# Synchronizing Device Clocks Using IEEE 1588 and Blackfin Embedded Processors

By Jiang Wu and Robert Peloquin

### Introduction

The IEEE 1588 standard, introduced in 2002, defines a protocol to synchronize distributed clocks on a network. It is becoming the preferred clock synchronization method for many different applications, including test and measurement, telecommunications, and multimedia streaming. This standardized method for synchronizing clocks is cost-effective, supports heterogeneous systems, and provides nanosecond-level synchronization precision.

This article provides an introduction to both the original IEEE 1588-2002 standard and the enhancements incorporated as part of the updated IEEE 1588-2008 version. Dedicated hardware support for IEEE 1588 has been integrated into the ADSP-BF518[1] Blackfin® embedded processor because of the increasing importance of IEEE 1588 in some of its targeted applications. An overview of its capabilities is provided, followed by an example showing clock synchronization performance results obtained by an ADSP-BF518 processor solution.

### What Time Is It?

It is common for a system to need to maintain its own sense of time using a local oscillator. Figure 1 shows how hardware and software combine to generate time information within a system.

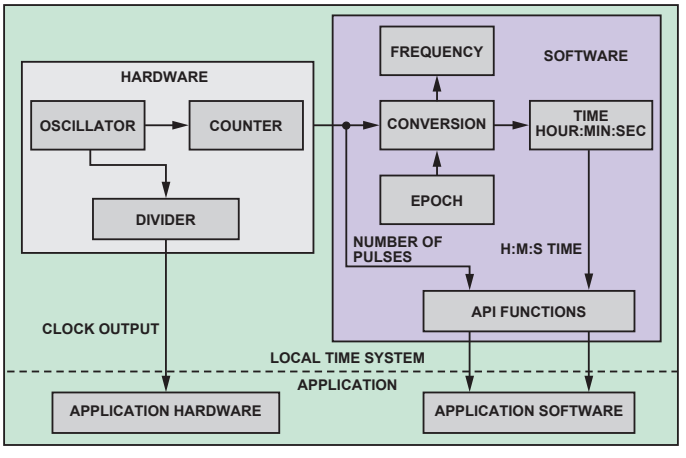


Figure 1. Local timekeeping.

This time information can be used by both hardware and software resources within the system. In hardware, one or more physical clock signals (clock outputs) are derived from the oscillator's clock and can be used to drive or trigger other parts of the system. The time maintained in software is typically referred to as *system time*. The system time can be represented in the form of numbers of clock pulses or in second/nanosecond notation. The system software derives the time from the number of oscillator clock pulses and its frequency information, and provides *application-programming-interface* (API) functions that other parts of the software use to retrieve and set the time. If an absolute time is desired, the provided time is associated with a predefined epoch, which identifies a reference point in time.

### Synchronize Your Watches

Many applications require two independent devices to operate in a synchronized fashion. If each device were to rely solely on its own oscillator, differences between the specific characteristics and operating conditions of the individual oscillators would limit the ability of the clocks to operate synchronously. Some possible simplistic solutions to address these limitations include:

- *All the devices could use a single physical oscillator.* This is only feasible for distributed systems in close proximity; a high-frequency clock signal cannot be reliably delivered over a long distance.
- *All the devices could utilize oscillators with nearly identical characteristics*. This approach is impractical due to the difficulty of acquiring nearly identical oscillators and keeping them from drifting apart over time. More importantly, each oscillator will be subjected to different operating conditions.
- *If all the devices are interconnected via a communications network such as Ethernet, they can dynamically adjust their individual clocks to a single "master" clock by exchanging time messages over the network*. With *network time protocol* (NTP), the traditional time synchronization protocol, every device in the system adjusts its clock according to the time information it retrieves from an NTP time server. However, this protocol can only achieve synchronization accuracy on the order of milliseconds.

IEEE 1588 defines a newer protocol capable of nanosecond synchronization accuracy. How it can achieve this level of clock synchronization is discussed in the following sections.

### What IEEE 1588 Does

The IEEE 1588 standard defines a protocol for time-synchronizing devices that are geographically dispersed but interconnected by some form of communications technology, for example, Ethernet. By exchanging timing messages between devices they can maintain the same absolute system time, which is represented in seconds and nanoseconds.

An intuitive way to achieve this goal is for one device, which has the "best" (most accurate) clock, and is designated as the *master-clock* device, to broadcast its time to the other devices. The other devices will adjust their times to match the time sent by the master clock. This solution has several limitations, though:

1. The master-clock device cannot broadcast the time at infinitesimal intervals, so the "slave" clock devices have to use their own independent and "inferior" oscillators to interpolate the time points between two broadcasts from the master-clock device. This results in degraded synchronization during the time between updates from the master clock.
2. Delays inevitably exist on the broadcast path, with magnitudes depending on the communications technology—the time that a physical signal takes to travel along a wire from one device to another, for example. This delay results in an additional offset between the master clock and each slave clock.
3. Differences among the broadcast paths between the master-clock device and each slave-clock device will further degrade the synchronization between individual slave-clock devices.

IEEE 1588 specifies a protocol that solves the second and third problems by measuring path delay. It also allows the slave clock to be adjusted to match the master clock's pace so as to mitigate the first problem. Where possible, the first problem can be further reduced by using smaller broadcasting intervals and higher-quality oscillators.

### How IEEE 1588 Measures Communication Delay

IEEE 1588-2002[2] defines four messages to measure the communication delay of the forward (*master to slave*) and backward (*slave to master*) paths: *Sync*, *Followup*, *DelayReq*, and *DelayResp*. The newer version, IEEE 1588-2008,[3] provides further mechanisms to measure the *peer-to-peer delay* with three additional messages: *PdelayReq*, *PdelayResp*, and *PdelayRespFollowup*.

Among these messages, Sync, DelayReq, PdelayReq, and PdelayResp, so-called *event* messages, must be *time-stamped* (recording the local time) when they leave and arrive at a device. There are two techniques to time-stamp packets:

1. *Software time-stamp* occurs when the messages are handled by the software. Usually occurring in the message's receive/transmit *interrupt service routine* (ISR), the time-stamp is the current value of the system time.
2. *Hardware time-stamp* occurs when the messages physically arrive at or leave the device. The time-stamp operation is executed by hardware, which maintains its own continuous time information.

Either time-stamp method is acceptable in IEEE 1588, but a hardware time-stamp can provide significantly better precision, as will be shown below.

### Delay from Master-Clock Device to Slave-Clock Device

The messages Sync and Followup are sent by the master-clock device; it is a slave-clock device's responsibility to receive them and calculate the communication path delay from the master-clock device to the slave-clock device.

In Figure 2, at time Tm1, the master-clock device software reads the current local system time (Tm1, the software time-stamp), inserts it into a Sync message, and sends the message out. The message leaves the master-clock device at a later time, Tm1', which is the hardware time-stamp. It arrives at slave-clock hardware at Ts1' (slave-clock device local time), and is received by the slave-clock device software at a later time, Ts1. The software will read the hardware time-stamp to get Ts1'. If there is no communication delay, Ts1' should be equal to (Tm1' + Tms), where Tms is the time difference between master clock and slave clock. The protocol's ultimate goal is to compensate for this difference.

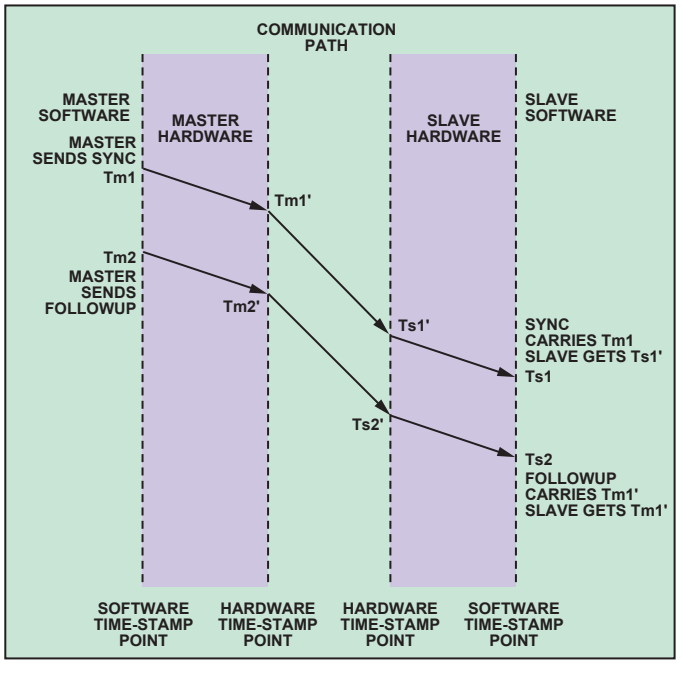


Figure 2. Measuring communication delay between master-clock and slave-clock devices.

After the Sync message has been sent, the master-clock device software reads the Sync message's departure time, Tm1', through the time-stamping unit, inserts it into a Followup message, and sends that message out at Tm2. This message is received by slave-clock device software at Ts2. At this point, the slave-clock device software has the two times, Ts1' (Sync arrival time) and Tm1' (Sync departure time). The master-to-slave path delay, Tmsd, is determined by Equation 1.

$$Tmsd = (Ts1' + Tms) - Tm1' \quad (1)$$

### Delay from Slave-Clock Device to Master-Clock Device

The DelayReq message is sent by the slave-clock devices, and the DelayResp message is sent by the master-clock device in response. With these messages, the slave-clock devices can calculate the communication path delay from the slave-clock device to the master-clock device.

At time Ts3 (Figure 3), the slave-clock device software reads the current local system time (Ts3), inserts it into a DelayReq message, and sends the message out. After the message is sent, the slave-clock device software reads the time-stamp to get the departure time of the message, Ts3', and waits for the response from the master-clock device.

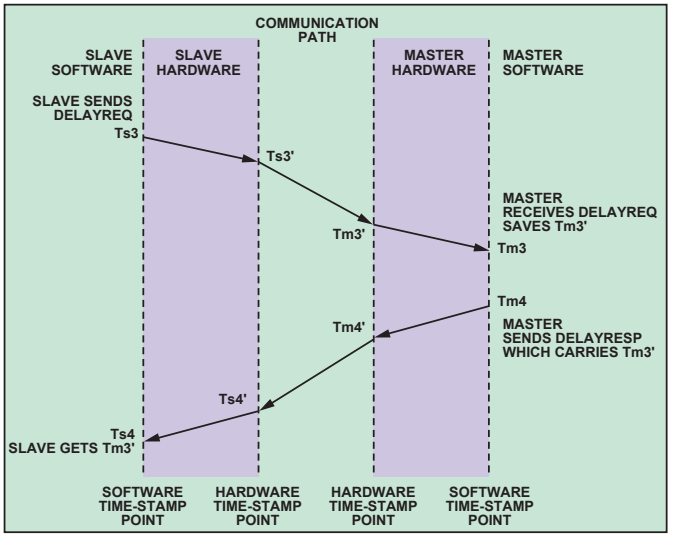


Figure 3. Measuring slave-master communication delay.

The DelayReq message arrives at the master-clock device at a later time, Tm3', and is processed by the master software at Tm3. The software then reads the time-stamp to get the arrival time, Tm3', puts it into the DelayResp message, and sends to a slave-clock device at Tm4. When the slave-clock device software receives the DelayResp message at Ts4, it can extract the time, Tm3', and calculate the slave-to-master delay, Tsmd, by Equation 2.

$$Tsmd = Tm3' - (Ts3' + Tms) \quad (2)$$

In both Equation 1 and Equation 2, there is an unknown variable, the master-slave time difference, Tms. So it is not possible to get either Tmsd or Tsmd individually. However, if one makes the usually acceptable assumption that the communication path is symmetric

$$Tmsd = Tsmd = Td \quad (3)$$

—a key assumption for IEEE 1588 to work correctly—then, adding Equation 1 and Equation 2 gives

$$Td = \frac{1}{2}[(Ts1' - Tm1') + (Tm3' - Ts3')] \quad (4)$$

All these calculations are performed by the slave-clock devices, since it is they who seek to synchronize themselves to the master-clock device. They get Tm1' from master-clock device's Followup message, Ts1' from their Rx (reception) time-stamping, Ts3' from their Tx (transmission) time-stamping, and Tm3' from the master-clock device's DelayResp message.

### How to Calculate the Time Difference Between a Slave Clock and Master Clock

Once the communication path delay, Td, is obtained, the slave-master time difference is easy to calculate, using either Equation 1 or Equation 2, as shown in Equation 5 and Equation 6.

$$Tms = Td - (Ts1' - Tm1') \quad (5)$$

$$Tms = (Tm3' - Ts3') - Td \quad (6)$$

### How to Adjust the Time of a Slave-Clock Device

With the time difference from the master clock known, each slave-clock device needs to adjust its own local time to match the master clock. This task has two aspects. First, slave-clock devices need to adjust their absolute time by adding the time difference to make their time perfectly match the master-clock time at this moment. Then, each slave-clock device needs to adjust its clock frequency to match the frequency of the master clock. We cannot rely on the absolute time alone, since the time difference is applied only at a certain period and could be either positive or negative; as a result, the adjustment will make the slave-clock time jumpy or even run backward. So, in practice, the adjustment takes two steps.

1. If the time difference is too big, for example, larger than one second, absolute time adjustment is applied.
2. If the time difference is small, a percentage change of frequency is applied to slave clocks.

Generally speaking, the system becomes a control loop, where master-clock time is the reference command, slave-clock time is the output tracking the master-clock time, and their difference drives the adjustable clock. PID control, which is commonly used by many IEEE 1588 implementations, could be used to achieve specific tracking performance. Figure 4 illustrates this control loop.

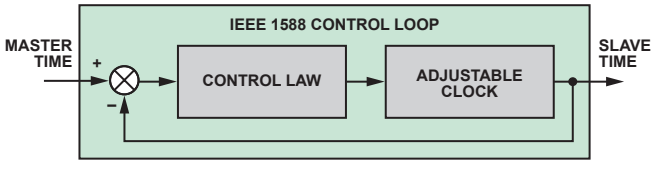


Figure 4. IEEE 1588 control loop.

### Peer-to-Peer Delay

The revised version, IEEE 1588-2008, introduces a new mechanism for measuring path delay, called *peer-to-peer* (P2P) delay. By contrast, the master-slave mechanism discussed in the previous sections is *end-to-end* (E2E) delay. In an IEEE 1588-2008-capable network, a master-clock device can be linked to slave-clock devices either directly or through multiple hops (stages). The E2E delay is actually the *total* delay from a master-clock device to a slave-clock device, including all the hops in between. However, the P2P delay is limited to two directly connected devices. The overall delay along the path is the sum of the P2P delay of all the hops. From the perspective of preserving path symmetry, the P2P mechanism provides better accuracy.

As noted earlier, IEEE 1588-2008 includes three additional messages, PdelayReq, PdelayResp, and PdelayRespFollowup, to measure P2P delay. They work in a manner similar to that explained above. Reference 3 provides more details.

### Factors Affecting Synchronization Performance

Well-designed IEEE 1588 devices are capable of achieving highly accurate clock synchronization, but it is important to recognize the key factors that directly affect performance. Some of these include:

1. **Path delay**: As noted earlier, the path delay measurement of IEEE 1588 assumes that the communication-path delays are symmetrical, that is, the transmission delay of the forward path is equal to the reverse transmission delay. In addition, the delay should not vary during the delay measurement. Variation in delay during measurement will produce asymmetry and delay jitter, which will have a direct impact on the synchronization precision. While delay symmetry and jitter cannot be controlled *outside* the boundaries of an IEEE 1588 device, both path symmetry and jitter can be improved *within* the device if measurements are based on *hardware* time-stamping. Hardware time-stamping eliminates the significant jitter resulting from software time-stamping—due to interrupt latency, context switch, and thread scheduling.
2. **Drift and jitter characteristics of clocks**: The frequency and phase of the master clock represent the inputs of the tracking control system, and the slave clock is the control object. Any time-varying behavior of the master clock will act as a disturbance to the control system and result in both steady-state- and transient errors. Clocks with less drift and jitter will, therefore, improve synchronization accuracy.
3. **Control law**: The control method determines how the errors in the slave-clock-device time are corrected in the adjustment of the slave clock. The control-law parameters, including settling time, overshoot, and steady-state error, will directly affect clock synchronization performance.
4. **Resolution of the clocks**: As shown in Figure 1, the resolution of the local time is determined by the frequency of the clock; the minimum increment of time is one period of the clock signal. The IEEE 1588 protocol runs on a time with a resolution of 1 ns for IEEE 1588-2002 and $2^{-16}$ ns for IEEE 1588-2008. It is not practical to have a clock of $2^{16}$ (!) GHz (or even 1 GHz). The quantization of the local clocks is expected to affect the precision of local time measurement and control.
5. **How often Sync messages are issued**: The frequency with which the slave clocks are updated ultimately affects the precision of synchronization. A longer period usually leads to larger time errors observed at the next Sync, since the time error is the integral accumulation of the slave-clock frequency error.
6. **How often delay measurement is conducted**: Delay measurement is performed periodically, at intervals based on the expectation that the delay does not change significantly between adjacent samples. If the IEEE 1588 network experiences large delay variations, then increasing the delay-measurement frequency will improve clock-synchronization performance.

### Which Is the Master-Clock?

Having considered how to accurately determine the time difference between master-clock devices and slave-clock devices, a relevant question is how to determine which device, among possibly hundreds of interconnected devices, will serve as the master clock.

IEEE 1588 defines a method called the *best master clock* (BMC) algorithm to choose the master clock device. For this approach, every device of an IEEE 1588 network maintains a data set describing the nature, quality, stability, unique identifier, and preference of its local clock. When a device joins an IEEE 1588 network, it will broadcast the dataset of its own clock and receive the datasets from all other devices. Using the datasets of all the participating devices, every device runs the same BMC algorithm to decide on the master clock and its own future status (master clock or slave clock). Because the same algorithm is executed independently by all the devices on the same data, all will come

to the same conclusion without requiring any negotiation among them. More information about the details of the BMC algorithm can be found in References 2 and 3.

### ADSP-BF518 Processor's Support for IEEE 1588

The Analog Devices ADSP-BF518 processor recently joined ADI's Blackfin DSP family. Like its predecessor, the ADSP-BF537,[4] it has a built-in *Ethernet media-access controller* (EMAC) module. Its capability to support EMAC functionality within the IEEE 1588 standard is extended by an additional *TSYNC* module, as well as extra features to support a wide range of IEEE 1588 applications on Ethernet. Figure 5 shows the block diagram of the TSYNC module. The *ADSP-BF51x Blackfin Processor Hardware Reference* provides additional information.[5]

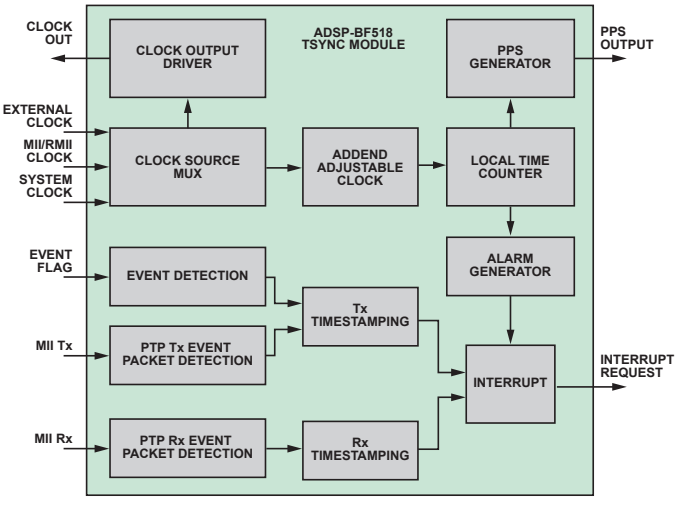


Figure 5. Block diagram of the ADSP-BF518 processor's TSYNC module.

### Packet Detection

The ADSP-BF518 processor can detect and provide hardware time-stamps for all IEEE 1588 event messages, including both incoming and outgoing packets. The precision of an IEEE 1588 system depends significantly on both the accuracy of the event-message time-stamps and on where they are taken, as these affect the requirement for symmetry and constancy of path delay. The ADSP-BF518's TSYNC module keeps monitoring the hardware interface between the MAC controller and the Ethernet *physical interface transceiver* (PHY), that is, the *media independent interface* (MII), and produces a hardware time-stamp whenever it detects an event message—a capability promoting higher synchronization precision with the ADSP-BF518.

The detection of event messages, designed to be programmable, can basically be configured to support either IEEE 1588-2002 (default) or IEEE 1588-2008. Furthermore, this programmability allows for the support of future versions of IEEE 1588, as well as other general protocols that require time-stamping—including being configured to time-stamp every Ethernet packet coming into and out of the processor.

### Flexible Clock Sources

The properties of local clocks are important for the performance of an IEEE 1588 system. To satisfy requirements of a variety of applications, the ADSP-BF518 processor allows three options for the local clock source: system clock, external clock, or Ethernet clock. If the application has a specific clock requirement, it can choose *external clock* and provide a customized clock source. The *Ethernet clock* option can offer good precision if the master-clock devices and the slave-clock devices are connected back-to-back, since the clock is inferred from the Ethernet lines, and the two devices are running on the same clock. A general application can take the *processor's system clock* as its clock source.

The selected source clock is also driven by the TSYNC module as an output of the processor, via the specific pin *Clockout*, to be used by other parts of the system for local time information.

### PPS Output

The *pulse-per-second* (PPS) signal is a physical representation of time information. It is nominally a 1-Hz signal with a pulse at each one-second transition of time. It can be used to control local devices or to provide an auxiliary time channel in case of network failure. It can also be used in testing. The phase difference between PPS signals at two devices is a physical measurement of their time offset.

The ADSP-BF518 processor provides a flexible PPS output. It uses a programmable *start time* (*PPS_ST*) and period (*PPS_P*) to generate a signal with pulses occurring at the times (PPS_ST + $n \times$ PPS_P), where n = 1, 2, 3... In the basic usage, the PPS signal can be created simply by setting PPS_P to 1 second, and PPS_ST to any future instant as a multiple of seconds. This PPS output capability allows for its use as the reference for generation of a periodic signal with a fully programmable frequency and start time.

### Auxiliary Snapshot

Some applications may need to time-stamp a certain event indicated by the toggle of a flag signal. The ADSP-BF518's TSYNC module facilitates this request by providing an auxiliary *snapshot* function, using a dedicated pin to accept an external flag. Toggling the flag will trigger the module to capture the current local time in a time-stamp register for software to access.

### Alarm

If an application needs to execute a task at a specific time, it can make use of the *alarm* feature of the TSYNC module. This feature allows an absolute local time to be set so as to trigger a processor interrupt when the time arrives. The software can then service the interrupt and run the task.

### Adjustable Clock

The adjustable clock of the TSYNC module is an *addend-based* clock. As shown in Figure 6, it takes a fixed input clock signal and outputs a "pulse-steal" version of the input: the value of *addend* is added to the accumulator at each input clock, and each time the accumulator overflows the carry bit drives the *local-time counter*, which gives the local time in terms of the number of pulses counted. The frequency of the local clock can be adjusted by changing the addend, since the addend decides how often the accumulator overflows, and so how often the local-time counter increments. If the frequency of the input clock is $F_{in}$, and the value of addend is $A$, then the local clock frequency will be

$$F_{out} = F_{in} \times \frac{A}{2^{32}} \quad (7)$$

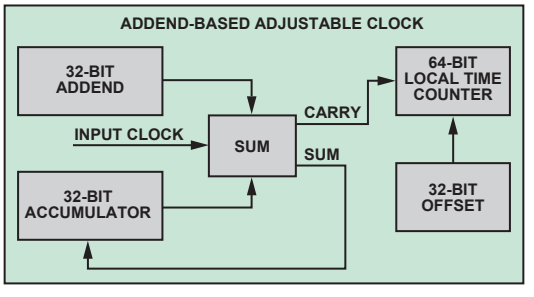


Figure 6. Addend-based adjustable clock.

### Implementation of IEEE 1588 on the ADSP-BF518 Processor

A complete IEEE 1588-2008-compliant system was built on an ADSP-BF518 processor as shown in Figure 7.

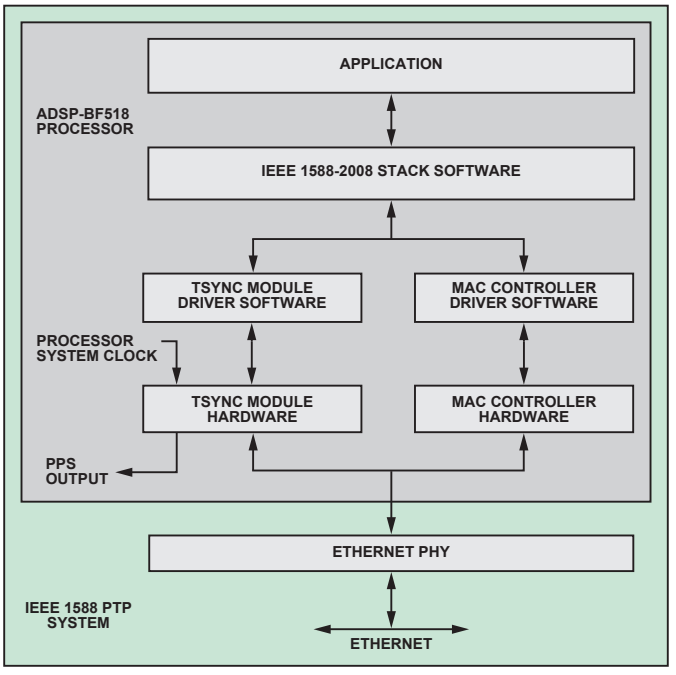


Figure 7. An implementation of IEEE 1588 on the ADSP-BF518.

The TSYNC module of the processor detects incoming and outgoing IEEE 1588 messages and uses hardware to time-stamp event messages. The IEEE 1588 stack software, provided by IXXAT (IXXAT Automation GmbH), implements the message-exchange protocol required by the standard. It makes use of the TSYNC driver to read, write, and adjust the TSYNC clock, and uses the MAC controller driver to send and receive messages on the Ethernet MAC layer (Layer 2 of the Open-Systems Interconnection Model). It also implements the control law and filtering of P2P delay measurements. The Ethernet PHY is National Semiconductor DP83848,[6] chosen because of its low jitter delay characteristics. For simplicity, the processor's system clock (80 MHz) was chosen to be the TSYNC module clock source.

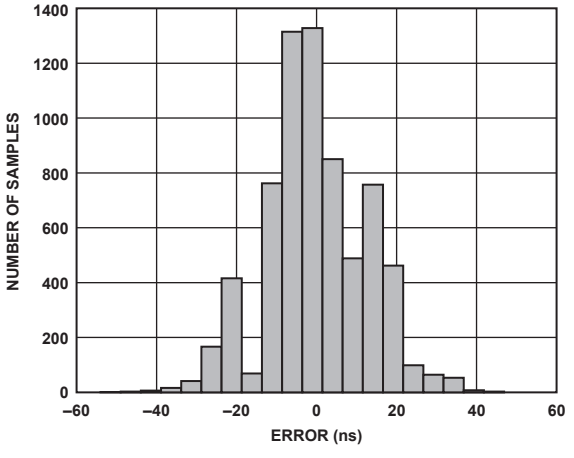


Figure 8. Histogram of slave-clock error of an IEEE 1588 system on ADSP-BF518.

Figure 8 shows the clock synchronization performance of the device as a histogram of the measured error between two identical ADSP-BF518 IEEE 1588 systems. 6938 measurements were taken over a period of approximately 1700 seconds. The resulting mean error is 0.015 ns, and the standard deviation is 12.96 ns. A Sync message interval of 0.25 seconds was used for this test.

### Conclusion

The IEEE 1588 standard provides a highly accurate, low-cost method for synchronizing distributed clocks. While hardware support is not explicitly required for IEEE 1588, hardware-assisted message detection and time stamping is critical to achieve the highest level of synchronization precision. The ADSP-BF518 processor provides hardware support for both IEEE 1588-2002 and IEEE 1588-2008, including features that can support a wide range of applications. High-precision clock synchronization has been demonstrated by implementing IEEE 1588 technology using the ADSP-BF518 processor and the IXXAT IEEE 1588-2008 protocol software.

### References


[1] ADSP-BF518 data sheet. http://www.analog.com/en/embedded-processing-dsp/blackfin/adsp-bf518/processors/product.html.

[2] IEEE Std. 1588-2002. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. http://ieee1588.nist.gov/PTTI_draft_final.pdf.

[3] IEEE Std. 1588-2008. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. http://ieee1588.nist.gov.

[4] ADSP-BF537 data sheet. http://www.analog.com/en/embedded-processing-dsp/blackfin/adsp-bf537/processors/product.html.

[5] *ADSP-BF51x Blackfin Processor Hardware Reference Preliminary, Revision 0.1* (Preliminary). January 2009. Analog Devices, Inc. http://www.analog.com/static/imported-files/processor_manuals/bf51x_hwr_rev_0-1.pdf.

[6] AN-1507: *DP83848 and DP83849 100Mb Data Latency*. 2006. National Semiconductor Corporation. http://www.national.com/an/AN/AN-1507.pdf.


### Authors


**Dr. Jiang Wu** [jiang.wu@analog.com] joined Analog Devices in 2006, where he works on real-time embedded systems and digital processing. In 2004, he received a PhD in electrical engineering from the University of Rhode Island. In 1996 and 1993, respectively, he received MS and BS degrees in automation, both from the Beijing Institute of Technology. His graduate research work was on modeling and instrumentation of biomedical systems. Prior to ADI, Jiang worked as a system engineer at Vivoda Communications.

**Robert Peloquin** [robert.peloquin@analog.com] joined Analog Devices in 2000. As a senior applications engineer, he contributes to many different aspects of ADI's digital signal processor solutions. Some recent areas include Ethernet AVB and IEEE 1588. Prior to joining ADI, Bob worked at the Naval Undersea Warfare Center performing applied research for unmanned undersea vehicle technologies. He holds a BSEE degree from Syracuse University and a MSEE degree from the University of Massachusetts, Dartmouth.