Management

Table of contents

- Performance Monitoring Counters
- Intelligent Platform Management Interface
- Redfish
- Logging
- SoC Management Interface
- BlueField OOB Ethernet Interface

Performance Monitoring Counters

The performance modules in NVIDIA® BlueField® are present in several hardware blocks and each block has a certain set of supported events.

The mlx_pmc driver provides access to all of these performance modules through a sysfs interface. The driver creates a directory under /sys/class/hwmon under which each of the blocks explained above has a subdirectory. Please note that all directories under /sys/class/hwmon are named as "hwmon<N>" where N is the hwmon device number corresponding to the device. This is assigned by Linux and could change with the addition of more devices to the hwmon class. Each hwmon directory has a "name" node which can be used to identify the correct device. In this case, reading the "name" file should return "bfperf".

The hardware blocks that include performance modules are:

- Tile (block containing 2 cores and a shared L2 cache) has 2 sets of counters, one set for HNF and HNF_NET events. These are present as "tile" and "tilenet" directories in the sysfs interface of the driver.
- TRIO (PCIe root complex) has 3 sets of counters, one each for TRIO, SMGEN and PCIE TLR events. The sysfs directories for these are called "trio", "triogen" and "pcie" respectively.
- MSS (memory sub-system containing the memory controller and L3 cache)
- GIC and SMMU with one set of counters each for the SMGEN events. These are simply labelled "gic" and "smmu" respectively.

The number of Tile, TRIO and MSS blocks depends on the system. There is a maximum of 8 Tile, 3 TRIO and 2 MSS blocks in BlueField, and this is added as a suffix to the sysfs directory names. For example, this is a list of directories present in a BlueField-2 system:

```
ubuntu@bf:/$ ls /sys/class/hwmon/hwmon0/
```

```
device l3cachehalf0 pcie0 smmu0 tile1 tilenet0 tilenet3 triogen0
ecc l3cachehalf1 pcie1 subsystem tile2 tilenet1 trio0 triogen1
gic0 name power  tile0 tile3 tilenet2 trio1 uevent
```

The PCIe TLR statistics for each TRIO are under the "pcie" block.

Performance Data Collection Mechanisms

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

1. Programming hardware counters to monitor specific events
2. Reading registers that hold performance/event statistics

All blocks except "ecc" and "pcie" use the mechanism 1.

Using Hardware Counters

For blocks that use hardware counters to collect data, each counter present in the block is represented by "event<N>" and "counter<N>" sysfs files.

For example:

```
ubuntu@bf:/$ ls /sys/class/hwmon/hwmon0/tile0/
counter0 counter1 counter2 counter3 event0 event1 event2 event3 event_list
```

An event<N> and counter<N> pair can be used to program and monitor events. The "event_list" sysfs file displays the list of events supported by that block along with the hexadecimal value corresponding to each event.

Use the echo command to write the event number to the event<N> file, and use the cat command to read the counter value from the corresponding counter (counter<N>).

The counters are enabled individually once the event number is written to the corresponding event file. However, the L3 cache performance counters cannot be enabled or disabled individually and can only be triggered or stopped all at the same time.

So in the example provided, all 4 event files may be programmed with the necessary event numbers and then the "enable" file may be used to start the counters. Writing `0` to the enable file stops the counters while `1` starts them.

Reading Registers

For "ecc" and "pcie" blocks, the counters cannot be started or stopped by the user, instead the statistics are automatically collected by HW and stored in registers. These register names are exposed within the directory and can be read by the user at any time.

List of Supported Events

SMGEN Performance Module

Hex Value	Name	Description
0x0	AW_REQ	Reserved for internal use
0x1	AW_BEATS	Reserved for internal use
0x2	AW_TRANS	Reserved for internal use
0x3	AW_RESP	Reserved for internal use
0x4	AW_STL	Reserved for internal use
0x5	AW_LAT	Reserved for internal use
0x6	AW_REQ_TBU	Reserved for internal use
0x8	AR_REQ	Reserved for internal use
0x9	AR_BEATS	Reserved for internal use
0xa	AR_TRANS	Reserved for internal use
0xb	AR_STL	Reserved for internal use
0xc	AR_LAT	Reserved for internal use
0xd	AR_REQ_TBU	Reserved for internal use

Hex Value	Name	Description
0xe	TBU_MISS	The number of TBU miss
0xf	TX_DAT_AF	Mesh Data channel write FIFO almost Full. This is from the TRIO toward the Arm memory.
0x10	RX_DAT_AF	Mesh Data channel read FIFO almost Full. This is from the Arm memory toward the TRIO.
0x11	RETRYQ_CRED	Reserved for internal use

Tile HNF Performance Module

Hex Value	Name	Description
0x45	HNF_REQUESTS	Number of REQs that were processed in HNF
0x46	HNF_REJECTS	Reserved for internal use
0x47	ALL_BUSY	Reserved for internal use
0x48	MAF_BUSY	Reserved for internal use
0x49	MAF_REQUESTS	Reserved for internal use
0x4a	RNF_REQUESTS	Number of REQs sent by the RN-F selected by HNF_PERF_CTL register RNF_SEL field
0x4b	REQUEST_TYPE	Reserved for internal use
0x4c	MEMORY_READS	Number of reads to MSS
0x4d	MEMORY_WRITES	Number of writes to MSS
0x4e	VICTIM_WRITE	Number of victim lines written to memory
0x4f	POC_FULL	Reserved for internal use
0x50	POC_FAIL	Number of times that the POC Monitor sent RespErr Okay

Hex Value	Name	Description
		status to an Exclusive WriteNoSnp or CleanUnique REQ
0x51	POC_SUCCESS	Number of times that the POC Monitor sent RespErr ExOkay status to an Exclusive WriteNoSnp or CleanUnique REQ
0x52	POC_WRITES	Number of Exclusive WriteNoSnp or CleanUnique REQs processed by POC Monitor
0x53	POC_READS	Number of Exclusive ReadClean/ReadShared REQs processed by POC Monitor
0x54	FORWARD	Reserved for internal use
0x55	RXREQ_HNF	Reserved for internal use
0x56	RXRSP_HNF	Reserved for internal use
0x57	RXDAT_HNF	Reserved for internal use
0x58	TXREQ_HNF	Reserved for internal use
0x59	TXRSP_HNF	Reserved for internal use
0x5a	TXDAT_HNF	Reserved for internal use
0x5b	TXSNP_HNF	Reserved for internal use
0x5c	INDEX_MATCH	Reserved for internal use
0x5d	A72_ACCESS	Access requests (Reads, Writes, CopyBack, CMO, DVM) from A72 clusters
0x5e	IO_ACCESS	Accesses requests (Reads, Writes) from DMA IO devices
0x5f	TSO_WRITE	Total Store Order write Requests from DMA IO devices
0x60	TSO_CONFLICT	Reserved for internal use
0x61	DIR_HIT	Requests that hit in directory
0x62	HNF_ACCEPTS	Reserved for internal use
0x63	REQ_BUF_EMPTY	Number of cycles when request buffer is empty
0x64	REQ_BUF_IDLE_MAF	Reserved for internal use

Hex Value	Name	Description
0x65	TSO_NOARB	Reserved for internal use
0x66	TSO_NOARB_CYCLES	Reserved for internal use
0x67	MSS_NO_CREDIT	Number of cycles that a Request could not be sent to MSS due to lack of credits
0x68	TXDAT_NO_LCRD	Reserved for internal use
0x69	TXSNP_NO_LCRD	Reserved for internal use
0x6a	TXRSP_NO_LCRD	Reserved for internal use
0x6b	TXREQ_NO_LCRD	Reserved for internal use
0x6c	TSO_CL_MATCH	Reserved for internal use
0x6d	MEMORY_READS_BYPASS	Number of reads to MSS that bypass Home Node
0x6e	TSO_NOARB_TIMEOUT	Reserved for internal use
0x6f	ALLOCATE	Number of times that Directory entry was allocated
0x70	VICTIM	Number of times that Directory entry allocation did not find an Invalid way in the set
0x71	A72_WRITE	Write requests from A72 clusters
0x72	A72_Read	Read requests from A72 clusters
0x73	IO_WRITE	Write requests from DMA IO devices
0x74	IO_Reads	Read requests from DMA IO devices
0x75	TSO_Reject	Reserved for internal use
0x80	TXREQ_RN	Reserved for internal use
0x81	TXRSP_RN	Reserved for internal use

Hex Value	Name	Description
0x82	TXDAT_RN	Reserved for internal use
0x83	RXSNP_RN	Reserved for internal use
0x84	RXRSP_RN	Reserved for internal use
0x85	RXDAT_RN	Reserved for internal use

TRIO Performance Module

Hex Value	Name	Description
0xa0	TPIO_DATA_BEAT	Data beats from Arm PIO to TRIO
0xa1	TDMA_DATA_BEAT	Data beats from Arm memory to PCI completion
0xa2	MAP_DATA_BEAT	Reserved for internal use
0xa3	TXMSG_DATA_BEAT	Reserved for internal use
0xa4	TPIO_DATA_PACKET	Data packets from Arm PIO to TRIO
0xa5	TDMA_DATA_PACKET	Data packets from Arm memory to PCI completion
0xa6	MAP_DATA_PACKET	Reserved for internal use
0xa7	TXMSG_DATA_PACKET	Reserved for internal use
0xa8	TDMA_RT_AF	The in-flight PCI DMA READ request queue is almost full
0xa9	TDMA_PBUF_MAC_AF	Indicator of the buffer of Arm memory reads is too full awaiting PCIe access
0xaa	TRIO_MAP_WRQ_BUF_EMPTY	PCIe write transaction buffer is empty
0xab	TRIO_MAP_CPL_BUF_EMPTY	Arm PIO request completion queue is empty
0xac	TRIO_MAP_RDQ0_BUF_EMPTY	The buffer of MAC0's read transaction is empty

Hex Value	Name	Description
0xad	TRIO_MAP_RDQ1_BUF_EMPTY	The buffer of MAC1's read transaction is empty
0xae	TRIO_MAP_RDQ2_BUF_EMPTY	The buffer of MAC2's read transaction is empty
0xaf	TRIO_MAP_RDQ3_BUF_EMPTY	The buffer of MAC3's read transaction is empty
0xb0	TRIO_MAP_RDQ4_BUF_EMPTY	The buffer of MAC4's read transaction is empty
0xb1	TRIO_MAP_RDQ5_BUF_EMPTY	The buffer of MAC5's read transaction is empty
0xb2	TRIO_MAP_RDQ6_BUF_EMPTY	The buffer of MAC6's read transaction is empty
0xb3	TRIO_MAP_RDQ7_BUF_EMPTY	The buffer of MAC7's read transaction is empty

L3 Cache Performance Module

> **Note**
>
> The L3 cache interfaces with the Arm cores via the SkyMesh. The CDN is used for control data. The NDN is used for responses. The DDN is for the actual data transfer.

Hex Value	Name	Description
0x00	DISABLE	Reserved for internal use
0x01	CYCLES	Timestamp counter

Hex Value	Name	Description
0x02	TOTAL_RD_REQ_IN	Read Transaction control request from the CDN of the SkyMesh
0x03	TOTAL_WR_REQ_IN	Write transaction control request from the CDN of the SkyMesh
0x04	TOTAL_WR_DBID_ACK	Write transaction control responses from the NDN of the SkyMesh
0x05	TOTAL_WR_DATA_IN	Write transaction data from the DDN of the SkyMesh
0x06	TOTAL_WR_COMP	Write completion response from the NDN of the SkyMesh
0x07	TOTAL_RD_DATA_OUT	Read transaction data from the DDN
0x08	TOTAL_CDN_REQ_IN_BANK0	CHI CDN Transactions Bank 0
0x09	TOTAL_CDN_REQ_IN_BANK1	CHI CDN Transactions Bank 1
0x0a	TOTAL_DDN_REQ_IN_BANK0	CHI DDN Transactions Bank 0
0x0b	TOTAL_DDN_REQ_IN_BANK1	CHI DDN Transactions Bank 1
0x0c	TOTAL_EMEM_RD_RES_IN_BANK0	Total EMEM Read Response Bank 0
0x0d	TOTAL_EMEM_RD_RES_IN_BANK1	Total EMEM Read Response Bank 1
0x0e	TOTAL_CACHE_RD_RES_IN_BANK0	Total Cache Read Response Bank 0
0x0f	TOTAL_CACHE_RD_RES_IN_BANK1	Total Cache Read Response Bank 1
0x10	TOTAL_EMEM_RD_REQ_BANK0	Total EMEM Read Request Bank 0

Hex Value	Name	Description
0x11	TOTAL_EMEM_RD_REQ_BANK1	Total EMEM Read Request Bank 1
0x12	TOTAL_EMEM_WR_REQ_BANK0	Total EMEM Write Request Bank 0
0x13	TOTAL_EMEM_WR_REQ_BANK1	Total EMEM Write Request Bank 1
0x14	TOTAL_RD_REQ_OUT	EMEM Read Transactions Out
0x15	TOTAL_WR_REQ_OUT	EMEM Write Transactions Out
0x16	TOTAL_RD_RES_IN	EMEM Read Transactions In
0x17	HITS_BANK0	Number of Hits Bank 0
0x18	HITS_BANK1	Number of Hits Bank 1
0x19	MISSES_BANK0	Number of Misses Bank 0
0x1a	MISSES_BANK1	Number of Misses Bank 1
0x1b	ALLOCATIONS_BANK0	Number of Allocations Bank 0
0x1c	ALLOCATIONS_BANK1	Number of Allocations Bank 1
0x1d	EVICTIONS_BANK0	Number of Evictions Bank 0
0x1e	EVICTIONS_BANK1	Number of Evictions Bank 1
0x1f	DBID_REJECT	Reserved for internal use
0x20	WRDB_REJECT_BANK0	Reserved for internal use
0x21	WRDB_REJECT_BANK1	Reserved for internal use
0x22	CMDQ_REJECT_BANK0	Reserved for internal use
0x23	CMDQ_REJECT_BANK1	Reserved for internal use
0x24	COB_REJECT_BANK0	Reserved for internal use
0x25	COB_REJECT_BANK1	Reserved for internal use
0x26	TRB_REJECT_BANK0	Reserved for internal use
0x27	TRB_REJECT_BANK1	Reserved for internal use

Hex Value	Name	Description
0x28	TAG_REJECT_BANK0	Reserved for internal use
0x29	TAG_REJECT_BANK1	Reserved for internal use
0x2a	ANY_REJECT_BANK0	Reserved for internal use
0x2b	ANY_REJECT_BANK1	Reserved for internal use

PCIe TLR Statistics

Hex Value	Name	Description
0x0	PCIE_TLR_IN_P_PKT_CNT	Incoming posted packets
0x10	PCIE_TLR_IN_NP_PKT_CNT	Incoming non-posted packets
0x18	PCIE_TLR_IN_C_PKT_CNT	Incoming completion packets
0x20	PCIE_TLR_OUT_P_PKT_CNT	Outgoing posted packets
0x28	PCIE_TLR_OUT_NP_PKT_CNT	Outgoing non-posted packets
0x30	PCIE_TLR_OUT_C_PKT_CNT	Outgoing completion packets
0x38	PCIE_TLR_IN_P_BYTE_CNT	Incoming posted bytes
0x40	PCIE_TLR_IN_NP_BYTE_CNT	Incoming non-posted bytes
0x48	PCIE_TLR_IN_C_BYTE_CNT	Incoming completion bytes
0x50	PCIE_TLR_OUT_C_BYTE_CNT	Outgoing posted bytes
0x58	PCIE_TLR_OUT_NP_BYTE_CNT	Outgoing non-posted bytes
0x60	PCIE_TLR_OUT_C_BYTE_CNT	Outgoing completion bytes

Tile HNFNET Performance Module

Hex Value	Name	Description
0x12	CDN_REQ	The number of CDN requests
0x13	DDN_REQ	The number of DDN requests
0x14	NDN_REQ	The number of NDN requests
0x15	CDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the CDN network
0x16	CDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the CDN network
0x17	CDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the CDN network
0x18	CDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the CDN network
0x19	CDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the CDN network
0x1a	CDN_DIAG_N_EGRESS	Packets sent out from north port in the CDN network
0x1b	CDN_DIAG_S_EGRESS	Packets sent out from south port in the CDN network
0x1c	CDN_DIAG_E_EGRESS	Packets sent out from east port in the CDN network
0x1d	CDN_DIAG_W_EGRESS	Packets sent out from west port in the CDN network
0x1e	CDN_DIAG_C_EGRESS	Packets sent out from core port in the CDN network
0x1f	CDN_DIAG_N_INGRESS	Packets received by north port in the CDN network
0x20	CDN_DIAG_S_INGRESS	Packets received by south port in the CDN network
0x21	CDN_DIAG_E_INGRESS	Packets received by east port in the CDN network

Hex Value	Name	Description
0x22	CDN_DIAG_W_INGRESS	Packets received by west port in the CDN network
0x23	CDN_DIAG_C_INGRESS	Packets received by core port in the CDN network
0x24	CDN_DIAG_CORE_SENT	Packets completed from core port in the CDN network
0x25	DDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the DDN network
0x26	DDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the DDN network
0x27	DDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the DDN network
0x28	DDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the DDN network
0x29	DDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the DDN network
0x2a	DDN_DIAG_N_EGRESS	Packets sent out from north port in the DDN network
0x2b	DDN_DIAG_S_EGRESS	Packets sent out from south port in the DDN network
0x2c	DDN_DIAG_E_EGRESS	Packets sent out from east port in the DDN network
0x2d	DDN_DIAG_W_EGRESS	Packets sent out from west port in the DDN network
0x2e	DDN_DIAG_C_EGRESS	Packets sent out from core port in the DDN network
0x2f	DDN_DIAG_N_INGRESS	Packets received by north port in the DDN network
0x30	DDN_DIAG_S_INGRESS	Packets received by south port in the DDN network

Hex Value	Name	Description
0x31	DDN_DIAG_E_INGRESS	Packets received by east port in the DDN network
0x32	DDN_DIAG_W_INGRESS	Packets received by west port in the DDN network
0x33	DDN_DIAG_C_INGRESS	Packets received by core port in the DDN network
0x34	DDN_DIAG_CORE_SENT	Packets completed from core port in the DDN network
0x35	NDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the NDN network
0x36	NDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the NDN network
0x37	NDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the NDN network
0x38	NDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the NDN network
0x39	NDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the NDN network
0x3a	NDN_DIAG_N_EGRESS	Packets sent out from north port in the NDN network
0x3b	NDN_DIAG_S_EGRESS	Packets sent out from south port in the NDN network
0x3c	NDN_DIAG_E_EGRESS	Packets sent out from east port in the NDN network
0x3d	NDN_DIAG_W_EGRESS	Packets sent out from west port in the NDN network
0x3e	NDN_DIAG_C_EGRESS	Packets sent out from core port in the NDN network
0x3f	NDN_DIAG_N_INGRESS	Packets received by north port in the NDN network

Hex Value	Name	Description
0x40	NDN_DIAG_S_INGRESS	Packets received by south port in the NDN network
0x41	NDN_DIAG_E_INGRESS	Packets received by east port in the NDN network
0x42	NDN_DIAG_W_INGRESS	Packets received by west port in the NDN network
0x43	NDN_DIAG_C_INGRESS	Packets received by core port in the NDN network
0x44	NDN_DIAG_CORE_SENT	Packets completed from core port in the NDN network

Programming Counter to Monitor Events

To program a counter to monitor one of the events from the event list, the event name or number needs to be written to the corresponding event file.

Let us call the /sys/class/hwmon/hwmon<N> folder corresponding to this driver as BFPERF_DIR.

For example, to monitor the event HNF_REQUESTS (0x45) on tile2 using counter 3:

```
$ echo 0x45 > <BFPERF_DIR>/tile2/event3
```

Or:

```
$ echo HNF_REQUESTS > <BFPERF_DIR>/tile2/event3
```

Once this is done, counter3 resets the counter and starts monitoring the number of HNF_REQUESTS.

To read the counter value, run:

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

To see what event is currently being monitored by a counter, just read the corresponding event file to get the event name and number.

```
$ cat <BFPERF_DIR>/tile2/event3
```

In this case, reading the event3 file returns "0x45: HNF_REQUESTS".

To clear the counter, write 0 to the counter file.

```
$ echo 0 > <BFPERF_DIR>/tile2/counter3
```

This resets the accumulator and the counter continues monitoring the same event that has previously been programmed, but starts the count from 0 again. Writing non-zero values to the counter files is not allowed.

To stop monitoring an event, write 0xff to the corresponding event file.

This is slightly different for the l3cache blocks due to the restriction that all counters can only be enabled, disabled, or reset together. So once the event is written to the event file, the counters will have to be enabled to start monitoring their respective events by writing "1" to the "enable" file. Writing "0" to this file will stop all the counters. The most reliable way to get accurate counter values would be by disabling the counters after a certain time period and then proceeding to read the counter values.

> **Note**
>
> Programming a counter to monitor a new event automatically stops all the counters. Also, enabling the counters resets the counters to 0 first.

For blocks that have performance statistics registers (mechanism 2), all of these statistics are directly made available to be read or reset.

For example, to read the number of incoming posted packets to TRIO2:

```
$ cat <BFPERF_DIR>/pcie2/IN_P_PKT_CNT
```

The count can be reset to 0 by writing 0 to the same file. Again, non-zero writes to these files are not allowed.

Intelligent Platform Management Interface

IPMB requests can be initiated in 2 directions:

- BlueField BMC-to-BlueField
- BlueField-to-BlueField BMC

Note

The NVIDIA® BlueField® networking platform's (DPU or SuperNIC) ipmb_dev_int driver is registered at the 7-bit I^2C address 0x30 by default. The I^2C address of the BlueField can be changed in the file /usr/bin/set_emu_param.sh.

- BlueField Controller cards provide connection from the host server BMC to BlueField Arm I^2C bus
- BlueField devices provide connection from the host server BMC to the BlueField NC-SI port
- BlueField Reference Platforms provide connection from its on-board BMC to BlueField Arm I^2C bus

BlueField BMC IPMI Commands

The BlueField BMC is able to retrieve data from BlueField software over its Intelligent Platform Management Bus (IPMB).

The BlueField BMC may request information about itself using the following command format:

```
$ ipmitool <ipmitool command>
```

Issue a command with the following format from the BlueField BMC to retrieve information from the BlueField:

```
ipmitool -I ipmb <ipmitool command>
```

The following table provides a list of supported ipmitool command arguments:

Command Description	Ipmitool Command	Relevant IPMI 2.0 Rev 1.1 Spec Section
Get device ID	mc info	20.1
Broadcast “Get Device ID”	Part of "mc info"	20.9
Get BMC global enables	mc getenables	22.2
Get device SDR info	sdr info	35.2
Get device SDR	"sdr get", "sdr list" or "sdr elist"	35.3
Get sensor hysteresis	sdr get <sensor-id>	35.7
Set sensor threshold	sensor thresh <sensor-id> <threshold> <setting> • sensor-id – name of the sensor for which a threshold is to be set	35.8

Command Description	Ipmitool Command	Relevant IPMI 2.0 Rev 1.1 Spec Section

	• threshold – which threshold to set
	  ◦ ucr – upper critical
	  ◦ unc – upper non-critical
	  ◦ lnc – lower non-critical
	  ◦ lcr – lower critical
	• setting – the value to set the threshold to
	
	To configure all lower thresholds, use : sensor thresh <sensor-id> lower <lnr> <lcr> <lnc>
	
	Note
	The lower non-recoverable <lnr> option is not supported
	
	To configure all upper thresholds, use: sensor thresh <sensor-id> upper <unc> <ucr> <unr>
	
	Note
	The upper non-recoverable <unr> option is not supported	

Get sensor threshold	sdr get <sensor-id>	35.9
Get sensor event enable	sdr get <sensor-id>	35.11
Get sensor reading	sensor reading <sensor-id>	35.14

Command Description	Ipmitool Command	Relevant IPMI 2.0 Rev 1.1 Spec Section
Get sensor type	sdr type <type>	35.16
Read FRU data	fru read <fru-number> <file-to-write-to>	34.2
Get SDR repository info	sdr info	33.9
Get SEL info	"sel" or "sel info"	40.2
Get SEL allocation info	"sel" or "sel info"	40.3
Get SEL entry	"sel list" or "sel elist"	40.5
Add SEL entry	sel add <filename>	40.6
Delete SEL entry	sel delete <id>	40.8
Clear SEL	sel clear	40.9
Get SEL time	sel time get	40.1
Set SEL time	sel time set "MM/DD/YYYY HH:M:SS"	40.11

List of IPMI Supported Sensors

Sensor	ID	Description
bluefield_temp	0	Support NIC monitoring of BlueField's temperature
ddr0_0_temp [1]	1	Support monitoring of DDR0 temp (on memory controller 0)
ddr0_1_temp [1]	2	Support monitoring of DDR1 temp (on memory controller 0)
ddr1_0_temp [1]	3	Support monitoring of DDR0 temp (on memory controller 1)
ddr1_1_temp [1]	4	Support monitoring of DDR1 temp (on memory controller 1)

Sensor	ID	Description
p0_temp	5	Port 0 temperature
p1_temp	6	Port 1 temperature
p0_link	7	Port0 link status
p1_link	8	Port1 link status

1. On BlueField-2 and BlueField-3 based boards, DDR sensors and FRUs are not supported. They will appear as no reading.

List of IPMI Supported FRUs

FRU	ID	Description
update_timer	0	set_emu_param.service is responsible for collecting data on sensors and FRUs every 3 seconds. This regular update is required for sensors but not for FRUs whose content is less susceptible to change. update_timer is used to sample the FRUs every hour instead. Users may need this timer in the case where they are issuing several raw IPMItool FRU read commands. This helps in assessing how much time users have to retrieve large FRU data before the next FRU update. update_timer is a hexadecimal number.
fw_info	1	NVIDIA® ConnectX® firmware information, Arm firmware version, and MLNX_OFED version. The fw_info is in ASCII format.
nic_pci_dev_info	2	NIC vendor ID, device ID, subsystem vendor ID, and subsystem device ID. The nic_pci_dev_info is in ASCII format.
cpuinfo	3	CPU information reported in lscpu and /proc/cpuinfo. The cpuinfo is in ASCII format.
ddr0_0_spd2	4	FRU for SPD MC0 DIMM 0 (MC = memory controller). The ddr0_0_spd is in binary format.

FRU	ID	Description
ddr0_1_spd2	5	FRU for SPD MC0 DIMM1. The ddr0_1_spd is in binary format.
ddr1_0_spd2	6	FRU for SPD MC1 DIMM0. The ddr1_0_spd is in binary format.
ddr1_1_spd2	7	FRU for SPD MC1 DIMM1. The ddr1_1_spd is in binary format.
emmc_info	8	eMMC size, list of its partitions, and partitions usage (in ASCII format). eMMC CID, CSD, and extended CSD registers (in binary format). The ASCII data is separated from the binary data with "StartBinary" marker.
qsfp0_eeprom	9	FRU for QSFP 0 EEPROM page 0 content (256 bytes in binary format)
qsfp1_eeprom	10	FRU for QSFP 1 EEPROM page 0 content (256 bytes in binary format)

ip_addresses	11	This FRU file can be used to write the BMC port 0 and port 1 IP addresses to the BlueField. It is empty to begin with.
		The file passed through the ipmitool fru write 11 <file> command must have the following format:
		`BMC: XXX.XXX.XXX.XXX`
		`P0: XXX.XXX.XXX.XXX`
		`P1: XXX.XXX.XXX.XXX`
		The size of the written file should be exactly 61 bytes.

dimms_ce_ue	12	FRU reporting the number of correctable and uncorrectable errors in the DIMMs. This FRU is updated once every 3 seconds.

FRU	ID	Description
eth0	13	Network interface 0 information. Updated once every minute.
eth1	14	Network interface 1 information. Updated once every minute.
bf_uid	15	BlueField UID
eth_hw_counters	16	List of ConnectX interface hardware counters

1. On BlueField-2 and BlueField-3 based boards, DDR sensors and FRUs are not supported. They will appear as no reading.

BlueField IPMI Commands

The BlueField is able to retrieve data from the BlueField BMC over IPMB.

Issue a command with the following format from the BlueField to retrieve information from the BMC:

```
$ ipmitool <ipmitool command>
```

The BlueField may request information about itself using the following command format:

```
$ ipmitool -U ADMIN -P ADMIN -p 9001 -H localhost <ipmitool command>
```

Note

The ipmb_host driver allows the BlueField to send requests to the BMC. Once set_emu_param.service is started, it will try to load the ipmb_host drivers. If the BMC is down or not responsive when BlueField tries to load the ipmb_host driver, the latter will not load successfully. In that case, make sure the BMC is up and operational, and run the following from BlueField's console:

```
echo 0x1011 > /sys/bus/i2c/devices/i2c-2/delete_device
rmmod ipmb_host
```

The set_emu_param.service script will try to load the driver again.

I2C Addresses for BMC-initiated Requests

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

I2C Addresses for BlueField-initiated Requests

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

Changing I2C Addresses

To use a different BlueField or BMC I^2C address, you must make changes to the following files' variables.

Filename Path	Parameter Change
/usr/bin/set_emu_param.sh	The ipmb_dev_int and ipmb_host drivers are registered at the following I^2C addresses: • IPMB_DEV_INT_ADD=<BlueField I2C Address 1> • IPMB_HOST_ADD=<BlueField I2C Address 2> These addresses must be different from one another. Otherwise, one of the drives will fail to register. To change the BMC I^2C address: IPMB_HOST_CLIENTADDR=<BMC I2C Address> <I2C Address> must be equal to: 0x1000+<7-bit I2C address>

External Host IPMI Commands

It is possible for the external host to retrieve data from the BlueField via the IPMI LAN interface (either OOB or ConnectX).

To do that:

1. Set the network interface address properly in progconf. For example, if the OOB IP address is 192.168.101.2, edit the OOB_IP variable in the /etc/ipmi/progconf file as follows:

```
root@localhost:~# cat /etc/ipmi/progconf
SUPPORT_IPMB="NONE"
LOOP_PERIOD=3
BF_FAMILY=$(/usr/bin/bffamily | tr -d '[:space:]')
OOB_IP="192.168.101.2"
```

2. Then reboot or restart the IPMI service as follows:

```
systemctl restart mlx_ipmid
```

3. To get information from the BlueField, issue commands from the external host in the following format:

```
ipmitool -I lanplus -H 192.168.101.2 -U ADMIN -P ADMIN <ipmitool command>
```

Loading and Using IPMI on BlueField Running CentOS

1. Load the BlueField CentOS image:

> **Note**
>
> The following steps are performed from the BlueField CentOS prompt. The BlueField is running CentOS 7.6 with kernel 5.4. The CentOS installation was done using the CentOS everything ISO image.
>
> The following drivers need to be loaded on the BlueField running CentOS:
>
> - jc42.ko
> - ee1004.ko
> - at24.ko
> - eeprom.ko
> - i2c-dev.ko

Example of loading ee1004.ko, at24.ko, and eeprom.ko:

```
modprobe ee1004
modprobe at24
modprobe eeprom
```

> **Info**
>
> The i2c-dev module is built into the kernel 5.4.60 on CentOS 7.6.

2. (Optional) Update the i2c-mlx driver if the installed version is older than i2c-mlx-1.0-0.gab579c6.src.rpm.

    1. Re-compile i2c-mlx. Run:

    ```
    $ yum remove -y kmod-i2c-mlx
    $ modprobe -rv i2c-mlx
    ```

    2. Transfer the i2c-mlx RPM from the BlueField software tarball under distro/SRPM onto the Arm. Run:

    ```
    $ rpmbuild --rebuild /root/i2c-mlx-1.0-0.g422740c.src.rpm
    $ yum install -y /root/rpmbuild/RPMS/aarch64/i2c-mlx-1.0-0.g422740c_5.4.17_mlnx.9.ga0bea68.aarch64.rpm
    $ ls -l /lib/modules/$(uname -r)/extra/i2c-mlx/i2c-mlx.ko
    ```

    3. Load i2c-mlx. Run:

    ```
    $ modprobe i2c-mlx
    ```

3. Install the following packages:

```
$ yum install ipmitool lm_sensors
```

If the above operation fails for ipmitool, run the following to install it:

```
wget http://sourceforge.net/projects/ipmitool/files/ipmitool/1.8.18/ipmitool-1.8.18.tar.gz
tar -xvzf ipmitool-1.8.18.tar.gz
cd ipmitool-1.8.18
./bootstrap
./configure
make
make install DESTDIR=/tmp/package-ipmitool
```

4. The i2c-tools package is also required, but the version contained in the CentOS Yum repository is old and does not work with BlueField. Therefore, please download i2c-tools version 4.1, and then build and install it.

```
# Build i2c-tools from a newer source
wget http://mirrors.edge.kernel.org/pub/software/utils/i2c-tools/i2c-tools-4.1.tar.gz
tar -xvzf i2c-tools-4.1.tar.gz
cd i2c-tools-4.1
make
make install PREFIX=/usr

# create a link to the libraries
ln -sfn /usr/lib/libi2c.so.0.1.1 /lib64/libi2c.so
ln -sfn /usr/lib/libi2c.so.0.1.1 /lib64/libi2c.so.0
```

5. Generate an RPM binary from the BlueField's mlx-OpenIPMI-2.0.25 source RPM.

   The following packages might be needed to build the binary RPM depending on which version of CentOS you are using.

```
$ yum install libtool rpm-devel rpmdevtools rpmlint wget ncurses-devel automake
$ rpmbuild --rebuild mlx-OpenIPMI-2.0.25-0.g581ebbb.src.rpm
```

> **Note**
>
> You may obtain this rpm file by means of scp from the server host's Bluefield Distribution folder. For example:
>
> ```
> $ scp <BF_INST_DIR>/distro/SRPMS/mlx-OpenIPMI-2.0.25-0.g4fdc53d.src.rpm <ip-address>:/<target_directory>/
> ```

If there are issues with building the OpenIPMI RPM, verify that the swig package is not installed.

```
$ yum remove -y swig
```

6. Generate a binary RPM from the ipmb-dev-int source RPM and install it. Run:

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

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

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

8. Load OpenIPMI, ipmb-host, and ipmb-dev-int RPM packages. Run:

```
$ yum install -y /root/rpmbuild/RPMS/aarch64/mlx-OpenIPMI-2.0.25-0.g581ebbb_5.4.0_49.el7a.aarch64.aarch64.rpm
$ yum install -y /root/rpmbuild/RPMS/aarch64/ipmb-dev-int-1.0-0.g304ea0c_5.4.0_49.el7a.aarch64.aarch64.rpm
$ yum install -y /root/rpmbuild/RPMS/aarch64/ipmb-host-1.0-0.g304ea0c_5.4.0_49.el7a.aarch64.aarch64.rpm
```

9. Load the IPMB driver. Run:

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

10. Install and start rasdaemon package. Run:

```
yum install rasdaemon
systemctl enable rasdaemon
systemctl start rasdaemon
```

11. Start the IPMI daemon. Run:

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

Redfish

Redfish provides a RESTful interface designed to manage IT infrastructure and is implemented using a modern toolchain (HTTP(s)/TLS/JSON).

Redfish supports the operations listed in this section.

BIOS Configuration Schema

The BIOS schema contains properties related to the BIOS attribute registry. The attribute registry describes the system-specific BIOS attributes and actions for changing to BIOS settings. It is likely that a client finds the @Redfish.Settings term in this resource, and if it is found, the client makes requests to change BIOS settings by modifying the resource identified by the @Redfish.Settings annotation.

URI	/redfish/v1/Systems/{ComputerSystemId}/Bios
Schema file	http://redfish.dmtf.org/schemas/v1/Bios.v1_1_1.json
Operations	GET; PATCH

Example response:


```
{
   "@Redfish.Settings": {
      "@odata.type": "#Settings.v1_3_5.Settings",
      "SettingsObject": {
         "@odata.id": "/redfish/v1/Systems/Bluefield/Bios/Settings"
      }
   },
   "@odata.id": "/redfish/v1/Systems/Bluefield/Bios",
   "@odata.type": "#Bios.v1_2_0.Bios",
   "Actions": {
      "#Bios.ChangePassword": {
         "target": "/redfish/v1/Systems/Bluefield/Bios/Actions/Bios.ChangePassword"
      },
      "#Bios.ResetBios": {
```

```
        "target": "/redfish/v1/Systems/Bluefield/Bios/Actions/Bios.ResetBios"
    }
},
"Attributes": {
    "Boot Partition Protection": false,
    "CurrentUefiPassword": "",
    "DateTime": "2024-04-24T19:56:59Z",
    "DefaultPasswordPolicy": true,
    "Disable PCIe": false,
    "Disable SPMI": false,
    "Disable TMFF": false,
    "EmmcWipe": false,
    "Enable 2nd eMMC": false,
    "Enable OP-TEE": false,
    "Enable SMMU": true,
    "Field Mode": false,
    "Host Privilege Level": "Privileged",
    "Internal CPU Model": "Embedded",
    "LegacyPasswordEnable": true,
    "NicMode": "DpuMode",
    "NvmeWipe": false,
    "OsArgs": "",
    "ResetEfiVars": false,
    "SPCR UART": "Disabled",
    "UefiArgs": "",
    "UefiPassword": ""
},
"Description": "BIOS Configuration Service",
"Id": "BIOS",
"Links": {
    "SoftwareImages": [
        {
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_ATF"
        },
        {
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_BOARD"
        },
        {
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_BSP"
        },
        {
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_NIC"
        },
        {
```

```
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_NODE"
          },
          {
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_OFED"
          },
          {
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_OS"
          },
          {
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_SYS_IMAGE"
          },
          {
            "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_UEFI"
          }
        ],
        "SoftwareImages@odata.count": 9
    },
    "Name": "BIOS Configuration",
    "ResetBiosToDefaultsPending": false
}
```


The following table explains each of the attributes listed in the code:

Attribute	Description
Boot Partition Protection	See description in section "System Configuration"
CurrentUefiPassword	See "Set Password" in section "System Configuration"
DateTime	See "Set RTC" in section "System Configuration"
DefaultPasswordPolicy	See "Password Settings" in section "System Configuration"
Disable PCIe	See description in section "System Configuration"
Disable SPMI	See description in section "System Configuration"
Disable TMFF	See description in section "System Configuration"
EmmcWipe	See description in section "System Configuration"
Enable 2nd eMMC	See description in section "System Configuration"
Enable OP-TEE	See description in section "System Configuration"
Enable SMMU	See description in section "System Configuration"

Attribute	Description
Field Mode	See description in section "System Configuration"
Host Privilege Level	See "BlueField Modes" in section "System Configuration"
Internal CPU Model	See "BlueField Modes" in section "System Configuration"
LegacyPasswordEnable	See "Password Settings" in section "System Configuration"
NicMode	See "BlueField Modes" under section "System Configuration"
NvmeWipe	See description in section "System Configuration"
OsArgs	Arguments to pass to the OS kernel
ResetEfiVars	See "Reset EFI Variables" in section "System Configuration"
SPCR UART	See " Select SPCR UART " in section "System Configuration"
UefiArgs	Arguments to pass to the UEFI
UefiPassword	See "Set Password" in section "System Configuration"

BlueField Platform Inventory

The NVIDIA® BlueField® networking platform (DPU or SuperNIC) provides inventory information in the ComputerSystemCollection schema. To identify the BlueField ComputerSystem instance, fetch the ComputerSystemCollection first.

BlueField devices are identified with the SystemType attribute DPU. The BlueField instance identifier value (DPU.Embedded.1_NIC.Slot.2 in this case) differs from one server vendor to another but will uniquely identify BlueField in all cases.

The following is a simple example of fetching Redfish inventory information from a server's BMC:

```
root@localhost:~$ python3 /usr/local/bin/redfishtool.py -r <bmc_ip> -u <USER> -p <PASSWORD> raw
GET /redfish/v1/Systems/
{
   "@odata.context": "/redfish/v1/$metadata#ComputerSystemCollection.ComputerSystemCollection",
   "@odata.id": "/redfish/v1/Systems",
   "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
   "Description": "Collection of Computer Systems",
   "Members": [
```

```
      {
         "@odata.id": "/redfish/v1/Systems/System.Embedded.1"
      },
      {
         "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2"
      }
   ],
   "Members@odata.count": 2,
   "Name": "Computer System Collection"
}

root@localhost:~$ python3 /usr/local/bin/redfishtool.py -r <bmc_ip> -u <USER> -p <PASSWORD> raw
GET /redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2
{
   "@odata.context": "/redfish/v1/$metadata#ComputerSystem.ComputerSystem",
   "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2",
   "@odata.type": "#ComputerSystem.v1_12_0.ComputerSystem",
   "Actions": {
      "#ComputerSystem.Reset": {
         "target": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/Actions/ComputerSystem.Reset",
         "ResetType@Redfish.AllowableValues": [
            "ForceRestart",
            "Nmi"
         ]
      }
   },
   "Bios": {
      "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/Bios"
   },
   "BiosVersion": null,
   "Boot": {
      "BootOptions": {
         "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/BootOptions"
      },
      "BootOrder": [],
      "BootOrder@odata.count": 0,
      "BootSourceOverrideEnabled": null,
      "BootSourceOverrideMode": null,
      "BootSourceOverrideTarget": null,
      "UefiTargetBootSourceOverride": null,
      "BootSourceOverrideTarget@Redfish.AllowableValues": []
   },
   "Description": "DPU System",
   "Id": "DPU.Embedded.1_NIC.Slot.2",
```

```
"Manufacturer": "DELL",
"Model": "NVIDIA Bluefield-2 25GbE 2p Crypto DPU",
"Name": "DPU System",



"Oem": {
   "Dell": {
      "@odata.type": "#DellComputerSystem.v1_1_0.DellComputerSystem",
      "DPUConfig": {
         "FQDD": "DPU.Embedded.1:NIC.Slot.2",
         "BootStatus": "OSBooting",
         "DPUBootSynchronization": "Enabled",
         "DPUTrust": "Enabled",
         "IdenticalSBDF": [
            "0:23:0:0",
            "0:23:0:1"
         ],
         "LastResetReason": null,
         "OSName": null,
         "OSReadyTimeout": 20,
         "OSInstallationTimeout": 30,
         "OSVersion": null,
         "OSVendor": null,
         "OSStatus": "Unknown",
         "Slot": "2",
         "PCIeSlotState": "Enabled",
         "PostCode": null,
         "VendorID": "0x15B3",
         "DeviceID": "0xA2D6",
         "SubVendorID": "0x15B3",
         "SubDeviceID": "0x0129"
      },
      "Name": "DPUConfig",
      "Id": "DPU.Embedded.1_NIC.Slot.2"
   }
},
"PartNumber": "JNDCMX01",
"SecureBoot": {
   "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/SecureBoot"
},
"SerialNumber": "IL740311A5000A",
"SKU": "0JNDCM",
"Status": {
```

```
      "Health": "Ok",
      "HealthRollup": "Ok",
      "State": "Enabled"
   },
   "SystemType": "DPU",
   "UUID": "ec6dd921-882a-ec11-8000-08c0eb5180ba",
   "@Redfish.Settings": {
      "@odata.context": "/redfish/v1/$metadata#Settings.Settings",
      "@odata.type": "#Settings.v1_3_3.Settings",
      "SettingsObject": {
         "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/Settings"
      }
   }
}
```

Boot Override

This example demonstrates how to boot a BlueField Platform while overriding the existing boot options and using HTTP boot to obtain the image.

Check the current boot override settings by doing a GET on ComputerSystem schema. Look for the Boot property.

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/ | python3 -m
json.tool
{
...
"Boot": {
      "BootNext": "",
      "BootOrderPropertySelection": "BootOrder",
      "BootSourceOverrideEnabled": "Disabled",
      "BootSourceOverrideMode": "UEFI",
      "BootSourceOverrideTarget": "None",
      "UefiTargetBootSourceOverride": "None",
      .....
      },
  ....
  "BootSourceOverrideEnabled@Redfish.AllowableValues": [
         "Once",
         "Continuous",
         "Disabled"
```

```
        ],
    "BootSourceOverrideTarget@Redfish.AllowableValues": [
            "None",
            "Pxe",
            "UefiHttp",
            "UefiShell",
            "UefiTarget",
            "UefiBootNext"
        ],
  ....
}
```

The sample output above shows the BootSourceOverrideEnabled property is Disabled and BootSourceOverrideTarget is None. The BootSourceOverrideMode property should always be set to UEFI. Allowable values of BootSourceOverrideEnabled and BootSourceOverrideTarget are defined in the meta-data BootSourceOverrideEnabled@Redfish.AllowableValues and BootSourceOverrideTarget@Redfish.AllowableValues respectively.

To perform boot override, you must perform a PATCH to pending settings URI:

```
curl -vk -X PATCH -d '{"Boot": {"BootSourceOverrideEnabled":"Once",
"BootSourceOverrideMode":"UEFI", "BootSourceOverrideTarget": "UefiHttp",
"HttpBootUri":"http://<HTTP-Server-Ip>/Image.iso"}}' -u "user:password"
https://<bmc_ip>/redfish/v1/Systems/SystemId/Settings | python3 -m json.tool
```

After performing the above PATCH successfully, reboot the BlueField Platform. Once UEFI has completed, check whether the settings are applied by performing a GET on ComputerSystem schema.

Note that the HttpBootUri property is parsed by the Redfish server and the URI is presented to BlueField as part of DHCP lease when BlueField performs the HTTP boot.

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/ | python3 -m
json.tool
{
...
"Boot": {
        "BootNext": "",
        "BootOrderPropertySelection": "BootOrder",
```

```
    "BootSourceOverrideEnabled": "Once",
    "BootSourceOverrideMode": "UEFI",
    "BootSourceOverrideTarget": "UefiHttp",
    "UefiTargetBootSourceOverride": "None",
    .....
    },
  .....
}
```

After confirming the settings are applied (see PATCH properties above), reboot BlueField for the settings to take effect. If BootSourceOverrideEnabled is set to Once, boot override is disabled and any related properties are reset to their former values to avoid repetition. If it is set to Continuous, then on every reboot, BlueField would keep performing boot override (HTTPBoot).

Boot Order

The following is an example of changing the boot order and fetching the details of a boot option.

1. Check the current boot order by doing GET on the ComputerSystem schema. Look for the BootOrder attribute under the Boot property.

2. Get the details of a particular entity in the BootOrder array by performing a GET to the respective BootOption URL. For example, to get details of Boot0006, run:

```
curl -vk -X GET -u "user:password"
https://<bmc_ip>/redfish/v1/Systems/SystemId/BootOptions/Boot0006 | python3 -m json.tool

{
   "@odata.type": "#BootOption.v1_0_3.BootOption",
   "@odata.id": "/redfish/v1/Systems/SystemId/BootOptions/Boot0006",
   "Id": "Boot0006",
   "BootOptionEnabled": true,
   "BootOptionReference": "Boot0006",
   "DisplayName": "UEFI HTTPv6 (MAC:B8CEF6B8A006)",
   "UefiDevicePath":
"PciRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/MAC(B8CEF6B8A006,0x1)/IPv6(000
}
```

3. To change the boot order, the entire BootOrder array must be PATCHed to the pending settings URI. For the above example of the BootOrder array, if you intend to have Boot0006 at the beginning of the array, then the PATCH operation is as follows.

```
curl -vk -X PATCH -d '{ "Boot": { "BootOrder": [ "Boot0006", "Boot0017", "Boot0001", "Boot0002",
"Boot0003", "Boot0004", "Boot0005", "Boot0007", ] }}' -u "user:password"
https://<bmc_ip>/redfish/v1/Systems/SystemId/Settings | python3 -m json.tool
```

> **Note**
>
> Updating the BootOrder array results in a permanent boot order change (persistent across reboots).

After a successful PATCH, reboot BlueField and check if the settings were applied by doing a GET on the ComputerSystem schema. If the BootOrder array is updated as intended, then the settings were applied and the BlueField Platform should boot as per the order in proceeding cycles.

BIOS Attributes

The following is an example of fetching and setting a BlueField BIOS attribute.

1. Check UEFI attributes and their values by doing a GET on Bios URL. Look for Attributes property.

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/Bios | python3
-m json.tool

{
    ....
    "Attributes": {
        "Boot Partition Protection": false,
        "CurrentUefiPassword": "",
        "DateTime": "2022-07-05T16:02:12Z",
        "Disable PCIe": false,
```

```
        "Disable SPMI": false,
        "Disable TMFF": false,
        "Enable 2nd eMMC": false,
        "Enable OP-TEE": false,
        "Enable SMMU": true,
        "Field Mode": false,
        "Host Privilege Level": "Privileged",
        "Internal CPU Model": "Embedded",
        "ResetEfiVars": false,
        "SPCR UART": "Disabled",
        "UefiPassword": ""
    },
   ....
}
```

> **Note**
>
> For Security reasons, CurrentUefiPassword and UefiPassword strings might be empty.

2. The following example updates the UEFI password. Perform PATCH to Bios pending settings URI as follows:

```
curl -vk -X PATCH -d '{"Attributes":{"CurrentUefiPassword": "CURRENTPASSWD", "UefiPassword": "NEWPASSWORD"}}' -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/Bios/Settings | python3 -m json.tool
```

> **Note**
>
> To update the password, both the current password and the new password (requesting) should be specified as

demonstrated above. Otherwise, the change does not work. To modify other attributes no password is required.

3. To confirm whether the PATCH request is successful, perform a GET to the BIOS pending settings URI:

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/Bios/Settings | python3 -m json.tool
```

4. For requests to take effect, reboot BlueField. If the `CurrentUefiPassword` is correct, then the UEFI password is updated during the UEFI Redfish phase of boot.

> **Info**
>
> The UEFI password is only required to enter the UEFI menu using the serial console.

Logging

RShim Logging

RShim logging uses an internal 1KB HW buffer to track booting progress and record important messages. It is written by the NVIDIA ® BlueField ® networking platform's (DPU or SuperNIC) Arm cores and is displayed by the RShim driver from the USB/PCIe host machine. Starting in release 2.5.0, ATF has been enhanced to support the RShim logging.

The RShim log messages can be displayed described in the following:

1. Check the DISPLAY_LEVEL level in file /dev/rshim0/misc.

```
# cat /dev/rshim0/misc
DISPLAY_LEVEL   0 (0:basic, 1:advanced, 2:log)
...
```

2. Set DISPLAY_LEVEL to 2.

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

3. Log messages are displayed in the misc file.

```
# cat /dev/rshim0/misc
...
----------------------------------------
       Log Messages
----------------------------------------
 INFO[BL2]: start
 INFO[BL2]: no DDR on MSS0
 INFO[BL2]: calc DDR freq (clk_ref 53836948)
```

```
INFO[BL2]: DDR POST passed
INFO[BL2]: UEFI loaded
INFO[BL31]: start
INFO[BL31]: runtime
INFO[UEFI]: eMMC init
INFO[UEFI]: eMMC probed
INFO[UEFI]: PCIe enum start
INFO[UEFI]: PCIe enum end
```

Info

This is an example output for BlueField-2.

The following table details the ATF/UEFI messages for BlueField-2 and BlueField-3:

Message	Explanation	Action
INFO[BL2]: start	BL2 started	Informational
INFO[BL2]: no DDR on MSS<N>	DDR is not detected on memory controller <N>	Informational (depends on device)
INFO[BL2]: calc DDR freq (clk_ref 156M, clk xxx)	DDR frequency is calculated based on reference clock 156M	Informational
INFO[BL2]: calc DDR freq (clk_ref 100M, clk xxx)	DDR frequency is calculated based on reference clock 100M	Informational
INFO[BL2]: calc DDR freq (clk_ref xxxx)	DDR frequency is calculated based on reference clock xxxx	Informational
INFO[BL2]: DDR POST passed	BL2 DDR training passed	Informational
INFO[BL2]: UEFI loaded	UEFI image is loaded successfully in BL2	Informational

Message	Explanation	Action
ERR[BL2]: DDR init fail on MSS<N>	DDR initialization failed on memory controller <N>	Informational (depends on device)
ERR[BL2]: image <N> bad CRC	Image with ID <N> is corrupted which will cause hang	Error message. Reset the device and retry. If problem persists, use a different image to retry it.
ERR[BL2]: DDR BIST failed	DDR BIST failed	Need to retry. Check the ATF booting message whether the detected OPN is correct or not, or whether it is supported by this image. If still fails, contact NVIDIA Support.
ERR[BL2]: DDR BIST Zero Mem failed	DDR BIST failed in the zero-memory operation	Power-cycle and retry. If the problem persists, contact your NVIDIA FAE.
WARN[BL2]: DDR frequency unsupported	DDR training is programmed with unsupported parameters	Check whether official FW is being used. If the problem persists, contact your NVIDIA FAE.
WARN[BL2]: DDR min-sys(unknown)	System type cannot be determined and boot as a minimal system	Check whether the OPN or PSID is supported. If the problem persists, contact your NVIDIA FAE.
WARN[BL2]: DDR min-sys(misconf)	System type misconfigured and boot as a minimal system	Check whether the OPN or PSID is supported. If the problem persists, contact your NVIDIA FAE.
Exception(BL2): syndrome = xxxxxxxx ...	Exception in BL2 with syndrome code and register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
PANIC(BL2): PC = xxx ...	Panic in BL2 with register dump. System will hung.	Capture the log, analyze the cause, and report to FAE if needed
ERR[BL2]: load/auth failed	Failed to load image (non-existent/corrupted), or image authentication failed when secure boot is enabled	Try again with the correct and properly signed image

Message	Explanation	Action
INFO[BL31]: start	BL31 started	Informational
INFO[BL31]: runtime	BL31 enters the runtime state. This is the latest BL31 message in normal booting process.	Informational
Exception(BL31): syndrome = xxxxxxxx cptr_el3 xx daif xx ...	Exception in BL31 with syndrome code and register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
PANIC(BL31): PC = xxx cptr_el3 xxx daif xxx ...	Panic in BL31 with register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
INFO[UEFI]: eMMC init	eMMC driver is initialized	Informational and should always be printed
INFO[UEFI]: eMMC probed	eMMC card is initialized	Informational and should always be printed
ASSERT(UEFI]: xxx : line-no	Runtime assert message in UEFI	Contact your NVIDIA FAE with this information. Usually the system is able to continue running.
INFO[UEFI]: PCIe enum start	PCIe enumeration start	Informational
INFO[UEFI]: PCIe enum end	PCIe enumeration end	Informational
ERR[UEFI]: Synchronous Exception at xxxxxx ERR[UEFI]: PC=xxxxxx ERR[UEFI]: PC=xxxxxx ...	UEFI Exception with PC value reported	Contact your NVIDIA FAE with this information

Message	Explanation	Action
ERR[BL2]: FW auth failed	Image authentication error	Wrong image has been used in the current secure lifecycle. Switch to the correct image.
ERR[BL2]: IROT cert sig not found	Failed to load attestation certificates	Contact your NVIDIA FAE with this information
ERR[BL2]: IROT cert sig not found	Failed to load certification update record **Info** Only relevant for certain BlueField-3 devices.	Contact your NVIDIA FAE with this information
INFO[BL31]: PSC Turtle Mode detected	PSC enters turtle mode **Info** BlueField-3 only.	Informational
INFO[BL31]: In Enhanced NIC mode	BlueField-3 enters enhanced NIC mode	Informational
ERR[BL31]: (set_page err | pmbus_lsb err | mfr_vr_mc err | set_vout err)	BlueField-3 power management programming error. **Info** Usually happens when the I2C	Contact your NVIDIA FAE with this information

Message	Explanation	Action
	voltage regulator is not accessible.	
INFO [BL31]: MB8: VDD adjustment complete	BlueField-3 MainBin 8-core board VDD CPU adjustment	Informational
INFO [BL31]: VDD adjustment complete	BlueField-3 (non-8-core board) VDD CPU adjustment	Informational
INFO [BL31]: VDD: xxx mV	BlueField-3 VDD CPU voltage	Informational
ERR[BL31]: cannot access vr0 (or access vr1)	BlueField-3 unable to access voltage regulator (vr0 or vr1) via I2C	Contact your NVIDIA FAE with this information
ERR[BL31]: ATX power not detected!	ATX power is not connected	Contact your NVIDIA FAE with this information
INFO[BL31]: PTMERROR: Unknown OPN	Unable to detect the OPN on this device	Contact your NVIDIA FAE with this information
INFO[BL31]: PTMERROR: VR access error	Unable to access the voltage regulator on this device **Info** This also means power capping will be disabled.	Contact your NVIDIA FAE with this information
INFO[BL31]: power capping disabled	BlueField-3 power capping disabled	Informational

Message	Explanation	Action
INFO[BL2]: boot mode (rshim | emmc | unknown)	Device boot mode (from external RShim or eMMC)	Informational
ERR[BL31]: ECC_SINGLE_ERROR_CNT=xxx	Single ECC error counter report	Contact your NVIDIA FAE with this information
ERR[BL31]: ECC_DOUBLE_ERROR_CNT=xxx	Double ECC error counter report	Contact your NVIDIA FAE with this information
ERR[BL31]: mss0|mss1: C0|C1 single-bit ecc, IRQ[%d]	MSS (0 or 1) channel (0 or 1) single-bit ECC error interrupt #	Contact your NVIDIA FAE with this information
ERR[BL31]: mss0|mss1: C0|C1 Double bit ecc, IRQ[%d]	MSS (0 or 1) channel (0 or 1) double-bit ECC error interrupt #	Contact your NVIDIA FAE with this information
ERR[BL31]: Double-bit ECC also detected in same buffer	Single/double ECC error detected in the same buffer	Contact your NVIDIA FAE with this information
ERR[BL31]: l3c: double-bit ecc	L3c double-bit ECC error detected	Contact your NVIDIA FAE with this information
ERR[BL31]: MSS%d DIMM%d single|double bit ECC error detected	MSS DRAM single (or double) bit error detected	Contact your NVIDIA FAE with this information
ERR[BL31]: MSS%d SRAM double bit ECC error detected	MSS SRAM double bit ECC error detected	Contact your NVIDIA FAE with this information

IPMI Logging in UEFI

During UEFI boot, the BlueField sends IPMI SEL messages over IPMB to the BMC in order to track boot progress and report errors. The BMC must be in responder mode to receive the log messages.

SEL Record Format

The following table presents standard SEL records (record type = 0x02).

Byte(s)	Field	Description
1 2	Record ID	ID used to access SEL record. Filled in by the BMC. Is initialized to zero when coming from UEFI.
3	Record Type	Record type
4 5 6 7	Timestamp	Time when event was logged. Filled in by BMC. Is initialized to zero when coming from UEFI.
8 9	Generator ID	This value is always 0x0001 when coming from UEFI
10	EvM Rev	Event message format revision which provides the version of the standard a record is using. This value is 0x04 for all records generated by UEFI.
11	Sensor Type	Sensor type code for sensor that generated the event
12	Sensor Number	Number of the sensor that generated the event. These numbers are arbitrarily chosen by the OEM.
13	Event Dir | Event Type	[7] – 0b0 = Assertion, 0b1 = Deassertion [6:0] – Event type code
14	Event Data 1	[7:6] – Type of data in Event Data 2 • 0b00 = unspecified • 0b10 = OEM code • 0b11 = Standard sensor-specific event extension [5:4] – Type of data in Event Data 3 • 0b00 = unspecified • 0b10 = OEM code • 0b11 = Standard sensor-specific event extension

Byte(s)	Field	Description
		[3:0] – Event Offset; offers more detailed event categories. See *IPMI 2.0 Specification* section 29.7 for more detail.
15	Event Data 2	Data attached to the event. 0xFF for unspecified. Under some circumstances, this may be used to specify more detailed event categories.
16	Event Data 3	Data attached to the event. 0xFF for unspecified.

See *IPMI 2.0 Specification* section 32.1 for more detail.

Possible SEL Field Values

BlueField UEFI implements a subset of the IPMI 2.0 SEL standard. Each field may have the following values:

Field	Possible Values	Description of Values
Record Type	0x02	Standard SEL record. All events sent by UEFI are standard SEL records.
Event Dir	0b0	All events sent by UEFI are assertion events
Event Type	0x6F	Sensor-specific discrete events. Events with this type do not deviate from the standard.
Sensor Number	0x06	UEFI boot progress “sensor”. If value is 0x06, the sensor type will always be “System Firmware Progress” (0x0F).

For Sensor Type, Event Offset, and Event Data 1-3 definitions, see next table.

Event Definitions

Events are defined by a combination of Record Type, Event Type, Sensor Type, Event Offset (occupies Event Data 1), and sometimes Event Data 2 (referred to as the Event Extension if it defines sub-events).

The following tables list all currently implemented IPMI events (with Record Type = 0x02, Event Type = 0x6F).

> **Note**
>
> Note that if an Event Data 2 or Event Data 3 value is not specified, it can be assumed to be Unspecified (0xFF).

Sensor Type	Sensor Type Code	Event Offset	Event Description, Actions to Take
System Firmware Progress	0x0F	0x00	System firmware error (POST error). Event Data 2: • 0x06 – Unrecoverable EMMC error. Contact NVIDIA support.
		0x02	System firmware progress: Informational message, no actions needed. Event Data 2: • 0x02 – Hard Disk Initialization. Logged when EMMC is initialized. • 0x04 – User Authentication. Logged when a user enters the correct UEFI password. This event is never logged if there is no UEFI password. • 0x07 – PCI Resource Configuration. Logged when PCI enumeration has started. • 0x0B – SMBus Initialization. This event is logged as soon as IPMB is configured in UEFI. • 0x13 – Starting OS Boot Process. Logged when Linux begins booting.

Reading IPMI SEL Log Messages

Log messages may be read from the BMC by issuing it a “Get SEL Entry Command” while it is in responder mode, either from a remote host, or from BlueField itself once it is booted.

```
$ ipmitool sel list
  7b | Pre-Init |0000691604| System Firmwares #0x06 | SMBus initialization | Asserted
  7c | Pre-Init |0000691604| System Firmwares #0x06 | Hard-disk initialization | Asserted
  7d | Pre-Init |0000691654| System Firmwares #0x06 | System boot initiated
$ ipmitool sel get 0x7d
SEL Record ID          : 007d
 Record Type           : 02
 Timestamp             : 01/09/1970 00:07:34
 Generator ID          : 0001
 EvM Revision          : 04
 Sensor Type           : System Firmwares
 Sensor Number         : 06
 Event Type            : Sensor-specific Discrete
 Event Direction       : Assertion Event
 Event Data            : c213ff
 Description           : System boot initiated
$ ipmitool sel clear
Clearing SEL.  Please allow a few seconds to erase.
$ ipmitool sel list
SEL has no entries
```

ACPI BERT Logging

ACPI boot error record table (BERT) is supported to log last boot error in Linux. Once Linux printk is enabled (e.g., by adding "kernel.printk=8" to /etc/sysctl.conf), it will try to report the errors automatically for last boot. The following is an example of such error reports:

```
[    2.635539] BERT: Error records from previous boot:
[    2.640434] [Hardware Error]: event severity: fatal
[    2.645331] [Hardware Error]:  Error 0, type: fatal
```

```
[    2.650236] [Hardware Error]:   section type: unknown, c6adf9e6-1108-4760-8827-003d059fe2e1
[    2.658606] [Hardware Error]:   section length: 0x35
[    2.663580] [Hardware Error]:   00000000: 52524520 4645555b 203a5d49 0a0d0a0d   ERR[UEFI]: ....
[    2.672284] [Hardware Error]:   00000010: 636e7953 6e6f7268 2073756f 65637845  Synchronous Exce
[    2.680987] [Hardware Error]:   00000020: 6f697470 7461206e 36783020 37313643  ption at 0x6C617
[    2.689696] [Hardware Error]:   00000030: 34 37 30 0d 0a
...
```

SoC Management Interface

The SoC management interface, formerly known as RShim, allows an external agent such as the host CPU or BMC to operate the NVIDIA® BlueField® networking platform (DPU or SuperNIC) and monitor its operational state. This interface allows provisioning of BlueField, resetting Arm cores, and obtaining logs.

> **Note**
>
> For instructions for Windows support, please refer to page "Windows Support".

Installation and Upgrade

Please refer to section Updating Repo Package on Host Side.

Configuration File

The configuration file for the SoC management interface is located at /etc/rshim.conf and includes the parameters listed in the table below.

Parameter	Default	Description
BOOT_TIMEOUT	150	Timeout value in seconds when pushing BFB while Arm side is not reading the boot stream.
DROP_MODE	0	Once set to 1, the RShim driver ignores all RShim writes and returns 0 for RShim read. This is used in cases such as during FW_RESET or bypassing the RShim PF to VM.
PCIE_RESET_DELAY	10	Delay in seconds for RShim over PCIe, which is added after chip

Parameter	Default	Description
		reset and before pushing the boot stream.
PCIE_INTR_POLL_INTERVAL	10	Interrupt polling interval in seconds when running RShim over direct memory mapping.
PCIE_HAS_VFIO	1	Setting this parameter to 0 disallows RShim memory mapping via VFIO.
PCIE_HAS_UIO	1	Setting this parameter to 0 disallows RShim memory mapping via UIO.

Note

Configuring RShim is optional. The default parameters are designed to support out-of-box deployment scenarios including multiple BlueField device s on a single host.

Users may control which RShim index maps to which device by following this procedure:

```
# Uncomment the 'rshim<N>' line to configure the mapping.
#
# device-name pci-device
rshim0     pcie-0000:21:00.2
rshim1     pcie-0000:81:00.2

#
# Ignored devices.
# Uncomment the 'none' line to configure the ignored devices.
#
#none       usb-1-1.4
#none       pcie-lf-0000:84:00.0
```

Note

If any of these configurations are changed, then the SoC management interface must be restarted by running:

```
systemctl restart rshim
```

Host-side Interface Configuration

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

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

A special SoC management driver must be installed and run on the host OS to expose the various BlueField management interfaces to the OS. Currently, this driver is named RShim and is automatically installed as part of the DOCA installation. Refer to section "Install RShim on Host" for information on how to obtain and install the host-side SoC management interface driver .

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

```
# systemctl status rshim
  rshim.service - rshim driver for BlueField SoC
    Loaded: loaded (/lib/systemd/system/rshim.service; disabled; vendor preset: enabled)
    Active: active (running) since Tue 2022-05-31 14:57:07 IDT; 1 day 1h ago
      Docs: man:rshim(8)
   Process: 90322 ExecStart=/usr/sbin/rshim $OPTIONS (code=exited, status=0/SUCCESS)
  Main PID: 90323 (rshim)
     Tasks: 11 (limit: 76853)
```

```
    Memory: 3.3M
    CGroup: /system.slice/rshim.service
            90323 /usr/sbin/rshim
May 31 14:57:07 ...  systemd[1]: Starting rshim driver for BlueField SoC...
May 31 14:57:07  ... systemd[1]: Started rshim driver for BlueField SoC.
May 31 14:57:07  ... rshim[90323]: Probing pcie-0000:a3:00.2(vfio)
May 31 14:57:07  ... rshim[90323]: Create rshim pcie-0000:a3:00.2
May 31 14:57:07  ... rshim[90323]: rshim pcie-0000:a3:00.2 enable
May 31 14:57:08  ... rshim[90323]: rshim0 attached
```

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

Virtual Ethernet Interface

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

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

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

Note

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

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

```
ssh ubuntu@192.168.100.2
```

SoC Management Interface Driver Support for Multiple BlueFields

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

> **Note**
>
> <N> correlates to the number of BlueField devices used where the SoC management interfaces of the first BlueField is 0, incrementing by 1 for each added device.

The following are some guidelines on how to set up the SoC management virtual Ethernet interfaces properly if multiple BlueField device s are installed in the host system.

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

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

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

- Configured using ifconfig. For example:

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

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

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

Multi-board Management Example

This example deals with two BlueField devices installed on the same server (the process is similar for more devices). The example assumes that the RShim package has been installed on the host server.

Configuring Management Interface on Host

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

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

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

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

```
DEVICE="br_tmfifo"
BOOTPROTO="static"
IPADDR="192.168.100.1"
NETMASK="255.255.255.0"
```

```
ONBOOT="yes"
TYPE="Bridge"
```

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

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

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

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

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

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

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

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

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

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

Configuring BlueField Side

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

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

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

Updating RShim Network MAC Address

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

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

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

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

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

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

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

4. Apply the new MAC address. Run:

```
sudo bfcfg
```

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

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

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

Updating IP Address

For Ubuntu:

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

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

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

2. Reboot the Arm. Run:

```
sudo reboot
```

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

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

For CentOS:

1. Access the file ifcfg-tmfifo_net0. Run:

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

2. Modify the value for IPADDR:

```
IPADDR=192.168.100.3
```

3. Reboot the Arm. Run:

```
reboot
```

Or perform `netplan apply`.

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

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

Permanently Changing Arm-side MAC Address

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

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

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

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

3. If not mounted, run:

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

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

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

1. Enable the configuration. Run:

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

2. Display the current setting. Run:

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

3. Modify the MAC address. Run:

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

Info

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

SoC Management Interface Features and Functionality

	Function	Command	Comments
1	Push BFB	`bfb-install –r rshim<N> -b <bfb> [-c bf.cfg]`	Using bf.cfg in the command is optional. For more details about bf.cfg, refer to "Customizing BlueField Software Deployment Using bf.cfg".
2	Open console	`screen /dev/rshim<N>/console 115200` `minicom -D /dev/rshim<N>/console`	The N index depends on the number of BlueField devices in your setup. Use Linux's screen or minicom console applications to access the BlueField console.
3	Configure a virtual network interface	`ip addr add dev tmfifo_net<N> 192.168.100.1/30`	The N index depends on the number of BlueField devices in your setup. Refer to section "SoC Management Interface Driver Support for Multiple BlueFields" for more information. The default IP address for the BlueField is 192.168.100.2/30. The IP used in the command (192.168.100.1/30) is for example purposes only.

	Function	Command	Comments
4	Log into the DPU	`ssh -6 user@fe80::21a:caff:feff:ff01%tmfifo_net<N>`	The N index depends on the number of DPUs in your setup. Refer to section "SoC Management Interface Driver Support for Multiple BlueFields" for more information.
5	PXE boot over RShim	N/A	Please refer to section "Deploying BlueField Software Using BFB with PXE" for more information.
6	Issue Arm software reset	`echo "SW_RESERT 1" > /dev/rshim<N>/misc`	
7	Expose log messages	N/A	For more information, please refer to section "Logging".

BlueField Configuration File

The bf.cfg file contains configuration that can be pushed to customize the installation of the BFB.

See "Customizing BlueField Software Deployment Using bf.cfg" for more information.

RShim Ownership

The RShim interface may be owned by the BlueField BMC or the host (Windows or Linux). In situations where users do not have access to the host, they would want to transfer RShim ownership to the BMC.

Assuming that /dev/rshim0 is the BlueField requesting ownership over the RShim interface, ownership may be transferred to the BlueField BMC by running the following command from the BMC console:

```
bf-bmc# echo "FORCE_CMD 1" > /dev/rshim0/misc
```

Tip

This method requires the RShim driver binary to be run with -F or --force option, or have the FORCE_MODE set to 1 in rshim.conf. Otherwise, /dev/rshim<N>/ would not be created if the driver is detached from RShim.

To set ownership priority to the host or BMC, users may forcibly do so using either of the following options:

- The command line option -F or --force
- Setting FORCE_MODE 1 in the configuration file rshim.conf

BlueField OOB Ethernet Interface

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

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

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

OOB Interface MAC Address

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

⚠ Warning

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

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

Note

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

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

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

3. Run:

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

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

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

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

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

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

```
dmpstore -d OobMacAddr
```

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

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

6. Run:

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

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

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

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

Supported ethtool Options for OOB Interface

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

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

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

Where <option> may be:

- <no-argument> – display interface link information
- -i – display driver general information
- -S – display driver statistics
- -d – dump driver register set
- -g – display driver ring information
- -G – configure driver ring(s)
- -k – display driver offload information
- -a – query the specified Ethernet device for pause parameter information
- -r – restart auto-negotiation on the specified Ethernet device if auto-negotiation is enabled

For example:

```
$ ethtool oob_net0
Settings for oob_net0:
        Supported ports: [ TP ]
        Supported link modes:   1000baseT/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  1000baseT/Full
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: Yes
```

```
    Advertised FEC modes: Not reported
    Link partner advertised link modes:  1000baseT/Full
    Link partner advertised pause frame use: Symmetric
    Link partner advertised auto-negotiation: Yes
    Link partner advertised FEC modes: Not reported
    Speed: 1000Mb/s
    Duplex: Full
    Port: Twisted Pair
    PHYAD: 3
    Transceiver: internal
    Auto-negotiation: on
    MDI-X: Unknown
    Link detected: yes
```

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

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

```
rx_filter_discard_pkts: 4
```

IP Address Configuration for OOB Interface

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

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

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

- CentOS configuration of IP interface:
    - Configuration file for oob_net0: /etc/sysconfig/network-scripts/ifcfg-oob_net0
    - For example, use the following to enable DHCP:

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

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

```
NAME="oob_net0"
DEVICE="oob_net0"
IPV6INIT="no"
NM_CONTROLLED="no"
PEERDNS="yes"
ONBOOT="yes"
```

```
BOOTPROTO="static"
IPADDR="192.168.200.2"
PREFIX=30
GATEWAY="192.168.200.1"
DNS1="192.168.200.1"
TYPE=Ethernet
```

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