Installation Troubleshooting and How-Tos

Table of contents

BlueField target is stuck inside UEFI menu

Upgrade to the latest stable boot partition images, see "How to upgrade the boot partition (ATF & UEFI) without re-installation".

BFB does not recognize the BlueField board type

If the .bfb file cannot recognize the BlueField board type, it reverts to low core operation. The following message will be printed on your screen:

```
***System type can't be determined***
***Booting as a minimal system***
```

Please contact NVIDIA Support if this occurs.

Unable to load BL2, BL2R, or PSC image

The following errors appear in console if images are corrupted or not signed properly:

Device	Error
BlueField	`ERROR: Failed to load BL2 firmware`
BlueField-2	`ERROR: Failed to load BL2R firmware`
BlueField-3	`Failed to load PSC-BL1` or `PSC VERIFY_BCT timeout`

CentOS fails into "dracut" mode during installation

This is most likely configuration related.

- If installing through the RShim interface, check whether /var/pxe/centos7 is mounted or not. If not, either manually mount it or re-run the setup.sh script.
- Check the Linux boot message to see whether eMMC is found or not. If not, the BlueField driver patch is missing. For local installation via RShim, run the setup.sh script with the absolute path and check if there are any errors. For a corporate PXE

server, make sure the BlueField and ConnectX driver disk are patched into the initrd image.

How to find the software versions of the running system

Run the following:

```
/opt/mellanox/scripts/bfvcheck:
root@bluefield:/usr/bin/bfvcheck# ./bfvcheck
Beginning version check...
-RECOMMENDED VERSIONS-
ATF: v1.5(release):BL2.0-1-gf9f7cdd
UEFI: 2.0-6004a6b
FW: 18.25.1010
-INSTALLED VERSIONS-
ATF: v1.5(release):BL2.0-1-gf9f7cdd
UEFI: 2.0-6004a6b
FW: 18.25.1010
Version checked
```

Also, the version information is printed to the console.

For ATF, a version string is printed as the system boots.

```
"NOTICE:  BL2: v1.3(release):v1.3-554-ga622cde"
```

For UEFI, a version string is printed as the system boots.

```
"UEFI firmware (version 0.99-18d57e3 built at 00:55:30 on Apr 13 2018)"
```

For Yocto, run:

```
$ cat /etc/bluefield_version
```

```
2.0.0.10817
```

How to upgrade the host RShim driver

See the readme at `<BF_INST_DIR>/src/drivers/rshim/README`.

How to upgrade the boot partition (ATF & UEFI) without re-installation

1. Boot the target through the RShim interface from a host machine:

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

2. Log into the BlueField target:

   ```
   $ /opt/mlnx/scripts/bfrec
   ```

How to upgrade ConnectX firmware from Arm side

The `mst`, `mlxburn`, and `flint` tools can be used to update firmware.

For Ubuntu, CentOS and Debian, run the following command from the Arm side:

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

How to configure ConnectX firmware

Configuring ConnectX firmware can be done using the mlxconfig tool.

It is possible to configure privileges of both the internal (Arm) and the external host (for DPUs) from a privileged host. According to the configured privilege, a host may or may not perform certain operations related to the NIC (e.g. determine if a certain host is allowed to read port counters).

For more information and examples please refer to the MFT User Manual which can be found at the following link.

How to use the UEFI boot menu

Press the "Esc" key when prompted after booting (before the countdown timer runs out) to enter the UEFI boot menu and use the arrows to select the menu option.

It could take 1-2 minutes to enter the Boot Manager depending on how many devices are installed or whether the EXPROM is programmed or not.

Once in the boot manager:

- "EFI Network xxx" entries with device path "PciRoot..." are ConnectX interface
- "EFI Network xxx" entries with device path "MAC(..." are for the RShim interface and the BlueField OOB Ethernet interface

Select the interface and press ENTER will start PXE boot.

The following are several useful commands under UEFI shell:

```
Shell> ls FS0:                          # display file
Shell> ls FS0:\EFI                        # display file
Shell> cls                              # clear screen
Shell> ifconfig -l                        # show interfaces
Shell> ifconfig -s eth0 dhcp                    # request DHCP
Shell> ifconfig -l eth0                      # show one interface
Shell> tftp 192.168.100.1 grub.cfg FS0:\grub.cfg    # tftp download a file
Shell> bcfg boot dump                        # dump boot variables
Shell> bcfg boot add 0 FS0:\EFI\centos\shim.efi "CentOS" # create an entry
```

How to Use the Kernel Debugger (KGDB)

The default Yocto kernel has CONFIG_KGDB and CONFIG_KGDB_SERIAL_CONSOLE enabled. This allows the Linux kernel on BlueField to be debugged over the serial port. A single serial port cannot be used both as a console and by KGDB at the same time. It is recommended to use the RShim for console access (/dev/rshim0/console) and the UART port (/dev/ttyAMA0 or /dev/ttyAMA1) for KGDB. Kernel GDB over console (KGDBOC) does not work over the RShim console. If the RShim console is not available, there are open-source packages such as KGDB demux and agent-proxy which allow a single serial port to be shared.

There are two ways to configure KGDBOC. If the OS is already booted, then write the name of the serial device to the KGDBOC module parameter. For example:

```
$ echo ttyAMA1 > /sys/module/kgdboc/parameters/kgdboc
```

To attach GDB to the kernel, it must be stopped first. One way to do that is to send a "g" to /proc/sysrq-trigger.

```
$ echo g > /proc/sysrq-trigger
```

To debug incidents that occur at boot time, kernel boot parameters must be configured. Add "kgdboc=ttyAMA1,115200 kgdwait" to the boot arguments to use UART1 for debugging and force it to wait for GDB to attach before booting.

Once the KGDBOC module is configured and the kernel stopped, run the Arm64 GDB on the host machine connected to the serial port, then set the remote target to the serial device on the host side.

```
<BF_INST_DIR>/sdk/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb
<BF_INST_DIR>/sample/vmlinux

(gdb) target remote /dev/ttyUSB3
Remote debugging using /dev/ttyUSB3
arch_kgdb_breakpoint () at /labhome/dwoods/src/bf/linux/arch/arm64/include/asm/kgdb.h:32
32          asm ("brk %0" : : "I" (KGDB_COMPILED_DBG_BRK_IMM));
(gdb)
```

<BF_INST_DIR> is the directory where the BlueField software is installed. It is assumed that the SDK has been unpacked in the same directory.

How to enable/disable SMMU

SMMU could affect performance for certain applications. By default, it is enabled on BlueField-3 and disabled on BlueField-2, and can be configured in different ways.

- Enable/disable SMMU in the UEFI System Configuration
- Set it in bf.cfg and push it together with the install.bfb (see section "Installing Popular Linux Distributions on BlueField")
- In BlueField Linux, create a file with one line with SYS_ENABLE_SMMU=TRUE, then run bfcfg.

The configuration change will take effect after reboot. The configuration value is stored in a persistent UEFI variable. It is not modified by OS installation.

See section "UEFI System Configuration" for information on how to access the UEFI System Configuration menu.

How to change the default console of the install image

On UART0:

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

On UART1:

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

On RShim:

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

How to change the default network configuration during BFB installation

On Ubuntu OS, the default network configuration for `tmfifo_net0` and `oob_net0` interfaces is set by the cloud-init service upon first boot after BFB installation.

The default content of `/var/lib/cloud/seed/nocloud-net/network-config` as follows:

```
# cat /var/lib/cloud/seed/nocloud-net/network-config
version: 2
renderer: NetworkManager
ethernets:
 tmfifo_net0:
  dhcp4: false
  addresses:
   - 192.168.100.2/30
  nameservers:
   addresses: [ 192.168.100.1 ]
  routes:
  - to: 0.0.0.0/0
   via: 192.168.100.1
   metric: 1025
 oob_net0:
  dhcp4: true
```

This content can be modified during BFB installation using `bf.cfg`. For example:

```
# cat bf.cfg
bfb_modify_os()
{
    sed -i -e '/oob_net0/,+1d' /mnt/var/lib/cloud/seed/nocloud-net/network-config
cat >> /mnt/var/lib/cloud/seed/nocloud-net/network-config << EOF
 oob_net0:
  dhcp4: false
  addresses:
```

```
  - 10.0.0.1/24
EOF
}

# bfb-install -c bf.cfg -r rshim0 -b <BFB>
```

Note

Using the same technique, any configuration file on the BlueField DPU side can be updated during the BFB installation process.

Sanitizing DPU eMMC and SSD Storage

During the BFB installation process, NVIDIA® BlueField® networking platform (DPU or SuperNIC) storage can be securely sanitized either using the `shred` or the `mmc` and `nvme` utilities in the `bf.cfg` configuration file as illustrated in the following subsections.

Note

By default, only the installation target storage is formatted using the Linux `mkfs` utility.

Using shred Utility

```
# cat bf.cfg
SANITIZE_DONE=${SANITIZE_DONE:-0}
export SANITIZE_DONE
if [ $SANITIZE_DONE -eq 0 ]; then
        sleep 3m
        /sbin/modprobe nvme
        if [ -e /dev/mmcblk0 ]; then
                echo Sanitizing /dev/mmcblk0 | tee /dev/kmsg
                echo Sanitizing /dev/mmcblk0 > /tmp/sanitize.emmc.log
                mmc sanitize /dev/mmcblk0 >>  /tmp/sanitize.emmc.log 2>&1
        fi
        if [ -e /dev/nvme0n1 ]; then
                echo Sanitizing /dev/nvme0n1 | tee /dev/kmsg
                echo Sanitizing /dev/nvme0n1 > /tmp/sanitize.ssd.log
                nvme sanitize /dev/nvme0n1 -a 2 >> /tmp/sanitize.ssd.log 2>&1
                nvme sanitize-log /dev/nvme0n1 >> /tmp/sanitize.ssd.log 2>&1
        fi
        SANITIZE_DONE=1
        echo ===================== sanitize.log ===================== | tee /dev/kmsg
        cat /tmp/sanitize.*.log | tee /dev/kmsg
        sync
fi
bfb_modify_os()
{
        echo ===================== bfb_modify_os ===================== | tee /dev/kmsg
        if ( /bin/ls -1 /tmp/sanitize.*.log > /dev/null 2>&1 ); then
                cat /tmp/sanitize.*.log > /mnt/root/sanitize.log
        fi
}
```

Using mmc and nvme Utilities

```
# cat bf.cfg
SANITIZE_DONE=${SANITIZE_DONE:-0}
export SANITIZE_DONE
if [ $SANITIZE_DONE -eq 0 ]; then
        sleep 3m
        /sbin/modprobe nvme

        if [ -e /dev/mmcblk0 ]; then
                echo Sanitizing /dev/mmcblk0 | tee /dev/kmsg
                echo Sanitizing /dev/mmcblk0 > /tmp/sanitize.emmc.log
                mmc sanitize /dev/mmcblk0 >>  /tmp/sanitize.emmc.log 2>&1
        fi
        if [ -e /dev/nvme0n1 ]; then
                echo Sanitizing /dev/nvme0n1 | tee /dev/kmsg
        echo Sanitizing /dev/nvme0n1 > /tmp/sanitize.ssd.log
        nvme sanitize /dev/nvme0n1 -a 2 >> /tmp/sanitize.ssd.log 2>&1
        nvme sanitize-log /dev/nvme0n1 >> /tmp/sanitize.ssd.log 2>&1
    fi
    SANITIZE_DONE=1
    echo ===================== sanitize.log ===================== | tee /dev/kmsg
    cat /tmp/sanitize.*.log | tee /dev/kmsg
    sync
fi
bfb_modify_os()
{
    echo ===================== bfb_modify_os ===================== | tee /dev/kmsg
    if ( /bin/ls -1 /tmp/sanitize.*.log > /dev/null 2>&1 ); then
        cat /tmp/sanitize.*.log > /mnt/root/sanitize.log
    fi
}
```