWhale reference platform). Every supported system has its own subdirectory under $SOURCE/plat/mellanox/bluefield/system/. If you are using your own system, you can use the “generic” platform as a starting point, create your own system’s subdirectory under the system directory, copy the files over from the generic subdirectory, and modify them to suit your particular machine. You can also pass the BUILD_BASE variable to specify where you want the files to be built. So to perform a basic build:

```
export CROSS_COMPILE=/path/to/cross/compiler/aarch64-poky-linux-
make BUILD_BASE=/path/to/build PLAT=bluefield TARGET_SYSTEM=bluewhale bl2 bl31
```

NOTE: If ATF is being built in an environment where the Yocto/Poky SDK script has been run (environment-setup-aarch64-poky-linux), the user needs to set the LDFLAGS to NULL (export LDFLAGS="").

After the build finishes the needed bl2.bin and bl31.bin may be found under $BUILD_BASE/bluefield/<target_system>/release/.

2.3.2 Trusted Board Boot

The other two files in the directory (mbedtls-2.2.1.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.

2.4 Building UEFI (EDK2)

After running the “edk2-*.patch” command in the directory “\src\edk2\” to set up a source tree for UEFI, cd into it and run “make -f /path/to/this/file”.

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

Note that 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.

2.4.1 Customizable Build Options

The following are the customizable UEFI build options:

- 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-none-elf-
```

NOTE: Note that 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 run. 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.

2.4.2 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 $@
```

2.5 Building Poky Initramfs

2.5.1 Basic Quick Start to Build Poky Initramfs

Run the script scripts-bluefield/yocto_dependencies from the /distro/yocto directory. Then run:

```
cd poky
source oe-init-build-env
cd conf
```

Customize your bblayers.conf file. To reproduce the shipped initramfs you can start with:

```
BBLAYERS ?= " \
  <path>/distro/yocto/poky/meta \
  <path>/distro/yocto/poky/meta-poky \
  <path>/distro/yocto/poky/meta-yocto-bsp \
  <path>/distro/yocto/meta-openembedded/meta-oe \
  <path>/distro/yocto/meta-openembedded/meta-python \
  <path>/distro/yocto/meta-openembedded/meta-networking \
  <path>/distro/yocto/meta-openembedded/meta-filesystems \
  <path>/distro/yocto/meta-openembedded/meta-perl \
  <path>/distro/yocto/meta-virtualization \
  <path>/distro/yocto/meta-linaro/meta-linaro \
  <path>/distro/yocto/meta-linaro/meta-linaro-integration \
  <path>/distro/yocto/meta-secure-core/meta-efi-secure-boot \
  <path>/distro/yocto/meta-secure-core/meta-signing-key \
  <path>/distro/yocto/meta-cloud-services/meta-openstack \
  <path>/distro/yocto/meta-bluefield \
  "
```

Replace “<path>” with your own path.

Customize your local.conf file. For example, change MACHINE to “bluefield”, include bluefield.conf, and set MLNX_OFED_PATH.

```
MACHINE ??= "bluefield"
include conf/bluefield.conf
MLNX_OFED_PATH="<path>/distro/mlnx_ofed"
```

Then run:

```
cd ..
bitbake core-image-initramfs
```

Note that the file system you just created is located in “poky/build/tmp/deploy/images/bluefield” while your kernel image will be in “tmp/deploy/images/bluefield”.

Common BlueField™ bitbake targets are:

- bitbake core-image-initramfs
- bitbake core-image-full
- bitbake core-image-full-sdk -c populate_sdk

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 following URL: https://www.yoctoproject.org/docs/latest/mega-manual/mega-manual.html.

2.5.2 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. “4.2-1.4.13.”). This is used with MLNX_OFED_PATH to find files.
- MLNX_OFED_BASE_OS – base OS version of MLNX OFED (e.g. “rhel7.3”). 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_BFB_PATH – if there are any bfb files (*.bfb) located at this location, they are copied into the root file system (/lib/firmware/Mellanox). See the "update_rootfs_bluefield" function in the meta-bluefield image recipes.
- 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.

2.5.3 Downloading Upstream Yocto and Building SDK

The meta-bluefield layer supports the Mellanox BlueField SoC.

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
```

NOTE: Mellanox is using Yocto Rocko 2.4 for this release.

2.6 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 SoC. This directory may be located anywhere you want.

Alternatively, you may download the upstream Yocto and build your own SDK; for more information, see "2.5.3 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.

2.7 RShim Host Driver

2.7.1 Building and Installing RShim Host Driver

In order to build and install the RShim host driver, run:

```
make -C /lib/modules/`uname -r`/build M=$PWD
make -C /lib/modules/`uname -r`/build M=$PWD modules_install
```

The following kernel modules are installed:

- Common modules:
  - rshim.ko – RShim common code including console support
  - rshim_net.ko – RShim network driver
- Different backends:
  - rshim_usb.ko – RShim USB backend
  - rshim_pcie.ko – RShim PCIe backend with firmware burnt
  - rshim_pcie_lf.ko – RShim PCIe backend in livefish mode

2.7.2 Loading Modules

Usually rshim.ko and rshim_usb.ko (or rshim_pcie.ko) are loaded automatically after reboot. If not, run “modprobe rshim” and “modprobe rshim_<usb | pcie | pcie_lf>.ko” to load it manually. The module rshim_net.ko creates an RShim network interface and can then be loaded on demand.

NOTE: Loading multiple backends for the same board is not recommended as it could cause potential data corruption when both backends read/write simultaneously.

2.7.3 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-bluewhale.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/rshim<N>/misc
  BOOT_MODE 1
  SW_RESET  0

# Initiate a SW reset.
echo "SW_RESET 1" > /dev/rshim<N>/misc
```

2.7.4 FAQ – What if USB and PCIe Access are Enabled?

In this case both rshim_usb.ko and rshim_pcie.ko are loaded automatically which causes conflict when they write to RShim simultaneously. One solution is to create a configuration file to pass "rshim_disable=1" to the specified kernel module.

The following is an example to disable RShim access via USB:

```
# Configuration file /etc/modprobe.d/rshim.conf
options rshim_usb rshim_disable=1
```

2.7.5 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

2.7.6 Permanently Changing the MAC Address of the Arm Side

The default MAC address is 00:1a:ca:ff:ff:01. It can be changed with 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-00e09803
2b8c
```

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.

2.8 OpenOCD on BlueField

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

1. Load host-side RShim drivers (assuming they have already been installed). Run:

```
$ sudo modprobe rshim_usb
```

NOTE: This is a USB use case, for PCI, a different driver must be used.

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

Set the environment variable to be used by OpenOCD. Run:

```
$ export RSHIM_DEST=/dev/rshim0/rshim
```

2. Run OpenOCD:

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

Once started, OpenOCD runs a gdb-server in the background to accept commands from a GDB client.

➢ ***To start the GDB client:***

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

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

2. Run 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...>
```

3 Programming

This chapter is meant for application developers and expert users who wish to develop applications over BlueField™ SW.

The sample directory contains sample Linux and initramfs content which can be used to validate that the user's hardware can boot up to the shell prompt. Typically the user would use their distribution's kernel and userspace filesystem contents instead (e.g. Yocto, RedHat, or Ubuntu).

The build-images script takes the sample kernel file ("Image") and the sample "initramfs" file and unpacks them into partition images and disk images which are typical of what might be burned into the eMMC device used to boot up. We create a disk image that is partitioned to have an initial boot filesystem and an additional root filesystem. The boot filesystem holds the image file and the initramfs; the root filesystem holds an unpacked version of the initramfs.

The build-bfb script then allows the user to utilize these images to create several different BlueField boot stream files which can boot the BlueField system via USB or PCI from a host system, or can be copied to BlueField's eMMC boot partition.

You can use the script to create boot stream files with the following properties:

```
build-bfb -i rshim
```

Provide the entire boot environment (ATF, UEFI, kernel, initramfs) in a single boot stream file.

```
build-bfb mmc0
build-bfb nvme0
```

Load the kernel from the boot partition, then boot using the root partition. The initramfs file on the boot partition is not used. The two variants are examples of how to configure the partition names for different devices.

```
build-bfb -i mmc0
build-bfb -i nvme0
```

Load the kernel and the initramfs from both the boot partition, and boot the kernel using the initramfs. The root partition is not used.

```
build-bfb --no-gpt --root /dev/nvme0n1p1 mmc0
```

Load the kernel from the the eMMC (configured as just a large boot partition without GPT), then boot using an NVMe root partition. The "initramfs" file is not used.

4 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.

4.1 Boot Option

The boot option is a unique identifier for a UEFI boot entry. This identifier is assigned when the boot entry is created, and it does not change. It also represents the boot option in several lists, including the BootOrder array, and it is the name of the directory on disk in which the system stores data related to the boot entry, including backup copies of the boot entry. A UEFI boot entry ID has the format “Bootxxxx” where xxxx is a hexadecimal number that reflects the order in which the boot entries are created.

Besides the boot entry ID, the UEFI boot entry has the following fields:

```
- Description
    (e.g: Yocto, CentOS, Linux from rshim)
- Device Path
    (e.g: VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)/Image)
- Boot arguments
    (e.g: console=ttyAMA0 earlycon=pl011,0x01000000 initrd=initramfs)
```

4.2 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.

```
                                                  Device Path :
Boot Option Menu                                  VenHw(F019E406-8C9C-11
                                                  E5-8797-001ACA00BFC4)/
Linux from rshim                                  Image
Yocto Poky
EFI Misc Device
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.0. .*
  00000020: 65 00 61 00 72 00 6C 00-79 00 63 00 6F 00 6E 00  *e.a.r.l.y.c.o.n.*
  00000030: 3D 00 70 00 6C 00 30 00-31 00 31 00 2C 00 30 00  *=.p.l.0.1.1.,.0.*
  00000040: 78 00 30 00 31 00 30 00-30 00 30 00 30 00 30 00  *x.0.1.0.0.0.0.0.*
  00000050: 30 00 20 00 20 00 69 00-6E 00 69 00 74 00 72 00  *0. . .i.n.i.t.r.*
  00000060: 64 00 3D 00 69 00 6E 00-69 00 74 00 72 00 61 00  *d.=.i.n.i.t.r.a.*
  00000070: 6D 00 66 00 73 00 00 00-                         *m.f.s...*
Option: 01. Variable: Boot0002
  Desc    - Yocto Poky
  DevPath - HD(1,GPT,3DCADB7E-BCCC-4897-A766-3C070EDD7C25,0x800,0xAE800)/Image
  Optional- Y
  00000000: 63 00 6F 00 6E 00 73 00-6F 00 6C 00 65 00 3D 00  *c.o.n.s.o.l.e.=.*
  00000010: 74 00 74 00 79 00 41 00-4D 00 41 00 30 00 20 00 *t.t.y.A.M.A.0. .*
  00000020: 65 00 61 00 72 00 6C 00-79 00 63 00 6F 00 6E 00  *e.a.r.l.y.c.o.n.*
  00000030: 3D 00 70 00 6C 00 30 00-31 00 31 00 2C 00 30 00  *=.p.l.0.1.1.,.0.*
  00000040: 78 00 30 00 31 00 30 00-30 00 30 00 30 00 30 00  *x.0.1.0.0.0.0.0.*
  00000050: 30 00 20 00 72 00 6F 00-6F 00 74 00 3D 00 2F 00  *0. .r.o.o.t.=./.*
  00000060: 64 00 65 00 76 00 2F 00-6D 00 6D 00 63 00 62 00  *d.e.v./.m.m.c.b.*
  00000070: 6C 00 6B 00 30 00 70 00-32 00 20 00 72 00 6F 00  *l.k.0.p.2. .r.o.*
  00000080: 6F 00 74 00 77 00 61 00-69 00 74 00              *o.t.w.a.i.t.*
    Option: 02. Variable: Boot0003
      Desc    - EFI Misc Device
      DevPath - VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)
      Optional- N
    Option: 03. Variable: Boot0004
      Desc    - EFI Network
      DevPath - MAC(001ACAFFFF01,0x1)
      Optional- N
    Option: 04. Variable: Boot0005
      Desc    - EFI Network 1
      DevPath - MAC(001ACAFFFF01,0x1)/IPv4(0.0.0.0)
      Optional- N
    Option: 05. Variable: Boot0006
      Desc    - EFI Network 2
      DevPath - MAC(001ACAFFFF01,0x1)/IPv6(0000:0000:0000:0000:0000:0000:0000:0000)
      Optional- N
```

```
    Option: 06. Variable: Boot0007
      Desc    - EFI Network 3
      DevPath - MAC(001ACAFFFF01,0x1)/IPv4(0.0.0.0)/Uri()
      Optional- N
    Option: 07. Variable: Boot0008
      Desc    - EFI Internal Shell
      DevPath - MemoryMapped(0xB,0xFE5FE000,0xFEAE357F)/FvFile(7C04A583-9E3E-4F1C-
AD65-E05268D0B4D1)
      Optional- N
```

NOTE: Boot arguments are printed in Hex mode, but you may recognize the boot parameters printed on the side in ASCII format.

4.3 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.

5 Installing Popular Linux Distributions on BlueField

5.1 Installing CentOS 7.x Distribution

This section provides instructions on how to install the Arm-based CentOS 7.x on a BlueField system.

5.1.1 Requirements

- Host machine running CentOS 7.x

NOTE: CentOS 6.2+ needs 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-1.0.xxxxxx.yyyyy.tar.xz and run:

```
# tar Jxvf BlueField-1.0.xxxxxx.yyyyy.tar.xz -C <PATH>.
```

Then the "BF_INST_DIR" could be found under the directory "<PATH>/BlueField-1.0.xxxxxx.yyyyy".

5.1.2 Host Machine Setup

1. Download the centos installation ISO file from the following URL: http://mirror.centos.org/altarch/7/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 web at http://www.mellanox.com/page/products_dyn?product_family=34&mtag=flexboot by selecting "DUD & kIOS Download" ➔ Version (like 4.2-1.4.10.0) ➔ RHEL/CentOS 7.4 ➔ x86_64 ➔ dd-rhel7.4-mlnx-ofed-xxx.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 PXE boot 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] [-t bluewhale | smartnic_MBF1M332A | smart-
nic_MBF1L332A] [-k]
```

NOTE: UART1 (ttyAMA1) is used by default. To specify a different console use "-c xxx". SmartNIC uses UART0, so it takes "-c ttyAMA0".

NOTE: The option "-k" enables automatic installation according to the kickstart file ks.cfg.

NOTE: Once the option "-t" is specified, a nonpxe.bfb is 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.

5.1.3 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).

   Reference platform:

   ```
   # cat <BF_INST_DIR>/sample/install-bluewhale.bfb > /dev/rshim0/boot
   ```

   MBF1M332A NIC:

   ```
   # cat <BF_INST_DIR>/sample/install-smartnic_MBF1M332A.bfb >
   /dev/rshim0/boot
   ```

   MBF1L332A NIC:

   ```
   # cat <BF_INST_DIR>/sample/install-smartnic_MBF1L332A.bfb >
   /dev/rshim0/boot
   ```

   The board will boot into Linux.

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.

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

   This step is needed to update the boot partition images.

5.1.4 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.

NOTE: 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.

5.1.5 CentOS Installation

6. Follow the installation wizard.

5.1.6 Post-installation

1. Enable “yum install” or external network access after CentOS installation.

   a. On the host side:

```
# modprobe rshim_net
# 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. On the BlueField™ device side:

```
# ifdown eth0; ifup eth0
```

2. Install driver RPMs from source.

   - 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).

   Below 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.el7a.aarch64.rpm
```

NOTE: 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).

   a. Remove pre-installed Kernel module.

```
# KVER=$(uname -r)
# mkdir /boot/tmp
# cd /boot/tmp
# gunzip < ../initramfs-${KVER}.img | cpio -i
# rm -rf lib/modules/${KVER}/extra
# rm -f lib/modules/${KVER}/updates/tmfifo*.ko
```

```
# cp ../initramfs-${KVER}.img ../initramfs-${KVER}.img-bak
# find | cpio -H newc -o | gzip -9 > ../initramfs-${KVER}.img
# rm -rf /boot/tmp
# depmod -a
```

b. Set time. Run the command “date” to verify whether it has the correct date setting. If it does not, use the command below as an example to get the date from the HOST machine, and use the command “date -s” to set it on the Arm side.

```
# date
Wed Jan 10 18:45:58 IST 2018   (sample output)

# date -s "Mon Apr 23 18:45:58 EST 2018"
```

c. Install OFED on the BlueField board.

Below is an example on how to build and install MLNX_OFED_LINUX-4.2-1.4.13.0-rhel7.4.

i. On the host side:

```
# scp MLNX_OFED_LINUX-4.2-1.4.13.0-rhel7.4alternate-aarch64.iso
root@192.168.100.2:/opt
```

ii. On the Arm side:

```
# mount /opt/MLNX_OFED_LINUX-4.2-1.4.14.0-rhel7.4alternate-
aarch64.iso /mnt
# cd /mnt
```

If the running kernel is “4.11.0-22.el7a.aarch64” then run:

```
# ./mlnxofedinstall --with-nvmf
```

Otherwise, run the commands below to rebuild drivers for the running kernel:

```
# yum install python-devel redhat-rpm-config rpm-build gcc
# yum install kernel-devel-$(uname -r) tcl gcc-gfortran tk
# ./mlnxofedinstall --add-kernel-support --skip-repo --with-nvmf
```

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

5.1.7 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.

5.1.8 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 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.

5.1.9 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.

5.1.10 Installation Troubleshooting and FAQ

5.1.10.1 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).

5.1.10.2 How to upgrade existing CentOS to some BlueField release without reinstallation

1. Follow Step 3 of “5.1.3 Basic Yocto Installation” to push the installation image via RShim (USB or PCIe).

NOTE: 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 “5.1.6 Post-installation” to install/upgrade the tmfifo driver and other drivers as needed from the source RPM.

5.1.10.3 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.

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

See section “2.7.62.7.6 Permanently Changing the MAC Address of the Arm Side”.

5.1.10.5 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
```

5.1.10.6 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.

5.2 Running RedHat on BlueField

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

A driver disk is required to support the eMMC hardware typically used to install the media onto. The driver disk also supports the tmfifo networking interface that allows creating a net-

work 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.

5.2.1 Provisioning ConnectX Firmware

Prior to installing RedHat, you should ensure that the ConnectX SPI ROM firmware has been provisioned. If the BlueField is connected to an external host via PCIe, and is not running in Secure Boot mode, this is typically done by using the Mellanox MFT tools 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.

NOTE: 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.

NOTE: 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.

5.2.2 Managing the Driver Disk

As discussed previously, you likely need a driver disk for RedHat installations. Mellanox provides a number of pre-built driver disks, as well as a documented flow for building one for any particular RedHat version. See section 5.1.7 “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).

5.3 Installing the Reference Yocto Distribution

The BlueField processor should be attached to an external host via a USB connection. The external host is required to be running Linux. This process has been tested with an external host running CentOS 7.4.

You will need to install the provided “rshim”, “rshim_net”, and “rshim_usb” drivers on the external host. These drivers are required to communicate with the BlueField device over the USB interface.

The initramfs contains ConnectX® firmware in the /lib/firmware/mellanox directory. MFT is also installed on the initramfs if you wish to update the ConnectX firmware.

1. On the external host.

   To use the default console, run:

   ```
   cat <bluefield_local_path>/sample/install-*.bfb > /dev/rshim0/boot
   ```

   To use Serial-Over-LAN for BlueField console on the reference platform:

   ```
   echo "console=ttyAMA0 earlycon=pl011,0x01000000 initrd=initramfs" > bootarg
   <bluefield_local_path>/bin/mlx-mkbfb --boot-args bootarg <bluefield_local_path>/install/
   sample/install-bluewhale.bfb install.bfb
   ```

   This creates a new install.bfb in your current directory which uses Serial-Over-LAN. Use this image in place of the install-bluewhale.bfb:

   ```
   cat install.bfb > /dev/rshim0/boot
   ```

2. Run this on the external host:

   ```
   modprobe rshim_net
   ifconfig eth<x> 192.168.100.1/24 up
   scp <bluefield_local_path>/core-image-full-bluefield.tar.xz root@192.168.100.2:/tmp/
   ```

3. On the BlueField console port, run:

```
/opt/mlnx/scripts/bfinst --fullfs /tmp/core-image-full-bluefield.tar.xz

shutdown -r now
```

6 Troubleshooting and FAQ

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-1.0.alpha5.10458.tar.xz). In this FAQ section, it is assumed that the tarball is uncompressed under directory "$BF_INSTALL".

The default images are located under $BF_INSTALL/sample.

How to install Yocto

See section "5.3 Installing the Reference Yocto Distribution" or /sample/README.install.

How to install CentOS

See section "5.1 Installing CentOS 7.x Distribution" or /distro/rhel/pxeboot/README.

How to find the software versions of the running system

- 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:

```
root@bluefield:~# cat /etc/bluefield_version
1.0.alpha7.10493
```

How to upgrade the host RShim driver

See section "2.7 RShim Host Driver" or /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_INSTALL/sample/<install-xxxxx.bfb> > /dev/rshim<N>/boot
```

2. Log into the BlueField target:

```
# /opt/mlnx/scripts/bfrec
```

How to upgrade ConnectX firmware

With Yocto running, the default firmware images are under /lib/firmware/mellanox/. The mst, mlxburn, and flint tools are also available which can be used to update firmware as usual.

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: https://www.mellanox.com/page/management_tools.

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:

- “EFU Network xxx” entries with device path “PciRoot...” are ConnectX interface
- “EFU Network xxx” entries with device path “MAC(...” are for the RShim 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 change the default console of the install image

On UART0:

```
# echo "console=ttyAMA0 earlycon=pl011,0x01000000 initrd=initramfs" > boot-
arg
# $BF_INSTALL/bin/mlx-mkbfb --boot-args bootarg \
       $BF_INSTALL/sample/<install-xxxxx.bfb> install.bfb
```

On UART1:

```
# echo "console=ttyAMA1 earlycon=pl011,0x01000000 initrd=initramfs" > boot-
arg
# $BF_INSTALL/bin/mlx-mkbfb --boot-args bootarg \
       $BF_INSTALL/sample/<install-xxxxx.bfb> install.bfb
```

On RShim:

```
# echo "console=hvc0 initrd=initramfs" > bootarg
# $BF_INSTALL/bin/mlx-mkbfb --boot-args bootarg \
       $BF_INSTALL/sample/<install-xxxxx.bfb> install.bfb
```

BlueField target is stuck inside UEFI menu

Upgrade to the latest stable boot partition images, see “How to upgrade the boot partition” above.

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 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:

```
root@bluefield:~# 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.

```
root@bluefield:~# 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.

```
$INSTALL/sdk/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb
$INSTALL/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)
```

$INSTALL is the directory where the BlueField software is installed. It is assumed that the SDK has been unpacked in the same directory.

Appendix A: SmartNIC Bring-Up and Driver Installation

A.1 Installing Linux on the SmartNIC

This section demonstrates CentOS 7.4 installation on the SmartNIC. Other OSs work similarly with the PXE boot installation process.

A.1.1 Software Requirements

Table 10 - Software Requirements

Requirement	Description
CentOS 7.4 Linux OS[a]	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`
Access to the latest SmartNIC SW bundle	Mellanox uses box.com to distribute BlueField software. Contact your sales/support representative for a custom link to download BlueField software releases.
Tarball	In this document, we assume the tarball BlueField-<version>.tar.gz is extracted at / root, to do this, run the following command: `tar -xvf BlueField-<version>.tar.xz -C /root`

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

A.1.2 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 SmartNIC, the host must be set up for it to be capable of provisioning the SmartNIC. The RShim USB driver is installed on the host to communicate with the RShim device on the BlueField SoC. 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 SmartNIC, it is not required on the end machine.

A.1.2.1 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 Section 5.1.2.2.

```
/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 \
  -t smartnic_MBF1M332A \
  -k
```

Note that there should be no firewall blocking the IP communication between the SmartNIC and the x86 host machine. If a firewall exists, disable it with the following commands:

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

Table 11 - Commands for Disabling Firewall

Flag	Function
"-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
"-i" Flag	Points to the OS installation disk. This is the image that is accessed via PXE boot to install the OS on the SmartNIC
"-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
"-c" flag	Specifies the default UART port for the OS to use since the BlueField SoC has two Arm UARTs. For the SmartNIC, "ttyAMA0" is used, which is UART0
"-t" Flag (Optional)	When specified and given the argument of what platform is set (SmartNIC 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
"-k" Flag (Optional)	Kickstarts auto-installation based on a default kickstart file which is installed as /var/pxe/ks/ks.cfg (optional)

A.1.2.2 Setup Procedure Without Installation Script

If the host is running CentOS 7 or equivalent, please refer to Section 5.1.2.1 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.

A.1.2.2.1 Step 1: Set up the RShim Interface

The RShim driver communicates with the RShim device on the BlueField SoC. The RShim is in charge of many miscellaneous functions of the SoC, 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 though firmware. This is done by adding “multi_function.rshim_pf_en = 0x1” to the [fw_boot_config] section in the firmware's ini file.

A.1.2.2.2 Step 2: Install RShim Drivers

The RShim drivers are installed as a part of MLNX_OFED_LINUX installation process on the host. See Installing MLNX_OFED on the Host on page 51 for further details.

- **rshim** is the base RShim kernel module required by all other RShim modules
- **rshim_usb** is a module that accesses the RShim via the USB connector
- **rshim_pci** is a module that accesses the RShim via the PCIe slot
- **rshim_net** is a module which creates a network interface to the RShim

The kernel modules do not compile on CentOS 5 or eariler.

A.1.2.2.3 Step 3: Configure RShim Net Interface

To use the RShim net interface, create a udev rule and a config file.

- To create the udev rule, run the following command:

```
cat >/etc/udev/rules.d/91-tmfifo_net.rules <<EOF
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="00:1a:ca:ff:ff:02", ATTR{type}=="1",
NAME    ="tmfifo_net0"
RUN+="/usr/sbin/ifup tmfifo_net0"
EOF
```

- To create the RShim interface config file, run the following command:

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

A.1.2.2.4 Step 4: Configure the TFTP Server

The host should be configured to act as a TFTP server to the SmartNIC via the USB RShim network. This server provides the required files by the SmartNIC 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/.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/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-<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=dhcp 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”.

A.1.2.2.5 Step 5: Set Up the DHCP Server

DHCP server set up on the host is required for SmartNIC to get a private IP from the host for PXE boot process completion. Configure the correct server names and domain names so that the SmartNIC 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 servers names 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.

A.1.2.2.6 Step 6: Set Up the HTTP Server

The TFTP server allows the PXE boot to load the initrd and kernel. The SmartNIC 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 Linx 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
```

A.1.3 Flashing the SmartNIC Bootloader Code

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

A.1.3.1 Opening a Terminal Connection to the SmartNIC

To open a console window to the SmartNIC, 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”.

Figure 8: Minicom Settings – Example

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

A.1.3.2 Using the Initial Install Bootstream

1. On the host side, ensure that the RShim kernel modules are loaded:

```
modprobe rshim_usb
modprobe rshim_net
```

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 SmartNIC by pushing a bootstream to it, which is done by writing a boot-stream 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 SmartNIC:

```
cat /root/BlueField-<version>/sample/install-smartnic_MBF1M332A.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.

Figure 9: Yocto Log

```
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 2.6.2  | VT102 |      Offline
```

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

A.1.4 Installing CentOS 7 on BlueField SmartNIC

If the error “no root is found” appears in the installation process, check or disable the firewall as needed on the x86 host machine.

A.1.4.1 Full PXE Boot Installation

1. Get to the UEFI boot menu.
   a. Reboot the SmartNIC 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.

Figure 10: UEFI Boot Menu

```
Continue
Select Language          <Standard English>
Boot Manager
Device Manager
Boot Maintenance Manager

                                              This selection will
                                              direct the system to
                                              continue to booting
                                              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.

Figure 11: UEFI Boot Manager

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.

Figure 12: Option to Install CentOS

5. 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.

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
```

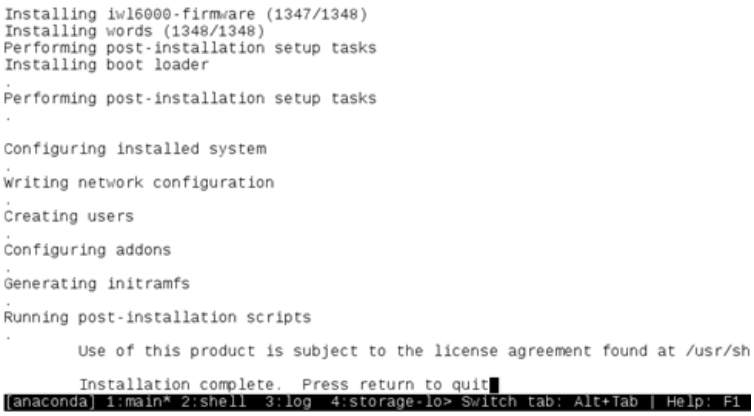
```
 5)  [ ] Basic Web Server                      Workstation
  Please make your choice from above ['q' to quit | 'c' to continue | 'r' to refresh]: c
================================================================================
================================================================================
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 (mmcblk0)

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
```

7. Enter “b” and press “enter” to initiate the installation process.

8. Press “Enter” to reboot into CentOS.

Figure 13: CentOS Installation Completion Screen



A.1.4.2 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
```

A.2 MLNX_OFED Installation

A.2.1 Installing MLNX_OFED on the Arm Cores

A.2.1.1 Prerequisite Packages for Installing MLNX_OFED

- MLNX_OFED installation requires some prerequisite packages to be installed on the system

  Currently, CentOS installed on the SmartNIC 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 SmartNIC 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 SmartNIC 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.

A.2.1.2 Removing Pre-installed Kernel Module

There are cases where the kernel is shipped with an earlier version of the mlx5_core driver taken from the upstream Linux code. This version does not support the BlueField Arm, but is loaded before the MLNX_OFED driver, and therefore, needs to be removed.

To remove the kernel module from the initramfs, run the following command:

```
mkdir /boot/tmp
cd /boot/tmp
gunzip  <  ../initramfs-4*64.img  |  cpio  -i
rm -f lib/modules/4*/updates/mlx5_core.ko
rm -f lib/modules/4*/updates/tmfifo*.ko
cp ../initramfs-4*64.img ../initramfs-4.11.0-22.el7a.aarch64.img-bak
find | cpio -H newc -o | gzip -9 > ../initramfs-4*64.img
rpm -e mlx5_core
depmod -a
```

A.2.1.3 Installing MLNX_OFED on the SmartNIC

1. Copy the MLNX_OFED image to the SmartNIC via the USB network. The MLNX_OFED images should be provided in the software drop:

   ```
   scp MLNX_OFED_LINUX-4.2-X.X.X.X-rhel7.4alternate-aarch64.iso \
     root@192.168.100.2:/root
   ```

2. Mount the image on the SmartNIC:

   ```
   mount /root/MLNX_OFED_LINUX-4.2-X.X.X.X-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 --bluefield
   ```

   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. Restart openibd:

   ```
   /etc/init.d/openibd restart
   ```

A.2.1.4 Setting ECPF as eSwitch Manager and Page Supplier

After installing MLNX_OFED on the Arm core, enable ECPF (Embedded CPU Physical Function) as esw_manager and assign ECPF as the page supplier for all functions, including the host PFs and VFs. Perform the following:

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
mlxconfig -d /dev/mst/mt41682_pciconf0 s ECPF_ESWITCH_MANAGER=1 ECPF_PAGE_SUPPLIER=1
mlxconfig -d /dev/mst/mt41682_pciconf0.1 s ECPF_ESWITCH_MANAGER=1 ECPF_PAGE_SUPPLIER=1
```

4. Power cycle the server.

A.2.1.5 Updating SmartNIC Firmware on the Host

The below steps demonstrate how to manually update the firmware if the 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-MBF1M332A-AENA_Ax.bin \
  root@192.168.100.2:/root
```

2. Start the MST service to locate the device. MST is installed along with MLNX_OFED:

```
 [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
```

3. 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-18_22_0205-MBF1M332A-AENA_Ax.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
```

4. Power cycle the x86 for the new firmware to take effect.

Figure 14: Firmware Burning in Process

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

  Device Type:      BlueField
  Part Number:      MBF1M332A-AENA_Ax
  Description:      Bluefield Network Adapter; 8 cores; dual-port SFP28
  PSID:             MT_0000000131
  PCI Device Name:  0000:03:00.0
  Base GUID:        0002c90300212111
  Base MAC:         0002c92dd111
  Versions:         Current        Available
     FW             18.99.3902     18.99.3902
     PXE            N/A            3.5.0402
     UEFI           N/A            14.15.0016

  Status:           Forced update required

---------
Found 1 device(s) requiring firmware update...

Device #1: Updating FW ...
  8%
```

After MLNX_OFED is installed on the Arm cores, use the mlx5_core driver to use the two Ethernet ports on the SmartNIC. If the Ethernet ports on the SmartNIC are connected to the network, there is no need to bridge the host via RShim net to access the network.

A.2.2 Installing MLNX_OFED on the Host

MLNX_OFED should be installed on any host using the SmartNIC. This includes the host used to provision the SmartNIC as well as the final system where the SmartNIC 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 Section 5.2.1.3, “Installing MLNX_OFED on the SmartNIC”, on page 48.

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.

A.3 Running Open vSwitch on the SmartNIC

A.3.1 Installing Open vSwitch

A.3.1.1 Resolving Dependencies

Before building Open vSwitch (OVS), resolve its dependencies. Run the following command:

```
yum install epel-release
yum install python-pip
yum install openssl-devel groff graphviz selinux-policy-devel
   python-sphinx python-twisted-core python-zope-interface libcap-ng-devel
```

A.3.1.2 Getting OVS Source

1. When using version 2.8.0 of the upstream OVS source, run a git clone to get it from GitHub:

```
git clone https://github.com/openvswitch/ovs.git
```

2. Checkout version 2.8.0:

```
cd ovs
git checkout v2.8.0
```

A.3.1.3 Building OVS from Source

To bootstrap OVS and build the RPMs, run:

```
./boot.sh
./configure
make -j8 rpm-fedora
```

A.3.1.4 Installing OVS RPMs

1. Find the RPM under the rpm/rpmbuild/RPMS directory.
2. To install the OVS RPM, run:

```
cd rpm/rpmbuild/RPMS/aarch64
rpm -i openvswitch-2.8.0-1.el7.centos.aarch64.rpm
cd ../noarch
rpm -I openvswitch-selinux-policy-2.8.0-1.el7.centos.noarch.rpm
systemctl start openvswitch
```

The RPMs built on the initial SmartNIC can be copied and installed on all the rest of the SmartNICs.

A.3.2 Bridging the Host to the Network Port with OVS

A.3.2.1 Connecting the Host Function to the SmartNIC Network Port

Prerequisites:

- The following network interfaces should appear on the SmartNIC:

```
[root@localhost ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
DEFAULT qlen 1000
    link/ether 00:1a:ca:ff:ff:01 brd ff:ff:ff:ff:ff:ff
3: enp3s0f0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq switchid
0002c92dd111 state DOWN mode DEFAULT qlen 1000
    link/ether 00:02:c9:2d:d1:11 brd ff:ff:ff:ff:ff:ff
4: enp3s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq switchid 0002c92dd112
state UP mode DEFAULT qlen 1000
    link/ether 00:02:c9:2d:d1:12 brd ff:ff:ff:ff:ff:ff
5: rep0-0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq switchid 0002c92dd111
state UP mode DEFAULT qlen 1000
    link/ether ee:10:47:91:95:92 brd ff:ff:ff:ff:ff:ff
6: rep1-0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq switchid 0002c92dd112
state UP mode DEFAULT qlen 1000
    link/ether 2e:c2:de:50:3e:8e brd ff:ff:ff:ff:ff:ff
```

The “eth0” interface is the RShim network interface. The “enp3s0f0” and “enp3s0f1” interfaces are the two outgoing Ethernet ports, “rep0-0” and “rep1-0” are the two physical representors that are connected to the host.

- The following network interfaces should appear on the host:

```
[root@bu-lab02 ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
DEFAULT qlen 1000
    link/ether 18:03:73:b9:34:4c brd ff:ff:ff:ff:ff:ff
3: p4p1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT qlen 1000
    link/ether 66:76:45:1a:9b:53 brd ff:ff:ff:ff:ff:ff
4: p4p2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT qlen 1000
    link/ether 12:a8:35:c6:40:eb brd ff:ff:ff:ff:ff:ff
```

The “p4p1” and “p4p2” interfaces are the representors linked to “rep0-0” and “rep1-0” on the SmartNIC.

If you do not see them, this might be because you did not manually load the host drivers due to the blacklisting operation. To manually load it, run “modprobe mlx5_core”.

Procedure:

Using OVS, bridge rep1-0 with enp3s0f1 so the host can directly use the network connection on the SmartNIC port. Follow the following steps:

1. Start the OVS service:

```
service openvswitch start
```

2. Create an OVS bridge called “armbr1”:

```
ovs-vsctl add-br armbr1
```

3. Add the two interfaces to the bridge:

```
ovs-vsctl add-port armbr1 enp3s0f1
ovs-vsctl add-port armbr1 rep1-0
```

If SR-IOV is enabled, the representor for each VF needs to be added as well.

4. To verify a successful bridge:

```
[root@localhost ~]# ovs-vsctl show
47b2b4e7-1e13-43e6-9f11-f729429217b0
    Bridge "armbr1"
        Port "rep1-0"
            Interface "rep1-0"
        Port "enp3s0f1"
            Interface "enp3s0f1"
        Port "armbr1"
            Interface "armbr1"
                type: internal
    ovs_version: "2.8.0"
```

Make sure the interfaces are up before the bridging process. An interface is up if it is displayed via “ifconfig” without the “-a” option. To bring up “rep1-0”, run “ifconfig rep1-0 up”.

5. The host is now connected to whatever is on the other side of the network port.

To enable the connection next time the SmartNIC boots, add the “openvswitch” service to the list of services to be started at boot time. For example, with “systemctl”, run “systemctl enable openvswitch”.

A.3.2.2 Verifying the Host Connection

A.3.2.2.1 Connected Peer-to-Peer

When the SmartNIC is connected to another SmartNIC 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 where the SmartNIC is connected to, run:

```
ifconfig p4p2 192.168.200.2/24 up
```

3. Have one ping the other. This is an example of SmartNIC pinging the host:

```
ping 192.168.200.1
```

If the ping does not work, it is due to mixing up of the two ports. On the other host, try using the other port instead.

A.3.2.2.2 Connected to Network with DHCP Server

To verify that the port is directly connected to a switch which is connected to the network, bring up the port like any regular Ethernet port.

1. Assuming the interface is called p4p2 on the host, create a file /etc/sysconfig/network-scripts/ifcfg-p4p2 with the following content:

```
NAME="p4p2"
DEVICE="p4p2"
ONBOOT=yes
NETBOOT=yes
IPV6INIT=yes
BOOTPROTO=dhcp
TYPE=Ethernet
```

2. Run:

```
ifdown p4p2
ifup p4p2
```

Running ifconfig, it is possible that the SmartNIC got its own IP address. Pinging other machines should work as well.

In cases where the host already has a network connection, bring it down first. Assuming the interface is “em1”, run “ifdown em1”. Having two interfaces to the same subnet might confuse the host and generate routing issues.

A.3.3 Enabling Offloading

Offloading the OVS and TC rules to the hardware by the SmartNIC is supported.

- To offload the OVS rules to the traffic controller, run:

```
ovs-vsctl set Open_vSwitch . Other_config:hw-offload=true
```

- To offload the TC rules to the hardware, run:

```
ethtool -K enp3s0f1 hw-tc-offload on
ethtool -K rep1-0 hw-tc-offload on
```

To verify that the rules are offloaded, dump the OVS and hardware rules and check if they match.

- To dump the OVS rules, run:

```
[root@localhost ~]# ovs-dpctl dump-flows --names type=offloaded
in_port(enp3s0f1),eth(src=7c:fe:90:f1:9f:88,dst=02:b7:db:c0:fd:14),eth_type(0x0800),
packets:423, bytes:31957, used:1.000s, actions:rep1-0
in_port(enp3s0f1),eth(src=00:1c:c4:17:8a:b1,dst=02:b7:db:c0:fd:14),eth_type(0x0800),
packets:2279, bytes:151852, used:1.000s, actions:rep1-0
in_port(rep1-0),eth(src=02:b7:db:c0:fd:14,dst=00:00:5e:00:01:04), eth_type(0x0800),
packets:334, bytes:80504, used:1.000s, actions:enp3s0f1
in_port(rep1-0),eth(src=02:b7:db:c0:fd:14,dst=00:1c:c4:17:8a:b1), eth_type(0x0800),
packets:4656, bytes:4215813, used:1.000s, actions:enp3s0f1
```

- To dump the hardware rules on the host, run:

```
mlxdump -d /dev/mst/mt41682_pciconf0 fsdump --no_zero 1 --type FT --gvmi 0x3 > /tmp/
hwDump
```

MST needs to be started for "/dev/mst/mt41682_pciconf0" to appear. If not started after rebooting the SmartNIC, run "mst start".

The GVMI value should be the port number +2 of the used port for the ECPF rules.

- To parse the output, run the pretty_dump script:

```
easy_install pip
pip install termcolor
git clone https://github.com/roidayan/pretty-fs-dump.git
```

- To use the script:

```
# cd pretty-fs-dump
# python pretty-fs-dump/pretty_dump.py -s /tmp/hwDump3 -vvv -c
in_port(uplink),eth(src=7c:fe:90:f1:9f:88,dst=02:b7:db:c0:fd:14),eth_type(0x0800),
counter:0x10, action:vport0
eth(src=02:b7:db:c0:fd:14,dst=00:00:5e:00:01:04),eth_type(0x0800), counter:0x11,
action:uplink
```

These two rules, shown by the pretty_dump.py script, correspond to the first and third OVS rules.

A.3.4 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.

A.3.4.1 Configuring VXLAN Tunnel

1. Consider the enp3s0f0 to be the local VXLAN tunnel interface.
2. Build a VXLAN tunnel over OVS arm-ovs.

```
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 rep0-0 to the same arm-ovs.
4. Run traffic over PF0 on x86 (the one connected to rep0-0) to the host the SmartNIC connected.

A.3.4.2 Querying OVS VXLAN hw_offload Rules

Run:

```
ovs-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), pack-
ets:1, bytes:98, used:0.900s, actions:set(tun-
nel(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=a
2:fb:09:85:84:48,dst=ae:fd:f3:31:7e:7b),eth_type(0x0800), packets:75, bytes:7350,
used:0.900s, actions:2
```

The MTU of the end points (rep0-0 in the example above) of the VXLAN tunnel must be smaller than the tunnel interfaces (enp3s0f0) as the size of the VXLAN headers.

A.3.5 Connection Tracking

Connection tracking is performed by OVS on the Arm cores. Connection tracking on arm-ovs currently works without HW offload, so each packet passes through OVS and is tracked by it.

Connection tracking rules are configured using OpenFlow. Rules can be matched based on various different parameters. In the following example, the OVS "in-port" parameter is used.

A.3.5.1 Querying Connection Tracking Active Flows

To query connection tracking active flows when the setup is clean:

```
ovs-ofctl dump-flows arm-ovs --names
cookie=0x0, duration=300.323s, table=0, n_packets=232, n_bytes=13920, arp actions=NORMAL
```

A.3.5.2 Configuring Connection Tracking

To configure connection tracking with action=drop over in_port enp3s0f1:

1. Set ARP to behave normally.

```
ovs-ofctl add-flow arm-ovs "table=0,arp,action=normal"
```

The MTU of the end points (rep0-0 in the example above) of the VXLAN tunnel must be smaller than the tunnel interfaces (enp3s0f0) as the size of the VXLAN headers.

2. Set to mark IP packets with the “trk” flag and then proceed to table 1.

```
ovs-ofctl add-flow arm-ovs "table=0,arp,action=normal"
```

3. Set new flows to be committed, and to then behave normally.

```
ovs-ofctl add-flow arm-ovs "table=1,priority=1,ip,ct_state=+trk+new,action=ct(commit),
normal"
```

4. Set established flows to behave normally.

```
ovs-ofctl add-flow arm-ovs "table=1,priority=1,ip,ct_state=+trk+est,action=normal"
```

5. Set new flows coming in from the uplink of priority1 to be dropped, superseding the previous rule regarding all new flows.

```
ovs-ofctl add-flow arm-ovs "table=1,priority=2,in_port=enp3s0f1,ip,ct_state=+trk+new,
action=drop"
```

A.3.5.3 Verifying Connection Tracking Active Flows

To verify connection tracking active flows, run:

```
ovs-ofctl dump-flows arm-ovs --names
                  cookie=0x0, duration=3689.13s, table=0, n_packets=11161,
                  n_bytes=5592331, priority=0 actions=NORMAL                  /
                  cookie=0x0, duration=300.323s, table=0, n_packets=0, n_bytes=0, arp
                  actions=NORMAL                                              /
                  cookie=0x0, duration=300.304s, table=0, n_packets=0, n_bytes=0,
                  ct_state=-trk,ip actions=ct(table=1)                        /
                  cookie=0x0, duration=300.286s, table=1, n_packets=0, n_bytes=0,
                  priority=1,ct_state=+new+trk,ip actions=ct(commit),NORMAL    /
                  cookie=0x0, duration=300.263s, table=1, n_packets=0, n_bytes=0,
                  priority=1,ct_state=+est+trk,ip actions=NORMAL               /
                  cookie=0x0, duration=300.250s, table=1, n_packets=0, n_bytes=0,
                  priority=2,ct_state=+new+trk,ip,in_port="enp3s0f1" actions=drop
```

Run traffic which must be sent from in_port=enp3s0f1:

Success rate must be 0%—all packets must be dropped.

```
ovs-ofctl dump-flows arm-ovs --names
                  cookie=0x0, duration=3727.103s, table=0, n_packets=11265,
                  n_bytes=5621973, priority=0 actions=NORMAL                  /
                  cookie=0x0, duration=380.323s, table=0, n_packets=8, n_bytes=480,
                  arp actions=NORMAL                                          /
                  cookie=0x0, duration=380.304s, table=0, n_packets=60, n_bytes=8720,
                  ct_state=-trk,ip actions=ct(table=1)                        /
                  cookie=0x0, duration=380.286s, table=1, n_packets=12, n_bytes=3632,
                  priority=1,ct_state=+new+trk,ip actions=ct(commit),NORMAL    /
                  cookie=0x0, duration=380.263s, table=1, n_packets=18, n_bytes=1908,
                  priority=1,ct_state=+est+trk,ip actions=NORMAL               /
                  cookie=0x0, duration=380.250s, table=1, n_packets=10, n_bytes=1060,
                  priority=2,ct_state=+new+trk,ip,in_port="enp3s0f1" actions=drop
```

Traffic through the same port but from the other direction (out_port=enp3s0f1) should pass.

Other traffic coming through any other port on the same vSwitch should pass.

A.4 Enabling SR-IOV on the SmartNIC

Virtual functions (VFs) cannot be probed before the Arm reconfigures itself after enabling SR-IOV. To ensure this does not happen, perform the following steps on the host side:

1. Set the driver to not probe VFs. Run:

   ```
   echo 0 > /sys/module/mlx5_core/parameters/probe_vf
   ```

2. Enable SR-IOV and create the VFs. Run:

   ```
   echo (num_vfs) > /sys/class/net/ens1f0/device/sriov_numvfs
   ```

3. Reset the driver to probe VFs. Run:

   ```
   echo 1 > /sys/module/mlx5_core/parameters/probe_vf
   ```

4. Wait a few seconds for Arm to reconfigure.
5. Unbind the device from the driver. Run:

   ```
   echo (PCI Bus:Device:Function) > /sys/bus/pci/drivers/mlx5_core/unbind
   ```

   For the BDF, the four leading 0's, for example in "0000:08:00.3", are required, even if this device appears as "08:00.3" in lspci.

6. Re-bind the device from the driver to cause the driver to load. Run:

   ```
   echo (PCI Bus:Device:Function) > /sys/bus/pci/drivers/mlx5_core/bind
   ```

Repeat steps 5 and 6 for each VF.

A.5 EMMC Back-up 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.

A.5.1 Backing Up the eMMC Image

Before backing up the eMMC, all of its partitions need to be unmounted to avoid data corruption. Unmounting the root partition of the CentOS is impractical, therefore using the initial Yocto running entirely on memory is a good option.

1. If the SmartNIC is currently running CentOS, issue a shutdown command so that the kernel unmounts the entire filesystem:

```
root@localhost:~[root@localhost ~]# shutdown -h now
[  OK  ] Started Show Plymouth Power Off Screen.
[  OK  ] Stopped Dynamic System Tuning Daemon.
[  OK  ] Stopped target Network.
         Stopping LSB: Bring up/down networking...
[  OK  ] Stopped LSB: Bring up/down networking.
         Stopping Network Manager...
[  OK  ] Stopped Network Manager.
……
         Unmounting /run/user/0...
         Unmounting /mnt...
         Unmounting /boot/efi...
[  OK  ] Deactivated swap /dev/disk/by-path/platform-PRP0001:00-part3.
[  OK  ] Deactivated swap /dev/disk/by-partu...4fa-7c6a-4fd4-a795-84415d19f840.
[  OK  ] Deactivated swap /dev/disk/by-id/mmc-R1J56L_0x353c1019-part3.
[  OK  ] Deactivated swap /dev/mmcblk0p3.
[  OK  ] Deactivated swap /dev/disk/by-uuid/...291-1ad6-4e3a-b5b4-9087950a3296.
[  OK  ] Unmounted /run/user/0.
[  OK  ] Unmounted /boot/efi.
         Unmounting /boot...
[  OK  ] Unmounted /mnt.
[14370.028599] XFS (mmcblk0p2): Unmounting Filesystem
[  OK  ] Unmounted /boot.
[  OK  ] Reached target Unmount All Filesystems.
[  OK  ] Stopped target Local File Systems (Pre).
[  OK  ] Stopped Remount Root and Kernel File Systems.
……
[14370.787777] reboot: Power down
ERROR:   System Off: operation not handled.
PANIC at PC : 0x0000000000459b9c
```

2. On the host, push the install.bfb through the RShim boot device for the BlueField to boot up running the Yocto mini system entirely on memory:

```
[root@bu-lab02 ~]# cat /root/BlueField-<version>_install-smartnic_MBF1M332A.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 this is the case, flush the rules by running the following:

      ```
      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 entirely optional. It is used to monitor the progress of the backup. The backup should finish when the total data consumed is 13.8G, which is approximately 6 minutes if using the representor port.

5. On BlueField, read the entire eMMC boot partition with the “dd” command and pass it to the host:

```
root@bluefield:~# dd if=/dev/mmcblk0 bs=64M | nc 192.168.200.1 12345
```

6. If the “pv” command is used, it will start showing the transfer speed and data:

```
[root@bu-lab02 ~]# nc -l 12345 | pv | gzip -1 > / backup.dd.gz
7.69GiB 0:04:44 [64.4MiB/s] [        <=>                                  ]
```

7. Once this is complete, the generated backip.dd.gz file on the host is the compressed eMMC image of the BlueField system.

Note that the backup does not include the eMMC boot partitions, as they are actually physically seperate 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
```

A.5.2 Restoring the eMMC Image

To restore the eMMC, the BlueField system cannot be using the eMMC when recovering it, thus the mini Yocto running entirely on memory is the solution.

1. Push the install.bfb for it to boot from memory and set up the network interface that is going to be used. This step is the same as when backing up the eMMC image. Instead of transferring the image from the BlueField to the host, it is done the other way around.
2. Set up the host to extract the image and set up a netcat server to send the image:

```
[root@bu-lab02 ~]# zcat /backup.dd.gz | pv | nc -l 12345
```

3. On the BlueField side, retrieve the image using netcat and write it to the eMMC:

```
root@bluefield:~# nc 192.168.200.1 12345 | dd of=/dev/mmcblk0 bs=64M
```

This can also be done with SSH. To have the same effect, on the host, run:

```
zcat /backup.dd.gz | pv | ssh 192.168.200.2 dd of=/dev/mmcblk0 bs=64M
```

4. When this is complete, the UEFI persistent variable needs to be set up so that UEFI knows where to boot grub from. This can be done using the efibootmgr tool included in the mini Yocto:

```
root@bluefield:~# mount -t efivarfs none /sys/firmware/efi/efivars
root@bluefield:~# efibootmgr -c -d /dev/mmcblk0 -p 1 -l "\EFI\centos\grubaa64.efi" -L
"CentOS 7.4"
```

Alternatively, if this is not done, at boot time UEFI would stop at the boot menu and you would have to go to the UEFI console and use the UEFI console bcfg command to achieve the same affect:

```
Shell> bcfg boot add 0 FS0:\EFI\centos\shim.efi "CentOS 7.4"
```

5. Exit the shell and select “continue booting”, and UEFI resumes the next stage of the booting process.

   As mentioned before, this does not update the eMMC boot partitions. Therefore, if this process is used to deploy a new SmartNIC, 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_srio-
   v_dis_aarch64
   ```

A.6 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.

A.6.1 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
   ```

The aforementioned 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 which is 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.

A.6.2 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:

```
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
```

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.

A.6.3 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 has the following lines.:

   ```
   cat /etc/httpd/conf.d/pxeboot.conf
   Alias /centos7 /var/www/pxeboot
   <Directory /var/www/pxeboot>
       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/www/pxeboot
   ```

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”.

Appendix B: SmartNIC Bring-Up Troubleshooting

Table 12 - General Troubleshooting

Server unable to find the SmartNIC	• Ensure that the SmartNIC is placed correctly • Make sure the SmartNIC slot and the SmartNIC are compatible • Install the SmartNIC in a different PCI Express slot • Use the drivers that came with the SmartNIC or download the latest • Make sure your motherboard has the latest BIOS • Try to reboot the server
The SmartNIC no longer works	• Reseat the SmartNIC 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 • Reboot the server
SmartNIC stopped working after installing another	• Try removing and re-installing all SmartNICs • 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 SmartNIC port are compatible
Link light is on, but with no communication established	• Check that the latest driver is loaded • Check that both the SmartNIC and its link are set to the same speed and duplex settings