NXP Semiconductors
Application Note

Document Number: AN12149
Rev. 1, 09/2018

使用 PTPd，FreeRTOS 和 lwIP TCP / IP 协议栈在 i.MX RT 上实现 IEEE 1588 V2

1. 介绍

本应用笔记描述了如何在 FreeRTOS OS 上实现 i. IEEE 1588 V2 精确时间协议（PTP）。IEEE 1588 标准为工业自动化应用中的分布式控制节点提供了准确的时钟同步。

演示软件在 SDK 2.4.x IDE 上运行，是一个使用 SDK IDE 上 lwIP TCP / IP 协议栈的 PTP 守护程序（PTPd），并可在 FreeRTOS OS 上运行。 PTPd 是 PTP 的开源实现。

本文档介绍了 IEEE 1588 协议基础知识，IEEE 1588 功能以及 IEEE 1588 演示软件的详细说明。文档描述了如何在 Amazon FreeRTOS OS 上接入 PTPd，以及如何利用 ENET 输出比较功能去检查时钟同步状态。 本文档还描述了如何构建和运行这个演示。

目录

2. IEEE 1588 基本概述

IEEE 1588 标准是一个精确的时钟同步协议，也被称为精确时间协议（PTP）。 IEEE 1588 PTP 使时间能够分布在 Ethernet™网络上，并使用带有时间戳的节点来精确同步。

该标准的技术最初是由安捷伦科技公司开发的，用于分布式测量和控制任务。把不同的网络设备同步时间是一个难点，并且使他们记录下相关联的测试值和精确的时间戳。。

IEEE 1588 时间同步的典型应用包括:

•对时间精确度要求高的信息传输，要求在通信节点之间进行精确的时间同步。
•工业网络交换机，可通过单线分布式控制网络同步传感器和执行器，以控制自动化装配过程。
•在大型分布式电网交换机之间同步的电力线网络，可实现平稳的电力传输。
•测试/测量设备必须在许多不同的操作环境中与被测设备保持准确的时间同步。
•打印机，协作机器人系统和住宅网络。

这些应用要求设备之间的时间同步精度在微妙范围内。IEEE 1588 的显著特征是：通过与标准的网帧进行连接可以实现这种同步精度。

该解决方案使几乎任何性能的设备都可以参与易于操作和配置的高精度同步网络。

IEEE 1588 协议的其他主要优点包括:

•对于具有不同时间，分辨率和稳定性的异构分布式设备之间的亚微秒级同步，收敛时间不到一分钟。
•自动配置和分段。每个节点使用最佳主时钟（BMC）算法来确定段中的最佳时间。每个 PTP 节点都存储在指定的数据集中。这些功能将传输到同步报文中的其他节点。基于此，其他节点能够将其数据集与实际主节点的功能同步，并可以调整其时间。BMC 的循环运行还允许热插拔，也就是说，可以在传播期间连接或删除节点。
•简单的配置和操作，低计算资源要求和网络带宽消耗。

2.1. 同步原理

网络时钟按主从层次结构组织。IEEE 1588 识别主机时钟，然后建立双向定时交换，主机通过该定时交换向其从机发送消息以启动同步。然后，每个从站都将响应以使其自身与主站同步。在整个指定网络中重复此序列，以实现并维持时钟同步。

该过程从一个节点（主时钟）传输包含时间的报文开始。这个报文的传输时间被一个时钟记录，并在第二个报文中传输。通过将第一和第二报文中包含的时间戳信息与其自己的时钟进行比较，接收器可以计算自己的时钟与主时钟之间的时差（见图 1）。 第一个同步报文和第二个在不同的时间点传输。一些 IEEE 1588 系统启用硬件时间戳，并将实际时间戳插入同步消息中。在这种情况下，不需要后续消息（单步操作模式）。

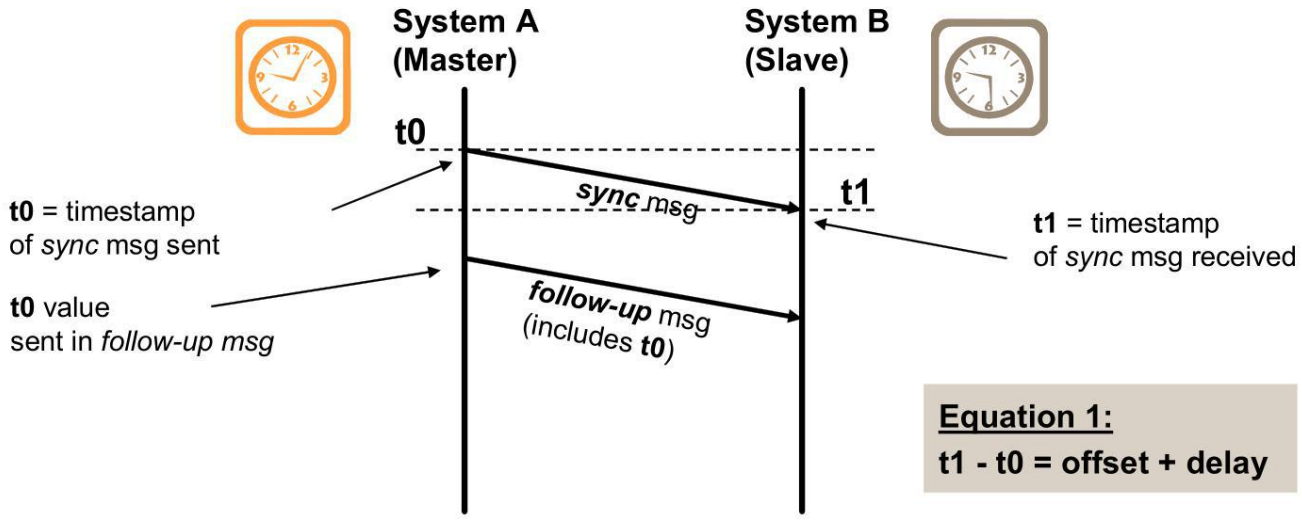

图 1. 偏移和延迟测量-同步消息，后续消息

报文的传播时间是在从机和主机之间的第二次传输过程中循环确定的（延迟报文）。然后，从机可以调整其时钟并使之适应当前的总线传播时间（见图 2）。 delay_req 和 delay_resp 消息是点对点的，但是出于简化的原因，它们在不同的时间发送。

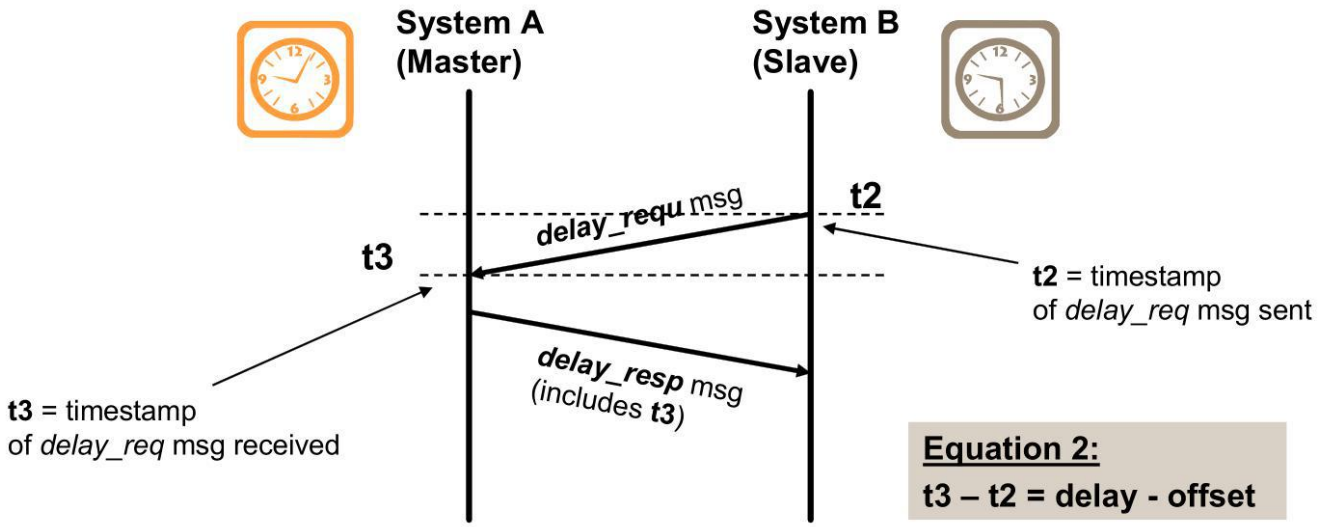

图 2. 偏移和延迟测量—延迟消息

图 3 显示了 IEEE 1588 同步序列（一个周期）的示例以及主节点和从节点之间的实际偏移和延迟的计算。

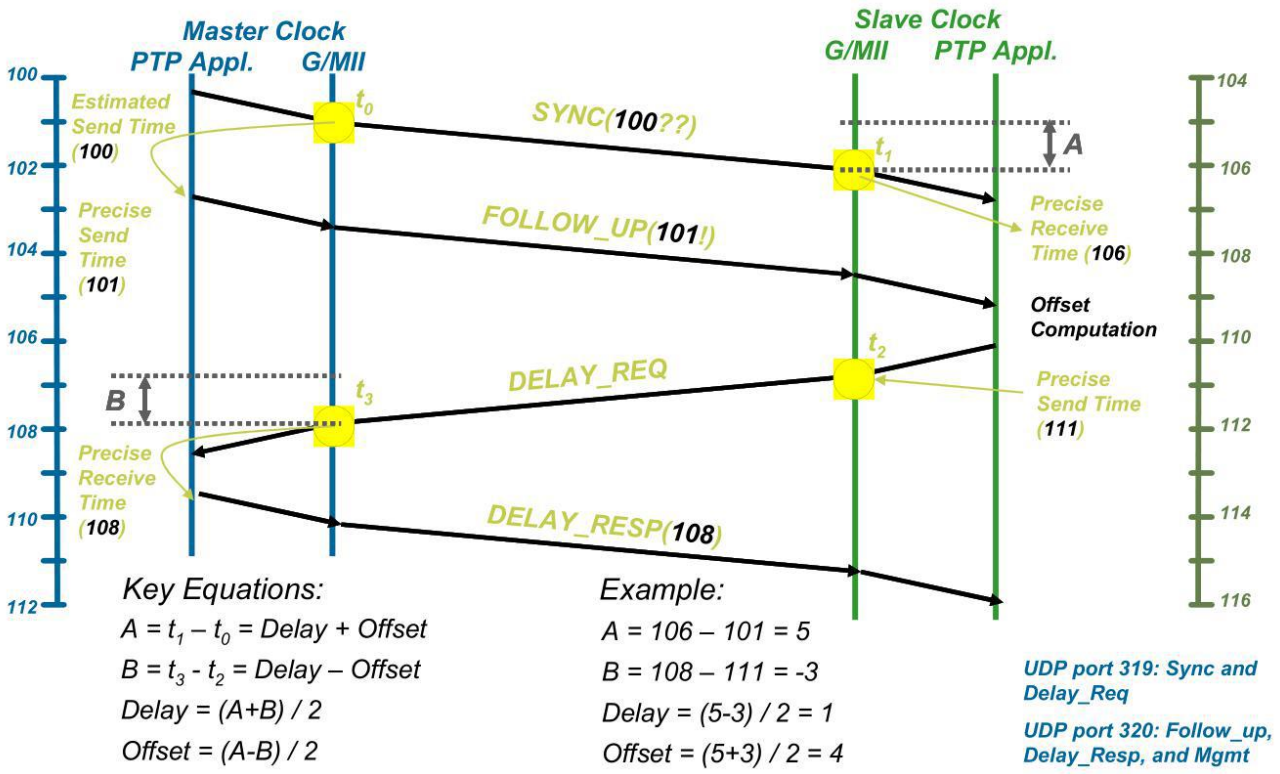

图 3. IEEE 1588 同步消息序列

若想了解有关 IEEE 1588 标准的更多信息，请访问美国国家标准技术研究院的网页（https://www.nist.gov/）。

2.2. 时间戳记

使用标准以太网模块，可以将 PTP 协议完全实现到软件中。由于时间戳信息是在应用层上的，因此在主设备和从设备上运行的软件堆栈引入的延迟波动意味着只能达到有限的精度（请参见图 4）。

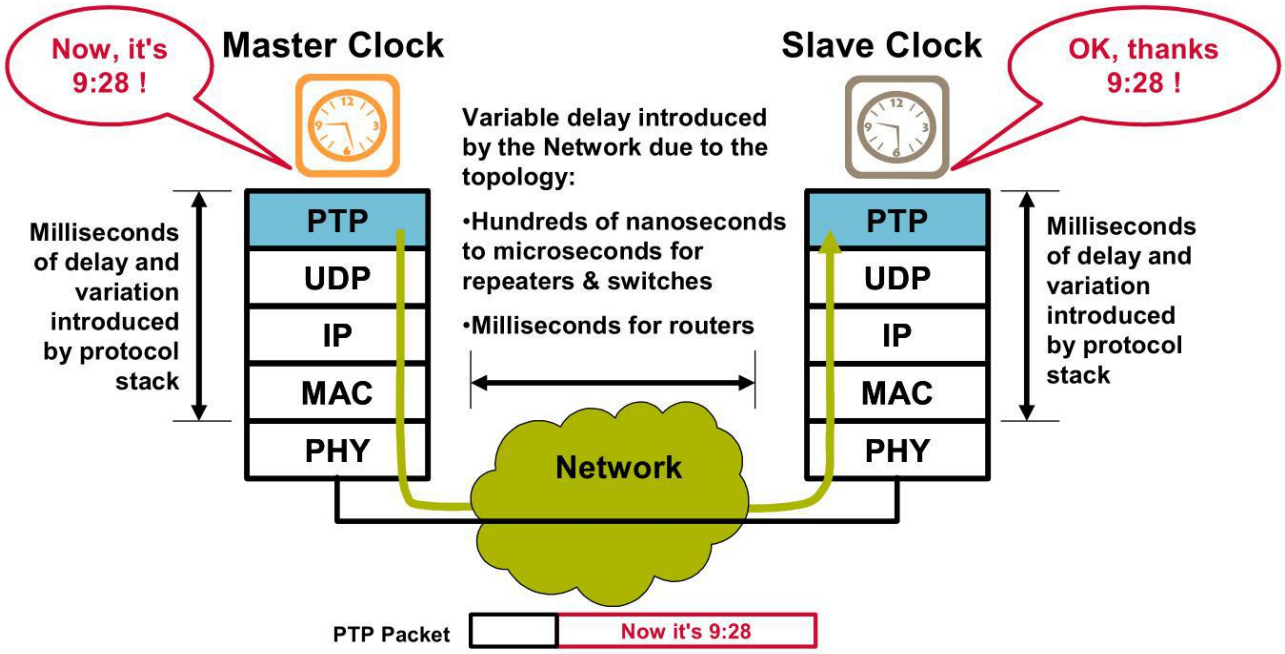

图 4. 软件时间戳实现

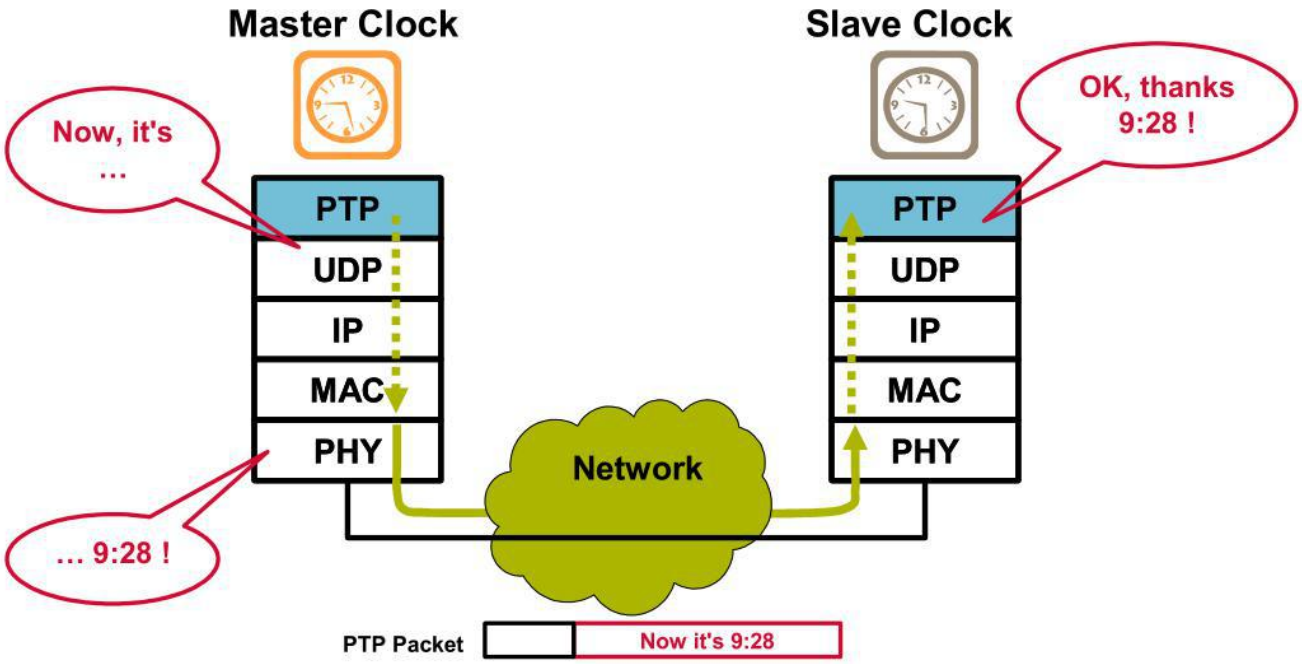

图 5. 硬件时间戳实现

通过使时间戳更接近物理接口，，可以最大程度地减少协议栈延迟的影响（请参见图 5）。具有时间戳功能的专用硬件可以显著提高同步准确性。

3. 3.i.MX RT 上的 IEEE 1588 功能

i.MX RT10xx 使用 MAC-NET 核心（与 10 / 100-Mbit / s MAC 结合使用）去加速各种常见网络协议（例如 IP，TCP，UDP 和 ICMP），从而为客户端应用程序提供了线速服务。ENET 内的 uDMA 可优化 ENET 内核与 SoC 之间的数据传输，并加强缓冲区的编程模型。为了实现 IEEE 1588（或类似的）时间同步协议，MAC 与时间戳模块结合在一，然后设置 ENET_ECR 中的 EN1588 比特启用 1588 支持。

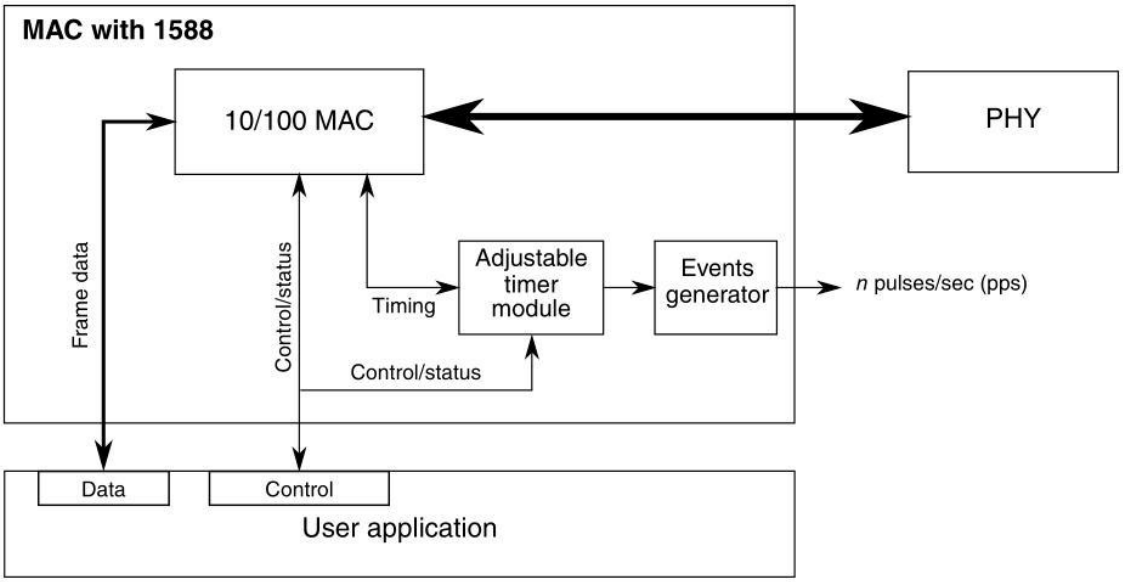

图 6. IEEE 1588 功能概述

3.1. 可调定时器模块

可调计时器模块（TSM）实现了自由运行计数器（FRC），该计时器会生成时间戳。 FRC 可以把时间戳时间设置为任意值。

专用的逻辑校正可以调整计时器以实现与远程主机的同步，并为本地系统提供同步的时序参考。为了使计时器和其他的同步功能顺利进行，可以为计时器设置中断时间。

计时器的基本单位通常是 1 秒；因此，其值的范围是 0 到（1×10^9）-1。如果触发了中断机制，软件可以保留秒和小时的时间值（如有必要）。

可调定时器包括一个可编程计数器和一个校正计数器。这两个计数器的周期及其递增速率可以自由配置，从而可以对计时器进行非常精细的调整。

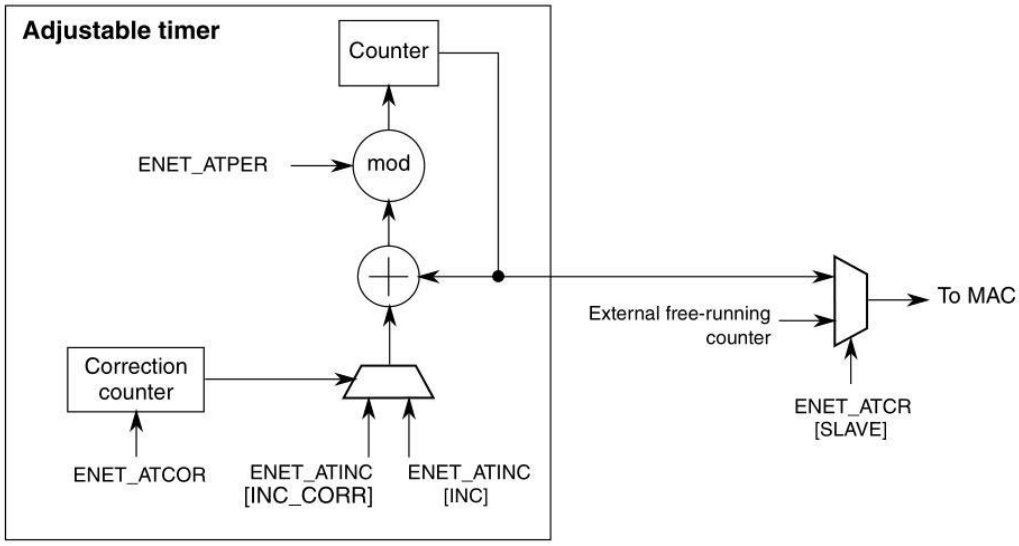

图 7. 可调计时器实现细节

计数器提供当前时间。在每个时间戳周期中，都会将恒定值添加到当前时间。该值取决于所选的时间戳时钟频率。例如，如果它工作在 125 MHz，则将增量设置为 8 表示 8 ns。

在 ENET_ATPER（定时器周期寄存器）中配置的周期定义了计数器圈数的模数。在通常的情况下，周期设置为 1×10^9，以便计数器每秒都计数。所有时间戳均表示 1 ns 内的纳秒。当达到此周期时，计数器将以周期为模，重新开始运行。这意味着它不一定从零开始，而是假设周期设置为 1×10^9，计数器会加载值（Current + Inc –（1×10^9））。

校正计数器是完全独立的，并且在每个带有时间戳的周期内加一。当计数器达到 ENET_ATCOR（定时器校正寄存器）中配置的值时，它将重新启动并增加一次校正值。

法线和校正增量通常在 ENET_ATINC 中配置。要加快计时器速度，请将校正增量设置为高于正常增量值。要降低计时器速度，请将校正增量设置为低于正常增量值。

校正计数器仅定义校正动作的距离，而不是量。 这样就可以进行非常精细的校正和降低晃动（在 1 ns 范围内），而与所选时钟频率无关。

3.2. 发送时间戳

传输时间戳需要 1588 个事件帧。客户端应用程序（例如，MAC 驱动程序）应检测到 1588 个事件帧，并与 TxBD（增强型发送缓冲区描述符）中的 TS 比特包装在一起。

如果设置了 TxBD [TS]，则 MAC 在 ENET_ATSTMP）中记录该帧的时间戳。 ENET_EIR（中断事件寄存器）中的 TS_AVAIL 位置表示有新的时间戳可用。

在 TxBD 中，该软件通过发送时间戳的时间和等待 ENET_EIR[TS_AVAIL] 的回应 来实现握手过程。然后从 ENET_ATSTMP 寄存器中读取时间戳。其他不适用 TxBD 的 客户不会影响时间戳的接受。

3.3. 接收时间戳

当信息的定界符被检测到时，MAC 会锁存计时器的值，并在 RxBD 中捕获时间戳，知道对所有接收到的帧完成此操作。

3.4. 时间同步

可调计时器模块可用于将本地时钟同步到远程主机。它实现了自由运行的 32 位计数器，还包含一个附加的校正计数器。

校正计数器增加或减小速率，从而使计时器能够进行非常精细的改进，并且只带来了很低的晃动率。

应用软件实现所需的控制算法（在从属方案中），设置校正值来抵消本地振荡器的漂移，并将定时器锁定到网络上的远程主时钟。

计时器和所有与时间戳相关的信息应配置等于 1 秒的纳秒值。因此，值的范围从 0 到（1×10^9）–1。在此应用中，秒计数是通过软件使用中断功能实现的，该中断功能在纳秒计数器以 1×10^9 换行时执行。

3.5. 输入捕捉和输出比较的代码块

输入捕捉和输出比较模块可用于为输入和输出事件提供精确的硬件时序。 IEEE 1588 计时器具有四个通道。每个通道都使用 1588 计数器支持输入捕获和输出比较。

在输入捕捉模式下，当发生相应的外部事件时，TCCRn（定时器比较捕捉寄存器，n = 1、2、3、4）锁存时间值。 这个事件可以是 1588_TMRn 信号的上升值或下降值。 事件使得相应的 TCSRn [TF]（定时器控制状态寄存器）被标记，说明发生了输入捕捉。 如果 TCSRn [TIE]字段启用了相应的中断，则可以生成一个中断。

在输出比较模式下，TCCRn 比较寄存器加载了相应事件发生的时间。 当 ENET 自由运行计数器值与 TCCRn 寄存器中的输出比较参考值匹配时，则做一个标记（TCSRn [TF]），表示发生了输出比较。如果相应的中断发生，则 1588_TMRn 输出信号 就会声明。

4.4. i.MX RT 的 IEEE 1588 实现

MIMXRT10xx 板可以在 IEEE 1588 V2 PTP 的硬件平台上操作。 该解决方案包括 NXP ENET 驱动程序，PHY 驱动程序， FreeRTOS OS 和 EVK lwIP TCP / IP 协议栈。 IEEE1588 V2 PTP 由 PTP 后台应用程序实现，该应用程序是开源的。 图 8 显示了此方案的硬件和软件组件。

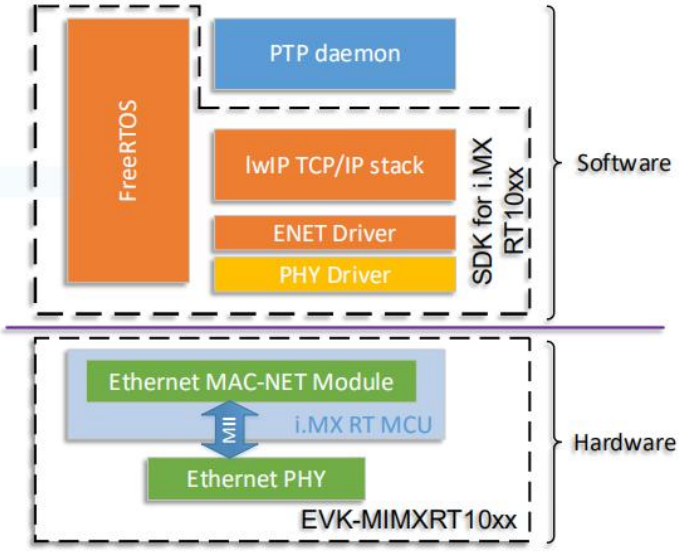

图 8. 适用于 i.MX RT10xx MCU 的 IEEE 1588 解决方案

4.1. 硬件部分

i.MX RT10xx EVK 板的作用是展示处理器。 i.MX RT10xx EVK 开发板是入门级开发板，可帮助您快速熟悉处理器并加快实现自己的设计的速度。 该开发板的主要功能包括：

•i.MX RT10xx MCU。

•32 MB @ 166 MHz SDRAM。
•512 Mb 超级闪存（仅适用于 i.MX RT1050），64 Mbit 的四 SPI 闪存和 TF 卡插槽。
•带有 KSZ8081RNB PHY 的 10/100 Mbit / s 以太网连接器。
•USB 2.0 OTG /主机连接器。
•3.5 毫米立体声音频耳机插孔，麦克风和扬声器输出连接器。
•显示连接器和 CMOS 传感器接口（在 i.MX RT1020 上不可用）。
•CAN 总线连接器，带有 DAP-link 的 OpenSDA 和 Arduino 接口。
•用于电源的 5 V DC 插孔。

i.MX RT1050 是业界第一款交叉处理器，这是一个新处理器系列，采用了恩智浦对 Arm®Cortex®-M7 内核的高级实现。它旨在通过具有高度集成性和安全性以及 MCU 级可用性的支持下一代物联网应用程序。它以高达 600 MHz 的速度运行，以提供具有最佳实时功能的高 CPU 性能。i.MX RT 1050 提供各种内存接口，包括 SDRAM，原始 NAND 闪存，NOR 闪存，SD / eMMC，四路 SPI，HyperBus，以及各种其他接口，用于连接外设，例如 WLAN，Bluetooth™，GPS，显示器以及 camera 传感器。与其他 i.MX 处理器一样，i.MX RT1050 还集成了丰富的音频和视频功能，包括 LCD 显示屏，基本 2D 图形，camera 接口以及 SPDIF 和 I2S 音频接口。

i.MX RT1060 将片上 SRAM 倍增至 1 MB，同时保持与 i.MX RT1050 的引脚对引脚兼容性。它引入了理想的实时应用附加功能，例如高速 GPIO，CAN-FD 和同步并行 NAND / NOR / PSRAM 控制器。它也以 600 MHz 运行。

i.MX RT1020 采用低成本 LQFP 封装提供了高性能功能集，从而进一步简化了电路板的设计和布局。 该处理器删除了多媒体组件，并针对低成本应用将片上 SRAM 减小到 256 KB。 i.MX RT1020 以 500 MHz 运行。

有关更多信息，请参见 www.nxp.com 上的相应参考手册。

4.2. 软件部分

IEEE 1588 软件实现包括用于 i.MX RT10xx EVB 板和 PTP 守护程序的 MCUXpresso SDK IDE。MCUXpresso SDK IDE 是一个开发环境，用于在 NXP MCU 上开发应用程序，包括外围驱动程序，中间件和实时操作系统。

4.2.1. FreeRTOS OS 操作系统

FreeRTOS OS 是一个实时内核（或实时调度程序），您可以在其之上构建满足严格实时要求的嵌入式应用程序。 FreeRTOS OS 提供了多种任务，互斥体，信号量和软件计时器的方法。 无节拍模式为低功耗应用提供。线程优先级决定应该执行哪个线程。 MCUXpresso SDK IDE 为开发板提供的 FreeRTOS 操作系统的版本是 10.0.1。 集成到 MCUXpresso IDE 中的 FreeRTOS OS 软件包具有以下功能：
•删除了与 SDK IDE 不相关的文件，例如 FreeRTOS OS 的扩展名（CLI，FAT_SL 和 UDP）和文件夹，例如演示文件夹和嵌套文件夹。

•将 SystemCoreClock 全局变量添加到 FreeRTOS OS port.c 和 FreeRTOSConfig.h 文件中。
•启用无滴答模式。 有关更多信息，请访问： www.nxp.com/freertos.
•启用了 KDS Task Aware Debugger，应用了 FreeRTOS 补丁以启用 configRECORD_STACK_HIGH_ADDRESS 宏。
•通过为 vTaskSwitchContext 添加__attribute __（(已使用)）在 GCC 中启用了-flto 优化。

有关 FreeRTOS 操作系统及其发行版的详细信息，请访问 https://www.freertos.org/。

4.2.2. lwIP TCP/IP 协议栈

lwIP 是 TCP / IP 协议的轻量级实现，可以在 C 源代码中免费获得，并且可以从开发网页下载。 它是完全模块化的，并且足够小以减少小型嵌入式系统的 RAM 使用量。 核心协议栈是 IP，您可以根据需要和可用的内存来选择添加 TCP， UDP， DHCP 和许多其他协议。若想了解有关 lwIP 的更多信息，请访问 http://www.nongnu.org/lwip/2_1_x/index.html。

EVK-MIMXRT1050 的 MCUXpresso SDK IDE 集成了 lwIP TCP / IP 协议栈，该协议栈在 MCUXpresso SDK IDE 上运行。SDK IDE 中用于 i.MX RT10xx EVB 板的 lwIP 软件包版本为 2.4.x。若想了解有关更多信息，请参见《 lwIP TCP / IP 堆栈和 MCUXpresso SDK 集成用户指南》（文档 MCUXSDKLWIPUG）。

4.2.3. PTP 后台程序

PTP 提供了与 LAN 连接的计算机的精确时间协调，该计算机主要用于仪表和控制系统。PTP 进程（PTPd）是 IEEE Std 1588-2008 定义的 PTP 版本 2 的开源实现。

PTPd 协调一组连接局域网的计算机时钟。 即使在有限的平台上，它也可以实现微秒级的协调。PTPd 在 C 源代码中可用，并且可以容易地在 FreeRTOS OS 上移植用于嵌入式系统。 大多数与系统有关的代码在 <install_dir> / src / dep 文件夹中。 本演示中使用的 PTPd 软件包版本为 2.2.2。可从 github.com/ptpd/ptpd/releases 获得。

5. IEEE1588 演示软件的详细说明

该演示程序仅适用于 Arm IDE 8.30 的 IAR EmbeddedWorkbench®进行编译和测试。 SDK 版本为 2.4.x，PTPd 版本为 2.2.2。 i.MX RT10xx ENET 支持带有硬件时间戳的 IEEE 1588。 若要启用硬件时间戳功能并在 i.MX RT10xx EVB 板上运行演示，必须更新与 lwIP TCP / IP 端口相关的原始代码。 需要 ENET 驱动程序更新才能测试时钟偏移收敛范围和与 IEEE 1588 功能相关的错误修复。

5.1. i.MXRT SDK IDE ENET 驱动程序更新

ENET 驱动程序更新包括一些错误修复程序并启用了 ENET 输出比较功能进行测试。启用输出比较功能可以监视本地从时钟偏移到远程主机的收敛。

为了获得正确的 IEEE 1588 计时器值，必须将以下红色代码添加到 fsl_enet.c 文件中的 ENET_Ptp1588GetTimer 函数中，并且 ENET_1588TIME_DELAY_COUNT 必须足够大，以具有至少六个时钟周期的延迟才能获得准确的 1588 time 。

```
void ENET_Ptp1588GetTimer(ENET_Type *base, enet_handle_t *handle, enet_ptp_time_t *ptpTime)
{
    ……
    uint16_t count = ENET_1588TIME_DELAY_COUNT;
    ……
    base->ATCR |= ENET_ATCR_CAPTURE_MASK;
    __DSB();
    /* Add at least six clock cycle delay to get accurate time.
       It's the requirement when the 1588 clock source is
       slower than the register clock.
    */
    while (count--)
    {
        __NOP();
    }
    ……
}
```

ENET 1588 计时器具有四个通道，这些通道支持使用 1588 计数器进行输入捕获和输出比较。 为了监视在运行时主时钟和从时钟之间的同步，必须启用 ENET 输出比较功能，这样使运行的计数器值与输出比较参考值匹配时生成每秒脉冲（PPS）信号。 如果主时钟和从时钟正确同步，并且主时钟和从时钟的输出比较参考值设置为相同，则可以使用示波器观察主时钟和从时钟同步的 1588 定时器输出信号。

代码更改为启用 fsl_enet.h 中的输出比较，包括枚举类型 enet_event_event 定义中的附加枚举值和 struct _enet_handle 类型中的两个成员：

红色的代码必须添加或修改。

```
typedef enum _enet_event
{
   ……
   kENET_TimeStampAvailEvent, /*!< Time stamp available event.*/
   kENET_TimeStampCaptureEvent /*!< Time Stamp capture event. */
} enet_event_t
```

```
struct _enet_handle
{
   ……
#ifdef ENET_ENHANCEDBUFFERDESCRIPTOR_MODE
   enet_ptp_time_data_ring_t txPtpTsDataRing;
   enet_ptp_timer_channel_t mPtpTmrChannel; /*!< PTP 1588 timer channel. */
   uint32_t ptpNextCounter;                 /*!< PTP 1588 next output compare counter value */
#endif                                      /* ENET_ENHANCEDBUFFERDESCRIPTOR_MODE */
};
```

要在 IEEE 1588 计时器中启用输出比较功能，必须在 ENET_Ptp1588Configure 和 ENET_Ptp1588TimerIRQHandler 函数中添加或修改红色代码。

```
void ENET_Ptp1588Configure(ENET_Type *base, enet_handle_t *handle, enet_ptp_config_t
*ptpConfig)
{
    ……
    handle->msTimerSecond = 0;
    handle->mPtpTmrChannel = ptpConfig->channel;
    /* Set the IRQ handler when the interrupt is enabled.
    */ s_enetTxIsr = ENET_TransmitIRQHandler;
    ……
}
```

```
void ENET_Ptp1588TimerIRQHandler(ENET_Type *base, enet_handle_t *handle)
{
    ……
    else if (kENET_TsAvailInterrupt & base->EIR)
    {
       ……
    }
    else if (base->CHANNEL[handle->mPtpTmrChannel].TCSR & ENET_TCSR_TF_MASK)
    {
        ENET_Ptp1588SetChannelCmpValue(base, handle->mPtpTmrChannel, handle->ptpNextCounter);
        do {
            ENET_Ptp1588ClearChannelStatus(base, handle->mPtpTmrChannel);
        } while (true == ENET_Ptp1588GetChannelStatus(base, handle->mPtpTmrChannel));
    }
    ……
}
```

5.2. lwIP TCP/IP 移植更新

本节描述了 lwIP 移植代码的修改，以支持 PTP 演示。这包括<sdk_install_dir> / middleware / lwip / port 文件夹中的 *lwipopts.h*，*ethernetif.h* 和 *ethernetif.c* 文件。

PTP 守护程序演示将 SO_REUSEADDR 选项用于套接字，DNS（域名系统）协议和 IGMP（Internet 组管理协议）协议。它不使用带有静态 IP 地址的 DHCP（动态主机配置协议）。

lwipopts.h 中的以下宏必须定义为相应的值：

```
/* 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
```

SDK IDE 版本中的默认 lwIP 软件包不支持 PTP。ENET 初始化功能不涉及任何 1588 计时器功能。*ethernetif.h* 和 *ethernetif.c* 的代码更新主要涵盖 ENET 1588 计时器例程，例如初始化 1588 计时器，启用用于测试的计时器通道输出比较功能，设置/获取时间，调整 1588 计时器频率以及获取 发送/接收帧的时间戳。 #if LWIP_TPT 和#endif 对包含与 PTP 相关的所有代码。

此代码段应添加到 *ethernetif.h* 中，以声明 1588 计时器的例程：

```
#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
```

以上功能在 *ethernetif.c* 文件中实现。 从 *enet_init（）* 函数返回之前，将实现并调用 ENET 1588 计时器初始化函数 *ethernet_ptptime_init（）*。 *ethernet_ptptime_enablepps（）* 函数用于启用计时器通道输出比较功能进行测试，并根据传递的参数在 *ethernet_ptptime_init（）* 函数中调用。

这是 *ethernet_ptptime_enablepps（）* 函数的代码：

```
static void ethernet_ptptime_enablepps(struct ethernetif *ethernetif,
                                       enet_ptp_timer_channel_t tmr_ch)
{
    uint32_t next_counter = 0;
    uint32_t tmp_val = 0;

    /* clear capture or output compare interrupt status if have.
    */ ENET_Ptp1588ClearChannelStatus(ethernetif->base, tmr_ch);

    /* It is recommended to double check the TMODE field in the
     * TCSR register to be cleared before the first compare counter
     * is written into TCCR register. Just add a double check.
    */ tmp_val = ethernetif->base->CHANNEL[tmr_ch].TCSR;
    do {
        tmp_val &= ~(ENET_TCSR_TMODE_MASK); ethernetif-
        >base->CHANNEL[tmr_ch].TCSR = tmp_val; tmp_val =
        ethernetif->base->CHANNEL[tmr_ch].TCSR;
    } while (tmp_val & ENET_TCSR_TMODE_MASK);

    tmp_val = (ENET_NANOSECOND_ONE_SECOND >> 1);

    ENET_Ptp1588SetChannelCmpValue(ethernetif->base, tmr_ch, tmp_val);

    /* Calculate the second the compare event timestamp */
    next_counter = tmp_val;

    /* Compare channel setting. */
    ENET_Ptp1588ClearChannelStatus(ethernetif->base, tmr_ch);

    ENET_Ptp1588SetChannelOutputPulseWidth(ethernetif->base, tmr_ch, false, 4, true);

    /* Write the second compare event timestamp and calculate
     * the third timestamp. Refer the TCCR register detail in the spec.*/
    ENET_Ptp1588SetChannelCmpValue(ethernetif->base, tmr_ch, next_counter);
    /* Update next counter */
    ethernetif->handle.ptpNextCounter = next_counter;
}
```

这段代码是 ENET 1588 计时器初始化函数 *ethernet_ptptime_init（）*及其相关的内存：

```
static struct ethernetif * ptp_ethernetif  = NULL;

/* Buffers for store receive and transmit timestamp. */
//static sys_mutex_t ptp_mutex;
enet_ptp_time_data_t g_rxPtpTsBuff[ENET_RXBD_NUM];
enet_ptp_time_data_t g_txPtpTsBuff[ENET_TXBD_NUM]

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

  assert(ethernetif);
  ptp_ethernetif = ethernetif;

  /* Config 1588 */
  memset(&ptp_cfg, 0, sizeof(enet_ptp_config_t));
  ptp_cfg.channel = tmr_ch;
  ptp_cfg.ptpTsRxBuffNum = ENET_RXBD_NUM;
  ptp_cfg.ptpTsTxBuffNum = ENET_TXBD_NUM;
  ptp_cfg.rxPtpTsData = &g_rxPtpTsBuff[0];
  ptp_cfg.txPtpTsData = &g_txPtpTsBuff[0];
  ptp_cfg.ptp1588ClockSrc_Hz = ptp_clk_freq;
  ENET_Ptp1588Configure(ptp_ethernetif->base, &ptp_ethernetif->handle, &ptp_cfg);

  if (true == pps_en)
  {
    ethernet_ptptime_enablepps(ptp_ethernetif, tmr_ch);
  }
  else
  {
    ENET_Ptp1588SetChannelMode(ptp_ethernetif->base, tmr_ch, kENET_PtpChannelDisable, false);
  }
}
```

红色语法用于调用 *enet_init（）*函数中的 *ethernet_ptptime_init（）*函数：

```
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,
                          PTP_TEST_APP_CHANNEL);
#endif
    ENET_ActiveRead(ethernetif->base);
}
```

*ethernet_ptptime_adjfreq（）*函数通过在 ENET_ATCOR 寄存器中设置非零校正回绕值来定义校正计时器时钟周期数，从而调整 1588 计时器频率。 校正增量值在 ENET_ATINC 寄存器的 INC_CORR 字段中设置。如果 INC_CORR 的值大于 INC 字段中的值，则加快 1588 计时器的速度。如果 INC_CORR 的值低于 INC 字段中的值，则减慢 1588 计时器的速度。

这是 *ethernet_ptptime_adjfreq（）* 函数的代码:

```
void ethernet_ptptime_adjfreq(int32_t incps)
{
    int32_t neg_adj = 0;
    uint32_t corr_inc, corr_period;

    assert(ptp_ethernetif);

    /*
     * incps means the increment rate (nanseconds 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)
    {
        ptp_ethernetif->base->ATCOR &= ~ENET_ATCOR_COR_MASK; /* Reset PTP frequency
        */ return;
    }

    if (incps < 0)
    {
        incps = - incps;
        neg_adj = 1;
    }

    corr_period = (uint32_t)PTP_CLOCK_FRE_RT / incps;

    /* neg_adj = 1, slow down timer, neg_adj = 0, speed up timer
    */ corr_inc = (neg_adj) ? (PTP_AT_INC - 1) : (PTP_AT_INC + 1);

    ENET_Ptp1588AdjustTimer(ptp_ethernetif->base, corr_inc, corr_period);
}
```

实现 *ethernet_ptptime_settime（）*，*ethernet_ptptime_gettime（）*，*enet_get_rxframe_time（）* 和 *enet_get_txframe_time（）* 函数来包装 *ENET_Ptp1588GetTimer（）*，*ENET_Ptp1588SetTimer（）*，*ENET_GetRxFrameTime（）*函数和 *ENET_GetRxFrameTime（）*函数。

5.3. 5.3 FreeRTOS 操作系统上的 PTPd 移植

默认的 PTPd 源代码适用于 FreeBSD，NetBSD，Mac OS X 和 Linux 操作系统。要使用 lwIP 和 SDK 驱动程序的操作系统，必须移植或添加与操作系统相关的代码，与网络相关的代码和硬件时间戳代码。移植的工作包括 FreeRTOS 操作系统下的 PTPd 任务，系统时间例程，系统服务，软件计时器，与网络套接字的交互以及对 PTP 协议的较小修改。本文档未讨论 IAR 嵌入式工作台 IDE 在编译和链接期间对 Arm 进行的简单代码修改。

ptpd/src 文件夹里的代码可适用于 PTPd 应用。PTPd 上的文件必须更新才能在 FreeRTOS 系统上使用。*ptpd.c* 文件中的原始 *main（）*函数更改为 *ptpd_thread（）*，该函数是作为 FreeRTOS OS 任务创建的。除了用于主机或者从机的变量外，其他的命令行参数都需要删除。当操作系统任务被创建时，这些变量会通过参数来传递。

PTPd 通用代码中的另一个重要修改是在 *protocol.c* 文件中。 默认的 PTPd 应用程序以一个真实的网点在类 UNIX 系统中运行。设备可以接收自己发送的消息，因为它们是使用 UDP / IP 多播消息发送的。设备收到原始协议源代码中自己发送的相应的同步消息或 Pdelay_Resp 事件消息后，将发送 *Follow_Up* 或 *Pdelay_Resp_Follow_Up* 消息。 由于此演示是通过点对点连接运行的，因此该设备不会收到其自身发送的事件消息。 发送相应的事件消息后，必须修改代码以发送这两个后续消息。 作为此更改的结果，必须使用特定的超时值调用 *netSelect（）*函数，替换原始的 NULL（无超时），因为初始 NULL（无超时）会阻塞 *select（）*函数无限期地等待文件描述符。

ptpd/src/dep 中的文件是特定于端口的源代码文件，并且取决于操作系统，TCP / IP 协议栈和硬件平台。主要更改涉及 *net.c*，*startup.c*，*sys.c* 和 *timer.c* 文件。它们的名称带有*_mcu* 后缀，以区别于原始文件。

在 *FreeRTOSConfig.h* 文件中将 *configUSE_TIMERS* 设置为 1 时，FreeRTOS OS 提供软件计时器功能。对 *timer_mcu.c* 文件的修改包括以下两个功能：

```
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);
  }
}
```

sys_mcu.c 文件具有一些与时间相关的例程，以提供与演示中同步的低级硬件计时器的接口。这些例程大多数都调用第 5.2 节“ lwIP TCP / IP 端口更新”中描述的功能。

*adjFreq（）*函数代码如下：

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

有两个睡眠函数可使用 *FreeRTOS vTaskDelay（）*函数将当前线程置于休眠状态。其余的更改仅删除信息输出和/或日志文件的代码。

```
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);
}
```

startup_mcu.c 文件将删除所有与 OS 相关的信号功能和命令行参数解析代码。 添加了 *ptpd_init（）*函数来为 PTPd 应用程序创建 FreeRTOS OS 任务。

原始代码中的传输帧的时间戳由 OS 提供，并且在调用 *recvmsg（）*函数之后，可以从接收到的数据中提取接收到的时间戳。 本演示使用 ENET 1588 计时器中的硬件时间戳功能。 这些代码必须移植到 i.MX RT10xx MCU 上的 FreeRTOS 操作系统和 ENET 1588 计时器上。 *net_mcu.c* 文件中的代码提供了移植的功能。

findIface（）函数直接返回 lwIP 中使用的默认网络接口，实现方式如下：

```
#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;
}
```

由于移植的代码不使用 *recvmsg*（）函数启用时间戳，因此 *netInit*（）函数中未调用 *netInitTimestamping*（）函数，也没有实现只返回 TRUE 的转储函数。

i.MX RT10xx MCU 上的 ENET 驱动程序提供了两个 API，以查询当时发送或接收的事件消息的时间戳。为了获得时间戳，应从包含已接收或已发送事件消息的缓冲区中打包 ENET PTP 消息数据和 *enet_ptp_time_data_t* 类型定义的时间戳数据。为此任务添加了 *netPackPtpData*（）函数，并在以下代码中明确列出了该函数：

```
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);
}
```

recv（）套接字函数替换了 *netRecvGeneral*（）函数中的 *recvmsg*（）函数，以读取常规消息的帧。这是 *netRecvGeneral*（）函数实现的代码：

```
ssize_t netRecvGeneral(Octet * buf, TimeInternal * time, NetPath * netPath)
{
       ssize_t ret;

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

       return ret;
}
```

*netRecvEvent（）*函数读取事件消息的帧并返回低级 ENET 驱动程序提供的时间戳。这是其实现的代码：

```
ssize_t netRecvEvent(Octet * buf, TimeInternal * time, NetPath * netPath)
{
	ssize_t ret;
	enet_ptp_time_data_t ptpTimeData;

	getTime(time); /* Current time for timestame in case of reading from driver failed.
	*/ ret = recv(netPath->eventSock, buf, PACKET_SIZE, MSG_DONTWAIT); if (ret <= 0) {

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

	if (!time) {
		ERROR("null receive time stamp argument\n");
		return 0;
	}

	netPackPtpData(buf, &ptpTimeData);
	if(!enet_get_rxframe_time(&ptpTimeData)){
		time->nanoseconds = ptpTimeData.timeStamp.nanosecond; time-
		>seconds = (Integer32)ptpTimeData.timeStamp.second;
	}
	return ret;
}
```

*netSentEvent（）*和 *netSentPeerEvent（）*函数均发送相应事件消息的帧并返回该帧的时间戳。默认实现不支持硬件时间戳。 为了启用硬件时间戳，*netSentEvent（）*和 *netSentPeerEvent（）*函数将指针的另一个输入参数添加到包含返回的时间戳的缓冲区中。

此代码段以红色显示添加到 *netSentEvent*（）和 *netSentPeerEvent*（）函数中的代码：

```
ssize_t netSendEvent(Octet * buf, UInteger16 length, TimeInternal * time, NetPath * netPath,
Integer32 alt_dst)
{
    ……
    enet_ptp_time_data_t ptpTimeData;
    ……

    getTime(time); /* Current time for timestame 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 操作系统任务和板卡配置

在演示应用程序中使用 FreeRTOS 操作系统创建了三个任务线程：

•*stack_init* 任务-在主函数中创建。此任务将初始化 lwIP TCP / IP 协议栈以及静态 IP 地址设置，网络掩码配置，网关地址配置，MAC 地址配置和以太网硬件初始化。然后，它启动 PTPd 任务。 最后，任务在初始化 lwIP 和 PTPd 任务后，通过调用 *vTaskDelete*（）函数来删除自身。
•*tcpip_thread* 任务-由 *stack_init* 任务在 TCP / IP 初始化期间创建。通过运行 lwIP 主任务来访问 lwIP 核心功能。
•*ptpd_thread* 任务-由 *stack_init* 任务创建。此任务运行 PTPd 应用程序。创建任务时传递的参数表示主机或从机。

演示启用了 1588 计时器的单通道输出比较功能。输出比较事件发生时，其输出信号将根据配置来确定。1588 计时器的通道 3 用于在此演示中为 i.MX RT1050 和 i.MX RT1060 MCU 生成输出比较事件。输出信号被路由至 GPIO_AD_B1_02 引脚。以下语法在 *pin_mux.c* 文件中将 GPIO_AD_B1_02 配置为 ENET_1588_EVNT2_OUT（通道 3 的输出信号）：

```
/* GPIO_AD_B1_02 is configured as 1588_EVENT2_OUT */
IOMUXC_SetPinMux(IOMUXC_GPIO_AD_B1_02_ENET_1588_EVENT2_OUT, 0U);

/* GPIO_AD_B0_12 PAD functional properties */
IOMUXC_SetPinConfig(IOMUXC_GPIO_AD_B1_02_ENET_1588_EVENT2_OUT, 0x10B0u);
```

1588 计时器的通道 2 用于在此演示中为 i.MX RT1020 MCU 生成输出比较事件。 输出信号被路由至 GPIO_SD_B1_02 引脚。 此语法在 *pin_mux.c* 文件中将 GPIO_SD_B1_02 配置为 ENET_1588_EVNT1_OUT（通道 2 的输出信号）：

```
/* GPIO_SD_B1_02 is configured as 1588_EVENT1_OUT */
IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B1_02_ENET_1588_EVENT1_OUT, 0U);

/* GPIO_SD_B0_12 PAD functional properties */
IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B1_02_ENET_1588_EVENT1_OUT, 0x10B0u);
```

1588 定时器由 *ref_enetpll2*（由以太网 PLL 生成）提供时钟，必须启用该定时器。以太网 PLL 初始化如下：

```
void BOARD_InitModuleClock(void)
{
    const clock_enet_pll_config_t config = {true, true, 1, 0};
    CLOCK_InitEnetPll(&config);
}
```

其他特定于板的初始化代码与 *<sdk_install_dir> / boards / evkmimxrt1050 / driver_examples / enet / txrx_ptp1588_transfer* 文件夹中的 *enet_rxtx_ptp1588* 示例相同。

6. 运行 IEEE1588 示例

本节介绍如何使用 i.MX RT10xx EVK 板和上述各节中所述的演示软件来设置 1588 演示。

6.1. 硬件设定

必须使用两块 i.MX RT10xx EVK 板并将它们相互连接以设置测试硬件。该演示具有点对点配置，其中两块板使用交叉以太网电缆直接连接。该演示使用一种简单的连接类型，通常用于评估系统的准确性和总体性能。使用两个 i.MX RT10xx EVK 板的点对点配置如图 9 所示。测试不需要特定的跳线设置。

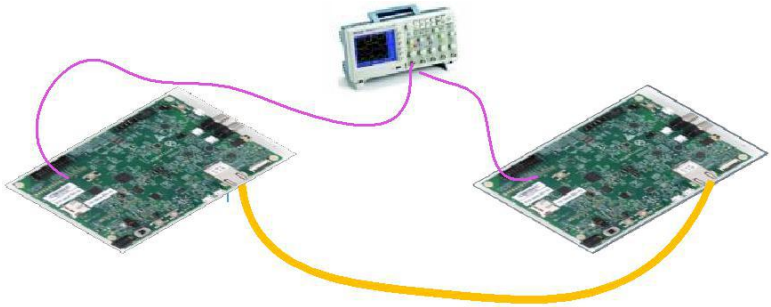

图 9. 示例的背对背配置

有关如何使用 i.MX RT10xx EVK 板和设置其跳线的详细信息，请参阅《 MIMXRT10xx EVK 板硬件用户指南》。

6.2. 时钟同步测量

该演示可以生成每秒脉冲（PPS）信号，以测量主机和从机之间时钟的同步性。 对于 i.MX RT1050 和 i.MX RT1060 MCU，PPS 信号直接从 1588 定时器的通道 3 生成，并配置为通过 GPIO_AD_B1_02 引脚输出。这个 GPIO 信号被路由到 Arduino 接口的 J22-7 引脚。对于 i.MX RT1020 MCU，PPS 信号直接从 1588 计时器的通道 2 生成，并配置为通过 GPIO_SD_B1_02 引脚输出。该 GPIO 信号被路由到 Arduino 接口的 J19-10 引脚。

要测量和比较来自两块板的 PPS 信号，请将两个示波器探头分别连接到 i.MX RT1050 和/或 i.MX RT1060 EVK 板上的 J22-7 引脚，和/或 i.MX RT1020 EVK 板上的 J19-10 引脚。在测试中，为两个板供电一段时间。示波器显示从 PPS 信号越来越靠近主 PPS 信号，并且偏移收敛到 1588 计时器的四个时钟周期之间变化。

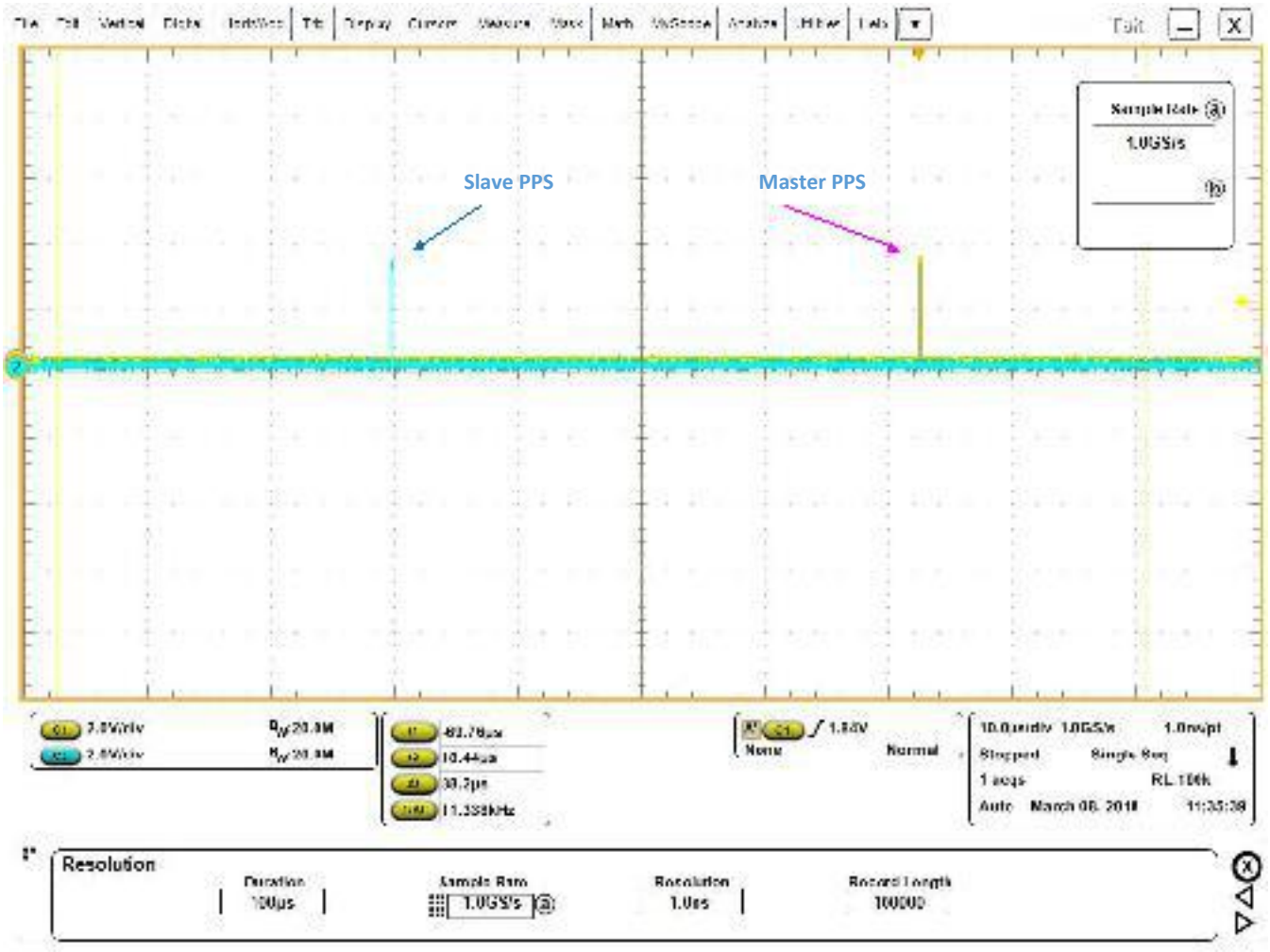

图 10. PTP 启动-主从之间的 PPS 距离

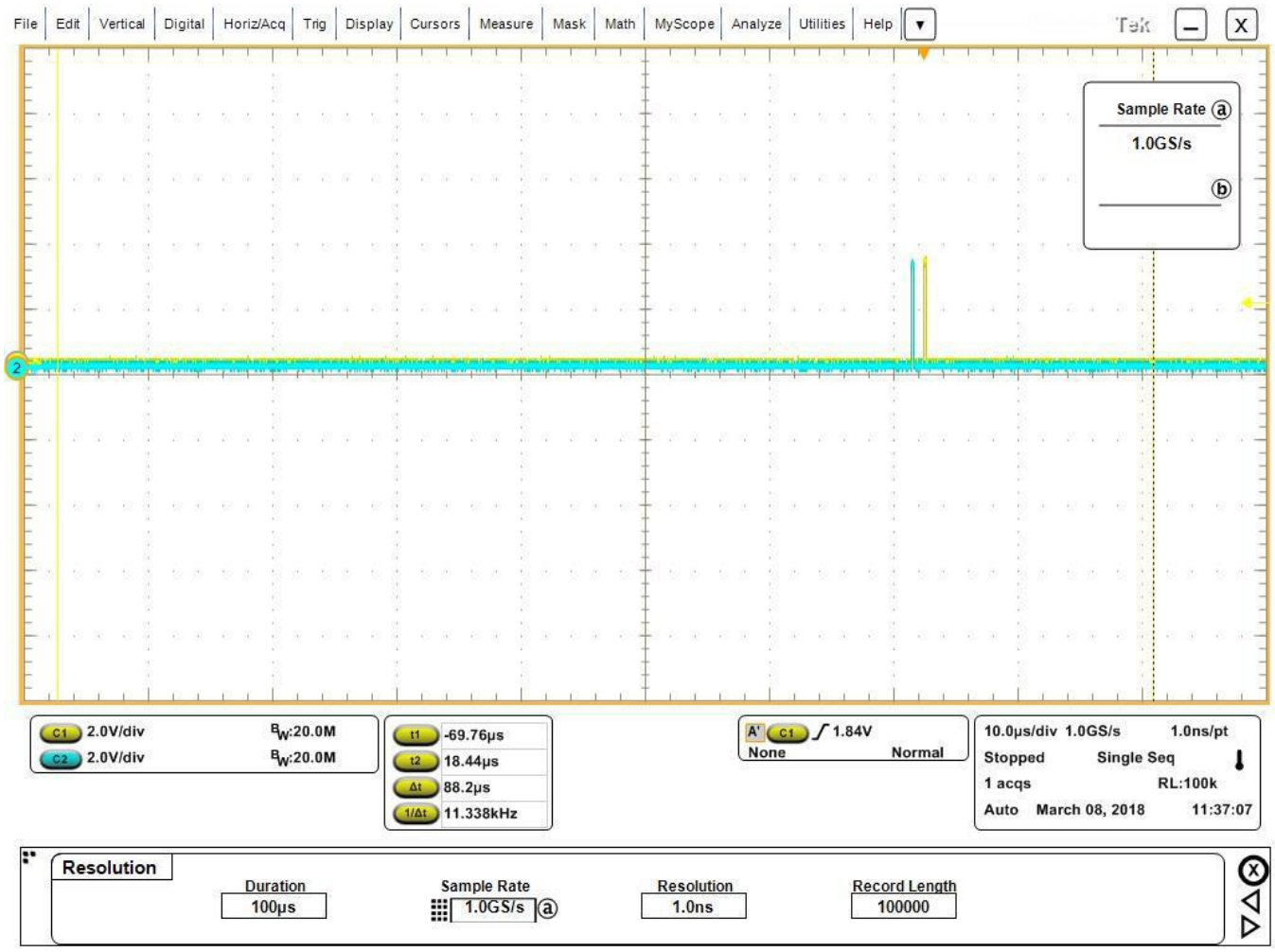

图 11. PTP 同步-从属 PPS 靠近主 PPS

图 12. PTP 同步-主站和从站之间的聚合 PPS 距离

7. 总结

本应用笔记描述了基于开源 PTP 守护程序，FreeRTOS OS，lwIP TCP / IP 协议栈，i.MX RT10xx 的 SDK 和 i.MX RT10xx 评估套件（EVK-MIMXRT1050）板上的 IEEE 1588 协议。 借助 FreeRTOS OS，lwIP 和 TCP / IP 协议栈支持，这个演示也可以从 i.MX RT 系列移植到其他处理器。

演示系统要求设备之间时钟精确同步且面向精度在亚微秒范围内的应用。

8. 缩略语

表 1. 缩写词

术语	含义
API	应用程序接口
BMC	最佳主时钟
DHCP	动态主机控制协议
DNS	域名系统
ENET	10/100 Mbit/s 以太网 MAC
GPIO	通用端口输入输出
ICMP	（Internet Control Message Protocol）因特网控制消息协议
IGMP	（Internet Group Management Protocol）互联网组管理协议
MAC	媒体访问控制
PPS	每秒脉冲
PTP	精确时间协议
PTPd	PTP 守护程序
RTOS	实时操作系统
SDK	软件开发工具包
TCP	传输控制协议
UDP	UDP 用户数据报协议

9. 修订史

表 2 总结了自初始发行以来对该文档所做的更改。

表 2. 修订史

版本号	日期	实质性变化
0	03/2018	初始发行
1	09/2018	将代码更新为 SDK2.4.x。 添加了对 RT1050，RT1060 和 RT1020。

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12149
Rev. 1
09/2018