

AN15137

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

Rev. 1.0 — 12 October 2026

Application note

Document information

Information	Content
Keywords	AN15137, MCX A577, FreeRTOS, IEEE 1588 V2, PTP, lwIP
Abstract	This application note describes the implementation of the IEEE 1588 V2 Precision Time Protocol (PTP) on the MCX A577 MCUs running FreeRTOS OS.



1 Introduction

This application note describes the implementation of the IEEE 1588 V2 Precision Time Protocol (PTP) on the MCX A577 MCUs running FreeRTOS OS. The IEEE 1588 standard provides accurate clock synchronization for distributed control nodes in industrial automation applications.

The implementation runs on the FRDM-MCXA577 board with the MCX A577 MCU. The demo software is based on the NXP MCUXpresso SDK for the FRDM-MCXA577 board. The demo is a Precision Time Protocol daemon (PTPd) using the lwIP TCP/IP stack shipped with the MCUXpresso SDK and runs on the FreeRTOS operating system. PTPd is an open source implementation of the PTP.

The demo supports two Ethernet network interface options:

1. Standard Ethernet (10/100BASE-T): The onboard LAN8741 Ethernet PHY is connected to the MCU through the MII/RMII interface. It provides network connectivity via the RJ45 connector, allowing communication with other network nodes over a standard Ethernet network.
2. 10BASE-T1S Single-Pair Ethernet: The MCU's integrated 10BASE-T1S digital PHY is connected to an external TC14 PMD transceiver (TJF1410A on the FRDM-MCXA577 board). Through a single twisted-pair cable, the board can communicate with other 10BASE-T1S network nodes.

The Ethernet MAC can be connected to either the external LAN8741 PHY or the internal 10BASE-T1S digital PHY through the SYSCON ENET_CTRL register. As a result, the same PTPd software framework operates over both conventional Ethernet and 10BASE-T1S networks.

This document introduces the basics of the IEEE 1588 protocol and describes the IEEE 1588 features supported on the MCX A577 MCU. It also provides a detailed explanation of the IEEE 1588 demo software, including the procedure for porting PTPd to Amazon FreeRTOS on the MCX A577 MCU. In addition, the document explains how to enable the ENET output compare function to monitor clock synchronization status. This document also describes how to build and run the demo.

2 IEEE 1588 basic overview

The IEEE 1588 standard is known as the Precision Clock Synchronization Protocol for Networked Measurement Control Systems, also known as Precision Time Protocol (PTP). IEEE 1588 PTP enables the clocks to be distributed across an Ethernet network and accurately synchronized using a process where the distributed nodes exchange timestamped messages.

The technology of the standard, originally developed by Agilent Technologies, Inc., is used for distributed measuring and control tasks. The challenge is to synchronize the networked measuring devices with each other in terms of time. Once synchronized, the devices can record measured values and provide them with a precise system timestamp. Based on this timestamp, the measured values can then be correlated with each other.

Typical applications of the IEEE 1588 time synchronization include:

- Time-sensitive telecommunication services that require precise time synchronization between communicating nodes.
- Industrial network switches that synchronize sensors and actuators over a single-wire distributed control network to control automated assembly processes.
- Powerline networks that synchronize across large-scale distributed power grid switches to enable smooth transfer of power.
- Test/measurement devices that must maintain accurate time synchronization with the device under test in many different operating environments.
- Printing machines, cooperative robotic systems, and residential Ethernet.

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

These applications require precise clock synchronization between the devices with accuracy in the sub-microsecond range. It is a remarkable feature of IEEE 1588 that this synchronization precision is achieved through regular Ethernet connectivity with standard Ethernet frames.

This solution enables nearly any device of any performance to participate in high-precision synchronized networks that are simple to operate and configure.

Other key benefits of the IEEE 1588 protocol include:

- Convergence times of less than a minute for sub-microsecond synchronization between heterogeneous distributed devices with different clocks, resolution, and stability.
- Automatic configuration and segmentation. Each node uses the Best Master Clock (BMC) algorithm to determine the best clock in the segment. Every PTP node stores its features within a specified dataset. These features are transmitted to other nodes within sync telegrams. Based on this information, the other nodes synchronize their data sets with the features of the actual master and can adjust their clocks. The cyclic running of the BMC also allows hot swapping; that is, nodes can be connected or removed during propagation time.
- Simple configuration and operation with low computing resource requirements and network bandwidth consumption.

2.1 Synchronization principle

Network clocks are organized in a master-slave hierarchy. IEEE 1588 identifies the master clock. It then establishes a two-way timing exchange by which the master sends messages to its slaves to initiate synchronization. Each slave then responds to synchronize itself to its master. This sequence is repeated throughout the specified network to achieve and maintain clock synchronization.

The process starts with one node (master clock) transmitting a sync telegram that contains the estimated transmission time. A clock captures the exact transmission time of the sync telegram. It is then transmitted in a second follow-up message. The timestamp information contained in the first and second telegrams is compared with its own clock. The receiver then calculates the time difference between its clock and the master clock (see [Figure 1](#)). The sync and follow-up messages are sent as a multicast. Some IEEE 1588 systems enable hardware timestamping and the insertion of actual timestamps into the sync messages. For this case, the follow-up messages are not needed (one-step mode of operation).

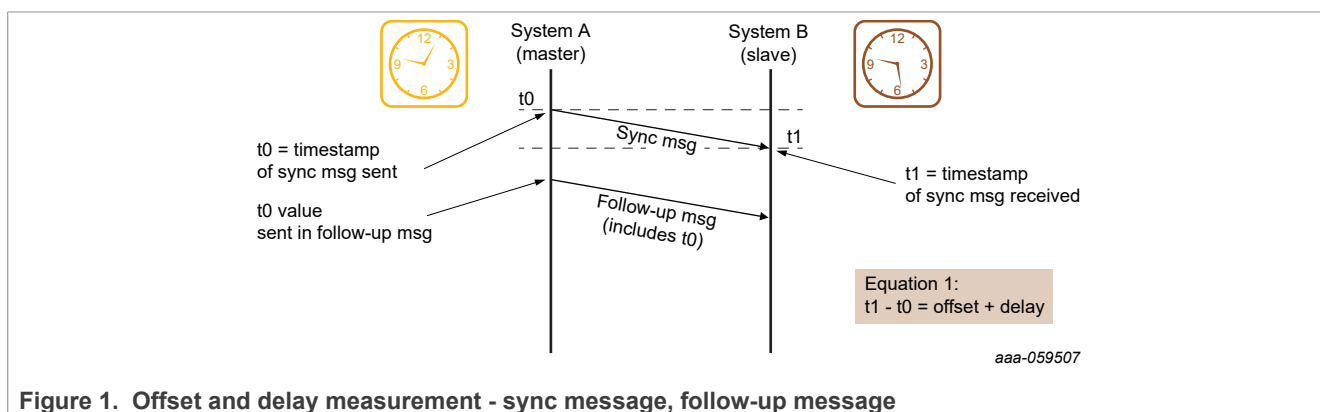


Figure 1. Offset and delay measurement - sync message, follow-up message

The telegram propagation time is determined cyclically in a second transmission process between the slave and the master (delay telegrams). The slave can then adjust its clock and adapt it to the current bus propagation time (see [Figure 2](#)). The delay_req and delay_resp messages are point-to-point, but sent with a multicast address for simplicity reasons.

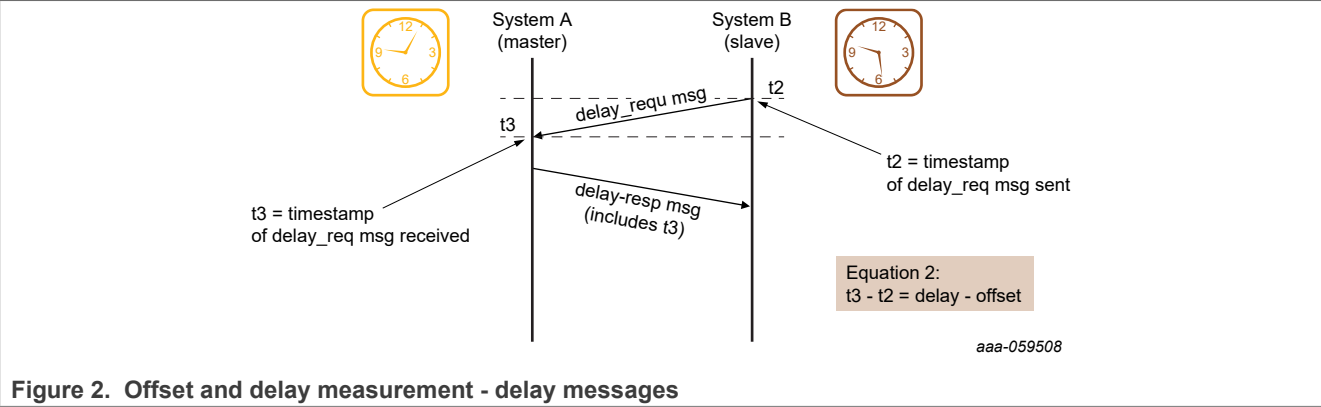
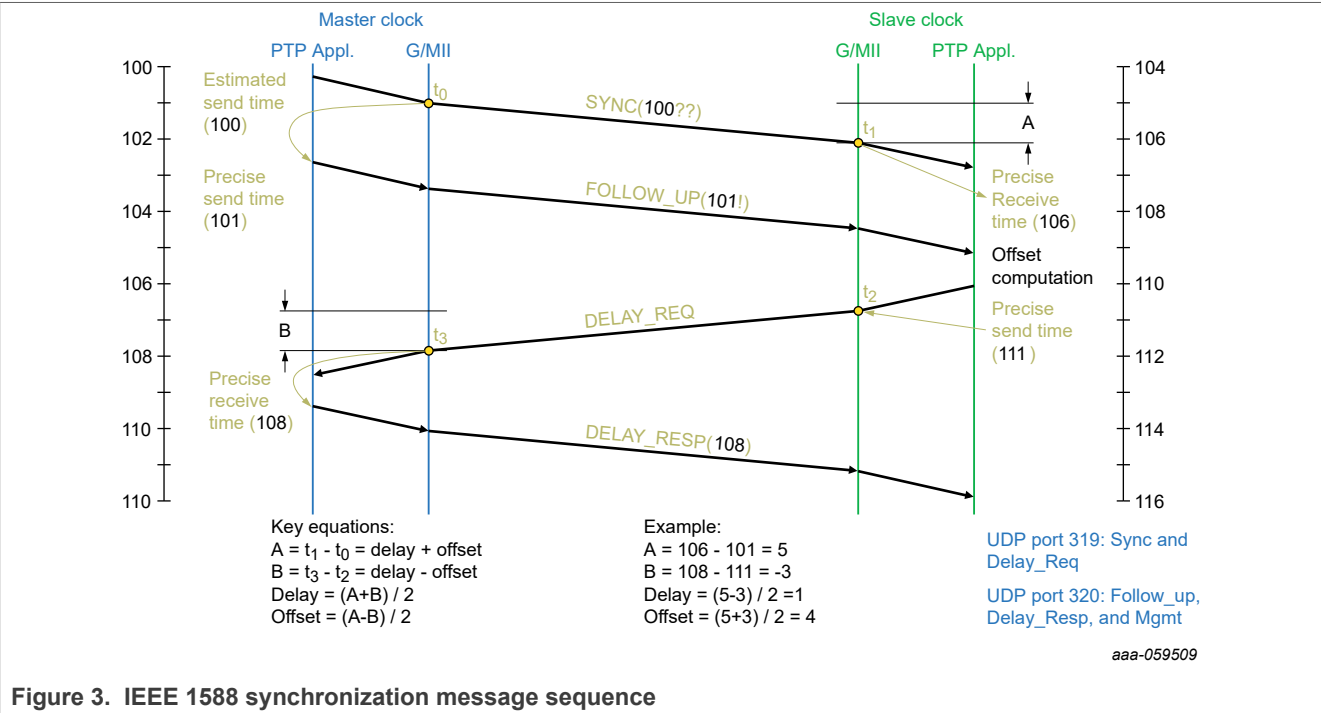


Figure 3 shows an example of the IEEE 1588 synchronization sequence (one cycle) and the calculation of the actual offset and delay between the master and slave nodes.



For more information about the IEEE 1588 standard, visit the webpage of the National Institute of Standards and Technology (www.nist.gov).

2.2 Timestamping

The PTP protocol can be implemented into the software completely, using a standard Ethernet module. The timestamp information is applied at the application level. As a result, delay fluctuations introduced by the software stack on both the master and slave devices limit the achievable synchronization precision (see [Figure 4](#)).

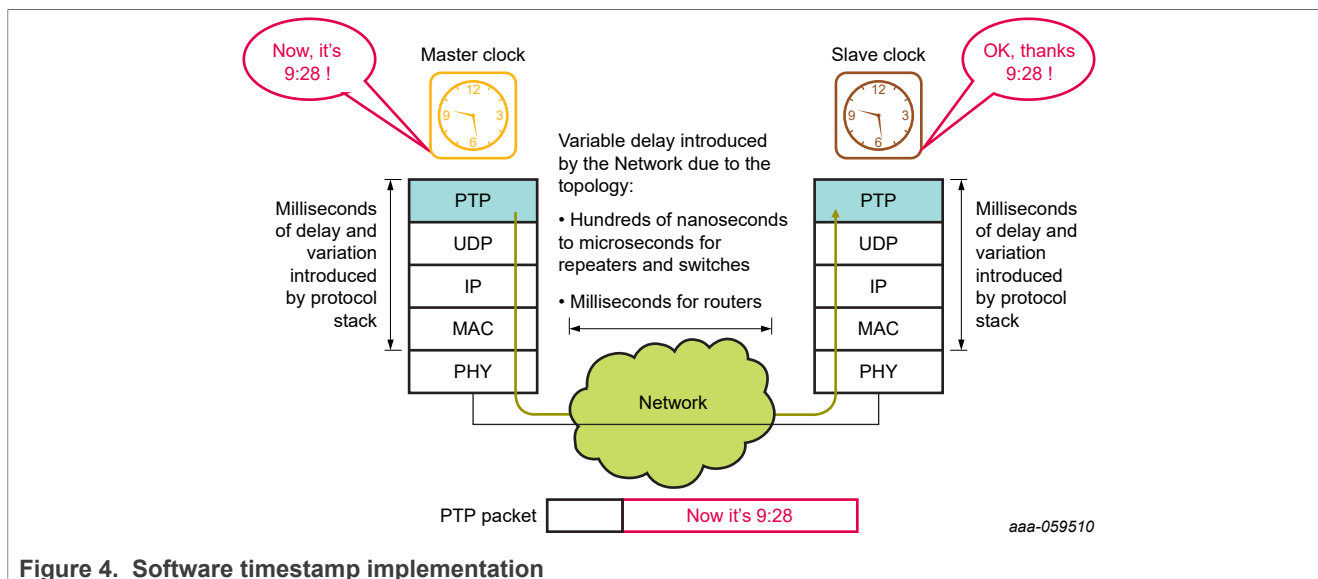


Figure 4. Software timestamp implementation

Minimize the impact of the protocol stack delay by taking timestamps closer to the physical interface, that is, at the MAC or PHY layers (see Figure 5). A dedicated hardware with timestamping capabilities (such as the Ethernet MAC (ENET) of the NXP MCX A577) enables synchronization with improved accuracy significantly.

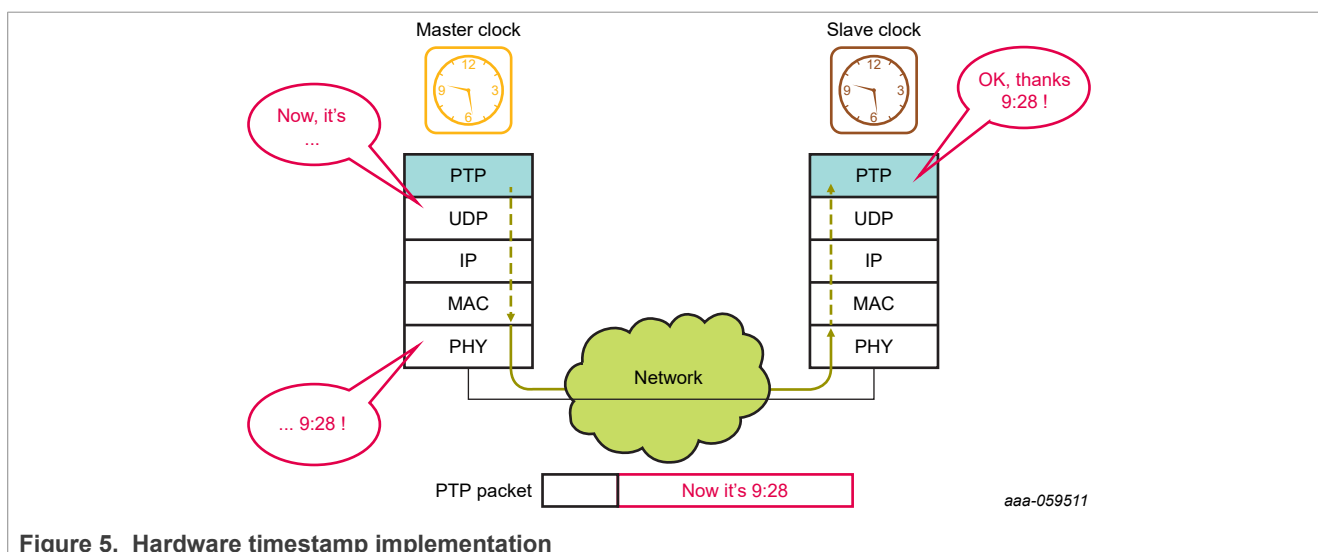


Figure 5. Hardware timestamp implementation

3 IEEE 1588 functions on MCX A577

MCX A577 device integrates a 10/100 Mbit/s Ethernet MAC (ENET) to accelerate the processing of various common networking protocols, such as IP, TCP, UDP, and ICMP. It provides wire-speed services to client applications. The ENET module uses a descriptor-based DMA that transfers frame data between the MAC and system memory. It supports the transmit and receive-descriptor timestamp fields, required for IEEE 1588 functionality.

To enable IEEE 1588 (or similar) time synchronization protocol implementations, MAC is combined with a timestamping module to support precise timestamping of incoming and outgoing frames. The timestamping logic is enabled through the MAC timestamp control register (MAC_TIMESTAMP_CONTROL[TSENA]).

3.1 System time register module

The 64-bit time is maintained in the ENET module and updated using the input reference clock.

This time is the source for taking snapshots (timestamps) of Ethernet frames being transmitted or received at the MII. The system time counter can be initialized or corrected using the coarse correction method. In this method, the initial value or the offset value is written to the timestamp update register. During initialization, the system time counter is loaded with the value in the timestamp update registers. During system time correction, an offset value is added to or subtracted from the system time.

In the fine correction method, a subordinate clock's frequency drift relative to the manager clock (as defined in IEEE 1588) is corrected gradually. This approach differs from coarse correction, in which the adjustment is made in a single clock correction step.

This approach helps maintain linear time and does not introduce drastic changes (or a large jitter) in the reference time between PTP sync message intervals. In this method, an accumulator sums up the contents of the addend register, as shown in [Figure 6](#). The arithmetic carries that the accumulator generator is used as a pulse to increase the system time counter. The accumulator and the addend are 32-bit registers. Here, the accumulator acts as a high-precision frequency multiplier or divider.

This algorithm is shown in [Figure 6](#).

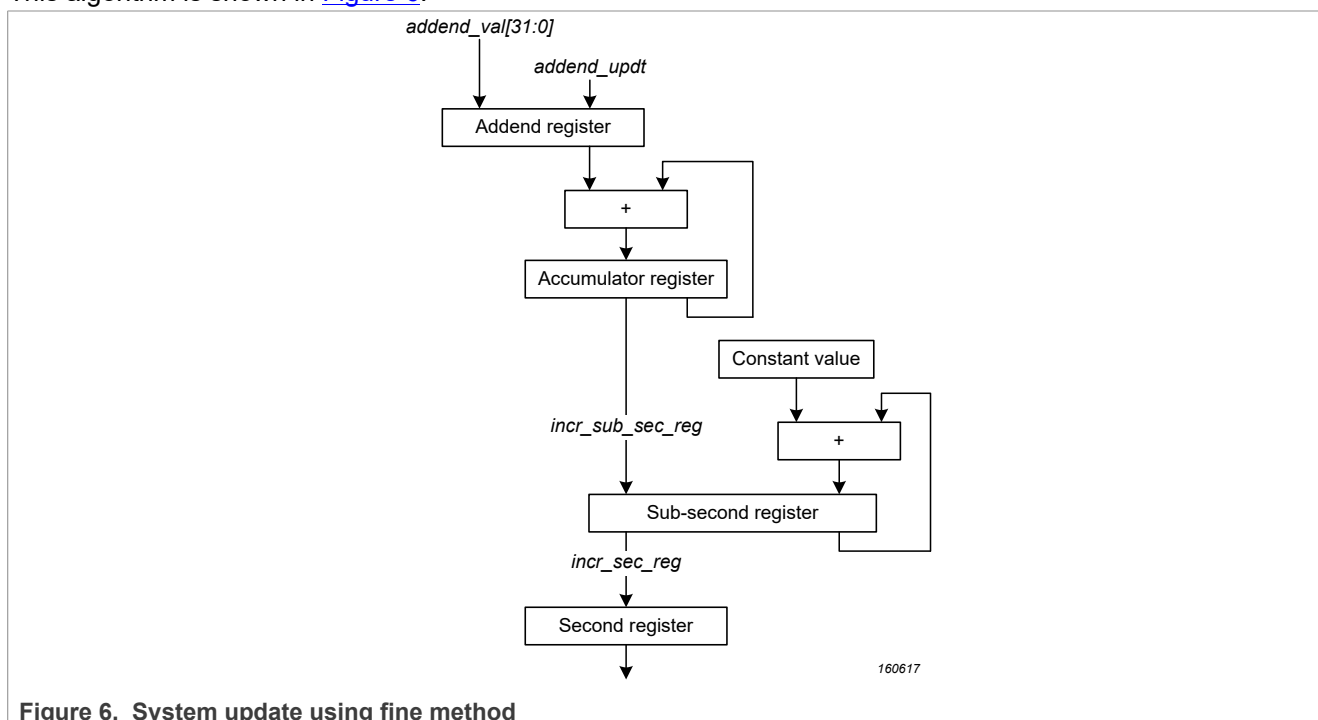


Figure 6. System update using fine method

The following example illustrates the system time update mechanism. To achieve a 20 ns accuracy, the sub-second value is incremented by 20 ns each time the accumulator overflows. Therefore, when TSCTRLSSR = 1, the SSINC bit field in the MAC_Sub_Second_Increment register must be set to 20. To maintain an accurate 1-second system time, the accumulator must overflow once every 20 ns. Therefore, the required accumulator overflow frequency is 50 MHz.

The frequency division is the ratio of the reference clock frequency (ptp_clk_i) to the required clock frequency (accumulator overflow frequency). Therefore, if the reference clock is, for example, 66 MHz, this ratio is calculated as $66 \text{ MHz} / 50 \text{ MHz} = 1.32$. Therefore, the default addend value to be programmed in the register is $2^{32} / 1.32$, 0xC1F07C1F.

If the reference clock drifts lower, for example to 65 MHz, the ratio is 65/50, or 1.3. The value to program in the addend register is $2^{32}/1.30$, or 0xC4EC4EC4. If the clock drifts higher, for example to 67 MHz, the addend register must be programmed to 0xBF0B7672. When the clock drift is nil, the default addend value of 0xC1F07C1F ($2^{32}/1.32$) must be programmed.

The constant value used to accumulate the sub-second register is decimal 43, which achieves an accuracy of 20 ns in the system time. In other words, it is incremented in 20 ns steps.

The software must calculate the drift in frequency based on the sync messages and update the addend register accordingly.

Initially, the subordinate clock is programmed with FreqCompensationValue0 in the addend register. This value is:

$$\text{FreqCompensationValue0} = 2^{32}/\text{FreqDivisionRatio}$$

Where:

- The frequency division (FreqDivisionRatio) is the ratio of the reference clock frequency to the required clock frequency.

For more detailed information, refer to the "Ethernet" chapter in the Reference Manual¹.

3.2 Transmit timestamping

MAC captures a timestamp when the start frame delimiter (SFD) of a frame is sent on MII. The frames for which you want to capture timestamps are controllable on a per-frame basis. In other words, each transmit frame can be marked to indicate whether a timestamp must be captured for that frame.

MAC does not process the transmitted frames to identify the PTP frames. Using the control bit in the transmit descriptor to specify the frames for which you want to capture timestamps.

MAC returns the timestamp to the software inside the corresponding transmit descriptor, therefore, connecting the timestamp automatically to the specific PTP frame. The 64-bit timestamp information is written to the TDES6 and TDES7 fields. The TDES7 field holds the 32 least significant bits of the timestamp.

3.3 Receive timestamping

MAC captures the timestamp of all frames received on the MII. MAC does not process the received frames to identify the PTP frames in the default mode, that is, when the advanced timestamp feature is not selected.

MAC can be programmed to capture the timestamp of all packets received on the MII interface or to process packets to identify the valid PTP messages. You can control the snapshot of the time to be sent to the application using the following options of the MAC_Timestamp_Control register.

- Enable snapshot for all packets.
- Enable snapshot for IEEE 1588 version 1 or version 2 timestamp.
- Enable snapshot for PTP packets transmitted directly over Ethernet or UDP-IP-Ethernet.
- Enable a timestamp snapshot for the received packet for IPv4 or IPv6.
- Enable timestamp snapshot only for EVENT messages (SYNC, DELAY_REQ, PDELAY_REQ, or PDELAY_RESP).
- Enable the node to be a manager or subordinate and select the snapshot type. This feature controls the type of messages for which snapshots are taken.

The DMA returns the timestamp to the software inside the corresponding receive descriptor. The extended status, containing the timestamp message status and the IPC status, is written in the normal descriptor RDES1.

¹ Contact your local field applications engineer (FAE) or sales representative.

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

The snapshot of the timestamp is written in the RDES0 and RDES1 fields of the context descriptor. The RDES0 field holds the 32 least significant bits of the timestamp.

3.4 PPS output

The ENET MAC also supports a Pulse Per Second (PPS) output. In this solution, the PPS output is enabled to measure the time synchronization error. The PPS configuration steps are as follows:

1. Configure P4_5 for the TRIG_OUT3 function and set the input source of the TRIG_OUT3 register in the INPUTMUX module to ENET PPS output. This configuration routes the ENET PPS signal to the P4_5 pin. To route the ENET PPS signal to other TRIG_OUT outputs, use the same approach.
2. Configure the PPSCtrl field in the MAC_PPS_Control register to 1, which sets the PPS to a low period of 537 ms and a high period of 463 ms. Alternatively, configure PPSCtrl with other values to adjust the PPS output frequency as required.

4 IEEE 1588 implementation for MCX A577

The FRDM board is used as the hardware platform for hardware timestamping-based IEEE 1588 V2 PTP. The solution uses the MCUXpresso SDK for the FRDM-MCXA577 board. This solution includes the NXP ENET driver of the MCX A577 MCU, PHY driver, ported FreeRTOS OS, and ported lwIP TCP/IP stack for the FRDM-MCXA577 board. The PTP daemon application, an open source implementation of the PTP, implements the IEEE1588 V2 PTP. [Figure 7](#) and [Figure 8](#) show the hardware and software components of this solution with external PHY and internal 10BASE-T1S PHY.

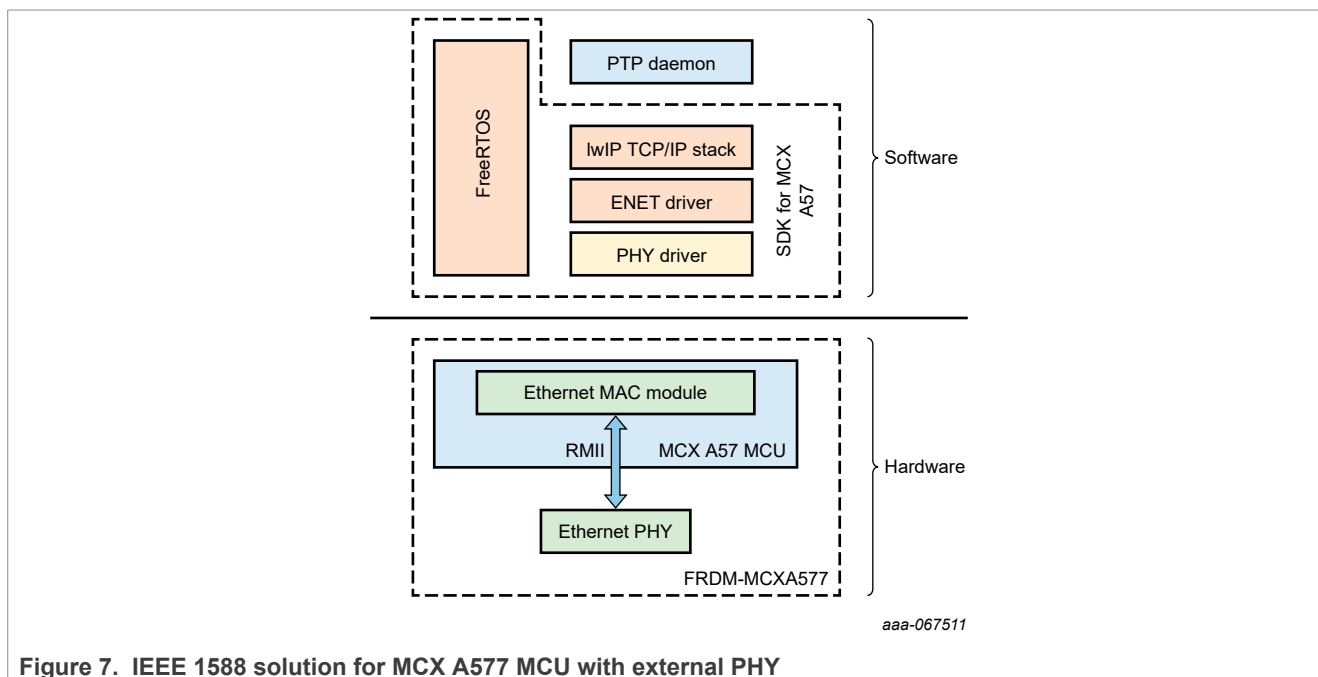


Figure 7. IEEE 1588 solution for MCX A577 MCU with external PHY

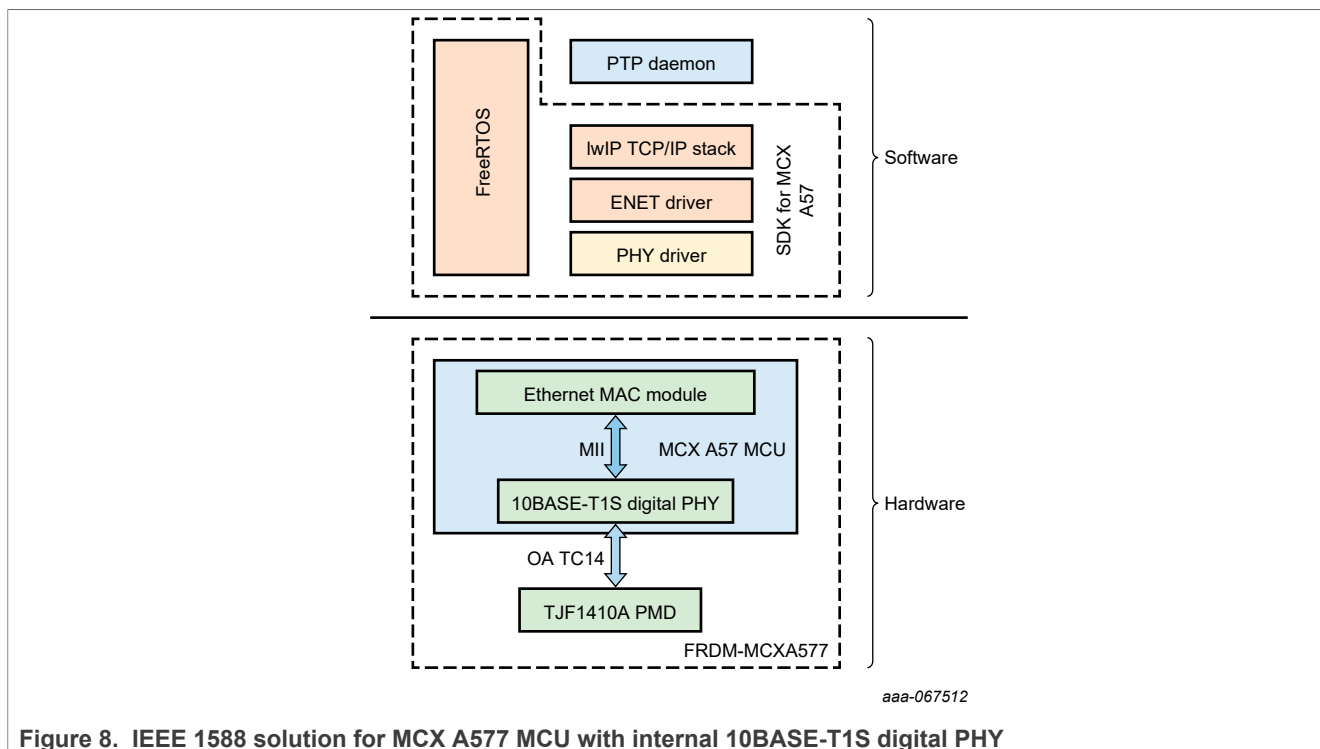


Figure 8. IEEE 1588 solution for MCX A577 MCU with internal 10BASE-T1S digital PHY

4.1 Hardware components

The FRDM-MCXA577 board is the platform designed to showcase the most common features of the MCX A577 processor. The FRDM-MCXA577 board is a development board, which helps you to become quickly familiar with the processor and expedites you to implement your own designs. The main features of the FRDM-MCXA577 board include:

- MCX A577 MCU
- On-board MCU-Link debugger
- An external QSPI flash
- RJ45 10/100-Mbit/s Ethernet using the Microchip LAN8741 PHY
- An on-chip 10BASE-T1S digital PHY (T1S)
- A 3-pin PMD transceiver TJF1410A compliant with Open Alliance TC14
- HS USB Type-C connector
- CAN bus connectors

Ethernet PHY options on the FRDM-MCXA577

FRDM-MCXA577 supports two physical layer (PHY) connections for the ENET MAC, and the demo can be built for either one:

1. External LAN8741 PHY. The Microchip LAN8741 is a 10/100-Mbit/s Ethernet transceiver connected to the ENET MAC through the MII/RMII interface and managed over MDIO. It drives the on-board RJ45 connector for a standard twisted pair Ethernet link. In the demo, `BOARD_NETWORK_USE_TENBASE_PHY == 0` selects this path and uses the LAN8741 PHY.
2. On-chip 10BASE-T1S digital PHY (T1S). The MCX A577 integrates a 10BASE-T1S digital PHY. It connects to the ENET MAC in MII mode and is managed through the same MDIO interface. In the demo, `BOARD_NETWORK_USE_TENBASE_PHY == 1` selects this path and uses the T1S PHY driver and the T1S0 controller.

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

Both PHY options are abstracted behind the common `phy_handle_t/phyOps` interface. Therefore, the PTPd, lwIP, and ENET 1588 timestamping code is identical for both; only the PHY driver, the MII mode, and the PHY resource selected in `app.h` differ.

4.2 Software components

The IEEE 1588 software implementation includes the MCUXpresso SDK for the FRDM-MCXA577 board and PTP daemon. The MCUXpresso SDK is a software framework for developing applications on NXP MCUs including peripheral drivers, middleware, and real-time operating system.

4.2.1 FreeRTOS OS

FreeRTOS OS is a real-time kernel (or real-time scheduler) on top of which you can build embedded applications that meet strict real-time requirements. FreeRTOS OS provides methods for multiple tasks, mutexes, semaphores, and software timers. A tickless mode is provided for low-power applications. The thread priorities used by the scheduler decide which thread must be executing. The version of the FreeRTOS OS provided by the MCUXpresso SDK for the FRDM-MCXA577 board is 10.0.1. The FreeRTOS OS package integrated into the MCUXpresso IDE has the following features:

- Removed the files not related to the SDK IDE, such as extensions to the FreeRTOS OS (CLI, FAT_SL, and UDP) and folders, such as the demo and nested folders.
- Added the `SystemCoreClock` global variable to the FreeRTOS OS `port.c` and `FreeRTOSConfig.h` files.
- Enabled the tickless mode. For more information, see <https://www.nxp.com/webapp/connect/displayPartnerProfile.sp?partnerId=1-8RW1-16&tid=vanFREERTOS>.
- Enabled the KDS Task Aware Debugger, applied the FreeRTOS patch to enable the `configRECORD_STACK_HIGH_ADDRESS` macro.
- Enabled the `-fto` optimization in GCC by adding `__attribute__((used))` for `vTaskSwitchContext`.

For detailed information about the FreeRTOS OS and its distribution, see <https://www.freertos.org/>.

4.2.2 lwIP TCP/IP stack

lwIP is a light-weight implementation of the TCP/IP protocol suite that is freely available in the C source code. It can be downloaded from the development webpage. It is modular and small enough to reduce the RAM usage for use in small embedded systems. The core stack provides an IP implementation. Depending on your requirements and the memory available in the designed system, you can add TCP, UDP, DHCP, and other protocols. For more information about lwIP, see https://www.nongnu.org/lwip/2_1_x/index.html.

MCUXpresso SDK for FRDM-MCXA577 integrates the lwIP TCP/IP stack, which runs on top of the MCUXpresso SDK Ethernet driver with the FRDM-MCXA577 board. The lwIP package version in the SDK IDE for the FRDM-MCXA577 board is 2.4.x. For more information, see the *lwIP TCP/IP Stack and MCUXpresso SDK Integration User's Guide* (document [MCUXSDKLWIPUG](#)).

4.2.3 PTP daemon

PTP provides precise time coordination of Ethernet LAN-connected computers, which is designed primarily for instrumentation and control systems. PTP daemon (PTPd) is an open source implementation of PTP version 2, as defined by IEEE STD 1588-2008.

PTPd coordinates the clocks of a group of LAN-connected computers with each other. It can achieve microsecond-level coordination even on the limited platform. PTPd is available in the C source code and easy to port on the FreeRTOS OS for embedded systems. Most of the system-related code is in the `<install_dir>/src/dep` folder. The PTPd package version used in this demo is version 2.3.1. It is available at github.com/ptpd/ptpd/releases.

5 IEEE 1588 demo software detailed description

This demo application is built and tested with the MCUXpresso IDE/GCC Arm toolchain for the FRDM-MCXA577 board. The PTPd version is 2.3.1 and the demo is built with the MCUXpresso SDK for MCX A577. The MCX A577 ENET supports IEEE 1588 with a hardware timestamp. To enable the hardware timestamp feature and run the demo on the FRDM-MCXA577 board, the original lwIP TCP/IP port-related code must be updated. The update for the ENET driver is needed to test the clock offset converging range and bug fixes related to the IEEE 1588 functions.

5.1 MCX A577 SDK ENET driver update

An `enet_ptp_time_data_t` structure must be added to the `fsl_enet.h` file.

```

/*! @brief Defines the structure for the ENET PTP message data and timestamp data.*/
typedef struct _enet_ptp_time_data
{
    uint8_t version; /*!< PTP version. */
    uint8_t sourcePortId[kENET_PtpSrcPortIdLen]; /*!< PTP source port ID. */
    uint16_t sequenceId; /*!< PTP sequence ID. */
    uint8_t messageType; /*!< PTP message type. */
    enet_ptp_time_t timeStamp; /*!< PTP timestamp. */
} enet_ptp_time_data_t;

```

To enable the fine method feature in the IEEE 1588 timer, the bold code must be added or modified in the `ENET_SetPtp1588()` function.

```

if (ptpConfig->fineUpdateEnable)
{
    base->MAC_TIMESTAMP_CONTROL |= ENET_MAC_TIMESTAMP_CONTROL_TSCFUPDT_MASK;
    /* Set the initial added value for the fine update. */
    control = 0x80000000;
    base->MAC_TIMESTAMP_ADDEND = control;
    base->MAC_TIMESTAMP_CONTROL |= ENET_MAC_TIMESTAMP_CONTROL_TSADDREG_MASK;
    while ((base->MAC_TIMESTAMP_CONTROL & ENET_MAC_TIMESTAMP_CONTROL_TSADDREG_MASK) !=
0U)
    {
    }
}

/* Enable the IEEE 1588 timestamping and snapshot for event message. */
control = ENET_MAC_TIMESTAMP_CONTROL_TSENA_MASK |
    ENET_MAC_TIMESTAMP_CONTROL_TSIPV4ENA_MASK |
    ENET_MAC_TIMESTAMP_CONTROL_TSIPV6ENA_MASK |
    ENET_MAC_TIMESTAMP_CONTROL_TSENALL_MASK |
    ENET_MAC_TIMESTAMP_CONTROL_TSEVENTENA_MASK |
    ENET_MAC_TIMESTAMP_CONTROL_SNAPTYPSEL_MASK |
    ENET_MAC_TIMESTAMP_CONTROL_TSCTRLSSR(ptpConfig->tsRollover) |
    ENET_MAC_TIMESTAMP_CONTROL_TSCFUPDT(ptpConfig->fineUpdateEnable);

if (ptpConfig->ptp1588V2Enable)
{
    control |= ENET_MAC_TIMESTAMP_CONTROL_TSVER2ENA_MASK |
    ENET_MAC_TIMESTAMP_CONTROL_TSIPENA_MASK;
}

/* Initialize the sub-second increment register. */
if (ptpConfig->tsRollover != kENET_BinaryRollover)
{
    #ifdef ENET_MAC_SUB_SECOND_INCREMENT_SNSINC_MASK
        base->MAC_SUB_SECOND_INCREMENT =
    ENET_MAC_SUB_SECOND_INCREMENT_SNSINC(ENET_NANOSECS_ONESECOND / ptpClkHHz);
    #else

```

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

```

base->MAC_SUB_SECOND_INCREMENT =
ENET_MAC_SUB_SECOND_INCREMENT_SSINC(ENET_NANOSECS_ONESECOND * 2 / ptpClkHz);
#endif
base->MAC_SYSTEM_TIME_NANOSECONDS_UPDATE = 0;
}

```

5.2 lwIP TCP/IP porting update

This section describes the modifications of the lwIP porting code to support the PTP demo. It involves the `lwipopts.h`, `ethernetif.h`, and `ethernetif.c` files in the `<sdk_install_dir>/middleware/lwip/port` folder.

The PTP daemon demo uses the `SO_REUSEADDR` option for the socket, Domain Name System (DNS) protocol, and Internet Group Management Protocol (IGMP). It does not use the Dynamic Host Configuration Protocol (DHCP) with a static IP address.

The following macros in `lwipopts.h` must be defined to the corresponding values:

```

/* SO_REUSE ==1: Enable SO_REUSEADDR option */
#define SO_REUSE 1

#ifndef LWIP_DHCP
#define LWIP_DHCP 0
#endif

/* ----- DNS options ----- */
#ifndef LWIP_DNS
#define LWIP_DNS 1
#endif

/* ----- IGMP options ----- */
/* LWIP_IGMP==1: Turn on IGMP module. */
#ifndef LWIP_IGMP
#define LWIP_IGMP 1
#endif

```

The default lwIP package in the SDK IDE release does not support PTP. The ENET initializing function for lwIP does not involve any 1588 timer functions. The code update for `ethernetif.h` and `ethernetif.c` mainly covers the ENET 1588 timer routines, such as initializing the 1588 timer, enabling the timer channel output compare function for the test, setting/getting time, adjusting the 1588 timer frequency, and getting the timestamp of the transmit/receive frames. The `#if LWIP_PTP` and `#endif` pair encloses any code related to PTP.

This code snippet must be added to `ethernetif.h` to declare the routines of the 1588 timer:

```

#if LWIP_PTP
#include "lwip_ptp.h"

#define ENET_NANOSECOND_ONE_SECOND 1000000000U
#define PTP_AT_INC (ENET_NANOSECOND_ONE_SECOND/PTP_CLOCK_FRE_RT)

void ethernet_ptptime_settime(enet_ptp_time_t *timestamp);
void ethernet_ptptime_gettime(enet_ptp_time_t *timestamp);
void ethernet_ptptime_adjfreq(int32_t ppb);
err_t enet_get_rxframe_time(enet_ptp_time_data_t *ptpTimeData);
err_t enet_get_txframe_time(enet_ptp_time_data_t *ptpTimeData);
#endif

```

The above functions are implemented in the `ethernetif.c` file. The ENET 1588 timer-initializing function `ethernet_ptptime_init()` is implemented and called before returning from the `enet_init()` function. The `ethernet_ptptime_enablepps()` function is implemented to enable the PPS output function for test purposes. It is called in the `ethernet_ptptime_init()` function according to the passed parameter.

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

Following is the code of the *ethernet_ptptime_enablepps()* function:

```
static void ethernet_ptptime_enablepps(struct ethernetif *ethernetif)
{
    /* P4_5, EXT TRIGOUT3, ENET_PPS_EVENT_0 */
    CLOCK_EnableClock(kCLOCK_GateINPUTMUX0);
    /* ENET PPS Output */
    INPUTMUX0->TRIG_OUT[3] = 0x16;

    /* Set PPSCTRL to 1 to enable the PPS output. */
    ethernetif->base->MAC_PPS_CONTROL =
        (ethernetif->base->MAC_PPS_CONTROL & ~ENET_MAC_PPS_CONTROL_PPSCTRL_PPSCMD_MASK) |
        ENET_MAC_PPS_CONTROL_PPSCTRL_PPSCMD(1U);
}
```

This code is the ENET 1588 timer initializing function *ethernet_ptptime_init()* and its related memories:

```
static void
ethernet_ptptime_init(struct ethernetif *ethernetif, uint32_t ptp_clk_freq,
                      bool pps_en)
{
    enet_ptp_config_t ptp_cfg;

    assert(ethernetif);
    ptp_ethernetif = ethernetif;

    if (true == pps_en)
    {
        ethernet_ptptime_enablepps(ptp_ethernetif);
    }
    else
    {
    }
}
```

The syntax in bold is used to call the *ethernet_ptptime_init()* function in the *enet_init()* function:

```
static void enet_init(struct netif *netif, struct ethernetif *ethernetif,
                    const ethernetif_config_t *ethernetifConfig)
{
    #if LWIP_PTP
    /* It's time to initialize the IE1588 function of ethernet */
    ethernet_ptptime_init(ethernetif, PTP_CLOCK_FRE_RT, PTP_TEST_APP_ENABLE);
    g_defaultAddend = ethernetif->base->MAC_TIMESTAMP_ADDEND;
    #endif
    ENET_ActiveRead(ethernetif->base);
}
```

The *ethernet_ptptime_adjfreq()* function adjusts the frequency of the 1588 timer by setting the non-zero correction counter wrap-around value in the MAC_TIMESTAMP_ADDEND register. This value defines the number of timer clock cycles to correct the time of the 1588 timer. The correction increment value is set in the INC_CORR field in the MAC_SUB_SECOND_INCREMENT register. The value of INC_CORR is larger than the value in the INC field to speed up the 1588 timer. The value of INC_CORR is lower than the value in the INC field to slow down the 1588 timer.

Following is the code of the *ethernet_ptptime_adjfreq()* function:

```
void ethernet_ptptime_adjfreq(int32_t incps)
{
}
```

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

```

int32_t neg_adj = 0;
uint32_t corr_inc, corr_period;
uint32_t addend;
uint64_t temp = 0;

assert(ptp_ethernetif);

/*
 * incps means the increment rate (nanoseconds per second) by which to
 * slow down or speed up the slave timer.
 * Positive ppb need to speed up and negative value need to slow down.
 */
if (0 == incps)
{
    /* No correction: restore nominal (0 ppb) addend. */
    addend = g_defaultAddend;
}
else
{
    if (incps < 0)
    {
        incps = -incps;
        neg_adj = 1;
    }

    /* Calculate new ABSOLUTE addend value from the fixed nominal base
     *
     * For speed up (neg_adj = 0):
     *   addend_new = g_defaultAddend * (1 + incps/10^9)
     *
     * For slow down (neg_adj = 1):
     *   addend_new = g_defaultAddend * (1 - incps/10^9)
     *
     * Using 64-bit arithmetic to avoid overflow:
     *   addend_new = (g_defaultAddend * (10^9 +/- incps)) / 10^9
     */
    if (neg_adj)
    {
        /* Slow down: addend_new = base * (1 - incps/10^9) */
        temp = (uint64_t)g_defaultAddend * (1000000000ULL - (uint64_t)incps);
    }
    else
    {
        /* Speed up: addend_new = base * (1 + incps/10^9) */
        temp = (uint64_t)g_defaultAddend * (1000000000ULL + (uint64_t)incps);
        if (temp >= 4294967295000000000ULL)
        {
            temp = 4294967295000000000ULL;
        }
    }

    addend = (uint32_t)(temp / 1000000000ULL);
}

/*
 * NOTE: do NOT write the result back into g_defaultAddend. It must stay the
 * constant nominal base so the next correction is again absolute.
 */

ENET_Ptp1588CorrectTimerInFine(ptp_ethernetif->base, addend);
}

```

The *ethernet_ptptime_settime()*, *ethernet_ptptime_gettime()*, *enet_get_rxframe_time()*, and *enet_get_txframe_time()* functions are implemented to wrap the *ENET_Ptp1588GetTimer()*, *ENET_Ptp1588SetTimer()*, *ENET_GetRxFrameTime()*, and *ENET_GetTxFrameTime()* functions in the *ethernetif.c* file.

5.3 PTPd porting on FreeRTOS OS

The default PTPd source code is for the FreeBSD, NetBSD, Mac OS X, and Linux operation systems. To port the code to FreeRTOS OS with the lwIP and SDK drivers for the FRDM-MCXA577 board, add or port the OS-related code, network-related code, and hardware timestamping code. The ported work covers the PTPd tasks under the FreeRTOS OS, system time routines, system services, software timer, interaction with network socket, and minor modification of the PTP protocol. The simple code modifications required by the toolchain during compiling and linking are not discussed in this document.

Even the code in the files in the `ptpd/src` folder is common to the PTPd application. Some files must be updated for the PTPd to work on the FreeRTOS OS. The original `main()` function in the `ptpd.c` file is changed to `ptpd_thread()`, which is created as a FreeRTOS OS task. The original command-line parameters are removed to simplify the demo functions, except for the variable to denote the master or the slave. The parameter passes this variable's value while the FreeRTOS OS task is being created.

Another significant modification in the PTPd common code is in the `protocol.c` file. The default PTPd application runs as a real network node within a UNIX-style system. The device can receive the frames of event messages sent by itself, because they are sent using UDP/IP multicast messages. The `Follow_Up` or `Pdelay_Resp_Follow_Up` messages are sent after the device receives the corresponding `Sync` or `Pdelay_Resp` event messages sent by itself in the original protocol source code. Because this demo runs with a point-to-point connection, the device does not receive the event message sent by itself. The code must be modified to send these two follow-up messages immediately after the corresponding event message is sent. As a result of this change, the `netSelect()` function must be called with a specific timeout value to replace the original `NULL` (no timeout) that blocks the `select()` function waiting for a file descriptor indefinitely.

The files in the `ptpd/src/dep` folder are port-specific source code files and depend on the operating system, TCP/IP stack, and hardware platform. The main changes involve the `net.c`, `startup.c`, `sys.c`, and `timer.c` files. Their names are suffixed with `_mcu` to distinguish them from the original files.

FreeRTOS OS provides software timer functionality when setting `configUSE_TIMERS` to 1 in the `FreeRTOSConfig.h` file. The modification of the `timer_mcu.c` file includes these two functions:

```
void catch_alarm(TimerHandle_t xTimer)
{
    (void)xTimer;

    elapsed++; /* be sure to NOT call DBG in asynchronous handlers! */
}

void initTimer(void)
{
    TimerHandle_t xptpTimer;

    xptpTimer = xTimerCreate("ptp_timer", pdMS_TO_TICKS(MS_TIMER_INTERVAL), pdTRUE, NULL,
        catch_alarm);

    if(xptpTimer != NULL)
    {
        xTimerStart(xptpTimer, portMAX_DELAY);
    }
}
```

The `sys_mcu.c` file has some time-related routines to provide interfaces to the low-level hardware timer that is synchronized in the demo. Most of these routines call the functions described in [Section 5.2 "lwIP TCP/IP porting update"](#).

The `adjFreq()` function code is as follows:

```
Boolean adjFreq(Integer32 adj)
{
```

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

```

    if (adj > ADJ_FREQ_MAX)
        adj = ADJ_FREQ_MAX;
    else if (adj < -ADJ_FREQ_MAX)
        adj = -ADJ_FREQ_MAX;
    ethernet_ptptime_adjfreq(adj);
    return TRUE;
}

```

Two sleep functions are provided to place the current thread into a dormant state that is implemented using the FreeRTOS *vTaskDelay()* function. The remaining changes are limited to removing code associated with the informational output and/or the log file output.

```

Boolean nanoSleep(TimeInternal * t)
{
    TickType_t time;

    time = pdMS_TO_TICKS(t->seconds * 1000 + t->nanoseconds / 1000000);
    vTaskDelay(time);
    return TRUE;
}

void milliSleep(int milli_seconds)
{
    TickType_t time;

    time = pdMS_TO_TICKS(milli_seconds);
    vTaskDelay(time);
}

```

The `startup_mcu.c` file removes all OS-related signal functions and the command-line parameter-parsing code. The *ptpd_init()* function is added to create a FreeRTOS OS task for the PTPd application.

The timestamp of the transmit frame in the original code is provided by the OS. The timestamp of the received frame can be extracted from the received data after calling the *recvmsg()* function supported by a Linux-style OS. This demo uses the hardware timestamping feature in the ENET 1588 timer. These codes must be ported to the FreeRTOS OS and the ENET 1588 timer on the MCX A577 MCU. The code in the `net_mcu.c` file provides the ported functions.

The *findIface()* function directly returns the default network interface used in lwIP and implemented as follows:

```

#define netGetDefaultNetif() (netif_default)

UInteger32 findIface(Octet * ifaceName, UInteger8 * communicationTechnology,
                    Octet * uuid, NetPath * netPath)
{
    struct netif * iface;
    (void) communicationTechnology;
    (void) netPath;
    iface = netGetDefaultNetif();

    memcpy(uuid, iface->hwaddr, iface->hwaddr_len);
    memcpy(ifaceName, iface->name, sizeof (iface->name));
    return iface->ip_addr.addr;
}

```

Because the ported code does not use the *recvmsg()* function to enable timestamps. The *netInitTimestamping()* function is not invoked from *netInit()*, nor is it implemented as a dummy function that unconditionally returns TRUE.

The ENET driver on the MCX A577 MCU provides two APIs to query the timestamp of the event message's frame that is transmitted or received at the time. To get the timestamp, the ENET PTP message data and the

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

timestamp data defined by the `enet_ptp_time_data_t` type must be packed from the buffer that contains the received or sent event messages. The `netPackPtpData()` function is added for this task and listed explicitly in this code:

```
static void netPackPtpData(Octet * buf, enet_ptp_time_data_t *pptpTimeData)
{
    pptpTimeData->messageType = (*(Enumeration4 *) (buf + 0)) & 0x0F;
    pptpTimeData->sequenceId = flip16(*(UInteger16 *) (buf + 30));
    pptpTimeData->version = (*(UInteger4 *) (buf + 1)) & 0x0F;
    memcpy(pptpTimeData->sourcePortId, (buf + 20), 10);
}
```

The `recv()` socket function replaces the `recvmsg()` function in the `netRecvGeneral()` function to read the general message's frame. The following is the code of the `netRecvGeneral()` function implementation:

```
netRecvGeneral(Octet * buf, NetPath * netPath)
{
    ssize_t ret = 0;

    ret = recv(netPath->generalSock, buf, PACKET_SIZE, MSG_DONTWAIT);
    if (ret <= 0) {
        if (errno == EAGAIN || errno == EINTR)
            return 0;

        return ret;
    }

    return ret;
}
```

The `netRecvEvent()` function reads the event message's frame and returns its timestamp provided by the low-level ENET driver. Following is the code of its implementation:

```
ssize_t netRecvEvent(Octet * buf, TimeInternal * time, NetPath * netPath, int flags)
{
    ssize_t ret = 0;

    enet_ptp_time_data_t pptpTimeData;
    struct lwip_sock *sock;

    getTime(time); /* Read the current time used for timestamp in case of reading from
driver failed. */
    ret = recv(netPath->eventSock, buf, PACKET_SIZE, MSG_PEEK);
    if (ret <= 0) {
        if (errno == EAGAIN || errno == EINTR)
            return 0;

        return ret;
    }

    /* get time stamp of packet */
    if (!time) {
        ERROR("null receive time stamp argument\n");
        return 0;
    }

    netPackPtpData(buf, &pptpTimeData);

    sock = lwip_socket_dbg_get_socket(netPath->eventSock);

    time->nanoseconds = sock->lastdata.netbuf->p->t_nsec;
    time->seconds = (Integer32)sock->lastdata.netbuf->p->t_sec;
```

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

```

ret = recv(netPath->eventSock, buf, PACKET_SIZE, MSG_DONTWAIT);

if (ret <= 0) {
    if (errno == EAGAIN || errno == EINTR)
        return 0;

    return ret;
}

return ret;
}

```

Both the *netSentEvent()* and *netSentPeerEvent()* functions send the frame of a corresponding event message and return the frame's timestamp. The default implementation does not support hardware timestamping. To enable hardware timestamping, the *netSentEvent()* and *netSentPeerEvent()* functions add another input parameter of the pointer to the buffer that contains the returned timestamp.

This code snippet shows that the code added into the *netSentEvent()* and *netSentPeerEvent()* functions in bold:

```

ssize_t netSendEvent(Octet * buf, UInteger16 length, TimeInterval * time, NetPath *
netPath, Integer32 alt_dst)
{
    .....
    enet_ptp_time_data_t ptpTimeData;
    .....
    enet_ptp_time_t t_pre;
    t_pre.second = (uint64_t)time->seconds;
    t_pre.nanosecond = (uint32_t)time->nanoseconds;
    getTime(time); /* Current time for timestamp in case of reading from driver failed.
*/
    netPackPtpData(buf, &ptpTimeData);
    if(!enet_get_txframe_time(&ptpTimeData)){
        time->nanoseconds = ptpTimeData.timeStamp.nanosecond;
        time->seconds = (Integer32)ptpTimeData.timeStamp.second;
    } else {
        /* suspend process 1 millisecond to make sure current frame was sent by enet */
        millisleep(1);
        /* Try to read timestamp again */
        if(!enet_get_txframe_time(&ptpTimeData)){
            /* return the timestamp gotten from driver */
            time->nanoseconds = ptpTimeData.timeStamp.nanosecond;
            time->seconds = (Integer32)ptpTimeData.timeStamp.second;
        }
    }
    return ret;
}

```

5.4 FreeRTOS OS tasks and board configuration

There are three task threads created using the FreeRTOS OS in the demo application:

- **stack_init** task: Created in the main function. This task initializes the lwIP TCP/IP stack and the static IP address, netmask configuration, gateway address, and MAC address. It also performs Ethernet hardware initialization. Then, it starts the PTPd task. Lastly, the task deletes itself by calling the *vTaskDelete()* function after initializing the *lwIP* and *PTPd* tasks.
- **tcpip_thread** task: Created by the *stack_init* task during the TCP/IP initialization. This task runs the main *lwIP* task to access the *lwIP* core functions.
- **ptpd_thread** task: Created by the *stack_init* task. This task runs the PTPd application. The parameter passed while this task is being created denotes either the master or the slave.

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

The demo enables ENET PPS output to TRIG_OUT3 on P4_5 pin. The TRIG_OUT3 pin is configured in the `pin_mux.c` file.

```
void BOARD_InitTRIGOUTPins(void)
{
    /* PORT2: Peripheral clock is enabled */
    CLOCK_EnableClock(kCLOCK_GatePORT4);
    /* PORT2 peripheral is released from reset */
    RESET_ReleasePeripheralReset(kPORT4_RST_SHIFT_RSTn);

    const port_pin_config_t port4_5_config = { /* Internal pull-up resistor is enabled */
                                                .pullSelect = kPORT_PullDisable,
                                                /* Low internal pull resistor value is
selected. */
                                                .pullValueSelect = kPORT_LowPullResistor,
                                                /* Fast slew rate is configured */
                                                .slewRate = kPORT_FastSlewRate,
                                                /* Passive input filter is disabled */
                                                .passiveFilterEnable =
kPORT_PassiveFilterDisable,
                                                /* Open drain output is disabled */
                                                .openDrainEnable = kPORT_OpenDrainDisable,
                                                /* Low drive strength is configured */
                                                .driveStrength = kPORT_LowDriveStrength,
                                                /* Normal drive strength is configured */
                                                .driveStrength1 = kPORT_NormalDriveStrength,
                                                /* Pin is configured as TRIG_OUT3 */
                                                .mux = kPORT_MuxAlt1,
                                                /* Digital input enabled */
                                                .inputBuffer = kPORT_InputBufferEnable,
                                                /* Digital input is not inverted */
                                                .invertInput = kPORT_InputNormal,
                                                /* Pin Control Register fields [15:0] are
not locked */
                                                .lockRegister = kPORT_UnlockRegister};

    /* PORT4_5 is configured as PPS OUT */
    PORT_SetPinConfig(PORT4, 5U, &port4_5_config);
}
```

The other board-specific initialization code follows the ENET examples provided in the MCUXpresso SDK for the FRDM-MCXA577 board.

6 Running the IEEE 1588 demo

This section describes how to set up the 1588 demo using the FRDM-MCXA577 board and the demo software described in the above sections.

6.1 Hardware setup

Two FRDM-MCXA577 boards must be used and connected to each other to set up the hardware for the test. The demo has a point-to-point configuration where two boards are connected directly using the crossover Ethernet cable. This demo uses a simple type of connection often used to evaluate the system's accuracy and overall performance.

6.1.1 Board connection when using external LAN8741 PHY

The point-to-point configuration using two FRDM-MCXA577 boards is shown in [Figure 9](#). Before using the RMII interface to connect LAN8741 PHY, the board must be reworked as follows:

- TXD2: Solder R35

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

- TXD3: Solder R36
- RMII RXDV: Short R43, remove R164, remove R166
- RMIISEL: Solder R53
- TXCLK: Remove R37, short 2-3 of SJ43, solder R132
- Solder Y4 50 MHz OSC or configure the MCU's TX_CLK pin to provide a 50 MHz reference clock to the PHY

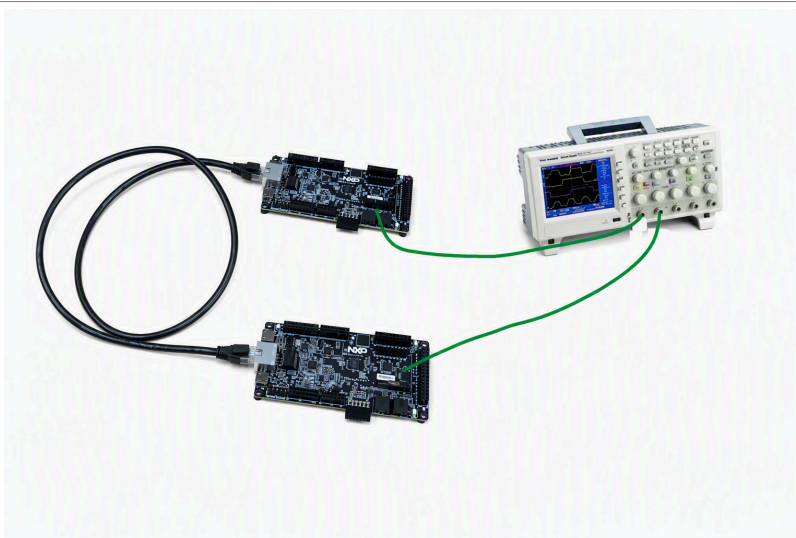


Figure 9. Board connection when using external LAN8741 PHY

For detailed information on how to use the FRDM-MCXA577 board, see the *FRDM-MCXA577 Board User Manual* (document UM12630)².

6.1.2 Board connection when using internal 10BASE-T1S PHY

To use internal 10BASE-T1S digital PHY, you must set BOARD_NETWORK_USE_TENBASET_PHY macro to 1.

² Contact your local field applications engineer (FAE) or sales representative.]

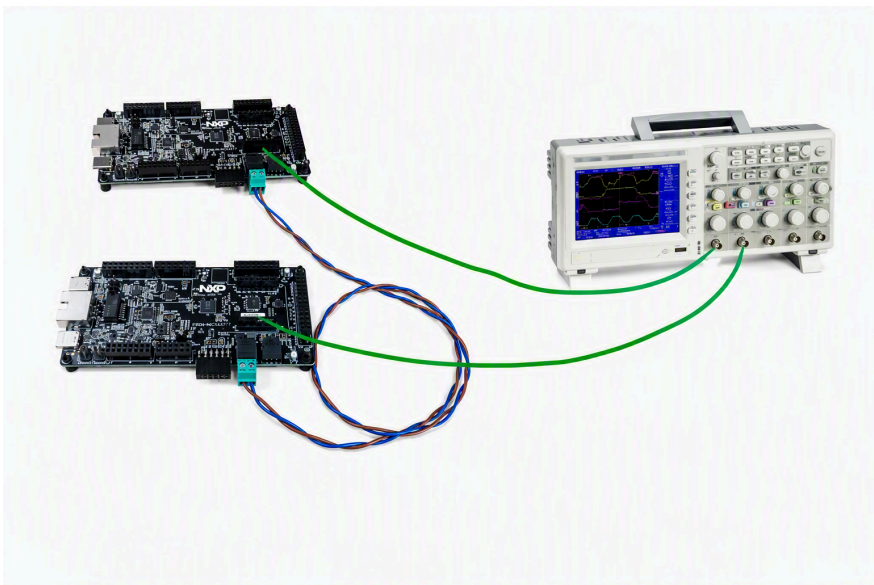


Figure 10. Board connection when using internal 10BASE-T1S PHY

6.2 Clock synchronicity measuring

This demo can generate a Pulse-Per-Second (PPS) signal to measure the synchronicity of the clocks between the master and the slave. As for the MCX A577 MCU, the PPS signal is routed to J6_5 (P4_5) pin.

To measure and compare the PPS signals from two boards, attach two oscilloscope probes to the ENET 1588 PPS output pins on the two FRDM-MCXA577 boards respectively. The two boards are powered for a time interval in the test. Oscilloscope measurements show that the slave PPS signal gradually converges toward the master PPS signal.

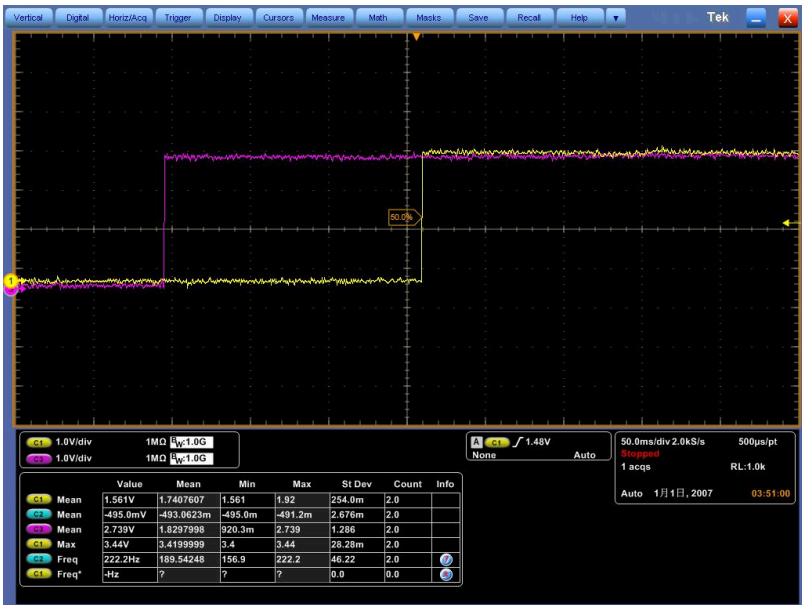


Figure 11. PTP startup - PPS distance between master and slave

6.2.1 Measurement results using an external Ethernet PHY

Figure 12 shows the clock synchronization results when using an external Ethernet PHY; the clock synchronization accuracy can be less than 50 ns.

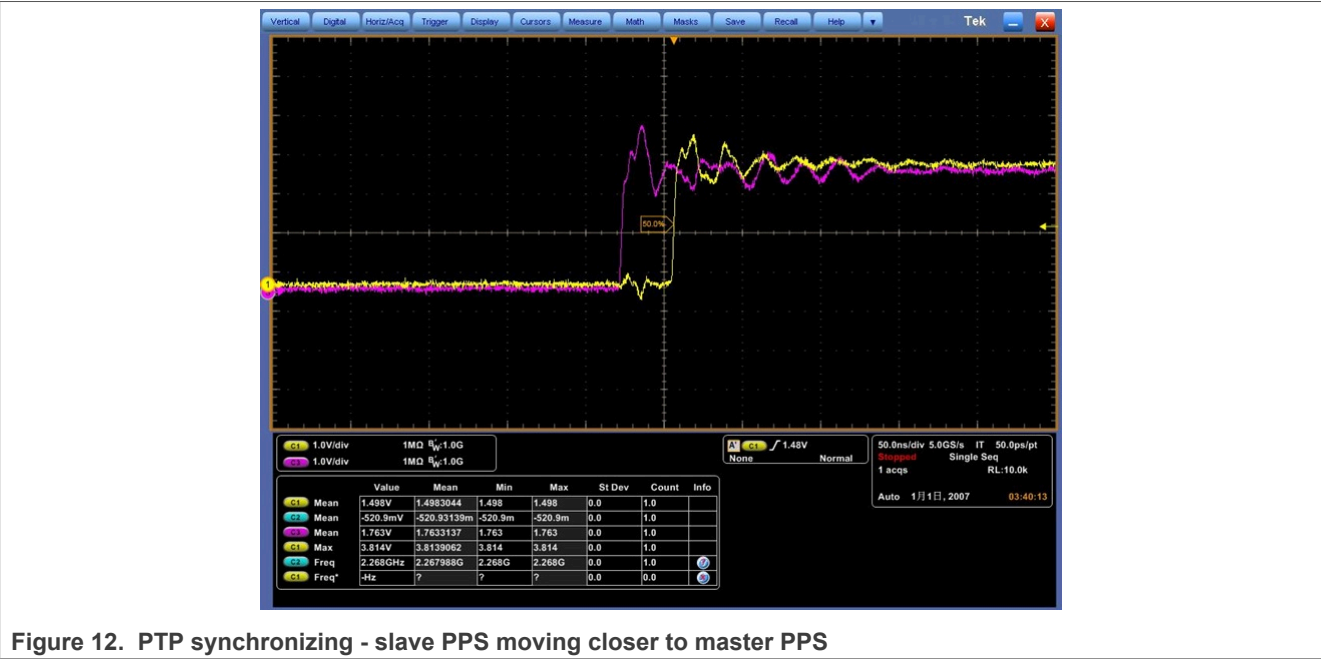


Figure 12. PTP synchronizing - slave PPS moving closer to master PPS

6.2.2 Measurement results using internal 10BASE-T1S PHY

Figure 13 shows the clock synchronization results when using internal 10BASE-T1S PHY; the clock synchronization accuracy can be less than 300 ns.



Figure 13. PTP synchronized - converged PPS distance between master and slave

7 Conclusion

This application note describes the IEEE 1588 Precision Time Protocol (PTP) demo application. The demo is based on the open source PTP daemon, the FreeRTOS operating system, and the lwIP TCP/IP stack. It also uses the SDK for the MCX A577 and the MCX A577 Freedom development board (FRDM-MCXA577). This demo can be easily ported to other processors from the MCX A57 series with the FreeRTOS OS, lwIP, and TCP/IP stack support.

The demo system is targeted for applications that require precise clock synchronization between devices with accuracy in the sub-microsecond range.

8 Acronyms

[Table 1](#) lists the acronyms used in this document.

Table 1. Acronyms

Term	Meaning
API	Application Programming Interface
BMC	Best Master Clock
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
ENET	10/100-Mbit/s Ethernet MAC
GPIO	General-Purpose Input/Output
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
MAC	Media Access Control
PPS	Pulse-Per-Second
PTP	Precision Time Protocol
PTPd	PTP Daemon
RTOS	Real-Time Operating System
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

9 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2026 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10 Revision history

[Table 2](#) summarizes the revisions to this document.

Table 2. Revision history

Document ID	Release date	Description
AN15137 v.1.0	12 October 2026	Initial release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <https://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Implementing IEEE 1588 V2 on the MCX A577 Using PTPd, FreeRTOS, and the lwIP TCP/IP Stack

Amazon Web Services, AWS, the Powered by AWS logo, and FreeRTOS
— are trademarks of Amazon.com, Inc. or its affiliates.

Contents

1	Introduction	2
2	IEEE 1588 basic overview	2
2.1	Synchronization principle	3
2.2	Timestamping	4
3	IEEE 1588 functions on MCX A577	5
3.1	System time register module	6
3.2	Transmit timestamping	7
3.3	Receive timestamping	7
3.4	PPS output	8
4	IEEE 1588 implementation for MCX A577	8
4.1	Hardware components	9
4.2	Software components	10
4.2.1	FreeRTOS OS	10
4.2.2	lwIP TCP/IP stack	10
4.2.3	PTP daemon	10
5	IEEE 1588 demo software detailed description	11
5.1	MCX A577 SDK ENET driver update	11
5.2	lwIP TCP/IP porting update	12
5.3	PTPd porting on FreeRTOS OS	15
5.4	FreeRTOS OS tasks and board configuration	18
6	Running the IEEE 1588 demo	19
6.1	Hardware setup	19
6.1.1	Board connection when using external LAN8741 PHY	19
6.1.2	Board connection when using internal 10BASE-T1S PHY	20
6.2	Clock synchronicity measuring	21
6.2.1	Measurement results using an external Ethernet PHY	22
6.2.2	Measurement results using internal 10BASE-T1S PHY	22
7	Conclusion	23
8	Acronyms	23
9	Note about the source code in the document	23
10	Revision history	24
	Legal information	25

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.