MLE Technical Brief 20260228
Network Protocol Accelerator Platform

Datasheet for a stand-alone TCP/UDP/IPv4 Stack Full-Accelerator Subsystem allowing communication at full line rate and low latency

Version: 2.9.0
Date: February 2026
Authors: Ulrich Langenbach, Thomas Glatte, Jakob Arndt, Oskar Szakinnis

1. Features

MLE's Network Protocol Accelerator Platform (NPAP) is a TCP/UDP/IPv4 network protocol Full-Accelerator for Multi-Gigabit Ethernet which instantiates the standalone 128 bit TCP/UDP/IPv4 Stack technology from the German Fraunhofer Heinrich-Hertz-Institute (HHI). With a focus on reliability and low, deterministic latency, HHI designed this for embeddable FPGA and ASIC systems with the following features:

- Interface to 1 / 2.5 / 5 / 10 / 25 / 40 / 50 / 100 / 200 / 400 Gigabit Ethernet[1]
- Full-duplex with 128 bit wide bidirectional datapath
- Full line rate of 70 Gbps, or more, per instance in FPGA
- Full line rate of over 100 Gbps per instance in ASIC
- Low one-way latency NPAP-to-NPAP (600 nanoseconds for 100 Bytes)
- Network diagnostics functions (optional)
- TCP session priority management (optional)
- Transport Layer Security (TLS) (optional)
- Time-Sensitive Networking (TSN) (optional)
- Network Impairment Generators (optional)

Designed for maximum flexibility, NPAP implements in programmable logic the relevant network communication protocols:

- **IPv4** The core of the most standards-based networking protocols
- **TCP** Reliable connectivity for direct secured connectivity
- **UDP** Widespread protocol to enable simple direct or multicast communication
- **RRRRP** Reliable, Rapid Request-Response Protocol based on Stanford HOMA[2]
- **ICMPv4** Diagnostic protocol to validate connections
- **IGMPv4** Enables joining of multicast groups (optional)

Due to the modularity NPAP can easily be enhanced by application specific protocols.

The 128 bit wide datapath in combination with a pipelined architecture allows to scale throughput to line-rates of 50 GbE, and beyond, when using modern FPGA fabric, and up to 100 Gbps for ASIC implementations. NPAP is available in versions which recently have been merged:

- Version 2 (currently 2.9.0) for new ASIC and AMD/Xilinx Versal, Ultrascale+, Ultrascale, Altera/Intel Agilex-7, Agilex-5E and Stratix-10, Microchip PolarFire and Lattice Avant G/X development, includes many recent resource optimizations plus timing optimizations focused on pipelines in FPGA, including Altera/Intel HyperFlex and Intel HyperFlex2
- Version 1 (currently 1.10.1) back-ports bug fixes from Version 2 for long-term customer support. **Not recommended for new design starts!**

[1] Obviously depending on the FPGA device and speedgrade.
[2] https://missinglinkelectronics.com/resources/technical-publications/how-bad-is-tcp-and-what-are-the-alternatives/

2. Applications

NPAP enhances your networked application with fast, scalable and reliable data connectivity. The powerful architecture of the underlying TCP/UDP/IPv4 Stack allows it to transfer data at line-rate with low processing latency without using any CPUs in the data path. The ubiquitous TCP/IPv4 and/or UDP/IPv4 communication protocol suite uses industry standard network infrastructure to address a wide-range of applications:

- High-Speed connectivity for distributed systems and Systems-of-Systems
- Scale-out datacenter connectivity
- Reliable, long-range chip-to-chip connectivity with backpressure
- FPGA-based SmartNICs
- High-Bandwidth Security with FPGA-based Smart Data Diodes
- In-Network Compute Acceleration (INCA)
- Hardware-only implementation of TCP/IPv4 in FPGA
- PCIe Long Range Extension [3]
- Networked storage, such as iSCSI or NVMe/TCP
- Test & Measurement connectivity
- Automotive backbone connectivity based on open standards
- High-speed, low-latency camera interfaces
- Video-over-IP for 3G / 6G / 12G transports
- Bring full TCP/UDP/IPv4 connectivity to FPGAs
- High-speed sensor data acquisition:
  stream data out of FPGAs into Network-Attached Storage (NAS)
- High-speed robotics control and machine-to-machine:
  Stream data from servers via FPGA into actuators
- Hyper-converged computational storage acceleration for “over-Fabric” NVMe/TCP
- Deterministic low-latency, high-bandwidth, secure alternative to lwIP or Linux on embedded CPU

[3] https://www.missinglinkelectronics.com/index.php/menu-products/menu-pcie-connectivity/439-art-pcie-over-xxx

3. NPAP Functionality Description

NPAP is a complete subsystem of a high-performance programmable-logic based, standalone network stack featuring transparent handling of complete TCP/IPv4 and UDP/IPv4 protocol tasks, e.g. packet encoding, packet decoding, acknowledge generation, link supervision, timeout detection, retransmissions and fault recovery.

NPAP supports complete automatic connection control including tear up and tear down. Compute and manage retransmission timers as in RFC 6298[4]. Transparent checksum generation and checksum checking, integrated flow control. RFC 9293[5] compatibility (TCP/IPv4 stack for Windows and Linux).

Depending on the project's needs, deliverables can be:

- HDL source code or netlist
- Integrated FPGA system implementation
- Testbenches and scripts for real-life testing
- Comprehensive documentation and interfacing guide
- Development & design-in support

NPAP has been optimized to ensure the best bandwidth-delay product performance for your application. To guarantee delivery of full performance and reliability Team MLE will support you in all engineering aspects:

1. **System-level architecture design** where aspects such as mapping ingress / egress data streams to TCP sessions, or handling TCP's congestion control, or optimizing the bandwidth-delay-product are handled in order to meet system-level bandwidth and latency requirements.

2. **Chip-design**, i.e. managing chip resources, interfacing with the Multi-Gigabit Transceivers (MGT), handling on-chip streaming (such as AXI beats) while integrating NPAP into your FPGA or ASIC device on your target hardware.

3. **Network administration** which includes configuring TCP/UDP servers and clients as well as planning and administering the MAC and IPv4 addresses within each system as well as across all shipped units.

4. **System verification**, testing, diagnosing and debugging throughout your product's lifecycle: From RTL simulation of NPAP integrated within your FPGA or ASIC over first engineering samples up to network diagnostics for products installed in the field.

This datasheet aims to educate system architects, chip designers, verification engineers, product management and technical product support on how to plan, manage and make best use of MLE NPAP.

[4] https://www.rfc-editor.org/info/rfc6298
[5] https://www.rfc-editor.org/info/rfc9293

NPAP follows a modular, parameterized implementation in VHDL. Modular means that blocks of functionality have been implemented by hierarchical VHDL modules. Some of these modules can be instantiated multiple times - for example a TCP core which handles a so-called TCP connection - or not at all, when, for example, you may not need any TCP protocol support at all and wish to save FPGA resources.

NPAP can be tuned and optimized for your target application via around 100 parameters, some of which are

- are **compile-time parameterizable**, mostly via VHDL generics,
- others are **runtime parameterizable**, typically via NPAP HAL - the Hardware Abstraction Layer between AXI Lite registers and Linux device drivers or Python scripts,
- some are both, where you can set defaults at compile-time with options for change during runtime.

Therefore, MLE NPAP comes with extensive documentation on how to architect, integrate, test, administer, and run this TCP/UDP/IPv4 Full Accelerator.

3.1. Technical Features

Feature	Specification
Supported on-chip Interfaces	128 bit wide AXI4-Stream
Compatibility with 3rd party Ethernet PHY interfaces	Standard IEEE Ethernet PHYs with RMII, GMII, XGMII, etc via PCS/PMA via ASIC/FPGA Ethernet Subsystem
Compatibility with 3rd party Ethernet Media Access Controllers	Fraunhofer HHI 10G/25G Low-Latency MAC AMD/Xilinx 10G/25G Ethernet Subsystem (PG210) AMD/Xilinx 100G Ethernet Subsystem (PG165, PG203, PG314) Altera 10G / 25G Ethernet FPGA IP Microchip PolarFire FPGA 10G Ethernet (UG0727) Lattice 10G / 25G Ethernet IP (FPGA-IPUG-02245)
Supported protocols (Hardware based)	Ethernet, ARP, IPv4, ICMPv4 (response only), IGMPv4, UDP & TCP, DHCP (client only)
Number of simultaneous connections	One per TCP Core instantiation - see “Architecture Choices” below, a TCP Core in NPAP relates to a TCP socket in Linux
Message Sizes	Support for Ethernet Jumbo Frames of arbitrary length
Interface to application	Datapath via AXI4-Stream 128-bit and separate custom TCP command interface
Supported FPGAs	Complete stack uses generic VHDL code (IEEE-1076 2002 or 2008, depending on NPAP version) AMD/Xilinx Virtex 4 to Virtex UltraScale+ AMD/Xilinx Kintex to Kintex UltraScale+ AMD/Xilinx Artix UltraScale+ AMD/Xilinx Zynq-7000 AMD/Xilinx Zynq UltraScale+ MPSoC AMD/Xilinx Zynq UltraScale+ RFSoC AMD/Xilinx Versal ACAP Series Altera Cyclone IV series Altera Cyclone 10 GX series Altera Stratix V Altera Stratix 10 GX series Altera Agilex 5 D, E Series Altera Agilex 7 F, I, M Series Lattice Avant-G, Avant-X Microchip Polarfire and PolarFire SoC

Performance	70 Gbps line rate, or more, for single TCP/IPv4 session (depending on clock rate, see below) Typ. 600 ns transport delay (depending on clock rate and packet size)
RTL Verification	A full RTL simulation environment, including support for Wireshark PCAP stimulus files can be made available upon special request
Diagnostics & Debug	RTL code plus instrumentations for extensive statistics and diagnostics can be made available upon special request Optional functionality for Network Impairment (a.k.a. Bit Error Insertion) can be made available upon special request

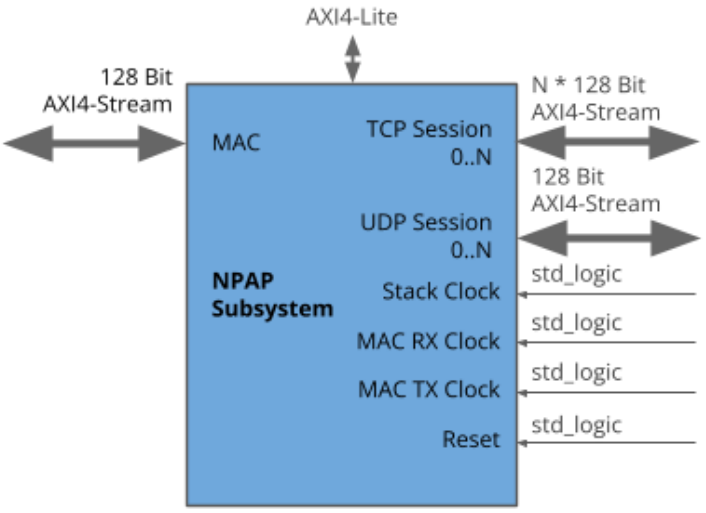
3.2. Implementation Details

NPAP implements a full accelerator, hence **all network protocol processing is running as digital logic**. Because NPAP does not rely on "soft" CPUs nor on external CPUs, NPAP shows very low and deterministic latency. Tradeoff cost vs performance over chip resources.

- Dataflow is full duplex 128 bits wide using AXI4 Stream. This enables high data throughput without "FPGA bloat" nor timing issues.
- Control-flow uses AXI4 Lite register interfaces, along with a hardware abstraction layer (HAL), Linux device drivers and Python scripts for NPAP administration.
- NPAP is intensively tested for performance and interoperability against many other TCP/UDP/IPv4 network stacks.
- NPAP brings its own 10G / 25G Low-Latency Ethernet MAC,
  but can interface with many Ethernet subsystems from the FPGA vendors.
- NPAP implements a complete TCP/UDP/IPv4 stack including functions like ARP, ICMPv4, IGMPv4, DHCP.
- NPAP is delivered with "Support IP blocks" including reference designs, design examples for setting MAC addresses and/or IPv4 addresses and/or TCP port numbers either from Programmable Logic / ASIC or via software running on a (Linux) host, either ARM or x86 based.
- For each TCP connection that remains open at the same time, there shall be a dedicated instance of a TCP Core. An AXI4-Lite interface may be used to prioritize TCP sessions during runtime.
- One and only one single instance of a UDP Core must be instantiated when UDP support is required. If there is no need to process UDP, then the UDP Core can be removed completely.
- NPAP is highly parameterizable to optimize for lowest FPGA resource needs while still delivering full functionality and performance: Number of UDP Cores (zero or one), number of TCP Cores (zero or many), for each TCP Core: Rx Buffer Size and, separately, Tx Buffer Size
- Other than Rx and Tx buffers and some buffers for clock domain crossings, NPAP hardly uses any buffers at all, which results in very low and deterministic latency.
- Your "Layer 7" Application can directly be connected to NPAP via AXI4 Stream which gives you the option of keeping all traffic inside the Programmable Logic / ASIC, and/or to interface with "software" running on (Linux) host, either ARM based or x86 based, via DMA.

A hierarchical design philosophy is used to integrate MLE NPAP together with the auxiliary blocks. Related blocks are packaged together in a "support wrapper". These are nested and joined within a top-level wrapper. Please refer to the NPAP Developer Documentation below.

At the highest level, a single NPAP based subsystem with NPAP Support IPs but with external MAC looks like this:



Throughout this datasheet and our documentation we use the following color coding legend for certain blocks of functionality:

	AXI4-Lite Control / Status Register Access
	128 bit AXI4-Stream bi-directional
Netperf	Netperf Latency and Bandwidth Benchmarking
Ex. App	Example Applications for TCP / UDP incl. test patterns
BERT	Network Impairment and Bit Error Rate Tests (BERT)
diag	Diagnostics Counters
NPAP	NPAP TCP and UDP Layer
NPAP	NPAP IPv4 Layer with ICMP and IGMP
NPAP	NPAP Ethernet Link Layer
MAC	Ethernet MAC (from MLE or from FPGA Vendor)
PCS/PMA	Ethernet PCS/PMA (from FPGA Vendor)

Besides the number of TCP Cores, NPAP is highly parameterizable: Some parameters can be set at Compile-Time, others at Runtime, and some both ways. Please see below.

3.2.1. Compile-Time Parameters

NPAP makes use of VHDL Generics to parameterize certain functionality as well as the NPAP design structure (such as the number of TCP cores) at compile-time.

Here some examples:

Parameter Name	NPAP Block	Design Category	Documented in
ERD System Clock Period	Infrastructure	Chip Design	Product Guide: NPAP Kernel
NPAP ARP Cache Size	Kernel	NW Admin	Product Guide: NPAP Kernel
NPAP Maximum Frame Size	Kernel	NW Admin	Product Guide: NPAP Kernel
Number of TCP Sessions	Kernel	System-Level	Product Guide: NPAP Kernel
TCP TX Buffer Size	Kernel	NW Admin	Product Guide: NPAP Kernel
TCP RX Buffer Size	Kernel	NW Admin	Product Guide: NPAP Kernel
TCP TX Maximum Segment Size	Kernel	NW Admin	Product Guide: NPAP Kernel
TCP RX Maximum Segment Size	Kernel	NW Admin	Product Guide: NPAP Kernel
TCP TX Maximum Frame Number	Kernel	NW Admin	Product Guide: NPAP Kernel
TCP Enable DGC	TCP Demo App	Testing and Debug	Product Guide: NPAP Kernel
UDP Enable	Kernel	System-Level	Product Guide: NPAP Kernel
UDP TX Buffer Size	Kernel	NW Admin	Product Guide: NPAP Kernel
…	…	…	…

A complete list of all Compile-Time Parameters and their use is available upon request. All are fully documented in the NPAP Developer Documentation.

3.2.2. Runtime Parameters

NPAP behavior can be controlled at runtime via the Runtime Parameters. Typically, these are implemented via AXI-Lite registers. The complete set of all relevant registers is exported via a hardware abstraction layer (NPAP HAL) which then can be altered at runtime via the NPAP Administration & Control-Flow.

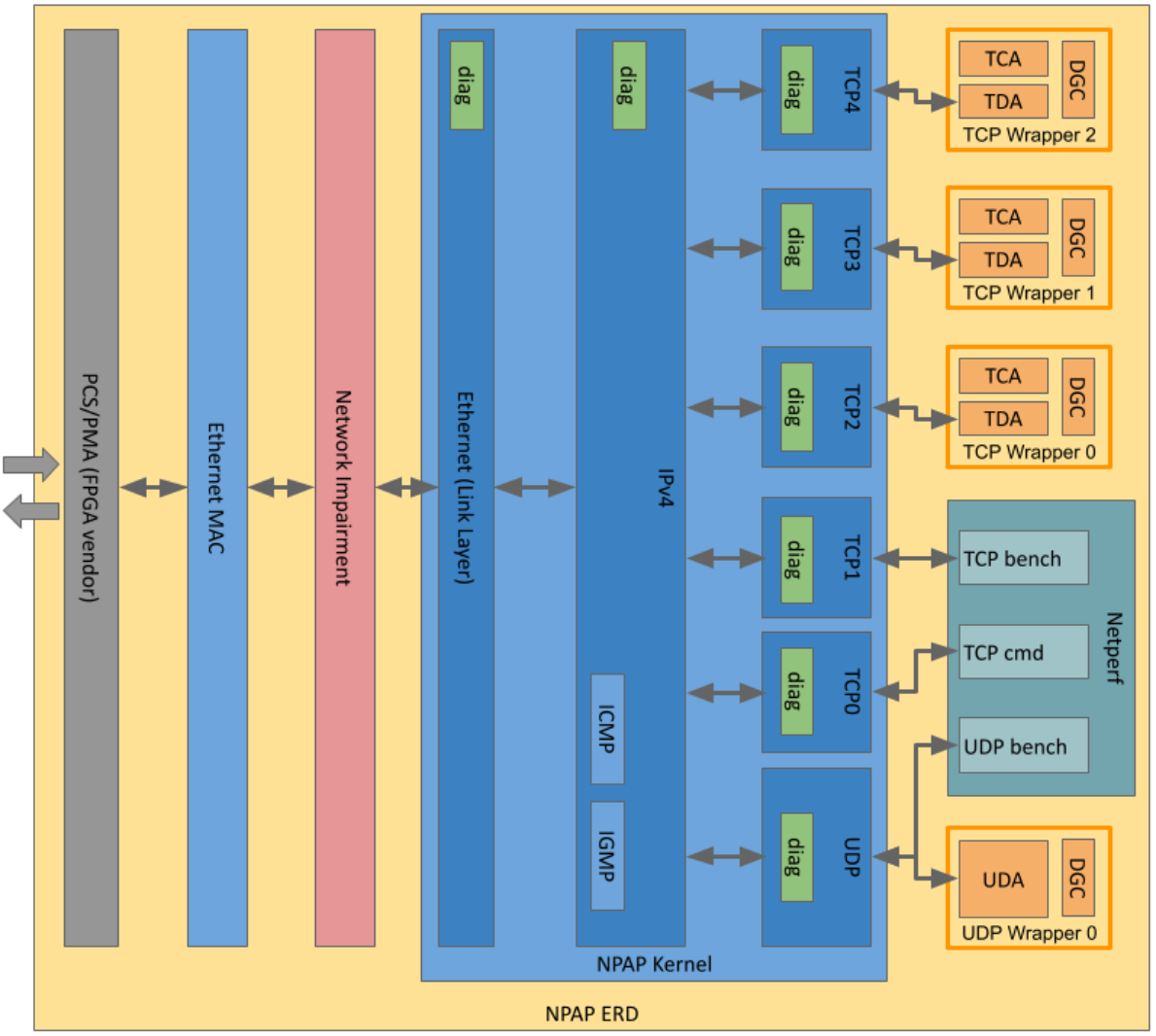
Here are some example run-time parameter:

Parameter Name	NPAP Block	Design Category	Documented in
NPAP Gateway IP	Kernel	NW Admin	Product Guide: NPAP Kernel
NPAP Local IP	Kernel	NW Admin	Product Guide: NPAP Kernel
MAC Address	Kernel, MAC	NW Admin	Product Guide: NPAP Kernel
NPAP Subnet Mask	Kernel	NW Admin	Product Guide: NPAP Kernel
TCP Session Data Processing	TCP Demo App	System-Level	Product Guide: TCP Demo App
TCP Session Metadata Processing	TCP Demo App	System-Level	Product Guide: TCP Demo App
TCP Session Loopback Mode	TCP Demo App	System-Level	Product Guide: TCP Demo App
TCP Session Local Port	TCP Demo App	NW Admin	Product Guide: TCP Demo App
TCP Session Remote IP	TCP Demo App	NW Admin	Product Guide: TCP Demo App
…	…	…	…

A list of all Runtime Parameters and their use is available upon request. All are fully documented in the NPAP Developer Documentation.

3.3. NPAP Dataflow and Block Diagram

The following shows the dataflow view of an exemplary design integrating NPAP with one UDP core and multiple TCP cores (3 for user-level plus 2 for Netperf), each with an example user application, plus Netperf (for bandwidth and latency benchmarking), plus network impairment (for Bit Error Rate Testing), plus diagnostics counters:



The example user applications serve as an example on how to send and/or receive data from programmable logic via TCP/UDP/IPv4. For TCP this logic is inside one (or more) **TCP Wrappers** which contain HDL code for handling the control and data flow:

- TCA - the TCP Command Application to open/close a TCP connection
- TDA - the TCP Demo Application which uses the TCA to control the TCP session and can forward data to and from external applications such as the DGC
- DGC - a Data Generator and Checker which can generate payload data for sending and at the same time can check received payload data

Similarly, for UDP this logic is inside the one (or more) **UDP Wrappers** which contain HDL code for handling the control and data flow:

- UDA - the UDP Demo Application which handles the control and data flow for one UDP port
- DGC - a Data Generator and Checker which can generate payload data for sending and at the same time can check received payload data

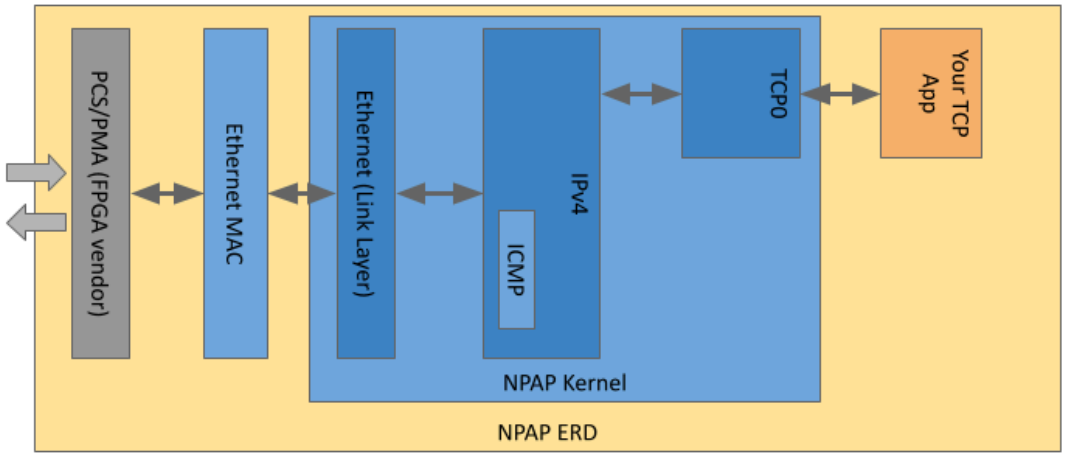
Optional **diagnostic counters** facilitate testing and tuning the quality of your system connectivity, including effects of signal quality issues in the physical layer. Please refer to the Product Guide of the NPAP Kernel for more details.

An optional **Network Impairment** block which sits between the Ethernet Link Layer and the Ethernet MAC mimics signal integrity issues by invalidating Ethernet packets based on an emulated bit error rate.

An optional **Netperf** block is a fully accelerated version of open source Netperf Version 2.6[6] which allows you to perform system-level performance analysis and benchmarking of bandwidth and latency / Round-Trip Times.

Obviously, all these optional blocks need extra FPGA resources, hence NPAP has options to integrate them for engineering-level lab testing, or for in-field diagnostics throughout your product's lifecycle (as needed sometimes), or for leaving them out.

For smaller FPGA devices, or when you need almost all FPGA resources for your own user logic, you can minimize resources by integrating a minimal NPAP with just one single TCP core, for example, along with your TCP application:

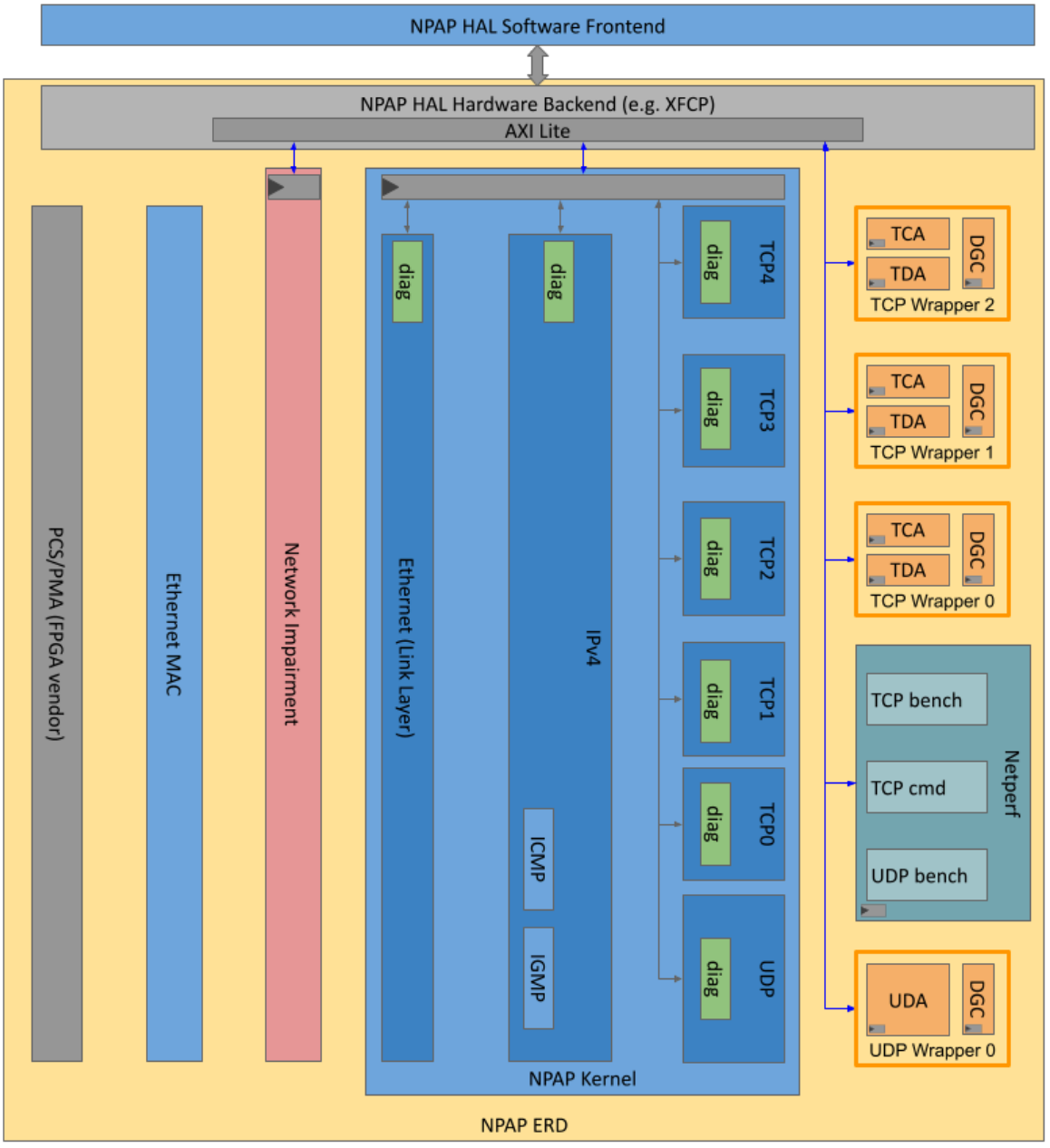


Good examples for the dataflow implementation, parameterization and integration of NPAP are the so-called NPAP Evaluation Reference Designs (ERD) running on many off-the-shelf FPGA platforms. Please read below for more information.

[6] https://github.com/HewlettPackard/netperf

3.4. NPAP Control-Flow View and Hardware Abstraction Layer

For administration and control at run-time, NPAP implements so-called Runtime Parameterization and administration via an AXI-Lite register space. Access to this interface is exported through the so-called NPAP Hardware Abstraction Layer (HAL). Along with a Python library, NPAP HAL provides a high-level API for Linux software, utilizing swappable backends to communicate with the hardware across diverse environments, from SOC processing systems to remote workstations.



Here a list of connectivity choices for NPAP HAL:

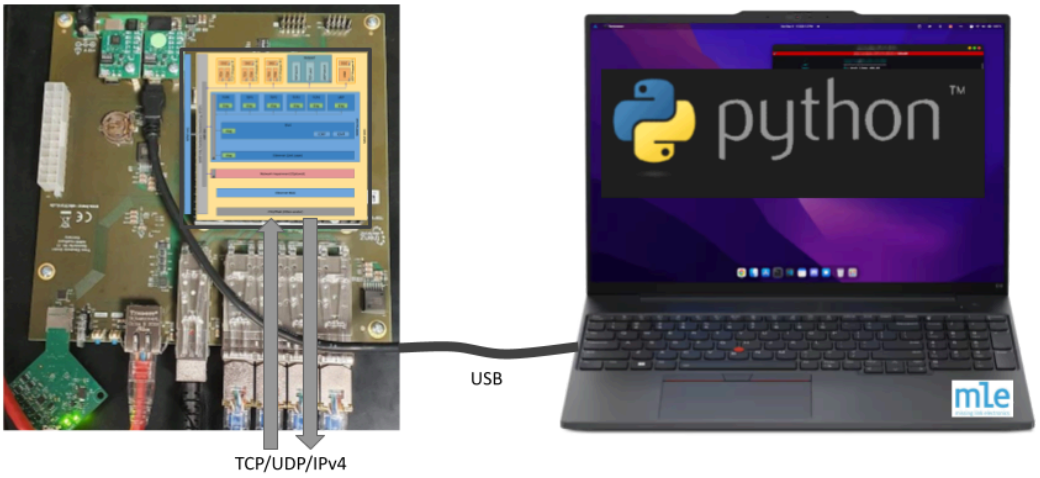
1. Via USB-UART or USB-JTAG or USB-IIC
2. Via the FPGA-integrated Processing System which can be ARM, RISC-V, MicroBlaze, NIOS, etc
3. Via PCIe connection with the host CPU
4. Via out-of-band Ethernet and UDP (on the roadmap)
5. Via in-band Ethernet and UDP (on the roadmap)

Besides NPAP HAL, MLE further provides a python based command-line tool, called **npap-admin**, that abstracts complex runtime parameterization into simple configuration file editing. Customers have been using npap-admin during evaluation and development, for Continuous Integration or in-the-field when NPAP-based products have been deployed.

Good design examples for Runtime parameterization and administration of NPAP are the so-called NPAP Evaluation Reference Designs (ERD) running on many off-the-shelf FPGA platforms. Please read below for more information.

3.4.1. NPAP Admin via USB-UART or USB-JTAG

This mode of administration is very useful when using off-the-shelf FPGA development kits which feature a USB-to-UART connection which is then connected to a Linux workstation running Python.



In this setup, the NPAP HAL utilizes the Extensible FPGA Control Platform (XFCP[7]) to bridge the USB connection to the AXI4-Lite register space on the FPGA.

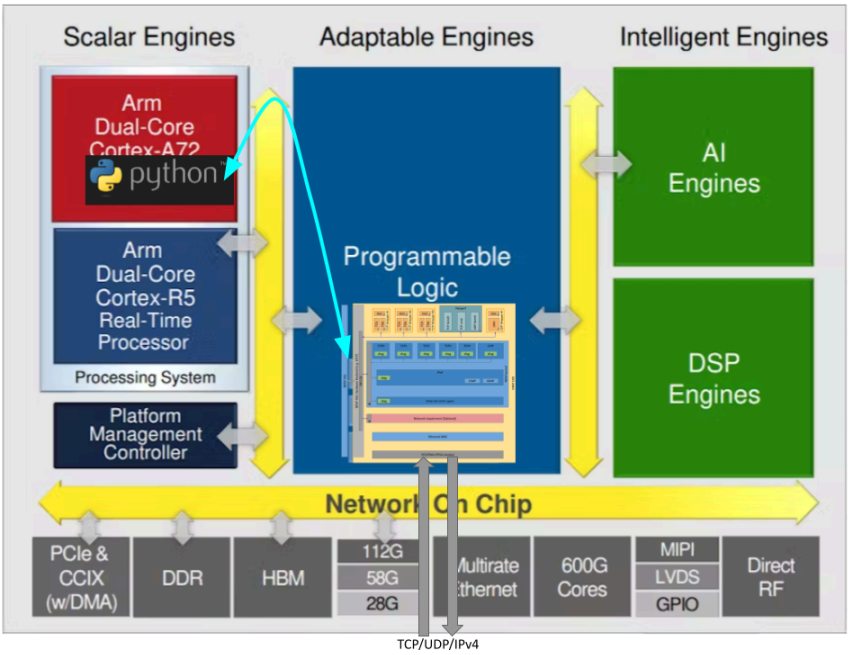
Here an outline of the connectivity stack:

NPAP Admin	
NPAP HAL with XFCP Backend	
XFCP	
Python Interpreter	
Linux Kernel	SW
USB to UART	
UART	HW
XFCP	
AXI-4 Lite	
NPAP ERD	

[7] https://github.com/alexforencich/xfcp

3.4.2. NPAP Admin via Embedded Processing System

This mode of administration is very useful when integrating NPAP into a SoC-FPGA with embedded CPUs (“hard” or “soft”) where an embedded CPU runs Python under Linux.



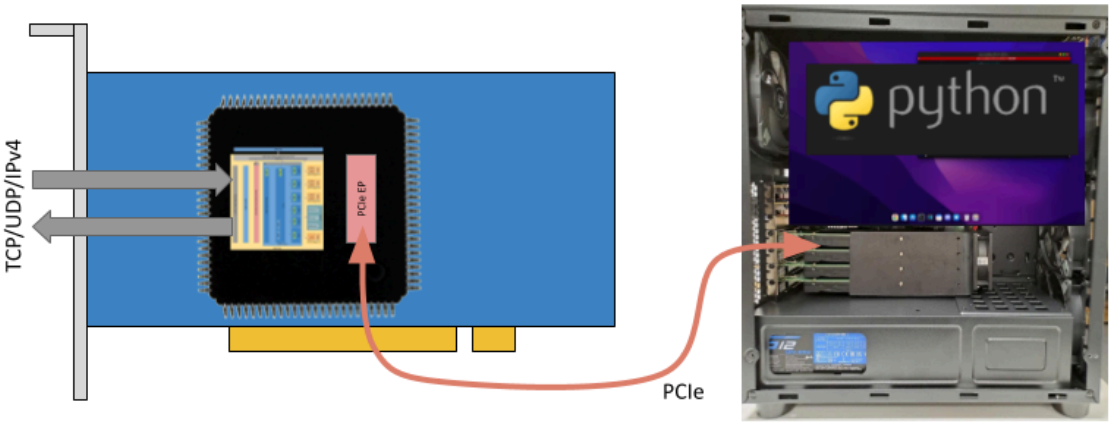
In this setup, the NPAP HAL interacts with the NPAP ERD registers by accessing sysfs[8] entries managed by a dedicated Linux driver. The connectivity stack, from the NPAP Admin CLI down through the Python HAL and the Linux kernel is outlined below:

NPAP Admin	
NPAP HAL with Sysfs backend	
Python Interpreter	
Sysfs	
Linux Driver	
Linux Kernel	PS
AXI-4 Lite	PL
NPAP ERD	

[8] https://docs.kernel.org/filesystems/sysfs.html

3.4.3. NPAP Admin via PCIe

This mode of administration is very useful when using off-the-shelf PCIe Cards where the FPGA is connected via PCIe to a Linux host CPU which then can run Python.



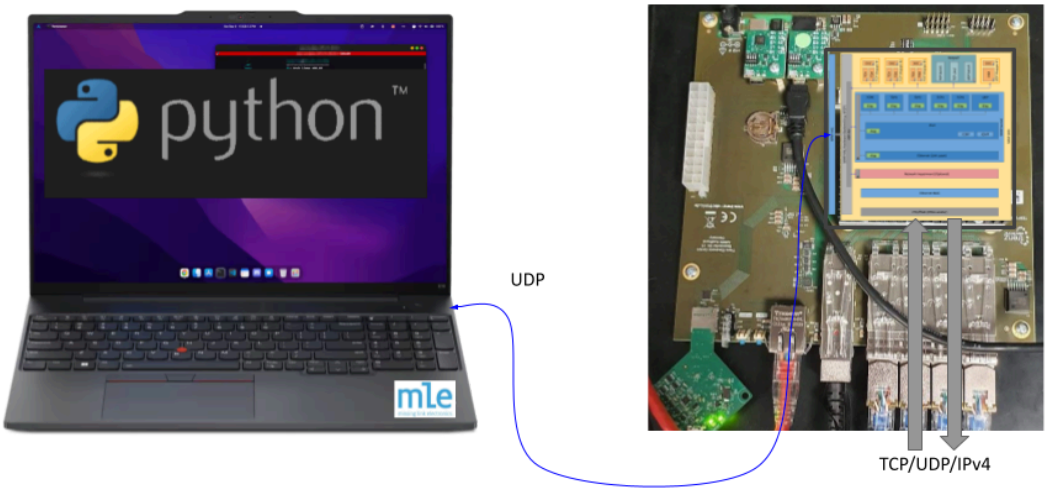
In this setup, the NPAP HAL uses its sysfs[9] backend to access the ERD's AXI4-Lite register space. The register space gets mapped into the Linux filesystem via the Linux PCIe driver subsystem. The connectivity stack is outlined below:

NPAP Admin	
NPAP HAL with Sysfs Backend	
Python Interpreter	
Sysfs	
PCIe Driver	
Linux Kernel	SW
AXI-4 Lite	HW
NPAP ERD	

[9] https://docs.kernel.org/filesystems/sysfs.html

3.4.4. NPAP Admin via Out-of-Band Ethernet / UDP

This mode of administration is very useful when using networked FPGA boards which feature a separate Ethernet connection (typically 1 GigE) which is then connected to a Linux workstation running Python.



In this setup, the NPAP HAL utilizes the Extensible FPGA Control Platform (XFCP[10]) to bridge the UDP Ethernet connection to the AXI4-Lite register space on the FPGA. The connectivity stack starting at npap-admin is outlined below:

NPAP Admin	
NPAP HAL with XFCP Backend	
XFCP	
Python Interpreter	
Linux Kernel	SW
UDP	
ETH	HW
XFCP	
AXI-4 Lite	
NPAP ERD	

[10] https://github.com/alexforencich/xfcp

3.5. NPAP Evaluation Choices

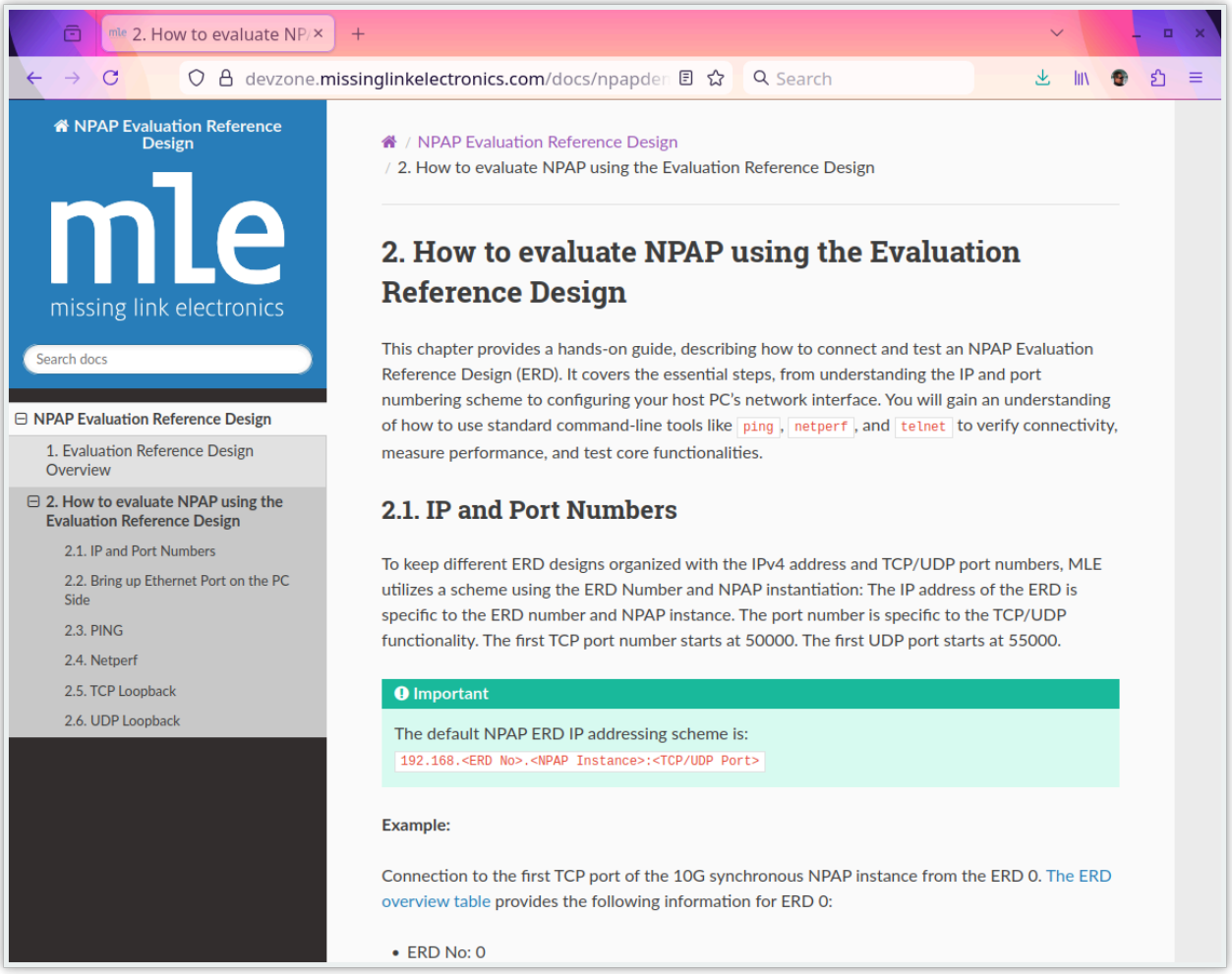
MLE offers multiple ways to evaluate and benchmark NPAP:

- Free-of-charge we provide so-called NPAP Evaluation Reference Designs (ERD) implemented on off-the-shelf hardware.
- A "Developers License" is a highly discounted extended evaluation license which gives you full source code access to integrate and run NPAP and NPAP ERDs within your target hardware.

For evaluating the functionality and the performance of NPAP MLE provides Evaluation Reference Designs (ERD) for several FPGA Development Kits:

ERD #	FPGA Board / Development Kit	Features
0	AMD/Xilinx ZCU102 with Zynq Ultrascale+ MPSoC ZU9EG	2x 1G
1	AMD/Xilinx ZCU102 with Zynq Ultrascale+ MPSoC ZU9EG	1x 10G
2	MLE's Network Protocol Accelerator Card "Ketch" with Altera Stratix 10 GX	2x 10G
3	AMD/Xilinx ZCU111 with Zynq Ultrascale+ RFSoC ZU28EG	1x 25G
5	AMD/Xilinx ZCU111 with Zynq Ultrascale+ RFSoC ZU28EG (early access)	1x 100G
7	Microchip PolarFire MPF300-EVAL-KIT	1x 10G
8	Trenz Electronics TE0950 with AMD Versal AI Edge VE2302	1x 25G
11	AMD/Xilinx Alveo U55C with Virtex UltraScale+	4x 25G
12	Lattice Avant-G Versa Board	1x 10G
13	Arrow AXE5-Eagle with Altera Agilex 5E	1x 10G
14	AMD/Xilinx ZCU111 with Zynq Ultrascale+ RFSoC ZU28EG	1x 10G
15	Tria AUB15p with AMD/Xilinx Artix Ultrascale+	1x 10G
101	AMD/Xilinx ZCU102 with Zynq Ultrascale+ MPSoC ZU9EG	1x 10G

In MLEs Developer Zone you will find an in depth guide of how to evaluate NPAP using the ERDs. The objective of an ERD is to facilitate design-in via a complete design project running on selected FPGA boards.



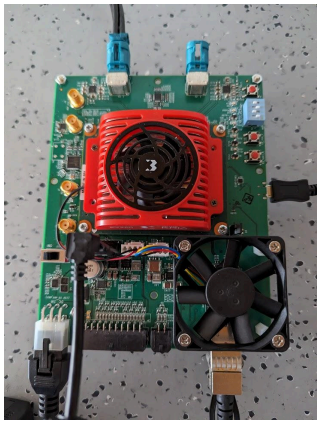
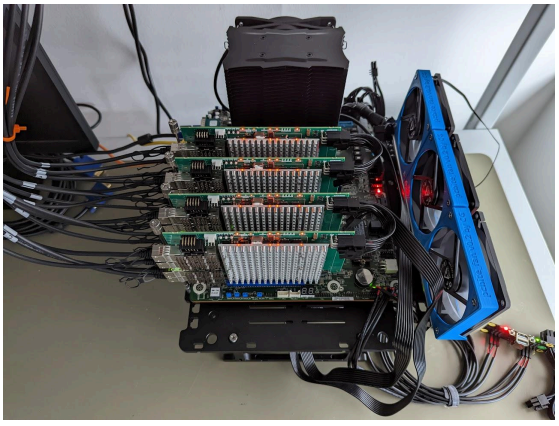
Each ERD typically instantiates the full stack including MAC, Ethernet, IPv4 (with ICMP and IGMP), plus one or more TCP session instances, the UDP block, a Netperf/Netserver implementation and an Network Impairment generator in programmable logic.

A more in-depth documentation of the workings of the ERD and the support IPs is available under license. Please contact your MLE sales representative for more information.

Here some exemplary setups from MLE's NPAP Test Lab:





3.6. IP Core Deliverables

Deliverables include IEEE 1685 IP-XACT packages which include NPAP Kernel plus NPAP Support IP blocks plus non-IP-XACT FPGA reference design projects. The following design blocks are part of a typical delivery package:

- Low-Latency Ethernet MAC for 10G/25G
- NPAP Kernel with IPv4, TCP, UDP
- NPAP Support IP
    - Data Generator Checker
      (great for system-level testing and for performance tuning)
    - TCP Command Application
    - TCP Demo Application
    - UDP Demo Application
    - Netperf Control Application
    - Hardware Abstraction Layer (HAL) python package
    - Network Statistics and Diagnostics (optional)
    - Network Impairment and Bit Error Insertion (optional)
- Evaluation Reference Design (as FPGA Design Project Archive)

3.7. Optional Diagnostics and Network Statistics

In particular at faster line rates, diagnosing network protocol issues can be costly, time-consuming and challenging. Therefore, MLE NPAP comes with specialized diagnostics blocks, providing a powerful suite of tools for in-depth network analysis and efficient troubleshooting of network behavior. These diagnostics offer TCP/UDP/IPv4 layer-specific visibility of counters and status information similar to those in Linux or Windows, for example:

- **TCP Core Activity:** Track performance and network usage of individual cores.
- **TCP Peer Balance:** Monitor data flow and identify peer misconfigurations.
- **Buffer Optimization:** Fine-tune buffer settings using diagnostic data and RTT.
- **General Troubleshooting:** Track packets, errors, and pinpoint issue sources.

MLE NPAP's diagnostics blocks are accessed using AXI4-Lite and enable efficient diagnosis and a faster path towards resolving network problems when using hardware acceleration.

3.7.1. Diagnostics Data Acquisition Modes

The MLE NPAP diagnostics system supports multiple data acquisition modes to cater to different debugging and analysis needs. These modes define how the underlying counter and event state registers are updated. The selected mode applies universally to all readable counter and event state registers within the respective diagnostics block:

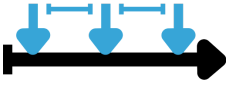
Continuous Mode

Registers the current state of events and counters every clock cycle. This mode provides the most immediate, real-time data feedback.
Use Case: Ideal for live monitoring of instantaneous activity and quick verification of data paths.
Note: Reading multiple diagnostics registers in quick succession may lead to incoherent data, as the values reflect different, continuous points in time.

Snapshot Mode

Takes an instantaneous, synchronous snapshot of all counter and event states upon a user-triggered write to the control register. The resulting data set is coherent across all registers.
Use Case: Essential for event-driven debugging where you need a single, consistent state of the system immediately following a critical event.

Window Mode



Automatically takes periodic snapshots of the current state of events and counters at user-defined time intervals.
Use Case: Perfect for long-term performance trending, resource analysis,

and historical comparison where you need to track changes over defined time segments.

3.7.2. Network Diagnostics Resource Costs

For better design visibility and for gathering network statistics and diagnostics MLE NPAP has the option to spend additional FPGA resources.

As a rule-of-thumb basic Network Diagnostics requires ~10% more LUTs and ~20% more FFs!

The following table is an example showing the **additional resources for diagnosing 6 TCP sessions** in an AMD/Xilinx UltraScale+ device. Please compare this to the absolute resource values in Section 6. below!

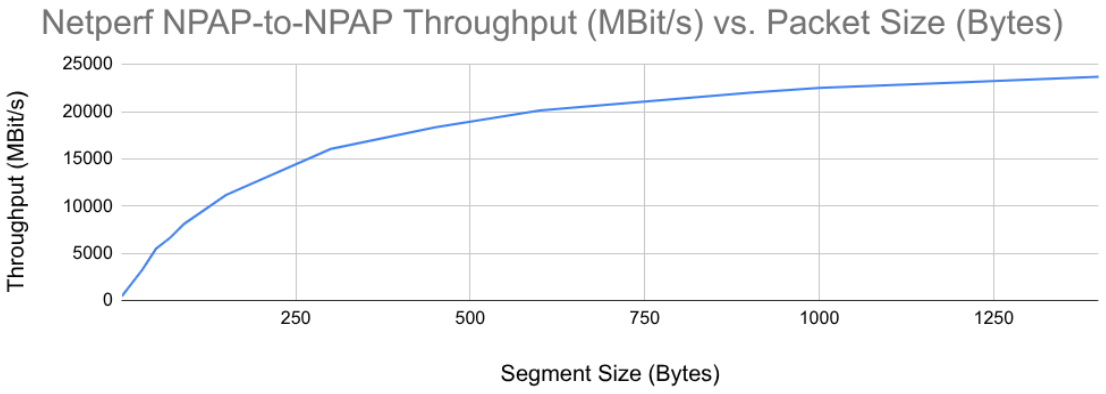
Module Name	LUTs	FFs
npap_diag_capture_u0	6262	17828
npap_diag_control_eth	137	203
npap_diag_control_ipv4	137	203
npap_diag_control_tcp	137	203
npap_diag_control_udp	137	203
npap_diag_counter_tcp [0]	760	2695
npap_diag_counter_tcp [1]	1016	2695
npap_diag_counter_tcp [2]	1080	2695
npap_diag_counter_tcp [3]	825	2695
npap_diag_counter_tcp [4]	761	2695
npap_diag_counter_tcp [5]	1210	2695
npap_diag_register_u0	793	822
npap_diag_register_ethernet_u0	1	188
npap_diag_register_ipv4_u0	1	188
npap_diag_register_tcp_u0	60	258
npap_diag_register_udp_u0	1	188

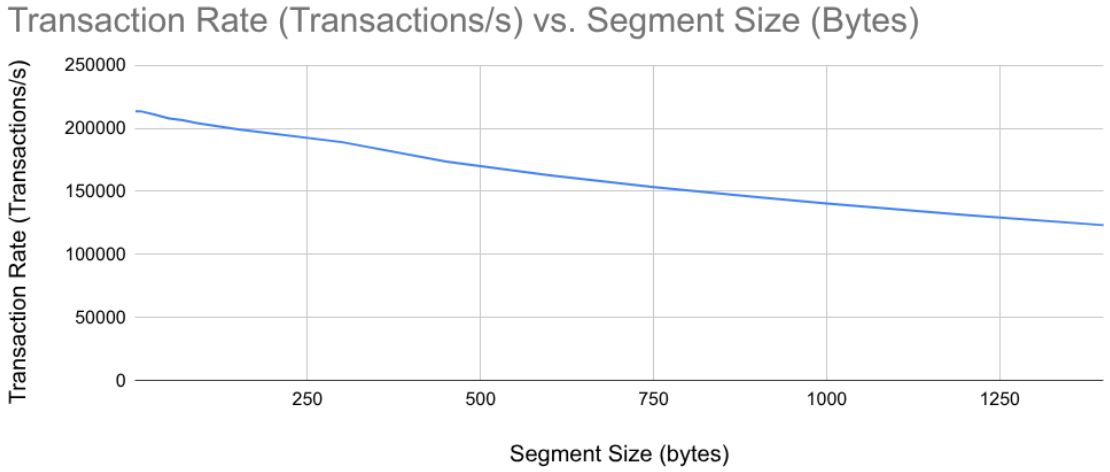
3.8. Optional Benchmarking using Netperf

MLE NPAP can instantiate a fully accelerated version of Netperf and many of our Evaluation Reference Designs do so. This Netperf/Netserver block is compatible with open source Netperf[11] 2.6 and can be used for system-wide tuning, for functionality analysis and for performance benchmarking.

The charts below show the results of a 25 GbE NPAP-to-NPAP benchmark, underlining NPAP's superior performance:

- **Consistent High Throughput:** The first chart, measuring a TCP_STREAM test, demonstrates NPAP's ability to achieve consistent, close-to-line-rate throughput.
- **Ultra-Low Latency:** The second chart, measuring a TCP_RR (Request/Response) test, highlights the ultra-high and deterministic transaction rates which is a key advantage that only a full-hardware network stack can deliver due to the required low round-trip-time (RTT) and low packet loss.





[11] https://hewlettpackard.github.io/netperf/

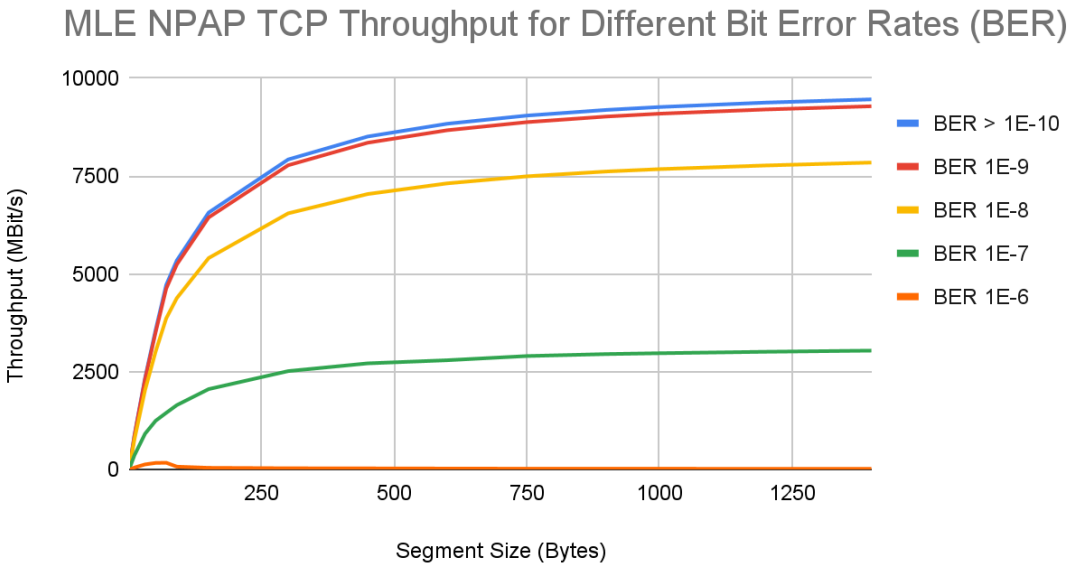
3.9. Network Impairment and Bit Error Insertion

MLE NPAP has an optional integrated Network Impairment Generator (Network Impairment is also known as Bit Error Insertion). This helps to rigorously test and validate the robustness of your network by emulating real-world network impairments, such as Bit Error Rates (BER), at line speed.

The Ethernet standard requires a BER no higher than $1x10^{-12}$. NPAP's Network Impairment Generator allows you to test your system far beyond this ideal requirement, simulating real-world conditions like those found in high-EMI environments or when using slip rings, for example.

The Network Impairment Generator IP Core sits on the data path between the TCP/UDP/IPv4 layer and the Ethernet MAC layer, allowing for precise, controlled injection of errors. This enables you to perform stress testing and gain valuable insights into system performance under non-ideal conditions, ensuring your implementation remains resilient against link quality degradation.

The graph below illustrates how different levels of Bit Error Rate (BER) over a 10 GBit/s link, emulated using NPAP's Network Impairment Generator, affect the TCP throughput of a TCP connection between two MLE NPAP instances. Obviously, the TCP re-transmissions "eat" into the net data throughput:



This chart shows that MLE's TCP/IPv4 stack can deliver good throughput - even on very noisy networks.

4. System-Level Architecture Using TCP/UDP/IPv4

Team MLE has gained long and deep experiences with integrating NPAP into networked and distributed systems and will support you in identifying, implementing and testing the right architecture choice. Key aspects are outlined below. Please contact us for more details.

4.1. Picking the "Right" TCP Peer

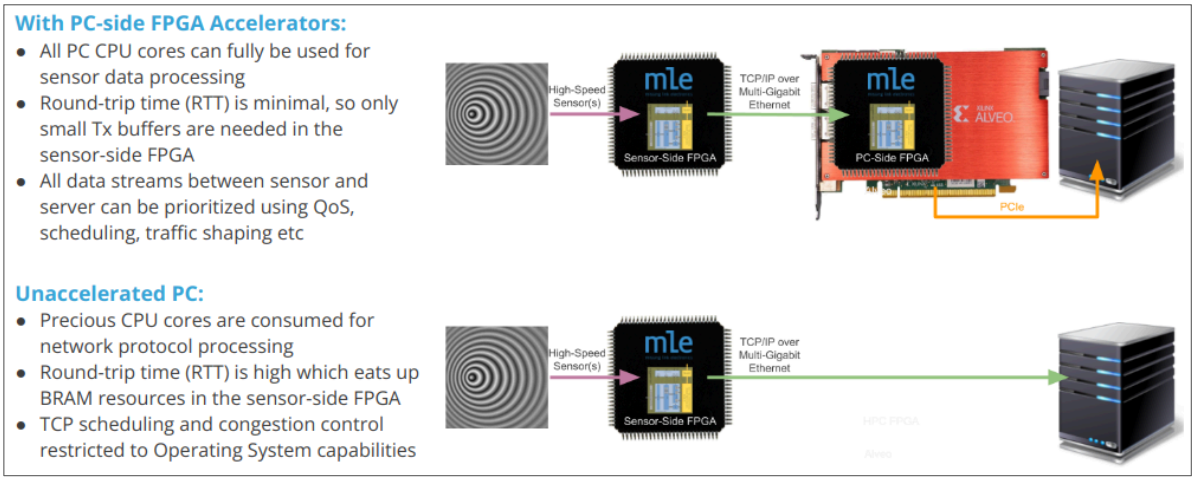
MLE NPAP has a clear focus on highest performance in a very resource efficient FPGA implementation which is reliable (the key reason to use TCP) and interoperable with (almost) any other TCP/UDP/IPv4 stack.

When you optimize data transports towards low latency and/or high bandwidth, keep in mind how both, **TCP flow control and TCP congestion control**, function and do parameterize both peers accordingly.

NPAP runs the entire protocol stack as a digital circuit. So, when NPAP is on the receiving side, TCP packets will be checked, and acknowledged (ACK'ed), in a very short time and at a very high rate (close to line rate). That may challenge a "slow sender". Similarly, when NPAP is on the sender side, TCP packets will be generated and sent in a very short time and at a very high rate (close to line rate). That may challenge a "slow receiver".

Experimenting, and tweaking parameters on either side, is key to delivering good performance. Predictable high bandwidth and low latency is typically delivered by a "balanced" TCP connection, for example by putting NPAP on both sides.

When implementing so-called High-Speed Data Acquisition systems[12] we do recommend investigating NPAP not only for sensor-side TCP/IPv4 acceleration but also for PC-side:



[12] https://devzone.missinglinkelectronics.com/application-notes/high-speed-data-acquisition-systems/

4.2. Picking the Right TCP Rx/Tx Buffer Sizes

Obviously, larger Tx and Rx buffers deliver higher bandwidths, but at the cost of transport latency. Worse, NPAP Tx and Rx buffers require expensive FPGA BRAM resources. To help you pick a good tradeoff, here is the metric to determine TCP buffer sizes for TCP (keep in mind, TCP buffers are placed on both ends: Tx side and Rx side):

- Buffer size (in bits) = Bandwidth (in bits-per-second) * RTT (in seconds)

RTT is the Round-Trip Time which is the time for the sender to transmit the data plus the time-of-flight for the data, plus the time it takes the recipient to check for packet correctness (CRC), plus the time for the recipient to send out the ACK, plus the time-of-flight for the ACK, plus the time it takes the sender to process the ACK and release the buffer. Here examples:

1. Assuming a direct connection with the recipient being NPAP, and a 25 GBit/s link, ACK times can be assumed to be less than 1 microsecond. This translates to minimal buffer sizes of 25 Kbits, which is equivalent to a single 36 Kbit FPGA BRAM.
2. Assuming a 10 GBit/s link and the recipient being software, then RTT can be much longer, mostly due to the longer processing times in the OS on the recipient side. For a modern Linux we can assume RTTs of 50 microseconds, or longer (you can run 'Netperf' on your machine to find out). Means buffer sizes shall be around 500K bits, or the 64K Bytes of BRAM we typically instantiate.

NPAP allows to set Tx and Rx buffer sizes separately and individually for each TCP Core, to facilitate optimizations for better performance/resource trade-offs in more unidirectional dataflows.

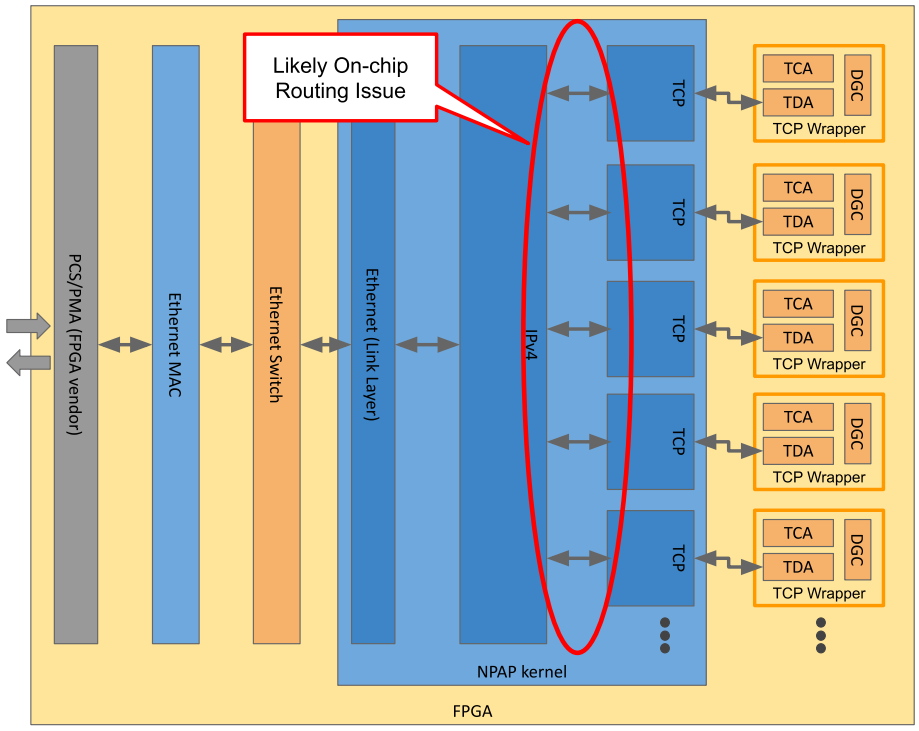
To ensure these receive buffers are used effectively, especially when they exceed the standard 65,535-byte TCP window, MLE NPAP implements the **TCP Window Scale** option (RFC 7323). This is essential for achieving line-rate performance on high-latency networks (i.e., those with a large Bandwidth-Delay Product). MLE NPAP automatically calculates the correct scale factor based on the specified Rx Buffer Size for each TCP core. It then advertises this full receive capacity during the connection handshake, enabling the remote peer to send enough data to fill the pipe and maximize throughput.

4.3. Picking the Right Number of TCP Cores

In typical software systems, the cost of opening a TCP connection is quite CPU expensive, and may take a long time because of RTT and processing times in the operating system. Therefore, most software driven systems keep a TCP connection alive "forever" rather than closing it. In software, the low costs of system RAM for storing each TCP connection's state are not worth the CPU processing costs.

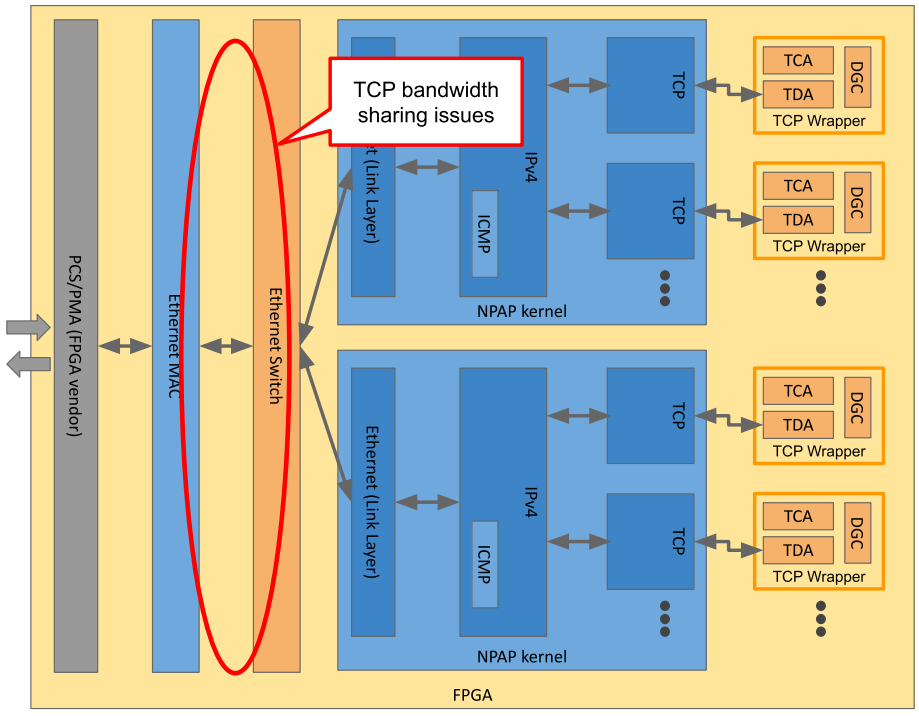
For NPAP, each TCP connection which is open at the same time requires a dedicated TCP Core - which costs FPGA / ASIC resources. However, if RTT is low such as in a LAN, and with the very low costs of opening and closing a TCP connection in NPAP (a few hundred FPGA clock cycles), "time sharing" TCP Cores can save a lot of FPGA resources without any negatives. Please refer to the MLE Technical Brief TB20201203 "Deterministic Networking with TSN-10/25/50/100G" for more information on "time sharing" TCP Cores[13].

Typically, FPGA routing is a key limiting factor: The more TCP Cores you instantiate, the more on-chip AXI-Stream FPGA routing congestion you will see, the harder it is to achieve timing closure. As a rule of thumb, more than 20 TCP Cores may cause FPGA routing issues (obviously depending on your FPGA device and toolchain).

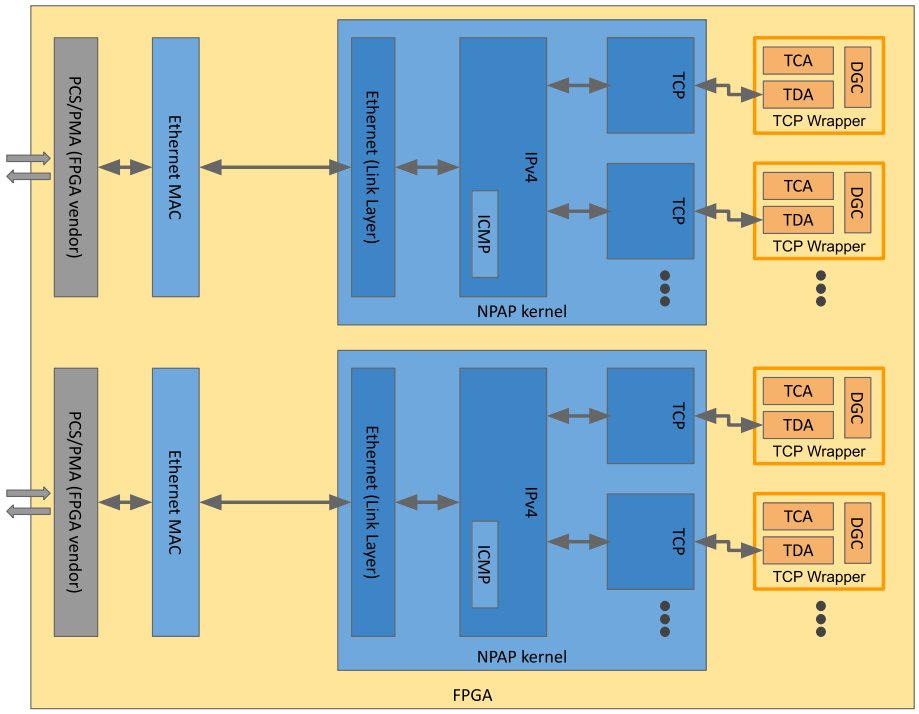


Hence, another option is to instantiate multiple NPAP subsystems along with Ethernet switch logic between the Ethernet port and the multiple NPAP instances. This relaxes the on-chip AXI-Stream FPGA routing. However, now more TCP connections share the Ethernet link's line-rate which may cause other performance issues.

[13] https://devzone.missinglinkelectronics.com/application-notes/deterministic-networking-with-tsn-10-25-50-100g/



So, yet another option is to partition your TCP connections over multiple Ethernet links so one TCP connection does not “disturb” the other. Then, instantiate multiple NPAP subsystems, each wired to a separate network port, with a separate IPv4 address:



4.4. Optimizing NPAP for Linerate Performance

MLE is constantly working with FPGA vendors to improve NPAP clock frequency. While NPAP originally was designed for ASIC implementation, MLE has adopted NPAP for efficient implementation using modern FPGA fabric. Unlike other TCP stacks for FPGA, NPAP features a 128 bit wide bi-directional datapath which puts NPAP into a unique position for realizing high-bandwidth FPGA-based SmartNICs without "FPGA bloat".

Larger bit widths, 512 bits or more, cause "FPGA bloat" which is wasting FPGA resources. Smaller bit widths, 64 bits or less, do require unrealistic high clock frequencies to deliver high line rates as the following table shows:

	10 Gbps	**25 Gbps**	**50 Gbps**	**100 Gbps**
32 bit	312.5 MHz	781.25 MHz	1,562.5 MHz	3,125.0 MHz
64 bits	156.25 MHz	390.625 MHz	781.25 MHz	1,562.5 MHz
128 bits	78.125 MHz	195.3 MHz	390.625 MHz	781.25 MHz
512 bits	19.5 MHz	48.8 MHz	97.7 MHz	195.3 MHz

Please refer to MLE Technical Brief TB20230523 "Put a TCP/UDP/IPv4 Turbo Into Your FPGA-SmartNIC"[14] as this discusses pipelining the data paths to avoid "FPGA bloat".

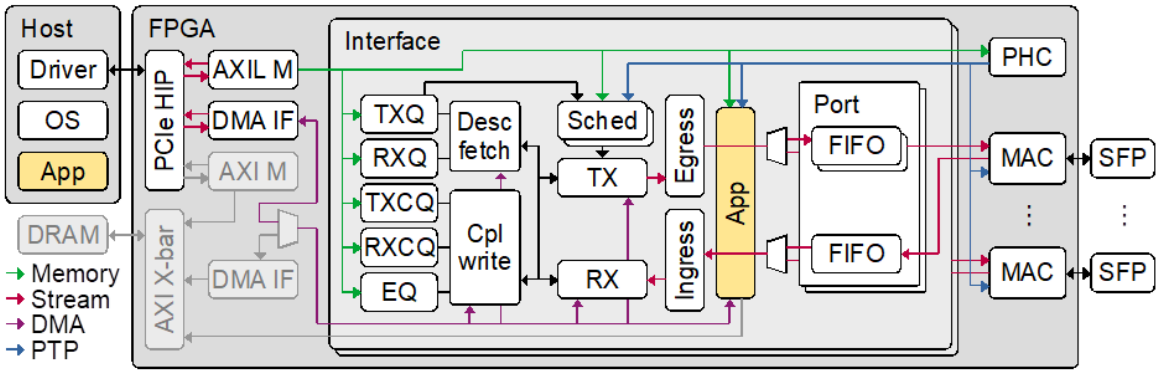
[14] https://devzone.missinglinkelectronics.com/application-notes/put-a-tcp-udp-ip-turbo-into-your-fpga-smartnic/

4.5. Combining NPAP With FPGA Network Interface Cards (NIC)

An FPGA Network Interface Card (NIC) does some network packet handling in FPGA logic and then DMAs the data into a (Linux) host computer. At MLE we have been using (and contributing to) the Corundum project: http://corundum.io

Corundum is an open-source, high-performance FPGA-based NIC and platform for In-Network Compute. Features include a high performance datapath, 10G/25G/100G Ethernet, PCIe connectivity to the host, a custom, high performance, tightly-integrated PCIe DMA engine, many (1000+) transmit, receive, completion, and event queues, scatter/gather DMA, MSI, multiple interfaces, multiple ports per interface, per-port transmit scheduling including high precision TDMA, flow hashing, RSS, checksum offloading, and **native IEEE 1588 PTP timestamping**. A Linux driver is included that integrates with the Linux networking stack. Development and debugging is facilitated by an extensive simulation framework that covers the entire system from a simulation model of the driver and PCI express interface on one side to the Ethernet interfaces on the other side (https://docs.corundum.io/en/latest/contents.html).

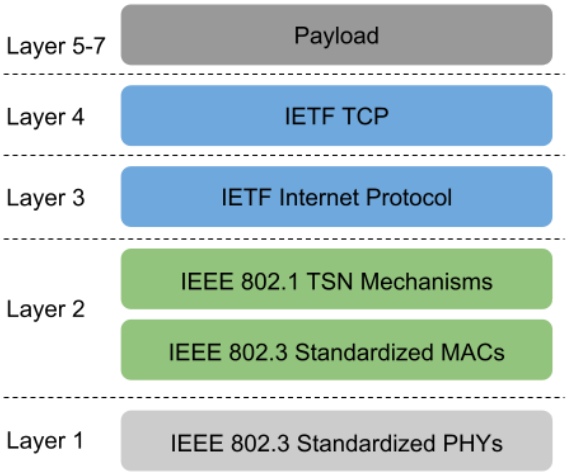
One of the key advantages of the Corundum architecture is support for In-Network Processing inside the FPGA logic, shown as "App" in the block diagram:



These Corundum "Apps" can serve as a "turbo", and one of those turbos can be NPAP. This is quickly evolving so please contact us for more information!

4.6. Implementing Time-Sensitive Networking (TSN)

TSN has become a set of emerging, open IEEE standards with momentum in industrial markets (for 10/100/1000 Mbps speed) and in next-generation Automotive Zone architectures (for 10/25/50 Gbps speeds). Aspects such as Time-Aware Traffic Shaping also find application in telecommunication, Provider Back-Bone (PBB) Switching or Software-Define Wide Area Networks (SD-WAN), for example. TSN and TCP can be combined according to the OSI Layers.



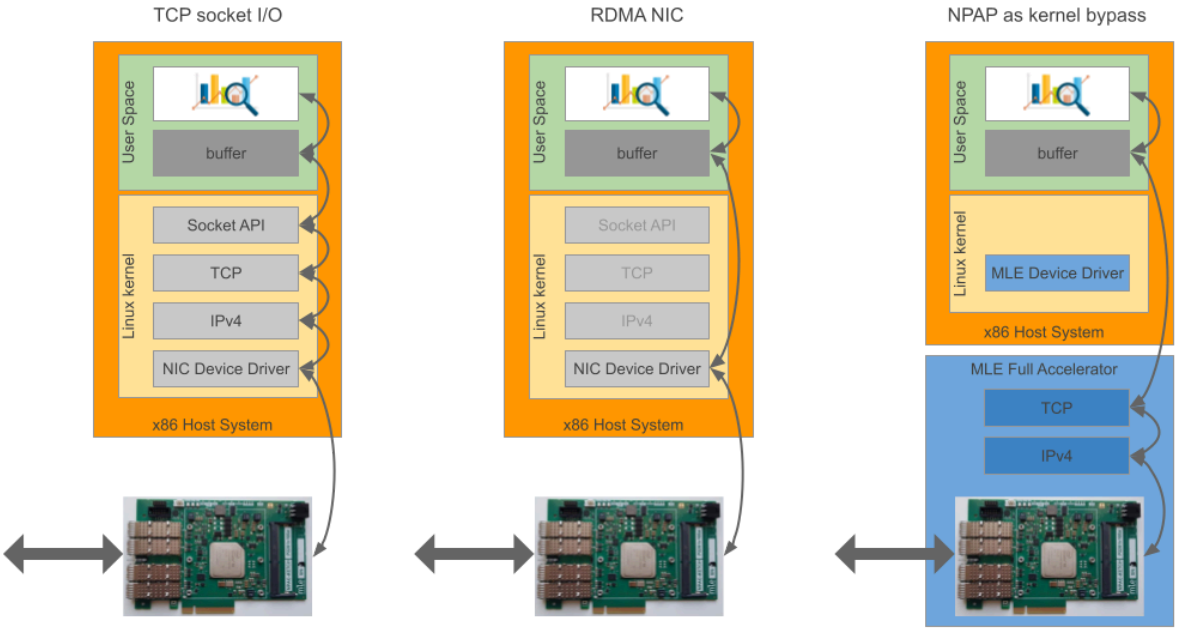
The outcome is a deterministic and reliable network protocol, which makes TCP/IPv4 over TSN a very good candidate for all networking where IT (Information Technology) and OT (Operations Technology) converge, or in Systems-of-Systems backbones. TSN itself is quickly evolving so please contact us for more information!

4.7. Linux Kernel Bypass With NPAP

Increased Ethernet speeds push a need to offload CPUs from the burden of TCP/UDP/IPv4 processing. Various kernel bypass options exist, some include so-called RDMA (Remote DMA). NPAP can provide architecture choices for implementing such kernel bypasses.

Normally, from a CPU's perspective, TCP socket I/O means sending and/or receiving (raw) network data between the host CPU and the NIC. The kernel runs network protocol processing for IPv4, TCP, and socket APIs. User space applications that generate and/or digest the network data add to the CPU processing burden.

RDMA skips most steps and a so-called rNIC (RDMA NIC) device driver directly interfaces with user space memory, effectively bypassing the kernel.



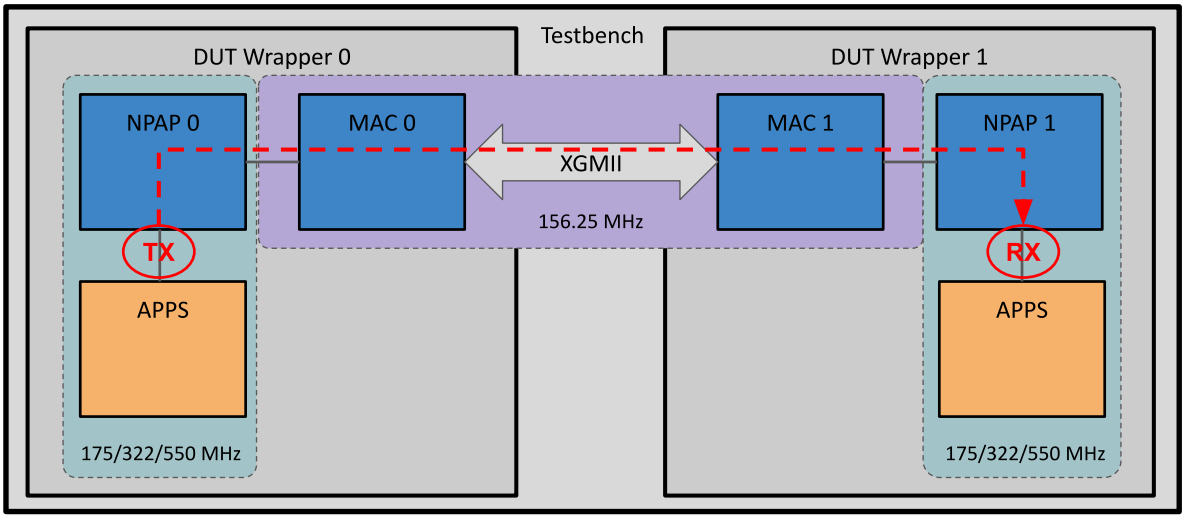
NPAP can operate in a similar way, where NPAP does all TCP/IPv4 processing in dedicated ASIC / FPGA logic and then a (Linux) device driver copies the payload data between user space memory and NPAP.

4.8. Latency Analysis Results

MLE analyzed processing latency using RTL simulation of two instances of NPAP (using different clock speeds) connected via 10G LL MAC via XGMII (clocked at 156.25 MHz).

TCP Payload Size [Byte]	Clock cycles	Latency [ns] at 175 MHz	Latency [ns] at 322 MHz	Latency [ns] at 550 MHz
1	62	354.3	192.5	112.7
32	67	382.9	208.1	121.8
64	73	417.1	226.7	132.7
160	91	520.0	282.6	165.5
448	145	828.6	450.3	263.6
960	241	1,377.1	748.4	438.2
1216	289	1,651.4	897.5	525.5
1456	334	1,908.6	1,037.3	607.3

Latency was measured "one-way, door-to-door": Using RTL simulation we count the number of clock cycles it takes from sending payload data from one NPAP instance (TX) via the full NPAP kernel until the other instance of NPAP receives that payload data (RX). Here the system-level block diagram:



Obviously, increasing the NPAP clock frequency will reduce latency for asynchronous NPAP subsystems. More information on dependable latency numbers can be found in our Technical Brief "Myth-Busting Latency Numbers for TCP Offload Engines"[15]

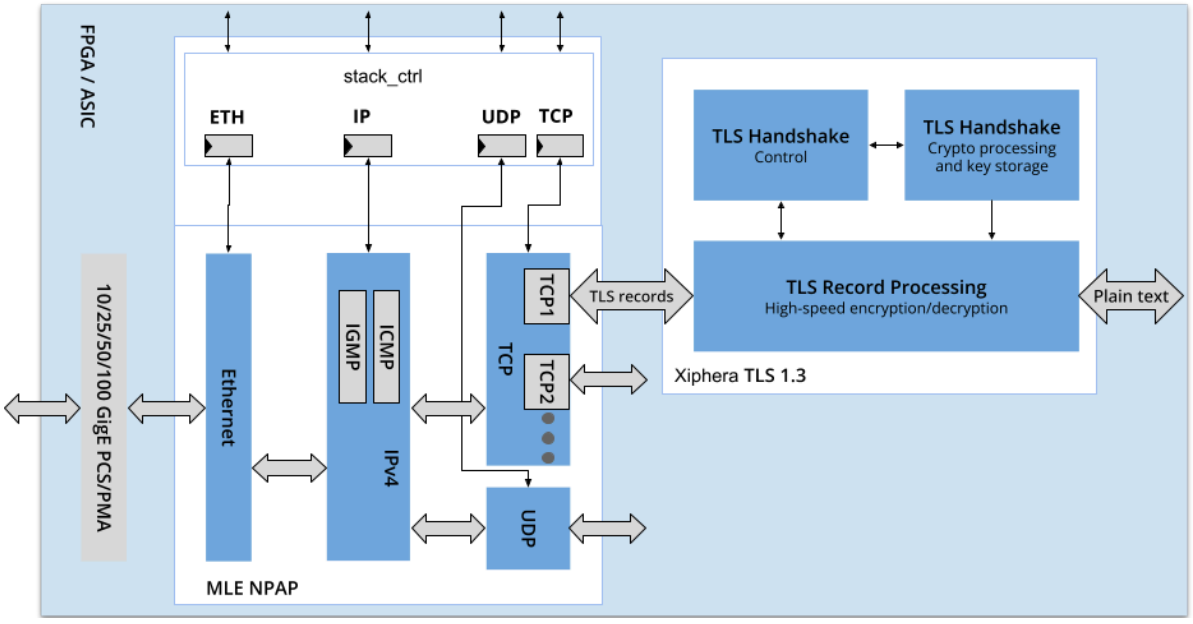
[15] https://devzone.missinglinkelectronics.com/application-notes/myth-busting-latency-numbers-for-tcp-offload-engines/

4.9. Optional Transport Layer Security (TLS)

MLE has been working with partner Xiphera to integrate NPAP with Xiphera's TLS IP Cores for FPGA. Successful integration has been delivered to first customers.

TLS is a cryptographic protocol that provides end-to-end data security, on top of the Transmission Control Protocol (TCP) layer. Implementing TLS has become a standard practice for building secure web apps. With growing needs for security compliance, for example under IEC 62443, TLS is also an option for protecting sensitive data transported using MLE NPAP.

Collaboration within the FPGA ecosystem created joint solutions combining MLE's TCP/IPv4 Network Protocol Accelerator Platform (NPAP) with Xiphera's TLS 1.3 to ensure secure and reliable connection between devices over LAN and WAN.



Since the TCP/IPv4 stack and the TLS 1.3 security protocol – including importantly both key exchange and key management – are both executed entirely in hardware, the joint solution has both scalable high-speed performance and minimizes attack surface, especially when compared to a software-based approach. The FPGA hereby utilizes the hardware Root of Trust, best suited for applications like critical communication in defense, space technology, and energy production and distribution.

5. Chip Design and NPAP Integration

NPAP is available as VHDL RTL source code or as a netlist suitable for synthesis by 3rd party and FPGA vendor tools which makes it possible to support almost any FPGA or ASIC target platform. The following toolchain versions have been tested at MLE:

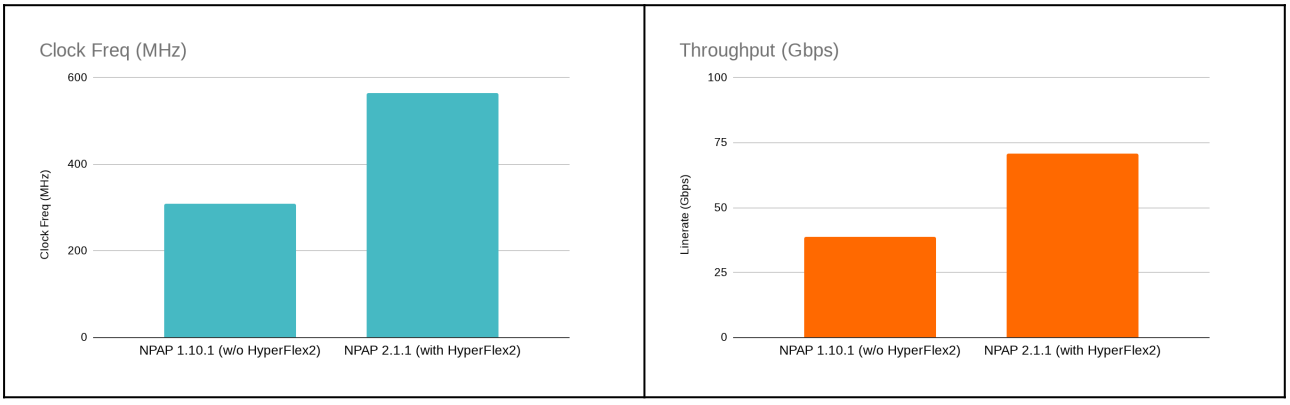
Version	AMD/Xilinx	Altera/Intel	Lattice Semi	Microchip
NPAP 2.5.0 (or newer)	Vivado 2022.2 Vivado 2024.2	Quartus Prime Pro 22.4, 24.2	Radiant 2024.2	Libero 2024.2
NPAP 2.4.5 (latest)	Vivado 2022.2	Quartus Prime Standard 22.4	n/a	n/a
NPAP 1.10.1	Vivado 2018.3	Quartus Prime Pro 22.1	n/a	Libero 2021.2

Starting with version 2.5.0, NPAP is delivered as **VHDL IEEE-1076 2008 RTL source code**. This standard is fully supported by almost all modern FPGA toolchains, including those listed in the table above.

However, some older toolchains, such as Altera Quartus Prime Standard (required for legacy devices like Stratix V), do not offer full VHDL-2008 support. For these specific tools, the latest compatible release is NPAP v2.4.5.

Optimized for FPGA Pipelining

Starting with NPAP 2.0, RTL code has been optimized for FPGA pipelining such as Altera/Intel Hyperflex, Altera/Intel HyperFlex2 or AMD/Xilinx IMux. This avoids so-called “FPGA bloat”: Unlike other FPGA IP Cores which rely on 256 or even 512 bit wide datapaths, NPAP comes with more shallow combinatorial circuits to increase the FPGA clock speed, and thereby the on-chip throughput.



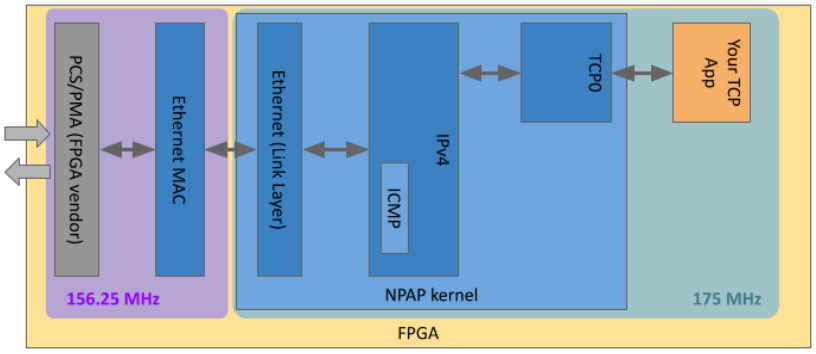
For more information please refer to MLE Technical Brief TB20230523[16].

[16] https://devzone.missinglinkelectronics.com/application-notes/put-a-tcp-udp-ip-turbo-into-your-fpga-smartnic/

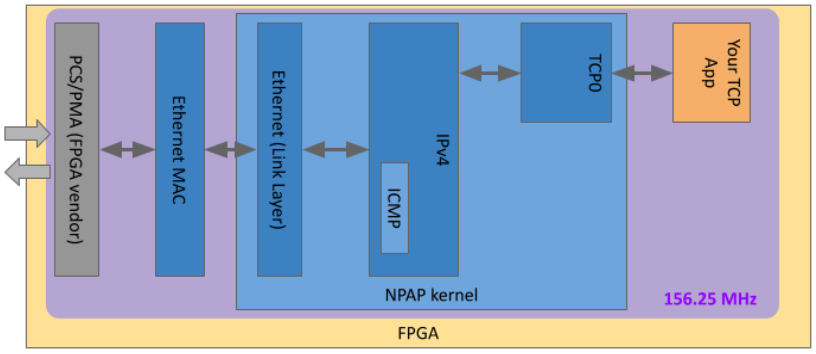
Synchronous vs. Asynchronous Clocking

When instantiating NPAP you have two options for on-chip clocking the Ethernet MAC:

- **Asynchronous clocking** - where the Ethernet MAC block is in a different clock domain (here 156.25 MHz) than the other functions including the NPAP kernel, the TCP Cores, the UDP Core, etc (here at 175 MHz):



- **Synchronous clocking** - where the Ethernet MAC block is within the very same clock domain (here 156.25 MHz) as the other functions (IPv4, TCP, UDP block).



Both are equally good, but depending on your overall design, your device utilization, potential timing challenges, or your performance requirements, one may be better than the other. Please contact us in case you need technical support!

RTL Simulation and Testbench

MLE's engineering team extensively verifies, tests and optimizes NPAP using RTL simulation. This includes replaying stimuli from Wireshark PCAP capture files to diagnose unexpected behaviour or to investigate performance issues.

A complete RTL Simulation environment is available upon request - so please do not hesitate to contact us!

FPGA Resource Estimates

For your reference please find below resource reports for different FPGA target devices.If you are interested in integrating NPAP into another target technology including ASIC, please contact us!

Resource estimates are directly tied to the specific parameterization of NPAP: A typical NPAP integration can range from a minimum of 10K LUTs to, literally, 1 million LUTs (for dozens of TCP cores).

5.1. Resource Estimates for AMD Versal AI Edge Series

The following table shows resources for AMD/Xilinx Versal AI Edge Series fabric (xcve2302-sfva784-1LP-e-S-es1) compiled with AMD/Xilinx Vivado/Vitis 2024.1 - instantiating the following design features

- Ethernet block
- IPv4 block
- 1 UDP block
- 4 instances of TCP blocks
- Diagnostics blocks

	Total LUTs	Logic LUTs	LUTRAMs	SRLs	FFs	RAMB36	RAMB18	URAM	DSP Blocks
npap_tcp_udp_wrapper_u0	70944	69010	1914	20	62052	25	1	70	4
npap_tcp_udp_top_u0	70944	69010	1914	20	62052	25	1	70	4
axis_register_udp_u0	228	228	0	0	430	0	0	0	0
axis_register_udp_u1	292	292	0	0	431	0	0	0	0
gen_hhi_to_axis_adapter	1214	1214	0	0	1978	0	0	0	0
npap_configuration_block_u0	103	103	0	0	339	0	0	0	0
npap_priority_manager_configurati	29	29	0	0	43	0	0	0	0
npap_visibility_block	9500	9480	0	20	13324	0	0	0	0
npap_diag_capture_u0	8297	8277	0	20	12511	0	0	0	0
(npap_diag_capture_u0)	46	26	0	20	694	0	0	0	0
npap_diag_control_eth	157	157	0	0	278	0	0	0	0
npap_diag_control_ipv4	157	157	0	0	278	0	0	0	0
npap_diag_control_tcp	157	157	0	0	203	0	0	0	0
npap_diag_control_udp	157	157	0	0	278	0	0	0	0
npap_diag_counter_tcp_loop[0]	1803	1803	0	0	2695	0	0	0	0
npap_diag_counter_tcp_loop[1]	1961	1961	0	0	2695	0	0	0	0
npap_diag_counter_tcp_loop[2]	1936	1936	0	0	2695	0	0	0	0
npap_diag_counter_tcp_loop[3]	1923	1923	0	0	2695	0	0	0	0
npap_diag_register_u0	1204	1204	0	0	813	0	0	0	0
(npap_diag_register_u0)	809	809	0	0	0	0	0	0	0
npap_diag_register_ethernet_u	1	1	0	0	190	0	0	0	0
npap_diag_register_ipv4_u0	1	1	0	0	189	0	0	0	0
npap_diag_register_tcp_u0	393	393	0	0	245	0	0	0	0
npap_diag_register_udp_u0	1	1	0	0	189	0	0	0	0
wrapper_ll_ip_tcp_u0	59583	57669	1914	0	45507	25	1	70	4
(wrapper_ll_ip_tcp_u0)	633	633	0	0	765	0	0	0	0
g0.i_tcpTxMux	0	0	0	0	2	0	0	0	0
gen_WithUdp.i_udp	4864	4864	0	0	4265	5	0	8	0
gen_tcpConnections[0].i_tcp	12007	11573	434	0	8830	4	0	17	1
gen_tcpConnections[1].i_tcp	11174	10764	410	0	8365	1	0	11	1
gen_tcpConnections[2].i_tcp	12007	11573	436	0	8846	4	0	17	1
gen_tcpConnections[3].i_tcp	12011	11575	436	0	8931	4	0	17	1
iBusScheduler8	531	531	0	0	36	0	0	0	0
i_internetLayer	3383	3265	118	0	3108	0	0	0	0
i_networkLayer	2947	2867	80	0	2404	7	1	0	0
i_resetStretchStack	54	54	0	0	45	0	0	0	0

5.2. Resource Estimates for AMD/Xilinx Ultrascale+ Series

The following table shows resources for AMD/Xilinx Zynq Ultrascale+ MPSoC ZU9EG compiled with AMD/Xilinx Vivado 2022.2 - instantiating the following design features:

- 10 GigE Low-Latency MAC from Fraunhofer HHI
- Ethernet block
- IPv4 block
- UDP block
- 10 instances of TCP blocks
- Diagnostics blocks

Instance	Total LUTs	Logic LUTs	LUTRAMs	SRLs	FFs	RAMB36	RAMB18	URAM	DSP Blocks
top	111324	106686	4540	0	129415	707	0	0	10
mac_10_gbe_wrapper_u0	1790	1780	10	0	1016	0	0	0	0
mac10gbe_top_u0	1790	1780	10	0	1016	0	0	0	0
mac10Gbe_struct_u0	1790	1780	10	0	1016	0	0	0	0
mac_interface_adapter_generic_64_u	161	161	0	0	370	0	0	0	0
axis_dwidth_converter_128_to_64_	119	119	0	0	222	0	0	0	0
axis_dwidth_converter_64_to_128_	42	42	0	0	148	0	0	0	0
reset_stretch_reset_rx_mac	6	6	0	0	17	0	0	0	0
reset_stretch_reset_tx_mac	6	6	0	0	17	0	0	0	0
reset_sync_reset_system_to_reset_m	0	0	0	0	2	0	0	0	0
reset_sync_reset_system_to_reset_m	0	0	0	0	2	0	0	0	0
npap_tcp_udp_wrapper_u0	109534	104906	4530	98	128399	707	6	0	10
npap_tcp_udp_top_u0	109534	104906	4530	98	128399	707	6	0	10
axis_register_udp_u0	100	100	0	0	428	0	0	0	0
axis_register_udp_u1	263	263	0	0	424	0	0	0	0
gen_hhi_to_axis_adapter	3112	3112	0	0	266	0	0	0	0
npap_configuration_block_u0	135	135	0	0	338	0	0	0	0
npap_priority_manager_confi	52	52	0	0	52	0	0	0	0
npap_visibility_block	11118	11068	0	50	29770	0	0	0	0
npap_diag_capture_u0	10311	10261	0	50	28912	0	0	0	0
(npap_diag_capture_u0)	115	65	0	50	1150	0	0	0	0
npap_diag_control_eth	137	137	0	0	203	0	0	0	0
npap_diag_control_ipv4	137	137	0	0	203	0	0	0	0
npap_diag_control_tcp	137	137	0	0	203	0	0	0	0
npap_diag_control_udp	137	137	0	0	203	0	0	0	0
npap_diag_counter_tcp_l	9651	9651	0	0	2695	0	0	0	0
npap_diag_register_u0	808	808	0	0	858	0	0	0	0
wrapper_ll_ip_tcp_u0	94739	90161	4530	48	91959	707	6	0	10
(wrapper_ll_ip_tcp_u0)	800	800	0	0	192	0	0	0	0
g0.i_tcpTxMux	0	0	0	0	4	0	0	0	0
gen_WithUdp.i_udp	3649	3601	0	48	3889	47	5	0	0
gen_tcpConnections[0].i_t	8403	7977	426	0	8153	68	0	0	1
gen_tcpConnections[1].i_t	8052	7618	434	0	7792	41	0	0	1
gen_tcpConnections[2].i_t	9720	9286	434	0	8410	68	0	0	1
gen_tcpConnections[3].i_t	8440	8006	434	0	8297	68	0	0	1
gen_tcpConnections[4].i_t	8449	8015	434	0	8296	68	0	0	1
gen_tcpConnections[5].i_t	8446	8012	434	0	8297	68	0	0	1
gen_tcpConnections[6].i_t	8446	8012	434	0	8300	68	0	0	1
gen_tcpConnections[7].i_t	8442	8008	434	0	8311	68	0	0	1
gen_tcpConnections[8].i_t	8456	8022	434	0	8297	68	0	0	1
gen_tcpConnections[9].i_t	8397	8003	434	0	8298	68	0	0	1
iBusScheduler8	1108	1108	0	0	69	0	0	0	0
i_internetLayer	1954	1836	118	0	3043	0	0	0	0
i_networkLayer	1894	1814	80	0	2276	7	1	0	0
i_resetStretchStack	48	48	0	0	35	0	0	0	0

5.3. Resource Estimates for AMD/Xilinx 7-Series

The following table shows resources for AMD/Xilinx 7-Series Kintex fabric (XC7Z045-2) compiled with AMD/Xilinx Vivado 2014.4 - instantiating the following design features:

- 10 GigE Low-Latency MAC from Fraunhofer HHI
- Ethernet block
- IPv4 block
- UDP block
- 2 instances of TCP blocks
- No diagnostics

Instance	Module	Total LUTs	Logic LUTs	LUT RAMs	SRLs	FFs	RAMB36	RAMB18	DSP48 Blocks
wrapper_mac_10gStack	(top)	29459	28425	904	130	25727	50	6	4
(wrapper_mac_10gStack)	(top)	241	241	0	0	0	0	0	0
i_10GStack	Wrapper_LL_IP_TCP__parameterized0	27045	26013	904	128	24644	50	6	4
g0.i_tcpTxMux	TcpTxMux__parameterized0	0	0	0	0	3	0	0	0
gen_WithUdp.i_udp	wrapper_udp__parameterized0	3538	3350	92	96	3732	14	0	0
gen_tcpConnections[0].i	Wrapper_TCP__parameterized0	9347	8963	384	0	7938	16	3	2
gen_tcpConnections[1].i	Wrapper_TCP__parameterized0_1	9349	8965	384	0	7938	16	3	2
iBusScheduler8	BusScheduler8__parameterized0	568	568	0	0	31	0	0	0
i_internetLayer	Wrapper_IP__parameterized0	2989	2913	44	32	3437	2	0	0
i_netLayerConv	MacNetworkLayerConversion__parameterized0	139	139	0	0	72	0	0	0
i_networkLayer	Wrapper_Networklayer__parameterized0	1125	1125	0	0	1491	2	0	0
i_txLinkResetSync	ResetSync__parameterized0_2	0	0	0	0	2	0	0	0
i_ResetStretch_aux	ResetStretch__parameterized0	75	74	0	1	35	0	0	0
i_ResetStretch_stack	ResetStretch__parameterized2	75	74	0	1	34	0	0	0
i_clk_stretch	clk_stretch	1	1	0	0	29	0	0	0
i_gen100ms	GenClk100ms__parameterized0	69	69	0	0	30	0	0	0
i_mac10GbE	mac10Gbe_Wrapper__parameterized0	1957	1957	0	0	955	0	0	0
(i_mac10GbE)	mac10Gbe_Wrapper__parameterized0	0	0	0	0	1	0	0	0
i_mac10Gbe	mac10Gbe__parameterized0	1957	1957	0	0	954	0	0	0

5.4. Resource Estimates for Altera/Intel Agilex 5E

The following table shows resources synthesized for Agilex 5E A5ED065BB32AE4SR0 using Altera Quartus v24.2 - instantiating the following design features:

- Ethernet block
- IPv4 block
- UDP block
- 7 instances of TCP blocks
- Diagnostics blocks

Compilation Hierarchy Node	ALMs used in final placement	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	M20Ks
|npap_tcp_udp_wrapper_u0|	78569.5	103514	117217	18305712	1046
|npap_tcp_udp_top_u0|	78569.5	103514	117217	18305712	1046
|axis_register_udp_u0|	191.0	223	486	0	0
|axis_register_udp_u1|	172.8	222	427	0	0
|gen_hhi_to_axis_adapter[0-6]	1736.1	2240	3920	0	0
|hhiudp_axis_adapter_u0|	20.1	42	4	0	0
|npap_configuration_block_u0|	222.2	254	735	0	0
|npap_priority_manager_configuration_b	47.3	68	60	0	0
|npap_visibility_block|	13698.9	15617	22381	0	0
|npap_diag_capture_u0|	12937.2	14732	21099	0	0
|npap_diag_control_eth|	125.7	247	240	0	0
|npap_diag_control_ipv4|	128.3	247	246	0	0
|npap_diag_control_tcp|	127.7	247	257	0	0
|npap_diag_control_udp|	124.9	247	249	0	0
|npap_diag_counter_tcp_loop[0-6]	9046.8	10535	18609	0	0
|npap_diag_register_u0|	761.7	885	1282	0	0
|npap_diag_register_ethernet_u0|	191.4	252	381	0	0
|npap_diag_register_ipv4_u0|	144.8	150	232	0	0
|npap_diag_register_tcp_u0|	277.2	349	478	0	0
|npap_diag_register_udp_u0|	148.3	134	191	0	0
|wrapper_ll_ip_tcp_u0|	62479.9	84846	89199	18305712	1046
|g0.i_tcpTxMux|	8.8	13	4	0	0
|gen_WithUdp.i_udp|	3102.2	4519	5322	1753088	91
|gen_tcpConnections[0].i_tcp|	7926.0	10966	11470	2482096	140
|gen_tcpConnections[1].i_tcp|	6987.8	9801	9836	1375184	85
|gen_tcpConnections[2].i_tcp|	7970.8	10884	11148	2482688	141
|gen_tcpConnections[3].i_tcp|	8033.7	10912	10900	2482688	141
|gen_tcpConnections[4].i_tcp|	8061.9	10914	11347	2482688	141
|gen_tcpConnections[5].i_tcp|	7976.0	10891	11336	2482688	141
|gen_tcpConnections[6].i_tcp|	8015.6	10887	11038	2482688	141
|i_internetLayer|	2305.1	2955	2961	5312	7
|i_networkLayer|	1905.1	1897	3351	276592	18
|i_resetStretchStack|	24.5	42	47	0	0

5.5. Resource Estimates for Altera/Intel Stratix-10

The following table shows resources for compiled with Altera Quartus Prime v22.4 for 1SX280HN2F43E2VG - instantiating the following design features:

- 10 GigE Low-Latency MAC from Fraunhofer HHI
- Ethernet block
- IPv4 block
- UDP block
- 3 instances of TCP blocks
- No diagnostics

Compilation Hierarchy Node	ALMs used in final placement	ALMs used for memory	Dedicated Combinational ALUTs	Logic Registers	Block Memory Bits	M20Ks	DSP Blocks
	35194.0 (4.2)	200.0 (0.0)	44839 (18)	34020 (0)	2112512	163	3
gen_loopback_tcp_interface_wrapper_top[0].u	100.2 (0.0)	0.0 (0.0)	57 (0)	172 (0)	0	0	0
gen_loopbackServer.i_loopbackServer	100.2 (100.2)	0.0 (0.0)	57 (57)	172 (172)	0	0	0
gen_loopback_tcp_interface_wrapper_top[1].u	99.1 (0.0)	0.0 (0.0)	57 (0)	172 (0)	0	0	0
gen_loopbackServer.i_loopbackServer	99.1 (99.1)	0.0 (0.0)	57 (57)	172 (172)	0	0	0
gen_loopback_tcp_interface_wrapper_top[2].u	101.4 (0.0)	0.0 (0.0)	57 (0)	172 (0)	0	0	0
gen_loopbackServer.i_loopbackServer	101.4 (101.4)	0.0 (0.0)	57 (57)	172 (172)	0	0	0
mac_10_gbe_wrapper_u0	2089.5 (0.0)	0.0 (0.0)	2926 (0)	1378 (0)	0	0	0
mac10gbe_top_u0	2089.5 (0.0)	0.0 (0.0)	2926 (0)	1378 (0)	0	0	0
npap_tcp_udp_wrapper_u0	32624.9 (0.0)	200.0 (0.0)	41429 (0)	31867 (0)	2112512	163	3
npap_tcp_udp_top_u0	32624.9 (2.2)	200.0 (0.0)	41429 (4)	31867 (1)	2112512	163	3
wrapper_ll_ip_tcp_u0	32622.7 (67.2)	200.0 (0.0)	41425 (1)	31866 (196)	2112512	163	3
GEN_MAC_NET_CONV_HHI.i_netLayerConv	1948.2 (9.2)	0.0 (0.0)	785 (15)	2655 (0)	0	0	0
gen_64bitAlign.i_rxAlign	74.2 (74.2)	0.0 (0.0)	13 (13)	145 (145)	0	0	0
gen_rxSyncFifo.i_fifoCDC	1855.4 (1855.4)	0.0 (0.0)	744 (744)	2507 (2507)	0	0	0
i_nlToMacMux	9.3 (9.3)	0.0 (0.0)	13 (13)	3 (3)	0	0	0
g0.i_tcpTxMux	1.7 (1.7)	0.0 (0.0)	3 (3)	3 (3)	0	0	0
gen_WithUdp.i_udp	485.7 (0.8)	0.0 (0.0)	626 (0)	888 (2)	278560	18	0
gen_tcpConnections[0].i_tcp	8480.2 (35.9)	60.0 (0.0)	11408 (12)	7936 (87)	593568	45	1
gen_tcpConnections[1].i_tcp	8460.5 (40.8)	60.0 (0.0)	11357 (12)	7600 (87)	593568	45	1
gen_tcpConnections[2].i_tcp	8497.9 (34.8)	60.0 (0.0)	11376 (12)	7801 (86)	593568	45	1
iBusScheduler8	86.2 (86.2)	0.0 (0.0)	136 (136)	55 (55)	0	0	0
i_internetLayer	2741.3 (6.2)	20.0 (0.0)	3560 (10)	2657 (5)	17920	6	0
i_networkLayer	1851.8 (0.0)	0.0 (0.0)	2173 (0)	2071 (0)	35328	4	0
i_rxLinkResetSync	1.0 (1.0)	0.0 (0.0)	0 (0)	2 (2)	0	0	0
i_txLinkResetSync	1.0 (1.0)	0.0 (0.0)	0 (0)	2 (2)	0	0	0

5.6. Resource Estimates for Lattice Avant-G

The following table shows resources synthesized for Lattice Avant-G LAT-AT-G70ES-3LFG1156C using Lattice Radiant Version 2024.2.1 - instantiating the following design features:

- Ethernet block
- IPv4 block
- UDP block
- 3 instances of TCP blocks
- Diagnostics blocks

Instance	Logic	DRAM	Ripple Logic	Registers	EBR
top	61003	1374	18962	53694	272
npap_tcp_udp_wrapper_u0	57027	1374	18814	52958	270
npap_tcp_udp_top_u0	57027	1374	18814	52958	270
axis_register_udp_u0	224	0	0	425	0
axis_register_udp_u1	219	0	0	413	0
gen_hhi_to_axis_adapter	808	0	0	1258	0
hhiudp_axis_adapter_u0	47	0	0	4	0
npap_configuration_block_u0	349	0	0	338	0
npap_priority_manager_co...	60	0	10	35	0
npap_visibility_block	6756	0	4160	10387	0
npap_diag_capture_u0	5645	0	4160	9620	0
npap_diag_register_u0	1111	0	0	767	0
wrapper_ll_ip_tcp_u0	48518	1374	14644	40098	270
g0.i_tcpTxMux	469	0	0	29	0
gen_WithUdp.i_udp	4884	24	1434	6142	52
gen_tcpConnections[0]....	12646	264	4120	9838	81
gen_tcpConnections[1]....	11406	276	3976	8597	48
gen_tcpConnections[2]....	12769	312	3980	9786	81
iBusScheduler8	93	0	14	31	0
i_internetLayer	2836	498	776	3252	0
i_networkLayer	3358	0	216	2093	8
i_resetStretchStack	14	0	32	34	0
mac_phy_u0	3673	0	148	3066	2
mac_interface_adapter_u0	303	0	0	370	0

5.7. Resource Estimates for Microchip Polarfire

The following table shows resources synthesized for Microchip PolarFire MPF300TS-1FCG1152I using Microchip Libero 2024.2 - instantiating the following design features:

- Ethernet block
- IPv4 block
- UDP block
- 3 instances of TCP blocks
- No diagnostics

Instance	Fabric 4LUT	Fabric DFF	Interface 4LUT	Interface DFF	uSRAM 1K	LSRAM 18K	Math 18x18	Chip Global
npap_tcp_udp_wrapper_u0	60339	31116	6792	6792	122	146	2	12
npap_tcp_udp_top_u0	60339	31116	6792	6792	122	146	2	12
Primitives	13	1	0	0	0	0	0	0
interface_adapter (all)	57	1	0	0	0	0	0	0
wrapper_ll_ip_tcp_u0	60269	31115	6792	6792	122	146	2	12
Primitives	349	195	0	0	0	0	0	0
i_netLayerConv	249	206	0	0	0	0	0	0
i_tcpTxMux	123	2	0	0	0	0	0	0
gen_WithUdp.i_udp	5598	3997	1548	0	24	35	0	0
gen_tcpConnections[0].i	15560	7037	1740	1740	31	37	0	3
gen_tcpConnections[1].i	13287	6649	1068	1068	26	19	2	3
gen_tcpConnections[2].i	15707	7112	1752	1752	35	37	0	3
iBusScheduler8	106	34	0	0	0	0	0	0
i_internetLayer	3717	2995	216	216	6	4	0	1
i_networkLayer	5306	2665	504	504	0	14	0	0

6. Network Administration for NPAP

Managing a network infrastructure commonly involves setting IPv4 addresses, managing TCP sessions, and inspecting hardware statistics. In a standard Linux environment, these tasks are typically performed using command-line tools that interact with the software-based network stack and NIC hardware. With NPAP, similar configurations are possible in two distinct ways.

You can parameterize NPAP using a wide range of compile-time parameters in HDL.

Or, you can use run-time parameterization via NPAP's Python-based software which comprises the NPAP HAL python package and the command-line tool (`npap-admin`). This software interfaces with the NPAP ERD through a wide range of options such as UART, I2C or PCIe, making it easy to adopt NPAP to your use case (see Section 3.4).

NPAP HAL utilizes the NPAP ERD's AXI4-lite register interface to provide a wide range of runtime parameterizations. These include network administration tasks such as setting the IPv4 or MAC address and configuring TCP/UDP sessions, as well as testing and diagnostic functions such as setting up throughput tests with the data generator checker or the Netperf IP core or managing the kernel's diagnostic blocks.

In an NPAP ERD (and most likely within your design as well) the local IPv4 and MAC address is configured on a per NPAP instance and applies to all TCP and UDP sessions within that same NPAP instance.

To demonstrate the simplicity of using NPAP HAL and the Python script npap-admin the following example shows how to configure the IPv4 and MAC address of an NPAP subsystem.

First you write a YAML configuration file as follows:

```Textproto
boards:
  - name: "example-board-name"
    port: "/dev/ttyUSBX"
    baudrate: 115200

    instances:
      - name: "Instance 1"
        local_ip: "192.168.2.12"
        mac: "02:0A:35:08:00:01"
        sessions: []
```

Then, you execute the Python script npap-admin. This example YAML configuration file sets the MAC and IPv4 address of the first NPAP instance on the board:

```Shell
npap_admin <path to YAML configuration file>
```

The correct configuration can then be checked by pinging the configured IPv4 address, since NPAP responds to ICMP echo requests.

Please refer to the documentation for the NPAP ERDs at MLE's Developer Zone: https://devzone.missinglinkelectronics.com/docs/npapdemo/erd_overview/erd_overview.html

7. System Verification via RTL Simulation

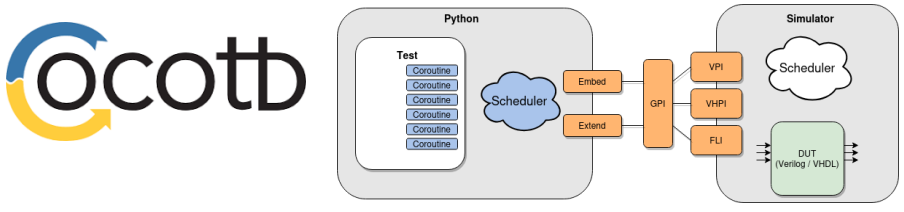
MLE ensures the highest quality and reliability for every NPAP release through a rigorous, multi-layered validation strategy built on a robust Continuous Integration (CI) framework. This automated process validates NPAP from initial simulation through final hardware deployment.

Our Quality Assurance process is documented in the MLE Quality Manual (QAM001 Rev. 13a) and involves:

- **Extensive Simulation-Based Verification**: We conduct comprehensive RTL simulations (testing NPAP-to-NPAP) as well as advanced, system-level co-simulations (testing NPAP-to-software). This automated test suite validates all aspects of protocol behavior, from register access and TCP connection management to complex corner-case scenarios like retransmissions, backpressure, and priority handling.
- **Rigorous Hardware Validation**: Simulation is followed by automated nightly hardware regression testing on a large, physical test farm. We validate NPAP across a wide array of target platforms from all major FPGA vendors, including AMD, Intel, Lattice, and Microchip.
- **Real-World Interoperability Testing**: Hardware tests cover numerous real-world use cases, including board-to-board latency and performance benchmarks, and board-to-PC interoperability against standard Linux/Windows stacks using tools like Netperf. We also perform stress tests under adverse link conditions using the integrated Network Impairment Generator (see Section 3.5).

This comprehensive approach guarantees that NPAP delivers deterministic, high-performance, and reliable operation across all supported target technologies.

The extensive RTL simulation suite comprises the complete RTL for all NPAP components, along with integration with cocotb[17], the open source coroutine-based cosimulation Python testbench environment for VHDL and SystemVerilog RTL.



NPAP RTL simulation and testbench is available upon request. Please contact us!

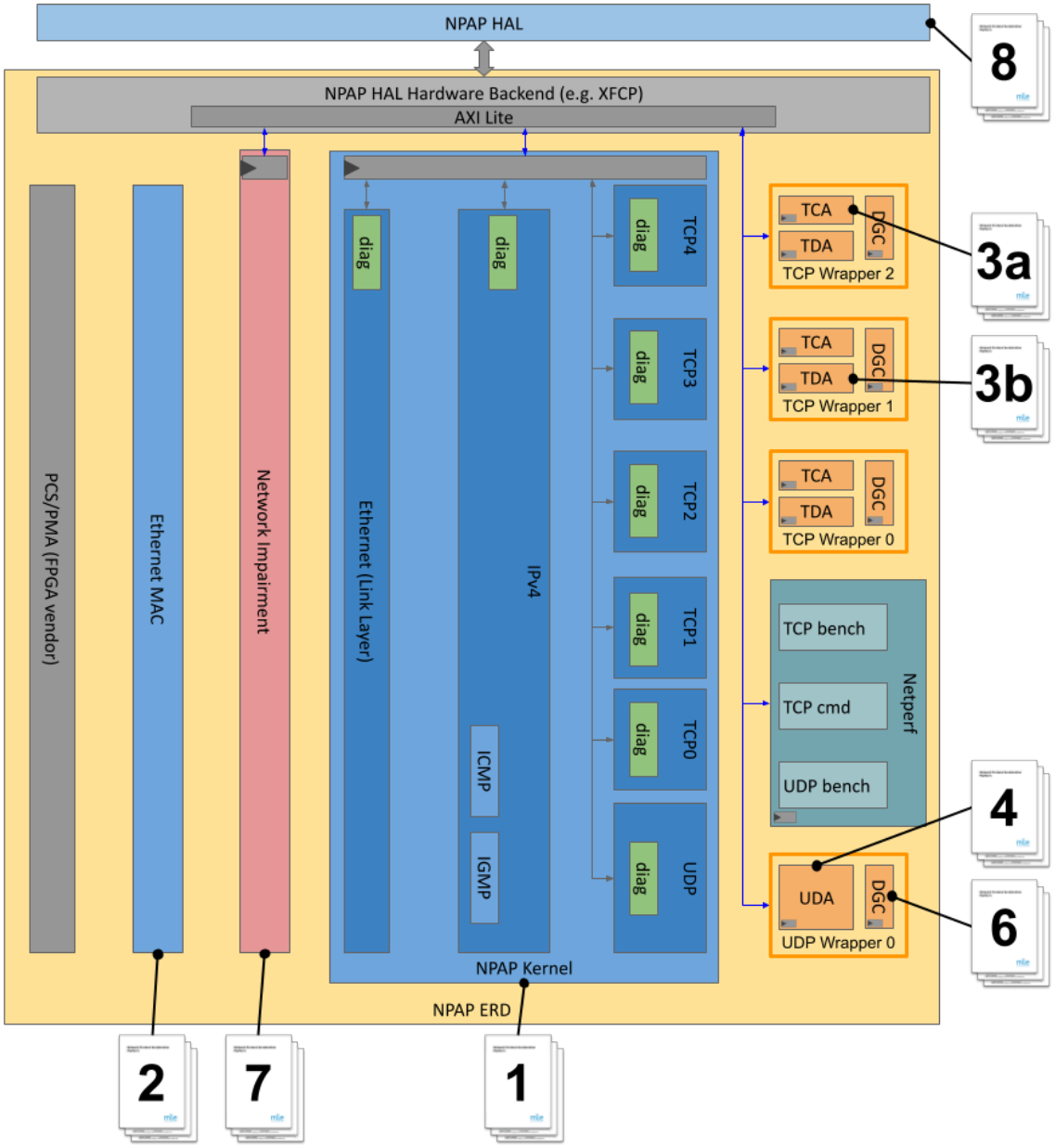
MLE recommends using Siemens EDA Questa One Sim[18].

[17] https://www.cocotb.org/
[18] https://www.siemens.com/en-us/products/ic/questa-one/simulation/questa-one-sim/

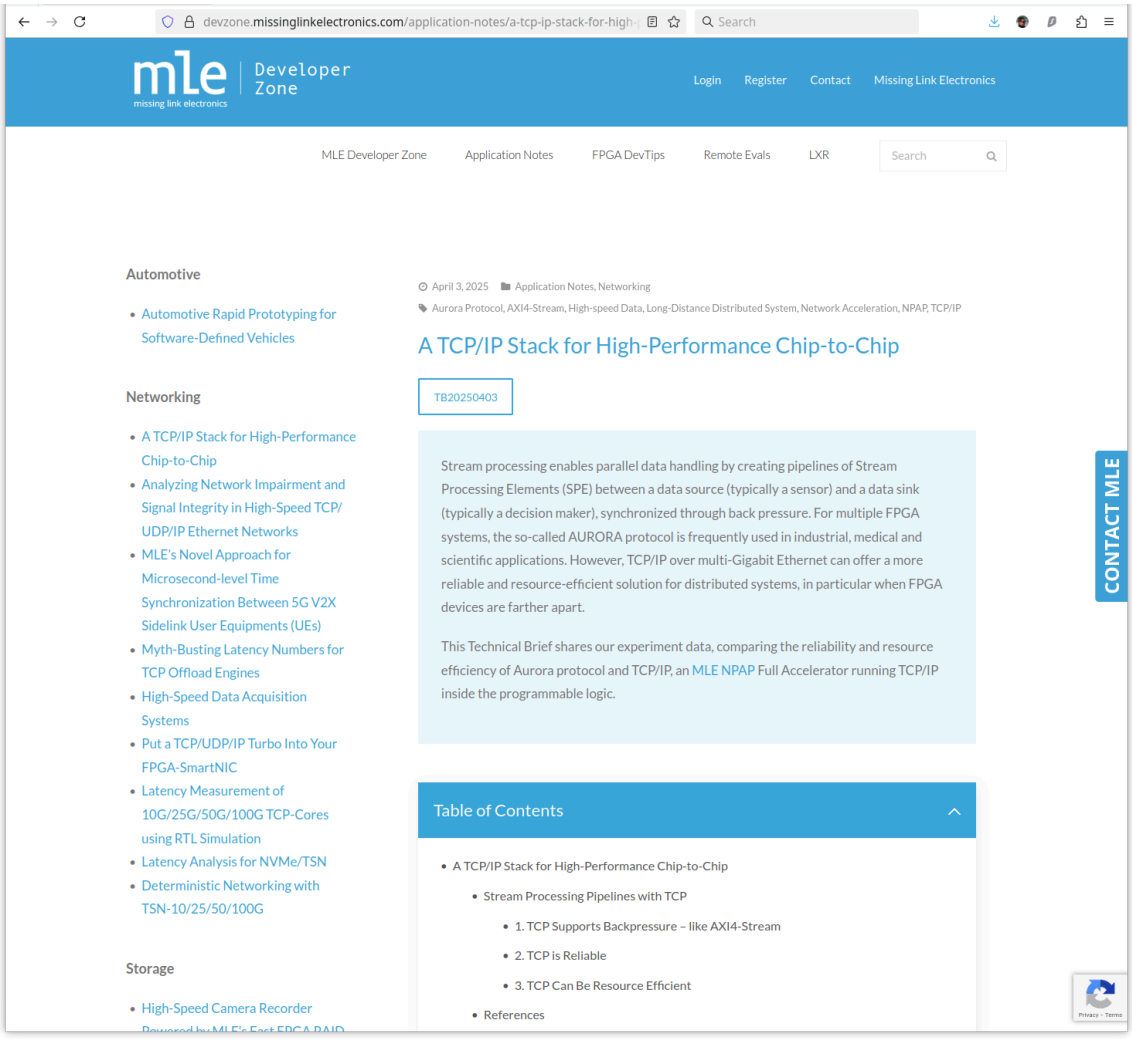
8. Developer Documentation

A comprehensive product design guide and a detailed description of the support IPs and their documentation is available under license:

1. **NPAP Product Guide for the Kernel**
   Documents the features, limitations, the configuration parameters and the interfaces of the NPAP Kernel.
2. **NPAP Product Guide for the LL-MAC**
   Documents the features, configuration parameter and interfaces for the MLE MAC IP Core running at 10GBit/s or 25Gbit/s line rates
3. **NPAP Product Guides for the TCP App**
   a. **TCP Command Application**
      Documents attributes, ports and AXI4-lite registers of the IP Core responsible for configuring a TCP session in the NPAP Kernel.
   b. **TCP Demo Application**
      Documents attributes, ports and AXI4-lite registers of the IP Core responsible for handling the data flow of a TCP session.
4. **NPAP Product Guide for the UDP Demo Application**
   Documents attributes, ports and AXI4-lite registers of the IP Core handling an UDP session.
5. **NPAP Product Guide for the Netperf Control Application**
   Documentation is currently work in progress
6. **NPAP Product Guide for the Data Generator / Checker (DGC)**
   Documents everything necessary to include, configure or interact with the data generator and checker in your design.
7. **NPAP Product Guide for the Impairment Generator Application**
   Documents attributes, ports and AXI4-lite registers of the IP core, which can introduce a variable bit-error rate.
8. **NPAP User Guide for the Hardware Abstraction Layer (HAL)**
   Documents the CLI tool npap-admin and the corresponding python module npap-hal used to configure the NPAP ERD during runtime.


NPAP HAL
NPAP HAL Hardware Backend (e.g. XFCP)
AXI Lite
PCS/PMA (FPGA vendor)
Ethernet MAC
Network Impairment
Ethernet (Link Layer)
IPv4
ICMP
IGMP
diag
TCP4
TCP3
TCP2
TCP1
TCP0
UDP
NPAP Kernel
NPAP ERD
TCA
TDA
DGC
TCP Wrapper 2
TCP Wrapper 1
TCP Wrapper 0
TCP bench
TCP cmd
UDP bench
Netperf
UDA
UDP Wrapper 0
8
3a
3b
4
6
2
7
1

We also suggest to visit MLE's Developer Zone where you can find a growing list of Technical Briefs with expert insights into Networking.



The following pages are a preview of the documents you will receive when purchasing MLE NPAP.

Network Protocol Acceleration Platform

Product Guide

NPAP Kernel

Document Version: 2.9.0
IP Core Version: 2.9.0

2026-02-28, g064de65

Contents

Network Protocol Acceleration Platform

Product Guide

Data Generator Checker

Document Version: 1.1.1
IP Core Version: 2.0.0

2026-02-28, g611e72d

Contents

Network Protocol Acceleration Platform

Product Guide

Impairment Generator

Document Version: 1.1.2
IP Core Version: 1.1.0

2026-02-28, g611e72d

Contents

Network Protocol Acceleration Platform

Product Guide

TCP Command Application

Document Version: 5.0.0
IP Core Version: 5.0.0

2026-02-28, g611e72d

Contents

Network Protocol Acceleration Platform

Product Guide

TCP Demo Application

Document Version: 5.0.0
IP Core Version: 5.0.0

2026-02-28, g611e72d

Contents

Network Protocol Acceleration Platform

Product Guide

UDP Demo Application

Document Version: 1.1.3
IP Core Version: 2.0.1

2026-02-28, g611e72d

Contents

Network Protocol Acceleration Platform

User Guide

NPAP HAL

Document Version: 1.0.0
Release: 2.1.0

2026-02-28, g059f177

Contents

9. Changelog

The following lists MLE's engineering changelog for NPAP. With the release of NPAP v2.2.0 the development cycle has changed from 1.x to 2.x.

Version 2.4.5 uses the least amount of resources and includes no diagnostics.
Version 2.5.0 and onwards requires VHDL-2008. VHDL-2008 is not fully supported in Quartus Prime Standard, for example. Please inquire with any questions regarding tool support.

9.1. NPAP Version 2 Development

- 20260228
  - NPAP Kernel v2.9.0
    - TCP
      - #6668 - rework handling of RTO settings
      - #7734 - add simulation timeout to TCP transmit module to shorten establish time in simulation
  - ERD v3.4.11
    - all
      - bump ERD to v3.4.11
      - #6525 - fix naming by using different names for same signals
      - #6668 - update RTO signals naming between TDA and TCA
      - #7804 - fix psh signal connection between TDA and TCA
    - IP
      - update to NPAP v2.9.0
      - update to TDA v5.0.0
      - update to TCA v5.0.0
- 20260131
  - NPAP Kernel v2.8.1
    - GENERAL
      - #7630 - add support for Quartus Prime Standard Edition
    - TCP
      - #7516 - fix data segment handling if TCP receiver is turned off
      - #7722 - fix TcpCmdStopRecv command does not disable TCP receiver
      - #7723 - improve code readability and quality
  - ERD v3.4.10
    - all
      - bump ERD to v3.4.10
    - altera_storey_peak_source_full_10g_async_npap2
      - #7625 - new ERD 105 for Microsoft Catapult v2 Storey Peak
    - IP
      - update to NPAP v2.8.1
      - update to Netperf v4.0.1

- update to MAC 10GBE v.1.2.1

- 20251031
  - ERD v3.4.7
    - all
      - bump all ERD to v3.4.7
    - amd_zcu111_source_full_25g_sync_async_npap2
      - #7289 - ERD deprecated
    - amd_zcu111_source_full_25g_async
      - #7289 - new ERD 3 variant
    - amd_zcu111_source_full_10g_sync_async_npap2
      - #7289 - enable sync NPAP variant by default
- 20253009
  - NPAP Kernel v2.8.0
    - GENERAL
      - #7246 - fix synthesis error in case G_TCP_CONNECTIONS is set to 0
    - DIAGNOSTICS
      - #7362 - implement diagnostic results reset on snapshot and on window capture
    - TCP
      - #5662 - adapt window scaling algorithm to match the compile time specified buffer sizes as described in RFC 7323
      - #7440 - fix RFC 7323 conformity on window scale option in <SYN,ACK> segment
      - #7457 - fix rare RX window underflow when an unaligned payload nearly fills the window
  - ERD v3.4.6
    - #7246 - fix UDP only ERD generation
    - bump all ERDs to v3.4.6
- 20250820
  - Updated documentation and datasheet
- 20250731
  - NPAP Kernel v2.7.0
    - DIAGNOSTICS
      - #7147 - Implement diagnostics registers for TCP TX RTO min/max value
      - #7148 - Implement memory-mapped reset for diagnostics results reset (either all session or single session only)
  - ERD v 3.4.5
    - bump all ERDs to v3.4.5
- 20250615
  - NPAP Kernel v2.6.1
    - GENERAL
      - #6836 - split FIFO implementation into separate synchronous FIFO and asynchronous FIFO modules
    - TCP
      - #6927 - fix half-closed TCP session handling
  - UDP Demo Application v2.0.1

        - #7298 - fix tvalid being asserted one clock cycle too late in loopback mode
    - Netperf v4.0.0
        - #6836 - split FIFO implementation into separate synchronous FIFO and asynchronous FIFO modules
        - #6927 - update error handling to work with NPAP v2.6.1 update
    - ERD v3.4.4
        - bump all ERDs to v3.4.4
- 20250531
    - NPAP Kernel v2.6.0
        - DIAGNOSTICS
            - #7145 - improve diagnostics: add window control SW read active bit, remove shadow registers for TCP COUNT registers, reduce number of register stages
            - #7146 - change diagnostics window duration load
    - ERD v3.4.3
        - bump all ERDs to v3.4.3
- 20250430
    - ERD v3.4.2
        - bump all ERDs to v3.4.2
        - lattice_avant_g_source_full_10g_sync
            - #6987 - add new Lattice Avant G ERD
- 20250402
    - NPAP Kernel v2.5.0
        - DIAGNOSTICS
            - #6941 - add diagnostics register space for reporting events and counters thereof
    - ERD v3.4.1
        - bump all ERDs to v3.4.1
- 20250320
    - NPAP Kernel v2.4.5
        - GENERAL
            - #7074 - fix internal NPAP version register
- 20250228
    - ERD v3.4.0
        - bump all ERDs to v3.4.0
        - #6745 - add Impairment Generator Application (IPA)
    - IP
        - add IPA v1.0.0
- 20250130
    - ERD v3.3.0
        - bump all ERD to v3.3.0
        - #6291 - update ERD setting and add application loopback
        - amd_u55c_source_full_25g_async_npap4
            - #6521 - add new Alveo U55C ERD
    - IP
        - update DGC to v2.0.0
        - update NCA to version 1.0.4

        - update UDA to version 2.0.0
- 20241220
    - NPAP Kernel v2.4.4
        - GENERAL
            - #6731 - add fifo variant for Microchip FPGA
    - ERD v3.2.2
        - microchip_mpf300_eval_kit_source_full_10g_sync
            - #6731 - add HHI fifo variant for Microchip
        - intel_axe5_eagle_source_full_10g_async
            - #6685 - add new Agilex 5 ERD
- 20241130
    - ERD v3.2.1
        - amd_te0950_source_full_25g_async
            - #6734 - update ERD 8 (TE0950) to non es silicon
- 20241028
    - NPAP Kernel v2.4.3
        - TCP
            - #6479 - fix possible TCP session lock in half closed state if priority manager is enabled
            - #6613 - increase internal debug support by adding new debug signals
    - ERD v3.2.1
        - #5739 - add new register to top level register space
        - amd_zcu102_source_full_10g_sync
            - #5739 - add top level info block to ER
        - microchip_mpf300_eval_kit_source_full_10g_sync
            - #6581 - add XFCP
- 20240930
    - NPAP Kernel v2.4.2
        - TCP
            - #6593 - fix consistency of TCP TX ram type implementation
            - #6645 - fix session stall corner case when the congestion window is full, the last packet does not get ACKed, and no more packets are written into the TX buffer
            - #6649 - fix session stall corner case when an ACK for a retransmitted packet is lost, and the TX buffer on the ACK side is empty
    - ERD v3.1.6
        - #6119 - change UDP default datagram size to 1472 byte
        - amd_zcu111_source_full_25g_sync_async_npap2
            - #6634 - fix DGC default setting
        - amd_zcu111_source_full_100g_sync
            - #6634 - fix DGC default setting
        - IP
            - update TDA to v4.0.4
            - update MAC to v1.2.0
- 20240813

    - NPAP Kernel v2.4.1
        - GENERAL
            - #6590 - rewrite FIFO constrains from xdc to tcl
- 20240731
    - NPAP Kernel v2.4.0
        - GENERAL
            - #5878 - update generic fifo constraints
            - #6512 - multiple internal coding style fixes and signal clean-ups
            - #6592 - fix npap_configuration_block register reset
        - TCP
            - #4682 - rework of receive window handling
            - #6494 - change window update calculation naming
            - #6513 - add support for receive side Keep-Alive segments
            - #6514 - add support for receive side Zero-Window-Probe
    - ERD v3.1.5
    - IP
        - update DGC to v1.4.0
- 20240630
    - ERD v3.1.4
        - microchip_mpf300_eval_kit_source_full_10g_sync
            - #5877 - replace reset stretch and sync module
- 20240430
    - NPAP Kernel v2.3.4
        - GENERAL
            - #6413 - fix bus scheduler early release hang
    - ERD v3.1.3
        - #6248 - DGC modes are configurable in top level
        - amd_zcu102_source_full_1g_sync_async_npap2
            - #6063 - add 1G ERD
        - amd_zcu102_source_full_1g_10g_sync_async_npap4
            - #6063 - deprecate ERD
- 20240331
    - ERD v3.1.2
    - amd_te0950_source_full_25g_async
        - add TE0950 25 ERD
- 20240311
    - NPAP Kernel v2.3.3
        - GENERAL
            - #6399 - fix possible UDP checksum mismatch
            - #6377 - fix UDP only design
- 20240229
    - NPAP Kernel v2.3.2
        - GENERAL
            - #4909 - fix bus scheduler fairness
            - #5987 - fix G_PRIORITY_WIDTH synthesis error
            - #6243 - fix G_ENABLE_UDP = 0 timing loop
    - ERD v3.1.1

        - #6087 - fix UDP session - 1 configuration error
        - #6093 - fix Netperf UDP test
        - #6201 - fix default value settings
        - #6235 - support no DGC operation
        - #6282 - support UDP and TCP only operation
        - #6298 - fix local port not assigned any more
        - amd_au200_source_full_25g_async
            - add AU200 25G ERD
        - amd_zcu111_source_full_100g_sync
            - add ZCU111 100G ERD
    - IP
        - update Netperf to v3.0.2
        - update DGC to v1.1.5
        - update TCA to v4.0.3
- 20240101
    - NPAP Kernel v2.3.1
        - GENERAL
            - #6005 - update IP packaging to support Versal FPGA
    - ERD v3.1.0
        - microchip_mpf300_eval_kit_source_full_10g_sync
            - build with obfuscated MAC license
            - +1 commit to ERD release tag
        - intel_npac_ketch_source_full_10g_async_npap2
            - add NPAC Ketch 10G ERD
        - intel_npac_ketch_source_full_10g_async_npap3
            - deprecated
    - IP
        - update Netperf to v3.0.1
        - update MAC 10/25 G to 1.1.2 (some ERDs)
        - update DGC to v1.1.4
        - update NCA to v1.0.2
        - update TCA to v4.0.2
        - update TDA to v4.0.2
        - update UDA to v1.1.1
- 20231129
    - NPAP Kernel v2.3.0
        - GENERAL
            - #5743 - add AXIL-Lite control and status interface (NPAP Configuration Block)
            - #5948 - update Priority Manager to new AXI4-Lite structure
- 20231031
    - ERD v3.0.5
        - #5891 - update generic and clock structure
        - microchip_mpf300_eval_kit_source_full_10g_sync
            - add MPF300 10G ERD
- 20230930
    - NPAP Kernel v2.2.5
        - GENERAL

- #5963 - update timeouts to be clock depended

- 20230919
  - NPAP Kernel v2.2.4
    - TCP
      - #5954 - harden TCP against lost ACK
- 20230831
  - NPAP Kernel v2.2.3
    - GENERAL
      - #5880 - remove unused reset stretch modules
  - ERD v3.0.3
    - amd_zcu102_source_full_1g_10g_sync_async_npap4
      - #5899 - fix AXI-L connectivity
    - amd_zcu111_source_full_25g_sync_async_npap2
      - #5899 - fix AXI-L connectivity
      - #5908 - increase system clock
    - intel_npac_ketch_source_full_10g_async_npap3
      - #5880 - update constraints and reset structure
- 20230731
  - NPAP Kernel v2.2.2
    - TCP
      - #5805 - fix out of order received ACK handling
  - ERD v3.0.2
    - amd_zcu102_source_full_1g_10g_sync_async_npap4
      - #5784 - add zcu102 1G / 10G ERD
    - amd_zcu111_source_full_25g_sync_async_npap2
      - #5816 - add zcu111 25G ERD
- 20230630
  - NPAP Kernel v2.2.1
    - UDP
      - #5707 - fix UDP meta data handling
  - ERD v3.0.1
    - amd_zcu102_source_full_10g_sync
      - #5814 - fix generic names
  - IP
    - #5815 - update NPAP v2.2.1
    - #5859 - update DGC v1.1.2
    - #5860 - update NCA v1.0.1
    - #5861 - update Netperf v3.0.0
- 20230531
  - NPAP Kernel v2.2.0
    - GENERAL
      - #5652 - internal code merge and structure update (technically NPAP v2.2.0 is identically with v2.1.2, the code history has a different merge / rebase history)
  - ERD v3.0.0
    - amd_zcu102_source_full_10g_sync

        - #4775 - add ZCU102 10G ERD
      - intel_npac_ketch_source_full_10g_async_npap3
        - #5708 - add NPAC Ketch 10G ERD
    - IP
      - #5760 - update Netperf v2.0.0
      - #5761 - update UDA v1.1.0
      - #5763 - update DGC v1.1.1
- 20230411
  - NPAP Kernel v2.1.2
    - TCP
      - #5562 - add support for partially ACKed packages
- 20230331
  - NPAP Kernel v2.1.1
    - version changes not included
      - sync with NPAP 1.10.1
    - IPGUI
      - #5560 - remove AXI4-S MAC TDATA width setting from customisation GUI
      - #5569 - buffer size: make IP customisation GUI default the same as HDL default
    - GENERAL
      - #5668 - clock are now associated to the AXI4-L interface
- 20230306
  - NPAP Kernel v2.1.0
    - TCP
      - #5554 - fix signal overflow on TCP transmit controller
      - #5555 - fix signal overflow on TCP receive buffer calculation signal
- 20220822
  - NPAP Kernel v2.0.1
    - version changes not included
      - NPAP 1.9.2 to 1.10.0
    - GENERAL
      - #4836 - add priority scheduler
- 20220715
  - NPAP Kernel v2.0.0
    - version changes not included
      - NPAP 1.9.2 to 1.10.0
    - GENERAL
      - #2854 - remove NPAP application CDCs from code base
      - #3944 - NPAP Performance Enhancement and Clean Up (parts)
      - #4398 - remove 8bit data path
      - #4399 - remove dma code fragments
      - #4834 - add QoS interfaces and generics
      - #4835 - add register interface for priority settings
    - ETHERNET
      - #4601 - change MAC interface to standard 128 Bit AXIS

9.2. NPAP Version 1 Development

Not recommended for new design starts!

- 1.10.1 (20230331)
  - TCP
    - #5554 - fix overflow on allow payload size register
    - #5557 - fix minimum value for G_TCP_RX_MAX_FRAME_NUMBER and G_TCP_TX_MAX_FRAME_NUMBER
    - #5639 - fix TCP window calculation after window scale is set
- 1.10.0 (20220930)
  - TCP
    - #4389 - add TCP Cmd TcpCmdSetTcpPsh
    - #4888 - fix TCP session handling with same destination port
    - #4916 - fix TCP splitter generating overlong packages
    - #5007 - fix bug where changing RTO values could lead to TCP Cmd interface to hang
- 1.9.2 (20220718)
  - GENERAL
    - #4805 - TCP session do not transfer data reliably on first connection (Microchip only)
- 1.9.1 (20220503)
  - GENERAL
    - #4741 - fix wrong license header
- 1.9.0 (20220430)
  - TCP
    - #4700 - add per TCP session configurable buffer sizes and configurable MSS
  - UDP
    - #4700 - add new parameter G_TX_MAX_DATAGRAM_SIZE
- 1.8.0 (20220331)
  - ETHERNET
    - #4606 - add padding for frames smaller than 60 Bytes
  - TCP
    - #4609 - remove maximum TCP Session limit
  - UDP
    - #4557 - fix length assignment in UDP Interface Adapter
- 1.7.1 (20211220)
  - GENERAL
    - #3951 - add missing reset signal
    - #4412 - fix buffer generic ranges
    - #4416 - fix subnet mask assignment for no UDP setup
  - TCP
    - #4367 - fix tcp space available calculation for buffer sizes above 64KB
- 1.7.0 (20211101)
  - GENERAL
    - #3951 - remove unused altera_attribute

        - #4139 - allow set off RTO values in TCP session establish state
- 1.6.2 (20210913)
    - GENERAL
        - #3981 - remove unused altera_attribute
- 1.6.1 (20210801)
    - UDP
        - #3969 - fix use of ceil function
- 1.6.0 (20210701)
    - GENERAL
        - #3916 - rename / fix generic names
- 1.5.3 (20210301)
    - UDP
        - #3549 - fix bus scheduler bus handling
- 1.5.2 (20201101)
    - TCP
        - #3091 - fix ack command fifo interface handling
- 1.5.1 (20201001)
    - ARP
        - #2924 - fix handshake between ARP and bus scheduler
    - TCP
        - #2539 - fix possible wrong MAC usage in multi session configuration
        - #2923 - fix data acceptance criteria from application
- 1.5.0 (20200901)
    - GENERAL
        - #2254 - change 100ms clock not generated inside NPAP wrapper
        - #2848 - change application clocking by removing TCP and UDP application clock
    - IP
        - #2239 - change default configuration for IPv4 filter, now enabled
    - DHCP
        - #2858 - change increase DHCP usability by adding try counter, valid signal and new timeout behaviour
        - #2737 - fix DHCP lease calculation
    - TCP
        - #2703 - fix payload length update behaviour in transmit controller
        - #2727 - fix acknowledgment number update behaviour in transmit controller
        - #2831 - fix FSM handshake in transmit controller which could lead to TCP session freeze
- 1.4.9 (20200702)
    - IGMP
        - #2704 - change disable IGMP per default
        - #2496 - fix compiler warning about latch implementation
    - TCP
        - #2706 - change TCP tx data path in asynchronous mode
        - #2627, #2688, #2693, #2701, #2702 - fix TCP tx splitter, rework after multiple bugs
        - #2692 - fix used TCP tx splitter generic

        - #2711 - fix transmit controller fsm reset generation
- 1.4.8 (20200430)
    - TCP
        - #2477 - fix TCP retransmission buffer delete handling
- 1.4.7 (20200331)
    - TCP
        - #2180 - change TCP tx splitter to work with byte granularity
        - #2097 - fix TCP multi session reset synchronization
        - #2339 - fix TCP application reset clock domain crossing
- 1.4.6 (20200303)
    - GENERAL
        - #2469 - fix default gateway IPv4 address usage
    - TCP
        - #2468 - add register stage to TCP TX application interface to ease timing on Virtex 6
- 1.4.5 (20200220)
    - GENERAL
        - #2449 - fix Xilinx ISE 14.7 workflow
- 1.4.4 (20200213)
    - TCP
        - #2086 - fix retransmission lockup
        - #2112 - fix fsm lockup in transmit controller
- 1.4.3 (20200116)
    - ARP
        - #2179 - fix ARP cache IPv4 address lookup
- 1.4.2 (20200113)
    - GENERAL
        - #2294 - change delivered IP XACT constraint file
- 1.4.1 (20200107)
    - TCP
        - fix data type and IP core GUI handling of TCP sequence number initialization
- 1.4.0 (20191126)
    - GENERAL
        - #1930 - add AXI4-Stream TCP application interface
        - #2151 - add customized block design configuration gui
        - #2166 - add example constrain file to IP XACT packaging
        - #2148 - fix block design gui NPAP name generic
    - TCP
        - #1939 - fix space available calculation which lead to duplicated data beat
        - #2181 - fix TCP tx splitter timeout
- 1.3.0 (20191002)
    - GENERAL
        - #1682 - add IP XACT TCP/UDP wrapper and packaging
- 1.2.0 (20190920)
    - GENERAL
        - #2075 - add packaging infrastructure for TCP source code release
- 1.1.0 (20190705)

    - ARP
        - #1719 - add new ARP cache size generic
        - #1698 - fix internal race condition during initialization
    - UDP
        - #1720 - fix retry mechanism on failed ARP lookup
- 1.0.0 (20181023)
    - GENERAL
        - #1320 - add NPAP to MLE Vivado build toolchain
        - #1326 - add IP XACT UDP wrapper and packaging
        - #1133 - fix bus scheduler grant timeout
    - ETHERNET
        - #1323 - add 64 and 128 bit AXI4-Stream interface option
    - IP
        - #1328 - fix IPv4 header decoder data valid calculation for payloads of 1 to 3 byte
    - UDP
        - #1322 - add AXI4-Stream UDP application interface
        - #1324 - add new generic to disable UDP TX aligner
        - #1132 - fix UDP tx fifo write count calculation
        - #1133 - fix UDP header encoder fsm timeout for ARP
        - #1327 - fix UDP throughput bottleneck for payload sizes less than 100 byte
        - #1330 - fix corrupt UDP data multiplexing for zero TCP connections

10. Detailed Protocol Support (RFC1122 excerpt)

10.1. Ethernet Layer

Feature	Section	Must	Must not	Implemented
Send Trailers by default without negotiation	2.3.1		x	x
ARP	2.3.2			
Flush out-of-date ARP cache entries	2.3.2.1	x		(x)
Prevent ARP floods	2.3.2.1	x		(x)
Ethernet and IEEE 802 Encapsulation	2.3.3			
Host able to:	2.3.3			
Send & receive RFC-894 encapsulation	2.3.3	x		x
Send K1=6 encapsulation	2.3.3		x	
Use ARP on Ethernet and IEEE 802 nets	2.3.3	x		x
Link layer report b'casts to IPv4 layer	2.4	x		
IPv4 layer pass TOS to link layer	2.4	x		
No ARP cache entry treated as Dest. Unreach.	2.4		x	x

10.2. IPv4 & ICMP Layer

Feature	Section	Must	Must not	Implemented
Implement IPv4 and ICMP	3.1	x		x
Handle remote multihoming in application layer	3.1	x		x
Meet gateway specs if forward datagrams	3.1	x		-
Silently discard Version != 4	3.2.1.1	x		x
Verify IPv4 checksum, silently discard bad dgram	3.2.1.2	x		x
Addressing:				
Subnet addressing (RFC-950)	3.2.1.3	x		-
Src address must be host's own IPv4 address	3.2.1.3	x		x
Silently discard datagram with bad dest addr	3.2.1.3	x		x
Silently discard datagram with bad src addr	3.2.1.3	x		x
Support reassembly	3.2.1.4	x		-

TOS:				
Allow transport layer to set TOS	3.2.1.6	x		-
TTL:				
Send packet with TTL of 0	3.2.1.7		x	x
Discard received packets with TTL > 2	3.2.1.7		x	-
Allow transport layer to set TTL	3.2.1.7	x		-
Fixed TTL is configurable	3.2.1.7	x		x
IPv4 Options:				
Allow transport layer to send IPv4 options	3.2.1.8	x		-
Pass all IPv4 options rcvd to higher layer	3.2.1.8	x		-
IPv4 layer silently ignore unknown options	3.2.1.8	x		x
Silently ignore Stream Identifier option	3.2.1.8b	x		x
Source Route Option:				
Originate & terminate Source Route options	3.2.1.8c	x		-
Datagram with completed SR passed up to TL	3.2.1.8c	x		-
Build correct (non-redundant) return route	3.2.1.8c	x		-
Send multiple SR options in one header	3.2.1.8c	x		-
ROUTING OUTBOUND DATAGRAMS:				
Use address mask in local/remote decision	3.3.1.1	x		x
Operate with no gateways on conn network	3.3.1.1	x		x
Maintain "route cache" of next-hop gateways	3.3.1.2	x		-
If no cache entry, use default gateway	3.3.1.2	x		x
Support multiple default gateways	3.3.1.2	x		-
Able to detect failure of next-hop gateway	3.3.1.4	x		-
Ping gateways continuously	3.3.1.4		x	-
Ping only when traffic being sent	3.4.1.4	x		-
Ping only when no positive indication	3.3.1.4	x		-
Switch from failed default g'way to another	3.3.1.5	x		-
Manual method of entering config info	3.3.1.6	x		-
REASSEMBLY and FRAGMENTATION:				
Able to reassemble incoming datagrams	3.3.2	x		-
Transport layer able to learn MMS_R	3.3.2	x		-
Send ICMP Time Exceeded on reassembly timeout	3.3.2	x		-
Pass MMS_S to higher layers	3.3.3	x		-
MULTIHOMING:				
Allow application to choose local IPv4 addr	3.3.4.2	x		x
BROADCAST:				
Broadcast addr as IPv4 source addr	3.2.1.3		x	-
Recognize all broadcast address formats	3.3.6	x		-
Use IPv4 b'cast/m'cast addr in link-layer b'cast	3.3.6	x		-

INTERFACE:				
Allow transport layer to use all IPv4	3.4	x		-
mechanisms	3.4	x		-
Pass interface ident up to transport layer	3.4	x		-
Pass all IPv4 options up to transport layer				
Transport layer can send certain ICMP	3.4	x		-
messages	3.4	x		-
Pass spec'd ICMP messages up to transp. layer	3.4	x		-
Include IPv4 hdr+8 octets or more from orig.				
ICMP:	3.2.2.6	x		-
Echo server	3.2.2.6	x		x
Echo client	3.2.2.6	x		x
Use specific-dest addr as Echo Reply src	3.2.2.6	x		x
Send same data in Echo Reply	3.2.2.6	x		x
Pass Echo Reply to higher layer	3.2.2.6	x		-
Reverse and reflect Source Route option	3.3.6	x		-
Use IPv4 b'cast/m'cast addr in link-layer				
b'cast				

10.3. TCP Layer

Feature	Section	Must	Must not	Implemented
Push flag				
ESEND call can specify PUSH	4.2.2.2			-
If cannot: sender buffer indefinitely	4.2.2.2		x	
If cannot: PSH last segment	4.2.2.2	x		x
Window				
Treat as unsigned number	4.2.2.3	x		x
Robust against shrinking window	4.2.2.16	x		-
Sender probe zero window	4.2.2.17	x		(x)
Allow window stay zero indefinitely	4.2.2.17	x		x
Sender timeout OK conn with zero wind	4.2.2.17		x	x
TCP Options				
Receive TCP option in any segment	4.2.2.5	x		x
Ignore unsupported options	4.2.2.5	x		x

Cope with illegal option length	4.2.2.5	x		-
Implement sending & receiving MSS option	4.2.2.6	x		x
Send-MSS default is 536	4.2.2.6	x		x
Calculate effective send seg size	4.2.2.6	x		x
TCP Checksums				
Sender compute checksum	4.2.2.7	x		x
Receiver check checksum	4.2.2.7	x		x
Use clock-driven ISN selection	4.2.2.9	x		x
Opening Connections				
Support simultaneous open attempts	4.2.2.10	x		-
SYN-RCVD remembers last state	4.2.2.11	x		-
Passive Open call interfere with others	4.2.2.18		x	-
Function: simultan. LISTENs for same port	4.2.2.18	x		
Ask IPv4 for src address for SYN if necc.	4.2.3.7	x		x
Otherwise, use local addr of conn.	4.2.3.7	x		x
OPEN to broadcast/multicast IPv4 Address	4.2.3.14		x	-
Silently discard seg to bcast/mcast addr	4.2.3.14	x		-
Closing Connections				
Inform application of aborted conn	4.2.2.13	x		x
In TIME-WAIT state for 2 x MSL seconds	4.2.2.13	x		x
Retransmissions				-
Jacobson Slow Start algorithm	4.2.2.15	x		-
Jacobson Congestion-Avoidance algorithm	4.2.2.15	x		-
Karn's algorithm	4.2.3.1	x		-
Jacobson's RTO estimation alg.	4.2.3.1	x		-
Exponential backoff	4.2.3.1			
Generating ACK's:				
Process all Q'd before send ACK	4.2.2.20	x		x
Receiver SWS-Avoidance Algorithm	4.2.3.3	x		-

MLE Contact Info

Missing Link Electronics, Inc.
2880 Zanker Road, Suite 203
San Jose, CA 95134

Missing Link Electronics GmbH
Industriestraße 10
89231 Neu-Ulm, Germany

Missing Link Electronics GmbH
Ringstrasse 66
12105 Berlin, Germany

Email: sales-web@mlecorp.com

http://www.missinglinkelectronics.com

Founded in 1949, the German Fraunhofer-Gesellschaft undertakes applied research of direct utility to private and public enterprise and of wide benefit to society. With a workforce of over 23,000, the Fraunhofer-Gesellschaft is Europe's biggest organization for applied research, and currently operates a total of 67 institutes and research units. The organization's core task is to carry out research of practical utility in close cooperation with its customers from industry and the public sector.

Fraunhofer HHI was founded in 1928 as "Heinrich-Hertz-Institut für Schwingungsforschung" and joined in 2003 the Fraunhofer-Gesellschaft as the "Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut". Today it is the leading research institute for networking and telecommunications technology, "Driving the Gigabit Society" .