Moxa Computer Time-synchronization Settings for IEEE 1588 and Precision Time Protocol

Version 1.0, June 2022

www.moxa.com/products

Moxa Computer Time-synchronization Settings for IEEE 1588 and Precision Time Protocol

Copyright Notice

Trademarks

Disclaimer

Technical Support Contact Information

www.moxa.com/support

Table of Contents

1. Overview

This guide describes the IEEE 1588 and Precision Time Protocol (PTP) settings in Windows and is applicable to the following Moxa products:

Product Series	Supported
DA-820C	Yes
DA-682C	Coming soon
DA-681C	Coming soon
DA-720	Coming soon

2. Windows PTP Client Settings

This chapter describes how to configure the PTP Client on a Windows 10 system.

Hardware and Software Requirements

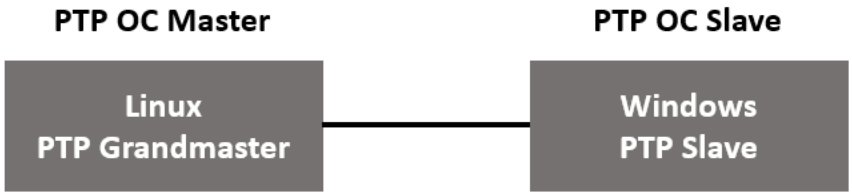
The hardware and software requirements are listed here:

- **Hardware**: Network interface cards (NICs) with IEEE 1588 support (e.g., Intel® I210)
- **Software**: Windows OS kernel with PTP hardware timestamp support (e.g., Windows 10 Pro 20H2 or later); build 19042 or later

  To check the build number, run **`winver`** from the Windows start menu. Confirm that the build version is 19042 or later.

- **Other**: One Linux device as PTP Grandmaster and one or more Windows devices as PTP Slaves (Clients)

  Because Windows only supports the PTP slave (client) in OC mode, we will need another device with Linux OS to be the PTP Grandmaster to perform the time synchronization.

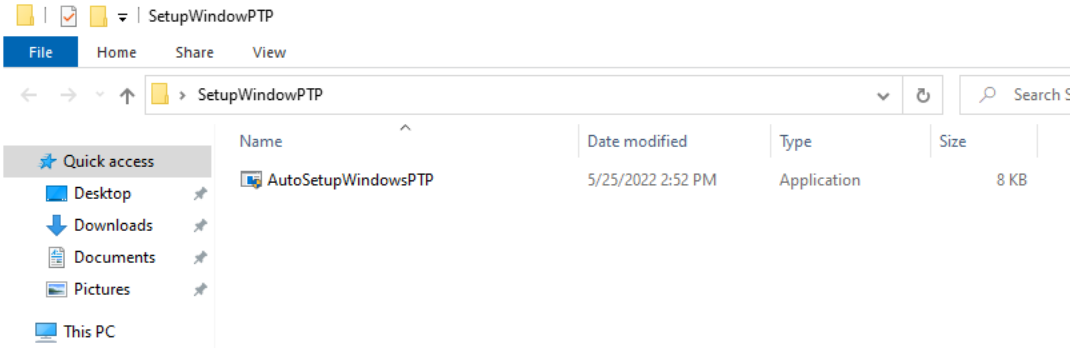


Setting Up the Windows PTP Slave (Client)

The Windows Time Service (w32tm) is a Windows service that keeps your computer clock accurate. We will be using the **AutoSetupWindowsPTP.exe** file to configure the w32tm service and apply the PTP function.

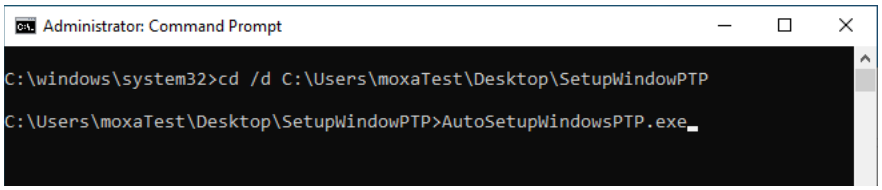
NOTE

Contact a Moxa representative for the **AutoSetupWindowsPTP.exe** file to configure and run the PTP function on your computer.

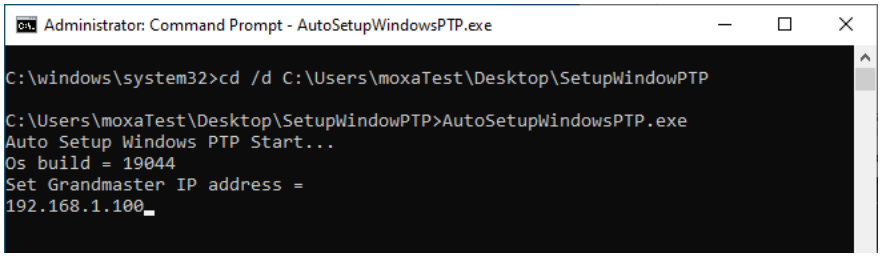


To set up the Windows PTP Slave (Client), do the following:

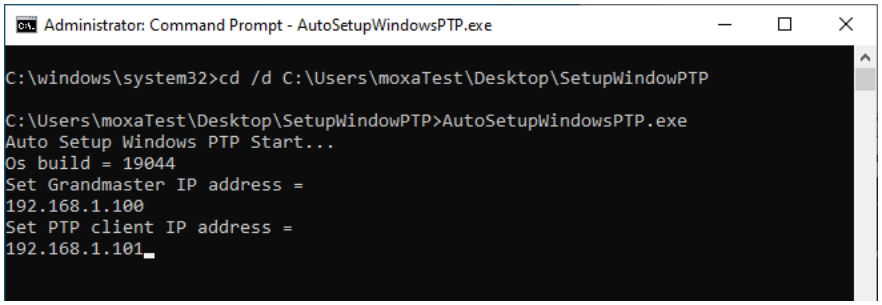
1. Run cmd from the Windows start menu and then run the **AutoSetupWindowsPTP.exe** file.



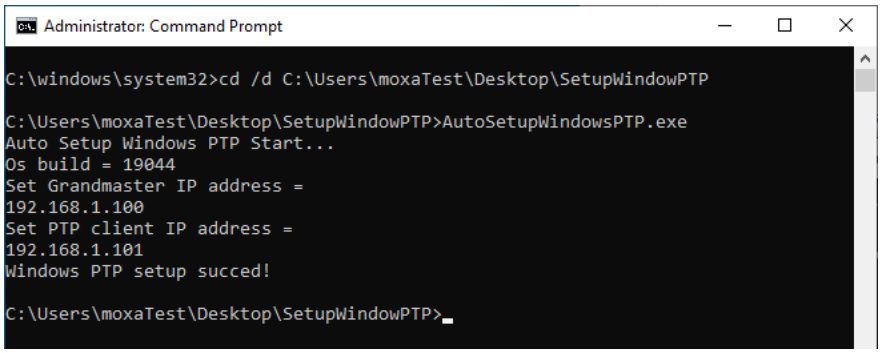
2. Enter the IP for the Linux PTP Grandmaster.



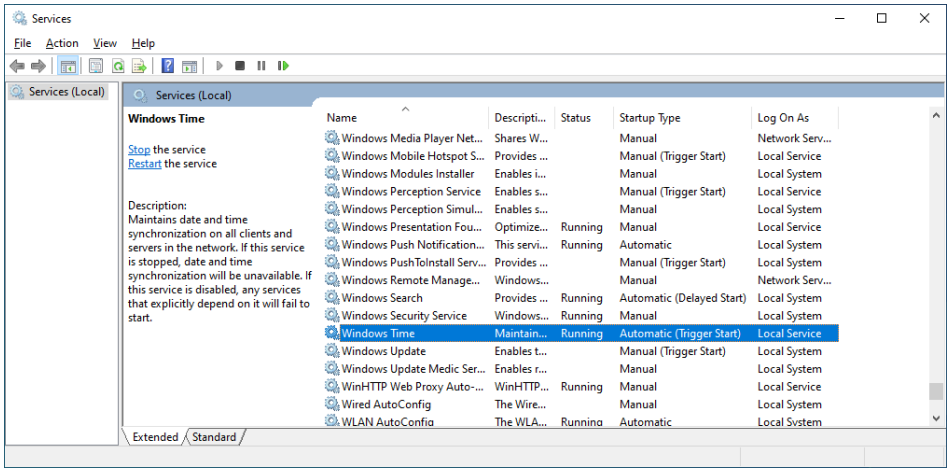
3. Enter the IP address for the PTP Slave.



4. Wait until the program sets up the configuration for the w32tm service for the PTP function.



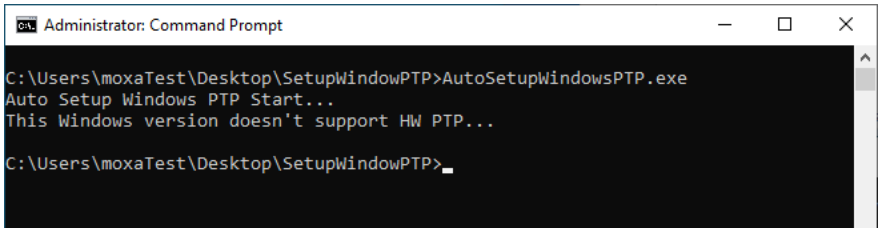
After the setup process is completed, the Windows Time service will start automatically.



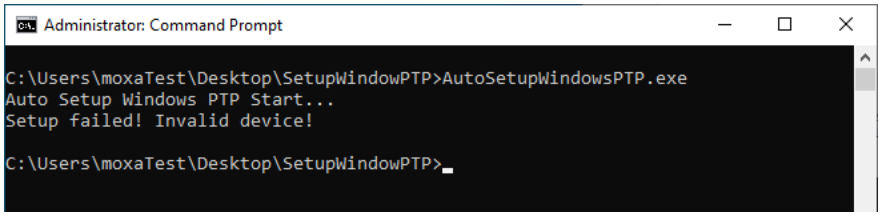
5. Restart the device to apply the configuration to the OS.

Additional Information

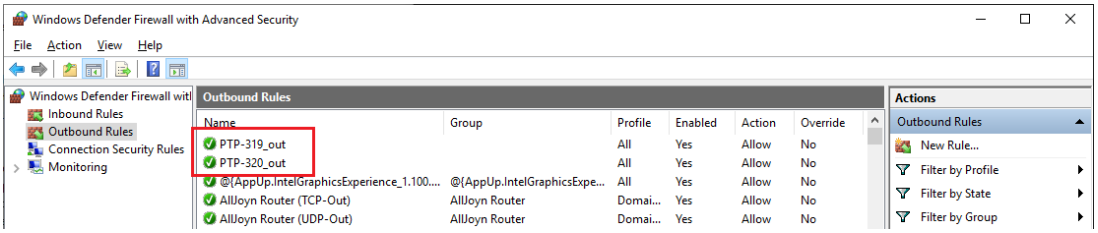
- If the program shows the error "This Windows version doesn't support HW PTP", the Windows version is too old to support HW PTP. Install a newer supported version on your device.

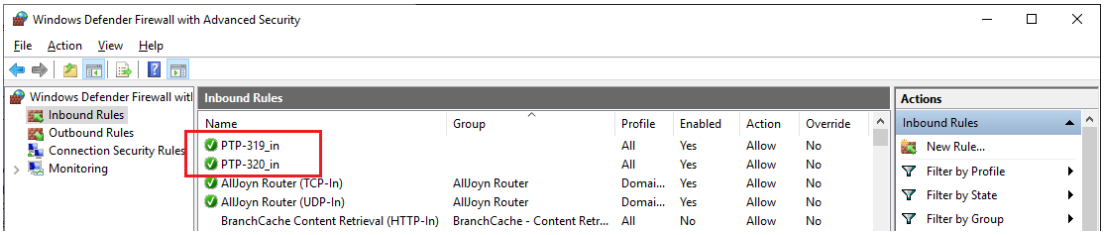


- If the program shows the error "Invalid device!", the program cannot setup Windows PTP on the device. Note that only Moxa devices are supported.



- PTP messages may use the User Datagram Protocol over Internet Protocol (UDP/IP) for transport. The **AutoSetupWindowsPTP.exe** program will automatically create the firewall rules to allow the PTP Slave to communicate with the time server.





Configuring the PTP Grandmaster Message Settings

To configure the PTP message settings for the Grandmaster and generate a PTP message based on the settings, do the following:

1. Use the follow settings to configure a PTP massage on the Linux PTP Grandmaster device.

Delay Mechanism	E2E
Network Transport	UDP IPV4
Time Stamping	Hardware
Multicast	Enable
ptpTimescale	1

2. On the Linux PTP Grandmaster device, run the **ptp4l** command.

   `ptp4l -E -4 -H -m -i enp2s0`

3. After the **`ptp4l`** starts, type the following command to show the configuration settings.

   `sudo pmc -u -b 0 'GET GRANDMASTER_SETTINGS_NP'`

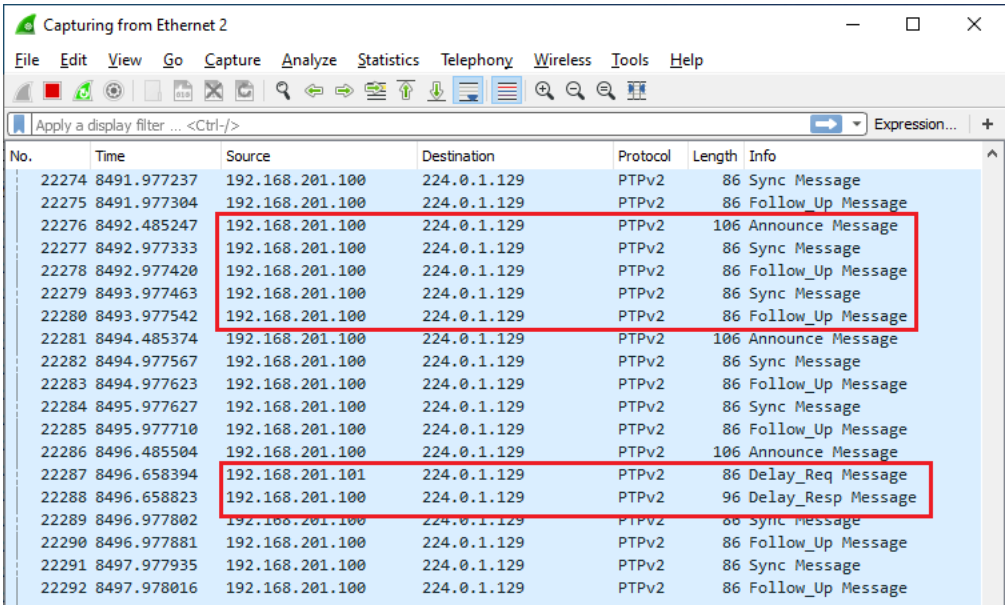
4. To modify the setting when the ptp4l is running, run the following command:

   `sudo pmc -u -b 0 \ "SET GRANDMASTER_SETTINGS_NP clockClass 248 clockAccuracy 0xfe offsetScaledLogVariance 0xffff currentUtcOffset 37 leap61 0 leap59 0 currentUtcOffsetValid 0 ptpTimescale 1 timeTraceable 0 frequencyTraceable 0 timeSource 0xa0"`

Checking the PTP Time Sync Function

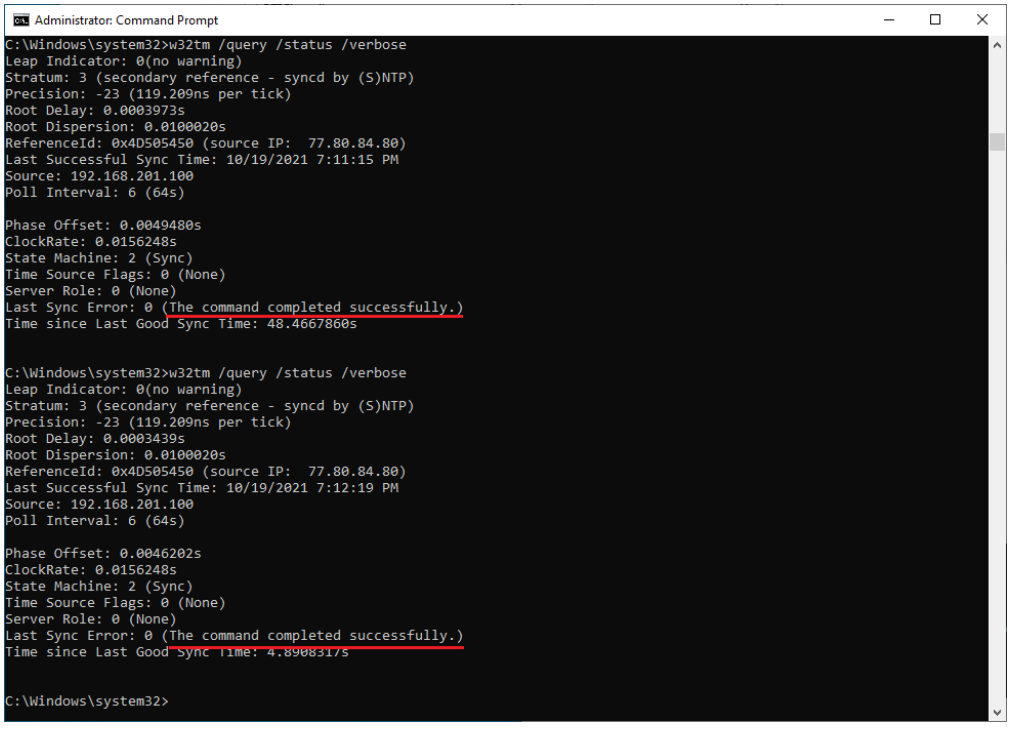
After the PTP Grandmaster settings are completed, the Windows time service will start automatically and check for the PTP message from the PTP Grandmaster.

- Using WireShark to check the message information from the specific LAN.
  a. The PRP Master will generate a "Announce Msg + Sync Msg + Follow_Up Msg + Sync Msg + Follow_Up Msg" at each loop.
  b. The W32Time service will send a Delay_Req Msg to the PTP master and PTP master will respond with a Delay_Resp Msg for the time sync.



- In the Windows Start menu, run **cmd** as an administrator and enter **w32time.exe /query /status /verbose.**

  If the "Last Sync Error" shows "0 (The command completed successfully)", the time sync was successful.



Prerequisites

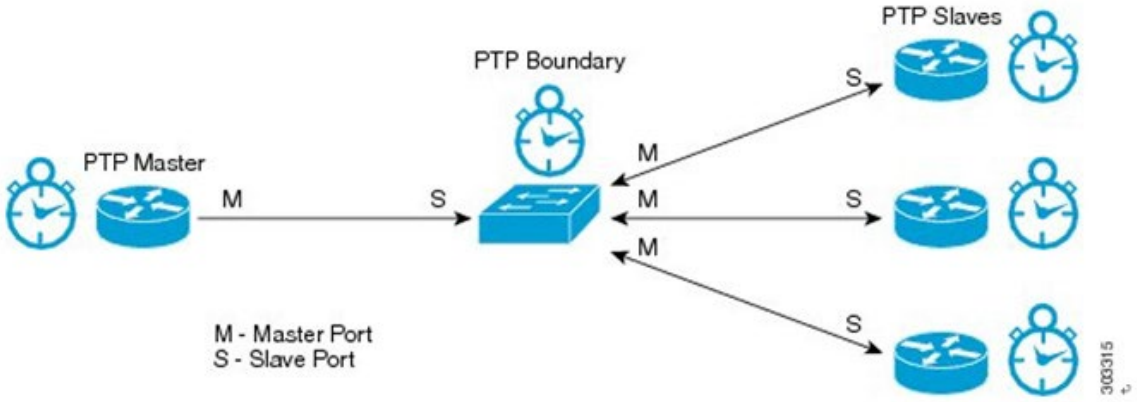
1. Install Debian 11.
2. Install the **linuxptp** package.

```
apt update
apt upgrade
apt install ssh linuxptp ethtool build-essential
```

3. Disable the **systemd time sync** daemon service to avoid unexpected operations.

```
systemctl stop systemd-timesyncd
systemctl disable systemd-timesyncd
```

A Simple Topology



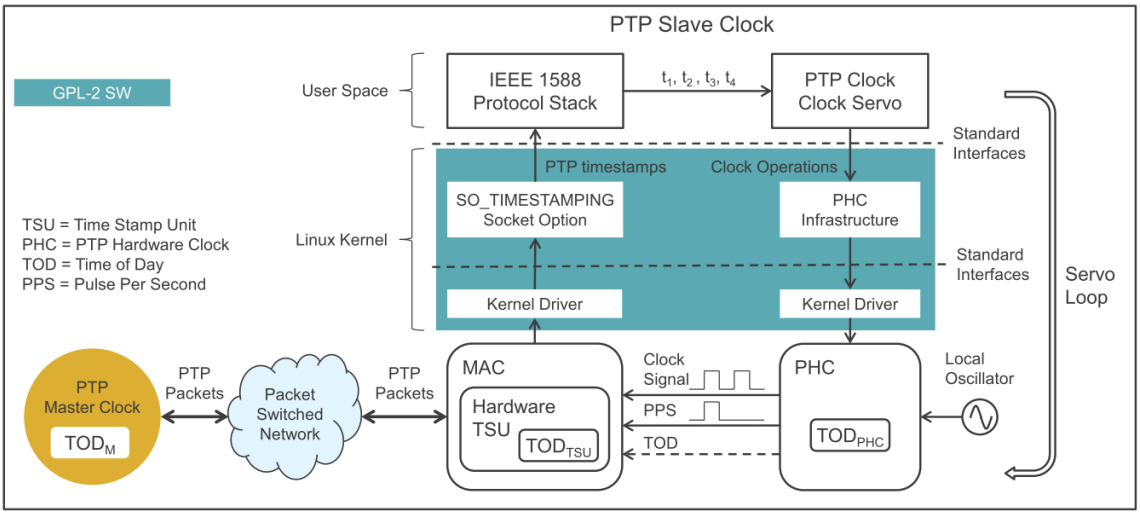


Figure 2. Typical PTP Slave Clock Implementation Using Hardware Timestamping

Debian linuxptp Package

The **linuxptp** is an implementation of the Precision Time Protocol (PTP) for Linux according to the IEEE standard 1588. Features include:

- Support for hardware and software time stamping via Linux
- SO_TIMESTAMPING socket option
- Support for the Linux PTP Hardware Clock (PHC) subsystem by using the **clock_gettime** family of calls, including the **new clock_adjtimex** system call
- Implementation of Boundary Clock (BC) and Ordinary Clock (OC)
- Transport over UDP/IPv4, UDP/IPv6, and raw Ethernet (Layer 2)
- Support for IEEE 802.1AS-2011 in the role of an end station

Additional information is available at: https://packages.debian.org/bullseye/linuxptp

PTP provides higher precision and faster synchronization than NTP even without hardware support. And, with hardware support, sub-microsecond accuracy can be expected. Whereas NTP is intended for WAN use. PTP is designed for LAN environments and makes use of UDP multicast.

```
usage: ptp4l [options]

 Delay Mechanism

 -A        Auto, starting with E2E
 -E        E2E, delay request-response (default)
 -P        P2P, peer delay mechanism

 Network Transport

 -2        IEEE 802.3
 -4        UDP IPV4 (default)
 -6        UDP IPV6

 Time Stamping

 -H        HARDWARE (default)
 -S        SOFTWARE
 -L        LEGACY HW

 Other Options

 -f [file] read configuration from 'file'
 -i [dev]  interface device to use, for example 'eth0'
           (may be specified multiple times)
 -p [dev]  Clock device to use, default auto
           (ignored for SOFTWARE/LEGACY HW time stamping)
 -s        slave only mode (overrides configuration file)
 -l [num]  set the logging level to 'num'
 -m        print messages to stdout
 -q        do not print messages to the syslog
 -v        prints the software version and exits
 -h        prints this message and exits
```

OC Mode

Layer 2

Set as:

- OC master (as a PTP Grandmaster) mode
- Layer 2
- P2P mode, peer delay mechanism

```
# Assume interface device is 'enp4s0'
ptp4l -m -2 -P -i enp4s0
```

Set as:

- OC slave mode
- Layer 2
- P2P mode, peer delay mechanism

```
# Assume interface device is 'enp5s0'
ptp4l -m -2 -P -s -i enp5s0

# with log: ptp4l -m -2 -s -P -i enp5s0 2>&1 | tee $(date
+%Y%m%d%H%M%S.log)
```

Layer 4 (UDP IPV4)

Set as:

- OC master mode
- Layer 4
- P2P mode, peer delay mechanism

```
# Assume interface device is 'enp4s0'
ptp4l -m -4 -P -i enp4s0
```

Set as:

- OC slave mode
- Layer 4
- P2P mode, peer delay mechanism

```
# Assume interface device is 'enp5s0'
ptp4l -m -4 -P -s -i enp5s0

# with log: ptp4l -m -4 -s -P -i enp5s0 2>&1 | tee $(date
+%Y%m%d%H%M%S.log)
```

BC Mode

1. Set as BC mode.

   a. clock_type

   Specifies the PTP clock type. Valid values are "OC" for ordinary clock, "BC" for boundary clock, "P2P_TC" for peer-to-peer transparent clock, and "E2E_TC" for end-to-end transparent clock. A multi-port ordinary clock will automatically be configured as a boundary clock. **The default is "OC"**.

   b. boundary_clock_jbod

   When running as a **boundary clock** (that is, when more than one network interfaces are configured), **ptp4l** performs a sanity check to ensure that all the ports share the same hardware clock device. This option allows **ptp4l** to work as a boundary clock using a bunch of devices that are not synchronized to each other. For this mode, the collection of clocks must be synchronized by an external program, for example **phc2sys(8)** in **automatic** mode. The default is 0 (disabled).

```
# For example, edit config file 'bc.cfg'
# and assume 'enp12s0' and 'enp4s0' are connected network interface
[global]
sanity_freq_limit        0
step_threshold           0.000002
tx_timestamp_timeout     10
logMinPdelayReqInterval  0
logSyncInterval          0
logAnnounceInterval      0
announceReceiptTimeout   3
syncReceiptTimeout       2
twoStepFlag              1
summary_interval         0
clock_type               BC
priority1                128
priority2                127
delay_mechanism          P2P

[enp12s0]
boundary_clock_jbod          1
network_transport            UDPv4
fault_reset_interval         0

[enp4s0]
boundary_clock_jbod          1
network_transport            UDPv4
fault_reset_interval         0

# run the ptp4l procedure
ptp4l -m -f bc.cfg

# use phc2sys to sync sys clock for 10Hz
phc2sys -a -m -r -R 10
```

2. Set as OC master (p2p).

```
# edit 'oc_gm_p2p.cfg', assume 'enp5s0' as NIC
[global]
twoStepFlag 1
priority1               127
masterOnly 1
delay_mechanism         P2P

[enp5s0]
network_transport UDPv4

# exec as OC master
ptp4l -m -f oc_gm_p2p.cfg
```

3. Set as OC slave.

```
# edit 'oc_sl.cfg', assume 'enp4s0' as NIC
[global]
twoStepFlag 1
slaveOnly 1
delay_mechanism         P2P

[enp4s0]
network_transport UDPv4

# exec as OC slave
ptp4l -m -s -f oc_sl.cfg
```

phc2sys

The phc2sys program, included in the **linuxptp** package, synchronizes two or more clocks in the system. Typically, it is used to synchronize the system clock to a PTP hardware clock (PHC), which itself is synchronized by the **ptp4l(8)** program.

```
phc2sys -a -m -r -R 10
```

Additional information is available at: https://manpages.debian.org/bullseye/linuxptp/phc2sys.8.en.html

One Pulse Per Second (1PPS)

One Pulse Per Second (1PPS) is a time synchronization feature that allows the adapter to be able to send or receive 1 pulse/sec on a dedicated pin on the adapter card.

Example Code for Setup

In this example we have two servers connected back-to-back with the 1PPS pin on the adapter. One server will generate the 1PPS signal (1PPS out), while the other server will receive the 1PPS signal (1PPS in).

https://raw.githubusercontent.com/torvalds/linux/master/tools/testing/selftests/ptp/testptp.c

```
wegt
https://raw.githubusercontent.com/torvalds/linux/master/tools/testing/sel
ftests/ptp/testptp.c
gcc -Wall -lrt testptp.c -o testptp

./testptp -h
usage: testptp [options]
 -c          query the ptp clock's capabilities
 -d name     device to open
 -e val      read 'val' external time stamp events
 -f val      adjust the ptp clock frequency by 'val' ppb
 -g          get the ptp clock time
 -h          prints this message
 -i val      index for event/trigger
 -k val      measure the time offset between system and phc clock
             for 'val' times (Maximum 25)
 -l          list the current pin configuration
 -L pin,val configure pin index 'pin' with function 'val'
             the channel index is taken from the '-i' option
             'val' specifies the auxiliary function:
             0 - none
             1 - external time stamp
             2 - periodic output
 -p val      enable output with a period of 'val' nanoseconds
 -H val      set output phase to 'val' nanoseconds (requires -p)
 -w val      set output pulse width to 'val' nanoseconds (requires -p)
 -P val      enable or disable (val=1|0) the system clock PPS
 -s          set the ptp clock time from the system time
 -S          set the system time from the ptp clock time
 -t val      shift the ptp clock time by 'val' seconds
 -T val      set the ptp clock time to 'val' seconds
 -z          test combinations of rising/falling external time stamp flags
```

Configuration

Use **ethtool** to check the Time Synchronization support on the relevant port

```
# Assume HSR-PRP-I210 interlink interface is 'enp4s0'

# To get 'PTP Hardware Clock' index
root@ptp2:/home/moxa# ethtool -T enp4s0
Time stamping parameters for enp4s0:
Capabilities:
        hardware-transmit
        software-transmit
        hardware-receive
        software-receive
        software-system-clock
        hardware-raw-clock
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off
        on
Hardware Receive Filter Modes:
        none
        all

# Get index is '0', the corresponding ptp device node is '/dev/ptp0'

# Query the ptp clock's capabilities.
root@ptp2:/home/moxa# ./testptp -d /dev/ptp0 -c
capabilities:
  62499999 maximum frequency adjustment (ppb)
  0 programmable alarms
  2 external time stamp channels
  2 programmable periodic signals
  1 pulse per second
  4 programmable pins
  0 cross timestamping
  0 adjust_phase

# Check the current pin configuration:
# name: Pin name.
# index: The Pin number.
# func (function): The pin Configuration
# 0 - none
# 1 - 1PPS In
# 2 - 1PPS Out (periodic output)
# chan (channel) - Reserved field.

root@ptp2:/home/moxa# ./testptp -d /dev/ptp0 -l
name SDP0 index 0 func 2 chan 0
name SDP1 index 1 func 0 chan 0
name SDP2 index 2 func 0 chan 0
name SDP3 index 3 func 0 chan 0
```

Setting Up the 1PPS In Server

```
# Assume PTP device node is '/dev/ptp0'

# Setup function as '1' (1PPS In) with default channel 0
root@ptp1:/home/moxa# ./testptp -d /dev/ptp0 -L 0,1

# Check the func value was changed.
root@ptp1:/home/moxa# ./testptp -d /dev/ptp0 -l
name SDP0 index 0 func 1 chan 0
name SDP1 index 1 func 0 chan 0
name SDP2 index 2 func 0 chan 0
name SDP3 index 3 func 0 chan 0

# On the 1PPS in server
# set the maximum number of events to be set/handled by the application.
root@ptp1:/home/moxa#./testptp -d /dev/ptp0 -e 1000
```

Some applications may need to read the timestamp from a file. To write the time stamps into a file, enable the **pps_enable sysfs** parameter in the **1 PPS in server**.

```
root@ptp1:/home/moxa# echo 1 > /sys/class/ptp/ptp0/pps_enable
root@ptp1:/home/moxa# watch -n 1 'cat /sys/class/pps/pps0/assert'
```

No changes are needed on the **1 PPS out server**.

To disable it again and get the timestamps to the terminal, run the following commands:

```
root@ptp1:/home/moxa# echo 0 > /sys/class/ptp/ptp0/pps_enable
root@ptp1:/home/moxa#./testptp -d /dev/ptp0 -e 1000
```

Setting Up the 1PPS Out Server

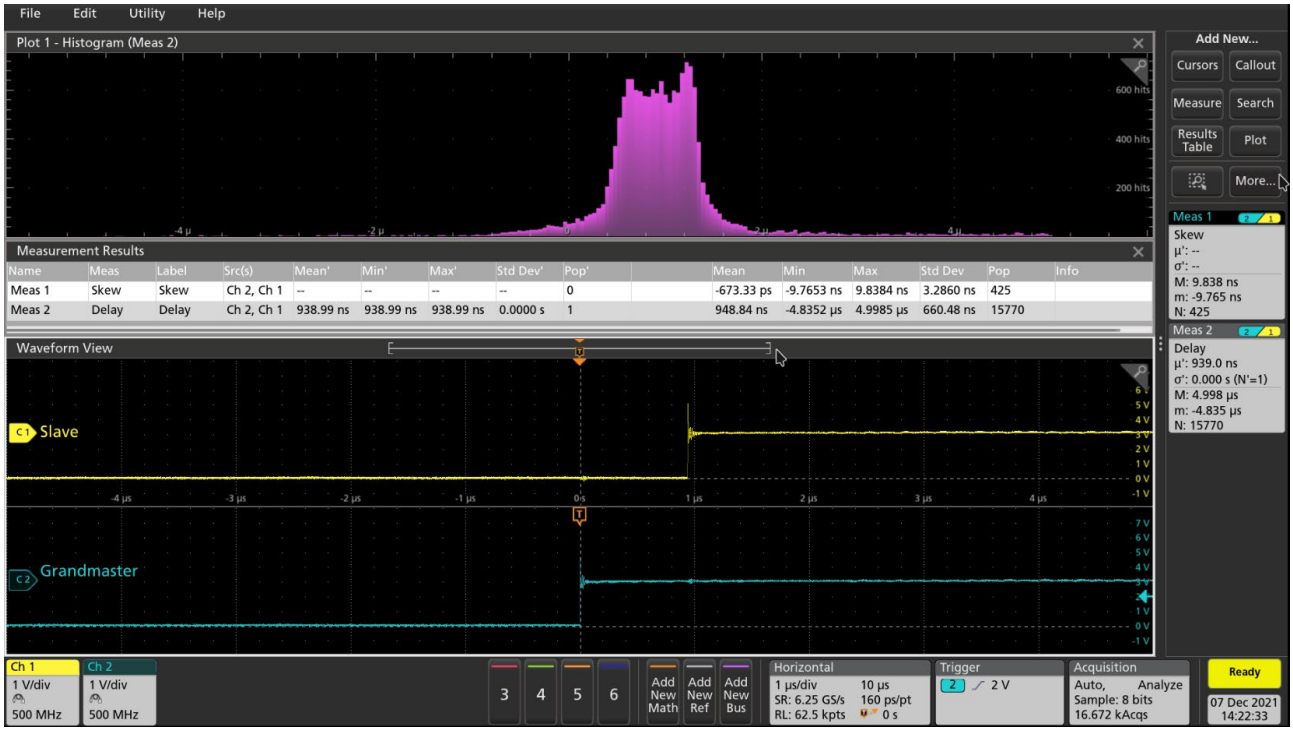
```
# Assume PTP device node is '/dev/ptp0'

# Setup function as '2' (1PPS Out) with default channel 0
root@ptp2:/home/moxa# ./testptp -d /dev/ptp0 -L 0,2

# Check the func value was changed.
root@ptp2:/home/moxa# ./testptp -d /dev/ptp0 -l
name SDP0 index 0 func 2 chan 0
name SDP1 index 1 func 0 chan 0
name SDP2 index 2 func 0 chan 0
name SDP3 index 3 func 0 chan 0

# On the 1PPS out server, enabled 1PPS using -p 1000000000 (in nsec)
root@ptp2:/home/moxa# ./testptp -d /dev/ptp0 -p 1000000000
```

Example for Latching Signal SDP0



Additional References

- https://www.kernel.org/doc/html/latest/driver-api/ptp.html
- https://community.mellanox.com/s/article/How-To-Test-1PPS-on-Mellanox-Adapters#jive_content_id_1PPS_out
- https://documentation.suse.com/zh-tw/sled/15-SP2/html/SLED-all/cha-tuning-ptp.html
- REN_Linux-Kernel-Supp-IEEE-1588-HW-TimeS_WHP_20210129.pdf