**TELKOMNIKA**, Vol. 11, No. 9, September 2013, pp. 5105~5111
ISSN: 2302-4046



# A Hardware Time Stamping Method for PTP Messages Based on Linux System

**Zhi Li*[1, 2, a], Zhenlin Zhong[1, b], Wangchun Zhu[1], Binyi Qin[1]**
[1]School of Electronic Engineering and Automation, Guilin University of Electronic Technology, Guilin, 541004, China
[2]Guilin University of Aerospace Technology, Guilin 541004, China
*Corresponding author, e-mail: [a]cclizhi@ guet.edu.cn, [b]forzzlin@163.com

***Abstract***

*In order to achieve high-precision clock synchronization among instrument nodes in distributed LXI bus test systems. A Hardware time stamping method for PTP messages is introduced in this paper. It is implemented in Linux systems, which runs on ARM9 S3C2440 processor. The special PHYTER DP83640 is used to mark the time of PTP messages in physical layer. The new socket option SO_TIMESTAMPING is used to obtain the hardware timestamps in user space programming. A clock synchronization test has been done after transplanting the open source protocol software PTPd on this platform. The results indicate that this method is feasible. It is able to achieve nanosecond level clock synchronization accuracy.*

***Keywords:*** *PTP, IEEE 1588, SO_TIMESTAMPING, hardware timestamp, synchronization*



## 1. Introduction

The IEEE1588 Precision Time Protocol (PTP) is a key technology of the LXI standard. It must be implemented in the class A and class B instruments. It makes the instruments in the Local Area Network (LAN) to achieve clock synchronization through network communication, so as to coordinate the instruments to work by using IEEE 1588 Time-Based triggers. It plays an important role in distributed automatic test system. Obtaining the timestamps of PTP messages is the precondition to implement IEEE1588 clock synchronization algorithm. The protocol stipulates that slave clocks exchange messages with the master clock via the LAN. The transmit timestamps and receive timestamps is used to calculate the time offset from the master by slaves, then it adjusts the local time and the clock rate to achieve clock synchronization with the master. There are two ways to obtain timestamps for the PTP messages: one is to use software, it can achieve microsecond-level clock synchronization; the other is special hardware, it can eliminate the uncertainty delay and jitter caused by PTP messages passing through the network protocol stack to achieve nanosecond-level clock synchronization [1].

This paper provides a method to obtain PTP messages hardware timestamps base on embedded Linux operating system. It will be introduced in three sections including hardware design, network device driver and a standard Application Programming Interface (API) for user space programs. This method assists the implementation of PTP with high precision clock synchronization.

## 2. IEEE1588 Clock Synchronization Principle

In the PTP system, the most stable and accurate clock is selected as the master clock by using the Best Master Clock (BMC) algorithm. It provides reference time for one or more slave clocks. Slave clocks can obtain four valid timestamps by exchanging messages with the master through network. It calculates the time offset from master clock using these four timestamps, then adjusts the local clock base on this time offset to achieve clock synchronization. The precision of clock synchronization is influenced by the precision of timestamps [2-3].

*Received March 27, 2013; Revised June 4, 2013; Accepted June 17, 2013*

Figure 1 shows the procedure of clock synchronization. The master sends *Sync* message by period T (T=2^n s, typically 2 s) and records the send time (tm1). If the *Flow_Up* message is enabled, a *Flow_Up* message containing tm1 will be sent at once after sending *Sync* message. Slaves record the receipt time of *Sync* message (ts1) and extract the tm1 from *Flow_Up* message or Sync message (when *Flow_Up* is disabled). The propagation delay of message from master to slave is:

$$dm2s = ts1 - tm1 \quad (1)$$

Slaves send *Delay_Req* message intermittently and record the send time (ts2), and the master records the receipt time (tm2). The master will send a *Delay_Resp* message containing tm2 to slaves. The propagation delay of message from slave to master is:

$$ds2m = ts2 - tm2 \quad (2)$$

Assuming the path is symmetric in two directions, and then the message propagation delay is:

$$dprop = (dm2s + ds2m) / 2 \quad (3)$$

So the time offset between slave and master is:

$$offset = dm2s - dprop \quad (4)$$

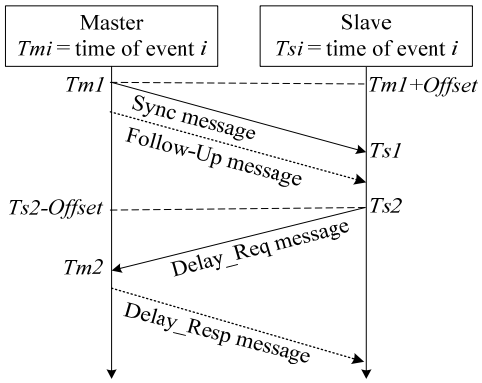


Figure 1. Procedure of Clock Synchronization

## 3. Hardware Design

### 3.1. Hardware System Block Diagram

The Hardware System Block Diagram is shown in Figure 2. It mainly consists of three parts: a processor, a MAC controller and a PHY. SAMSUNG's S3C2440 is a high performance 16/32-bit RISC microprocessor which developed with ARM920T core, it supports Linux system well. The DM9000 is a cost-effective single chip Fast Ethernet MAC controller with a general processor interface to connect the processor, and provides a MII interface to connect the external PHY. DP83640 is a precision PHYTER which designed to achieve high-precision clock synchronization supporting IEEE 1588 V1 and V2. It supports 10/100 Mb/s network and provides RMII and MII interface to connect with the MAC.

### 3.2. IEEE 1588 Clock Source

DP83640 clock input pins must be connected to a 25 MHz 0.005% (±50ppm) clock source when MII interface mode is used. An internal Phase Generation Module (PGM) inside the DP83640 can generate a nominal 125 MHz reference clock which is provided for the IEEE 1588 PTP logic operation, so as to get 8ns timestamp resolution [4]. The IEEE 1588 clock

supports directly read/writable and frequency scalable. It is convenient for local clock to implement the precise synchronization with the master clock.

### 3.3. Hardware Timestamp Generation

PTPv1 messages include *Sync*, *Follow_Up*, *Delay_Req*, *Delay_Resp* and *Management*. They are divided into two types: *Sync* and *Delay_Req* are event messages; *Follow_Up*, *Delay_Resp* and *Management* are general messages. The IEEE 1588 protocol stipulates only event messages need to be time stamped. As the Figure 3 shows, there is a packet parser in both the send end and the receipt end inside the DP83640. It is able to identify the incoming and outgoing PTP event messages and generates timestamps automatically. There are two ways for software to obtain the hardware timestamps:

(1) Reading timestamps from DP83640 registers.

Whether the timestamp is available or not, it can be determined by reading the PTP Status Register (PTP_STS). The transmit timestamps are provided in the PTP Transmit Timestamp Register (PTP_TXTS) and the receive timestamps are provided in the PTP Receive Timestamp Register (PTP_RXTS).

(2) Getting timestamps through parsing the Status Frame.

If the PHY Status Frame is enabled, DP83640 will generate a Status Frame when it detects an incoming or outgoing PTP messages. The Status Frame contains transmit timestamp and receive timestamp, as well as other status messages. It is delivered to the MAC via MII interface.

Reading timestamps from registers need to continue to query the PTP_STS register, so getting timestamps through Status Frame has higher efficiency.

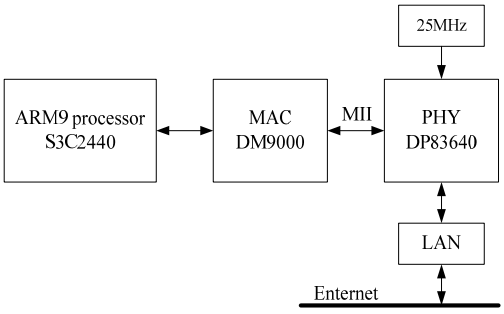


Figure 2. Hardware System Block Diagram

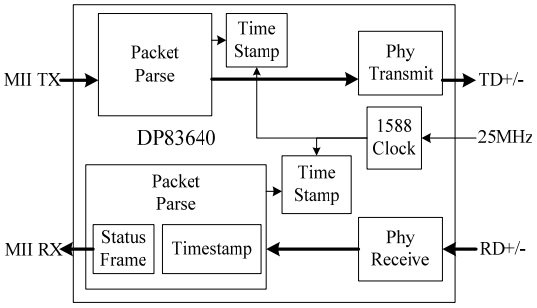


Figure 3. DP83640 Internal Block Diagram [5]

## 4. Network Device Driver

In operating system, the device drivers hide the underlying hardware implementation details for upper applications. The applications only need to call the standard Application Programming Interface (API) to control the hardware. The network device drivers mainly include a DM9000 driver and a DP83640 driver. This paper aims to modify the existing network device drivers to implement hardware time stamping for PTP messages. These modified drivers will be ported to the embedded Linux system. It provides a standard API for the IEEE 1588 applications to obtain the hardware timestamps. Figure 4 shows the hardware time stamping implementation in the Linux system. The Linux kernel should be 2.6.30 or later.

a. The receiving process

The receiving message type is identified in function *dm9000 _rx()*. If it is an ordinary message, it will be delivered to upper network stack directly through function *netif_rx()*. If it is a PTP event message, firstly the *skb* (a *sk_buff* type data structure) will be added to a receive queue, then it waits for a corresponding Status Frame generated by DP83640, when the Status

Frame arrives, the receive hardware timestamp will be extracted and inserted to the responding *skb* in the receive queue, finally the *skb* will be delivered to the upper layer network stack.

b. The transmitting process

In function *dm9000_start_xmit()*, the transmitting message type will be identified. If it is a PTP event message, firstly the *skb* will be cloned, and the cloned *skb* is added to a transmit queue, then it waits for a corresponding Status Frame generated by DP83640, when the Status Frame arrives, the transmit hardware timestamp will be extracted and inserted into the responding *skb* in the transmit queue, finally the *skb* will be added to the socket's error queue.

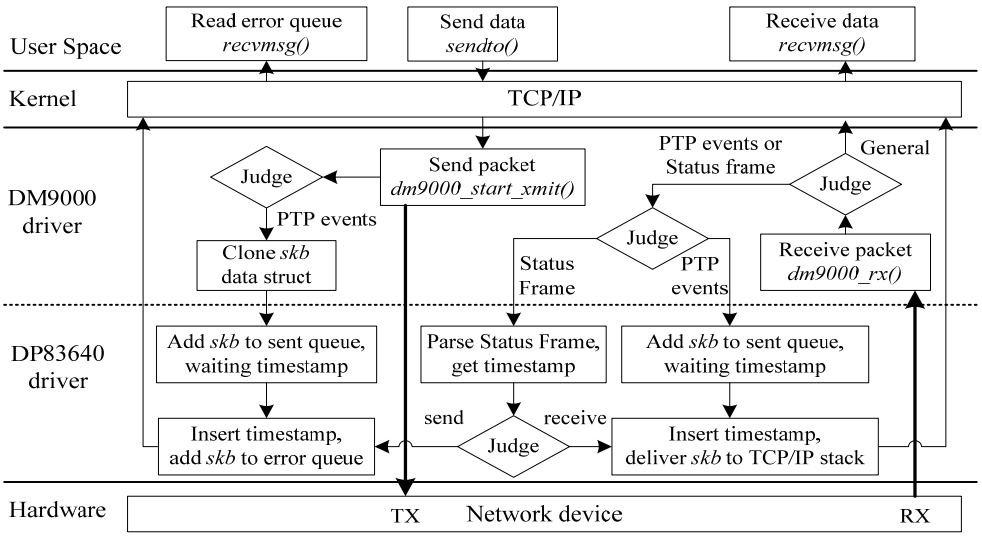


Figure 4. Hardware Time Stamping Implementation in Linux System

In order to implement hardware time stamping for PTP event messages, modification of the network device drivers include the following aspects:

(1) Establishing connection between the DM9000 and DP83640.

The original DM9000 driver operates the internal PHY by default, if it wants to operate the external PHY DP83640, the Network Control Register (NCR) bit 7 needs to be set. Because of DM9000 accesses the DP83640 via the MII interface, the driver is designed using the idea that separating the host and peripheral. Such advantages are the DM9000 driver implementation does not need to care about the MII peripherals and the same as the DP83640 driver implementation to the specific hosts. They communicate to each other through the kernel API, the hosts and the peripherals can be combined randomly. That improves the portability of the drivers. The following steps should be done:

Firstly, a MII bus instance needs to be registered in the DM9000 driver to establish connection between DM9000 and the MII bus, codes as follow:

*struct mii_bus *smi_bus;*
*mdiobus_register(smi_bus);*

Secondly, DP83640 needs to be registered as a PHY device of the MII bus:

*mdiobus_scan(smi_bus, addr);//addr is the PHY address*

Finally, the connection between dm9000 and dp83640 is established:

*phy_attach(db->ndev,dev_name(&phy->dev),0, PHY_INTERFACE_MODE_MII);*

(2) The *SIOCSHWTSTAMP* command needs to be supported by the MAC driver.

It makes users are able to initialize network device using *ioctl(SIOCSHWTSTAMP)*, the initialization includes determining whether the device is expected to hardware time stamping, and specifying what kind of messages will be filtered. *phy_mii_ioctl()* is a kernel MII control

function which contains the implementation details of the *SIOCSHWTSTAMP* command, it can be used in the *dm9000_ioctl()* function.

(3) Hardware time stamping for outgoing PTP event messages.

The function *skb_tx_timestamp()* should be called in the function *dm9000_start_xmit()* as soon as possible after giving the *skb* to the MAC hardware, but before freeing the *skb*. It is a system function supported by Linux 2.6.30 or later, if the outgoing packet is a PTP event message, it will process the packet correspondingly by calling the DP 83640 driver function *txtstamp()*.

(4) Hardware time stamping for incoming PTP event messages.

The function *skb_defer_rx_timestamp()* should be called in the function *dm9000_rx()* before delivering *skb* to the upper layer. It is a system function supported by Linux 2.6.30 or later, if the incoming packet is a general message, it returns false, otherwise, it processes the packet correspondingly by calling the dp83640 driver function *rxtstamp()*.

## 5. User Space API

Linux 2.6.30 or later versions provide a new API for the user space to get the network packets time stamps. In addition to the existing socket options *SO_TIMESTAMP* and *SO_TIMESTAMPNS*, it increases the *SO_TIMESTAMPING*. Both *SO_TIMESTAMP* and *SO_TIMESTAMPNS* generate timestamp for each incoming packet by using the system time, *SO_TIMESTAMP* returns microseconds resolution, while *SO_TIMESTAMPNS* returns nanoseconds resolution. Using *SO_TIMESTAMPING* option, time stamps can be generated by software or hardware. Different parameter settings for function *setsockopt()* can achieve different return results. The parameter is an integer with some of the bits set showed in Table 1. Setting other bits is an error and doesn't change the current state [6-8].

Table 1. Parameter Settings of *SO_TIMESTAMPING*

| Parameter | Description |
|---|---|
| *SOF_TIMESTAMPING_TX_HARDWARE* | try to obtain send time stamp in hardware |
| *SOF_TIMESTAMPING_TX_SOFTWARE* | if *SOF_TIMESTAMPING_TX_HARDWARE* is off or fails, then do it in software |
| *SOF_TIMESTAMPING_RX_HARDWARE* | return the original, unmodified time stamp as generated by the hardware |
| *SOF_TIMESTAMPING_RX_SOFTWARE* | if *SOF_TIMESTAMPING_RX_HARDWARE* is off or fails, then do it in software |
| *SOF_TIMESTAMPING_RAW_HARDWARE* | return original raw hardware time stamp |
| *SOF_TIMESTAMPING_SYS_HARDWARE* | return hardware time stamp transformed to the system time base |
| *SOF_TIMESTAMPING_SOFTWARE* | return system time stamp generated in software |

*SOF_TIMESTAMPING_TX/RX* determines how time stamps are generated.
*SOF_TIMESTAMPING_RAW/SYS* determines how they are reported.

When the device receives a PTP event message, the receive timestamp as an ancillary data which is *SO_TIMESTAMPING* type in *SOL_SOCKET* protocol level storing in the control message. The function *recvmsg()* can be used to obtain network data and the control message. All the ancillary data can be accessed using these macros: *CMSG_FIRSTHDR()*, *CMSG_NXTHDR()* and *CMSG_DATA()*. It indicates the ancillary data of *SO_TIMESTAMPING* type is found when the two equations *cmsg_level==SOL_SOCK* and *cmsg_type== SO_TIMESTAMPING* are satisfied. Part of the implementation codes as follow:

*recvmsg(sock, &msg, MSG_DONTWAIT); // receive data and get the control message*

```
for (cmsg = CMSG_FIRSTHDR(&msg); cmsg != NULL; cmsg = CMSG_NXTHDR(&msg, cmsg)) {
    struct timespec *stamp;  // point to the time stamp
    if (cmsg->cmsg_level != SOL_SOCKET)
        continue;
    switch (cmsg->cmsg_type) {
    case SO_TIMESTAMPING:
    stamp = (struct timespec*)CMSG_DATA(cmsg); // timestamp is found
    ……
     }
……
 }
```

For transmit timestamp, the outgoing packet is looped back to the socket's error queue with the transmit timestamp attached. It can be received with *recvmsg(flags=MSG_ERRQUEUE)*.

## 6. Experiment and result

The open source PTP daemon (PTPd) is modified and ported to Linux system based on the API described above. PTPd is designed as a pure software solution to implement the IEEE 1588 Precision Time Protocol. It can achieve microsecond-level clock synchronization accuracy. It is improved to obtain higher accuracy [9]. About three hours' clock synchronization test had been done by using a self-designed device connect with an Agilent E5818A LXI Class-B trigger box via a ten meters twisted pair directly. The E5818A is used as the master clock, and the self-designed device is the slave clock. The test data is processed by MATLAB shown in the Figure 5, it can be seen that after a period of clock synchronization, the time offset between the slave clock and the master clock is stable within nearly ±100ns.

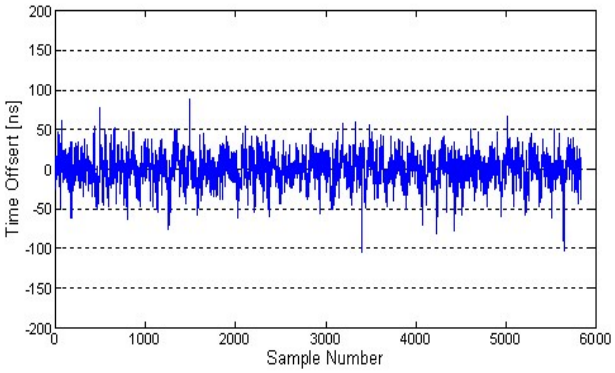


Figure 5. Clock Synchronization Test

## 7. Conclusion

The results indicated the method described above correctly. Nanosecond resolution for clock synchronization is achieved using hardware timestamp to implement the IEEE 1588 protocol. Linux is becoming a leading embedded operating system because of its excellent performances. It is widely used in the areas like measurement, control and automation. This paper details a method to obtain the hardware timestamp of PTP message base on ARM and Linux platform. It provides a great convenience for the IEEE 1588 Precision Time Protocol applications to achieve high-precision clock synchronization.

**Acknowledgements**

This research was supported by Guangxi Natural Science Foundation of China (Grant No. 2013GXNSFAA019332).

**References**

[1] Park Jae Won, Hwang Jin Ha, Chung Won Young. Design time stamp hardware unit supporting IEEE 1588 standard. 2011 International SoC Design Conference, ISOCC 2011: 345-348.
[2] IEEE Instrumentation and Measurement Society: IEEE Std 1588 -2002.
[3] K Correll, N Barendt, M Branicky. *Design considerations for software only implementations of the IEEE 1588 precision time protocol.* Proceedings of the IEEE 1588 Conference, Zurich, Switzerland, 2005.
[4] National Semiconductor Corporation. Precision PHYTER-IEEE 1588 Precision Time Protocol Transceiver. 2010.
[5] National Semiconductor Corporation. National Semiconductor Ethernet PHYTER Software Development Guide. 2009.
[6] Patrick Ohly. net: new user space API for time stamping of incoming and outgoing packets. https://lkml.org/lkml/2008/12/15/145, 2008.
[7] Richard Cochran, Cristian Marinescu. Design and Implementation of a PTP Clock Infrastructure for the Linux Kernel. ISPCS 2010, United States: IEEE Computer Society. 2010: 116-121.
[8] Richard Cochran, Cristian Marinescu, Christian Riesch. Synchronizing the linux system time to a PTP hardware clock. *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication*. ISPCS 2011: 87-92.
[9] Wang Te-Kwei, Chang Fan-Ren. *Application of PTPd to find the threshold of precision time synchronization*. Proceedings 2011 International Conference on System Science and Engineering, ICSSE 2011: 161-164.
[10] Liu Ying, Zhu Yantao, Li Yurong. The embedded information acquisition system of forest resource. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(6): 1843-1848.
[11] Feng Tianrui, Sen Ouyang. Design of a distribution network power quality monitoring system based on metering automation systems and its application. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(7): 1547-1553.
[12] Yu Wilson. LXI instrument development platform based on an open embedded operating system. *Yi Qi Yi Biao Xue Bao/Chinese Journal of Scientific Instrumen*. 2007; 28(5): 788-791.