MLE Technical Brief 20240819
Network Protocol Accelerator Platform

A stand-alone TCP/UDP/IP Stack Full-Accelerator Subsystem allowing communication at full line rate and low latency

Version:	2.4.1
Date:	August 2024
Authors:	Ulrich Langenbach, Thomas Glatte, Endric Schubert, F. Maximilian Sokol

Features

MLE’s Network Protocol Accelerator Platform (NPAP) for 1/2.5/5/10/25/40/50/100 Gigabit Ethernet is a TCP/UDP/IP network protocol Full-Accelerator subsystem which instantiates the standalone 128 bit TCP/IP Stack technology from German Fraunhofer Heinrich-Hertz-Institute (HHI). This Fraunhofer HHI 10 GbE TCP/IP Stack was designed for embeddable FPGA and ASIC system solutions and offers the following features:

- Interface to 1 / 2.5 / 5 / 10 / 25 / 40 / 50 / 100 Gigabit Ethernet[1]
- Full-duplex with 128 bit wide bidirectional datapath
- Full line rate up to 70 Gbps per NPAP instance
- Full line rate >100 Gbps per individual TCP session in ASIC
- Low round trip time NPAP-to-NPAP 700 nanoseconds for 100 Bytes RTT

Designed for maximum flexibility, NPAP implements in programmable logic the most common 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

ICMP Diagnostic protocol to validate connections

IGMP Enables joining of multicast groups (optional)

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

Originally targeted to deliver close to the theoretical line-rate of 10 Gigabit Ethernet, a 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 2.4.1) for new ASIC and AMD/Xilinx Versal, Ultrascale+, Ultrascale, Intel Stratix-10, Agilex-5E and Agilex 7, Microchip PolarFire and Lattice Avant G/X development, includes many recent resource optimizations plus timing optimizations focused on pipelines in FPGA, including 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.

Applications

NPAP enhances your real-time application with a leading fast data connectivity. The powerful architecture of the underlying TCP/UDP/IP Stack allows it to transfer data at line-rate with low processing latency without using any CPUs in the data path. The widespread TCP/IP and/or UDP/IP communication protocol suite using industry standard network infrastructure addresses a wide-range of applications:

- FPGA-based SmartNICs
- High-Bandwidth Security with FPGA-based Smart Data Diodes
- In-Network Compute Acceleration (INCA)
- Hardware-only implementation of TCP/IP in FPGA
- PCIe Long Range Extension [2]
- 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/IP 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 alternative to lwIP or Linux on embedded CPU

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

IP Core Description

High performance programmable logic based, standalone TCP/IP stack featuring transparent handling of complete TCP/IP and UDP protocol tasks, e.g. packet encoding, packet decoding, acknowledge generation, link supervision, timeout detection, retransmissions and fault recovery. Complete automatic connection control including tear up and tear down. Compute and manage retransmission timers as in RFC 6298[3]. Transparent checksum generation and checksum checking, integrated flow control. RFC 9293[4] compatibility (TCP/IP 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 is optimized to ensure the best bandwidth-delay product performance for your application. The IP core described herein is easy to port to FPGA and ASIC target platforms.

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) Intel 10G / 25G Ethernet FPGA IP Microchip PolarFire FPGA 10G Ethernet (UG0727)
Supported protocols (Hardware based)	Ethernet, ARP, IPv4, ICMPv4, IGMPv4, UDP & TCP, DHCP
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

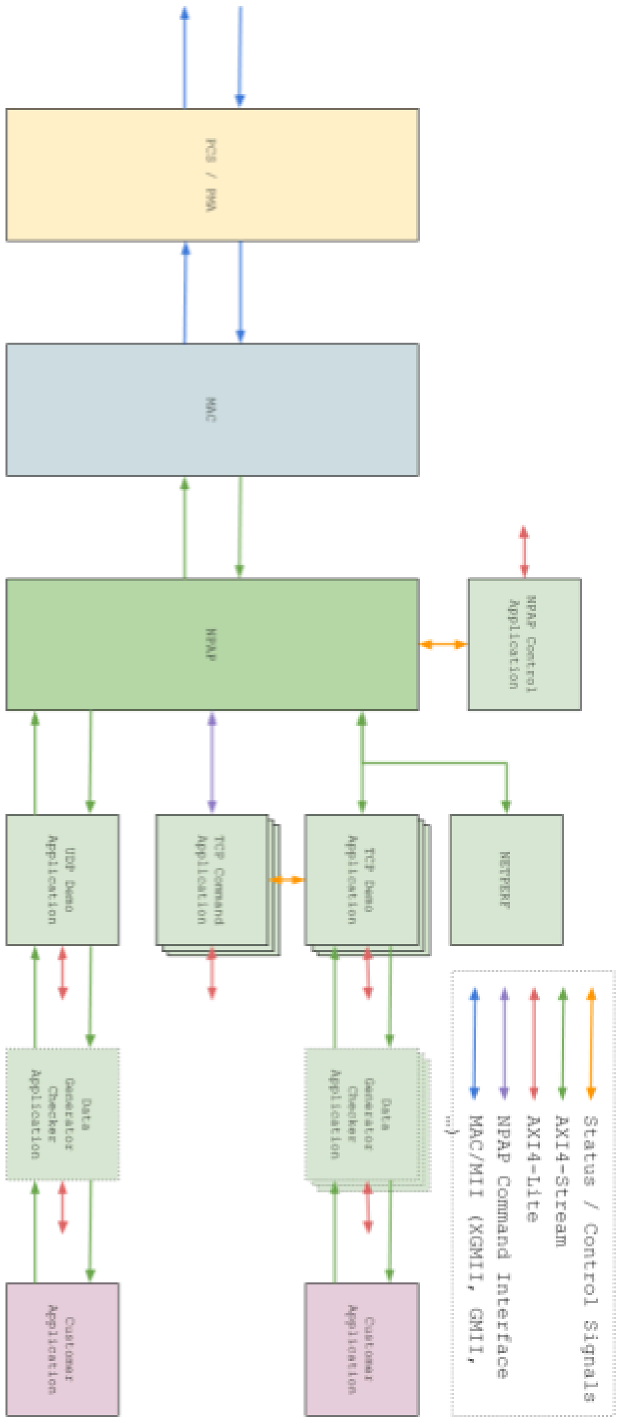
[3] https://datatracker.ietf.org/doc/html/rfc6298
[4] https://www.rfc-editor.org/info/rfc9293

Interface to application	Datapath via AXI4-Stream 128-bit and separate custom TCP command interface
Supported FPGAs	Complete stack uses generic VHDL code 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 Achronix Speedster 7t[1] Intel Cyclone IV series Intel Cyclone 10 GX series Intel Stratix V Intel Stratix 10 GX series Intel Agilex 5 D, E Series[1] Intel Agilex 7 F, I, M Series Lattice Avant G, X[1] Microchip Polarfire and PolarFire SoC
Performance	Over 60 Gbps line rate for single TCP/IP session (depending on clock rate, see below) Less than 800 ns “door-to-door” RTT (depending on clock rate and packet size)

[1] R&D Work-in-progress or on product release road-map

Implementation Details

Highly modular implementation using standard AXI4 interfaces to support RTL synthesis flows for various FPGA vendors and ASIC RTL design flows.



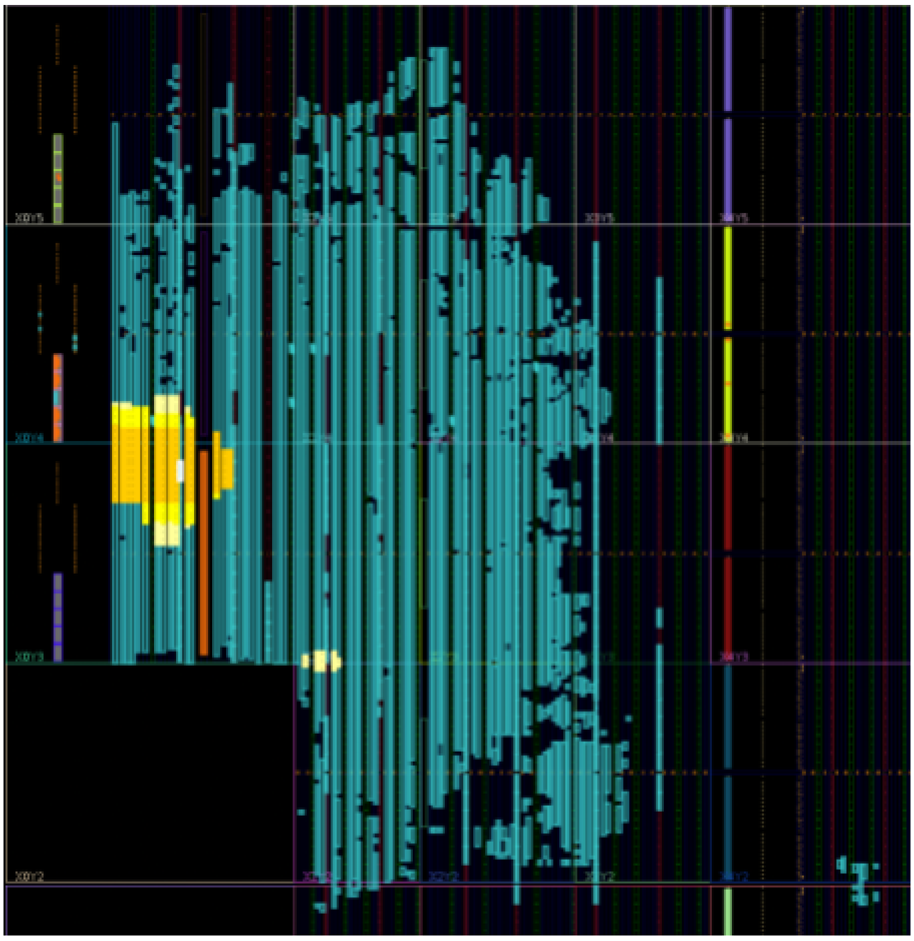
For instantiating NPAP you have two options:

- Asynchronous clocking - where the Ethernet MAC block is in a different clock domain than the other functions (IPv4, TCP, UDP block).

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

Both are equally good, but depending on your overall design, the device utilization, your performance requirements, etc., one may be better than the other. Just giving you choices...

Starting with Release 2.0 RTL is optimized for FPGA pipelining such as Intel Hyperflex or AMD/Xilinx IMux. Starting with Release 2.0 RTL has further been optimized for Intel HyperFlex and Intel HyperFlex2.



IP Core Deliverables

Deliverables include IEEE 1685 IP-XACT packages including “NPAP Support IP” blocks plus non-IP-XACT FPGA reference design project. The following design blocks are part of our delivery package:

- Low-Latency Ethernet MAC for 10G/25G
- NPAP core with IPv4, TCP, UDP
- NPAP Support IP
  - Data Generator Checker
    (great for system-level testing and for performance tuning)
  - NPAP Control Application
  - Netperf Control Application
  - TCP Command Application
- TCP Demo Application
- UDP Demo Application
- Evaluation Reference Design (as FPGA Design Project Archive)

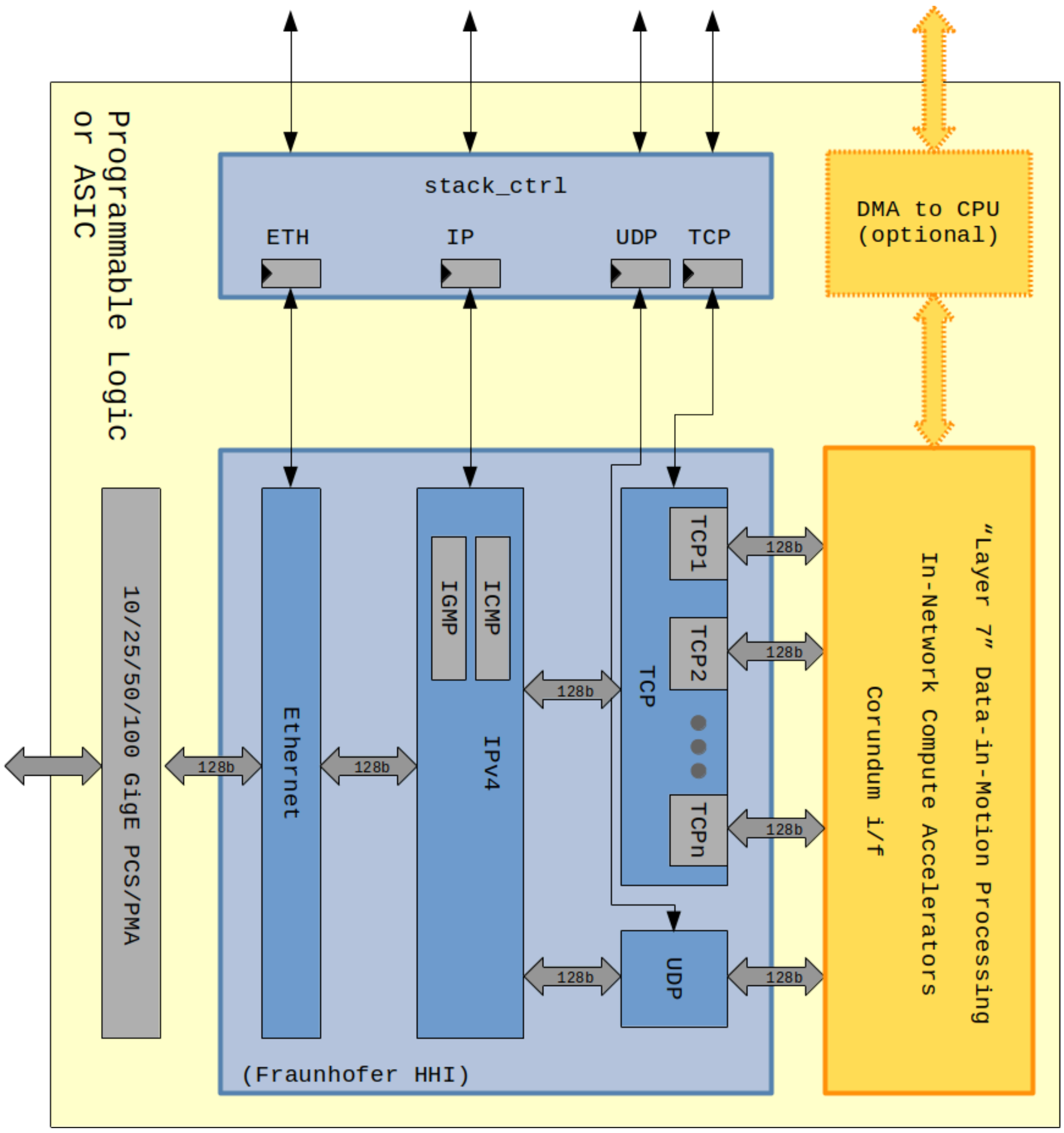
NPAP is available as RTL source code or as a netlist suitable for synthesis by 3rd party and FPGA vendor tools:

Version	AMD/Xilinx	Intel	Microchip
NPAP 2.4.1	Vivado 2022.2	Quartus 22.4	Libero 2023.1
NPAP 1.10.1	Vivado 2018.3	Quartus 22.1	Libero 2021.2

Architecture Choices

NPAP implements a full accelerator, hence all network protocol processing is running as digital logic. NPAP does not instantiate any “soft” CPUs, nor does NPAP rely on external CPUs to fully function. This gives NPAP a very deterministic low latency and makes performance very scalable over circuit area / FPGA resources.

The following block diagram highlights key implementation aspects:


Programmable Logic or ASIC
stack_ctrl
ETH
IP
UDP
TCP
DMA to CPU (optional)
10/25/50/100 GigE PCS/PMA
128b
Ethernet
IGMP
ICMP
IPv4
TCP
TCP1
TCP2
TCPn
UDP
(Fraunhofer HHI)
"Layer 7" Data-in-Motion Processing
In-Network Compute Accelerators
Corundum i/f

- The (horizontal) datapath is full duplex 128 bits wide using AXI4 Stream
- The (vertical) control path is AXI4 Lite register interfaces with HDL wrappers
- NPAP brings its own 10G/25G Low-Latency Ethernet MAC,
  but can also interface with Ethernet subsystems from the FPGA vendors
- NPAP implements a complete TCP/UDP/IPv4 stack including functions like ARP, ICMP, IGMP, DHCP
- NPAP is delivered with “Support IP blocks” including design examples for setting MAC addresses and/or IP 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 one instance of a TCP Core
- 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
    - 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

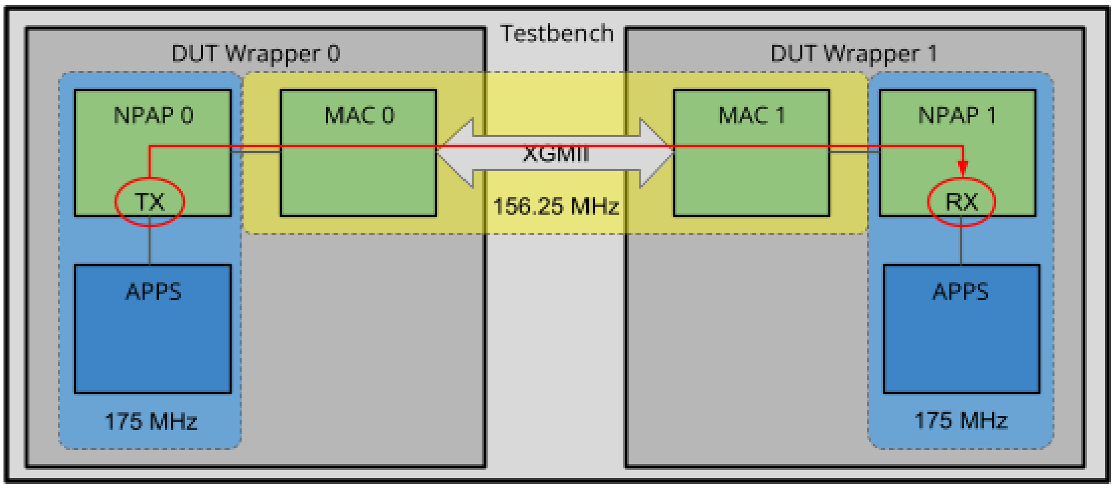
Team MLE has gained long and deep experiences with integrating NPAP into systems and will support you in identifying, implementing and testing the right architecture choice. Key aspects are outlined below. Please contact us for more details.

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	RTT [ns] at 175 MHz	RTT [ns] at 322 MHz	RTT [ns] at 550 MHz
1	62	708.6	385.1	225.5
32	67	765.7	416.1	243.6
64	73	834.3	453.4	265.5
160	91	1,040.0	565.2	330.9
448	145	1,657.1	900.6	527.3
960	241	2,754.3	1,496.9	876.4
1216	289	3,302.9	1,795.0	1,050.9
1456	334	3,817.1	2,074.5	1,214.5

Latency was measured "door-to-door", i.e. we measured the time difference between sending payload data from one NPAP instance via TCP/IP until receiving that payload data at the other instance of NPAP, see the system-level block diagram:



Obviously, increasing the NPAP clock frequency will reduce RTT 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"[5]

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

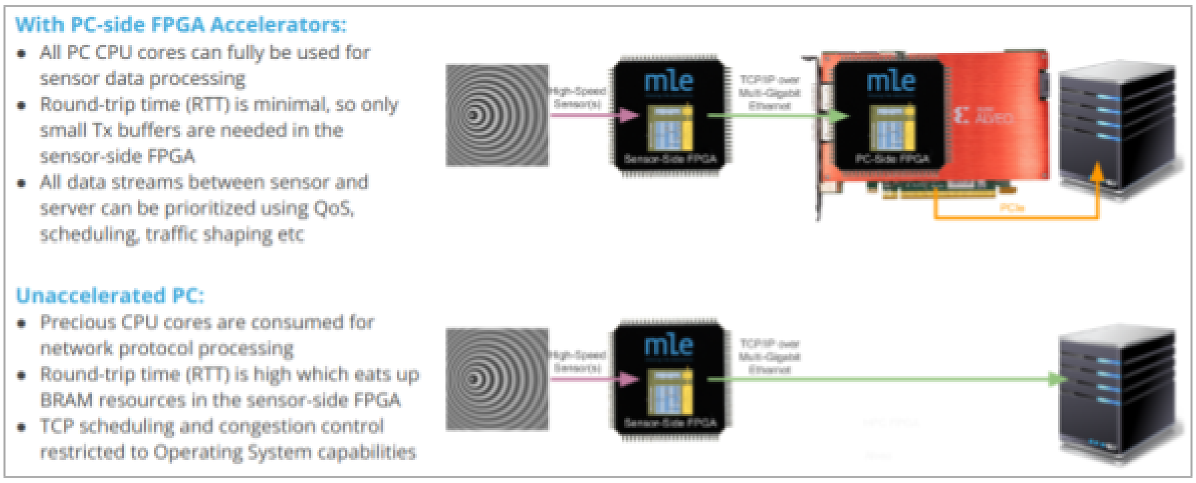
Picking the “Right” TCP Peer

NPAP is fully interoperable with (almost) any other TCP/UDP/IP 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 sending 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 deliver good performance. Predictable high bandwidth and low latency is typically delivered by a “balanced” TCP connection such as putting NPAP on both sides.

When implementing so-called High-Speed Data Acquisition systems[6] we do recommend investigating NPAP not only for sensor-side TCP/IP acceleration but also for PC-side:



With PC-side FPGA Accelerators:

- All PC CPU cores can fully be used for sensor data processing
- Round-trip time (RTT) is minimal, so only small Tx buffers are needed in the sensor-side FPGA
- All data streams between sensor and server can be prioritized using QoS, scheduling, traffic shaping etc

Unaccelerated PC:

- Precious CPU cores are consumed for network protocol processing
- Round-trip time (RTT) is high which eats up BRAM resources in the sensor-side FPGA
- TCP scheduling and congestion control restricted to Operating System capabilities

6 https://devzone.missinglinkelectronics.com/application-notes/high-speed-data-acquisition-systems/

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, that 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. If the recipient is NPAP in a direct connection then we can assume ACK times less than 20 microseconds, i.e. buffer sizes shall be 200k bits. Means in this case one single 32 kBytes FPGA BRAM will be sufficient.
2. If the recipient is 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 RTT of 100 microseconds, or longer (you can run 'Netperf' on your machine to find out). Means buffer sizes shall be around 1M bits, or the 128K Bytes of BRAM we typically instantiate.

NPAP allows to set Tx and Rx buffer sizes individually and per each TCP Core, to facilitate optimizations for more unidirectional dataflows.

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. The low costs of system RAM for storing each TCP connections' 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.

Optimizing NPAP for Linerate Performance

MLE has been working with FPGA vendors to constantly improve NPAP clock frequency. While NPAP originally was designed for ASIC implementation, MLE has adopted NPAP for efficient implementation using modern FPGA devices. 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.

Larger bit widths, 512 bits or more, cause "bloat" which is wasting FPGA resources. Smaller bit widths, 64 bits or less, do require unrealistic high clock frequencies to deliver high linerates 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/IP Turbo Into Your FPGA-SmartNIC"[7] as this discusses pipelining the data paths to avoid "FPGA bloat".

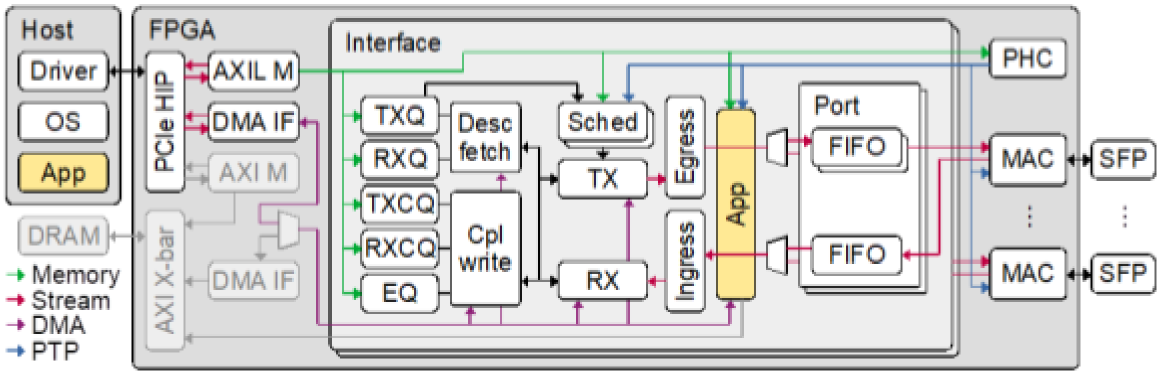
[7] https://devzone.missinglinkelectronics.com/application-notes/put-a-tcp-udp-ip-turbo-into-your-fpga-smartnic/

Combining NPAP With FPGA Network Interface Cards (NIC)

An FPGA Network Interface Card (NIC) does some network packet handling in FPGA logic and then DMA's 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).

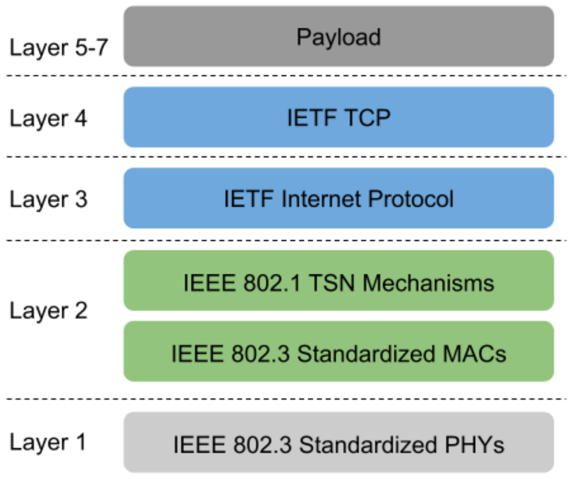
One of the key advantages of the Corundum architecture is support for In-Network Processing inside the FPGA logic, shown as "App" in the block diagram:



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!

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.


Layer 5-7
Payload
Layer 4
IETF TCP
Layer 3
IETF Internet Protocol
Layer 2
IEEE 802.1 TSN Mechanisms
IEEE 802.3 Standardized MACs
Layer 1
IEEE 802.3 Standardized PHYs

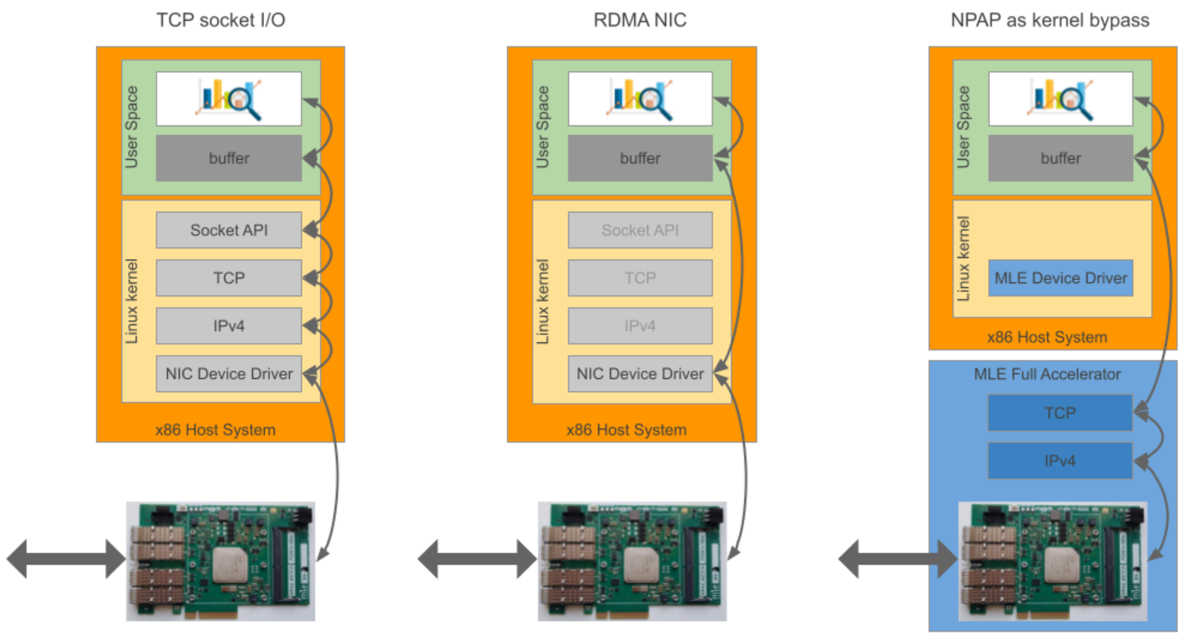
The outcome is a deterministic and reliable network protocol, which makes TCP/IP over TSN a very good candidate for all networking where IT (Information Technology) and OT (Operations Technology) converge, or in Systems-of-Systems backbones. TSN itself is quickly evolving so please contact us for more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

Linux Kernel Bypass With NPAP

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

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.



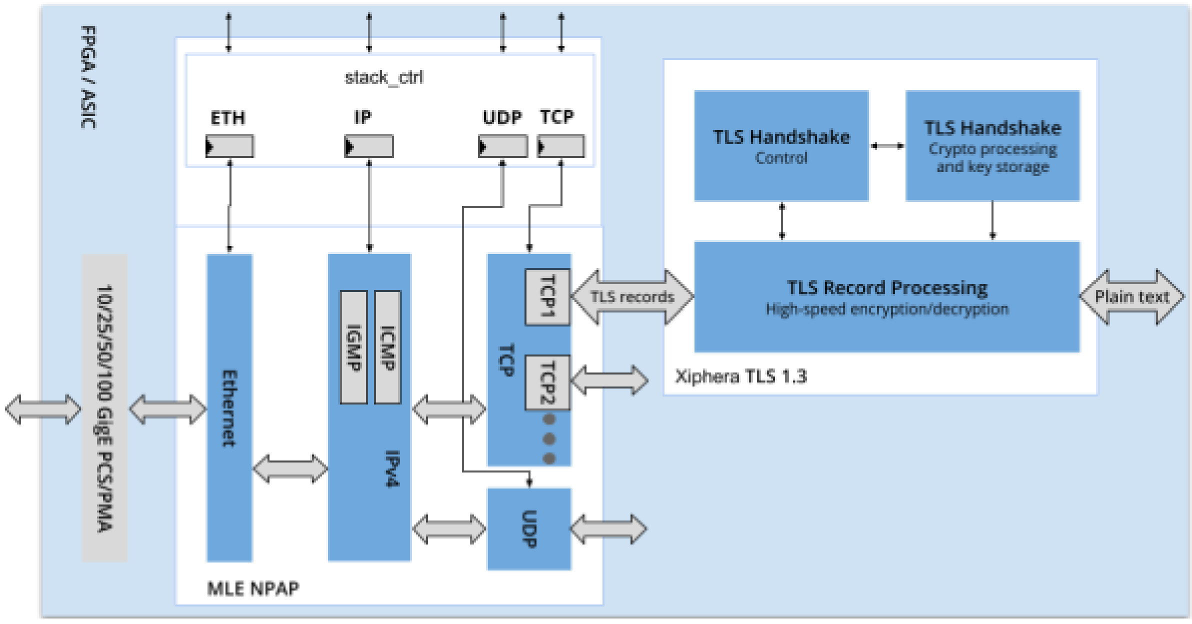
NPAP can operate in a similar way, where NPAP does all TCP/IP processing in dedicated ASIC / FPGA logic and then a (Linux) device driver copies the payload data between user space memory and NPAP.

Adding 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/IP 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/IP 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 .

NPAP Evaluation Choices

MLE offers multiple ways to evaluate and benchmark NPAP:

- Our Remote Evaluation System (RES) hosts a dedicated NPAP installation in the "Cloud"
- Free-of-charge Evaluation Reference Designs are available for a limited set of 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 within your target hardware

Evaluation Reference Designs

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

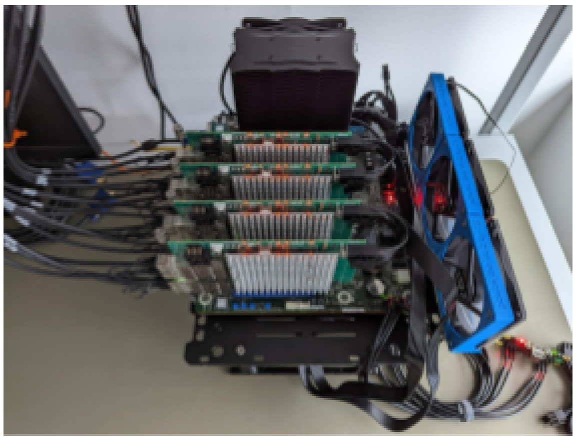
- MLE's Network Protocol Accelerator Card "Ketch" with Intel Stratix 10 GX
- NPAP-10G on AMD/Xilinx ZCU102 with Zynq Ultrascale+ MPSoC ZU9EG
- NPAP-25G on AMD/Xilinx ZCU111 with Zynq Ultrascale+ RFSoC ZU28EG
- NPAP-100G on AMD/Xilinx ZCU111 with Zynq Ultrascale+ RFSoC ZU28EG
- NPAP-25G on AMD Alveo U200
- NPAP-25G on Trenz Electronics TE0950 with AMD Versal AI Edge VE2302
- NPAP-25G on Intel N6001-PL (Agilex AGF014 F Series FPGA)
- NPAP-25G on Intel Agilex 5 E-series A5E 043B / 065B
- NPAP-10G on Microchip PolarFire MPF300-EVAL-KIT

Each ERD typically instantiates the full stack including MAC, Ethernet, IPv4 (with ICMP and IGMP), plus 10 TCP session instances, plus the UDP block, plus Netperf/Netserver implementation in programmable logic. The Netperf/Netserver block is compatible with open source Netperf/Netserver 2.6 and can be used for functionality analysis and for performance benchmarking.

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

mle
MLE ZCU111-2
MLX-4LX
ZCU102-7.0
ZCU102-6.2
ZCU102-6

mle
MLE ZCU111-2



Resource Estimates for ASIC and FPGA

While NPAP has been implemented using re-targetable RTL HDL code we just cannot provide resource estimates for all possible target technologies. If you are interested in integrating NPAP into another target technology, please contact us.

Resource Estimates for AMD/Xilinx Ultrascale+ Series

The following table shows resources for AMD/Xilinx Zynq Ultrascale+ MPSoC ZU19EG compiled with Xilinx Vivado 2018.3 - instantiating the following design features:

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

```
+-------------------------------------+---------------------------+--------+--------+------+------+-------+--------+--------+--------+
|                                     |                           |  Total |  Logic | LUT  |      |       |        |        | DSP48  |
| Instance                            |           Module          |  LUTs  |  LUTs  | RAMs | SRLs |  FFs  | RAMB36 | RAMB18 | Blocks |
+-------------------------------------+---------------------------+--------+--------+------+------+-------+--------+--------+--------+
| npap_tcp_udp_wrapper_u0             |      npap_tcp_udp_wrapper |  33277 |  31755 | 1506 |   16 | 35034 |     71 |     10 |      6 |
|   npap_tcp_udp_wrapper_u0           |      npap_tcp_udp_wrapper |  31359 |  29837 | 1506 |   16 | 34053 |     71 |     10 |      6 |
|     (npap_tcp_udp_wrapper_u0)       |      npap_tcp_udp_wrapper |     65 |     65 |    0 |    0 |     0 |      0 |      0 |      0 |
|     npap_tcp_udp_top_u0             |          npap_tcp_udp_top |  31294 |  29772 | 1506 |   16 | 34053 |     71 |     10 |      6 |
|       (npap_tcp_udp_top_u0)         |          npap_tcp_udp_top |      0 |      0 |    0 |    0 |     1 |      0 |      0 |      0 |
|       gen_hhi_to_axis_adapter[0].u  |     hhi_to_axis_adapter_5 |      8 |      8 |    0 |    0 |     0 |      0 |      0 |      0 |
|       gen_hhi_to_axis_adapter[1].u  |     hhi_to_axis_adapter_6 |      8 |      8 |    0 |    0 |     0 |      0 |      0 |      0 |
|       gen_hhi_to_axis_adapter[2].u  |     hhi_to_axis_adapter_7 |      8 |      8 |    0 |    0 |     0 |      0 |      0 |      0 |
|       wrapper_ll_ip_tcp_u0          |         Wrapper_LL_IP_TCP |  31270 |  29748 | 1506 |   16 | 34052 |     71 |     10 |      6 |
|         (wrapper_ll_ip_tcp_u0)      |         Wrapper_LL_IP_TCP |     42 |     42 |    0 |    0 |   203 |      0 |      0 |      0 |
|         GEN_MAC_NET_CONV_HHI.u      | MacNetworkLayerConversion |    370 |    290 |   80 |    0 |   554 |      0 |      0 |      0 |
|         g0.i_tcpTxMux               |                  TcpTxMux |    242 |    242 |    0 |    0 |     3 |      0 |      0 |      0 |
|         gen_WithUdp.i_udp           |               wrapper_udp |   2226 |   2144 |   82 |    0 |  3243 |     19 |      1 |      0 |
|         gen_tcpConnections[0].u     |    Wrapper_TCP__xdcDup__1 |   8030 |   7598 |  432 |    0 |  8113 |     16 |      3 |      2 |
|         gen_tcpConnections[1].u     |    Wrapper_TCP__xdcDup__2 |   7962 |   7530 |  432 |    0 |  8117 |     16 |      3 |      2 |
|         gen_tcpConnections[2].u     |               Wrapper_TCP |   8018 |   7586 |  432 |    0 |  8117 |     16 |      3 |      2 |
|         iBusScheduler8              |             BusScheduler8 |     85 |     85 |    0 |    0 |    53 |      0 |      0 |      0 |
|         i_internetLayer             |                Wrapper_IP |   2286 |   2222 |   48 |   16 |  2908 |      2 |      0 |      0 |
|         i_networkLayer              |      Wrapper_Networklayer |   2051 |   2051 |    0 |    0 |  2737 |      2 |      0 |      0 |
|         i_rxLinkResetSync           |     ResetSync__xdcDup__20 |      0 |      0 |    0 |    0 |     2 |      0 |      0 |      0 |
|         i_txLinkResetSync           |     ResetSync__xdcDup__21 |      0 |      0 |    0 |    0 |     2 |      0 |      0 |      0 |
|   mac_10_gbe_wrapper_u0             |          mac10gbe_wrapper |   1918 |   1918 |    0 |    0 |   981 |      0 |      0 |      0 |
|     mac10gbe_top_u0                 |              mac10gbe_top |   1918 |   1918 |    0 |    0 |   981 |      0 |      0 |      0 |
|       mac10Gbe_struct_u0            |           mac10Gbe_struct |   1918 |   1918 |    0 |    0 |   981 |      0 |      0 |      0 |
+-------------------------------------+---------------------------+--------+--------+------+------+-------+--------+--------+--------+
```

Resource Estimates for AMD/Xilinx 7-Series

The following table shows resources for AMD/Xilinx 7-Series Kintex fabric (XC7Z045-2) compiled with 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

```
+-----------------------------+-------------------------------------------+-------+-------+------+------+-------+--------+--------+--------+
|                             |                                           | Total | Logic | LUT  |      |       |        |        | DSP48  |
|          Instance           |                  Module                   | LUTs  | LUTs  | RAMs | SRLs |  FFs  | RAMB36 | RAMB18 | 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 |
+-----------------------------+-------------------------------------------+-------+-------+------+------+-------+--------+--------+--------+
```

Resource Estimates for Intel Stratix-10

The following table shows resources for compiled with Quartus Prime v19.3 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

```
+-----------------------------------------------+-------------------+-------------+--------------+-------------+-----------+-------+---
                                                                                             -----+
 |                                               |      ALMs used    |             |    Dedicated |             |     Block |       |
                                                                     |
 |                                               |      in final     |  ALMs used  | Combinational |      Logic |    Memory |       |
                                                                 DSP |
 |Compilation Hierarchy Node                     |     placement     |  for memory |        ALUTs |   Registers |      Bits |  M20Ks|
                                                              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_l1_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 |
+-----------------------------------------------+-------------------+-------------+--------------+-------------+-----------+-------+---
                                                                                             -----+
```

Resource Estimates for Microchip Polarfire

The following table shows resources synthesized for Microchip PolarFire MPF300TS-1FCG1152I using Libero 2021.1 - instantiating the following design features:

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

```
+-------------------------------+--------+--------+-----------+-----------+-------+-------+-------+--------+
|                               | Fabric | Fabric | Interface | Interface | uSRAM | LSRAM |  Math | Chip   |
|             Instance          |   4LUT |    DFF |      4LUT |       DFF |    1K |   18K | 18x18 | 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 |
+-------------------------------+--------+--------+-----------+-----------+-------+-------+-------+--------+
```

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 2023.2.2 - instantiating the following design features:

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

```
+-------------------------------------+------------+------------+---------+-------+--------+--------+------+------------+
|              Instance               | Total LUTs | Logic LUTs | LUTRAMs |  FFs  | RAMB36 | RAMB18 | URAM | DSP Blocks |
+-------------------------------------+------------+------------+---------+-------+--------+--------+------+------------+
| npap_tcp_udp_wrapper_u0             |      47022 |      45446 |    1576 | 37940 |    118 |      4 |    9 |          3 |
|   npap_tcp_udp_wrapper_u0           |      47022 |      45446 |    1576 | 37940 |    118 |      4 |    9 |          3 |
|     npap_tcp_udp_top_u0             |      47022 |      45446 |    1576 | 37940 |    118 |      4 |    9 |          3 |
|       axis_register_udp_u0          |        101 |        101 |       0 |   428 |      0 |      0 |    0 |          0 |
|       axis_register_udp_u1          |        293 |        293 |       0 |   426 |      0 |      0 |    0 |          0 |
|       gen_hhi_to_axis_adapter[0]...|        156 |        156 |       0 |   296 |      0 |      0 |    0 |          0 |
|       gen_hhi_to_axis_adapter[0]...|        179 |        179 |       0 |   296 |      0 |      0 |    0 |          0 |
|       gen_hhi_to_axis_adapter[1]...|         35 |         35 |       0 |    56 |      0 |      0 |    0 |          0 |
|       gen_hhi_to_axis_adapter[1]...|        105 |        105 |       0 |   146 |      0 |      0 |    0 |          0 |
|       gen_hhi_to_axis_adapter[2]...|        249 |        249 |       0 |   296 |      0 |      0 |    0 |          0 |
|       gen_hhi_to_axis_adapter[2]...|        179 |        179 |       0 |   296 |      0 |      0 |    0 |          0 |
|       npap_configuration_block_u...|        122 |        122 |       0 |   339 |      0 |      0 |    0 |          0 |
|       npap_priority_manager_conf...|         30 |         30 |       0 |    39 |      0 |      0 |    0 |          0 |
|       wrapper_ll_ip_tcp_u0          |      45578 |      44002 |    1576 | 35322 |    118 |      4 |    9 |          3 |
|         (wrapper_ll_ip_tcp_u0)      |        448 |        448 |       0 |   574 |      0 |      0 |    0 |          0 |
|         g0.i_tcpTxMux               |          0 |          0 |       0 |     2 |      0 |      0 |    0 |          0 |
|         gen_WithUdp.i_udp           |       4855 |       4697 |     158 |  4593 |     12 |      0 |    0 |          0 |
|         gen_tcpConnections[0].i_...|         49 |         49 |       0 |    34 |      0 |      0 |    0 |          0 |
|         gen_tcpConnections[0].i_...|      11718 |      11304 |     414 |  8489 |     34 |      1 |    4 |          1 |
|         gen_tcpConnections[1].i_...|         49 |         49 |       0 |    34 |      0 |      0 |    0 |          0 |
|         gen_tcpConnections[1].i_...|      10895 |      10505 |     390 |  8025 |     31 |      1 |    1 |          1 |
|         gen_tcpConnections[2].i_...|         48 |         48 |       0 |    34 |      0 |      0 |    0 |          0 |
|         gen_tcpConnections[2].i_...|      11722 |      11306 |     416 |  8499 |     34 |      1 |    4 |          1 |
|         iBusScheduler8              |        565 |        565 |       0 |    33 |      0 |      0 |    0 |          0 |
|         i_internetLayer             |       2984 |       2866 |     118 |  3106 |      0 |      0 |    0 |          0 |
|         i_networkLayer              |       2242 |       2162 |      80 |  1852 |      7 |      1 |    0 |          0 |
|         i_resetStretchStack         |         54 |         54 |       0 |    47 |      0 |      0 |    0 |          0 |
+-------------------------------------+------------+------------+---------+-------+--------+--------+------+------------+
```

Detailed protocol support according RFC1122 (excerpt)

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 IP layer	2.4	x		
IP layer pass TOS to link layer	2.4	x		
No ARP cache entry treated as Dest. Unreach.	2.4		x	x

IP & ICMP Layer

Feature	Section	Must	Must not	Implemented
Implement IP 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 IP 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 IP 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
IP Options:				
Allow transport layer to send IP options	3.2.1.8	x		-
Pass all IP options rcvd to higher layer	3.2.1.8	x		-
IP 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 IP addr	3.3.4.2	x		x
BROADCAST:				
Broadcast addr as IP source addr	3.2.1.3		x	-
Recognize all broadcast address formats	3.3.6	x		-
Use IP b'cast/m'cast addr in link-layer b'cast	3.3.6	x		-

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

TCP Layer

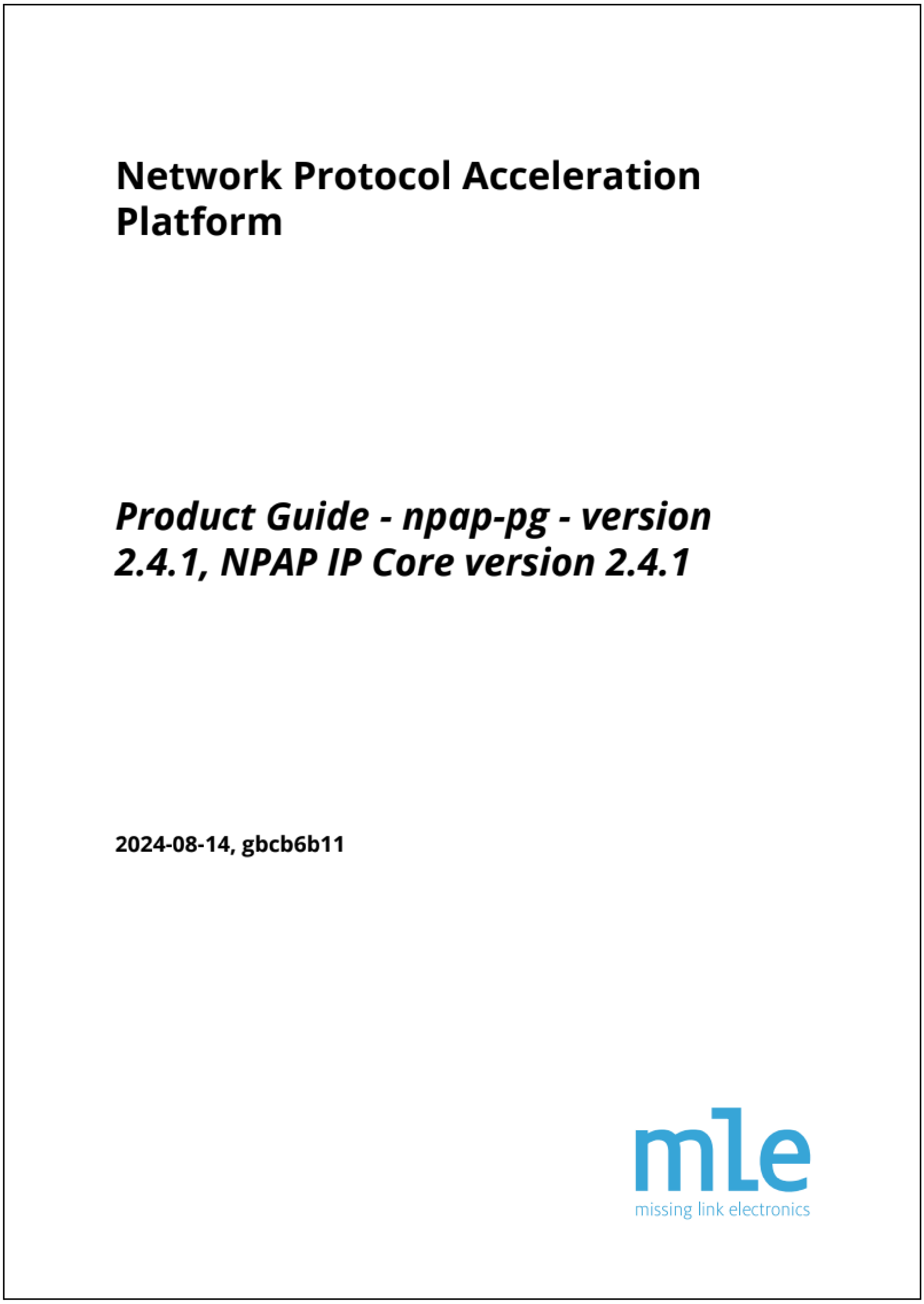
Feature	Section	Must	Must not	Implemented
Push flag				
ESEND call can specify PUSH	4.2.2.2			-
If cannot: sender buffer indefinitely	4.2.2.2		x	
If cannot: PSH last segment	4.2.2.2	x		x
Window				
Treat as unsigned number	4.2.2.3	x		x
Robust against shrinking window	4.2.2.16	x		-
Sender probe zero window	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 IP 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 IP 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		-

Developer Documentation

A comprehensive product design guide (currently version 2.4.1) with detailed description of the functions and how to integrate is available under license:

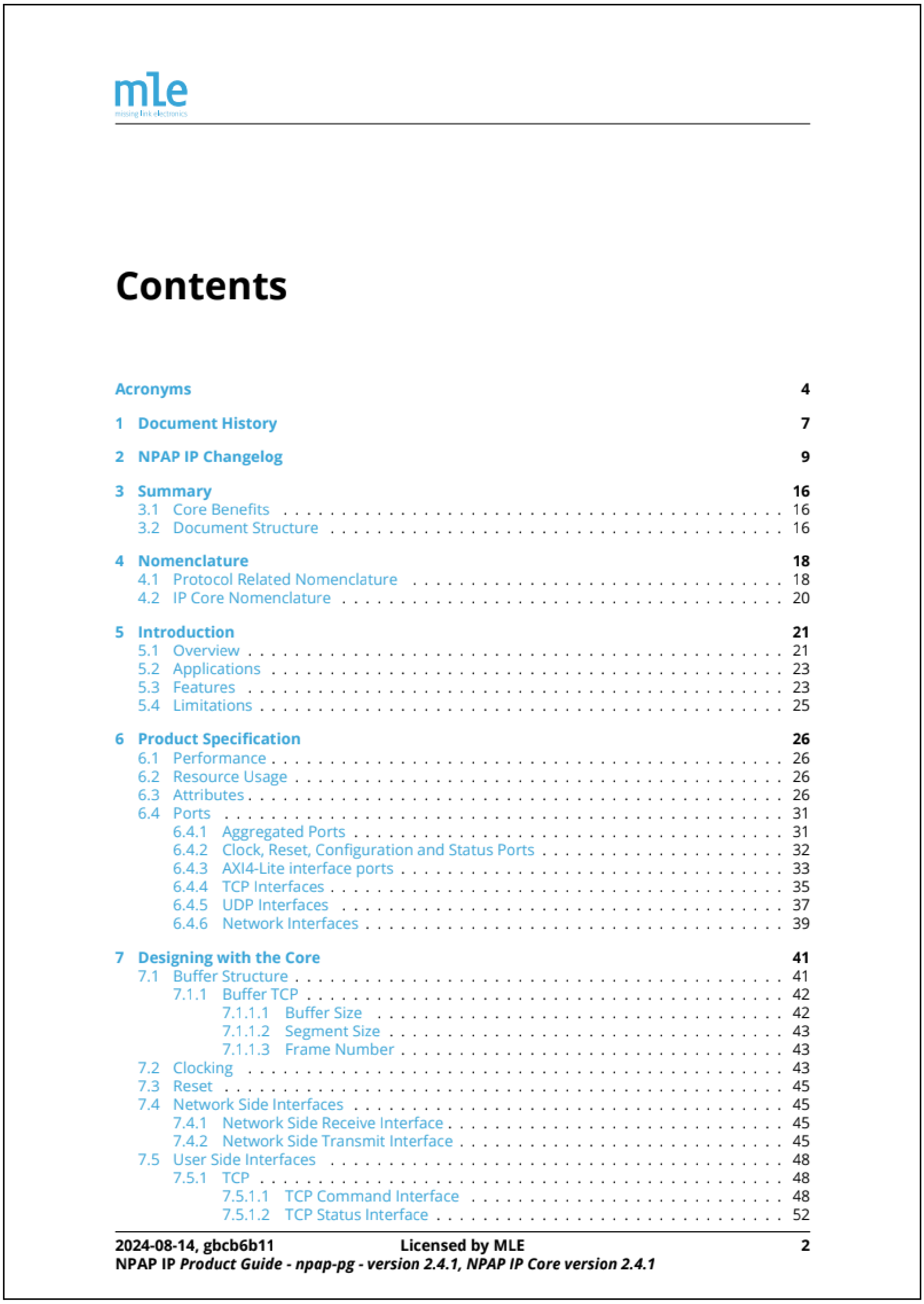
- NPAP Product Guide
- NPAP Control Application Product Guide
- NPAP Data Generator / Checker Product Guide



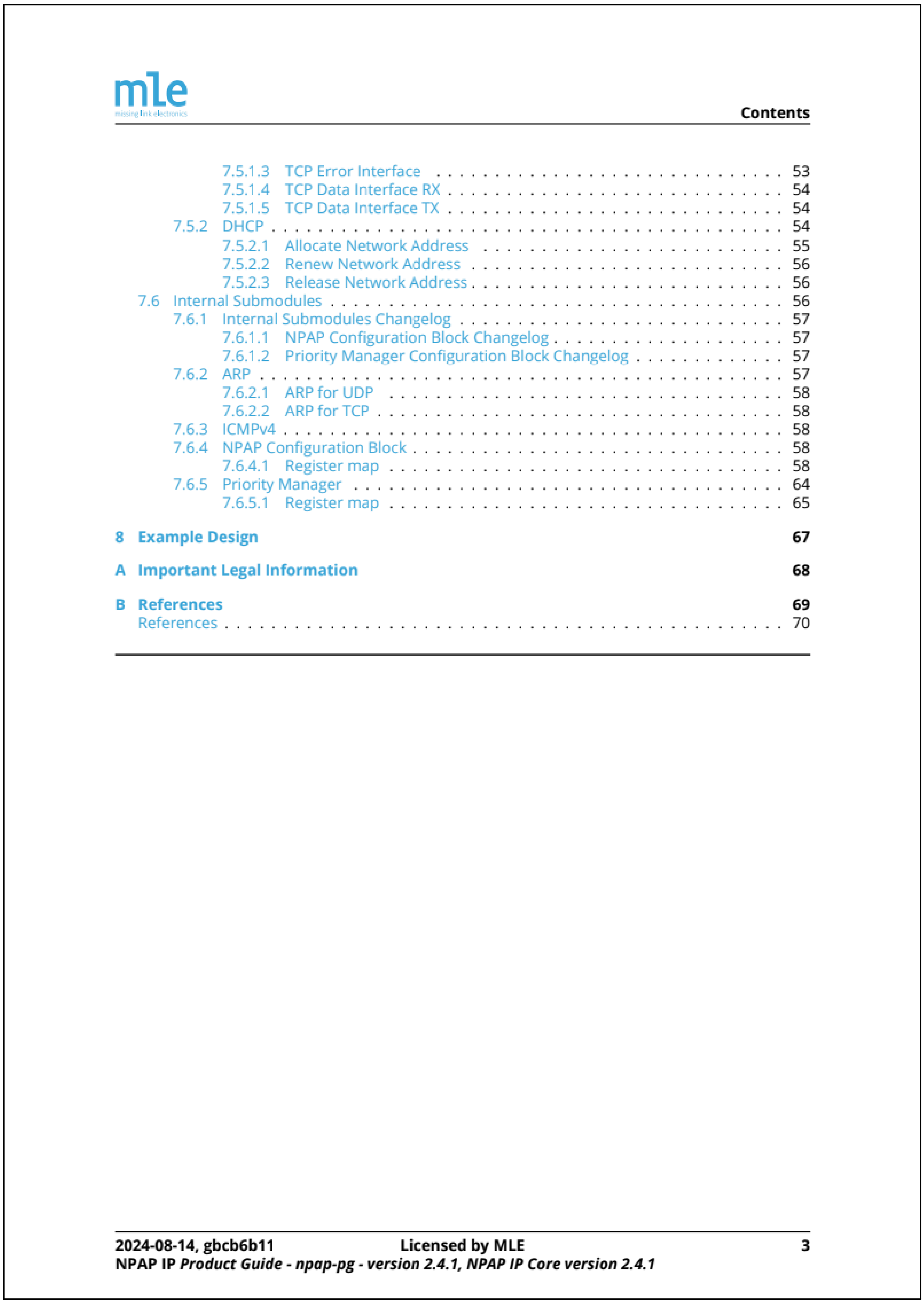
Network Protocol Acceleration Platform

Product Guide - npap-pg - version 2.4.1, NPAP IP Core version 2.4.1

2024-08-14, gbcb6b11



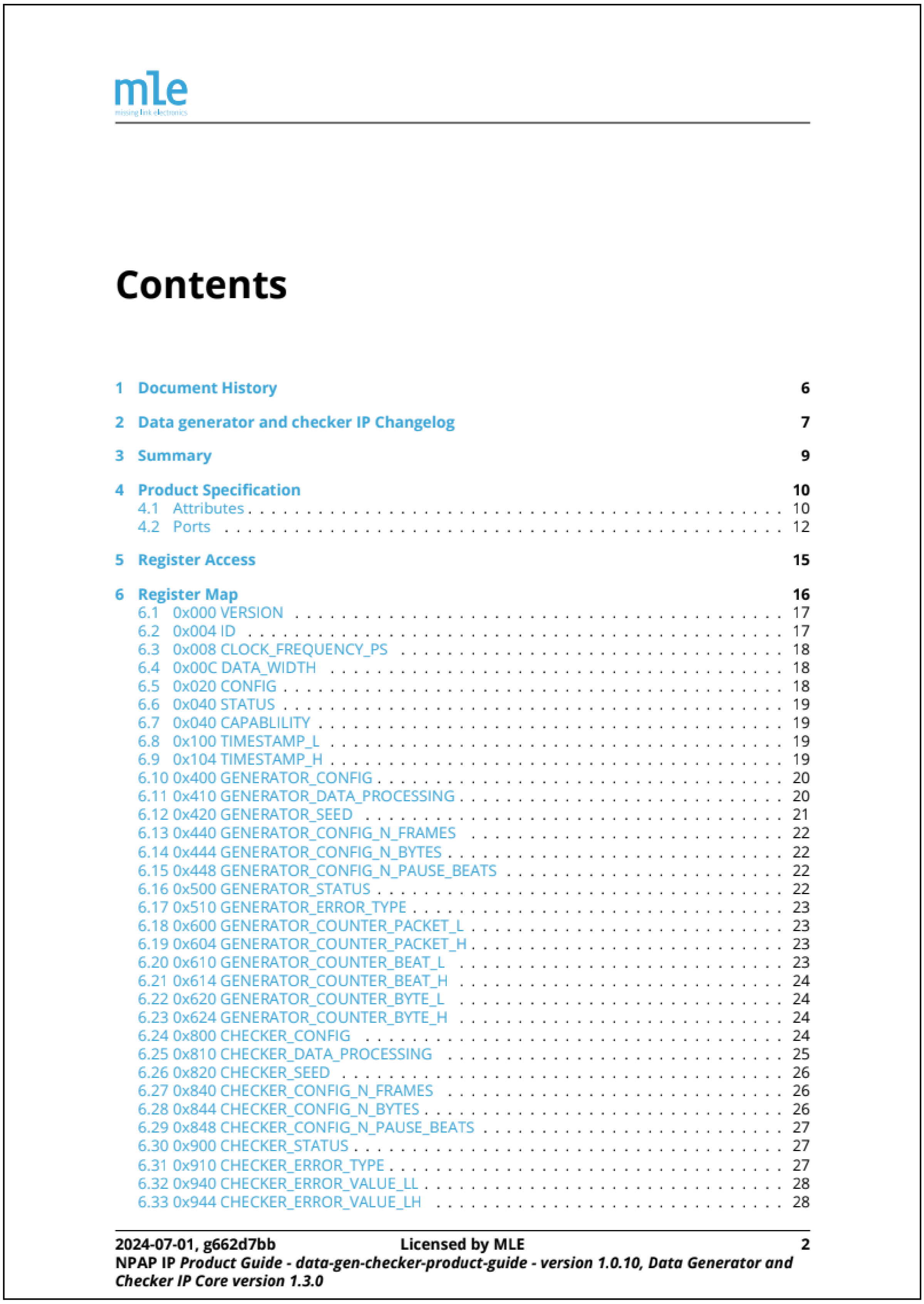
Contents



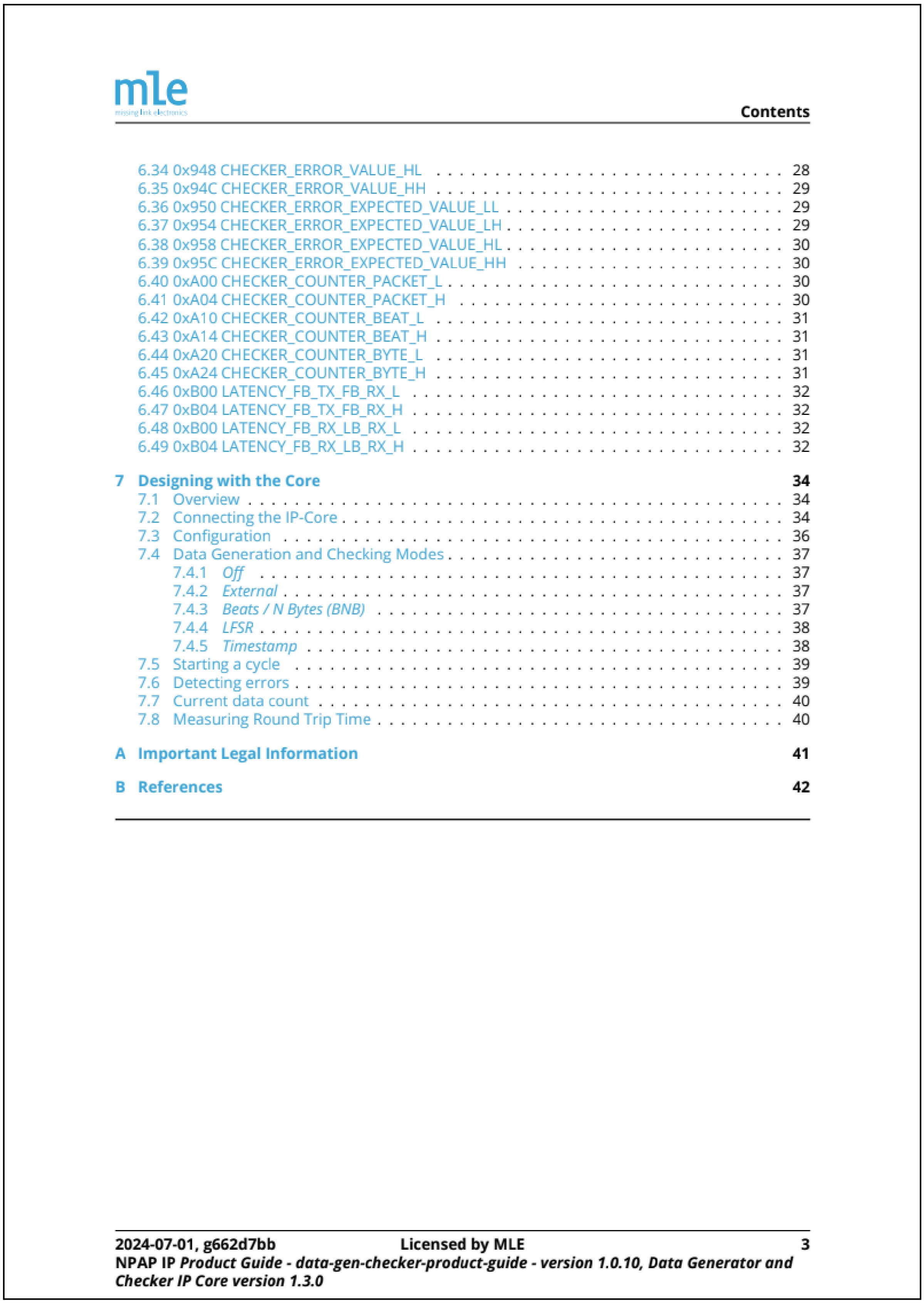
Network Protocol Acceleration Platform

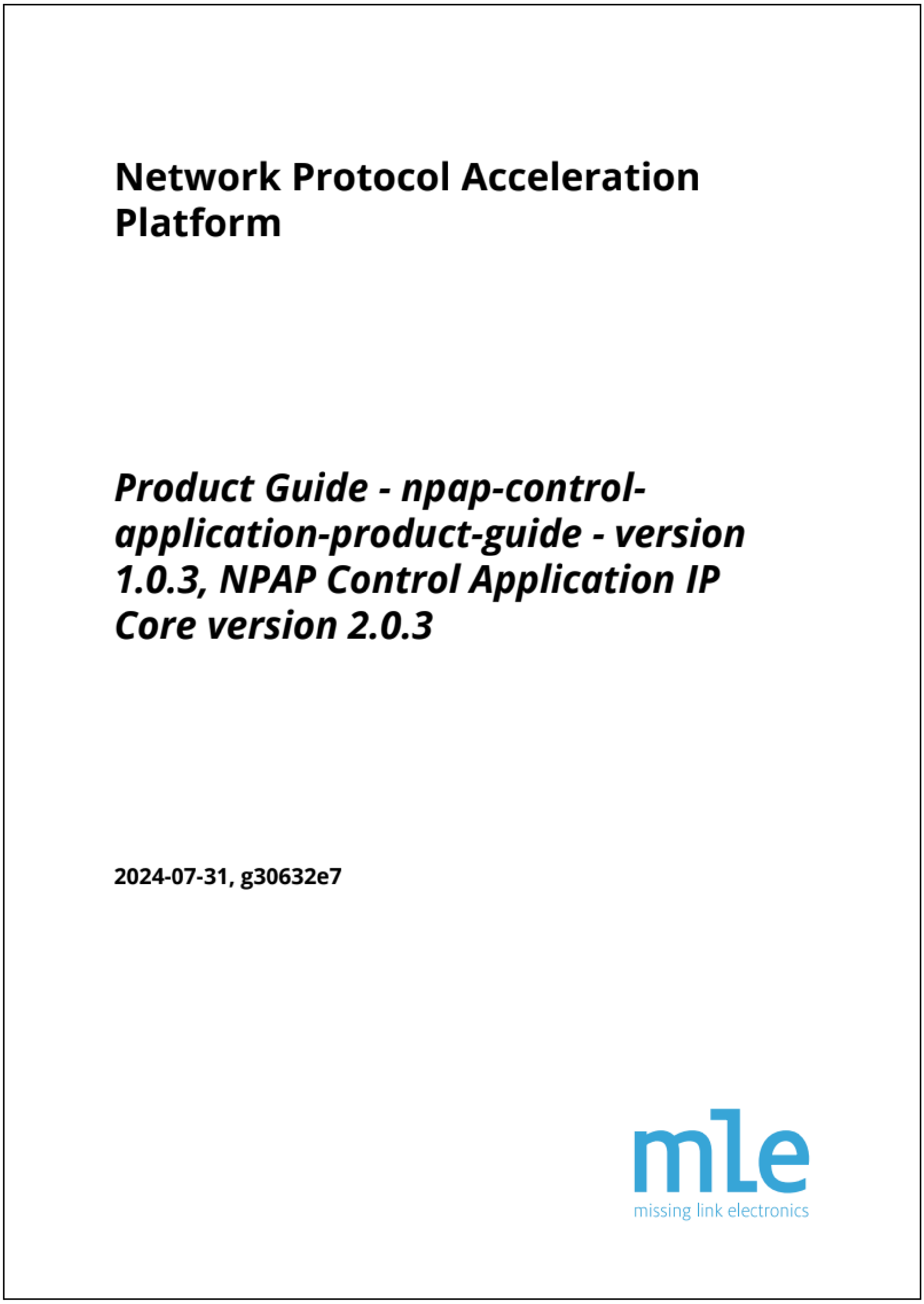
Product Guide - data-gen-checker-product-guide - version 1.0.10, Data Generator and Checker IP Core version 1.3.0

2024-07-01, g662d7bb



Contents

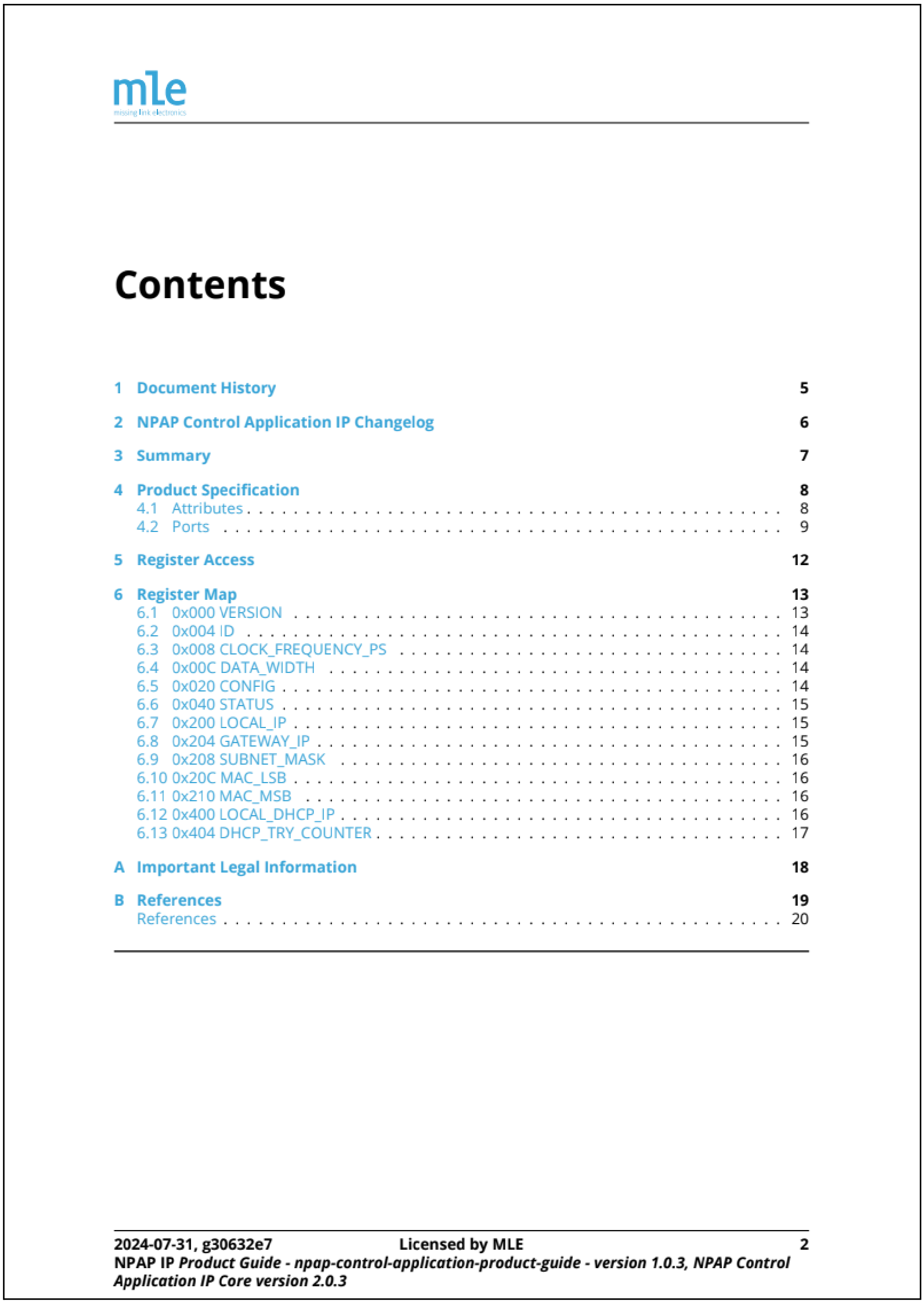




Network Protocol Acceleration Platform

Product Guide - npap-control-application-product-guide - version 1.0.3, NPAP Control Application IP Core version 2.0.3

2024-07-31, g30632e7



Contents

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.

NPAP Version 2 Development

- 2.4.1 (20240813)
    - GENERAL
        - #6590 - rewrite FIFO constrains from xdc to tcl
- 2.4.0 (20240731)
    - 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
- 2.3.4 (20240430)
    - GENERAL
        - #6413 - fix bus scheduler early release hang
- 2.3.3 (20240311)
    - GENERAL
        - #6399 - fix possible UDP checksum mismatch
        - #6377 - fix UDP only design
- 2.3.2 (2024029)
    - GENERAL
        - #4909 - fix bus scheduler fairness
        - #5987 - fix G_PRIORITY_WIDTH synthesis error
        - #6243 - fix G_ENABLE_UDP = 0 timing loop
- 2.3.1 (20240101)
    - GENERAL

    - #6005 - update IP packaging to support Versal FPGA
- 2.3.0 (20231129)
  - GENERAL
    - #5743 - add AXIL-Lite control and status interface (NPAP Configuration Block)
    - #5948 - update Priority Manager to new AXI4-Lite structure
- 2.2.5 (20230930)
  - GENERAL
    - #5963 - update timeouts to be clock depended
- 2.2.4 (20230919)
  - TCP
    - #5954 - harden TCP against lost ACK
- 2.2.3 (20230831)
  - GENERAL
    - #5880 - remove unused reset stretch modules
- 2.2.2 (20230731)
  - TCP
    - #5805 - fix out of order received ACK handling
- 2.2.1 (20230630)
  - UDP
    - #5707 - fix UDP meta data handling
- 2.2.0 (20230531)
  - 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)
- 2.1.2 (20230411)
  - TCP
    - #5562 - add support for partially ACKed packages
- 2.1.1 (20230331)
  - 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
- 2.1.0 (20230306)
    - TCP
        - #5554 - fix signal overflow on TCP transmit controller
        - #5555 - fix signal overflow on TCP receive buffer calculation signal
- 2.0.1 (20220822)
    - version changes not included
        - NPAP 1.9.2 to 1.10.0
    - GENERAL
        - #4836 - add priority scheduler
- 2.0.0 (20220715)
    - 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

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 IP 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 IP 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 ip 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 IP 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

Contact Info

Missing Link Electronics, Inc.
2880 Zanker Road, Suite 203
San Jose, CA 95134
Phone: +1-408-475-1490

Missing Link Electronics GmbH
Industriestraße 10
89231 Neu-Ulm
Tel. +49 731 141149-0

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” .