ÜberNIC Family
User Guide

v2.04

February 3, 2026

Table of Contents

1. Overview/Purpose

This document contains information about the implementation and usage of ÜberNIC and its constituent components.

2. Version/Change History

Version	Summary	Date
1.00	Initial Version	2023-10-24
1.01	Added: 1. (Section 3) Deployment Timeline 2. (Section 4) Host I/O Transfer 3. (Section 5) Configuration Tools a) NMCLI b) NMTUI c) IFCONFIG d) IP e) ETHTOOL 4. (Section 6) Example Application 5. (Section 7) Linux Stack 6. (Section 8) ÜberSock 7. (Section 9) ÜberClear 8. (Section 10) Capabilities	2024-01-02
1.02	Reformatted this document by removing details about how to configure ÜberNIC using Linux (NMCLI, NMTUI, IFCONFIG, IP, ETHTOOL) from Section 5 and created new documents that explain in detail about Linux network configuration instead. These new documents are: Linux Network Configuration User Guide (NMCLI & NMTUI) Linux Network Configuration User Guide (IFCONFIG) Linux Network Configuration User Guide (IP) Linux Network Configuration User Guide (ETHTOOL)	2024-01-16
1.03	Added a new section, “Getting Started” after the “Host I/O Transfer” section. This section explains the required preparation to enable CXL in ÜberNIC using Red Hat as the OS.	2024-02-05
1.04	Added details on PTM enablement under the “Getting Started” section. This section explains the required steps to enable PTM in ÜberNIC using Red Hat as the OS.	2024-02-19
1.05	Added details on PPS enablement under the “Getting Started” section. This section explains the required steps to enable PPS in ÜberNIC using Red Hat as the OS.	2024-02-27
1.06	Added details for Debian-based OS under the “Driver and Supporting Software Installation” section.	2024-03-01
1.07	Added new section “Content Conventions” to describe the typographic conventions used in this document.	2024-03-04
1.08	Added new section “ÜberSock” Updated formatting	2024-03-07
1.10	Added details on configuring a default gateway and dns nameservers	2024-03-07
1.11	Added details on configuring Ubuntu and Debian for CXL and Supported OSs	2024-03-19
1.12	Added description placeholders for example ÜberSock API designs/applications	2024-03-25
1.13	1. Colored OS names throughout the document with blue and purple. Updated the content conventions with two new entries, blue and purple. 2. Added section 1. Cooling, within Hardware Installation 3. Amended PTM install instructions to limit display of advertised capabilities to only ÜberNIC, excluding the host.	2024-03-27

<table>
<tr><td>1.14</td><td>Renamed Chapter 7 ÜberSock to Network I/O, amended and added additional details.</td><td>2024-03-29</td></tr>
<tr><td>1.15</td><td>Added details for common issues and how to address them under the Troubleshooting section.</td><td>2024-04-05</td></tr>
<tr><td>1.16</td><td>Added details on LMS FTP Drop Site</td><td>2024-04-08</td></tr>
<tr><td>1.17</td><td>Reorganized document sections for improved clarity.
The following table lists out the sections involved and their new location:
<table>
<tr><th>Name</th><th>Previous Location</th><th>New Location</th></tr>
<tr><td>Operating Systems</td><td>Chapter 6, Section C</td><td>Chapter 6, Section B</td></tr>
<tr><td>CXL BIOS Enablement</td><td>Chapter 6, Section B</td><td>Chapter 7, Section A</td></tr>
<tr><td>Linux Installation</td><td>Chapter 7, Section A</td><td>Chapter 7, Section C</td></tr>
<tr><td>Verifying ÜberNIC Accessibility to Host OS</td><td>Chapter 7, Section C</td><td>Chapter 7, Section E</td></tr>
<tr><td>Installing CXL Module for Linux Kernel (RHEL8 only)</td><td>Chapter 7, Section E</td><td>Chapter 7, Section F</td></tr>
<tr><td>CXL Enablement in CXL Enablement in ÜberNIC</td><td>Chapter 7, Section F</td><td>Chapter 7, Section G</td></tr>
</table></td><td>2024-04-17</td></tr>
<tr><td>1.18</td><td>Added details on running ÜberSock API examples on Linux TCP/IP stack
Updated ftp drop site details to include directory tree
Added interface management examples
Added additional example design application (udp2tcp_loop_ex)</td><td>2024-04-25</td></tr>
<tr><td>1.19</td><td>Reorganized document sections for improved clarity.
The following table lists out the sections involved and their new location:
<table>
<tr><th>Name</th><th>Previous Location</th><th>New Location</th></tr>
<tr><td>Qualified Vendor List</td><td>Chapter 13</td><td>Chapter 7</td></tr>
<tr><td>Supported Capabilities and Availability Roadmap</td><td>Chapter 14</td><td>Chapter 8</td></tr>
<tr><td>Getting Started</td><td>Chapter 7</td><td>Chapter 9</td></tr>
<tr><td>Network I/O</td><td>Chapter 8</td><td>Chapter 10</td></tr>
<tr><td>ÜberNIC Network Port Management and User Space I/O</td><td>Chapter 12</td><td>Chapter 11</td></tr>
<tr><td>ÜberNIC FPGA Logic Management - ÜberBMC</td><td>Chapter 11</td><td>Chapter 12</td></tr>
<tr><td>Precision Time Measurement (PTM)</td><td>Chapter 9</td><td>Chapter 13</td></tr>
<tr><td>TGPIO PPS - *Experimental*</td><td>Chapter 10</td><td>Chapter 14</td></tr>
<tr><td>Troubleshooting</td><td>Chapter 14</td><td>Chapter 15</td></tr>
</table></td><td>2024-05-07</td></tr>
<tr><td>1.20</td><td>Added a new chapter on ÜberTime, a utility tool to manage PTPv2 and PPS in ÜberNIC</td><td>2024-05-08</td></tr>
<tr><td>1.21</td><td>Added details for ÜberBMC command syntax and utility function examples.</td><td>2024-05-09</td></tr>
<tr><td>1.22</td><td>Added a new entry for the troubleshooting section for when lspci fails to identify ÜberNIC.</td><td>2024-05-22</td></tr>
<tr><td>1.23</td><td>For better flow and clarity, chapter TGPIO PPS (which was previously in chapter 14) has been relocated to follow chapter ÜberTime.</td><td>2024-05-23</td></tr>
<tr><td>1.24</td><td>Added a note in the Network I/O chapter about the current limitation in ÜberNIC’s routing capabilities.</td><td>2024-05-27</td></tr>
<tr><td>1.25</td><td>Added a new section regarding ÜberNIC’s supported session in the Network I/O chapter.</td><td>2024-05-28</td></tr>
<tr><td>1.26</td><td>1. Added details on updating ÜberNIC’s software logic
2. Updated the tree structure for ÜberNIC’s Driver and SW directory</td><td>2024-05-29</td></tr>
</table>

	3. Updated the FTP drop site information to include a browser link and updated the directory/tree file structure 4. Added details on changing the ÜberSock API example code's local and remote port values	
1.27	Added New Section 9. Getting Started, B. Hardware Installation, 1) Sensors and Thresholds Moved Section 9. Getting Started, B. Hardware Installation, Cooling from 1) to 2)	2024-05-31
1.28	Minor grammar and formatting corrections	2024-06-12
1.29	1. Renamed ÜberClear to ÜberLoad 2. Added details on ÜberL2, ÜberSock, and ÜberLoad. 3. Added a new section on using ÜberSock with UDP 4. Added a new section on using ÜberSock with TX timestamp 5. Added a new ÜberSock API example design, udp_recv_multicast_ex 6. Renamed Red Hat 9.3 to Red Hat 9.4 to reflect the current release version 7. Added details to better differentiate between CXL setup and PCIe setup in the Getting Started chapter. 8. Added a new entry in the troubleshooting chapter for 'dax memory' related issues	2024-06-26
1.30	Updated to include v20240704 of PCIe, CXL, and Driver_API Logic	2024-07-04
1.31	Updated to include Thread-Safe and Non-Thread-Safe command line options for ÜberLoad.	2024-07-12
1.32	Updated Sections 3, 7, and 8	2024-07-16
1.33	Added a crucial notice regarding disabling IOMMU to prevent adverse effects on ÜberNIC's performance in chapter 9 and 10.	2024-07-23
1.34	Added new ÜberLoad options related to ÜberNIC's interface IP in chapter 10, section G: ÜberLoad	2024-07-30
1.35	Added a new section, "Incompatibilities and Known Limitations" before the "LMS FTP Drop Site" chapter. Moved the known limitations for ÜberLoad from chapter 10 to this new section as well.	2024-08-05
1.36	Added a new sub-section, "ÜberNIC Sample Applications" in chapter 11, "Network I/O" and a new sub-section, "Disabling IOMMU" in chapter 10, "Getting Started".	2024-08-08
1.37	1. Added new chapters related to ÜberNIC tools, ÜberINFO and ÜberDUMP. 2. Added new best practice for ÜberLoad. 3. Added a new entry in the troubleshooting chapter for ÜberLoad related issues.	2024-08-23
1.38	Updated 5. Incompatibilities and Known Limitation: 1. Added a new Section B. Previous Limitations – Now Supported 2. Updated C. Known Limitations	2024-09-16
1.39	Updated 11. Network I/O 1. Updated the software logic installation steps to check for ÜberNIC Interface names when using with PCIe 2. Updated the ÜberNIC Sample Applications for ÜberSock 3. Updated the ÜberLoad command line options	2024-09-19
1.40	1. Kernel-integrated Driver: initial implementation (Only PCIe, Not CXL) 2. New Interface naming: unic_<PCIe Address>_<FP#> 3. Support for typical IF admin tools, ex. ethtool, ip, ifconfig, nmcli, nmtui 4. Support for 8 Fiber Pairs: a) Data Only; 8FPs, Max 6 UDP Session & 4 TCP Sessions per-FP b) PTP; Data 7FPs, Max 6 UDP Session & 4 TCP Sessions per-FP c) WRT; Data 7FPs, Max 6 UDP Session & 4 TCP Sessions per-FP 5. Utilization of full routing table in ÜberLoad	2024-10-21

	6. Improved stability of Uber_BMC 7. RX Timestamp (3 mechanisms) in ÜberLoad (TX pending) 8. Epoll: fixed synchronization between kernel and accelerated sockets 9. White Rabbit: initial implementation	
1.41	1. Updated TGPIO_PPS: a) Support for kernel 6.9.1 b) Support for Xeon 6th Gen (Birch Stream) in addition to Xeon 4th & 5th Gen c) Updated installation instructions 2. Added White Rabbit installation and monitoring instructions 3. Resolved bug in uber_bmc_mng 4. Implemented patch to work-around FPGA bug affecting 256B data chunks	2024-12-20
1.42	1. Support for LinuxPTP introduced 2. Expanded supported OS to include Solus and SUSE 3. Added new section OS Requirements in Linux Installation 4. Added new monitoring tool uber_bmc_monitor	2025-01-31
2.00	1. initial v2.0 Release	2025-04-18
2.01	v2.1 Release 1. Removed support for CentOS 2. Performance-optimized MAC-level PCAP'ing (ps) 3. Optimized TX and RX timestamping (ns) for host-consumption 4. Initial support for VLAN (802.1Q) 5. Eliminates dependency on BittWare SDK and Programming Cable 6. Machine-readable monitor for ÜberNIC and ÜberWR 7. Support for AMD EPYC Turin SDCI 8. CXL moved to maintenance-mode due to immaturity of server oem ecosystem support	2025-06-20
2.02	v2.2 Release 1. Section 5, BB, corrected number of supported sessions. 2. Added new Section 7, Board Revisions 3. Section 9, Updated supported CPUs: Xeon 6 6714P, 6527P, 6507P 4. Section 18, D, new programing method power-cycling PCIe slot rather than power-cycling server 5. Section 17, uber_dump, updated into essentially a new application delivering full-line rate TX & RX PCAP'ing (essentially, an internal TAP).	2025-07-14
2.03	v2.3 Release New Features 1. Ethtool support for -m & -S 2. Production 25GbE (Preview Available on 4FP HW Logic) 3. New zip-based HW installation mechanism (with clock configuration) 4. Netmasked multicast function with dedicated API 5. RX buffer size configurable (default: 4M) 6. TimeToLive (TTL) support for TCP, UDP, Multicast, and Unicast 7. ÜberLoad support for eventfd & timerfd Fixes 1. MCE error 0x100000 when using uber_bmc_load_image.sh to program HW logic in FPGA during PCIe slot shutdown and restart 2. Drop indicator when host RX buffer almost full (access via Ethtool -S) 3. TX interframe gap (<40 bits in corner cases) 4. Install error for Fedora versions <40 5, ÜberLoad multicast handling (independent bind/subscription order)	2025-12-10

	6. ÜberLoad tracing itself in a loop on TCP accept 7. Routing on directly reachable route outside the subnet	
2.04	v2.4 Release New Features 1. Ethtool support for -i 2. Background-affinity assignment for ÜberLoad background thread 3. Netmasked Multicast design example Other 1. Added note on link speeds (page 20) 2. Added content related to ÜberNIC ML (to be officially released in v2.5)	2026-02-03

3. Content Conventions

This document includes the following typographic conventions.

Green	Used to note content updated in latest User Guide version
Blue	Used for clickable links.
Purple	Used to draw attention to a specific OS name and version
Bold	Used to highlight a particular topic or point.
Italic	Used when drawing attention to a specific word or phrase such as a filename, command, or a directory name.
`Courier New`	Used for entry that you must type exactly as shown on your system's command line prompt. Upon typing these entries, you must press the **Enter** key on your keyboard for the command to run.
`Courier New Italics`	Used as a place holder for text you must determine and type on your own. Examples include but are not limited to filename, driver name, program name, etc.
Green Highlight	Used to draw attention to important reference
Red Highlight	Used to draw attention to important warning
< >	Used to indicate command key or key

4. Incompatibilities and Known Limitations

A. Incompatibilities

As of the date of this document, ÜberNIC is incompatible with:

- **(For CXL ONLY) IOMMU**:
  Users are strongly advised to disable IOMMU to use ÜberNIC. Failure to do so will lead to unexpected behavior during software installation for ÜberNIC.

- **(For CXL ONLY) Linux kernel version older than 6.4:**
  Kernel versions older than 6.4 do not natively contain the CXL module which is necessary for ÜberNIC's CXL functionality. LMS strongly advises users to use only the supported OS listed in section 7, B to avoid introducing incompatibilities that may affect ÜberNIC's performance.

- **(For CXL ONLY) Servers that do not support CXL:**
  ÜberNIC's CXL requires a server with CXL support to function correctly. Since not all servers have CXL capabilities, using a server that does not support CXL will result in incompatibility issues with ÜberNIC when attempting to use CXL. LMS strongly recommends using only the supported servers and motherboards listed in section 7, A to avoid incompatibilities that may affect ÜberNIC's performance and unexpected behavior.

- **(For CXL ONLY) Operating Systems that do not support CXL**
  ÜberNIC utilizes CXL interface, which is not supported by all operating systems. As a result, ÜberNIC's functionality may be limited or unavailable on certain platforms. LMS

strongly advises users to use only the supported OS listed in section 7, B to avoid unexpected behavior when using ÜberNIC.

B. Known Limitations

The known limitations listed below are subject to change due to the future expansion of support for ÜberNIC, specifically ÜberLoad.

- The MTU limit is1500B, a TCP payload will be properly fragment on TX and a fragmented TCP payload will be properly handled on RX, for UDP payloads fragmentation is not currently implemented for TX or RX.
- No support for blocking socket TIMEOUT (setsockopt/SO_RCVTIMEO, SO_SNDTIMEO)
- Multiple getsockopt/setsockopt options are not implemented and will not return with an error message.
- When using ÜberLoad with TCP, any `bind` that is not specifically bound to an ÜberNIC interface will be handled by the Linux kernel.
- IPV4 jumbo frames not supported
- IPv6 not supported, any IPv6 or dual stack request will be redirected to the Linux kernel.
- In terms of performance, there is a significant difference between thread-safe and non-thread-safe behavior with thread-safe limited (depending on CPU) to approximately 2.4Gbps to 3.2Gbps (depending on message size) while non-thread-safe behavior supporting full like rate regardless of message size.

- Fiber Pair & Supported Sessions:

  Currently, a maximum of eight (8) interfaces (ÜberNIC Ultra+) or four (4) interfaces (ÜberNIC ML) are supported with the fixed configurations detailed below.

1) ÜberNIC Ultra+

a. 10GbE: LinuxPTP:

Fiber Pair 1) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 2) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 3) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 4) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 5) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 6) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 7) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 8) 10GbE; 32 TCP Session and 128 UDP Sessions

b. 10GbE: ÜberPTP or ÜberWR:

Fiber Pair 1) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 2) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 3) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 4) 1GbE Time Sync Network Only
Fiber Pair 5) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 6) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 7) 10GbE; 32 TCP Session and 128 UDP Sessions

Fiber Pair 8) 10GbE; 32 TCP Session and 128 UDP Sessions

c. 25GbE: ÜberWR:

Fiber Pair 1) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 2) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 3) 25GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 4) 1GbE Time Sync Network Only
Fiber Pair 5) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 6) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 7) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 8) 10GbE; 32 TCP Session and 128 UDP Sessions

2) ÜberNIC ML

a. 10GbE: LinuxPTP:

Fiber Pair 1) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 2) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 3) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 4) 10GbE; 32 TCP Session and 128 UDP Sessions

b. 10GbE: ÜberPTP or ÜberWR:

Fiber Pair 1) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 2) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 3) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 4) 1GbE Time Sync Network Only

c. 25GbE: ÜberWR:

Fiber Pair 1) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 2) 10GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 3) 25GbE; 32 TCP Session and 128 UDP Sessions
Fiber Pair 4) 1GbE Time Sync Network Only

5. LMS FTP Drop Site

To access the latest IP logics, applications, drivers, and documentations related to ÜberNIC from LMS, please access the FTP drop site, either via Browser or FTP:

1) Browser: http://ftp.lms.io/

2) FTP:

Address	ftp.lms.io
Username	ubernic
Password	lmspw
Port	21 (i.e. FTP, not SFTP)

Passive	Yes

The following is the tree structure of the ftp site:

```
├── BittWare SDK/BMC
│   └── altera-related
│       ├── bmc
│       │   └── bmc30-0.8.0+16.zip
│       ├── sdk
│       │   ├── Ubuntu: bittware-sdk-ubuntu20.04-2024.1.0.deb
│       │   ├── Ubuntu: bittware-sdk-ubuntu22.04-2024.1.0.deb
│       │   ├── RHEL: bittware-sdk-2024.1.0-1.el8.rpm
│       │   └── RHEL: bittware-sdk-2024.1.0-1.el9.rpm
│       └── csp
│           ├── Ubuntu: bittware-csp-ia440i-ubuntu20.04-2024.1.0.deb
│           ├── Ubuntu: bittware-csp-ia440i-ubuntu22.04-2024.1.0.deb
│           ├── RHEL: bittware-csp-ia440i-2024.1.0-1.el8.rpm
│           └── RHEL: bittware-csp-ia440i-2024.1.0-1.el9.rpm
├── Documentation
│   ├── IA-440i_Hardware_Reference_Guide.pdf
│   ├── IA-440i_LMS_heatsink_removal_and_assembly.pdf
│   ├── ÜberNIC_Board_Management_and_Logic_Programming_User_Guide_v20240422.pdf
│   └── UberNIC_v2.0_User_Guide_v20250714.pdf
├── TaSR
│   ├── 20250522_TaSR_IP.rbf
│   └── 20250522_TaSR_Driver.tar.gz
├── UberNIC Driver/API_Logic
│   └── 20251210_UberNIC_v2.3.tar.gz
│   └── 20260130_UberNIC_v2.4.tar.gz
└── UberNIC IP_Logic
    ├── 20250609_Rev0_Golden_Image.rbf
    ├── 20250609_Rev1_Golden_Image.rbf
    ├── 20251210_UberNIC_v2_3_PCIe_LinuxPTP.zip
    ├── 20251210_UberNIC_v2_3_PCIe_LinuxPTP_4FP.zip
    ├── 20251210_UberNIC_v2_3_PCIe_UberPTP.zip
    ├── 20251210_UberNIC_v2_3_PCIe_UberWR.zip
    ├── 20251210_UberNIC_v2_3_PCIe_UberWR_25G_4FP.zip
    ├── 20260120_UberNIC_v2_4_PCIe_LinuxPTP.zip
    ├── 20260120_UberNIC_v2_4_PCIe_LinuxPTP.rbf
    ├── 20260120_UberNIC_v2_4_PCIe_UberWR.zip
```

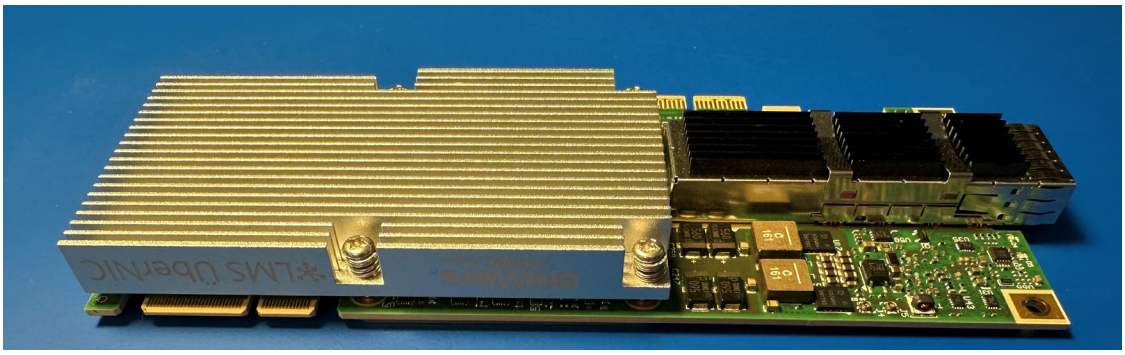
├── 20260120_UberNIC_v2_4_PCIe_UberWR.rbf
├── 20260120_UberNIC_v2_4_PCIe_UberPTP.zip
├── 20260120_UberNIC_v2_4_PCIe_UberPTP.rbf
├── 20260120_UberNIC_v2_4_PCIe_LinuxPTP_4FP.zip
├── 20260120_UberNIC_v2_4_PCIe_LinuxPTP_4FP.rbf
├── 20260120_UberNIC_v2_4_PCIe_UberWR_25G_4FP.zip
└── 20260120_UberNIC_v2_4_PCIe_UberWR_25G_4FP.rbf

6. FPGA Boards

A. ÜberNIC Ultra+ (BittWare IA-440i)

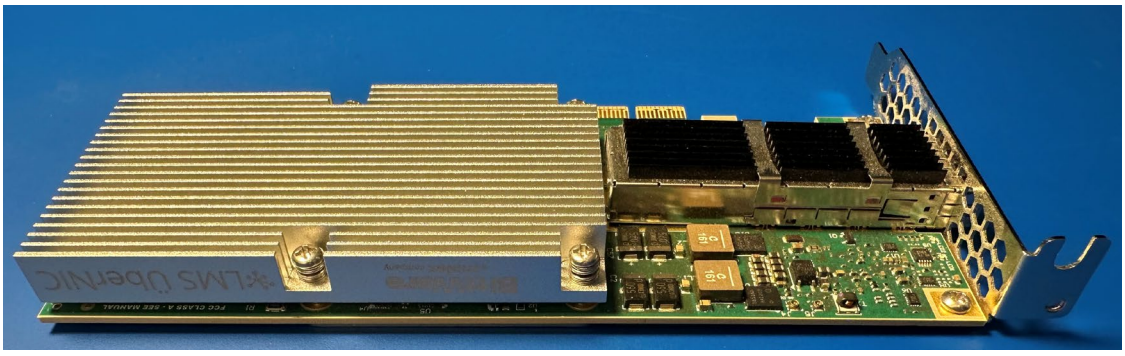
1) Revision 0

FPGA: Altera Agilex 7 AGI 023, Speed Grade 2
Note: Interconnection Golden Fingers at bottom-left of photo



2) Revision 1

FPGA: Altera Agilex 7 AGI 023, Speed Grade 1
Note: No Interconnection Golden Fingers at bottom-left of photo



B. ÜberNIC ML (AMD X3522pv)

Coming Soon

7. Supported Servers, Motherboards, and Operating Systems

A. Servers and Motherboards

					PTM		
Vendor	Model	CPU	PCIe	CXL	PCIe	CXL	TGPIO
ASRock	SP2C741D16-2T	Xeon 4th & 5th Scalable	Y	Y	Y	Y	Y
ASRock	W7908UD-1L1N2T	Xeon 4th & 5th Workstation	Y	Y	Y	Y	Y
ASRock	GNRD8-2L2T	Xeon 6th	Y	Y	Y	TBD	Y
Asus	Pro W790E-SAGE	Xeon 4th Workstation	Y	Y	Y	Y	-
Asus	X670-P-CSM	Ryzen 9	Y	-	Y	-	-
Asus	WRX90E-SAGE	Ryzen ThreadRipper PRO	Y	TBD	Y	TBD	-
Dell[1]	R760 / R660	Xeon 4th Scalable	Y	Y[1]	Y	N	-
Gigabyte	MS03-CE0	Xeon 4th Scalable	Y	Y	Y	N	-
Hypertec	HF-410	Xeon 4th Workstation	Y	Y	Y	Y	-
Jabil	J322-S	Xeon 4th & 5th Scalable	Y	Y	Y	Y	Y
Jabil	J422-S	Xeon 6th (Sierra Forest)	Y	Y	Y	Y	Y
Jabil	J422-S	Xeon 6th (Granite Rapids)	Y	Y	Y	Y	Y
Jabil	J421E-S	AMD EPYC Zen 5 Turin 9575F	Y	-	-	-	-
SuperMicro	X13SEI-F	Xeon 4th & 5th Scalable	Y	Y	Y	N	-
CPUs Supporting AVX2 Intel Xeon 2nd & 3rd Scalable Intel Core i5, i7, i9 AMD Ryzen 9 AMD WX			Y	-	-	-	-

Note: 1. Requires prototype ID Module available from LMS or Dell

B. Operating Systems

OS	PCIe	CXL	Notes on CXL/PCIe Support
RHEL 10.x	Y	TBD	
RHEL 9.x	Y	Y	RHEL 9.3 and greater natively support CXL. Thus, it is highly advised for users to use at least these versions.
RHEL 8.7 >	Y	Y	CXL module is not available natively in RHEL 8, so CXL module needed to be installed separately. Please go to Section 9, Getting Started, F: Installing CXL Module for Linux Kernel for the installation steps.
AlmaLinux 10.x	Y	TBD	
AlmaLinux 9.x	Y	Y	
AlmaLinux 8.x	Y	N	
Rocky Linux 10.x	Y	TBD	
Rocky Linux 9.x	Y	Y	
Rocky Linux 8.x	Y	N	

Fedora 38	Y	Y	
Fedora 39	Y	Y	
Solus 24-10	Y	N	
SUSE 15-SP5	Y	N	
Ubuntu 22.04	Y	N	
Ubuntu 23.10	Y	Y	
Debian 12	Y	Y	By default, IOMMU is enabled in Debian 12. To avoid adverse effects on ÜberNIC's CXL performance, IOMMU must be disabled.

Note: Y = Supported; N = Not Supported.

8. Qualified Vendor List

CPU	Type
Intel Xeon 4th Gen - 6430	Server
Intel Xeon 4th Gen - 6434	Server
Intel Xeon 4th Gen - 8426Y	Server
Intel Xeon 4th Gen - 8470L	Server
Intel Xeon 4th Gen - 8480+	Server
Intel Xeon 4th Gen - 6458Q	Server
Intel Xeon 4th Gen - 6448Y	Server
Intel Xeon 4th Gen - 6442Y	Server
Intel Xeon 4th Gen - W7-2495X	Workstation
Intel Xeon 4th Gen - W9-3495X	Workstation
Intel Xeon 5th Gen - W7-2595X	Workstation
Intel Xeon 5th Gen - 6542Y	Server
Intel Xeon 5th Gen - 6558Q	Server
Intel Xeon 5th Gen - 8562Y	Server
Intel Xeon 6th Gen - 6710E	Server
Intel Xeon 6th Gen – 6527P	Server
Intel Xeon 6th Gen – 6714P	Server
Intel Xeon 6th Gen – 6507P	Server
AMD Ryzen 9 7950X	Desktop
AMD Ryzen ThreadRipper PRO 7965WX	Workstation
AMD EPYC Turin 9575F	Server

Transceiver & Fiber	Type	Vendor
200G-SR8 QSFP-DD P/N 1837150046	Transceiver	Molex
8 FP Cable P/N 1062836338	Breakout Cable	Molex
10G-SR SFP+ P/N UACC-OM-MM-10G-D-2	Transceiver	Ubiquiti
10G-SR SFP+ P/N S+85DLC03D	Transceiver	MicroTik
10G-SR SFP+ 455883-B21	Transceiver	SwissGBIC
10G-SR SFP+ P/N SFP-10GSR-85	Transceiver	FS.com
25G-SR SFP28 P/N SFP28-25GSR-85	Transceiver	FS.com
40G-SR4 QSFP+ P/N QSFP-SR4-40G	Transceiver	FS.com
100G-SR4 QSFP28 P/N QSFP28-SR4-100G	Transceiver	FS.com
QDD-SR4-2x100G	Transceiver	FS.com
40G QSFP+ to 10G SFP+ P/N CVR-QSFP-SFP10G	Converter	FS.com
40G QSFP+ to 10G SFP+ P/N QSFP-SFP10G-CVR	Converter	FS.com

100G QSFP28 to 25G SFP28 P/N VR-QSFP28-SFP28	Converter	FS.com
4 FP Cable P/N 8FMTPLCOM4	Breakout Cable	FS.com
8 FP Cable P/N 24FMTPLCOM4	Breakout Cable	FS.com
40G to 4x10G Passive DAC P/N QSFP-40G-4SPC03	Breakout Cable	FS.com
40G to 4x10G Passive DAC P/N QSFP-4SFP10G-CU3M-C	Breakout Cable	SwissGBIC
100G to 4x25G Passive DAC P/N Q-4S28PC02	Breakout Cable	FS.com

9. Supported Capabilities and Availability Roadmap

Please refer to the Deployment Timeline to get the specific dates of availability.

A. Network Link Speeds

Speed	Standard	Availability
1GbE Over Optical Fiber	IEEE-802.3z	Live
1GbE Over Twisted Pair	IEEE-802.3ab	TBD
2.5GbE Over Twisted Pair	802.3bz	TBD
5GbE Over Twisted Pair	802.3bz	TBD
10GbE Over Optical Fiber	802.3ae	Live
10GbE Over Twisted Pair	802.3an	TBD
25GbE Over Optical Fiber (SR) wo/FEC	802.3by	Live
25GbE Over Optical Fiber (SR) w/FEC	802.3by	TBD
25GbE Over Optical Fiber (LR/ER) wo/FEC	802.3by	Live
25GbE Over Optical Fiber (LR/ER) wFEC	802.3by	TBD
40GbE Over Optical Fiber (LR4)	802.3ba	TBD
40GbE Over Optical Fiber (ER4)	802.3bm	TBD
40GbE Over Optical Fiber (FR)	802.3bg	TBD
50GbE Over Optical Fiber (SR/LR/FR)	802.3cd	TBD
50GbE Over Optical Fiber (ER)	802.3cn	TBD
100GbE Over Optical Fiber		ETA: 2026 Q4
200GbE Over Optical Fiber		ETA: 2026 Q4
400GbE Over Optical Fiber		ETA: 2026 Q4

Note:

Unlike traditional multi-rate interfaces, only fixed-rate are currently supported. Interfaces are configured, within the HW logic file, to 1GbE, 10GbE, or 25GbE. This methodology is implemented to ensure optimal and deterministic low-latency performance. To provide the absolute best performance, the interface must remain locked to a single line rate.

Any mechanism that enables dynamic rate switching—such as dual-rate support or auto-negotiation—introduces additional logic layers, switching cycles and recovery sequences. These mechanisms add variable latency, which is unacceptable for applications requiring consistent and minimal processing delay.

To ensure correct operation, the user must install appropriate optical cabling and optics capable of supporting the required rate distribution on the transceiver (1GbE / 10GbE / 25GbE as

applicable), for example, a typical implementation utilizes a 200G-SR8 QSFP-DD (emulating NRZ) supporting 8 (eight) individual interfaces via a breakout cable.

Why a Fixed Rate?

B. Network Fiber Configurations

Speed	Availability
40G QSFP+ to 10G SFP+ Adapter Converter Module	Live
40G QSFP to 4x 10G via Breakout Cable	Live
100G QSFP28 to 25G SFP28 Adapter Converter Module	Live
200G QSFP-DD to 8x 10G via Breakout Cable	Live
100G QSFP-DD to 4x 25G via Breakout Cable	Live
200G QSFP-DD to 8x 25G via Breakout Cable	Live

C. PCIe Transfer Mechanisms

Item	Availability
PCIe Gen 3 x16 Lanes	Live
PCIe Gen 4 x16 Lanes	Live
PCIe Gen 5 x16 Lanes	Live
CXL v1.1 Type 1 CXL.cache + CXL.mem x16 Lanes	TBD
CXL v1.1 Type 2 CXL.cache + CXL.mem x16 Lanes	Live
CXL v1.1 Type 3 CXL.io + CXL.mem x16 Lanes	Live
CXL v2.0 Type 1 CXL.cache + CXL.mem x16 Lanes	TBD
CXL v2.0 Type 2 CXL.cache + CXL.mem + CXL.io x16 Lanes	TBD
CXL v2.0 Type 3 CXL.io + CXL.mem x16 Lanes	TBD

D. Features and Standards-Compliance

Item	Standard	Availability
IPv4	RFC-791	Live
IPv4 Explicit Congestion Notification	RFC-3168	TBD
IPv6	RFC-8200	TBD
TCP	RFC-9293	Live
TCP Extensions for High Performance	RFC-7323	TBD
TCP Defending Sequence Number Attacks	RFC-1948	TBD
TCP Denial-of-Service Considerations	RFC-4732	TBD
TCP SYN Flooding Attacks and Mitigations	RFC-4987	TBD
TCP Network Ingress Filtering	RFC-2827	Live:
TCP Multihomed Networks Ingress Filtering	RFC-3704	TBD
TCP Extensions for High Performance	RFC-1323	TBD
TCP Extensions for Long-Delay Paths	RFC-1072	TBD
TCP Selective Acknowledgment Options	RFC-2018	TBD
TCP Congestion Control	RFC-5681	TBD
TCP Retransmit Timer	RFC-6298	Live
UDP	RFC-768	Live
VLAN Increasing Max Frame Size to 1522	802.3ac	Live
VLAN Tagging	IEEE-802.1Q	Live
VLAN QinQ	IEEE-802.1ad	TBD
Link Aggregation	IEEE-802.1AX	TBD
Link Bonding	IEEE-802.3ad	TBD

ARP	RFC-826	Live
ICMP	RFC-792	Live
IGMP	RFC-3376	Live
Precision Time Protocol (PTPv2.1)	IEEE 1588-2019 (ex-HA)	Live
Synchronous Ethernet (SyncE)	G.8262/G.8264	Live
White Rabbit	IEEE 1588-2019 (HA)	Live

E. User Network I/O Technology

Item	Availability
ÜberL2 (Host, Kernel-Stack)	Live
ÜberSock (Host, Kernel Bypass C API)	Live
ÜberL2 (Host, Sockets Compliant, LD_PRELOAD)	Live
ÜberFPGA FDK	ETA: 2026 Q3

10. Getting Started

This section explains how to prepare the host system for use with ÜberNIC and then how to handle and install ÜberNIC into the system. To communicate with the host system, ÜberNIC can be set to communicate via CXL or PCIe. Unless it is mentioned explicitly that the steps listed in the sub-section are for CXL, users are expected to follow each sub-section carefully, as these steps are meant to be followed regardless of interface chosen for ÜberNIC.

Once the host system has been configured with the necessary BIOS settings and software, ÜberNIC can be unboxed and installed in the system.

A. Hardware Installation

Before installing ÜberNIC in your system, make sure to handle the board with precaution to prevent damage from electro-static discharge (ESD) when unpacking it.

Please refer to Bittware's IA-440i Getting Started Guide or IA-780i Getting Started Guide for further details on how to install ÜberNIC.

1) Sensors and Thresholds

a) ÜberNIC Ultra+ (BittWare IA-440i)

Name	Unit	Critical	Fatal
FPGA Temperature	C	90.0	100.00
QSFP Temperature	C	67.5	70.00
Total Input Power	W	-	75.00
VCC 12V0 PCIe Voltage	V	13.4	13.50
VCC 12V0 PCIe Current	A	6.5	7.00
VCC 12V0 PCIe Power	W	-	66.00
VCC 3V3 PCIe Voltage	V	-	3.96
VCC 3V3 PCIe Current	A	-	-
VCC 3V3 PCIe Power	W	-	9.00
VCC Core Voltage	V	-	0.92

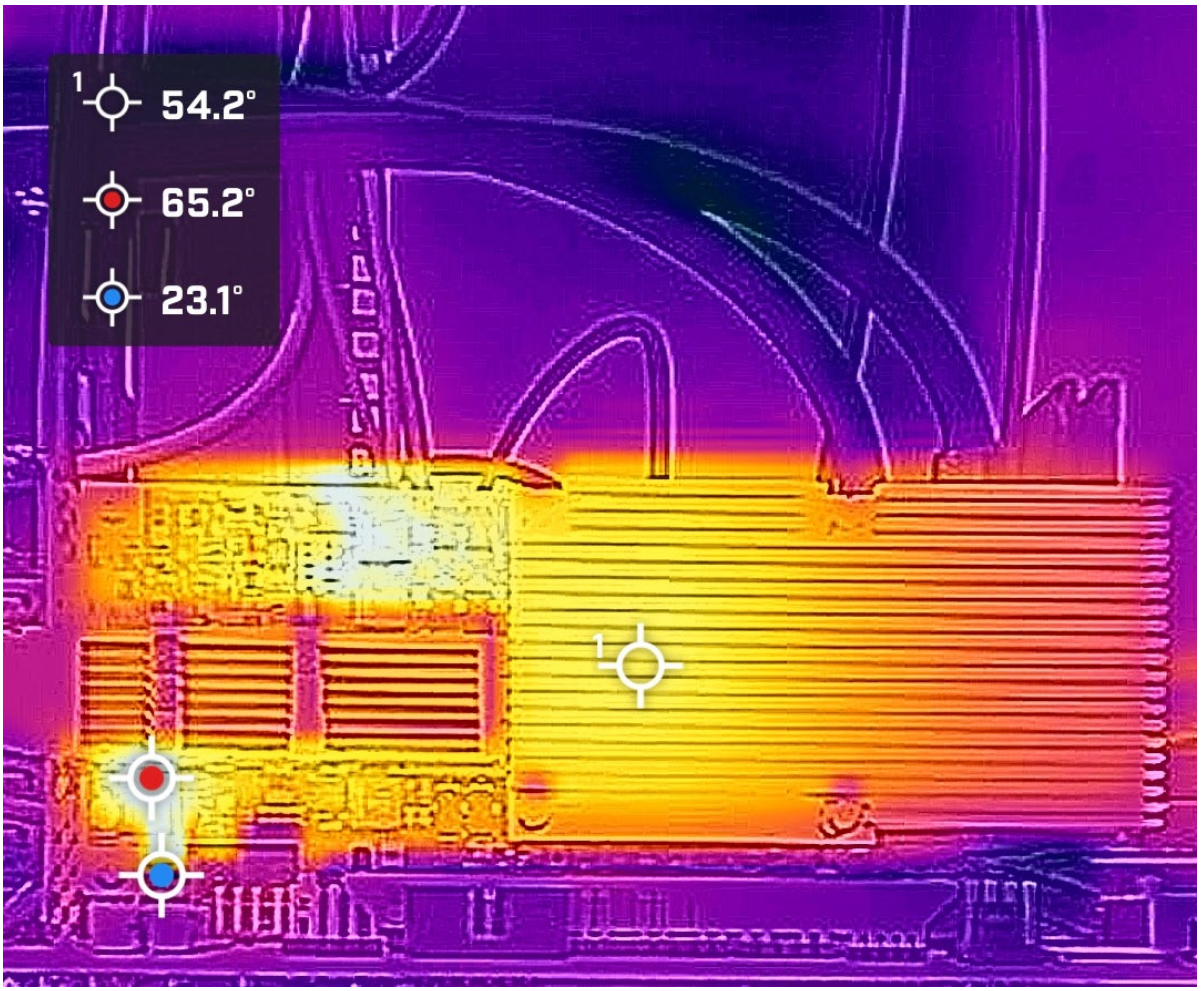
VCC Core Current	A	-	68.00
SmartVID Voltage	V	-	-
Front Temperature	C	-	100.00
Note: C=Celsius; W=Watt; V=Volt; A=Amp			

b) ÜberNIC ML (AMD X3522pv)

Coming Soon

2) Cooling

Like all semiconductors, the FPGA chip within ÜberNIC generates heat. This heat typically ranges from 50 °C to 90 °C. To avoid damage to ÜberNIC and its FPGA chip it is absolutely critical to ensure adequate cooling at all times – especially when ÜberNIC is installed within a customized chassis or an open skeleton chassis environment. To avoid doubt, ensure that fans utilized to cool ÜberNIC are initially manually set to 85% or greater of their duty cycle or maximum rpm and then adjusted to ensure adequate cooling at an acceptable duty cycle.



FLIR photo of ÜberNIC on an open-skeleton chassis with 140mm fan operating at 3,000 rpm.

B. Linux Installation

Skip this section if you already have Linux installed in your host system.

When installing an operating system (OS), the OS source media must be available to the server. There are a few installation files options, such as:

- Optical DVD drive
- Bootable USB device
- PXE network boot
- Mounted ISO-file (through Virtual Media function)

The following describes how to install an operating system (OS) remotely (using Red Hat 9.3 as an example) with a bootable USB device using the freeware tool called Rufus. For servers without an optical drive (DVD) or a configured remote access controller, this serves as an alternative way of updating all server firmware.

1) OS Requirements

Every Linux distribution is at least slightly different than others and the components included within a particular build ex. RHEL Minimum vs SLES Online will vary.

Specific dependencies required to install the ÜberNIC Driver/API/Tools Package are noted below along with OS-specific requirements.

a) Cross-OS Requirements

- make must be installed
- gcc must be installed
- kernel headers (for the currently running version) must be installed
- iommu must be disabled

b) SUSE SLES 15 SP5

ÜberNIC ships with a kernel module which must be loaded for ÜberNIC to function. This module is as yet not “officially” supported by SUSE.

For SUSE, all supported Linux kernel modules (modules shipped with SLES) contain an internal "supported" flag which denotes that module as being officially supported by SUSE. Beginning with SLES 11, 3rd party modules that do not contain this flag are prevented from being loaded (automatically or manually) by the modprobe command.

This limitation is overridable.

In order for the ÜberNIC kernel module to be properly loaded, one of the following must be implemented:

Method 1

Use --allow-unsupported parameter by typing the following commands:

1) modprobe -r <module_name>

2) modprobe <module_name> --allow-unsupported

Method 2

Modify the system environment config to load an “unsupported” driver insertion; add the following line to /etc/modprobe.d/10-unsupported-modules.conf:

1) allow_unsupported_modules 1

2) reboot

2) Red Hat 9.4

1. From the Red Hat Enterprise Linux ISO for Server and Workstations section, go to “Other RHEL versions” and then download RHEL 9.4 x86_64 DVD iso.
2. Insert the USB device on your laptop.
3. Start the Rufus tool.
   a. Make sure to select the correct USB device on Rufus. If Rufus is unable to find your desired USB device, click on "Show advanced drive properties" and then enable "List USB Hard Drives".
   b. Under “Boot selection”, choose the ISO file that was downloaded previously in step 1. Rufus will select the appropriate settings.
   c. Click on the “START” button to launch the process. Make sure to read all the popup messages carefully before proceeding to click the “OK” button.
   d. When Rufus has finished writing the USB, the status bar will be filled with green and the word “READY” will appear at the center. Select “CLOSE” to complete the writing process.
4. Plug in the USB drive into your desired server. Please go to section 7 to see which servers/motherboard that are validated with CXL.
5. Restart the server and boot from the created bootable USB device.
   a. If the Bootable ISO fails to run after being applied to a bootable USB, change the server boot mode to BIOS or UEFI mode and attempt the procedure again. If the Bootable ISO continues to fail, it is recommended to download the ISO again or try applying the ISO image to a different USB drive.
6. From the boot menu, select “Install Red Hat Enterprise Linux 9.4”.
7. Once the OS is installed successfully, select your language and location, and click on the “continue” button. This will open an “Installation Summary” window and display the default values for each setting.
8. Select system and go to “Installation Destination”. In the “Local Standard Disks” pane, select the target disk (which is your desired server/motherboard), modify the storage configuration to your liking and then click the “Done” button.
9. Next, select “System” and go to “Network & Hostname” to open the network and hostname window.
   a. In the network and hostname window, toggle the Ethernet switch to ON, and then click the “Done” button. The installer will connect to an available network and configure the devices available on the network.
10. Modify the user settings to add a root password for the server (if necessary) and to create a user account for the installation process to complete. If necessary, you can also make this newly created user account as an administrator.
11. Click the “Begin Installation” button to start the OS installation and wait until the installation is completed. This process usually takes a few minutes.
12. When the installation process is complete, click the “Reboot System” button to restart the system.

13. After the initial setup window, accept the licensing agreement and register your system with Red Hat. This is important to ensure that you can update the software later.
    a. Alternatively, you can also register the system using a command line by typing the following:

```
sudo subscription-manager register --username username --password password --auto-attach
```

    b. If the registration process fails, please contact Red Hat for further assistance.

14. Once the OS is installed, update the installed packages to the latest version using the package manager (yum/dnf).

    For example, using dnf:

```
sudo dnf update
```

    Note: To ensure you are accessing all available packages, please add using:

```
sudo subscription-manager repos --enable codeready-builder-for-rhel-9-x86_64-rpms
```

C. Minimal Software Installation

The following describes the necessary minimal software installation, based on the OS.

For **Red Hat, Fedora, CentOS Stream, AlmaLinux and Rocky Linux**, go to **section 1** and only run the commands listed in section 1.

For **Ubuntu and Debian**, go to **section 2** and only run the commands listed in section 2.

1) Red Hat, Fedora, CentOS Stream, AlmaLinux, Rocky Linux

a) Update your repositories to the latest version:

```
sudo dnf update -y
```

b) Install the “kernel-dev” package:

```
sudo dnf install kernel-devel -y
```

c) Reboot the system to make sure the changes are reflected:

```
sudo reboot
```

2) Debian, Ubuntu

a) Update your repositories to the latest version:

```
sudo apt update && sudo apt upgrade -y
```

b) Install the kernel header files:

```
sudo apt-get update && sudo apt-get install linux-headers-$(uname -r)
```

c) Install the “build-essential” package:

```
sudo apt-get install build-essential -y
```

d) Reboot the system to make sure the changes are reflected:

```
sudo reboot
```

D. Verifying ÜberNIC Accessibility to Host OS

The following describes how to verify ÜberNIC’s connection with the host server. These steps are intended to work in a Linux subsystem so you must have Linux installed on your server system. Please refer to the previous section on how to install Linux on your server for further details.

1) Verify ÜberNIC Visible in OS

Open a terminal and verify ÜberNIC’s connection to the server’s PCI subsystem using the `lspci` command by referencing the vendor number, “4c4d”. The Vendor Code “4c4d” is officially registered to “Liquid-Markets GmbH”.

Example Command:

```
lspci -vvv -d 4c4d:*
```

Output Example:

```
be:00.1     Ethernet controller: Liquid-Markets GmbH Device 9999
        Subsystem: Liquid-Markets GmbH Device 9999
        Control: I/O- Mem- BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr+ Stepping-
        SERR+ FastB2B- DisINTx-
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort-
        >SERR- <PERR- INTx-
        NUMA node: 0
        Region 0: Memory at 217effc00000 (64-bit, prefetchable) [disabled] [size=2M]
        Region 2: Memory at 217effa00000 (64-bit, prefetchable) [disabled] [size=2M]
        Capabilities: <access denied>
```

Take note of the PCIe address which is displayed in this format “[bus number]:[device number].[function number]” (highlighted in green) in the output. We will be using these values for our next step.

Syntax: `lspci -vvv -s (`*`bus number`*`):(`*`device number`*`).*`

Example Command:

```
lspci -vvv -s be:00.*
```

Output Example:

```
be:00.0     CXL: Intel Corporation Device 0ddb (rev 01) (prog-if 10 [CXL Memory Device (CXL 2.x)])
            Subsystem: Intel Corporation Device 0000
            Control: I/O- Mem- BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr+ Stepping-
            SERR+ FastB2B- DisINTx-
            Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort-
            >SERR- <PERR- INTx-
            Interrupt: pin A routed to IRQ 255
            NUMA node: 0
            Region 0: Memory at f9400000 (32-bit, non-prefetchable) [disabled] [size=2M]
            Region 2: Memory at 217effe00000 (64-bit, prefetchable) [disabled] [size=128K]
            Capabilities: <access denied>
be:00.1         Ethernet controller: Liquid-Markets GmbH Device 9999
            Subsystem: Liquid-Markets GmbH Device 9999
            Control: I/O- Mem- BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr+ Stepping-
            SERR+ FastB2B- DisINTx-
            Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort-
            >SERR- <PERR- INTx-
            NUMA node: 0
            Region 0: Memory at 217effc00000 (64-bit, prefetchable) [disabled] [size=2M]
            Region 2: Memory at 217effa00000 (64-bit, prefetchable) [disabled] [size=2M]
            Capabilities: <access denied>
```

E. Disabling IOMMU

By default, ÜberNIC is incompatible with systems with IOMMU enabled and active. If you don't disable IOMMU in your system before the software installation, the character driver for ÜberNIC will not be created and you will not be able to use ÜberNIC with your system.

There are multiple ways to disable IOMMU in your system, below are examples.

1) Disable in BIOS

Set 'IOMMU' to 'Disabled' in your BIOS by referring to your system's manual.

2) Disable in Kernel Command Line

Disable IOMMU in your GRUB boot parameters by appending '`intel_iommu=off`' (for Intel CPUs) or '`amd_iommu=off`' (for AMD CPUs) to GRUB_CMDLINE_LINUX in /etc/default/grub.

a) **Ubuntu/Debian – Add Kernel Parameter**:

i. Open the GRUB bootloader file in a text editor with root permissions:

```
sudo vi /etc/default/grub
```

ii. Find the line starting with GRUB_CMDLINE_LINUX_DEFAULT and append:

Intel CPUs:

```
intel_iommu=off
```

AMD CPUs:

```
amd_iommu=off
```

Example:

GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=off"

iii. Save the file and close the editor:

```
<Esc> <:wq> <Enter>
```

iv. Update GRUB's configuration file:

```
sudo update-grub
```

v. Reboot for changes to take effect

```
reboot
```

b) **Red Hat Family – Add Kernel Parameter**:

i. Add argument to the kernel command line:

```
sudo grubby --update-kernel=ALL --args="intel_iommu=off"
```

or

```
sudo grubby --update-kernel=ALL --args="amd_iommu=off"
```

ii. Check if the kernel argument was added successfully:

```
sudo grubby --info=ALL
```

Example output:

[lms@localhost ~]$ sudo grubby --info=ALL
index=0
kernel="/boot/vmlinuz-5.14.0-427.26.1.el9_4.x86_64"
args="ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=UUID=ed33d974-6bc0-4434-9132-2ef3be7e78ca rhgb quiet $tuned_params intel_iommu=off"
root="UUID=b0222740-f72c-4e67-a208-06e226b29a29"
initrd="/boot/initramfs-5.14.0-427.26.1.el9_4.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (5.14.0-427.26.1.el9_4.x86_64) 9.4 (Plow)"
id="456e442d42f749bab46aaf917d57654c-5.14.0-427.26.1.el9_4.x86_64"
index=1
kernel="/boot/vmlinuz-5.14.0-427.13.1.el9_4.x86_64"
args="ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=UUID=ed33d974-6bc0-4434-9132-2ef3be7e78ca rhgb quiet $tuned_params intel_iommu=off"
root="UUID=b0222740-f72c-4e67-a208-06e226b29a29"

initrd="/boot/initramfs-5.14.0-427.13.1.el9_4.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (5.14.0-427.13.1.el9_4.x86_64) 9.4 (Plow)"
id="456e442d42f749bab46aaf917d57654c-5.14.0-427.13.1.el9_4.x86_64"
index=2
kernel="/boot/vmlinuz-0-rescue-456e442d42f749bab46aaf917d57654c"
args="ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=UUID=ed33d974-6bc0-4434-9132-2ef3be7e78ca rhgb quiet intel_iommu=off"
root="UUID=b0222740-f72c-4e67-a208-06e226b29a29"
initrd="/boot/initramfs-0-rescue-456e442d42f749bab46aaf917d57654c.img"
title="Red Hat Enterprise Linux (0-rescue-456e442d42f749bab46aaf917d57654c) 9.4 (Plow)"
id="456e442d42f749bab46aaf917d57654c-0-rescue"

c) Reboot for changes to take effect:

```
reboot
```

3) Check Kernel Command Line

Command:

```
cat /proc/cmdline
```

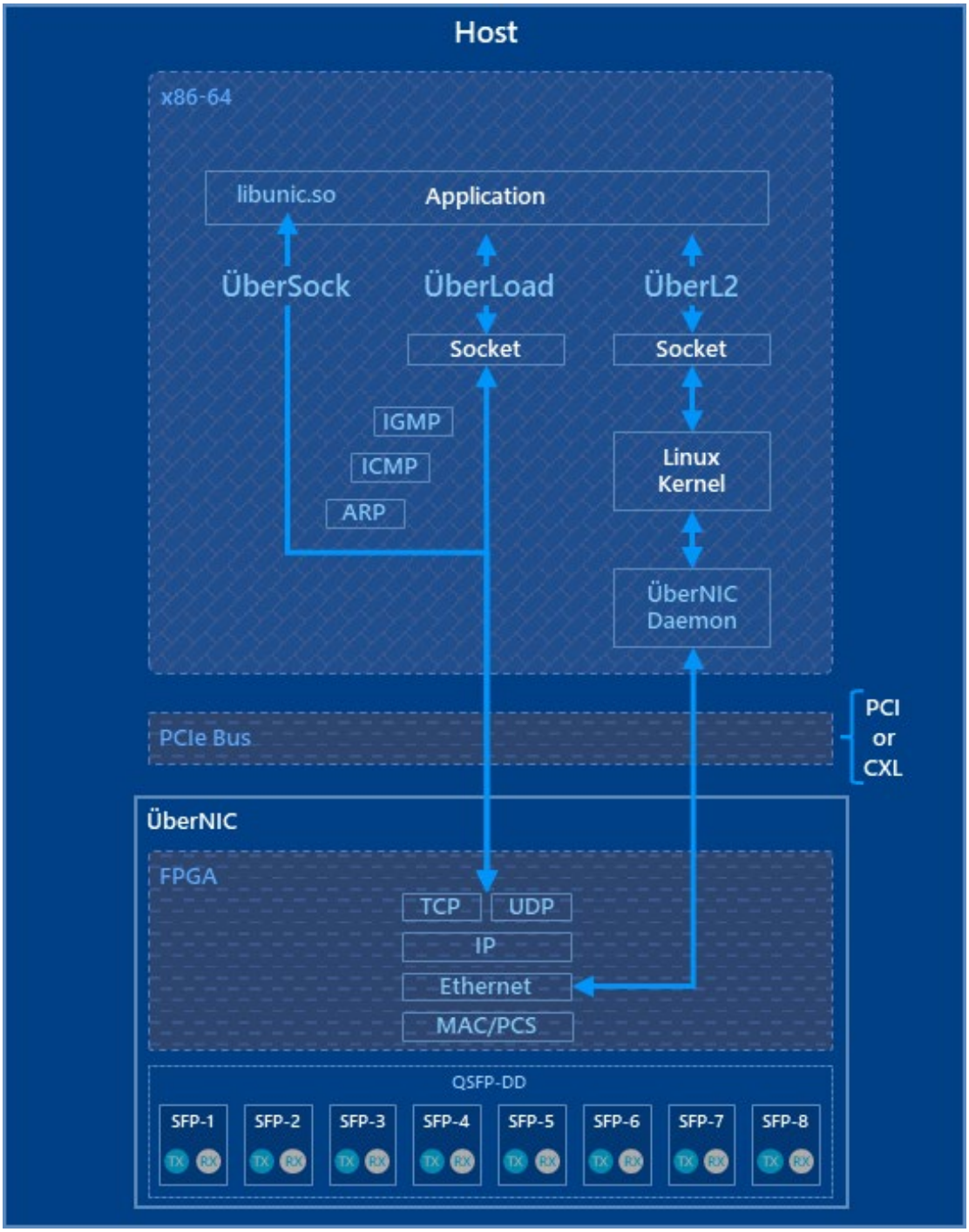
The output should have the boot parameter that we added previously, which in the following example is highlighted in green:

BOOT_IMAGE=(hd2,gpt2)/vmlinuz-5.14.0-427.26.1.el9_4.x86_64 root=UUID=b0222740-f72c-4e67-a208-06e226b29a29 ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=UUID=ed33d974-6bc0-4434-9132-2ef3be7e78ca rhgb quiet intel_iommu=off

11. Network I/O

A. Introduction

ÜberNIC implements three mechanisms for a Linux application to interact with the network, as described below. The totality of these methods serves to avoid inefficient use of host resources, including CPU cores and the PCIe bus, while at the same time minimizing the negative impact of context switching, system calls, and resource contention, among others. These three different mechanisms can be used depending on the requirements of your system.



1) ÜberL2

ÜberL2 is a Linux Kernel-bound mechanism supporting non-latency-sensitive network interaction, including the use of, if desired, standard, sockets-compliant, Linux kernel-bound network stack(s), and the overwhelming majority of Linux network commands including ping, nslookup, ip, ethtool, dnf update, etc.

2) ÜberSock

ÜberSock is an ultra-low-latency C API providing an interface to TCP and UDP functions implemented within the FPGA fabric of ÜberNIC. ÜberSock directly connects a user-space application to a fully offloaded network stack implementation running on the ÜberNIC FPGA via kernel bypass techniques such that data is transmitted to, or received from, the network without any involvement of the Linux Kernel.

3) ÜberLoad

ÜberLoad is a low-latency sockets-emulation mechanism providing an interface to TCP and UDP functions implemented within the FPGA fabric of ÜberNIC. ÜberLoad directly connects a user-space application to a fully offloaded network stack implementation running on the ÜberNIC FPGA via kernel bypass techniques such that data is transmitted to, or received from, the network without any involvement of the Linux Kernel.

Note: If you do not specifically modify an application to utilize either ÜberSock or ÜberLoad, it will, by default, utilize ÜberL2.

B. Supported Sessions

1) ÜberNIC Ultra & ÜberNIC Ultra+ (IA-440i)

ÜberNIC supports multiple configurations of fibers and sessions, below are default:

Number of Fibers	Configuration	Session Partition Per Fiber				
1	QSFP-DD to SFP + Adapter	UDP sessions: 6 TCP session: 4 ÜberL2: 1				
4	40G QSFP to 4x 10G via Breakout Cable		UDP	TCP	UberL2	1GbE*
		Fiber Pair 1	128	32	1	0
		Fiber Pair 2	128	32	1	0
		Fiber Pair 3	128	32	1	0
		Fiber Pair 4	0	0	0	1
8	200G QSFP-DD to 8x 10G via Breakout Cable		UDP	TCP	UberL2	1GbE*
		Fiber Pair 1	128	32	1	0
		Fiber Pair 2	128	32	1	0
		Fiber Pair 3	128	32	1	0
		Fiber Pair 4	-	-	-	1

		Fiber Pair 5	128	32	1	0
		Fiber Pair 6	128	32	1	0
		Fiber Pair 7	128	32	1	0
		Fiber Pair 8	128	32	1	0

2) ÜberNIC Ultra Wide & ÜberNIC Ultra+ Wide (IA-780i)

--Coming Soon—

C. Software Logic Installation

1) Updating Software Logic From Prior Installation

If you already have the previous software logic installed, and you would like to update it, please follow the following steps carefully:

a) Enter the ÜberNIC Driver and Supporting SW directory:

```
cd ubernic
```

b) Uninstall the previous ÜberNIC driver:

```
sudo ./uninstall.sh
```

c) Navigate to the parent directory that contains the ÜberNIC Driver and Supporting SW tar:

```
cd ../
```

d) Remove ÜberNIC directory and contents

```
rm -f -R ubernic
```

e) Check ÜberNIC kernel module removed; manually remove if necessary”

```
lsmod | grep ubernic
```

If kernel module still installed:

```
rmmod ubernic
```

f) Unpack the ÜberNIC Driver and Supporting SW tar:

```
tar -xvzf YYYYMMDD_UberNIC.tar.gz
```

g) Enter the ÜberNIC Driver and Supporting SW directory:

```
cd ubernic
```

h) Install ÜberNIC driver, api, and tools:

```
./install.sh
```

Note: Hardware timestamping is automatically enabled for all ÜberNIC models.

i) Reboot system:

```
sudo reboot
```

2) Installation

Important: Before installing the software logic, please ensure that IOMMU is disabled since the driver does not support IOMMU. Failure to do so will result in unexpected behavior during software logic installation and when using ÜberNIC.

a) Create directory to store LMS-related supporting directories and files:

```
mkdir <your_dir>
```

b) Enter the <your_dir> directory:

```
cd <your_dir>
```

c) Copy and unpack the ÜberNIC Driver and Supporting SW tar

```
tar -xvzf <yyyymmdd>_UberNIC_v2.0.tar.gz
```

d) Verification of Tree Contents and Structure

```
tree ubernic
```

Unpacking the `<yyyymmdd>_UberNIC_v2.0.tar.gz` should have resulted in the creation of the subdirectory “ubernic” with the following tree structure:

```
├── driver
│ ├── dkms.conf
│ ├── Makefile
│ ├── README.md
│ ├── src
│ │ ├── BUILD_INFO.h
│ │ ├── lms_chr.c
│ │ ├── lms_chr.h
│ │ ├── lms_dma.c
│ │ ├── lms_dma.h
│ │ ├── lms_dump_dma.c
│ │ ├── lms_dump_dma.h
│ │ ├── lms_eth.c
│ │ ├── lms_eth.h
│ │ ├── lms_irq.c
│ │ ├── lms_irq.h
│ │ ├── lms_ns_v2.c
│ │ ├── lms_ns_v2.h
│ │ ├── lms_pcie.c
│ │ ├── lms_pcie.h
│ │ ├── lms_print.h
│ │ ├── lms_ptp_aux.c
│ │ ├── lms_ptp_aux.h
│ │ ├── lms_ptp.c
│ │ ├── lms_ptp.h
│ │ ├── lms_txts.c
│ │ ├── lms_txts.h
│ │ ├── lms_wr.c
│ │ ├── lms_wr.h
│ │ ├── log.profile
│ │ ├── Makefile
│ │ ├── test
│ │ │ ├── Makefile
│ │ │ └── ubernic_ctrl_test.c
│ │ ├── ubernic_ctrl.c
│ │ └── ubernic.h
│ ├── ubernic_drv_ptp_shared.h
│ ├── ubernic_drv_usr_shared.h
│ ├── ubernic.spec
│ └── utils
│ └── gen_build_info.sh
├── examples
│ ├── broadcast_send_sock
```

```
├── broadcast_send_sock.c
├── eth_rx_timestamping_sock
├── eth_rx_timestamping_sock.c
├── eth_tx_timestamping_sock
├── eth_tx_timestamping_sock.c
├── libunic.so
├── Makefile
├── multicast_recv_sock
├── multicast_recv_sock.c
├── multicast_send_sock
├── multicast_send_sock.c
├── readme.txt
├── tcp2tcp_loop_sock
├── tcp2tcp_loop_sock.c
├── tcp_accept_ex
├── tcp_accept_ex.c
├── tcp_connect_ex
├── tcp_connect_ex.c
├── tcp_highbandwidth_send_ex
├── tcp_highbandwidth_send_ex.c
├── tcp_listen_ex
├── tcp_listen_ex.c
├── tcp_loop_ex
├── tcp_loop_ex.c
├── tcp_loop_txts_ex
├── tcp_loop_txts_ex.c
├── uberphc.c
├── uberphc_ctrl_time
├── uberphc_ctrl_time.c
├── uberphc.h
├── uberphc_stat_timex
├── uberphc_stat_timex.c
├── ÜberSock.h
├── udp2tcp_loop_ex
├── udp2tcp_loop_ex.c
├── udp2tcp_loop_sock
├── udp2tcp_loop_sock.c
├── udp2udp_loop_sock
├── udp2udp_loop_sock.c
├── udp_loop_ex
├── udp_loop_ex.c
├── udp_loop_txts_ex
├── udp_loop_txts_ex.c
```

```
│ ├── udp_recv_ex
│ ├── udp_recv_ex.c
│ ├── udp_recv_multicast_ex
│ ├── udp_recv_multicast_ex.c
│ ├── udp_recv_sock
│ ├── udp_recv_sock.c
│ ├── udp_send_ex
│ ├── udp_send_ex.c
│ ├── udp_send_sock
│ ├── udp_send_sock.c
│ ├── udp_txts_ex
│ └── udp_txts_ex.c
├── install_pkg.sh
├── install.sh
├── libunic.so
├── output.pcapng
├── uber_bmc_load_image.sh
├── uber_bmc_mng
├── uber_bmc_monitor
├── uber_bmc_monitor.conf
├── uber_bmc_monitor.service
├── uber_bmc_monitor.sh
├── uber_bmc_reload
├── uberdate
├── uber_dump
│ ├── uber_dump
│ ├── uber_dump_cfg
│ └── uber_dump_splitng
├── ÜberLoad
│ ├── libuberload.so
│ ├── libuberload_unsafe.so
│ └── ÜberLoad
├── ubernic_daemon
├── ubernic_ethernet.service
├── ubernic_ethernet_txtstamp.service
├── uberphc_ctrl
├── uber_ptp_mng
├── ÜberSock.h
├── uberwr
│ ├── uberwr.bin
│ ├── uberwr.cfg
│ ├── uberwr_diags
│ ├── uberwr_mng
```

```
├── uberwr_uart
│   └── wr_140M625_ref3
├── wr_140M625_ref3_clock.json
├── wr_140M625_ref3_definitions.py
├── wr_140M625_ref3_registers.nvm.py
└── wr_140M625_ref3.variants.json
└── uninstall.sh

10 directories, 127 files
```

3) Enter the ÜberNIC Driver and Supporting SW directory:

```
cd ubernic
```

4) SU to Root to run installation script:

```
sudo su root
```

5) Install the Driver and API:

```
sudo ./install.sh
```

6) Reboot system:

```
sudo reboot
```

7) ÜberNIC driver installation verification:

```
lsmod | grep ubernic
```

Expected Output:

8) If Using PCIe

Verify the ÜberNIC driver and module are correctly loaded using `lspci`. You should see “ubernic” listed in the terminal’s output, as bolded in the following example output:

```
lspci -vvv -d 4c4d:*
```

Output example:

be:00.0 Ethernet controller: Liquid-Markets GmbH Device 9997 (rev 01)
Subsystem: Liquid-Markets GmbH Device 9997
Physical Slot: 6

Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr+ Stepping- SERR+ FastB2B- DisINTx-
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
Latency: 0, Cache Line Size: 32 bytes
NUMA node: 0
Region 0: Memory at f8000000 (32-bit, non-prefetchable) [size=32M]
Region 2: Memory at f6000000 (32-bit, non-prefetchable) [size=32M]
Capabilities: <access denied>
Kernel driver in use: ubernic
Kernel modules: ubernic
Kernel modules: cxl_pci

9) Check ÜberNIC Interfaces Creation

The installer will set up a new systemD ÜberNIC daemon to manage the hardware stack for ÜberNIC with PCIe. To support DHCP and ensure that ÜberNIC can behave like any other Ethernet device, the kernel module will create 8 unique ÜberNIC interfaces (since there are 8 fiber pairs that can be used with ÜberNIC) automatically. The created interfaces behave like any other Ethernet device; you can use any Linux network configuration tools (ip, ethtool, ifconfig, nmcli, nmtui) to manage the interfaces.

ÜberNIC implements an intuitive naming convention allowing for rapid identification and interaction with specific ÜberNIC boards and interfaces

The format for the network interface created is as follows:

```
unic_<PCIe_Bus_Adress>_<Interface_#>
```

<PCIe_Bus_Adress> refers to the first two characters of the PCIe Bus address, with any ":" or "." removed, and *<Interface_#>* refers to the interface number, numbered from 0 through 4 or 7 (depending on ÜberNIC HW logic version).

To see the ÜberNIC network interfaces, you can utilize the <ip a> command:

```
ip a
```

Output example:

2: **unic_c7_0**: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
link/ether 3c:0d:2c:08:53:80 brd ff:ff:ff:ff:ff:ff
3: **unic_c7_1**: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
link/ether 3c:0d:2c:08:53:81 brd ff:ff:ff:ff:ff:ff
4: **unic_c7_2**: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
link/ether 3c:0d:2c:08:53:82 brd ff:ff:ff:ff:ff:ff
5: **unic_c7_3**: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000

link/ether 3c:0d:2c:08:53:83 brd ff:ff:ff:ff:ff:ff

D. Configuration of Network Interface(s)

As of the date of this document **ÜberNIC does not support DHCP when you are using ÜberNIC with CXL**. Therefore, ÜberNIC's network ports must be statically configured. To accomplish this, standard Linux tools are used to create an OSI Layer 2 IP Tunnel between the Host Operating System and the ÜberNIC network ports. The tunnel(s) are intended to support non-latency-sensitive network interaction, including the use of, if desired, standard, sockets-compliant, Linux kernel-bound network stack(s).

Once configured, the tunnel(s) are maintained across system warm or cold restarts and reboots.

Once configured, the tunnel(s) support the overwhelming majority of Linux network commands including ping, nslookup, etc.

Once configured, the network port (name, IP address, IP routing rules, etc) are available for use with the ÜberSock ultra-low-latency C API.

ÜberNIC supports a maximum of 8 (ÜberNIC Ultra and ÜberNIC Ultra+) or 16 (ÜberNIC Ultra Wide and ÜberNIC Ultra+ Wide) 1Gbps, 10Gbps, or 25Gbps network ports. The specific ports are referenced beginning with zero (0). The specific number of supported network parts is dependent upon the type of transceiver and breakout cable used. For example, a 40Gbps transceiver (ex: FS.com 40G-SR4 QSFP+ P/N QSFP-SR4-40G) will support four (4) network ports, numbered 0, 1, 2, and 3, while a 200Gbps (operating in NRZ emulation) transceiver (ex: Molex 200G-SR8 QSFP-DD P/N 1837150046) will support eight (8) network ports, numbered 0, 1, 2, 3, 4, 5, 6, and 7.

1) Creation of Interface Configuration

Syntax:

```
nmcli connection add [ARGUMENTS...]
```

Arguments:

```
type [connection type, Example: tun (tunnel)]
ifname [interface name, Example: unic0]
con-name [connection name, Example: unic0]
mode [tunnel mode, Example: tap]
tun.owner [user uid, Exmple: 1000]
ip4 [assigned ip4 address, Example: 10.50.0.3/24]
autoconnect [autoconnect on system start, Example: yes]
```

Example:

```
sudo nmcli connection add type ethernet ifname unic_8a_0 con-name
unic_8a_0 ipv4.method manual ipv4.addresses 10.50.0.3/24 ipv4.gateway
10.50.0.3 autoconnect yes
```

2) Creation of IP Tunnel(s) Using NMCLI (Required for CXL)

Syntax:

```
nmcli connection add [ARGUMENTS...]
```

Arguments:

```
type [connection type, Example: tun (tunnel)]
ifname [interface name, Example: unic0]
con-name [connection name, Example: unic0]
mode [tunnel mode, Example: tap]
tun.owner [user uid, Exmple: 1000]
ip4 [assigned ip4 address, Example: 10.50.0.3/24]
autoconnect [autoconnect on system start, Example: yes]
```

Example:

```
sudo nmcli connection add type tun ifname unic0 con-name unic0 \ mode
tap tun.owner 1000 ip4 10.50.0.3/24 autoconnect yes
```

3) Add Default Gateway

Note: Currently, there is a limitation on the number of routes ÜberNIC can be routed to as we are still in the early beta release. ÜberNIC is limited to only 1 default gateaway per fiber. We understand that this limitation may cause inconvenience and we apologize for any disruption to your workflow or reduced user experience you may have while using ÜberNIC. LMS's development team is actively working on a solution and will be released soon.

Syntax:

```
ip route add default {IP-ADDRESS}/{NETMASK} dev {INTERFACE-NAME}
ip route add default {NETWORK/MASK} via {GATEWAYIP}
```

Example:

```
sudo ip route add default 10.50.0.1/24 dev unic0
```

or

```
sudo ip route add 10.50.0.1/24 via 10.50.0.3
```

4) Configure DNS Nameserver

DNS nameservers may need to be configured for proper address resolution. There are multiple ways to configure DNS nameservers, one is shown below.

a) Open file for editing:

```
sudo vi /etc/resolv.conf
```

b) Add desired nameservers:

```
nameserver 8.8.8.8
```

c) Save:

```
<Esc> <:wq> <Enter>
```

d) Alternatively, using nmcli:

```
sudo nmcli connection modify unic_8a_0 ipv4.dns "10.50.0.1"
```

5) Edit Interface – Change Gateway

Syntax:

```
sudo nmcli connection modify {INTERFACE-NAME} ipv4.gateway "{IP-ADDRESS}"
```

Example:

```
sudo nmcli connection modify unic0 ipv4.gateway "192.168.188.1"
```

6) Edit Interface – Change IP

Syntax:

```
ifconfig {INTERFACE-NAME} {IP-ADDRESS}/{SUBNET MASK}
```

Example:

```
sudo ifconfig unic3 192.168.188.216/24
```

7) Delete Interface

Syntax:

```
nmcli connection delete id {INTERFACE-NAME}
```

Example:

```
sudo nmcli connection delete id unic0
```

E. ÜberL2

1) Components

ÜberL2 is provided in the form of:

- User space daemon (ubernic_daemon)

2) Overall functionality

For non-accelerated, i.e. non-latency-sensitive messaging, ÜberNIC falls back to a user-space daemon which bypasses the kernel and connects a Linux TAP device to ÜberNIC at Layer 2, this is called ÜberL2.

ÜberL2 currently operates in polling mode, polling once every 1 millisecond for messages in a particular receive buffer.

Note: ÜberL2 RX has shifted from a poll-based mechanism to an interrupt-based mechanism with the release of ÜberStack v2.0 on April 18, 2025.

F. Receive Buffer

ÜberNIC implements a single receive buffer per hardware session thereby guaranteeing that all received messages are inserted into the receive buffer in the order received from the network. Each ÜberNIC receive buffer is, by default, 4MB and each is end-user reconfigurable to a maximum of 128MB. Details:

Configuring TCP or UDP Receive Buffer Size

- Configurable Values

  The default receive buffer size is 4 MB.

  The minimum receive buffer size is 256 KB.

  The maximum receive buffer size is 128 MB.

  The specified receive buffer size should be multiple 4 KB DMA buffer pages, in all other cases the operation will fail and EINVAL errno will be returned
- ÜberLoad

  `getsockopt()` and `setsockopt()` can be used to read and write the `SO_RCVBUF` option in order to change TCP/UDP socket receive buffer size.
- ÜberSock

  `ÜberSock_get_opt()` and `ÜberSock_set_opt()` can be used to read and write the `UBER_OPT_RX_BUF_SIZE` option in order to change TCP or UDP socket receive buffer size.
  Please note that those configuration calls should be executed after `ÜberSock_acquire_handle()` and before `ÜberSock_udp_configure()` / `ÜberSock_udp_configure_multicast()` / `ÜberSock_tcp_connect()`.

G. TimeToLive (TTL)

ÜberNIC implements Time to live (TTL) as a hop limit limits the lifespan of data in the network. Once the defined hop limit is reached is discarded. TTL prevents a data packet from circulating infinitely.

Configuring IP TTL and Multicast TTL Options

- Configurable Values

  Default unicast IP TTL: 64

  Default multicast IP TTL: 1
- ÜberLoad

  `getsockopt()` and `setsockopt()` can be used to read and write the `IP_TTL` option in order to access the TCP or UDP transmitted TTL value.

  `getsockopt()` and `setsockopt()` can be used to read and write the `IP_MULTICAST_TTL` option in order to access UDP multicast TTL transmitted value.
- ÜberSock

  `ÜberSock_get_opt()` and `ÜberSock_set_opt()` can be used to read and write the `UBER_OPT_IP_TTL` option in order to access TCP or UDP transmitted TTL value.
  `getsockopt()` and `getsockopt()` can be used to read and write the `UBER_OPT_IP_MULTICAST_TTL` option in order to access UDP multicast transmitted TTL value.

H. ÜberSock

1) Components

ÜberSock is provided in the form of:

- Header file (ÜberSock.h)
- C library (libunic.so)

2) Overall Functionality

ÜberSock is a low-level access library for ÜberNIC FPGA via kernel bypass techniques, such that data is transmitted to, or received from, the network without any involvement of the Linux Kernel. ÜberSock allows packets to be transmitted and received using TCP/IP and UDP/IP protocols with minimum possible latency.

The ÜberSock API is a pure C function interface that is defined in the header file, ÜberSock.h. There are many ways to use the API, but one straightforward way is by including the header and linking it against the binary provided libunic.so library.

3) Prerequisite

Before using ÜberSock, you must ensure that the network port (name, IP address, IP routing rules, etc) for the desired ÜberNIC physical interface is configured.

4) ÜberSock and UDP

This section gives information about using UDP with ÜberSock. Do note that the ÜberSock library doesn't handle multicast IGMP registration, it should be done by the standard Linux socket programming or command line tools (eg: `socat, IP igmp`) and the UDP session must be configured accordingly.

Each UDP sessions can be configured as:

a) Unicast / Broadcast

By default, the receiving broadcast feature is enabled. For each UDP session, the receiving broadcast feature can be disabled separately, by using the `ÜberSock_set_opt` API to set the `UBER_OPT_UDP_RX_BROADCAST` to 0.

An incoming frame will be accepted by the UDP RX if the destination IP/port matches the local IP/port settings. Additionally, a broadcast UDP frame with a matching port is accepted as well, if the sender's destination port in the UDP frame matches the local port. Each local port is limited to only 1 UDP session.

b) Multicast

An additional socket parameter `multicast_ip` is defined in the ÜberSock API to enable the multicast functionality.

Since ÜberSock library doesn't handle multicast registration, you can join any multicast address (e.g. 230.0.0.1/24) using standard Linux socket programming or Linux networking tools.

For example, using `ip` command:

```
ip addr add 230.0.0.1/24 dev unic0 autojoin
```

You can verify that you've joined the multicast group successfully using `ip` too:

```
ip -f inet maddr show dev unic0
```

Example output:
5: unic0
inet 230.0.0.1

An incoming frame will be accepted by the UDP RX if the destination IP/port matches the multicast IP and local port settings. When the multicast functionality is enabled, any UDP packets with a local and broadcast IP are rejected. With the same local port, several UDP multicast sessions are possible.

A UDP packet can be sent to a multicast group by using the corresponding multicast address as the destination IP. The IP multicast address must be specified in the range from 224.0.0.0 to 239.255.255.255 as the destination address. In this case, the contained MAC address will be automatically converted to the multicast format so no specific MAC configuration is needed by the user.

c) Netmasked Multicast

Supports receiving multicast frames within an IP range instead of a single IP/Port combination (with no no port filtering), using a netmask to have more flexibility for receive flow filtering.

This mechanism supports receiving a far larger number of multicast channels within a single "hardware session" than is possible with traditional IP/Port-based mechanisms however additional port filtering is necessary in the user application.

The new API function is called by using:

int ÜberSock_udp_configure_rx_multicast_mask(ÜberSock_t *ÜberSock, const char *interface_name, uint32_t multicast_ip, uint32_t multicast_mask_ip);

For example, with multicast_ip = 233.0.100.128 and mask_ip=255.255.255.240 ,you will receive all multicast traffic from IPs 233.0.100.128 to 233.0.100.143

Please note : IGMP function is not internally supported for this function, therefore standard Linux registration must be done for the specific interface to join all groups you need.
(for example: ip addr add 230.0.0.1/24 dev unic0 autojoin).

Restriction: No IP address overlapping is supported / checked, each multicast/mask set must be carefully set up to not overlap/include any ip address mapped to another session.

This is only supported with ÜberSock, not ÜberLoad.

In parallel,the API for udp receive provides size, timestamp, source ip/port and destination ip/port. This will help for additional filtering in user applications.
int ÜberSock_udp_zc_receivefrom_full(ÜberSock_t *ÜberSock, void** data, uint64_t* ts,

uint32_t *sip, uint16_t *sport, uint32_t *dip, uint16_t *dport);

See the design example udp_recv_multicast_mask_ex (in the /ubernic/examples directory) for an operating example and source code.

5) ÜberSock and TX Timestamp

This section provides information about using TX timestamps with ÜberSock.

By default, TX timestamping feature is disabled. To enable this feature, you can use the `ÜberSock_enable_tx_timestamping` function. This function must be called **before** sending any data to guarantee a synchronized TX timestamp output, where all the timestamps are returned in the order the data were originally sent.

Once TX timestamping is enabled, it cannot be disabled. Also, timestamps must be requested periodically using `ÜberSock_get_tx_timestamp`. Infrequent calls to this function can lead to desynchronization of the timestamps. In this situation, tv_sec will be set to UINT32_MAX to show an error.

I. ÜberLoad

1) Components

ÜberLoad is provided in the form of:

- Script (ÜberLoad) – facilitates and enables the socket emulation
- C library (libuberload.so, libuberload_unsafe.so)

2) Overall Functionality

The ÜberLoad libraries (libuberload.so, libuberload_unsafe.so) directly connect user-space application to a fully offloaded network stack implementation running on the ÜberNIC FPGA via kernel bypass techniques.

Applications using ÜberLoad require no modification, since calls to the Linux socket APIs are intercepted by the library. However, it is worth noting that ÜberLoad is NOT a Linux network stack and therefore, not all network sockets functionality is supported.

ÜberLoad uses LD_PRELOAD to preload a shared library that emulates socket function calls. ÜberLoad does not work if the application is statically linked (to the C library, e.g. glibc). A statically linked application directly does the system calls which cannot be intercepted by ÜberLoad. Thus, an application will only work with ÜberLoad if it is dynamically linked (to the C library).

3) Prerequisite

Before using ÜberLoad, you must ensure that the network port (name, IP address, IP routing rules, etc) for ÜberNIC is properly configured. Also, it is extremely important to ensure that the ÜberNIC daemon is up and running properly. Refer to the previous section, Configuration of Network Interface(s) for more details.

4) Using ÜberLoad

Once the software logic is installed, ÜberLoad can be used by simply prefixing the `ÜberLoad` command with any applications using Linux kernel sockets in the command line.

Syntax:

```
ÜberLoad [ÜberLoad options] command [command options]
```

Example Command

```
ÜberLoad iperf3 -c 192.51.100.0
```

5) ÜberLoad Options

If no options or arguments are supplied when running `ÜberLoad`, it will print a version reference (example: ÜberLoad githash: B6F6CB1D) and display the information about the utility, like the `--help` option.

ÜberLoad has 4 different categories of options, noted in the following:

a) Background Thread Affinity

Option	Explanation
`--background-affinity <cpu core>`	Prevents background thread from running on the same CPU core as primary processing, thereby improving performance.

b) Thread Safety

By default, thread-safety is set to off, thread-safety can be set to on but will have a significant negative impact to latency performance.

Option	Explanation
`-t-s, --thread-safe`	Use ÜberLoad with thread safety enabled. Do note that when using ÜberLoad with thread safety enabled, latency might be extremely higher compared to when thread safety is disabled.
`-n-t-s, --non-thread-safe`	Use ÜberLoad with thread safety disabled.

c) Interface

Option	Explanation
`-i <interface-ip>`	Define ÜberLoad to use this ÜberNIC interface when binding to ANY (0.0.0.0) Example: -i 10.10.1.6
`--bind-interface <interface-ip>`	Define ÜberLoad to use this ÜberNIC interface

	when binding to ANY (0.0.0.0) Example: --bind-interface 10.0.50.0

d) Tracing

Option	Explanation
`-n, --notrace`	If enabled, use this option to disable tracing.
`-t <tracedir>,` `--tracedir <tracedir>`	Set destination directory for trace output log file. Example: -t /tmp

e) Help

Option	Explanation
`-h, --help`	Display the basic syntax and functionality of ÜberLoad and the version reference.

6) ÜberLoad Environment Variables

The following sections explain the available environment variables which can be modified to reflect the behavior of ÜberLoad. By default, ÜberLoad does not trace or log any details for highest performance.

a) NOTRACE

To enable tracing/logging, set NOTRACE to 0. By default, this variable is set to 1.

There are 3 ways to modify the NOTRACE variable, you can choose either one:

- Prefixing the NOTRACE variable before the `ÜberLoad` command in the command line terminal. This would result in a **temporary** setting, which means that it will only be available in the current session. If you open a new shell or if you log out, the setting will be lost:

Syntax:

```
NOTRACE=0 ÜberLoad [ÜberLoad options] command [command options]
```

- Exporting the NOTRACE variable directly to the terminal before running the `ÜberLoad` command. This would result in a **permanent/persistent** setting:

```
export NOTRACE=0
```

To remove the setting for NOTRACE, use the `unset` command:

```
unset NOTRACE
```

b) ÜBERLOAD_LOGFILE

To direct the socket call logs to a file instead of the terminal when using ÜberLoad, enable the ÜBERLOAD_LOGFILE. This logfile will be replaced for each run to avoid the accumulation of old log information. Hence, please be careful not to run two programs using the same logfile!

There are 3 ways to enable ÜBERLOAD_LOGFILE, you can choose either one:

- Prefixing the ÜBERLOAD_LOGFILE variable before the `ÜberLoad` command in the command line terminal. This will result in a **temporary** setting, which means that it will only be available in the current session. If you open a new shell or if you log out, the setting will be lost:

Syntax:
```
ÜBERLOAD_LOGFILE=./filename ÜberLoad <options> command <options>
```

Example Command:
```
ÜBERLOAD_LOGFILE=./ÜberLoad.log ÜberLoad -t-s iperf3 -c 192.168.1.1
```

- Exporting the ÜBERLOAD_LOGFILE variable directly on the terminal before running the `ÜberLoad` command. This will result in a **permanent** setting:

Syntax:
```
export ÜBERLOAD_LOGFILE=./filename ÜberLoad <options> command
<options>
```

Example Command:
```
export ÜBERLOAD_LOGFILE=”/tmp/ÜberLoad.log” ÜberLoad -t-s iperf3 -c
192.168.1.1
```

- Modify the ÜberLoad script. This would result in a **permanent** setting:

1. Open the ÜberLoad script:

```
vi ÜberLoad
```

2. Uncomment the following line in the script

#export ÜBERLOAD_LOGFILE="/tmp/ÜberLoad.log"

3. Save

```
<Esc> <:wq> <Enter>
```

c) VERBOSE

To enable a verbose output for logging/tracing, set VERBOSE to 1.

There are 3 ways to set VERBOSE:

- Prefixing the VERBOSE variable before the `ÜberLoad` command in the command line terminal. This would result in a **temporary** setting, which means that it will only be available in the current session. If you open a new shell or if you log out, the setting will be lost:

Syntax:

```
VERBOSE=1 ÜberLoad <options> command <options>
```

- Exporting the VERBOSE variable directly to the terminal before running the `ÜberLoad` command. This would result in a **permanent** setting:

```
export VERBOSE=1
```

To remove the setting for VERBOSE, use the `unset` command:

```
unset VERBOSE
```

- Modify the ÜberLoad script. This would result in a **permanent** setting:

1. Open the ÜberLoad script:

```
vi ÜberLoad
```

2. Uncomment the following line in the script

#export VERBOSE=1

3. Save

```
<Esc> <:wq> <Enter>
```

7) Best Practices

- Wherever possible, please avoid mixing normal Linux sockets and ÜberNIC sockets in calls like `poll` or `select` as this will increase the jitter/latency for ÜberNIC sockets and waste more CPU cycles.
  - This also applies for common file descriptors for non-sockets (file access, pipes, etc.)
- Avoid using `select` in general, even when you're not using ÜberNIC. Use `poll` instead.
- For the lowest latency reception, don't use `poll` or `select` and then `recv`. Please use a blocking `recv` only.
- For high rate send or receive operations, use only one thread to call functions on a single socket at a time. Using multiple threads may create lock contention which will lead to a drop in total performance.

- When using normal thread safe UDP with ÜberLoad, it is highly suggested to use two sockets, if possible. For example, one socket is used for sending and the other socket is used for receiving UDP.
- If you are using a C application with ÜberLoad, the application must be dynamically linked to ensure ÜberLoad can intercept the system calls on a library level.

J. ÜberNIC Design Example Applications

Example applications are located within the ubernic/examples directory. All applications ending with "_ex" are using the low level ÜberSock API, while applications ending with "_sock" are using the Linux standard sockets API and may be further accelerated with ÜberLoad.

1) ÜberLoad Design Example Applications

These applications can be run on their own with any sockets-compliant mechanism or accelerated with ÜberLoad.

a) broadcast_send_sock

This is an example application which enables UDP broadcast on a socket (which is necessary for Linux kernel socket, not ÜberLoad) and send a simple message upon successful connection.

By default, the destination port is set to '2222'. You can modify this value with the desired port number by directly modifying the provided C source file and recompile the program by using "make" command.

The application expects a destination IP address as an argument.

Usage: `./broadcast_send_sock` *`DESTINATION-IP`*

b) multicast_recv_sock

This is an example application that joins a multicast group membership and binds the UDP socket to a specified multicast IP address and interface IP. There are two ways to join a multicast group membership, which is written in the example application.

By default, the multicast IP port is set to '2222'. You can modify this value with the desired port number by directly modifying the provided C source file and recompile the program by using "make" command.

The application expects a multicast IP destination and local interface IP as arguments.

Usage: `./multicast_recv_sock` *`MULTICAST-IP INTERFACE-IP`*

c) multicast_send_sock

This is an example application which attempts to send a multicast datagram to the multicast cast destination.

By default, the multicast IP port is set to '2222'. You can modify this value with the desired port number by directly modifying the provided C source file and recompile the program by using "make" command.

The application expects a multicast IP destination and local interface IP as arguments. The interface IP provided will be used as the source IP to send the multicast frames.

Usage: `./multicast_send_sock` *`MULTICAST-IP INTERFACE-IP`*

d) tcp2tcp_loop_sock

This is an example application that receives data via TCP and forwards the payload to a TCP connection.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as ‘3333’ by default and recompile the program by using “make” command.

The application expects remote IP address and local IP address as arguments.

Usage: `./tcp2tcploop_sock` *`REMOTE-IP LOCAL-IP`*

e) udp2tcp_loop_sock

This is an example application that receives data via UDP and forwards the payload to a TCP connection.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as ‘3333’ by default and recompile the program by using “make” command.

The application expects remote IP address and local IP address as arguments.

Usage: `./udp2tcploop_sock` *`REMOTE-IP LOCAL-IP`*

f) udp2udp_loop_sock

Details Pending

g) udp_recv_sock

Details Pending

h) udp_send_sock

Details Pending

i) eth_rx_timestamping_sock

Details Pending

j) eth_tx_timestamping_sock

Details Pending

2) ÜberSock Design Example Applications

These applications can be run, (ex: udp_loop_ex), and the corresponding C header file (ÜberSock.h) reviewed, to assist in the modification of your application(s) to implement ÜberSock.

These API examples can also be run on the Linux TCP/IP or UDP/IP stack using the emulation library, libunic_emu.so. The emulation library is provided to support testing without an ÜberNIC hardware and may have limited performance compared to the libunic.so library.

There are two ways to use the libunic_emu.so library:

1. Move the emulator library, libunic_emu.so to a separate directory and rename it to libunic.so. Next, update the LD_LIBRARY_PATH environment variable to use this newly renamed emulator library path.
2. Preload the emulator library, libunic_emu.so before executing the application using LD_PRELOAD. This will load both libraries, libunic.so and libunic_emu.so, but will give the emulation library a higher priority.

For example, running tcp_listen_ex on eno1:

```
LD_PRELOAD=libunic_emu.so ./tcp_listen_ex eno1
```

a) TCP

1. tcp_connect_ex

This is an example application which actively attempts to establish a TCP connection. Upon successful connection establishment, the application will send one simple message and disconnect with the TCP socket.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as '2000' by default.

The application expects an interface name and the remote IP address as arguments.

Usage: `./tcp_connect_ex` *`INTERFACE-NAME REMOTE-IP`*

2. tcp_highbandwidth_send_ex

This is an example application which sends high bandwidth data over a TCP connection after a successful TCP connection establishment. The application will keep on sending messages infinitely, so you can stop the application by pressing ctrl+c on the keyboard.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as '2000' by default.

The application expects an interface name and the remote IP address as arguments.

Usage: `./tcp_highbandwidth_send_ex` *`INTERFACE-NAME REMOTE-IP`*

3. tcp_listen_ex

This is an example application which actively announces the willingness to accept TCP connection requests. Upon successful connection establishment, the application will accept and write the messages received until there's no more data. The application will stop running once the connection initiator disconnects.

You can specify the desired local port number by directly modifying the provided C source file since it is set as '2000' by default.

The application expects an interface name as an argument.

Usage: `./tcp_listen_ex` *`INTERFACE-NAME REMOTE-IP`*

4. tcp_loop_ex

This is an example application which loops back the data received on the same socket/session. The application will keep on looping the sent and received message in an endless loop, so users can stop the application by pressing ctrl+c on the keyboard.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as '2200' and '9999' by default, respectively.

The application expects an interface name and the remote IP address as arguments.

Usage: `./tcp_loop_ex` *`INTERFACE-NAME REMOTE-IP`*

5. tcp_loop_txts_ex

In this example, the application receives UDP frames on a specific port and resends them back to the specified IP address and port as a TCP frame. Statistics is collected on the time delta (in nanoseconds) between receiving the UDP packet and resending it as a TCP packet.

In this example, TX timestamping is enabled and configured to keep track of 16000 (configurable in the code) simultaneous timestamps. It is to be noted that TX timestamping is enabled only on the TCP connection as all transmits are through this connection. For each incoming UDP frame, its RX timestamp is recorded. When the frame is re-sent via TCP, its TX timestamp tracking ID is stored in an array for future retrieval. After 10000 packets (configurable in the code), the tracking IDs are used to retrieve the TX timestamp and statistics (mean, min, median, 99-percentile, max, standard deviation) is collected on the delta between UDP-RX and TCP-TX timestamps in nano seconds.

Usage: `./tcp_loop_txts_ex.c INTERFACE-NAME REMOTE-IP`

6. tcp_accept_ex

Details Pending

b) UDP

1. udp_recv_ex

This is an example application which actively listens for incoming connections and is ready to receive data. Also, this example shows how to disable the UDP broadcast option using `ÜberSock_set_opt()`.

You can specify the desired local port number by directly modifying the provided C source file since it is set as ‘2000’ by default.

The application expects an interface name and the remote IP address as arguments.

Usage: `./udp_recv_ex` *`INTERFACE-NAME REMOTE-IP`*

2. udp_send_ex

This is an example application which attempts to establish a UDP connection with the specified IP and port and sends messages.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as ‘2000’ for both by default.

The application expects an interface name and the remote IP address as arguments.

Usage: `./udp_send_ex` *`INTERFACE-NAME REMOTE-IP`*

3. udp_loop_ex

This is an example application that receives UDP frames on specified port and resends them back to the specified IP address and port.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as ‘2200’ and ‘9999’ by default, respectively.

The application expects an interface name and the remote IP address as arguments.

Usage: `./udp_loop_ex` *`INTERFACE-NAME REMOTE-IP`*

4. udp_recv_multicast_ex

This is an example application which actively listens for incoming multicast connection and is ready to receive multicast data.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as '2000' for both by default.

The application expects an interface name, multicast IP address and local port (optional) as arguments.

Usage: `./udp_recv_multicast_ex <interface_name> <multicast_ip> <local_port>`

5. udp_recv_multicast_mask_ex

This example receives multicast flow configured by the multicast base ip and the extension mask ip. For example, with multicast_ip = 233.0.100.128 and mask_ip=255.255.255.240 ,you will receive all multicast traffic from ips 233.0.100.128 to 233.0.100.143

In the example , the received frame with all information (size,source ip/port,destination ip/port) is printed out (This is not low-latency but only a visual example of multicast receiving)

Usage: ./udp_recv_multicast_ex <interface_name> <multicast_ip> <mask_ip>

6. udp_loop_txts_ex

This is an example application that receives UDP frames on a specified port, resends them back to the specified IP address and port and measures statistics on the time delta (in nanoseconds) between reception and transmission timestamps.

In this example, TX timestamping is enabled and configured to keep track of 16000 (configurable in the code) simultaneous timestamps. For each incoming frame, its RX timestamp is recorded. When the frame is re-sent, its TX timestamp tracking ID is stored in an array for future retrieval. After 10000 packets (configurable in the code), the tracking IDs are used to retrieve the TX timestamp and statistics (mean, min, median, 99-percentile, max, standard deviation) is collected on the delta between RX and TX timestamps in nano seconds.

The application expects interface and remote IP address as arguments. The local and remote ports can be modified in the code and are set to 2000 as default.

Usage: `./udp_loop_txts_ex INTERFACE-NAME REMOTE-IP`

7. udp_txts_ex

This example shows how to enable TX timestamping and retrieve TX timestamps for data sent on a UDP socket. In this example, when a packet is sent, its TX timestamp is immediately retrieved and printed on the screen before sending the next packet. The user provides the interface name, remote IP address, number of timestamps to track simultaneously and the number of packets to transmit as arguments. The user can specify the desired local and remote port number by directly modifying the provided C source file since it is set as '2000' by default.

Usage: `./udp_txts_ex INTERFACE-NAME REMOTE-IP NUMBER-OF-TS-TO-TRACK NUMBER-OF-PACKETS-TO-SEND`

c) UDP-to-TCP

1. udp2tcp_loop_ex

This is an example application that loops the data received via UDP to a TCP connection. The other side is supposed to listen on the target IP or TCP port.

You can specify the desired local and remote port number by directly modifying the provided C source file since it is set as '2000' by default.

The application expects an interface name and the remote IP address as arguments.

Usage: `./udp2tcploop_ex` *`INTERFACE-NAME REMOTE-IP`*

3) ÜberSock API Usage Example

a) TCP (tcp_connect_ex and tcp_listen_ex)

Overview: Create a simple TCP client/server application where server send some message to the client when getting connected. In this example, the application is run together with the emulator library, libunic_emu.so and is using the same interface but different ports.

Instructions to Execute:

1. Get the name of the network interface and IP. You can use `ifconfig` command to get the details.

   Example:
   ```
   ifconfig
   ```

   eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
   inet 192.168.1.101 netmask 255.255.255.0 broadcast 192.168.1.255
   inet6 fe80::a00:27ff:fe4c:1052 prefixlen 64 scopeid 0x20<link>
   ether 08:00:27:4c:10:52 txqueuelen 1000 (Ethernet)
   RX packets 890 bytes 800890 (781.9 KiB)
   RX errors 0 dropped 0 overruns 0 frame 0
   TX packets 634 bytes 75834 (74.0 KiB)
   TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

   Get 2 ports – one for TCP connect and the other for TCP listen. To list open ports on your machine, use `ss`. Ports that are not listed in the output are closed, so you can use them.

   Example:
   ```
   ss -ant
   ```

   Example output:

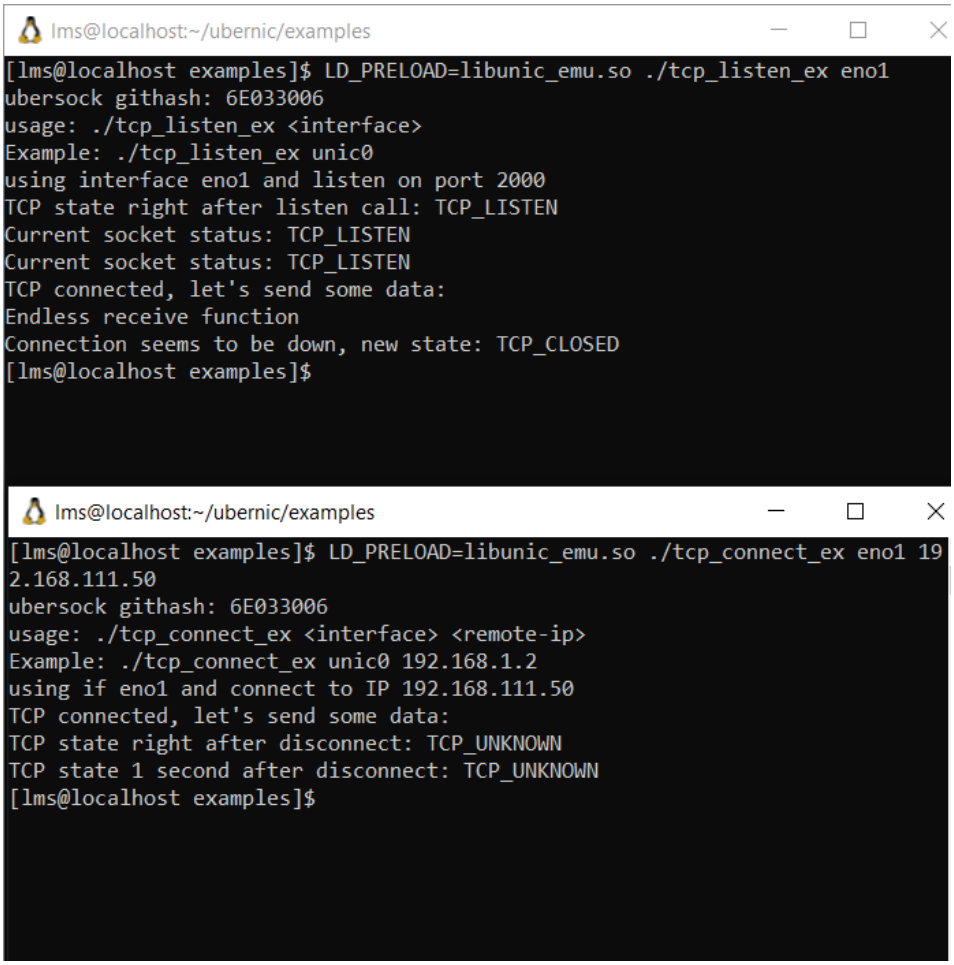
   StateRecv-Q Send-QLocal Address:PortPeer Address:PortProcess
   LISTEN0 4096 127.0.0.1:6310.0.0.0:*
   LISTEN0 1280.0.0.0:22 0.0.0.0:*
   ESTAB 0 80 192.168.111.50:22 192.168.111.26:50690
   LISTEN0 4096 [::1]:631[::]:*
   LISTEN0 128[::]:22[::]:*

   From the output, only port number **22** and **631** are used. Hence, we can use any other ports from 1 to 65535, aside from those two numbers.

2. Modify the C source files, tcp_connect_ex.c and tcp_listen_ex.c local port and remote port values. Make sure the local port value in tcp_listen_ex.c file matches the remote port value in tcp_connect_ex.c file.

tcp_listen_ex.c	tcp_connect_ex.c
const uint16_t local_port = 2000;	const uint16_t local_port = 2200; const uint16_t remote_port = 2000;

3. Open two terminals on your machine and compile the tcp_connect_ex and tcp_listen_ex programs in different terminals.
4. Run the tcp_listen_ex program first, followed by the tcp_connect_ex program.



4) ÜberSock TX Timestamping

a) General Workflow

The user enables the TX timestamping feature on ÜberSock before configuring the socket or sending any data. Once enabled, ÜberSock provides an option to track the timestamp of each transmitted packet. This is done by providing to the user a unique ID for the packet, which can be used at a later point of time to retrieve its TX timestamp. During the enable step, the user specifies the number of timestamps they wish to track simultaneously without running the risk of an overwrite. In other words, delaying the retrieval of a timestamp beyond this number could result in the timestamp being expired or lost. Once TX timestamping feature is enabled, it cannot be disabled.

b) Enabling

To enable, the user can use the ÜberSock_enable_tx_timestamping function and provide as input, the number of timestamps they wish to track simultaneously. As noted previously, this step must be completed before configuring the socket or sending any data.

When data is to be sent, the user has an option to track its TX timestamp by providing in the send function, a pointer to the memory location where ÜberSock has to store its tracking ID. If no pointer is provided, the timestamp for that packet will not be tracked. The user can then pass this tracking ID to ÜberSock_get_tx_timestamp function to retrieve its corresponding timestamp.

Depending on when this retrieval happens, the function can return the actual timestamp, or that the timestamp is not ready yet (ÜBERSOCK_TS_NOTREADY), or that the timestamp has expired (ÜBERSOCK_TS_EXPIRED) i.e. overwritten.

c) Adjusting

The RX and TX timestamps are adjusted by default to account for internal FPGA latency at the point of measurement (xcvr+emib+phy+pma) and external latency if any.

Put another way, timestamps are adjusted from an “internal” perspective in which some aspects of latency (transceiver, phy, pma, etc.) are excluded, to an “external” perspective in which all aspects of latency are included. The practical outcome of these adjustments is that timestamps reflect the physical point where data enters or exits the RX and TX transceivers.

The default value for internal adjustment is obtained from the FPGA and the default value for external adjustment is 0. The adjustments are done as follows:

Adjusted TX timestamp (ns) = TX timestamp captured on FPGA (ns) + (internal adjustment (ns) + external adjustment (ns))

Adjusted RX timestamp (ns) = RX timestamp captured on FPGA - (internal adjustment (ns) + external adjustment (ns))

The user can override these values by providing a configuration file for this interface in /etc/lms/ubernic with the interface name as the filename. To facilitate this, a template file will be created during installation at /etc/lms/ubernic/net_if_config_template. This file can be copied and renamed to the name of the interface and its contents modified to provide the internal and external TX, RX timestamp adjustment values. The configuration parameters pertaining to timestamp adjustments are as follows

ts_adjustment_enable=true/false : If false timestamps are not adjusted, else adjusted by default

ts_rx_internal_adjustment_ps = Value : RX internal FPGA latency in pico seconds

ts_rx_external_adjustment_ps = Value : RX external FPGA latency in pico seconds

ts_tx_internal_adjustment_ps = Value : TX internal FPGA latency in pico seconds

ts_tx_external_adjustment_ps = Value : TX external FPGA latency in pico seconds

Note: The default values for TS adjustment is obtained from the FPGA if they are commented out in the configuration file

12. ÜberNIC Network Port Management and User Space I/O

A. Overview

ÜberNIC implements four layers of host/network performance and scope, specifically:

Component	Method	Availability	Representation
ÜberL2	Kernel Bound	v2.0	Breadth-Focused Support for standard Configuration, Admin and Linux Network Stack I/O
ÜberSock	Kernel Bypass	v2.0	Performance Focused Handling of TCP and UDP messages
ÜberLoad	Kernel Bypass	v2.0	Plug and Play Sockets with LDPreload
ÜberFPGA FDK	Intra-FPGA	v2.6	Intra-FPGA Sandbox for Network Stack I/O with Host Interaction

B. Linux Network Management

In Unix-like operating systems, there are a few essential command-line and graphic interfaces that can be used to configure, manage, and query a network interface. These, and their capabilities vis-à-vis ÜberNIC, are detailed below.

1) NMCLI

`nmcli` (NetworkManager Command Line Interface) is a command-line utility which enables users and scripts to interact with NetworkManager and report network information. `nmcli` is used to create, display, edit, delete, activate, and deactivate network connections, as well as control and display network device status. Besides, `nmcli` can be used on systems without a GUI such as servers to control all aspects of NetworkManager.

NetworkManager is installed by default on Red Hat Enterprise Linux.

Some of the typical uses for `nmcli` includes:

a) **Scripts**: Utilize `nmcli` instead of managing network connections manually. `nmcli` supports a terse output format which is better suited for script processing. Note that NetworkManager can also execute scripts, called “dispatcher scripts”, in response to network events.

b) **Servers, headless machines, and terminals**: `nmcli` can be used to control NetworkManager without a GUI, including creating, editing, starting, and stopping network connections and viewing network status.

Refer to the separately provided Linux Network Configuration (NMCLI & NMTUI) User Guide for more information on `nmcli`.

2) NMTUI

`nmtui` (NetworkManager Text User Interface) is another tool to configure NetworkManager using a simple text-based menu system. This tool is used in a terminal window. Compared to `nmcli`, `nmtui` does not support all types of connection. It can only be used to manage wired and wireless devices only. Besides, `nmtui` cannot manage remote network connections.

Refer to the separately provided Linux Network Configuration (NMCLI & NMTUI) User Guide for more information on `nmtui`.

3) IFCONFIG (Deprecated)

`ifconfig` (interface configuration) is also a command line interface used for:

- Displaying current network configuration information
- Setting up an IP address, netmask, or broadcast address to a network interface
- Creating an alias for the network interface
- Setting up hardware (MAC) address
- Enabling or disabling network interfaces

Refer to the separately provided Linux Network Configuration (IFCONFIG) User Guide for more information on `ifconfig`.

4) IP

`ip` is a networking command line utility that was intended to replace the `ifconfig` command. Although the `ip` command has more capabilities compared to `ifconfig`, it still works in a similar manner.

The functions that `ip` command supports are:

- Display routing, network devices, interfaces, and tunnels information
- Configure network routing rules (e.g., default and static routing, IP routing, routing table)
- Manage ARP/neighbor cache table (e.g., adding an entry, remove an entry, replace contents)
- Assign and remove IP address, netmask, broadcast address, or link of a network interface
- Modify the status of network interface

Refer to the separately provided Linux Network Configuration (IP) User Guide for more information on `ip`.

5) ETHTOOL

`ethtool` is also a Linux command-line interface which is used to:

- Obtain details of network devices, drivers, and hardware settings
- Modify network devices settings such as speed, port, auto-negotiation

Common Options:

a) Display current HW logic and SW logic versions

```
Command Syntax
sudo ethtool -i or –driver <interface_name>

Example
$ sudo ethtool -i unic_bc_0
driver: LMS Ubernic
version: deadbeef (v2.3.0)
firmware-version: 6127f14 (v2.5.0)
expansion-rom-version:
bus-info: 0000:bc:00.0
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
```

b) Display Interface Statistics

```
Command Syntax
sudo ethtool -S <interface_name>

Example
$ sudo ethtool -S unic_bc_0
NIC statistics:
fpga_ver_major: 2
fpga_ver_minor: 5
fpga_ver_patch: 0
l2_rx_packets: 0
l2_tx_packets: 0
txts_fiber_en: 1
pmapcsmac_tx_delay_ps: 0
pmapcsmac_rx_delay_ps: 0
udp_drop_count_pcie_bp (accel): 0
```

```
udp_drop_count_rx_buf (accel): 0
tcp_drop_count_pcie_bp (accel): 0
tcp_drop_count_rx_buf (accel): 0
l2_drop_count_pcie_bp: 0
l2_drop_count_rx_buf: 0
udp_rx_frame_count (accel): 0
udp_rx_broadcast_count (accel): 0
udp_rx_multicast_count (accel): 0
tcp_rx_frame_count (accel): 0
l2_rx_frame_count: 0
udp_tx_frame_count (accel): 0
tcp_tx_frame_count (accel): 0
l2_tx_frame_count: 0
```

c) Display Interface Transceiver Diagnostic Details

Command Syntax

```
sudo ethtool -m <interface_name>
```

Example

```
$ ethtool -m unic_bc_0
Identifier: 0x11 (QSFP28)
Extended identifier: 0xcc
Extended identifier description: 3.5W max. Power consumption
Extended identifier description: CDR present in TX, CDR present in RX
Extended identifier description:  High Power Class (> 3.5 W) not enabled
Power set: Off
Power override: Off
Connector: 0x0c (MPO Parallel Optic)
Transceiver codes: 0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Transceiver type: 100G Ethernet: 100G Base-SR4 or 25GBase-SR
Encoding: 0x03 (NRZ)
BR Nominal: 25500Mbps
Rate identifier: 0x02
Length (SMF): 0km
Length (OM3): 70m
Length (OM2): 0m
Length (OM1): 0m
Length (Copper or Active cable): 50m
Transmitter technology: 0x00 (850 nm VCSEL)
Laser wavelength  : 850.0000nm
Laser wavelength tolerance: 100.0000nm
Vendor name: FS
Vendor OUI: 64:9d:99
Vendor PN : QSFP28-SR4-100G
Vendor rev: 1A
Vendor SN : S2202646025
```

```
Date code: 220310
Revision Compliance: Unallocated
Rx loss of signal: [ Yes, Yes, Yes, Yes ]
Tx loss of signal: None
Rx loss of lock: [ Yes, Yes, Yes, Yes ]
Tx loss of lock: [ Yes, Yes, Yes, Yes ]
Module temperature: 54.12 degrees C / 129.43 degrees F
Module voltage: 3.3250 V
Alarm/warning flags implemented: Yes
Laser tx bias current (Channel 1): 6.792 mA
Laser tx bias current (Channel 2): 6.786 mA
Laser tx bias current (Channel 3): 6.812 mA
Laser tx bias current (Channel 4): 6.806 mA
Transmit avg optical power (Channel 1): 1.1101 mW / 0.45 dBm
Transmit avg optical power (Channel 2): 1.0523 mW / 0.22 dBm
Transmit avg optical power (Channel 3): 1.0582 mW / 0.25 dBm
Transmit avg optical power (Channel 4): 1.1225 mW / 0.50 dBm
Rcvr signal avg optical power (Channel 1): 0.0001 mW / -40.00 dBm
Rcvr signal avg optical power (Channel 2): 0.0001 mW / -40.00 dBm
Rcvr signal avg optical power (Channel 3): 0.0001 mW / -40.00 dBm
Rcvr signal avg optical power (Channel 4): 0.0001 mW / -40.00 dBm
Laser bias current high alarm (Chan 1): Off
Laser bias current high alarm (Chan 2): Off
Laser bias current high alarm (Chan 3): Off
Laser bias current high alarm (Chan 4): Off
Laser bias current low alarm (Chan 1): Off
Laser bias current low alarm (Chan 2): Off
Laser bias current low alarm (Chan 3): Off
Laser bias current low alarm (Chan 4): Off
Laser bias current high warning (Chan 1): Off
Laser bias current high warning (Chan 2): Off
Laser bias current high warning (Chan 3): Off
Laser bias current high warning (Chan 4): Off
Laser bias current low warning (Chan 1): Off
Laser bias current low warning (Chan 2): Off
Laser bias current low warning (Chan 3): Off
Laser bias current low warning (Chan 4): Off
Laser tx power high alarm (Chan 1): Off
Laser tx power high alarm (Chan 2): Off
Laser tx power high alarm (Chan 3): Off
Laser tx power high alarm (Chan 4): Off
Laser tx power low alarm (Chan 1): Off
Laser tx power low alarm (Chan 2): Off
Laser tx power low alarm (Chan 3): Off
Laser tx power low alarm (Chan 4): Off
Laser tx power high warning (Chan 1): Off
Laser tx power high warning (Chan 2): Off
Laser tx power high warning (Chan 3): Off
```

```
Laser tx power high warning (Chan 4): Off
Laser tx power low warning (Chan 1): Off
Laser tx power low warning (Chan 2): Off
Laser tx power low warning (Chan 3): Off
Laser tx power low warning (Chan 4): Off
Laser rx power high alarm (Chan 1): Off
Laser rx power high alarm (Chan 2): Off
Laser rx power high alarm (Chan 3): Off
Laser rx power high alarm (Chan 4): Off
Laser rx power low alarm (Chan 1): On
Laser rx power low alarm (Chan 2): On
Laser rx power low alarm (Chan 3): On
Laser rx power low alarm (Chan 4): On
Laser rx power high warning (Chan 1): Off
Laser rx power high warning (Chan 2): Off
Laser rx power high warning (Chan 3): Off
Laser rx power high warning (Chan 4): Off
Laser rx power low warning (Chan 1): On
Laser rx power low warning (Chan 2): On
Laser rx power low warning (Chan 3): On
Laser rx power low warning (Chan 4): On
Module temperature high alarm : Off
Module temperature low alarm  : Off
Module temperature high warning: Off
Module temperature low warning: Off
Module voltage high alarm : Off
Module voltage low alarm  : Off
Module voltage high warning: Off
Module voltage low warning: Off
Laser bias current high alarm threshold: 11.000 mA
Laser bias current low alarm threshold: 2.000 mA
Laser bias current high warning threshold : 10.000 mA
Laser bias current low warning threshold  : 3.000 mA
Laser output power high alarm threshold: 2.7542 mW / 4.40 dBm
Laser output power low alarm threshold: 0.0912 mW / -10.40 dBm
Laser output power high warning threshold : 2.1878 mW / 3.40 dBm
Laser output power low warning threshold  : 0.1445 mW / -8.40 dBm
Module temperature high alarm threshold: 75.00 degrees C / 167.00
degrees F
Module temperature low alarm threshold: -5.00 degrees C / 23.00 degrees
F
Module temperature high warning threshold: 70.00 degrees C / 158.00
degrees F
Module temperature low warning threshold: 0.00 degrees C / 32.00 degrees
F
Module voltage high alarm threshold: 3.6300 V
Module voltage low alarm threshold: 2.9700 V
Module voltage high warning threshold: 3.4650 V
Module voltage low warning threshold: 3.1350 V
```

```
Laser rx power high alarm threshold: 2.1878 mW / 3.40 dBm
Laser rx power low alarm threshold: 0.0631 mW / -12.00 dBm
Laser rx power high warning threshold : 1.7378 mW / 2.40 dBm
Laser rx power low warning threshold  : 0.1259 mW / -9.00 dBm
```

14. VLAN Support

1) Overview

From v2.0 support for VLAN functionality (IEEE 802.1Q) is implemented.

This section provides information about using VLANs with ÜberSock. The ÜberSock library doesn't handle VLAN interface registration/association itself, it should be done by the standard Linux socket programming or command line tools (eg: IP)

Note:

1. Currently, for acceleration, the per fiber-pair limit of unique VLANs is 15. Accelerated UDP or TCP sessions can only use the 15 first VLAN sets. Additional VLANs are only non-accelerated (L2).
2. When configuring:
   a. Unless running as `root`, prefix commands with "`sudo`"
   b. Disable IPv6:

```
sudo sysctl -w net.ipv6.conf.<interface>/<vlan_id>.disable_ipv6=1
```

2) Setup and Useful Commands

An übernic interface can be associated to a VLAN following this process:

To setup VLAN id 700 on unic_85_0 interface:

```
ip link add link unic_85_0 name unic_85_0.601 type vlan id 601
```

15. ÜberNIC Monitor - (uber_bmc_monitor)

To monitor the health of the ÜberNIC board and its components use uber_bmc_monitor:

1) Retrieve Log File

Obtain ÜberNIC Device_id; Command:

```
ls -la /dev/ub*
```

Example Output:

```
/dev/ubernic_85000
```

Extract log file; Command:

```
<path>/ubernic/uber_bmc_monitor -d <device_id> log
```

2) Real-Time Monitoring – Human Readable

Command:

```
<path>/ubernic/uber_bmc_monitor -d <device_id> monitor --rate="1s"
```

3) Real-Time Monitoring – Machine Readable

Command:

```
<path>/ubernic/uber_bmc_monitor -d <device_id> monitor --rate="1s" --csv
```

16. ÜberNIC FPGA State Capture - (uberinfo)

uberinfo is a tool used to capture and dump all the information of the current FPGA state for all fibers in ÜberNIC.

ÜberINFO requires the `/dev/ubernic_*xxxxx*` driver whereby "*xxxxx*" refers to the PCIe Bus address, with any ":" or "." removed, which is created by the ÜberNIC driver.

For added flexibility, ÜberNIC driver implements dynamic creation and automatic numbering – which means that you don't need to worry about manually reinstalling and getting a new PCIe Bus address when you change ÜberNIC slots.

A. Command Syntax

`uberinfo [OPTION…]`

If no options or arguments are supplied, `uberinfo` will not display anything. Options, if specified, have the following meanings:

Option	Explanation
`-h, --help`	Show context-sensitive help
`-v, --verbose`	Provide a more verbose output
`-d DEVICE`	Specify device file in /dev/

B. Display All Fiber States

Syntax:

`uberinfo -d /dev/ubernic_XXXX`

This command scans all the fiber and dumps the current information for ÜberNIC.

For example, let's say we are interested to get the information related to unic0 which is on the 1st fiber. We can go ahead and use the `uberinfo` tool and supplement it with the built-in grep command.

Here's an example command:

```
uberinfo -d /dev/ubernic_bc001 | grep fiber=0
```

Do note that when referring to the fibers using `uberinfo`, the 1st fiber is designated as fiber0, 2nd fiber is designated as fiber1, 3rd fiber is designated as fiber2, and 4th fiber is designated as fiber3.

17. ÜberNIC Network Packet Dump – (uberdump)

ÜberDUMP (uberdump) is a tool that can be used to capture all traffic, two-way, on a specific ÜberNIC interface and write it to a PCAP file, put another way, ÜberDUMP is essentially an internal tap. Data is captured and timestamped (currently nanosecond timestamps) at the intersection of the MAC/PCS and the Ethernet core.

ÜberDUMP writes .pcapng PCAP files to the filesystem location specified by the user.

ÜberDUMP utilizes DDR4 memory on ÜberNIC as a short-term buffer. To support ÜberDUMP, ÜberNIC's default memory is configured such that each of the eight (8) ÜberNIC interfaces is allocated one (1) GB of DDR4 memory. From the DDR4 memory ÜberDUMP then writes across the PCIe bus to the filesystem. ÜberDUMP does not pre-empt other traffic traversing the PCIe bus; network protocol traffic such as UDP and TCP is given the highest priority.

ÜberDUMP operates at full line-rate, therefore it is critical that sufficient storage – size and r/w speed – is available. For example, two-way 10Gbps results in 2.5GBps of data storage need. It is critical to implement a storage and data-movement strategy which successfully supports your data usage/generation profile.

ÜberDUMP consists of three individual applications, a) uber_dump_cfg, b) uber_dump, and c) uber_dump_splitng, each is described below.

A. Command Syntax

1) uber_dump_cfg

uber_dump_cfg is used to configure the dump at initialization, ideally in a systemd job at boot time. A new configuration requires to discard all the packets already present in the dump, and no dump application to run. It is used to dynamically configure the dump memory. At the moment and for simplicity, only enabling the dump on all configured fibers with the same amount of memory (1GB per-interface) is possible.

Option	Explanation
`-e <fibers>`	Comma-separated list of interfaces to enable. Default is all
`-o <fibers>`	If memory full, comma-separated list of interfaces to discard new packets and retain older, default discards older packets
`-f`	Force reconfiguration. If not set, reconfigure only if the selected fibers are not configured. The -f is used to avoid reconfiguring the fibers if they are already configured, this is so that if placed in a systemd, a reboot does not reset the dump but retains the already dumped packets.

Syntax and example below assume a path prefix of `<path>/ubernic/uber_dump/`

Syntax:

```
uber_dump_cfg -d <device_id> <options>
```

Example Command:

```
uber_dump_cfg -d /dev/ubernic_17000 -e 0,1
```

2) uber_dump

uber_dump is the dump application, it reads the packets that are present on the FPGA DDR, ack them allowing the FPGA to write in the free space, and write to pcap file.

The application can run on one or multiple interfaces, but not multiple applications can run on the same interface.

Example, uber_dump can run on the network interface unic_XX_0 and unic_bc_1 at the same time to write on one pcap file, or it can be used two times on each unic_XX_0 and unic_XX_1. To run uber_dump twice on unic_XX_0 will result in an error.

The default file format is pcapng, this is to have all TX and RX for a particular interface in a single pcapng file. The pcapng format can be converted to pcap a number of ways, one example:

tcpdump -r output.pcapng -w output.pcap --time-stamp-precision=nano

Timestamp resolution is currently nanoseconds, eta to support picosecond timestamps is Q4 2025.

Option	Explanation
`-F <pcap format>`	Optional, PCAP format, pcapng or pcap, defaults to pcapng
`-w <filename>`	PCAP filename, must include “.pcapng” extension
`-r`	Clears ÜberNIC DDR4 buffer
`-i <interface>`	Specify ÜberNIC interface (unic_<pcie address>_<interface #>) specify multiple interfaces with comma-separate list
`-s`	Silent, no logging

Syntax and example below assume a path prefix of `<path>/ubernic/uber_dump/`

Syntax:

```
uber_dump -d <device_id> <interface> <options>
```

Example Command:

```
uber_dump -d /dev/ubernic_17000 -i unic_17_0 -i unic_17_1 -w
<path>/17_0_output.pcapng
```

3) uber_dump_splitng

Helper application to split a pcapng file into separate RX and TX files by interface outputting in .pcap format.

Syntax and example below assume a path prefix of `<path>/ubernic/uber_dump/`

Syntax:

```
uber_dump_splitng -r <input filename> -w <output filename prefix>
```

Example Command:

```
uber_dump_splitng -r <path>/unic_17_output.pcapng -w pcap
```

Example Output:

```
-rwxr-xr-x  1 root root14169 Jul 15 12:39 pcap_unic_17_1_tx.pcap
-rwxr-xr-x  1 root root24 Jul 15 12:39 pcap_unic_17_1_rx.pcap
-rwxr-xr-x  1 root root24 Jul 15 12:39 pcap_unic_17_0_tx.pcap
-rwxr-xr-x  1 root root24 Jul 15 12:39 pcap_unic_17_0_rx.pcap
```

18. ÜberNIC FPGA Logic Management – (uber_bmc_mng)

ÜberBMC (uber_bmc_mng) is a BMC management application tool for ÜberNIC via PCIe. It is used to manage FPGA images, and fetch BMC information such as model, serial number, and versions.

ÜberBMC requires the `/dev/ubernic_*xxxxx*` directory as a “device_id” input whereby “*xxxxx*” refers to the PCIe Bus address, with any “:” or “.” removed, which is created by the ÜberNIC driver.

For added flexibility, the ÜberNIC driver implements dynamic creation and automatic numbering – which means that you don’t need to worry about manually reinstalling and getting a new PCIe Bus address when you change ÜberNIC slots.

A. Command Syntax

`All commands below assume the path (<path>/ubernic/) precedes the command syntax and command examples, ex:`

```
<path>/ubernic/uber_bmc_mng -d /dev/ubernic_bc001 display
```

`uber_bmc_mng [OPTION…]`

If no options or arguments are supplied, `uber_bmc_mng` will not display anything. Options, if specified, have the following meanings:

Option	Explanation
`-h, --help`	Show context-sensitive help (also try --help-long and --help-man)
`-v, --version`	Show application version

`-d, --device=DEVICE`	Specify device file in /dev/
`program`	Program FPGA image into flash
`display`	Display BMC info
`default`	Set the default image

B. Display BMC and ÜberNIC Images Info

Syntax:

```
<path>/ubernic/uber_bmc_mng -d /dev/ubernic_XXXX display
```

Example command and output:

```
<path>/ubernic/uber_bmc_mng -d /dev/ubernic_bc001 display
```

-- BMC INFO --
IA-440I serial number 8170073 part number IA-440i-S-B231E2W-33-S0X-1-2A-0E-9
Active Slot: 1
Slot 1 version: 0.7.0+23
Slot 2 version: 0.4.0+8

-- IMAGES INFO --
UberNIC_v1.0_PCIe_UberWR.rbf <- priority 0
20250609_Rev0_Golden_Image.rbf <- priority 255

Note: Images are configured with a priority from 0 to 255 with -1 being used for non-configured priorities. The image with the lowest priority number will be assigned as the default image (i.e. the highest priority) in ÜberNIC. If the image is corrupted, the board will fall back to the image with higher priority.

Priority 255 is used as “LMS fallback image” to ensure that ÜberNIC has the minimum required logic to run the BMC management application.

C. Setting ÜberNIC Default Image

Syntax:

```
uber_bmc_mng -d <device_id> default <fimename>
```

Example:

```
uber_bmc_mng -d /dev/ubernic_bc001 default UberNIC_v2.0_PCIe_UberWR.rbf
```

D. Programming New Logic

1) Program New Logic: Method 1: Power-Cycle Server

a) Program New Logic

Syntax:

```
uber_bmc_mng -d <device_id> program <filename>
```

Example:

```
uber_bmc_mng -d <device_id> program UberNIC_v2.0_PCIe_UberWR.rbf
```

b) Confirm New Logic Programmed Successfully

Syntax:

```
<path>/ubernic/uber_bmc_mng -d /dev/ubernic_XXXX display
```

Example command and output:

```
<path>/ubernic/uber_bmc_mng -d /dev/ubernic_bc001 display
```

-- BMC INFO --
IA-440I serial number 8170073 part number IA-440i-S-B231E2W-33-S0X-1-2A-0E-9
Active Slot: 1
Slot 1 version: 0.7.0+23
Slot 2 version: 0.4.0+8

-- IMAGES INFO --
UberNIC_v2.0_PCIe_UberWR.rbf <- priority 0
20250609_Rev0_Golden_Image.rbf <- priority 255

c) Power-Cycle Server

2) Program New Logic: Method 2: Power-Cycle PCIe Slot

To implement this programming method your motherboard BIOS and operating system must support, and have enabled, PCIe hot-plugging. The mechanism implemented here relies on PCIe hot-plugging to temporarily remove, and then restore, power from the PCIe slot in which ÜberNIC is inserted and thereby transfer logic loaded from ÜberNIC's flash memory to ÜberNIC's FPGA.

a) Program New Logic

Syntax:

```
uber_bmc_load_image.sh <device_id> <filename>
```

Example:

```
uber_bmc_load_image.sh /dev/ubernic_bc001 UberNIC_v2.0_PCIe_UberWR.rbf
```

b) Confirm New Logic Programmed Successfully

Syntax:

```
<path>/ubernic/uber_bmc_mng -d /dev/ubernic_XXXX display
```

Example Command and Output:

```
<path>/ubernic/uber_bmc_mng -d /dev/ubernic_bc001 display
```

-- BMC INFO --
IA-440I serial number 8170073 part number IA-440i-S-B231E2W-33-S0X-1-2A-0E-9
Active Slot: 1
Slot 1 version: 0.7.0+23
Slot 2 version: 0.4.0+8

-- IMAGES INFO --
UberNIC_v2.0_PCIe_UberWR.rbf <- priority 0
20250609_Rev0_Golden_Image.rbf <- priority 255

19. Time Synchronization - Precision Time Measurement (PTM)

Pre-requisite:

1. ÜberNIC is connected to your server. Please refer to the previous section on how to verify the connection.
2. The necessary driver and supporting software are installed. Please refer to the previous section for the installation instructions.

The following describes in more detail how to install the driver which enables the full support for PTM in ÜberNIC, using Red Hat OS as an example.

A. Driver Installation

If you have already installed the ÜberNIC driver and supporting software, please skip to step 2. Else, please follow through the following steps to ensure a successful PTM enablement:

1) On your system's command line, navigate to the driver's directory:

```
cd driver
```

2) Then, compile the Makefile inside the directory:

```
sudo make
```

3) Install the driver which supports PTM:

```
sudo make install
```

4) Reboot the server to ensure the driver is inserted properly:

```
sudo reboot
```

B. Obtaining ÜberNIC PCIe Address:

Command:

```
lspci | grep Liquid-Markets
```

Expected Output:

c7:00.0 Ethernet controller: Liquid-Markets GmbH

The PCIe address is highlighted in green.

C. Verify ÜberNIC is PTM-Enabled

1) Using the PCIe address obtained above, retrieve the advertised capabilities of ÜberNIC; limit output by grepping only for PTM:

```
sudo lspci -s c7:00.0 -vvv | grep PTM
```

Expected output:

PTMCap: Requester:+ Responder:- Root:-

PTMClockGranularity: Unimplemented

PTMControl: Enabled:+ RootSelected:-

PTMEffectiveGranularity: 2ns

Note: If ÜberNIC is not PTM-enabled, contact LMS to obtain a PTM license

2) Check if the "/dev/ptp*X*" directory exists and find out which clock ÜberNIC is using. *X* refers to a number like 0, 1, 2, or 3 and more. The value of *X* depends on the other devices present (for example TGPIO PPS, other network card with PTP).

Replace "X" with the ptp clock reference returned by the command below:

```
grep -H ubernic_ptp /sys/class/ptp/ptp*/clock_name
```

Example output:

/sys/class/ptp/ptp0/clock_name:ubernic_ptp

In the example above, the *X* value returned in the output is, as indicated by the green highlighting, 0, therefore, ÜberNIC is using ptp0. Make sure to keep note of the clock name since we will be using this later.

Also, if there is no output displayed when you used the command, then there might be an issue with your driver. Even without enabling PTM, the directory "/dev/ptp*X*" should exist.

3) Next, install the "linuxptp" package:

```
sudo dnf install linuxptp -y
```

For **Debian-based OS**, use:

```
sudo apt install linuxptp -y
```

4) Disable NTP by using "timedatectl":

```
sudo timedatectl set-ntp 0
```

5) Sync the ÜberNIC clock to the Host via PTM:

Syntax:

```
sudo phc2sys -s /dev/ptpX -O 0 -m -R 1
```

Here, *X* is the value returned in the output from step a.

Example Command:

```
sudo phc2sys -s /dev/ptp0 -O 0 -m -R 1
```

20. Time Synchronization - LinuxPTP

ÜberNIC supports standard LinuxPTP.

To enable, during the ÜberNIC installation enable hardware timestamps.

See LinuxPTP documentation for further details on installation and configuration of LinuxPTP.

21. Time Synchronization - ÜberPTP

This section describes ÜberPTP an implementation of IEEE 1588-2019 within ÜberNIC's FPGA.

ÜberPTP, and its associated tools, is used to acquire and synchronize time utilizing PTPv2 and PPS, and to configure and monitor such time acquisition and synchronization logic.

ÜberPTP consists of two primary components uber_ptp_mng and uber_ptp_tap.

ÜberPTP requires the ÜberNIC daemon to be started on the corresponding fiber (currently the 4th fiber) which is reserved solely for use with time synchronization network segments. As time synchronization networks are currently almost universally 1GbE, the fiber used for the time synchronization network is 1GbE.

ÜberPTP requires the `/dev/uber_ptp xxxxx` driver ("*xxxxx*" refers to the PCIe Bus address, with any ":" or "." removed) which is created by the ÜberNIC driver. For added flexibility, ÜberNIC driver implements dynamic creation and automatic numbering – which means that you don't need to worry about manually reinstalling and renumbering when you change ÜberNIC slots on the server.

With ÜberPTP, you can perform any of the following actions:

1) View ÜberPTP hardware information status
2) Enabling PTP and PPS in ÜberNIC
3) Disabling PTP and PPS in ÜberNIC
4) Get the PPS snapshot
5) Get the hardware time from the host
6) Set the hardware time to the host time
7) Adjust ÜberNIC time source
8) Dump all packets going through the fiber with PTP logic

A. Command Syntax

```
uber_ptp_mng [OPTION…]
uber_ptp_tap (-d | --device) /dev/ubernic_XXXX OUTPUT_FILE
```

If no options or arguments are supplied, `uber_ptp_mng` displays the information about the utility, like the `--help` option.

Options, if specified, have the following meanings:

Option	Explanation
`-e, --enable`	Enable the PTP/PPS from the configuration file
`--disable`	Disable the PPS and PTP (default) or target specifically PPS or PTP only
`-i, --info`	Display the status and configuration of the device
`-l, --log`	Log the offset and adjustment at a rate of 1 per second
`-c, --config`	Specify the configuration file name to be used
`-d, --device`	Specify the device name
`--pps-snapshot`	Get the PPS snapshot hardware timestamp
`--default-config`	Display the default configuration
`--gettime`	Get the current time in ÜberNIC
`--settime`	Set ÜberNIC time to host
`--source`	Set the hardware time source
`--csv`	Display result in csv format
`-h, --help`	Display the basic syntax and functionality of uber_ptp_mng utility
`-E`	Use end-to-end delay mechanism
`-P`	Use point-to-point delay mechanism
`-2`	Use L2 network transport
`-4`	Use IPv4 network transport
`--pps`	PPS module
`--ptp`	PTP module

B. Displaying Hardware Information Status

Syntax:

```
uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX (--info | -i)
```

This command is used to get the state of the link, the fiber used for the timing and status of the timing module.

Example command and output:

```
uber_ptp_mng -d /dev/uber_ptp_85000 -i
```

```
# LINK INFO :
TIME_FIBER                          : 3
LINK_UP                             : True

#CLOCK INFO:
VERSION                             : 1.2.0
CURRENT_TIME                        : 2024-05-09T03:18:45.110731943
STATUS.IN_SYNC                      : True
STATUS.IN_HOLDOVER                  : False
CLOCK_SOURCE.SELECT                 : hw
CLOCK_SOURCE.SELECTED               : pps

# PPS OUT INFO :
STATUS.ENABLE                       : True
STATUS.IN_SYNC                      : True
CABLE_DELAY                         : 0

# PTP ORDINARY CLOCK INFO :
ENABLE                              : True
VERSION                             : 2.0.1
PROFILE.PROFILE                     : default
PROFILE.TWO_STEP                    : False
PROFILE.SIGNALING                   : False
PROFILE.LAYER                       : layer2
PROFILE.DELAY_MECHANISM             : e2e
VLAN.ENABLE                         : False
DEFAULT.CLOCK_ID                    : 66106CFFFEA1211C
DEFAULT.DOMAIN_NUMBER               : 0
DEFAULT.PRIORITY1                   : 128
DEFAULT.PRIORITY2                   : 128
DEFAULT.CLOCK_VARIANCE              : 65535
DEFAULT.CLOCK_ACCURACY              : 254
DEFAULT.CLOCK_CLASS                 : 248
DEFAULT.GM_ID                       : 0
DEFAULT.GM_TIME_INACCURACY          : 50
DEFAULT.NUMBER_OF_PORTS             : 1
PORT.PEER_DELAY                     : 0
PORT.PORT_STATE                     : pre_master
PORT.PDELAYREQ_INTERVAL             : 0
PORT.DELAYREQ_INTERVAL              : 0
PORT.ANNOUNCE_INTERVAL              : 1
PORT.ANNOUNCE_TIMEOUT               : 3
PORT.SYNC_INTERVAL                  : 0
PORT.ASYMMETRY                      : 0
CURRENT.STEPS_REMOVED               : 0
CURRENT.OFFSET                      : 0
CURRENT.DELAY                       : 0
```

PARENT.PARENT_CLOCK_ID	: 66106CFFFEA1211C
PARENT.PORT_NUMBER	: 0
PARENT.GM_PRIORITY2	: 128
PARENT.GM_PRIORITY1	: 128
PARENT.GM_CLOCK_ID	: 66106CFFFEA1211C
PARENT.GM_CLOCK_VARIANCE	: 65535
PARENT.GM_CLOCK_ACCURACY	: 254
PARENT.GM_CLOCK_CLASS	: 248

Output explanation:

--LINK INFO--

TIME_FIBER	: the index for fiber that has the 1G and PTP logic. The index starts from 0
LINK_UP	: sets to `True` if the corresponding fiber is properly connected

--CLOCK INFO--

VERSION	: version of the clock module
CURRENT_TIME	: current time inside the FPGA. Uses TAI format
STATUS.IN_SYNC	: sets to `True` if the clock is receiving updates from an external master and is synchronized to this master
STATUS.IN_HOLDOVER	: sets to `True` if the clock lost its connection to the master and is in holdover mode
CLOCK_SOURCE.SELECT	: tells the selected clock source. Valid values are: • **host**: the host acts as the master (e.g., via PTM) • **ptp**: PTP module • **pps:** PPS module • **hw:** the source is either ptp or pps if available. PTP is used for second adjustment, PPS for precise adjustment
CLOCK_SOURCE.SELECTED	: tells the source clock currently in use. Valid values are ptp, pps and hw.

--PPS OUT INFO--

STATUS_ENABLE	: tells whether PPS is enabled or not. PPS needs to be enabled to perform time adjustment
STATUS.IN_SYNC	: tells if PPS signals are received every second or not

CABLE_DELAY	: cable delay to be compensated in calculations
--PTP ORDINARY CLOCK INFO--	
ENABLE	: tells whether PTP is enabled or not
VERSION	: version of the PTP module
PROFILE.PROFILE	: PTP Profile used. Valid values are default, power, utility and tsn.
PROFILE.TWO_STEP	: tells if a follow-up message containing tx timestamps are sent
PROFILE.SIGNALING	: tells if signaling messages are sent or not
PROFILE.LAYER	: tells which PTP profile layer is being used. Valid values are: • **layer2**: Peer-to-Peer synchronization on Layer 2 • **layer3**: End-to-End synchronization on Layer 3
PROFILE.DELAY_MECHANISM	: tells which delay mechanism is used. Valid values are: • **e2e**: End-to-End • **p2p**: Peer-to-Peer
VLAN.ENABLE	: tells whether VLAN is enabled or not
DEFAULT.CLOCK_ID	: the clock identity of the PTP module. This value is derived from the MAC address
DEFAULT.DOMAIN_NUMBER	: domain to run PTP on
DEFAULT.PRIORITY1	: PTP Priority
DEFAULT.PRIORITY2	: PTP Priority
DEFAULT.CLOCK_VARIANCE	: PTP Clock Parameters
DEFAULT.CLOCK_ACCURACY	: PTP Clock Parameters
DEFAULT.CLOCK_CLASS	: PTP Clock Parameters
DEFAULT.GM_ID	: Grand Master ID. This is only for power profile
DEFAULT.GM_TIME_INACCURACY	: Grand Master time inaccuracy. This is only for power profile
DEFAULT.NUMBER_OF_PORTS	: number of ports. Value will always be 1
PORT.PEER_DELAY	: peer mean delay value. In nanoseconds.
PORT.PORT_STATE	: port state as defined in IEEE 1588
PORT.PDELAYREQ_INTERVAL	: Peer Delay Request value in log2. For example, the value 0 means 1s; Value 1 means 2s; Value -1 means 0.5s
PORT.DELAYREQ_INTERVAL	: Delay Request value in log2. For example, the value 0 means 1s; Value 1 means 2s; Value -1

	means 0.5s
PORT.ANNOUNCE_INTERVAL	: Announce Message Interval value. In log2
PORT.ANNOUNCE_TIMEOUT	: number of announce messages that can be missed before timeout
PORT.SYNC_INTERVAL	: Sync Message Interval value. In log2
PORT.ASYMMETRY	: signed asymmetry in nanoseconds
CURRENT.STEPS_REMOVED	: number of network hops between the Master and the Slave. If ÜberNIC is Master, the value will be 0
CURRENT.OFFSET	: current offset to the Master in nanoseconds
CURRENT.DELAY	: current End-to-End delay in nanoseconds
PARENT.PARENT_CLOCK_ID	: clock identity of the current master. If ÜberNIC is master, this is the default clock id
PARENT.PORT_NUMBER	: Master port number
PARENT.GM_PRIORITY2	: Grand Master priority
PARENT.GM_PRIORITY1	: Grand Master priority
PARENT.GM_CLOCK_ID	: Grand Master clock id
PARENT.GM_CLOCK_VARIANCE	: Grand Master clock parameters
PARENT.GM_CLOCK_ACCURACY	: Grand Master clock parameters
PARENT.GM_CLOCK_CLASS	: Grand Master clock parameters

C. Getting ÜberNIC Timing Default Configuration

Syntax:

```
uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX --default-config
```

This command displays the default configuration used when user enables PTP and PPS in ÜberNIC. Users can save the output of this command to a file and modify it with the desired configurations.

Example command and output:

```
uber_ptp_mng -d /dev/uber_ptp_85000 --default-config
```

```
[pps]
enable=1
#PPS in delay in ns
cable_delay=0

[ptp]
enable=1
#Uncomment to enable
# vlan=3

[profile]
#Supported profile: default|power|utility|tsn
profile=default
#Supported delay_mechanism: e2e|p2p
```

```
delay_mechanism=e2e
#Supported layer: layer2|layer3
#Only for default profile, else ignored
layer=layer2
signaling=0
two_step=0

[default]
priority1=128
priority2=128
domain_number=0
clock_variance=65535
clock_accuracy=254
clock_class=248
#Only for power profile, else ignored
gm_id=3
#Only for power profile, else ignored
gm_time_inac=32

[port]
#Signed, log2 (0->1s; 1->2s; -1->0.5s)
delayreq_interval=0
#Signed, log2
pdelayreq_interval=0
#In ms
announce_timeout=3
#Signed, log2
announce_interval=1
#Signed, log2
sync_interval=0
#Signed ns
asymmetry=0
#In ns
max_pdelay=800
```

D. Enabling PTP and PPS in ÜberNIC

Syntax:

`uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX (--enable | -e) [(--config-file | -f) CONFIG_FILE]`

This command enables PTP and PPS in ÜberNIC.

By default, ÜberTime will use End-to-End delay mechanism on Layer 2 using the PTP default profile number 0, class 2. These configurations can be changed by supplying the optional argument *CONFIG_FILE* when using the command.

Before enabling PTP and PPS in ÜberNIC, please make sure that the ubernic_daemon is running on the 4th fiber which is dedicated to 1GbE PTPv2.

Example commands and outputs:

1) Enabling PPS and PTP with the default configuration:

```
uber_ptp_mng -d /dev/uber_ptp_85000 -e
```

```
[pps]
enable=1
cable_delay=0

[ptp]
enable=1

[profile]
profile=default
delay_mechanism=e2e
layer=layer2
signaling=0
two_step=0

[default]
priority1=128
priority2=128
domain_number=0
clock_variance=65535
clock_accuracy=254
clock_class=248

[port]
delayreq_interval=0
pdelayreq_interval=0
announce_timeout=3
announce_interval=1
sync_interval=0
asymmetry=0
max_pdelay=800
```

2) Enabling PPS and PTP with custom configuration:

```
uber_ptp_mng -d /dev/uber_ptp_85000 -e -f custom_config.cfg
```

E. Disabling PTP/PPS in ÜberNIC

Syntax:

`uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX --disable [--pps] [--ptp]`

This command disables PTP and/or PPS. By default, if the option `--pps` or `--ptp` is not supplied, the command will disable both PTP and PPS.

Example command:

```
uber_ptp_mng -d /dev/uber_ptp_85000 --disable
```

F. Logging Real-Time Offset and Adjustment in ÜberNIC

Syntax:

`uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX (--log | -l) RATE [--csv]`

This command logs the offset and adjustment at a rate of per second. The optional `--csv` flag can be added to get the result in csv format.

Example command and output:

```
uber_ptp_mng -d /dev/uber_ptp_85000 -l 1
```

sync:True holdover:False src:pps offset:1 adj:-10 drift:+590
sync:True holdover:False src:pps offset:4 adj:+6 drift:+581
sync:True holdover:False src:pps offset:6 adj:-8 drift:+584
sync:True holdover:False src:pps offset:-3 adj:+0 drift:+584
sync:True holdover:False src:pps offset:-4 adj:+2 drift:+584

Output explanation:

sync	: Defined if the clock is synchronized with the source
holdover	: Sets to `False` if there is a valid clock source
offset	: Last offset measured that will be adjusted in nanoseconds
adj	: Adjust the offset of the hardware clock in nanoseconds
drift	: frequency difference between the master and slave clock, in ppb

G. Getting ÜberNIC PPS Snapshot

Syntax:

```
uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX --pps-snapshot
```

This command allows for an easy view of the rising time and falling time of the received PPS signal.

Example command and output:

```
uber_ptp_mng -d /dev/uber_ptp_85000 --pps-snapshot
```

rising: 1715320512.999999994 falling: 1715320513.010000006
rising: 1715320513.999999990 falling: 1715320514.009999994
rising: 1715320514.999999987 falling: 1715320515.009999999
rising: 1715320515.999999995 falling: 1715320516.010000007
rising: 1715320517.000000014 falling: 1715320517.010000026
rising: 1715320518.000000015 falling: 1715320518.010000027
rising: 1715320518.999999999 falling: 1715320519.010000011
rising: 1715320519.999999991 falling: 1715320520.010000003

H. Displaying Hardware Time

Syntax:

```
uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX -gettime
```

This command displays the hardware time.

Example command and output:

```
uber_ptp_mng -d /dev/uber_ptp_85000 --gettime
```

1715159050.930602222
2024-05-08T09:04:10.930602222

I. Setting Hardware Time to Host

Syntax:

```
uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX --settime
```

This command sets the hardware time to the host time.

Example command and output:

```
uber_ptp_mng -d /dev/uber_ptp_85000 --settime
```

Time set to:

1715158985.268520159
2024-05-08T09:03:05.268520159

J. Adjusting ÜberNIC Time Source

Syntax:

```
uber_ptp_mng (--device | -d) /dev/uber_ptp_XXXX --source SRC
```

Valid values for SRC are:

- **hw**: uses PTP and switches to PPS if available. This is the default value.
- **host**: uses host time
- **ptp**: uses PTP module
- **pps**: uses PPS module
-

This command can change ÜberNIC's time source to use either hardware, host, PTP or PPS. By default, ÜberNIC time source is set to hardware (hw).

Example command:

```
uber_ptp_mng -d /dev/uber_ptp_85000 --source ptp
```

K. Tap PTP Layer for Debugging

Syntax:

```
uber_ptp_tap /dev/ubernic_XXXX output.pcap
```

For debugging purposes, `uber_ptp_tap` application is used to dump all packets going through the fiber that has the PTP logic.

This gives an advantage to users since PTP messages are usually consumed by the PTP module and cannot be viewed from tcpdump on the corresponding interface.

Example command:

```
uber_ptp_tap /dev/ubernic_85000 output.pcap
```

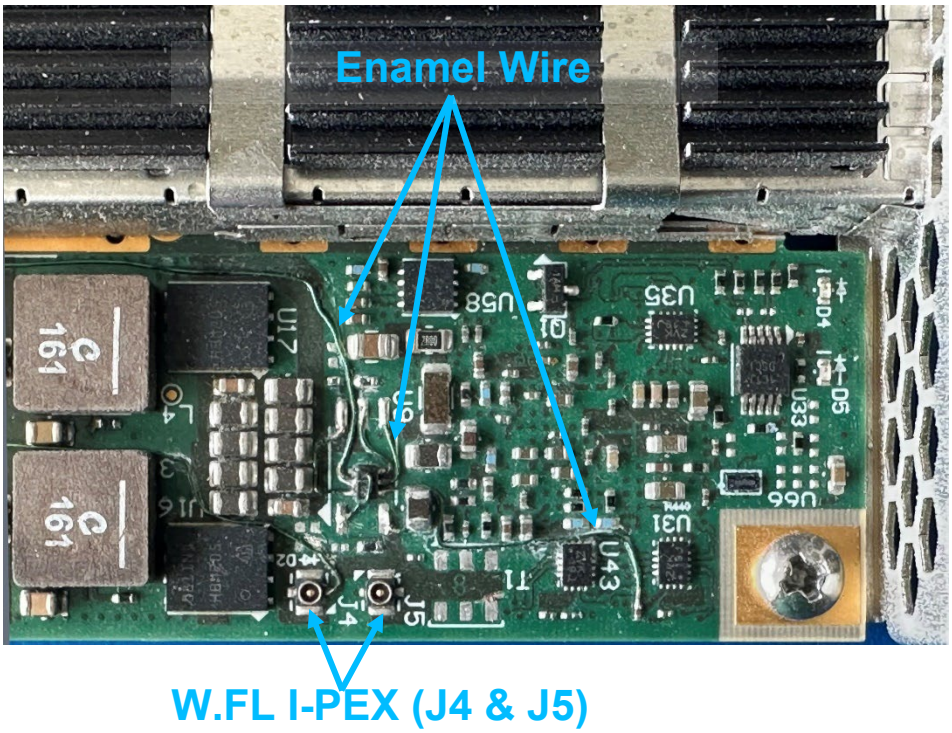
22. Time Synchronization - ÜberWR (White Rabbit)

Notes:

1) **White Rabbit requires both an ÜberNIC specifically modified to support White Rabbit (ÜberWR) and a compatible upstream White Rabbit device**. Please contact LMS to obtain additional information.
2) The White Rabbit assets are installed as part of the standard ÜberNIC installation within the "uberwr" directory within the "ubernic" parent directory.
3) **Please note that if you are using an ÜberNIC PoC board you MUST complete Steps "A" through "C" prior to any other installation steps.**
4) **ÜberWR** supports any dual-fiber optical transceiver but such transceiver must be calibrated properly; please see Section D, Calibrate Transceiver, below.

A. PoC Installation and Configuration

If you are using a PoC ÜberNIC, it has been manually reworked a) via the inclusion of surface-routed enamel wiring to b) allow the two W.FL I-PEX (J4 and J5) to (configurably) provide PPS In or PPS Out Capability. With White Rabbit, both are configured as PPS Out. Board details:



- Unlike production ÜberNICs shipped to you by BittWare, the PoC ÜberNIC requires the manual loading of a custom clock configuration file. For production ÜberNICs, BittWare installs the clock configuration file at the factory prior to shipment.
- Installation of the BittWare SDK and use of the Pico-Lock to USB Type-A Programming Cable is required prerequisite. Please see “ÜberNIC Board Management and Logic Programming User Guide v20240422.pdf” available on ftp.lms.io

B. Physical Installation

1) Set the server fan-speed PWM-equivalent to 85% of maximum

2) Power-down the server

3) Connect a Pico-Lock Programming Cable to ÜberNIC

4) Install ÜberNIC into an available PCIe (ideally Gen 5 x16 lanes) slot

5) Connect the Programming Cable’s USB Type-A connector to the server

6) Power-up the server.

C. Download Logic Files

1) Driver_API: (wget http://ftp.lms.io/<yyyymmdd>_UberNIC_v2.0.tar.gz)

2) Golden Image: (wget http://ftp.lms.io/<yyyymmdd>_GoldenImage.rbf)

3) ÜberWR (wget http://ftp.lms.io/<yyyymmdd>_UberNIC_v2.0_PCIe_UberWR.rbf)

4) Custom Clock Configuration (wget http://ftp.lms.io/wr_140M625_ref3_clock.json)

D. Update/Install Driver_API Package

1) If updating from a prior installation, uninstall the current Driver_API package

```
cd <path>/ubernic/
```

```
./uninstall.sh
```

```
lsmod | grep ubernic
```

If lsmod returns “ubernic”:

```
rmmod ubernic
```

```
cd..
```

```
rm -f -R ubernic
```

2) Install the new Driver_API package

```
tar -xzvf <path>/<yyyymmdd>_UberNIC_v2.0.tar.gz
```

```
cd <path>ubernic/
```

```
./install.sh
```

3) Check For ÜberBMC Compatibility

```
ls -la /dev/ub*
```

Example output:

```
/dev/ubernic_85000
```

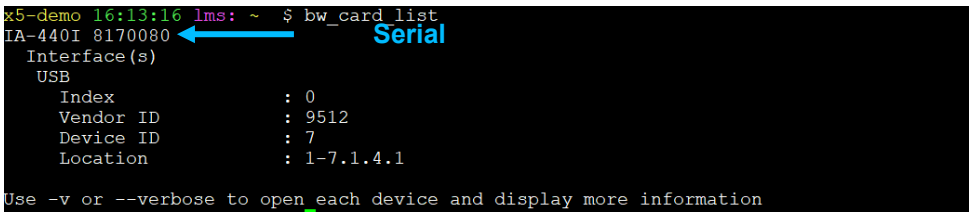
4) Install Golden Image

a) If Step 3, “Check For ÜberBMC Compatibility”, did not identify a valid device, use the BittWare SDK and Programming Cable to complete Steps “i” and “v” below.

i. Obtain the ÜberNIC serial number; Command:

```
bw_card_list
```

Expected Output:



ii. Program the Golden Image to ÜberNIC’s flash memory; Command:

```
bw_bmc_fpga_load -d USB:<serial number>
program<path>/<yyyymmdd>_GoldenImage.rbf
```

iii. Set the Golden Image priority; Command:

```
bw_bmc_fpga_load -d USB:<serial number> priority 255
<yyyymmdd>_GoldenImage.rbf
```

iv. Power-cycle the server

v. After the server restarts, repeat Step 3, Check For ÜberBMC Compatibility, if Step 3, “Check For ÜberBMC Compatibility”, returns a valid ÜberNIC Device_id proceed to Step 5 below.

b) If Step 3, “Check For ÜberBMC Compatibility”, returns a valid device proceed to Step 5 below.

5) Installing Custom Clock Configuration

```
cd <path>/ubernic/uberwr/wr_140M625_ref3/
```

```
<path>/ubernic/uber_bmc_mng -d <device_id> clock-program
wr_140M625_ref3_clock.json
```

6) Power-Cycle Server

E. Obtain ÜberNIC Device_id

Command:

```
ls -la /dev/ub*
```

Example output:

```
/dev/ubernic_85000
```

F. Installing ÜberWR Image

Note: To take full advantage of ÜberBMC enable “Hot Pluggable” for the PCIe slot in which ÜberNIC is installed.

With the installation of the Golden Image, ÜberNIC is now fully compatible with ÜberBMC and there is no further need to use the BittWare SDK and Programming Cable. ÜberBMC will load the new image file to ÜberNIC’s flash memory and “bounce” the PCIe slot in which ÜberNIC is installed. As power is restored to the PCIe slot the new image will be automatically loaded into the ÜberNIC FPGA. No server power-cycle is required.

Command:

```
<path>./uber_bmc_load_image.sh <device_id>
<path>/<yyyymmdd>_UberNIC_v2.0_PCIe_UberWR.rbf
```

G. Starting ÜberWR – Manual

From the “uberwr” directory:

Command syntax:

```
./uberwr_mng <UberNIC device> uberwr.cfg uberwr.bin
```

Example command:

```
./uberwr_mng /dev/ubernic_85000 uberwr.cfg uberwr.bin
```

H. Starting ÜberWR - Automatic

1) Enable systemd job

The installation of the ÜberNIC Driver_API package automatically installs, but does not enable, an ÜberWR system job - uber_wr.service. To enable:

Example command:

```
systemctl enable uber_wr.service
```

```
systemctl start uber_wr.service
```

I. Monitor White Rabbit

1) White Rabbit Switch

a) Open Connection

Command syntax:

```
ssh <user>@<ip_address>
```

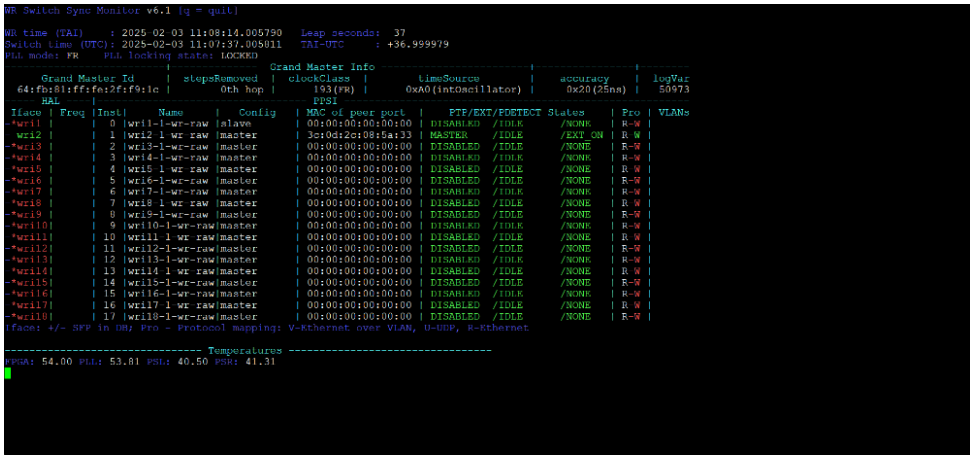
Example command:

```
ssh lms@192.168.188.245
```

b) Launch Monitor:

Command:

```
wr_mon
```



2) ÜberWR Monitor (Human Readable)

a) Open connection to uart:

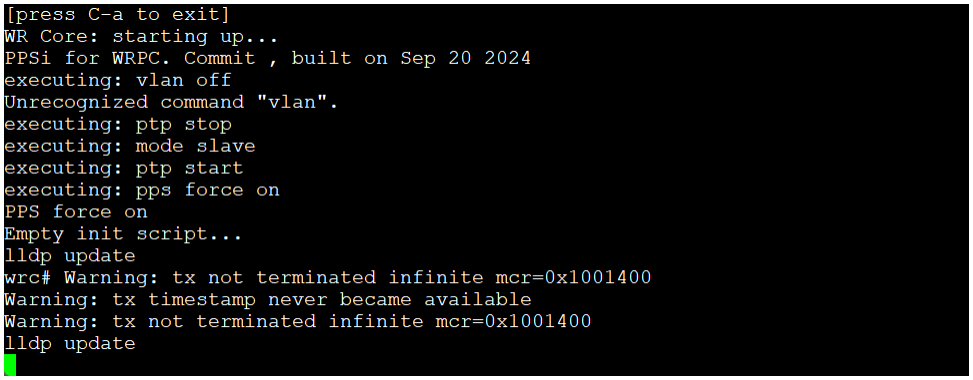
Command syntax:

```
./uberwr_uart <UberNIC device> uberwr.cfg uberwr.bin
```

Example command:

```
./uberwr_uart /dev/ubernic_85000 uberwr.cfg uberwr.bin
```

While the application is starting there will be some output (see below), this can be ignored; successful establishment of the connection is indicated by the output “lldp update” (see below).



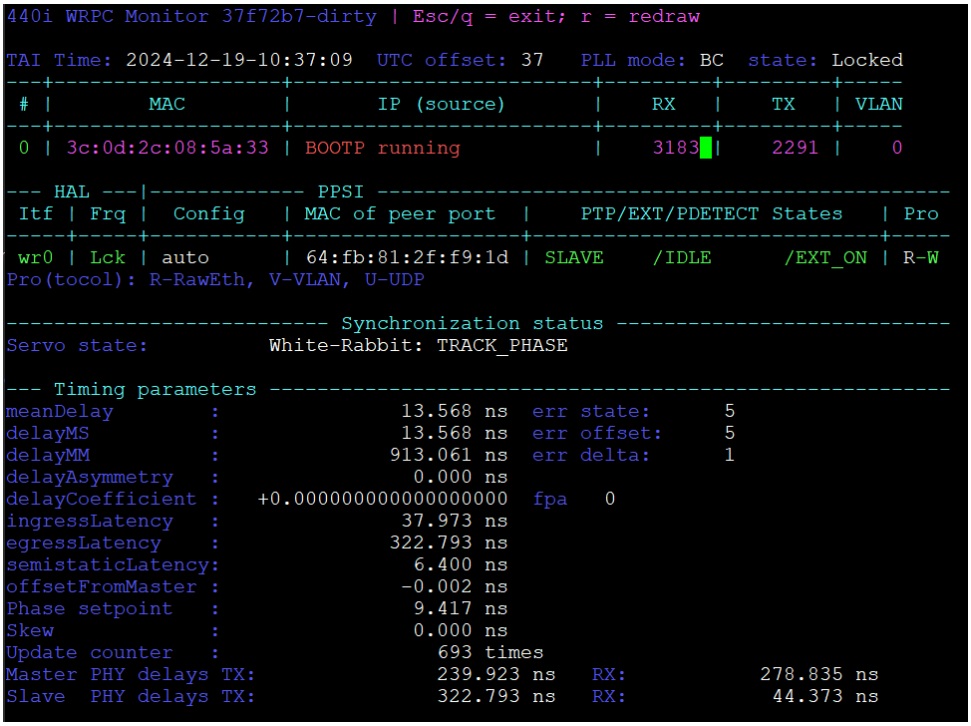
Press <Enter> you will then see the following prompt:

```
wrc#
```

b) Launch ÜberWR GUI

At the wrc# prompt, type “gui” and press <Enter>, to launch:

```
wrc# gui
```



The ÜberWR monitor application refreshes once per second.

The ÜberWR monitor application is extremely sensitive to the size of the terminal. If the terminal window is too small, the components of the application will not display properly. You will need to experiment with various width and height settings to determine a workable solution for your system.

Descriptions and definitions of White Rabbit monitor application components:

Option	***Description***
TAI Time	*International Atomic Time*
UTC Offset	*Difference between TAI and UTC in seconds*
PLL mode	*Role defined by PTP standard; BC: Boundary Clock; GM: Grand Master*
State	*PLL State; Locked (expected state), Unlocked (unexpected – no connection)*
MAC	*MAC address of the board*
IP (source)	*IP address assigned by BOOTP or defined statically*
RX	*Counter of received frames*
TX	*Counter of transmitted frames*
VLAN	*VLAN ID*
Itf	*Network Interface Name*
Frq	*Lck (frequency locked - expected state)*
Config	*Automatic or Manual (slave or master)*
MAC of peer port	*MAC address of the peer port*
PTP/EXT/PDETECT States	*/EXT_ON*
Pro	*Protocol, White Rabbit, High Accuracy, or PTP*

Option	Description
Servo State	The state of the servo (e.g., TRACK_PHASE)
meanDelay	Average of the delay between master and slave (RTT/2)

3) ÜberWR Monitor (Machine Readable)

The “uberwr_diags” application is a command line tool which delivers character delimited output of various KPIs once per second.

a) Open connection to uart

From the “uberwr” directory:

Command syntax:

```
<path>/ubernic/uberwr/uberwr_diags <argument(s)> <UberNIC device>
```

Example command:

```
<path>/ubernic/uberwr/uberwr_diags -g -h /dev/ubernic_85000
```

The user can choose to return one or more of twenty-two values, specifically:

Argument	Option	Description
-0:	SERVO_STATE	
-1:	LINK_UP	
-2:	PLL_LOCKED	The state of the PLL. When locked, any slight change in the input signal first appears as a change in phase between the input signal and the oscillator frequency. This phase shift then acts as an error signal to change the frequency of the local PLL oscillator to match the input signal.
-3:	PTP_STATE	PTP state as defined in IEEE 1588
-4:	TX_PTP_FRAME_CNTS	Number of PTP frames transmitted
-5:	RX_PTP_FRAME_CNTS	Number of PTP frames received
-6:	ROUND_TRIP	Delay from Master to Slave and from Slave to Master in picoseconds; also known as round-trip delay. This value includes the ingressLatency and egressLatency of both peers.
-7:	MASTER_SLAVE_DELAY	Delay between Master and Slave (one way) in picoseconds. This value does not include ingressLatency or egressLatency.

Argument	Option	Description
-8:	CLOCK_OFFSET	Offset between of a local and master clock; if this value is not zero then local servo adjusts the offset.
-9:	PHASE_SETPOINT	Phase offset between the RX clock recovered from the slave port and the WR clock, i.e., WR clock is syntonized to the RX clock with a programmable phase offset which is the phase setpoint.
-a:	UPDATE_COUNTER	Number of servo updates since the link was established.
-b:	BOARD_TEMPERATURE	Temperature of the board.
-c:	RX_ERROR_COUNT	
-d:	SERVO_UP_TIMESTAMP	
-e:	SERVO_RESTART_COUNT	
-f:	TRANSCEIVER_BITSLIDE	Semi-static latency of the RX path
-g:	DELTA_RX_M	This is a sum of the Master port ingressLatency (already includes SFPs RX delay used in master port) and its bitslide (semistaticLatency).
-h:	DELTA_RX_S	This is a sum of ingressLatency (already includes SFPs RX delay used in a slave port) and the bitslide (semistaticLatency) on the slave port of this device.
-i:	DELTA_TX_M	This is the Master port egressLatency (already includes SFPs TX delay used in master port).
-j:	DELTA_TX_S	This is egressLatency on the slave port of this device (already includes SFPs TX delay used in a slave port).
-k:	HELPER_DAC	
-l:	IDX_DIAGS_MAIN_DAC	

J. Calibrate Transceiver

The steps described below assume that your White Rabbit Switch is fully calibrated and aligned to the desired Time Master/GrandMaster.

To properly align “time” between ÜberWR and a directly connected White Rabbit switch it is necessary to compare the respective PPS signals in an oscilloscope. Given the level of granularity which White Rabbit provides the use of a 4-channel oscilloscope with at least 200 MHz bandwidth and a minimum simultaneous sampling rate of 500 MSa/s (across all 4 channels) should be considered the absolute minimum.

Note: Installation of the BittWare SDK and use of the Pico-Lock to USB Type-A Programming Cable is required. Please see “ÜberNIC Board Management and Logic Programming User Guide v20240422.pdf” available on lms.ftp.io

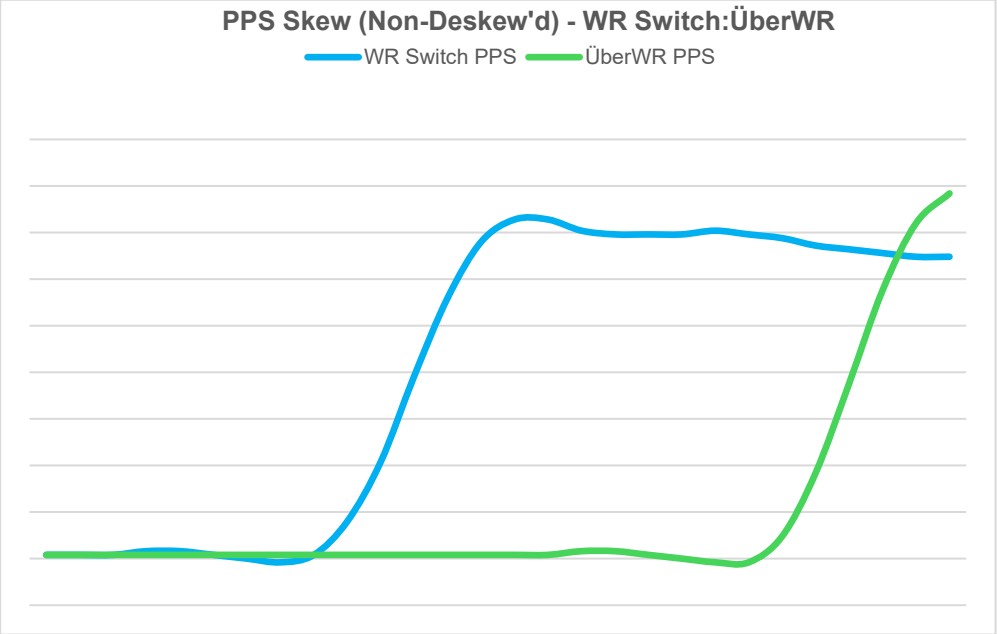
1) Connect PPS Cables

ÜberNIC implements two W.FL I-PEX (J4 and J5) on the PCB. Both of these connectors are, by default, PPS Out. Depending on the model, your White Rabbit Switch implements one or more SMA or SMC connectors. Please ensure that you have obtained the applicable equal-length and equal-impedance (ohm) co-axial cables (or combination of short-length adapter dongle to longer-length cable) to connect the ÜberNIC and White Rabbit Switch PPS Out connectors to the oscilloscope.

2) Measure Skew

a) Following the oscilloscope instructions, measure the skew between the White Rabbit Switch and ÜberNIC. Allow measurements to be acquired for at least several tens-of-minutes. In addition to a waveform, the oscilloscope should return several KPIs including mean, minimum, max, standard deviation, current, and record count. We are interested in the mean skew between ÜberNIC and the White Rabbit Switch.

Example output:



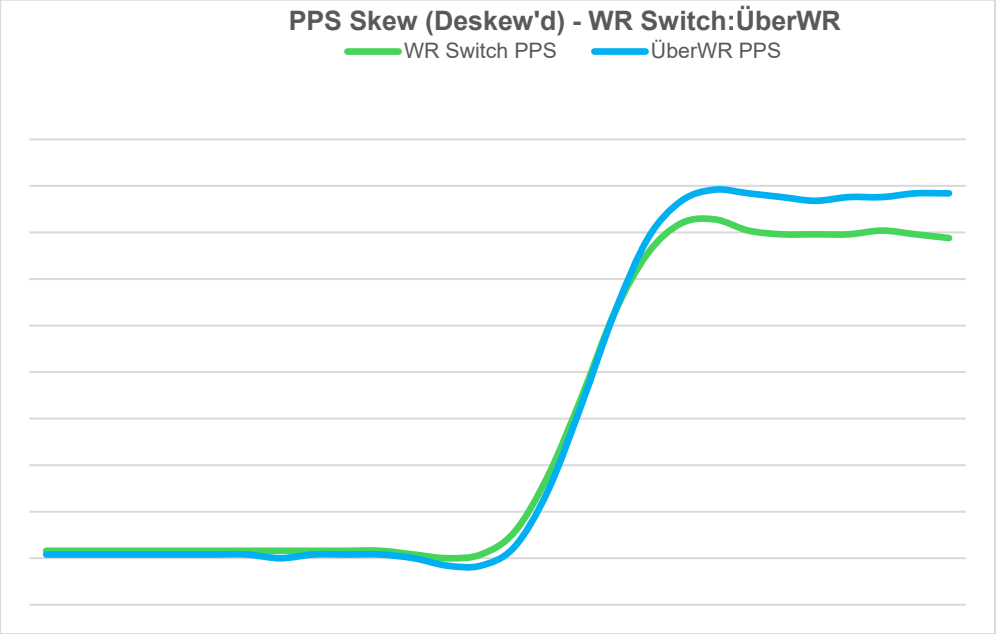
3) Deskew

a) As mentioned above, it is assumed that the White Rabbit Switch is fully calibrated and aligned to the desired Time Master/GrandMaster therefore it is necessary to calibrate/align ÜberNIC to the White Rabbit Switch to which it is connected.

b) Using the mean skew obtained above (#2, a), in the oscilloscope, using the channel to which ÜberNIC is connected, adjust the deskew value to equal the mean skew value obtained above (#2, a). The ÜberNIC waveform must converge to the White Rabbit Switch waveform not the opposite. Allow measurements to be acquired for at least several tens-of-minutes.

c) Fine-tune the deskew value to obtain the lowest-possible mean skew. Allow measurements to be acquired for at least several tens-of-minutes between adjustments.

Example output:



4) Identify Connected SFP/QSFP

a) Open connection to uart

Command syntax:

```
./uberwr_uart <UberNIC device> uberwr.cfg uberwr.bin
```

Example command:

```
./uberwr_uart /dev/ubernic_85000 uberwr.cfg uberwr.bin
```

While the application is starting there will be some output (see below), this can be ignored; successful establishment of the connection is indicated by the output “lldp update” (see below).

```
[press C-a to exit]
WR Core: starting up...
PPSi for WRPC. Commit , built on Sep 20 2024
executing: vlan off
Unrecognized command "vlan".
executing: ptp stop
executing: mode slave
executing: ptp start
executing: pps force on
PPS force on
Empty init script...
lldp update
wrc# Warning: tx not terminated infinite mcr=0x1001400
Warning: tx timestamp never became available
Warning: tx not terminated infinite mcr=0x1001400
lldp update
```

Press <Enter> you will then see the following prompt:

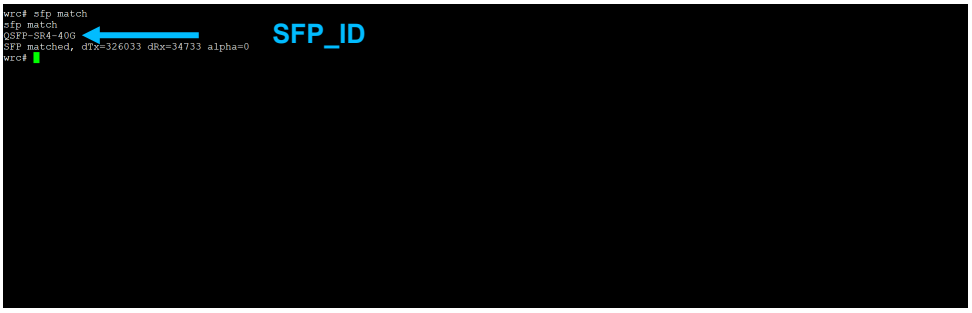
```
wrc#
```

b) Run “sfp match” to match the installed SFP/QSFP to those previously registered.

Command:

```
wrc# sfp match
```

Example output:



c) If “sfp match” does not find a match refer to #5.

d) If the SFP/GSFP matched note the dTx and dRx values and proceed to #6.

5) Add New SFP/QSFP

a) Using the SFP ID obtained at #4 using sfp match, add an entry to the uberwr.cfg file using the template below with your calibration values for alpha, delta_tx, delta_rx

Command:

```
cd <path>/ubernic/uberwr/
```

```
vi uberwr.cfg
```

[sfp.<SFP_ID>]
alpha=0
delta_tx=0
delta_rx=0

b) Initial calibration values:

i. With symmetrical links, the alpha is 0 and symmetrical delays will be compensated, so it is possible to use values from a different SFP and proceed to the calibration using the PPS (Step 6 below).

ii. If the SFP uses asymmetrical wavelengths TX/RX, the alpha is not 0, and symmetrical delays need to be compensated to account for the fiber asymmetry. In that case, proceed to the full calibration of the delays as defined in the WR calibration manual.

6) Adjust SFP/QSFP Calibration Values

a) Edit the applicable SFP/QSFP values in the uberwr.cfg in your preferred editor by first opening the configuration file.

Command:

```
Vi <path>/ubernic/uberwr/uberwr.cfg
```

b) Identify the applicable SFP/QSFP section in uberwr.cfg as returned above (#4, b) and note the existing dTx and dRx values.

c) Convert to final deskew value obtained above (#3, c) to picoseconds by multiplying by 1000. If the ÜberWR PPS is behind (to the right) of the White Rabbit Switch PPS, subtract the picosecond value from the existing dTx value and add the picosecond value to existing dRx value. If the ÜberWR PPS is ahead (to the left) of the White Rabbit Switch PPS, add the picosecond value to the existing dTx value and subtract the picosecond value from the existing dRx value.

d) Save changes to uberwr.cfg and exit.

e) To load new values:

Command Syntax:

```
cd <path>/ubernic/uberwr/wr_140M625_ref3/
```

```
bw_bmc_clock_programmer program SiT95148 wr_140M625_ref3.variants.json
wr_140M625_ref3_registers.nvm.py
```

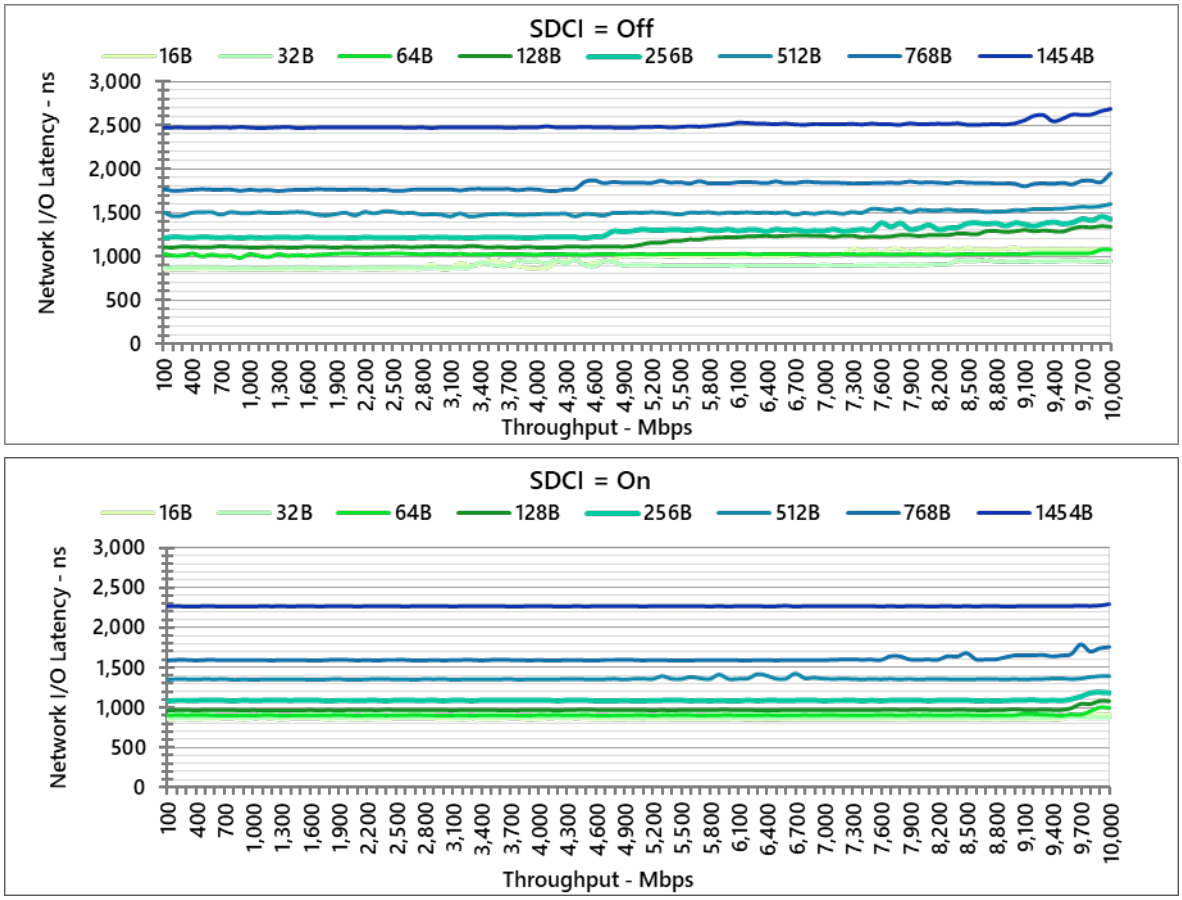
```
bw_bmc_clock_programmer reload SiT95148
```

f) Power-Cycle the server to load updated values from ÜberNIC's flash memory to the FPGA.

23. Experimental: AMD Smart Data Cache Injection (SDCI)

1. AMD's SDCI is currently only available in EPYC Turin (95xx) CPUs
2. SDCI must be explicitly enabled within the server BIOS

A. Latency Performance



B. To Run

1) Edit the ubernic_ethernet.service

Append the "-s" flag after the "-k" flag:

```
vi /etc/systemd/system/ubernic_ethernet.service
```

2) Restart the service

```
systemctl restart ubernic_ethernet.service
```

3) Verify SDCI is running

```
journalctl -u ubernic_ethernet.service
```

Expected Output:

```
ubernic_daemon[3054]: PCIe TLP transaction hints: 7
```

24. Experimental: Intel Time-Aware GPIO PPS

Notes:

1. As of the date of this document, Intel's TGPIO PPS is made available by Intel as an early-adopter prototype. LMS is at the forefront of bringing TGPIO PPS into the realm of general usage, including working with Server, Mainboard, and OS OEMs to enable TGPIO PPS within the broader ecosystem.
2. If you choose to enable and install TGPIO PPS, you must do so after the OS installation and upgrade to support CXL is complete but **BEFORE** enabling and installing ÜberNIC itself and PTM.
3. As of the date of this document, TGPIO PPS is available on standard production motherboards from Jabil (J322-S) and ASRockRack (W790D8UD-1L1N2T/BCM R3.03 and SP2C741D16-2T) by way of one or both of two uf.l connectors mounted on the motherboard.
4. The following steps describe in more detail the BIOS enablement, custom kernel compilation, and the driver and software installation required to enable full support of TGPIO PPS in ÜberNIC. Do note that the steps below were executed on **RHEL 9.3** OS with a prototype ASRock SP2C741D16-2T motherboard.

A. Preparing Kernel Build

Note: All commands should be run as root.

1) Preparing Kernel Build and Installing Kernel Patch

a) Create a directory named "kernel" in your system to store the custom kernel:

```
mkdir kernel
```

b) Navigate into the newly created directory:

```
cd kernel/
```

c) Copy the Time Aware GPIO Support patch zip obtained from LMS into the directory created previously and unzip the file:

```
unzip tgpio-patch.zip
```

d) Install git:

```
dnf install git -y
```

e) Download Linux by getting the kernel from git:

```
git clone\
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git
```

f) Once Linux is downloaded, a directory called “linux” will be created automatically. Navigate into the newly created “linux” directory:

```
cd linux/
```

g) Next, checkout the Linux kernel version you intended to use. In this example, we are using the version 6.9.1:

```
git checkout v6.9.1
```

h) Install the packages (patch, dwarves) needed to apply the patch:

```
dnf install patch dwarves -y
```

i) Next, apply the “Time Aware GPIO Support” patch to the kernel. Errors can be ignored:

If using Xeon Gen 4 or Gen 5:

```
patch -p1 < ../tgpio-patch/tgpio-kernel-6_9_1-static.patch
```

If using Xeon Gen 6:

```
patch -p1 < ../tgpio-patch/tgpio-kernel-6_9_1-static
_birchs.patch
```

B. Modifying the Kernel Configuration

1) Copy the boot configuration file:

```
cp /boot/config-$(uname -r) ./.config
```

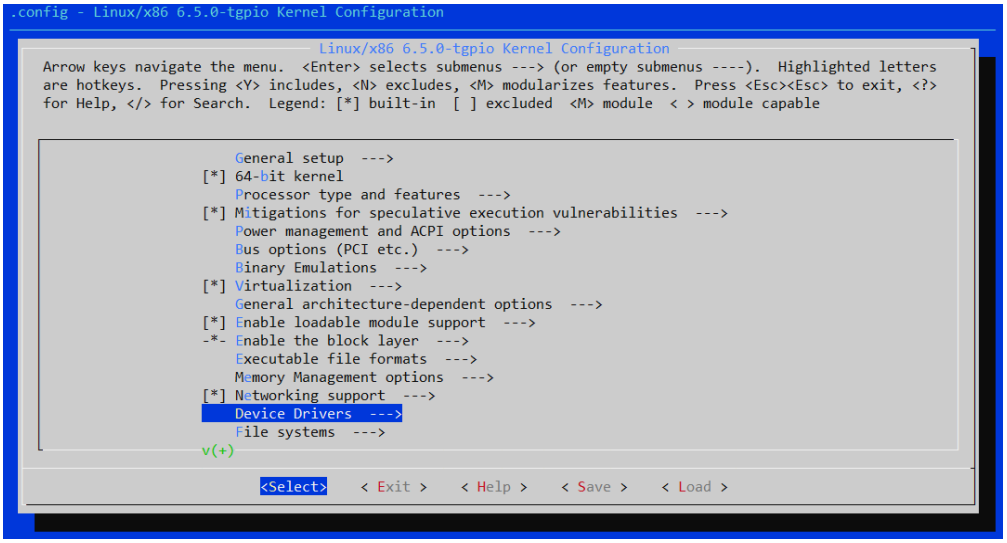
2) Install the “ncurses” package which is needed for the Linux Kernel Configuration GUI:

```
dnf install ncurses ncurses-devel -y
```

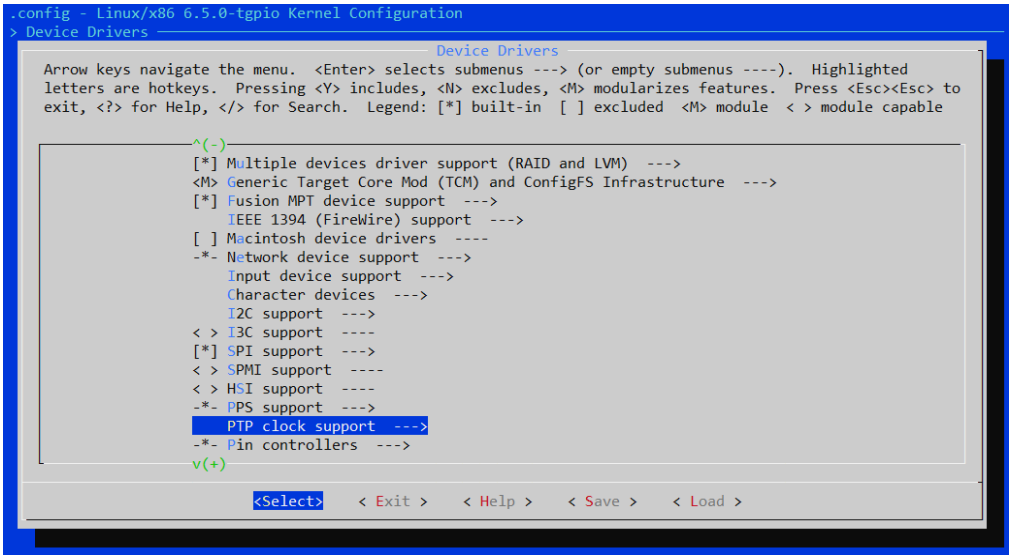
3) Open the GUI for Linux Kernel configuration:

```
make menuconfig
```

4) On the GUI, navigate to the “Device Drivers” menu and press the **Enter** button on your keyboard:

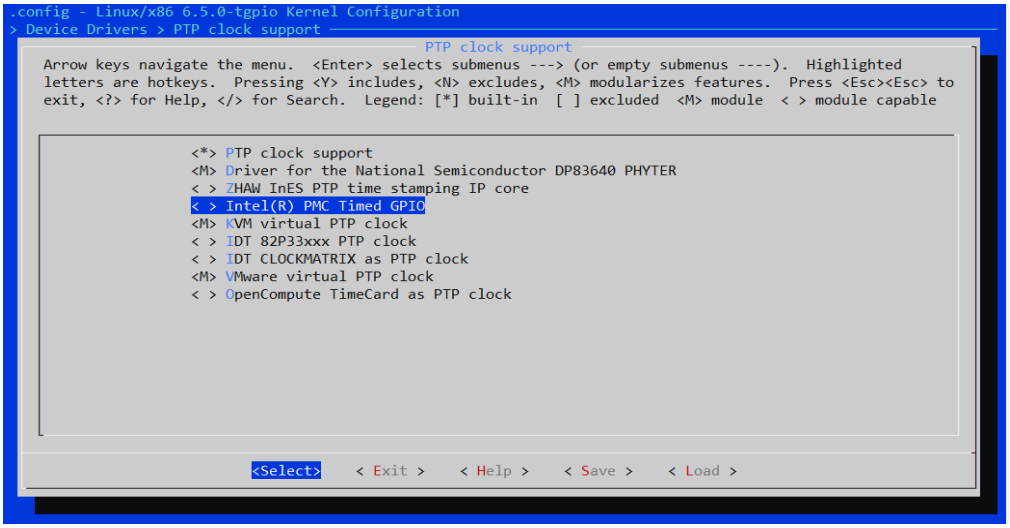


5) Navigate to the “PTP Clock Support” section and press <Enter>:

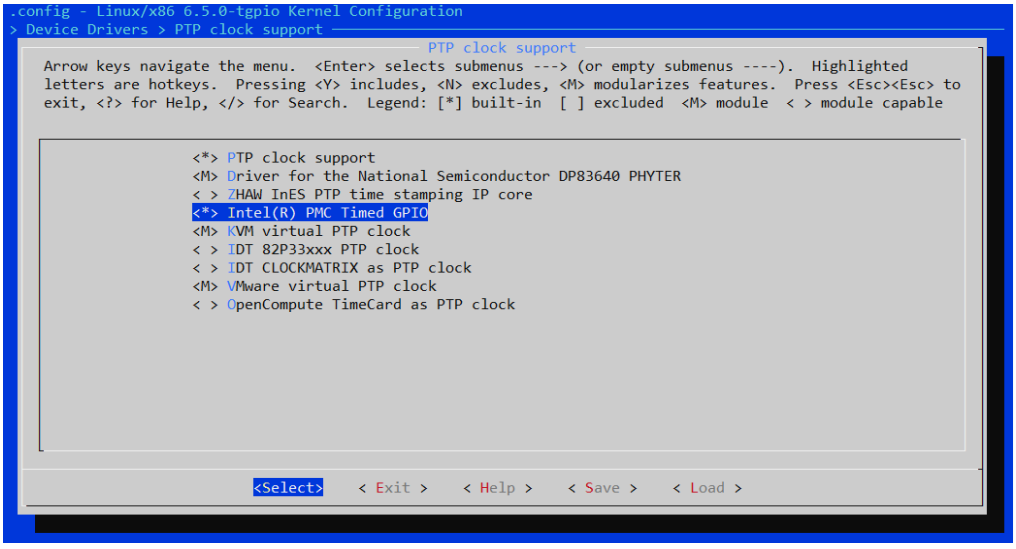


6) Next, navigate to “Intel(R) PMC Timed GPIO” and make sure this module is included in the Linux kernel by default once your system reboots. This can be done by changing the “Intel® PMC Timed GPIO” from “<>” to “<*>” by pressing <Space>.

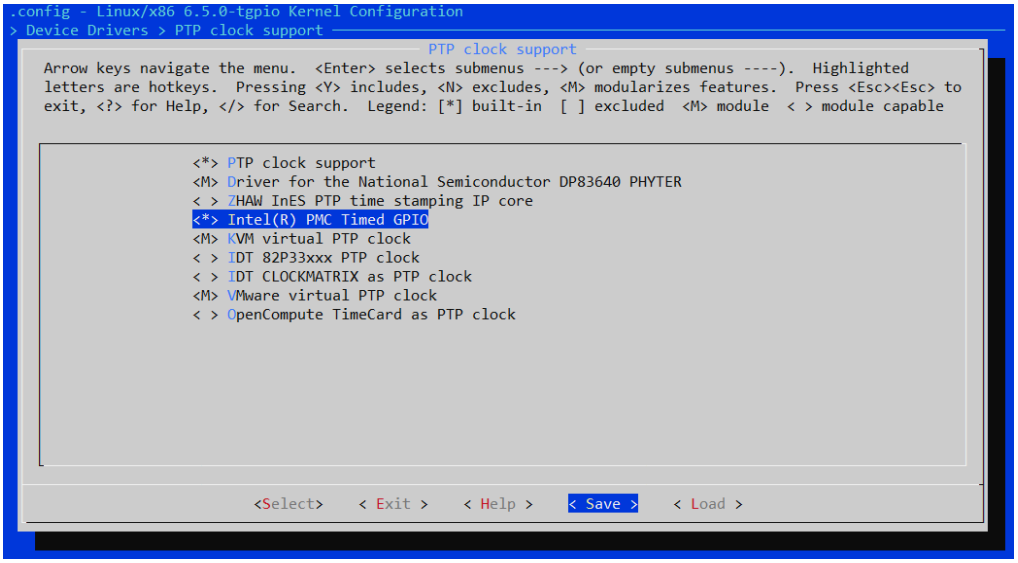
Before:



After:

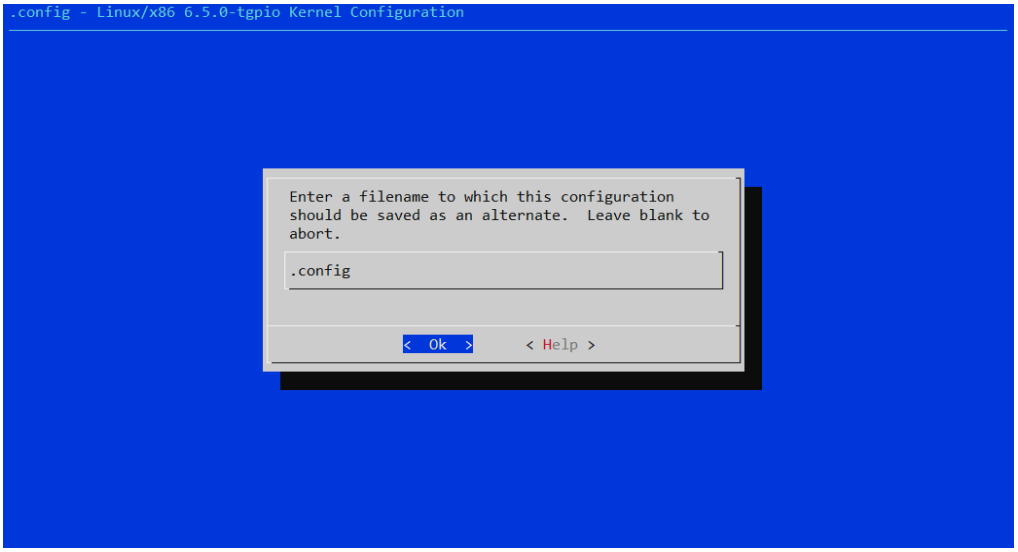


7) Save the configuration by navigating to <Save> and press <Enter>:

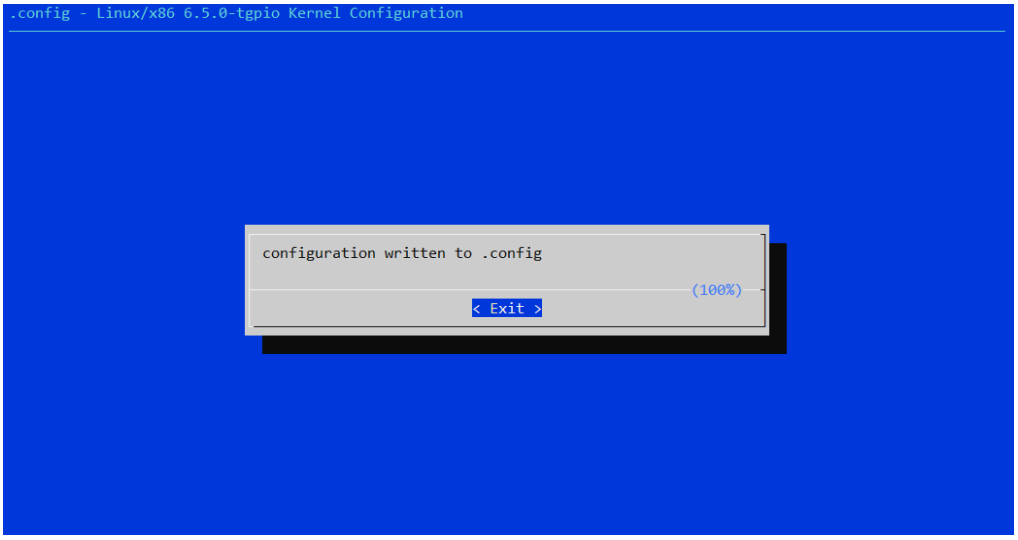


8) Then, enter the filename to which this configuration should be saved chose press <Enter> to choose “Ok”:

In this example, we just accepted “.config” which was the default name provided in the Kernel Configuration GUI:

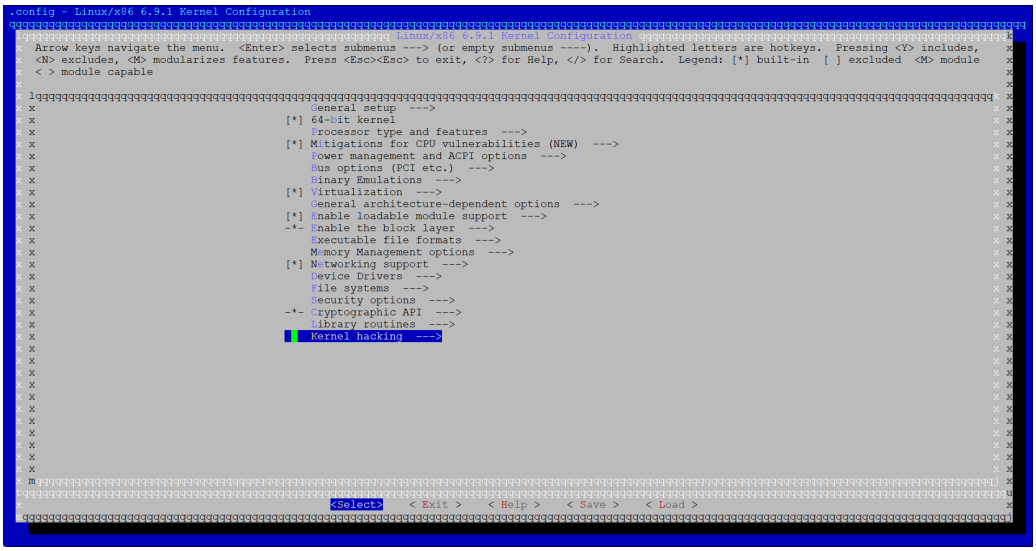


9) After saving the configuration, you should see a window saying the configuration is written to the filename you specified previously:

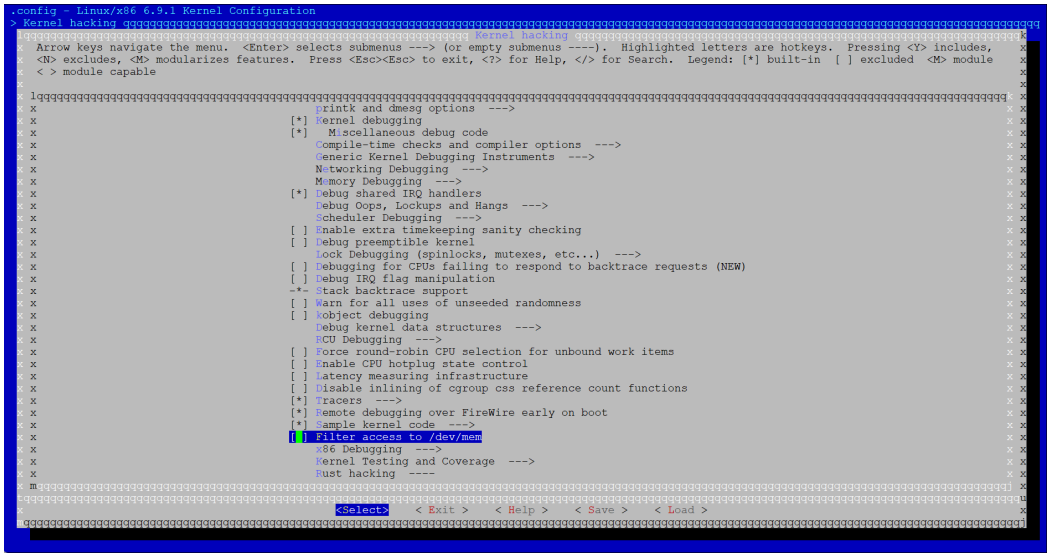


10) Return to the Top Menu by pressing <Esc>

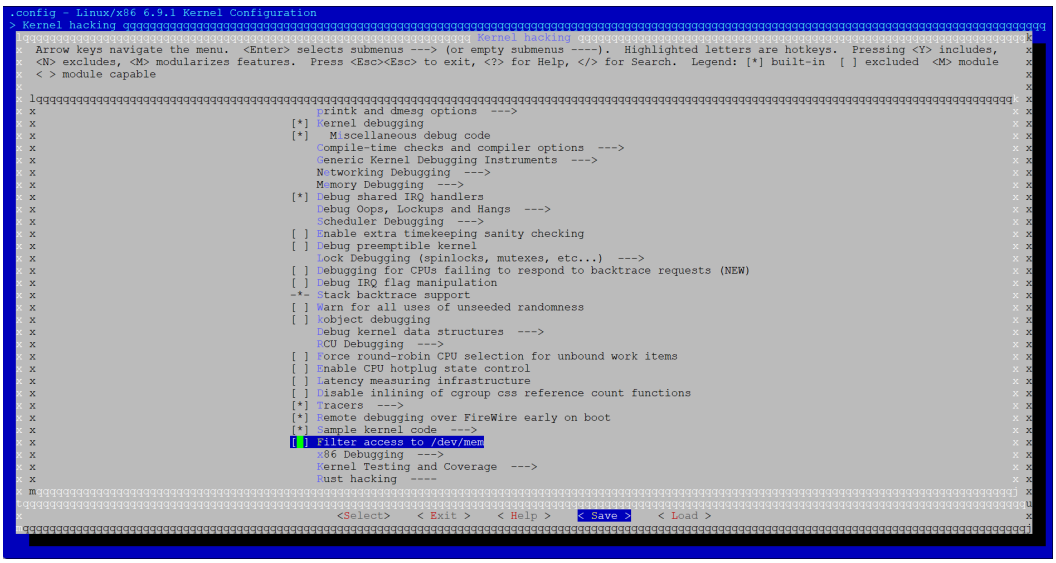
11) Navigate to the “Kernel hacking” section and press <Enter>:



12) Press <Space> to deselect “Filter access to /dev/mem” and “Filter I/O access to /dev/mem”:

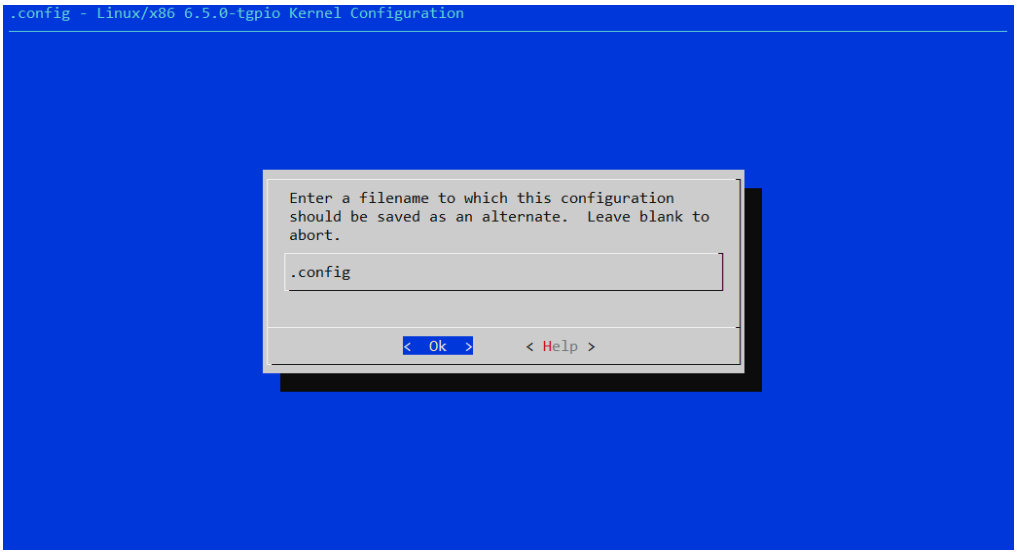


13) Save the configuration by navigate to <Save> and press <Enter>:

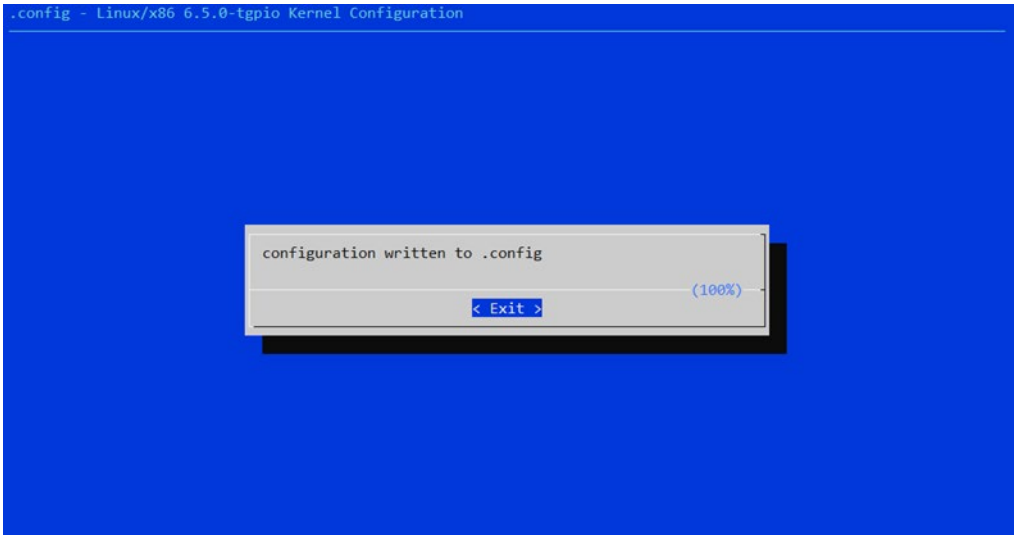


14) Then, enter the filename to which this configuration should be saved chose press <Enter> to choose “Ok”:

In this example, we just accepted “.config” which was the default name provided in the Kernel Configuration GUI:



15) After saving the configuration, you should see a window saying the configuration is written to the filename you specified previously:



16) Return to the Top Menu by pressing <Esc>

17) Exit the Kernel Configuration GUI by pressing <Esc>.

18) Once you are in your terminal, open the Makefile:

```
vi Makefile
```

19) Set the “EXTRAVERSION” field in the Makefile to “-tgpio”. This is important to make sure the new modules and references won’t overwrite those for the compiling machine. The outcome should look like this:
EXTRAVERSION = -tgpio

20) Save:
<Esc> <:wq> <Enter>

21) Open the configuration file that was created previously in step number 9:

```
vi .config
```

22) Comment the “CONFIG_SYSTEM_TRUSTED_KEYS” and the “CONFIG_MODULE_SIG_KEY” fields in the .config file.

The outcome should look similar to the following:
#
Certificates for signature checking
#
#CONFIG_MODULE_SIG_KEY=”certs/signing_key.pem”
CONFIG_MODULE_SIG_KEY_TYPE_RSA=y
#CONFIG_MODULE_SIG_KEY_TYPE_ECDSA is not set
CONFIG_SYSTEM_TRUSTED_KEYRING=y
#CONFIG_SYSTEM_TRUSTED_KEYS=”certs/rhel.pem”

23) Save the changes:

<Esc> <:wq> <Enter>

C. Kernel Compilation and Installation

1) For RHEL 9.x, the GCC may create invalid configuration entries in the Intel E810 driver.

A file needs to be checked for this error and if the error exists, modify the file:

```
vi drivers/net/ethernet/intel/ice/ice_ptp.c
```

The portions highlighted in **red** must be deleted.

Before Modification

```
static const struct ptp_pin_desc ice_pin_desc_e810t[] = {
/* nameidxfunc chan */
{ "GNSS",  GNSS, PTP_PF_EXTTS, 0, { 0, } },
{ "SMA1",  SMA1, PTP_PF_NONE, 1, { 0, } },
{ "U.FL1", UFL1, PTP_PF_NONE, 1, { 0, } },
{ "SMA2",  SMA2, PTP_PF_NONE, 2, { 0, } },
{ "U.FL2", UFL2, PTP_PF_NONE, 2, { 0, } },
};
```

After Modification

```
static const struct ptp_pin_desc ice_pin_desc_e810t[] = {
/* nameidxfunc chan */
{ "GNSS",  GNSS, PTP_PF_EXTTS, 0, 0, },
{ "SMA1",  SMA1, PTP_PF_NONE, 1, 0, },
{ "U.FL1", UFL1, PTP_PF_NONE, 1, 0, },
```

```
    { "SMA2",   SMA2, PTP_PF_NONE, 2, 0, },
    { "U.FL2", UFL2, PTP_PF_NONE, 2, 0, },
    };
```

2) Save the file changes:

   <Esc> <:wq> <Enter>

3) Compile the kernel:

```
make
```

   Note: Click through **Enter** any generated request/response queries.

   If the kernel compilation is successful, you should see something similar with the following example printed towards the end of the compilation.

   Example:
   Kernel: arch/x86/boot/bzImage is ready (#3)

   If you do not see the above message:

```
make
```

4) Install the modules:

```
sudo make modules_install
```

```
sudo make install
```

5) Verify the default kernel number:

```
sudo grubby --default-title
```

   If the default kernel is not the custom kernel that we created, do the following:

   a) Get the list of kernels and index number:

```
sudo grubby --info=ALL | grep -E "^kernel|^index"
```

      Example output:
      index=0
      kernel="/boot/vmlinuz-6.5.0-tgpio+"
      index=1
      kernel="/boot/vmlinuz-5.14.0-362.18.1.el9_3.x86_64"
      index=2
      kernel="/boot/vmlinuz-5.14.0-362.8.1.el9_3.x86_64"
      index=3
      kernel="/boot/vmlinuz-0-rescue-7e3d4d8cb1894417905f10874ae64417"

Note: please remember the index of the TGPIO kernel. In this example, it is 0, which is highlighted in green.

b) Change the default kernel to the custom kernel, do this:

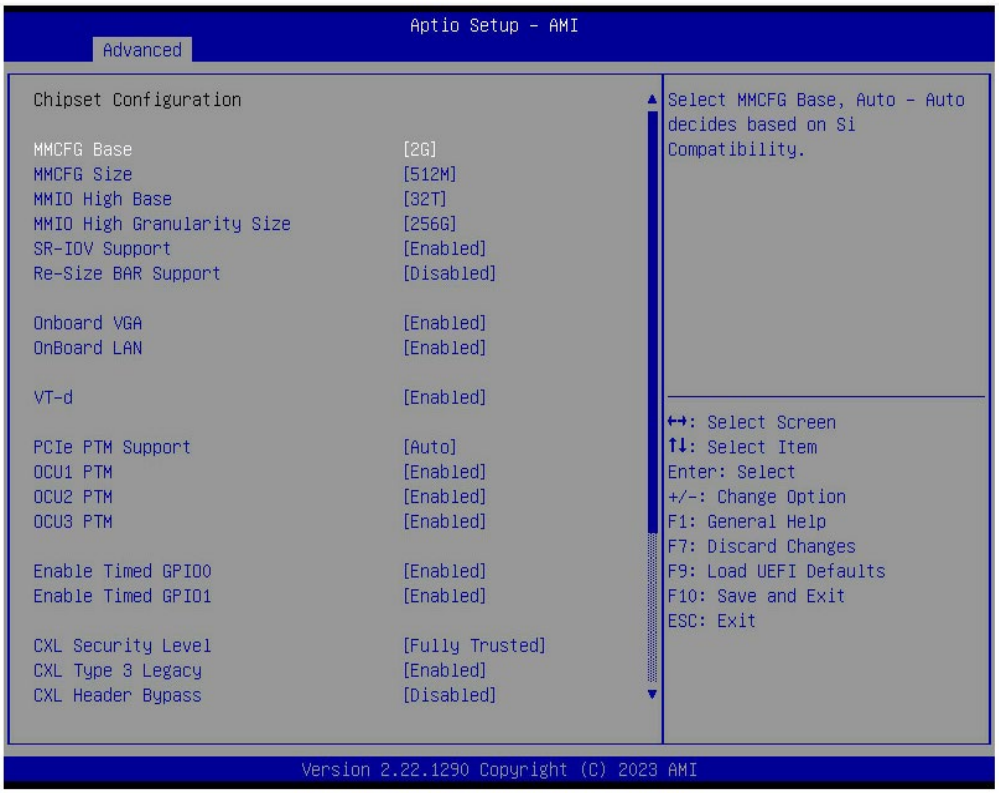
```
sudo grubby --set-default-index=0
```

6) Reboot the server to reflect the changes.

```
sudo reboot
```

D. BIOS Enablement

1) When the server reboots, open the BIOS, and make sure to enable the Timed GPIO settings. The expected BIOS settings should look like the following example:



Note: Your BIOS menu structure may differ.

2) Save the settings and exit the BIOS.

3) Once the server boots successfully, verify the updated kernel is running:

```
uname -r
```

Expected output:
6.9.1-tgpio+

E. Installation of TGPIO PPS Software Dependencies

1) Install the quadmath development library:

```
sudo dnf -y install libquadmath-devel
```

2) Install the “cpuid” package. Since this package is unavailable under RHEL9’s RPM, we would need to import the EPEL release RPM.

```
sudo subscription-manager repos --enable codeready-builder-for-rhel-9-$(arch)-rpms
```

```
sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm -y
```

```
sudo dnf install cpuid -y
```

3) Get the file path for the TGPIO driver:

```
grep -H "Intel PMC TGPIO" /sys/class/ptp/ptp*/clock_name
```

Example Output:
/sys/class/ptp/ptp0/clock_name:Intel PMC TGPIO

From the example above, the Intel TGPIO is in */dev/ptp0* directory. Make sure to remember this path since we will be using it in the following steps.

4) Determine the ratio and frequency of your system which will be used to enable the TGPIO PPS:

```
cpuid -1 | grep -E "(Time Stamp|TSC/clock|nominal)"
```

Example output:
Time Stamp Counter/Core Crystal Clock Information (0x15):
TSC/clock ratio = 160/2
nominal core crystal clock = 25000000 Hz

Note: Please remember the values highlighted in green above since we will be using these values later.

5) Navigate to the TGPIO PPS directory of the Time Aware GPIO Support directory:

```
cd kernel/Time-Aware-GPIO-Support/user-code/tgpio_pps
```

6) Open the pps.c file:

```
vi pps.c
```

7) Modify the “ART_FREQUENCY”, “ART_NUM” and “ART_DEN” fields in the pps.c file based on the values we captured previously.

For example, using the values obtained previously in step 4:

#define ART_FREQUENCY (25000000 /*cycles per second*/)
#define ART_NUM (160)
#define ART_DEN (2)

Note: "ART_FREQUENCY" should reflect the "nominal core crystal clock" field. "ART_NUM" and "ART_DEN" should reflect the first and second values of the "TSC/clock ratio" field, respectively.

8) Install the TGPIO PPS:

```
make
```

F. Installation Verification

1) Run PPS by specifying the TGPIO driver path, which was captured previously in section C, step number 3:

```
sudo ./pps -d /dev/ptp0
```

Example output:
1708918996.000(ART=42447719196.8)
1708918996.500(ART=42460219340.-12)
1708918997.000(ART=42472719484.-16)
1708918997.500(ART=42485219627.-14)
1708918998.000(ART=42497719770.7)

Output explanation:

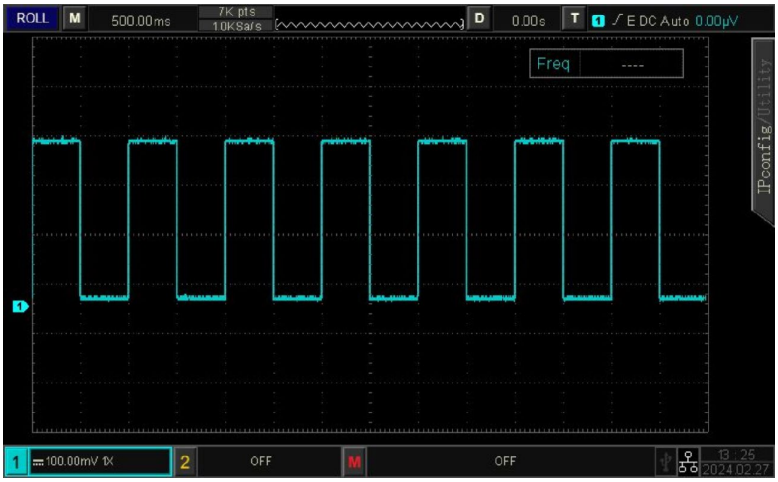
1708918996.000	: current time in milliseconds
ART=42447719196	: corresponding ART (Always Running Timer) value
.	: separator between ART value and residue value
8	: residue value in nanoseconds (since ART is running at 25MHz, there may be up to a 40ns max delta)

2) Using the corresponding ART value captured in the PPS output, we can use it to verify the PPS waveform that is seen on the oscilloscope. At a second, the PPS signal goes up and at half of a second, the PPS signal should go down.

For example:
1708919003.000(ART=42622721205.12)
At a second, the pps signal rises (this can be verified using an oscilloscope)

1708919003.500(ART=42635221349.-1)
At a half a second, the pps signal falls (this can be verified using an oscilloscope)

Example waveform:



25. Troubleshooting

The following section outlines common issues and how to address them.

cxl_pci module is not not loaded after running install.sh script	
Possible Cause	**Resolution**
Server/Motherboard used is not CXL-compliant	Use a server/motherboard which supports CXL. See the Server and Motherboard section for more details.
CXL is not enabled in the BIOS	Reboot the system and make sure to enable the BIOS with the recommended values as highlighted in the BIOS Enablement section.
Using an OS where CXL is not supported	Reinstall the system with an OS that supports CXL. It is recommended to use the latest OS version where CXL is available natively, such as RHEL 9. See the Operating System section for more details.

Cannot open dax memory service	
Possible Cause	**Resolution**
DAX is not running and correctly permissioned	Verify if the directory /dev/dax0.0 is available and set permissions for device_dax facility so that it is readable, writeable, and executable by all users. See the CXL Enablement in ÜberNIC section for more details.
Using an OS where CXL is not supported	Reinstall the system with an OS that supports CXL. It is recommended to use the latest OS version where CXL is available natively, such as RHEL 9. See the Operating System section for more details.
Using an old software driver that tries to find 'dax memory service' by default	Get the latest software driver release and reinstall the software logic.

lspci fails to identify ÜberNIC	
Possible Cause	**Resolution**
No priority was set for the objects programmed on ÜberNIC	Read the ÜberBMC section in this user guide or the other ÜberNIC board management and logic User Guide for more details on how to set the programmed object's priority. Make sure to reboot your system once you finish setting up the object's priority. If the problem persists after setting the priority of the objects, please contact LMS for assistance.
The system might not be fast enough to enumerate and detect that there's a new device inserted	Try doing a cold reboot of the server, during which the server must be physically disconnected from *all* power sources, to see if ÜberNIC is now detected using lspci.

ÜberLoad is not working as expected	
Possible Cause	**Resolution**

Using ÜberLoad with an application that is statically linked (to the C library, e.g. glibc).	Link the application to the C library dynamically, and rerun ÜberLoad.

--- END ---