NetTimeLogic GMBH

IEEE ISPCS 2024 October 7-11, 2024 | Tokyo, Japan

1 Mio Requests with less than 3 Watts

NTP and Client-Server PTP fully in Hardware

Content

NetTimeLogic GMBH

- Why NTP and CSPTP fully in Hardware?
- Explanation of the Design
- Performance and Limitations of the current Design
  - Measurements
  - Simulations
- Scheme for higher Link Speeds and higher Request Rates
- Scheme how to integrate Security
- Conclusion

Why NTP and CSPTP fully in Hardware?

What is the problem we want to solve?

NetTimeLogic GMBH

- **Both NTP and Client-Server PTP (CSPTP) are unicast Protocols**
  - One Server needs to serve many (million) Clients
- **Number of Clients can easily go into Millions in Datacenters or Public NTP Servers**
- **Handling of Millions of Request per Second is challenging with embedded CPUs**
  - Requires powerful CPUs due to the sequential handling of data
  - Interrupt load is quite high for high Ethernet traffic, CPUs are quite busy transferring data from/to the Ethernet interface
- **Powerful CPUs require a lot of Power and are expensive**
  - Easily goes up to 100s of Watts for 1Mio Requests/s

Why NTP and CSPTP fully in Hardware?

Why a Hardware/FPGA approach?

NetTimeLogic GMBH

- **High Performance with Low Power and Low Costs**
  - Unbeaten compared to other approaches => ~1Mio Request/s with 3Watts on an FPGA for ~40$
  - No CPU load => Coprocessor
- **Hardware Timestamping with Nanosecond resolution**
  - For both NTP and CSPTP
- **Handle all CSPTP frames as pure One-Step**
  - Reduce the message exchange to 2 messages (1 request => 1 response)
- **Line-Speed handling of frames (requests and responses)**
  - Pipelining, full parallelism in FPGA comes handy

Why NTP and CSPTP fully in Hardware?

Why a Hardware/FPGA approach?

NetTimeLogic GMBH

- **Stateless Protocols**
  - No Lookup etc. which makes the implementation pretty straight forward
  - Pure data processing: receive a request => create a response
- **Easy to scale**
  - Larger FPGAs (more cores), higher frequencies can easily increase the performance by factors of 10 and higher without the same factor in costs and power consumption
- **And of course because we can ☺**
  - Creating Synchronization cores for FPGAs as coprocessors is what we do

Explanation of the Design

Requirements first

NetTimeLogic GMBH

- **Shall handle at least 1G interfaces (10/100/1000) at line speed**
  - ~1Mio Requests/s
  - Shall be able to handle burst of requests
- **Shall answer ARP/ICMP request in hardware as well**
  - Shall not be the limiting factor for the overall system performance
- **Shall support IPv4, IPv6 for NTP**
  - SNTP Server (NTPv4) as Unicast/Multicast/Broadcast (no signing 1st)
- **Shall support L2, IPv4, IPv6 for CSPTP**
  - The CSPTP approach shall be FlashPTP at the time it was designed
- **Shall have high accuracy hardware timestamps (~4ns)**
- **Shall have one-step support for PTP**

Explanation of the Design

Requirements first

- **Keep NTP separate from CSPTP**
  - Not all need both, but share the same physical interface
- **Have a small resource footprint**
  - As small as possible by still being modular
  - To allow for scaling
- **Allow scaling**
  - Instantiate multiple cores for load sharing
- **Allow to supervise Status and do initial Configuration**
- **Complete Co-Processor, no run-time interaction shall be required**
- **Common on-chip-bus and conversion to outer-world interfaces**
  - So the interface to the PHY can be swapped easily (MII/GMII/RGMII…)

Explanation of the Design
CSPTP

NetTimeLogic GMBH

- **Why FlashPTP as CSPTP?**
  - Allows for pure one-step operation which brings down the message exchange to 2 per measurement (as for NTP)
    - This is the main reason, since this allows the highest throughput
    - Is something you need to do in hardware
  - Single frame type + TLV
    - Easy to parse and generate
    - Expected frame size is known
  - Mapping for L2, IPv4, IPv6
  - Matches pretty much our idea of CSPTP ☺
    - Once IEEE1588.1 is ready we can easily adapt to it since it's an FPGA

Explanation of the Design

Design considerations

- **Split design into an independent receive path and a transmit path**
- **Back pressure on data bus**
  - Receiving does not use back pressure on the data bus since data from PHY can not be slowed down
  - Transmitting allows back pressure on the data bus e.g. for lower link speeds or arbitration between multiple instances
- **Use a FIFO between the receive and transmit path**
  - Only forward Info from the request through the FIFO which is relevant for the response
  - Reading and writing time from/to the FIFO must be short (< frame)
  - Reception of new requests can happen during transmission of the response

Explanation of the Design

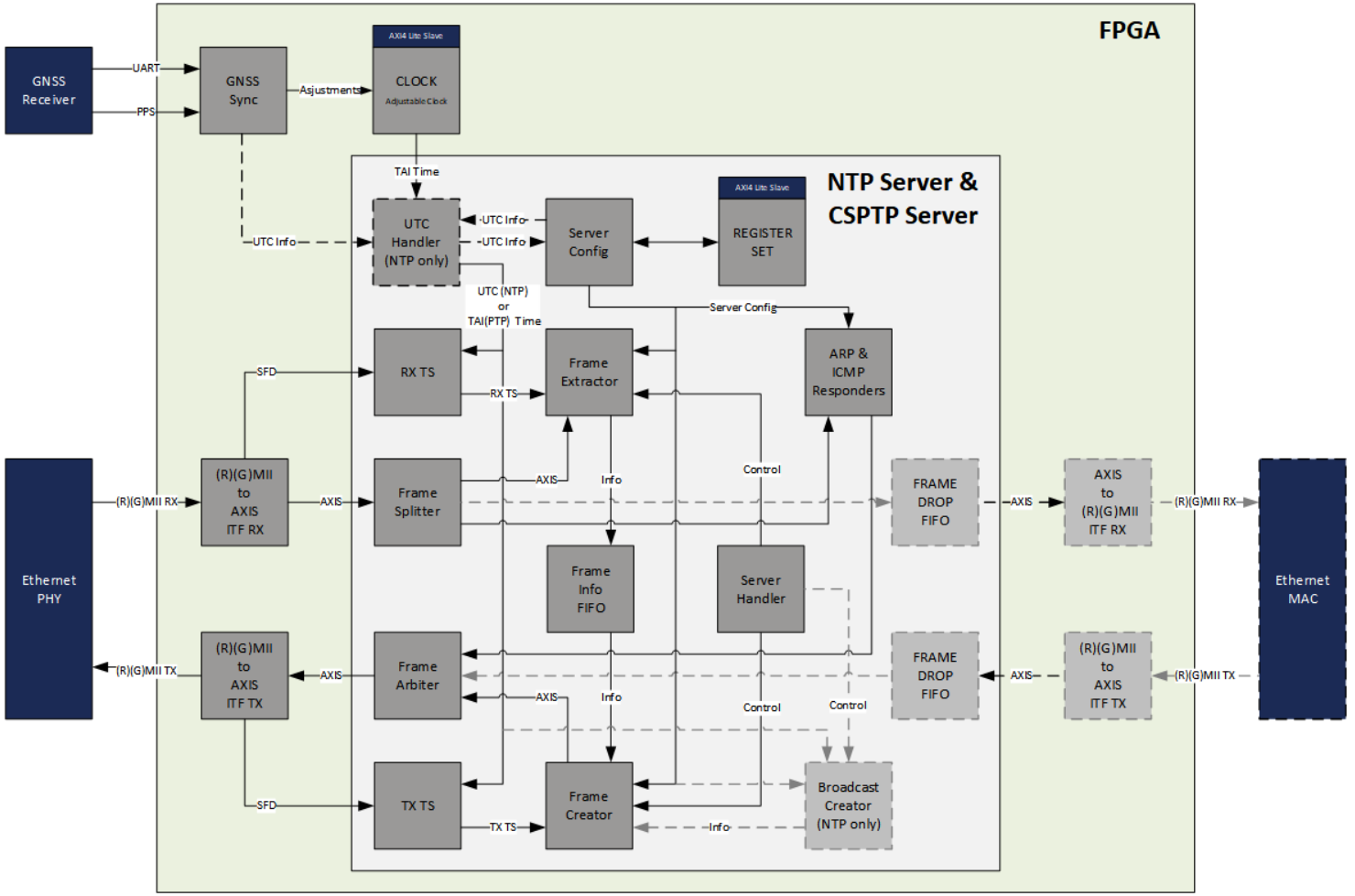
Design considerations

NetTimeLogic GMBH

- **AXI Stream (AXIS) as on-chip-bus for frames**
- **AXI4L Memory Mapped (AXI4L) for initial configuration and status supervision**
  - Only initial config required
- **Keep basic design identical for NTP and CSPTP**
- **Simple one-core solution for 10/100/1000Mbit Ethernet**
  - With (R)(G)MII interfaces
  - Higher Link speeds with more cores => explained later
- **50MHz System clock**
  - To allow also very low performance FPGAs
- **In the path between PHY and MAC (MAC is optional)**

Explanation of the Design

Key Modules and Functionalities



Performance
Measurements

- **AIONIX mini as NTP and CSPTP Server**
  - 100Mbit only (based on an Digilent Arty A7 DevBoard with an AIONYX shield and AIONYX PMOD for GNSS), USB Powered => ~100k Request/s possible

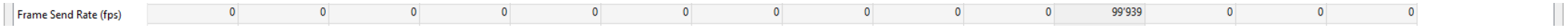
- **Using Ostinato™ on a PC as an NTP and CSPTP Request Sender Simulate ~100k Clients, Frame capture with ProfiShark™ on the wire**
  - Check if Responses come, check status counters on Server

- **With a Gigabit port the performance increases linear by 10 to 1Mio Requests per second**

Performance

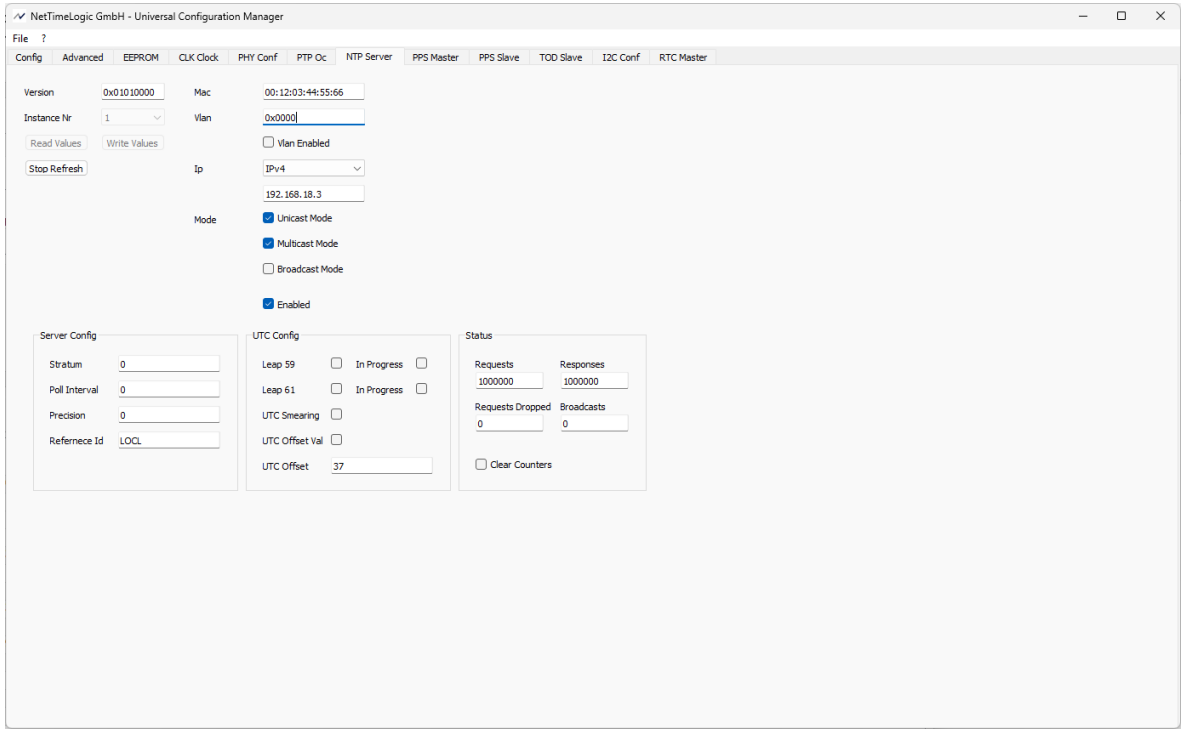
Measurements NTP

- **For NTP the theoretical max frame rate for requests is ~110k/s at 100Mbit/s:**
  - Request/Response is 94 bytes (IPv4, FCS)
  - An NTP frame with min itf and preamble represents 94*8+64+96bits = 912bits
  - 100Mbit/912bit per frame = max 109649 frames/s
- **Frame generator can generate ~<100k frames per second**



Performance
Measurements NTP

- Sent 1Mio Requests (~10s)
- Received 1Mio Responses
- Frame rate was ~100k Requests/s
- No Frame drops



Performance

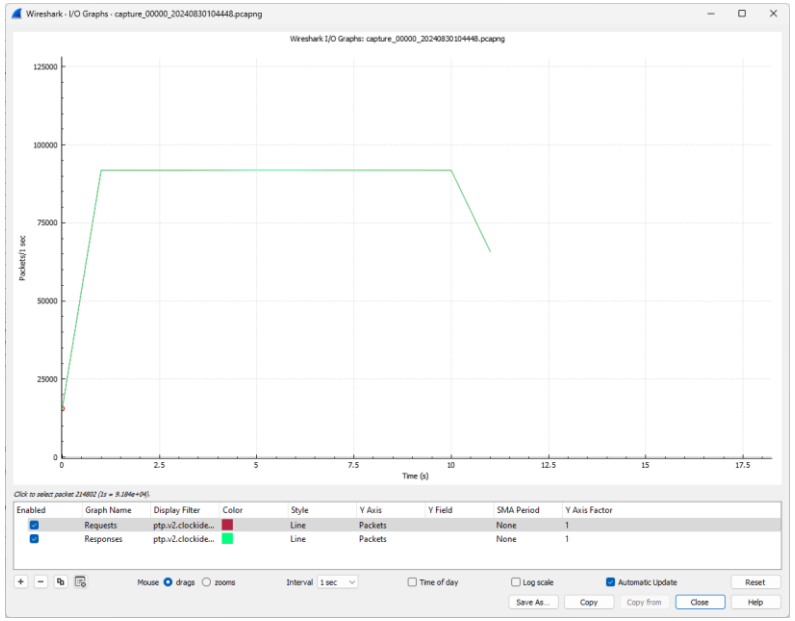
Measurements CSPTP

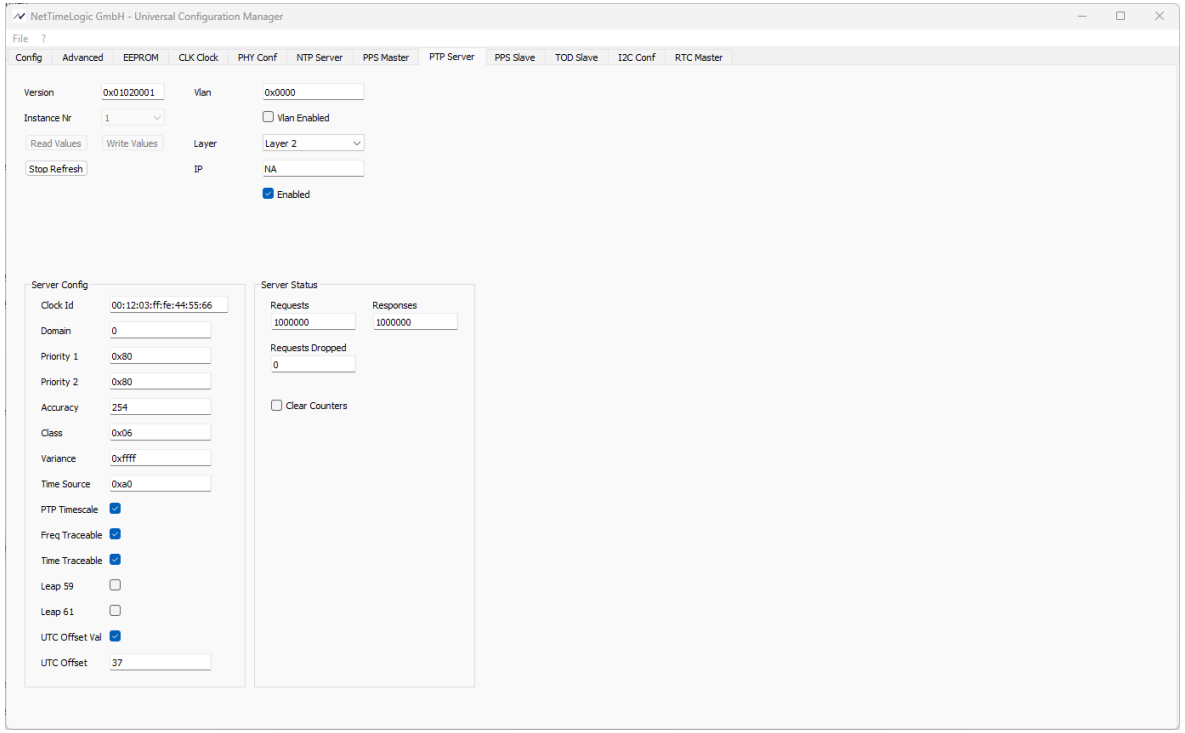
- **For CSPTP the theoretical max frame rate for requests is ~92k/s at 100Mbit/s:**
  - Request/Response is 116 bytes (L2, TLV with ClockInfo, FCS)
  - A CSPTP frame with min itf and preamble represents 116*8+64+96bits = 1088bits
  - 100Mbit/1088bit per frame = max 91911 frames/s
- **This rate is only given when both the Request and Response are a single Frame!**
  - One-Step operation
  - Two-Step will divide the performance by a factor of 2

Performance

Measurements CSPTP

- Sent 1Mio Requests (~10s)
- Received 1Mio Responses
- Frame rate was ~92k Requests/s
- No Frame drops





Performance
Simulation

NetTimeLogic GMBH

- **FPGA designs can be easily simulated e.g. with Mentor Modelsim®**
- **Simulate a 1G GMII interface and full line speed of Requests**
- **Check if Response is received for every Request and if timestamps correct**
  - Full line speed 1G of requests can be handled by the NTP Server
  - Depending on the CSPTP mode full line speed 1G of requests can be handled, if responses are larger than the requests some requests need to be dropped and the long term response rate is response-size/request-size (except initially where some buffering still works)
    - This is in principal amplification, FlashPTP avoids this by padding Requests to the size of Responses

Performance
Simulation

- **Simulations shows that the theoretical max can be achieved for 1Gb**
  - For NTP >1000000 Requests/s
  - For CSPTP >910000 Requests/s

- **The max when the internal bus AXIS without interface adapter (no interframe gap and preamble) is used is even ~20% higher**
  - This is interesting since we will show later how we can go to even higher rates

- **Testing on a different platform with a 1G port prove the simulation result**

Limitations

What is limiting the Performance?

NetTimeLogic GMBH

- **Link Speed**
- **Clock Frequency of the Core**
- **Interleaved Two-Step Requests for CSPTP are bad**
  - Two Step requests (for FlashPTP) are bad because the Server must merge Sync and FollowUp to have all information for a Response
  - In a situation of a very high request rate a Sync and FollowUp pair could be interleaved by other Syncs and FollowUps from other Clients which would require buffering and lookup of pairs
  - Current scheme is to expect only Sync and FollowUps from a Client back-to-back and drop if not a pair => IEEE1588.1 should limit the request to a single message as for NTP

Scheme for higher Link Speeds

How can this be applied to 10/25/100/400G Links

NetTimeLogic GMBH

- **Create an Interface Adapter for 10/25/100/400G Links**
  - Only run the required parts on high frequencies, wide bus, Core not
- **>10G requires also higher performance FPGAs due to ultra wide bus interfaces and high frequencies**
  - More power, more expensive
- **Add a filter for NTP/CSPTP and ARP/ICMP frames to drop unneeded frames to have the best use of the bandwith**
- **Reduce internal bandwidth to what can be done on the internal interface**
- **Deterministic buffering for downscaling and upscaling of bandwidth for hardware timestamping**

Scheme for higher Request Rates

How to go above 1Mio Request/s on higher Link Speeds

NetTimeLogic GMBH

- **Increase Clock Frequency of core: e.g. 100MHz instead of 50MHz**
  - Only ~doubles the throughput
  - Makes timing closure difficult or impossible on low performance FPGAs
  - Needs at some point higher performance FPGAs => more power, more expensive
  - Quite limited in scaling (~350Mhz is the max in a low range FPGA, for a simple design and not a more complex component as this)

Scheme for higher Request Rates

How to go above 1Mio Request/s on higher Link Speeds

NetTimeLogic GMBH

- **Instantiate the core multiple times**
  - Uses more resources but also gives higher performance
  - Scales well (as long as there are resources)
  - Max request rate per core is ~1.2Mio/s @ 50MHz
    - Roughly 1Gbit is 1Mio Requests/s meaning 1 core per Gbit
    - For full 10Gbit line speed Request processing => 10 cores
- **Create a load balancer to share between multiple cores**
  - Interface bandwidth to and from the core (~1.2G) are the limiting factor
  - Use a FIFO per core: fill with line Speed (e.g. 10G) and pull with ~1.2G
  - Check which FIFO has the most space left and fill this, if equal take next then previous, if non has space for a frame, drop it => shortest response time and equal share of load

Scheme for higher Request Rates

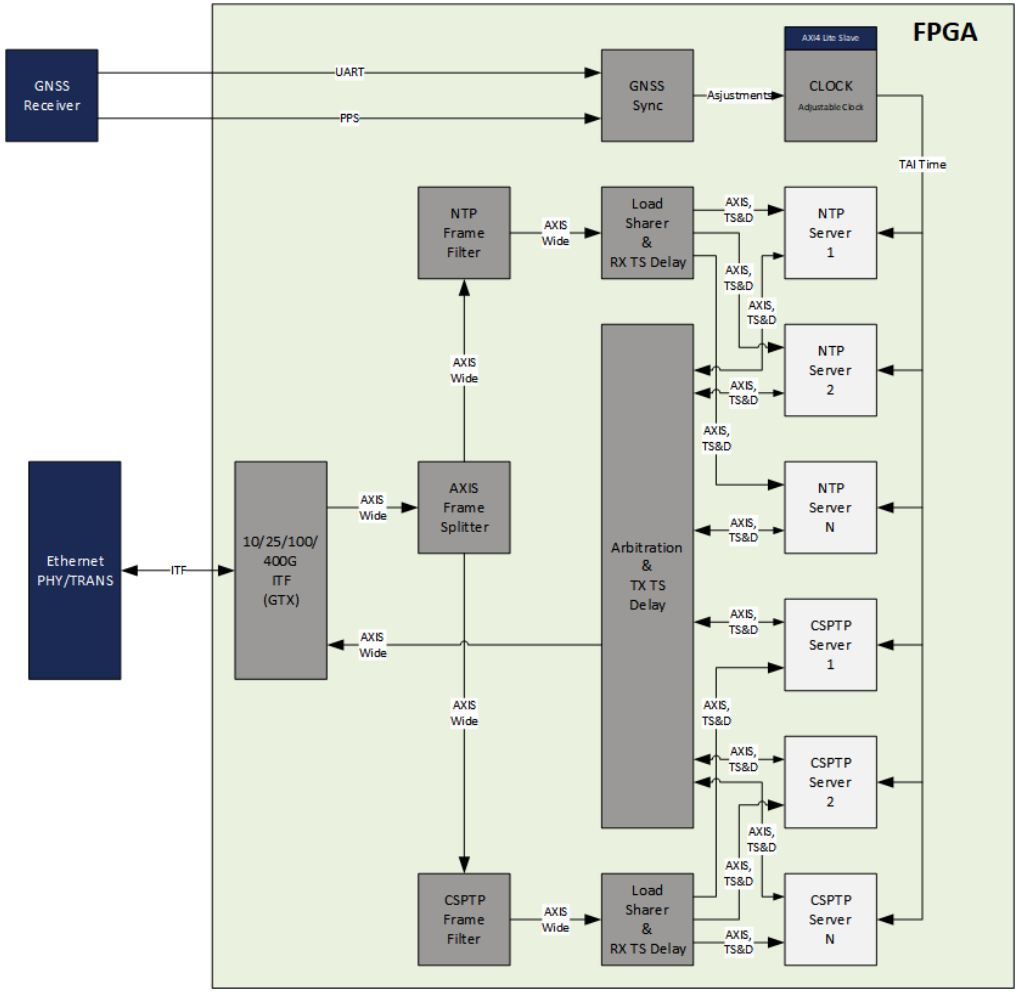
How to go above 1Mio Request/s on higher Link Speeds

NetTimeLogic GMBH

- **Arbitrate between multiple cores**
  - Create a deterministic latency from the timestamp signaling and actual sending
  - Bandwidth conversion from N times ~1.2G to e.g. 10G

Scheme for higher Request Rates

How to go above 1Mio Request/s on higher Link Speeds



Scheme how to integrate Security
AUTHENTICATION TLV for (CS)PTP?

NetTimeLogic GMBH

- **PTP foresees an AUTHENTICATION TLV for authentication of PTP messages**
  - Authenticates the whole message (direct mode)
- **All TCs must also recalculate the signing because of altering frames**
- **Symmetric Key exchange is something to be done on a CPU**
  - E.g. on a softcore CPU
  - It is important that no table is required on a per client base (group keys)
- **This can be added as add-on in a pipelined manner to the current design**
  - It must be assured that authentication and signing don't take longer than the frame reception/transmission

Scheme how to integrate Security
NTS?

NetTimeLogic GMBH

- **NTS is widely used for NTP in the public domain**
  - Especially for public Servers
  - For closed environments like a Datacenter this might be overkill
- **NTS for CSPTP does not exist yet**
  - This could potentially be added to CSPTP as option
- **Still stateless scheme for Server**
- **NTS is based on a trusted common Key Establishment Server for initial Key establishment and cookies**
  - This could be done on a CPU, since it is basically used initially only
  - It is ok that this takes a while

Scheme how to integrate Security
NTS?

NetTimeLogic GMBH

- **Cookies in the Client Request can be authenticated by the Server and the Cookies in the Server Response can be authenticated on the Client**
  - It must be assured that authentication can be done pipelined and will not take longer than a frame reception
  - It must be assured that signing can be done pipelined and will not take longer than a frame transmission
- **Response Cookie is based on the Request Cookie, Server Keys and Nonce with AEAD Algorithm (AES)**
  - Cookie generation also needs to be pipelined and must not take longer than the reception of a frame

Scheme how to integrate Security
NTS?

NetTimeLogic GMBH

- **Authentication and Signing can be pretty good pipelined in FPGAs**
  - However, pipelining is limited by the duration of a frame
  - At some point pipeline steps need to do more per cycle which means the resource usage goes up quite badly
- **Integration of NTS in the current concept (without KE) is possible**
  - Authentication can be done as a pipelined task as add-on
  - Cookie creation can be done as a pipelined task as add-on
  - Signing can be done as a pipelined task as add-on
- **Key Exchange could be moved to a Soft-core CPU**
  - Either only the Key Exchange part
  - Or also the KE-Server on the same CPU (as many NTP Servers do)

Scheme how to integrate Security

NTP vs CSPTP

- Since the schemes are not necessary the same for NTP and CSPTP it now made perfect sense to split the two designs to avoid to complicate the design

Conclusion

- A fully FPGA based NTP and CSPTP Server can achieve outstanding high performance
- The performance to power to costs ratio is unbeatable
  - 1Mio Requests with <3 Watts on an FPGA for <40$
- The fully FPGA based solution can scale quite easily to N-Mio requests per second with reasonable effort and costs
- Security aspects can be handled very efficient on the FPGA due to pipelining and full parallelism however authentication/signing uses a lot of resources and Key Exchange (NTS-KE Server, PTP Key exchange) are not useful to be built in FPGAs

Questions?

NetTimeLogic GMBH

Thank you!

See Network Timing, Network Redundancy and our modular Hardware Platform AIONYX live at our booth!!!

www.nettimelogic.com

contact@nettimelogic.com