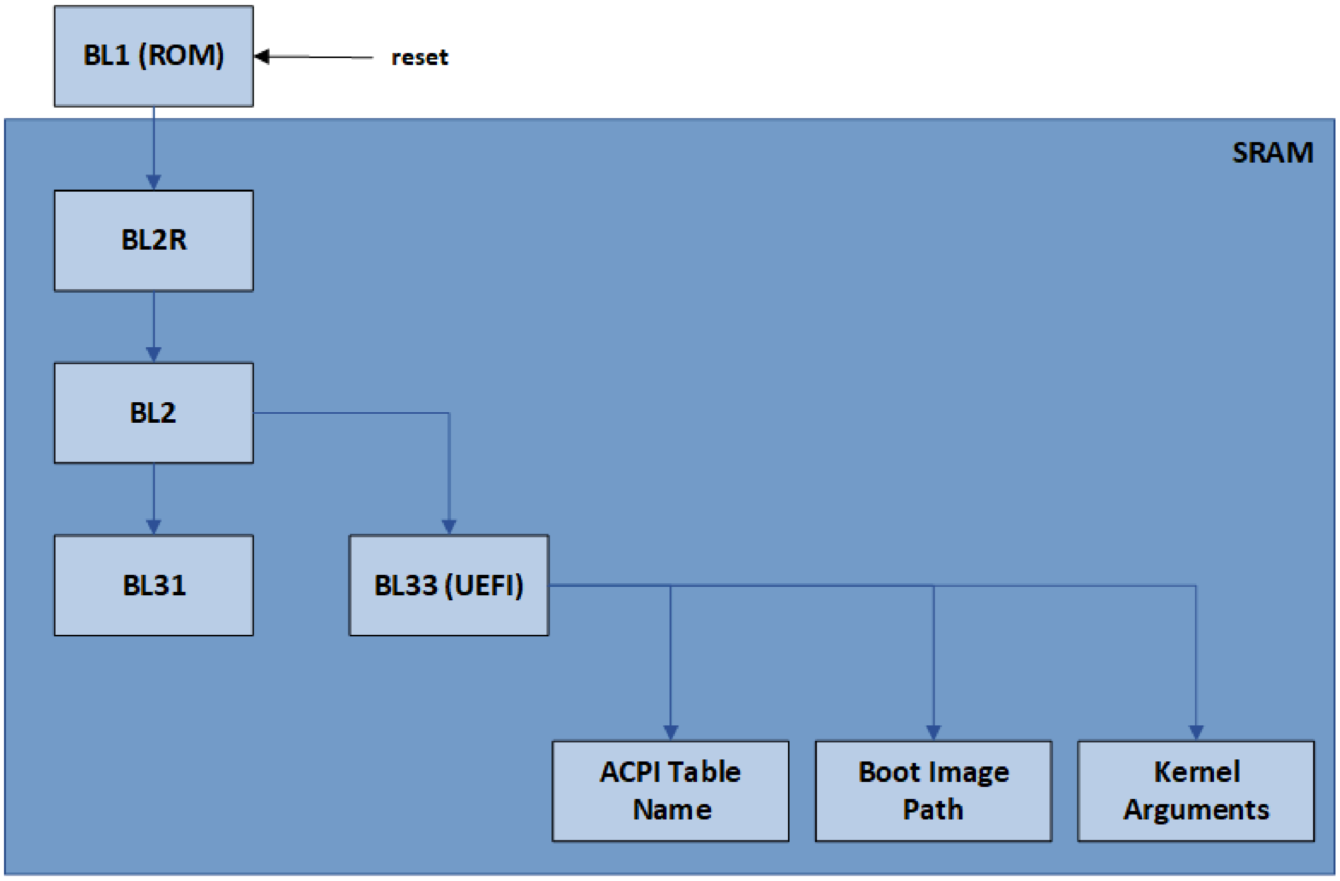
Upgrading Boot Software

Table of contents

This section describes how to use the NVIDIA® BlueField® networking platform's (DPU or SuperNIC) alternate boot partition support feature to safely upgrade the boot software. We give the requirements that motivate the feature and explain the software interfaces that are used to configure it.



BFB File Overview

The default BlueField bootstream (BFB) shown above (located at /lib/firmware/mellanox/boot/default.bfb) is assumed to be loaded from the eMMC. In it, there is a hard-coded boot path pointing to a GUID partition table (GPT) on the eMMC device. Once loaded, as a side effect, this path would be also stored in the UPVS (UEFI Persistent Variable Store) EEPROM. That is, if you use the bfrec tools provided in the mlxbf-bfscripts package to write this BFB to the eMMC boot partition (see bfrec man for more information), then during boot, the BlueField would load this from the boot FIFO, and the UEFI would assume to boot off the eMMC.

BFB files can be useful for many things such as installing new software on a BlueField. For example, the installation BFB for BlueField platforms normally contains an initramfs file in the BFB chain. Using the initramfs (and Linux kernel Image also found in the BFB) you can do things like set the boot partition on the eMMC using `mlx-bootctl` or flash new HCA firmware using MFT utilities. You can also install a full root file system on the eMMC while running out of the initramfs.

The following table presents the types of files possible in a BFB.

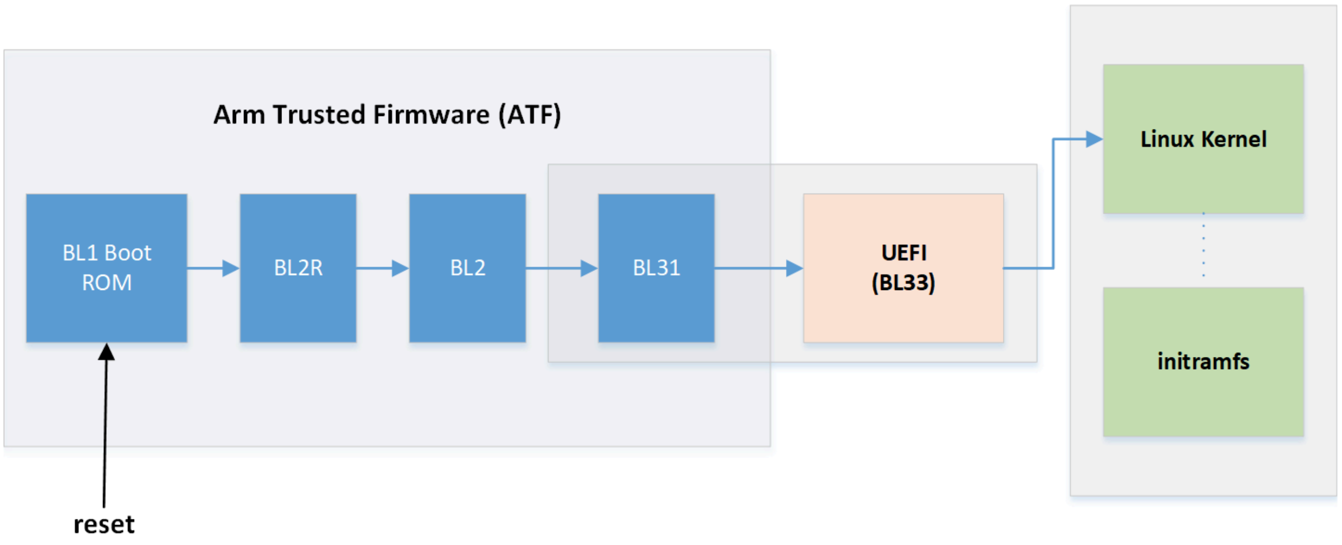
Filename	Description	ID	Read By
Bl2r-cert	Secure Firmware BL2R (RIoT Core) certificate	33	BL1
Bl2r	Secure Firmware BL2R (RIoT Core)	28	BL1
bl2-cert	Trusted Boot Firmware BL2 certificate	6	BL1/BL2R[a]
bl2	Trusted Boot Firmware BL2	1	BL1/BL2R[a]
trusted-key-cert	Trusted key certificate	7	BL2
bl31-key-cert	EL3 Runtime Firmware BL3-1 key certificate	9	BL2
bl31-cert	EL3 Runtime Firmware BL3-1 certificate	13	BL2
bl31	EL3 Runtime Firmware BL3-1	3	BL2
bl32-key-cert	Secure Payload BL3-2 (Trusted OS) key certificate	10	BL2
bl32-cert	Secure Payload BL3-2 (Trusted OS) certificate	14	BL2
bl32	Secure Payload BL3-2 (Trusted OS)	4	BL2
bl33-key-cert	Non-Trusted Firmware BL3-3 key certificate	11	BL2
bl33-cert	Non-Trusted Firmware BL3-3 certificate	15	BL2
bl33	Non-Trusted Firmware BL3-3	5	BL2
boot-acpi	Name of the ACPI table	55	UEFI
boot-dtb	Name of the DTB file	56	UEFI
boot-desc	Default boot menu item description	57	UEFI
boot-path	Boot image path	58	UEFI
boot-args	Arguments for boot image	59	UEFI
boot-timeout	Boot menu timeout	60	UEFI
image	Boot image	62	UEFI
initramfs	In-memory filesystem	63	UEFI

Note

[a] When BL2R is booted in BlueField-2 devices, both the BL2 image and the BL2 certificate are read by BL2R. Thus, the BL2 image and certificate are read by BL1. BL2R is not booted in BlueField-1 devices.

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

BlueField Boot Process



The BlueField boot flow is comprised of 4 main phases:

- Hardware loads Arm Trusted Firmware (ATF)
- ATF loads UEFI—together ATF and UEFI make up the booter software
- UEFI loads the operating system, such as the Linux kernel
- The operating system loads applications and user data

When booting from eMMC, these stages make use of two different types of storage within the eMMC part:

- ATF and UEFI are loaded from a special area known as the eMMC boot partition. Data from a boot partition is automatically streamed from the eMMC device to the eMMC controller under hardware control during the initial boot-up. Each eMMC device has two boot partitions, and the partition which is used to stream the boot data is chosen by a non-volatile configuration register in the eMMC.

- The operating system, applications, and user data come from the remainder of the chip, known as the user area. This area is accessed via block-size reads and writes, done by a device driver or similar software routine.

Upgrading Bootloader

In most deployments, the Arm cores of BlueField are expected to obtain their bootloader from an on-board eMMC device. Even in environments where the final OS kernel is not kept on eMMC—for instance, systems which boot over a network—the initial booter code still comes from the eMMC.

Most software stacks need to be modified or upgraded in their lifetime. Ideally, the user can to install the new software version on their BlueField system, test it, and then fall back to an older version if the new one does not work. In some environments, it is important that this fallback operation happen automatically since there may be no physical access to the system. In others, there may be an external agent, such as a service processor, which could manage the process.

To satisfy the requests listed above, the following must be performed:

1. Provision two software partitions on the eMMC, 0 and 1. At any given time, one area must be designated the primary partition, and the other the backup partition. The primary partition is the one booted on the next reboot or reset.

2. Allow software running on the Arm cores to declare that the primary partition is now the backup partition, and vice versa. (For the remainder of this section, this operation is referred to as "swapping the partitions" even though only the pointer is modified, and the data on the partitions does not move.)

3. Allow an external agent, such as a service processor, to swap the primary and backup partitions.

4. Allow software running on the Arm cores to reboot the system, while activating an upgrade watchdog timer. If the upgrade watchdog expires (due to the new image

being broken, invalid, or corrupt), the system automatically reboots after swapping the primary and backup partitions.

Updating Boot Partition

The Bluefield software distribution provides a boot file that can be used to update the eMMC boot partitions. The BlueField boot file (BFB) is located in the boot directory `<BF_INST_DIR>/boot/` and contains all the necessary boot loader images (i.e. ATF binary file images and UEFI binary image).

The table below presents the pre-built boot images included within the BlueField software release:

Filename	Description
bl1.bin	The trusted firmware bootloader stage 1 (BL1) image, already stored into the on-chip boot ROM. It is executed when the device is reset.
bl2r.bin	The secure firmware (RIoT core) image. This image provides support for crypto operation and calculating measurements for security attestation and is relevant to BlueField-2 devices only.
bl2.bin	The trusted firmware bootloader stage 2 (BL2) image
bl31.bin	The trusted firmware bootloader stage 3-1 (BL31) image
BLUEFIELD_EFI.fd	The UEFI firmware image. It is also referred to as the non-trusted firmware bootloader stage 3-3 (BL33) image.
default.bfb	The BlueField boot file (BFB) which encapsulates all bootloader components such as bl2r.bin, bl2.bin, bl31.bin, and BLUEFIELD_EFI.fd. This file may be used to boot the BlueField devices from the RShim interface. It also could be installed into the eMMC boot partition.

It is also possible to build bootloader images from sources and create the BlueField boot file (BFB). Please refer to the sections below for more details.

The software image includes various tools and utilities to update the eMMC boot partitions. It also embeds a boot file in `/lib/firmware/mellanox/boot/default.bfb`. To update the eMMC boot partitions using the embedded boot file, execute the following command from the BlueField console:

```
$ /opt/mellanox/scripts/bfrec
```

Note

bfrec is also available under /usr/bin.

The boot partitions update is initiated by the bfrec tool at runtime. With no options specified, the "bfrec" uses the default boot file /lib/firmware/mellanox/boot/default.bfb to update the boot partitions of device /dev/mmcblk0. This might be done directly in an OS using the "mlxbf-bootctl" utility, or at a later stage after reset using the capsule interface.

The syntax of bfrec is as follows:

```
Syntax: bfrec [--help]
              [--bootctl [<FILE>]]
              [--capsule [<FILE>]]


Description:
--help                : print help
--bootctl [<FILE>]     : update the boot partition via the kernel path. If no FILE is specified, then default is used.
--capsule [<FILE>]     : update the boot partition via the capsule path. If no FILE is specified, then default is used.
--policy POLICY        : determines the update policy. May be: single - updates the secondary partition and swaps to it, dual - updates both boot partitions, does not swap. If this flag is not specified, 'single' policy is assumed.
```

When bfrec is called with the option --bootctl, the tool uses the boot file FILE, if given, rather than the default /lib/firmware/mellanox/boot/default.bfb in order to update the boot partitions. The command line usage is as follows:

```
$ bfrec --bootctl
$ bfrec --bootctl FILE
```

Where FILE represents the BlueField boot file encapsulating the new bootloader images to be written to the eMMC boot partitions.

For example, if the new bootstream file which we would like to install and validate is called `newdefault.bfb`, download the file to the BlueField and update the eMMC boot partitions by executing the following commands from the BlueField console:

```
# /opt/mellanox/scripts/bfrec –-bootctl newdefault.bfb
```

The `--capsule` option updates the boot partition via the capsule interface. The capsule update image is reported in UEFI, so that at a later point the bootloader consumes the capsule file and performs the boot partition update. This option might be executed with or without additional arguments. The command line usage is as follows:

```
$ bfrec --capsule
$ bfrec –-capsule FILE
```

Where FILE represents the capsule update image file encapsulating the new boot image to be written to the eMMC boot partitions.

For example, if the new bootstream file which we want to install and validate is called "`newdefault.bfb`", download the file to the BlueField and update the eMMC boot partitions by executing the following commands from the BlueField console:

```
$ /opt/mellanox/scripts/bfrec --capsule newdefault.bfb $ reboot
```

For more information about the capsule updates, please refer to `<BF_INST_DIR>/Documentation/HOWTO-capsule`.

After reset, the BlueField platform boots from the newly updated boot partition. To verify the version of ATF and UEFI, execute the following command:

```
$ /opt/mellanox/scripts/bfver
```

mlxbf-bootctl

It is also possible to update the eMMC boot partitions directly with the mlxbf-bootctl tool. The tool is shipped as part of the software image (under /sbin) and the sources are shipped in the src directory in the BlueField Runtime Distribution. A simple make command builds the utility.

The syntax of mlxbf-bootctl is as follows:

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

Where:

- --device – use a device other than the default /dev/mmcblk0
- --bootstream – write the specified bootstream to the alternate partition of the device. This queries the base device (e.g. /dev/mmcblk0) for the alternate partition, and uses that information to open the appropriate boot partition device (e.g. /dev/mmcblk0boot0).
- --overwrite-current (used with "--bootstream") – overwrite the current boot partition instead of the alternate one

⚠ **Warning**

Not recommended as there is no easy way to recover if the new bootloader code does not bring the system up. Use --swap instead.

- --output (used with "--bootstream") – specify a file to which to write the boot partition data (creating it if necessary), rather than using an existing master device and deriving the boot partition device
- --watchdog-swap – arrange to start the Arm watchdog timer with a countdown of the specified number of seconds until it triggers; also, set the boot software so that it swaps the primary and alternate partitions at the next reset
- --nowatchdog-swap – ensure that after the next reset, no watchdog is started, and no swapping of boot partitions occurs

To update the boot partitions, execute the following command:

```
$ mlxbf-bootctl –-swap –-device /dev/mmcblk0 --bootstream default.bfb
```

This writes the new bootstream to the alternate boot partition, swaps alternate and primary so that the new bootstream is used on the next reboot.

It is recommended to enable the watchdog when calling mlxbf-bootcl in order to ensure that the Arm bootloader can perform alternate boot in case of a nonfunctional bootloader code within the primary boot partition. If something goes wrong on the next reboot and the system does not come up properly, it will reboot and return to the original configuration. To do so, the user may run:

```
$ mlxbf-bootctl --bootstream bootstream.new --swap --watchdog-swap 60
```

This reboots the system, and if it hangs for 60 seconds or more, the watchdog fires and resets the chip, the bootloader swaps the partitions back again to the way they were before, and the system reboots back with the original boot partition data. Similarly, if the system comes up but panics and resets, the bootloader will again swap the boot partition back to the way it was before.

The user must ensure that Linux after the reboot is configured to boot up with the `sbsa_gwdt` driver enabled. This is the Server Base System Architecture (SBSA) Generic WatchDog Timer. As soon as the driver is loaded, it begins refreshing the watchdog and preventing it from firing, which allows the system to finish booting up safely. In the example above, 60 seconds are allowed from system reset until the Linux watchdog kernel driver is loaded. At that point, the user's application may open /dev/watchdog explicitly, and the application would then become responsible for refreshing the watchdog frequently enough to keep the system from rebooting.

For documentation on the Linux watchdog subsystem, see Linux watchdog documentation.

To disable the watchdog completely, run:

```
$ echo V > /dev/watchdog
```

The user may select to incorporate other features of the Arm generic watchdog into their application code using the programming API as well.

Once the system has booted up, in addition to disabling or reconfiguring the watchdog itself if the user desires, they must also clear the "swap on next reset" functionality from the bootloader by running:

```
$ mlxbf-bootctl --nowatchdog-swap
```

Otherwise, next time the system is reset (via reboot, external reset, etc.) it assumes a failure or watchdog reset occurred and swaps the eMMC boot partition automatically.

LVFS and fwupd

Officially released bootloaders (ATF and UEFI) may be alternatively installed from the LVFS (Linux Vendor Firmware Service). LVFS is a free service operated by the Linux Foundation, which allows vendors to host stable firmware images for easy download and installation.

Note

BlueField must have a functioning connection to the Internet.

Interaction with LVFS is carried out through a standard open-source tool called fwupd. fwupd is an updater daemon that runs in the background, waiting for commands from a management application. fwupd and the command line manager, fwupdmgr, comes pre-installed on the BlueField Ubuntu image.

To verify bootloader support for a fwupd update, run the following command:

```
$ fwupdmgr get-devices
```

If "UEFI Device Firmware" device appears, then your currently installed bootloader supports the update process. Other devices may appear depending on your distribution of choice. Version numbers similar to 0.0.0.1 may appear if you are using an older version of the bootloader.

1. Before updating, a fresh list of release metadata must be obtained. Run:

   ```
   $ fwupdmgr refresh
   ```

2. Optionally, to confirm if a new release is available, run:

   ```
   $ fwupdmgr get-releases
   ```

3. Update your system bootloader, run "upgrade" with the GUID of the UEFI device. Run:

   ```
   $ fwupdmgr upgrade 39342586-4e0e-4833-b4ba-1256b0ffb471
   ```

This will upgrade the ATF and UEFI to the latest available stable version of the bootloader through a UEFI capsule update, without upgrading the root file system. If your system is already at the latest available version, this upgrade command will do nothing.

4. Reboot BlueField to complete the upgrade.

Note

Installing boot firmware directly through mlxbf-bootctl may cause fwupdmgr to detect an incorrect version string. If your workflow depends on fwupd, try to update the bootloader through capsule update (i.e. bfrec --capsule) or fwupdmgr only.

For more information about LVFS and fwupd, please refer to the official website of LVFS.

Updating Boot Partitions with BMC

The Arm cores notify the BMC prior to the reboot that an upgrade is about to happen. Software running on the BMC can then be implemented to watch the Arm cores after reboot. If after some time the BMC does not detect the Arm cores come up properly, it can use its USB debug connection to the Arm cores to properly reset the Arm cores. It first sets a suitable mode bit that the Arm bootloader responds to by switching the primary and alternating boot partitions as part of resetting into its original state.

Creating BlueField Boot File

The BlueField software distribution provides tools to format and to package the bootloader images into a single bootable file.

To create the BlueField boot file, use the `mlx-mkbfb` tool with the appropriate images. The bootloader images are embedded within the BSD under `<BF_INST_DIR>/boot/`. It is also possible to build the binary images from sources. Please refer to the following sections for further details.

1. First, set the PATH variable:

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

2. Then, generate the boot file by using the `mlx-mkbfb` command:

   ```
   $ mlx-mkbfb \ --bl2 bl2.bin \ --bl31 bl31.bin \ --bl33 BLUEFIELD_EFI.fd \ --boot-acpi "=default" \
   default.bfb
   ```

   This command creates the `default.bfb` from `bl2.bin`, `bl31.bin`, and `BLUEFILED_EFI.fd`. The generated file might be used to update the eMMC boot partitions.

To verify the content of the boot file, run:

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

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

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

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

UEFI Boot Management

The UEFI firmware provides boot management function that can be configured by modifying architecturally defined global variables which are stored in the UPVS EEPROM. The boot manager will attempt to load and boot the OS in an order defined by the persistent variables.

The UEFI boot manager can be configured; boot entries may be added or removed from the boot menu. The UEFI firmware can also effectively generate entries in this boot menu, according to the available network interfaces and possibly the disks attached to the system.

Boot Option

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

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

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

List UEFI Boot Options

To display the boot option already installed in the NVIDIA® BlueField® system, reboot and go to the UEFI menu screen. To get to the UEFI menu, hit Esc when prompted (in the RShim or UART console) before the countdown timer runs out.

```
Press <ESC> twice to enter UEFI menu
3 seconds remaining
2 seconds remaining
1 seconds remaining
```

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

```
�����������������������������������������
```

```
�                         Boot Manager                          �
���������������������������������������������������������
                                 Device Path :
  Boot Option Menu                          HD(1,GPT,B55B6B71-964E
                                 -714B-AAF8-7AE8D768372
  focal0                            7,0x800,0x19000)/\EFI\
  ubuntu                            ubuntu\shimaa64.efi
  Linux from rshim
  Linux from mmc0
  EFI Internal Shell
  EFI Misc Device
  EFI Network
  EFI Network 1
  EFI Network 2
  EFI Network 3
  EFI Network 4
  EFI Network 5
                               v
���������������������������������������������������������
�                                             �
� ^v=Move Highlight      <Enter>=Select Entry      Esc=Exit                 �
���������������������������������������������������������
```

It is also possible to retrieve more details about the boot entries. To do so, select "EFI Internal Shell" entry from the Boot Manager screen.

```
UEFI Interactive Shell v2.1
EDK II
UEFI v2.50 (EDK II, 0x00010000)
Mapping table
      FS1: Alias(s):F1:
          VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)
      FS0: Alias(s):HD0b:;BLK1:
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)/HD(1,GPT,3DCADB7E-BCCC-4897-A766-
3C070EDD)
     BLK0: Alias(s):
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)
     BLK2: Alias(s):
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)/HD(2,GPT,9E61E8B5-EC9C-4299-8A0B-
1B42E3DB)
```

```
Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
Shell>
```

From the UEFI shell, you may run the following command to display the option list:

```
Shell> bcfg boot dump -v
```

Here `-v` displays the option list with extra info including boot parameters. The following is an output example:

```
Option: 00. Variable: Boot0000
  Desc    - Linux from rshim
  DevPath - VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)/Image
  Optional- Y
  00000000: 63 00 6F 00 6E 00 73 00-6F 00 6C 00 65 00 3D 00  *c.o.n.s.o.l.e.=.*
  00000010: 74 00 74 00 79 00 41 00-4D 00 41 00 30 00 20 00  *t.t.y.A.M.A.0. .*
  00000020: 65 00 61 00 72 00 6C 00-79 00 63 00 6F 00 6E 00  *e.a.r.l.y.c.o.n.*
  00000030: 3D 00 70 00 6C 00 30 00-31 00 31 00 2C 00 30 00  *=.p.l.0.1.1.,.0.*
  00000040: 78 00 30 00 31 00 30 00-30 00 30 00 30 00 30 00  *x.0.1.0.0.0.0.0.*
  00000050: 30 00 20 00 20 00 69 00-6E 00 69 00 74 00 72 00  *0. . .i.n.i.t.r.*
  00000060: 64 00 3D 00 69 00 6E 00-69 00 74 00 72 00 61 00  *d.=.i.n.i.t.r.a.*
  00000070: 6D 00 66 00 73 00 00 00-                         *m.f.s...*
Option: 01. Variable: Boot0002
  Desc    - Yocto Poky
  DevPath - HD(1,GPT,3DCADB7E-BCCC-4897-A766-3C070EDD7C25,0x800,0xAE800)/Image
  Optional- Y
  00000000: 63 00 6F 00 6E 00 73 00-6F 00 6C 00 65 00 3D 00  *c.o.n.s.o.l.e.=.*
  00000010: 74 00 74 00 79 00 41 00-4D 00 41 00 30 00 20 00 *t.t.y.A.M.A.0. .*
  00000020: 65 00 61 00 72 00 6C 00-79 00 63 00 6F 00 6E 00  *e.a.r.l.y.c.o.n.*
  00000030: 3D 00 70 00 6C 00 30 00-31 00 31 00 2C 00 30 00  *=.p.l.0.1.1.,.0.*
  00000040: 78 00 30 00 31 00 30 00-30 00 30 00 30 00 30 00  *x.0.1.0.0.0.0.0.*
  00000050: 30 00 20 00 72 00 6F 00-6F 00 74 00 3D 00 2F 00  *0. .r.o.o.t.=./.*
  00000060: 64 00 65 00 76 00 2F 00-6D 00 6D 00 63 00 62 00  *d.e.v./.m.m.c.b.*
  00000070: 6C 00 6B 00 30 00 70 00-32 00 20 00 72 00 6F 00  *l.k.0.p.2. .r.o.*
  00000080: 6F 00 74 00 77 00 61 00-69 00 74 00              *o.t.w.a.i.t.*
    Option: 02. Variable: Boot0003
      Desc    - EFI Misc Device
      DevPath - VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)
```

```
     Optional- N
   Option: 03. Variable: Boot0004
     Desc    - EFI Network
     DevPath - MAC(001ACAFFFF01,0x1)
     Optional- N
   Option: 04. Variable: Boot0005
     Desc    - EFI Network 1
     DevPath - MAC(001ACAFFFF01,0x1)/IPv4(0.0.0.0)
     Optional- N
   Option: 05. Variable: Boot0006
     Desc    - EFI Network 2
     DevPath - MAC(001ACAFFFF01,0x1)/IPv6(0000:0000:0000:0000:0000:0000:0000:0000)
     Optional- N
   Option: 06. Variable: Boot0007
     Desc    - EFI Network 3
     DevPath - MAC(001ACAFFFF01,0x1)/IPv4(0.0.0.0)/Uri()
     Optional- N
   Option: 07. Variable: Boot0008
     Desc    - EFI Internal Shell
     DevPath - MemoryMapped(0xB,0xFE5FE000,0xFEAE357F)/FvFile(7C04A583-9E3E-4F1C-AD65-
E05268D0B4D1)
     Optional- N
```

Note

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

UEFI System Configuration

UEFI System Configuration menu can be accessed under UEFI menu   Device Manager   System Configuration.

The following options are supported:

- Set Password – set a password for UEFI. Default: No password.
- Select SPCR UART – choose UART for Port Console Redirection. Default: Disabled.
- Enable SMMU – enable SMMU in ACPI. Default: Disabled.
- Disable SPMI – disable/enable ACPI SPMI Table. Default: Enabled.
- Enable 2nd eMMC – this option is relevant only for some BlueField Reference Platform boards. Default: Disabled.
- Boot Partition Protection – enable eMMC boot partition so it can be updated by the UEFI capsule only
- Disable PCIe – disable PCIe in ACPI. Default: Enabled.
- Disable ForcePXERetry – if ForcePXE is enabled from the BMC, the boot process keeps retrying PXE boot if it fails unless this option is enabled. If ForcePXERetry is disabled, the boot process only attempts PXE boot once, then it retries the normal boot flow if all PXE boot entries fail.
- Reset EFI Variables – clears all EFI variables to factory default state and disables SMMU and wipes the BOOT option variables and secure boot keys
- Reset MFG Info – clears the manufacturing information

Note

All the above options, except for password and the two reset options, are also programmatically configurable via the BlueField Linux `/etc/bf.cfg`. Refer to section "bf.cfg Parameters" for further information.