



Hardware-Assisted IEEE 1588* Implementation in the Intel® IXP46X Product Line

White Paper

March 2005



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

BunnyPeople, Celeron, Chips, Dialogic, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel Centrino, Intel Centrino logo, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Xeon, Intel XScale, IPLink, Itanium, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, Sound Mark, The Computer Inside, The Journey Inside, VTune, and Xircom are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2005, Intel Corporation. All Rights Reserved.

Contents

1.0 Introduction	5
1.1 Reference Documents	5
2.0 IEEE 1588* Overview	5
3.0 Hardware Versus Software IEEE 1588* Implementations	6
4.0 1588* Implementation in the Intel® IXP46X Product Line of Network Processors	8
4.1 1588* Hardware Implementation in the Intel® IXP46X Product Line of Network Processors	9
4.1.1 Overview	9
4.1.2 Taking Timestamps	10
4.1.3 Frequency-Compensated Clock	12
4.1.4 Schedule Interrupts to the Intel XScale® Core	14
4.1.4.1 Message Errors	14
4.1.4.2 1588* Logic Design	14
4.2 1588* Software Implementation in Intel® IXP46X Product Line of Network Processors	14
4.2.1 Clock Synchronization Algorithm	14
4.3 Target Applications	16
5.0 Conclusions	17

Figures

1 Capturing Timestamps at Different Layers Have Different Protocol Delay Fluctuations	6
2 1588* Logic Implementation Inside Intel® IXP46X Network Processors	8
3 1588* Hardware Logic in Intel® IXP46X Network Processors	10
4 Timestamp Reference Point	11
5 Frequency Compensated Clock Logic	12
6 Sync Message Timing	15

Tables

1 FreqCompValue Hexadecimal Examples	13
--	----

Revision History

Date	Revision	Description
March 2005	001	Initial release.

1.0 Introduction

This document describes a hardware-assisted IEEE 1588* implementation in the IXP46X product line of network processors. An overview of the 1588 standard is presented, and the general pros and cons of hardware- versus software-oriented 1588 implementations is discussed. Detailed description of both the 1588 hardware logic in IXP46X network processors, and the Intel XScale® Core-based software-programming model, is provided. Examples of 1588 applications are also included.

1.1 Reference Documents

The following publications are referenced in this document.

Document Title	Web Site
[1] <i>A Frequency Compensated Clock for Precision Synchronization using IEEE 1588 Protocol and its Application to Deterministic Ethernet.</i> Sivaram Balasubramanian, Kendal R. Harris and Anatoly Moldovansky, Rockwell Automation. September 24, 2003.	http://IEEE1588.nist.gov , click on "Past Conferences", "Proceedings of the Workshop..."
[2] <i>Application of IEEE 1588 to Distributed Motion Control.</i> Kendal R. Harris, Sivaram Balasubramanian, and Anatoly Moldovansky, Rockwell Automation. September 24, 2003.	
[3] <i>IEEE 1588: Running Real-time on Ethernet.</i> Dirk S. Mohl, The OnLine Industrial Ethernet Book, Issue 17, November 2003.	http://IEEE1588.nist.gov , click on "Publications"
[4] <i>IEEE-1588™ Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.</i> December 3-5, 2002.	

2.0 IEEE 1588* Overview

The need for distributed network technologies increases as measurement and control system applications become more complex with larger numbers of nodes. Most of these applications can be enhanced through the use of local clocks (for example, in sensors, actuators, or other devices) at each node to achieve an accurate distribution-wide sense of time. Each of these individual clocks, however, tend to 'drift apart' due to instabilities inherent in source oscillators and environmental conditions such as temperature, air circulation, mechanical stresses, vibration, aging, etc.^[1] Therefore, some kind of correction is required to synchronize the individual clocks to maintain a common and accurate notion of distribution-wide time.

The publication *IEEE-1588™ Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems* ^[4] defines a message-based protocol to enable accurate synchronization of clocks (that have varying precision, resolution, and oscillator stability) in distributed systems supporting multi-cast messaging including, but not limited to, Ethernet.

The 1588 protocol defines a 'Best Master Clock (BMC) Algorithm', where each clock in a distributed system identifies the most accurate 'master' clock — to which all other 'slave' clocks then synchronize.

The protocol defines several periodic messages (independent of the distributed network technology) that trigger a clock to capture a timestamp and communicate timestamp information between master and slave clocks. This method of using timestamps allows each slave clock in a distribution to analyze the master's timestamp and the network propagation delay, and ultimately

determine its clock delta from the master in the slave clock’s synchronization algorithm. The 1588 protocol does not define the means by which a slave clock will adjust itself to synchronize with the master clock nor does it define how timestamps are captured.

With minimal network and local computing resources support, sub-microsecond-range synchronization accuracy is achievable. The default 1588-specified behavior allows simpler systems to be installed without system-administrator attention. Many Ethernet-based applications, with hardware assistance, have delivered synchronization accuracy on the order of tens and hundreds of nanoseconds [3].

3.0 Hardware Versus Software IEEE 1588* Implementations

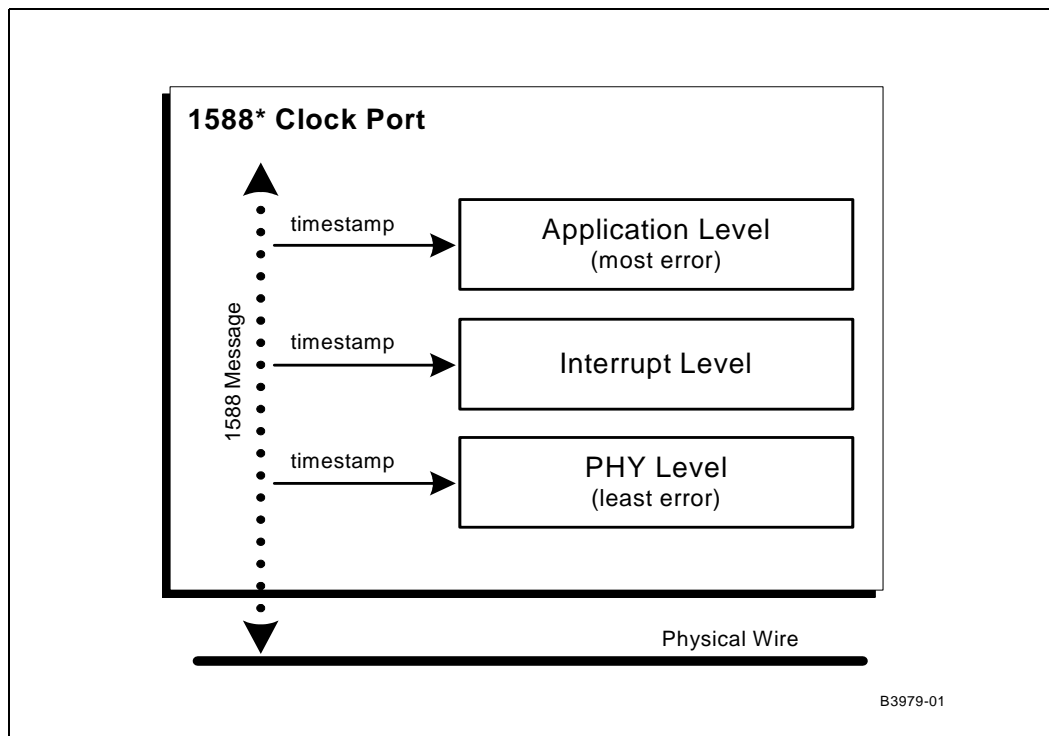
Two implementations exist: software-only implementations, and hardware-assisted software implementations. Hardware-assisted implementations deliver more precise clock synchronization than software-only implementations.

The following sections describes four factors that limit clock-synchronization accuracy in a 1588 system [4]:

1. Network Protocol Stack Delay Fluctuation

Figure 1 illustrates three layers at which a timestamp may be captured within a clock node implementing 1588.

Figure 1. Capturing Timestamps at Different Layers Have Different Protocol Delay Fluctuations



The simplest 1588 implementations include ordinary applications at the top of the network protocol stack and generate timestamps at the application level. This implementation incurs the largest protocol stack delay fluctuation, thus yielding the least accuracy as the largest amount of error is introduced into the timestamp. Depending on the operating system, errors induced typically range on the order of a hundred microseconds to milliseconds.

Lower interrupt level processing further reduces the network protocol delay fluctuation. Depending on whether or not the interrupts are being used by other applications, the errors introduced into the timestamp typically range in the tens of microseconds.

Hardware-assist methods achieve the greatest accuracy and generate timestamps as close to the wire as possible, at the physical layer. Network-protocol-delay fluctuations for these implementations range in nanoseconds to sub-microsecond. For example, in an Ethernet-based system, these errors originate from phase lock delays of physical layer chips that recover the data synchronization binary clock. However, the effect of such a narrow range of error can be reduced by appropriate design of the clock synchronization algorithm.

2. Network Technology Components' Delay Fluctuation

Many networks contain combinations of repeaters, switches or routers, which vary the size and complexity of the distributed network. These network components each induce a varying range of fluctuation in the propagation time of 1588 messages. These fluctuations directly affect the accuracy of the clock synchronization since the synchronization algorithm includes the propagation delay calculation.

Network repeaters don't have any store or forward capability and so they usually introduce the same narrow range of nanoseconds delay fluctuation as the PHY level of the network protocol stack. Appropriate design of the clock synchronization algorithm reduces this fluctuation.

Network switches, such as those found in large Ethernet subnets, may contain store and forward capability, which, depending on traffic load and switch design, can introduce delay fluctuation on the order of microseconds. Averaging algorithms can be employed over the range of delay in the clock synchronization algorithm to help minimize this delay.

Routers can introduce large delay fluctuations (on the order of milliseconds and above), which prevent a straightforward application of 1588 to achieve a high level of synchronization accuracy. The 1588 protocol uses the concept of a 'boundary clock' ^[4], the description of which is outside the scope of this white paper. Placed at a router, the boundary clock allows multi-millisecond router delay fluctuations to be replaced with the much-reduced delay fluctuations of lower network protocol stack implementations discussed earlier.

3. Clock Timestamp Accuracy

The accuracy of the clock requires consistent resolution, (i.e., the number of logical bits of the clock generating the timestamps in a clock node) with the required accuracy of the system design. A mathematical example of this calculation, shown in [Section 4.0, "1588* Implementation in the Intel® IXP46X Product Line of Network Processors"](#), describes this requirement.

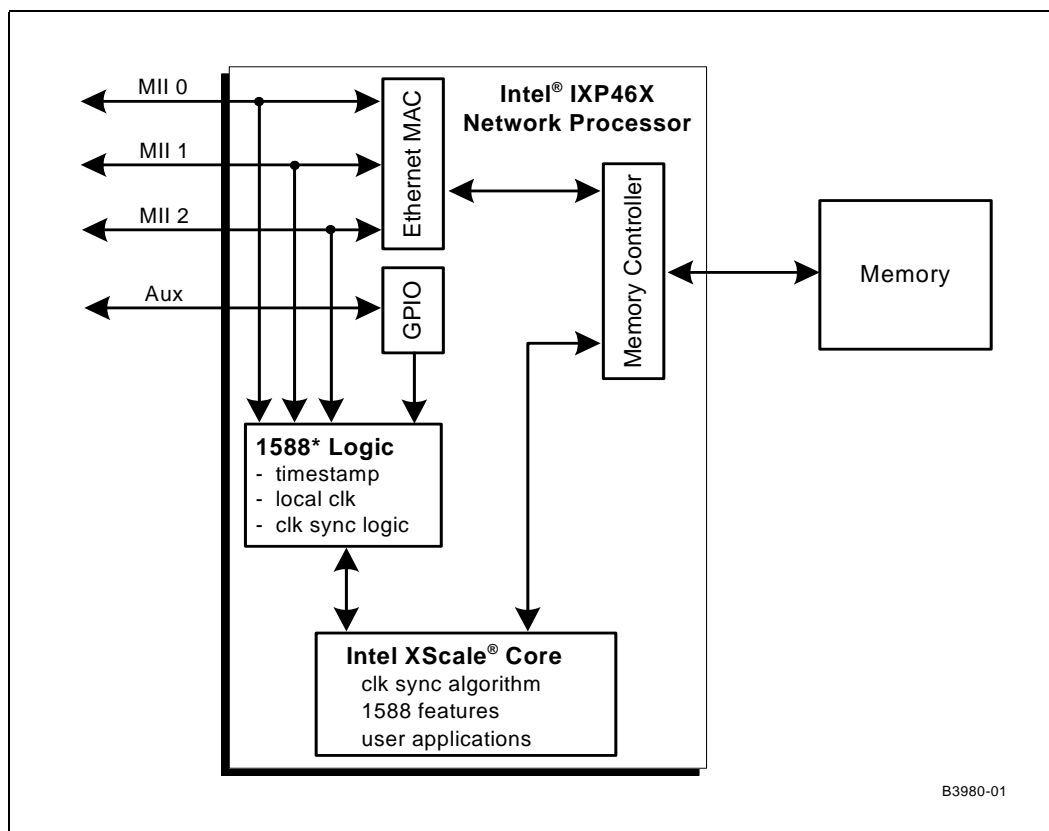
4. Clock Oscillator Stability

The inherent instability at each clock's oscillator creates drift between all of the clocks in a distributed system. The algorithms used to reduce delay fluctuation will not correct these local clock drifts in the time in between the 1588 periodic synchronization message, which, at a minimum, is one second. The drift that occurs in this time depends on the accuracy of the oscillator purchased and the control of its operating environment. For example, designers often use more-expensive, higher-accuracy oscillators for the oscillator source. The designer must consider the trade-off between cost and stability.

4.0 1588* Implementation in the Intel® IXP46X Product Line of Network Processors

To deliver a high level of clock synchronization accuracy (in the hundreds of nanoseconds to sub-microseconds range) on three MII interfaces for Ethernet networks, Intel chose to integrate a hardware-assisted 1588 implementation into the IXP46X product line. By integrating this logic into the network processor rather than into an off-chip FPGA, savings in printed circuit board area and cost can be realized. Figure 2 shows the position of the 1588 logic in the network processor. This logic also supports one non-MII “auxiliary” interface via general-purpose I/O (GPIO) pins.

Figure 2. 1588* Logic Implementation Inside Intel® IXP46X Network Processors



The software that utilizes this logic executes on the Intel XScale core, thus increasing the designer’s flexibility to implement any range of the specified 1588 feature set. The hardware-assisted clock synchronization algorithm employs a ‘frequency scaling’ method specially designed to deliver highly precise time synchronization using standard and inexpensive oscillators.

A detailed description of the hardware and software implementations follows.

4.1 1588* Hardware Implementation in the Intel® IXP46X Product Line of Network Processors

4.1.1 Overview

The hardware implementation, shown in [Figure 3](#), consists of a 64-bit System time clock register that maintains the local clock time. A 64-bit Target time clock register is loaded with a value such that when the System time clock equals or exceeds the set Target time clock, the hardware generates an interrupt to Intel XScale core, and thus provides the means to schedule events from the synchronized System clock.

The logic runs at a 66-MHz binary clock, shown in [Figure 5](#), generated from the network processor's clock circuitry via an off-chip 33.3-MHz standard oscillator. The term 'FreqOscillator' describes the 66-MHz binary clock.

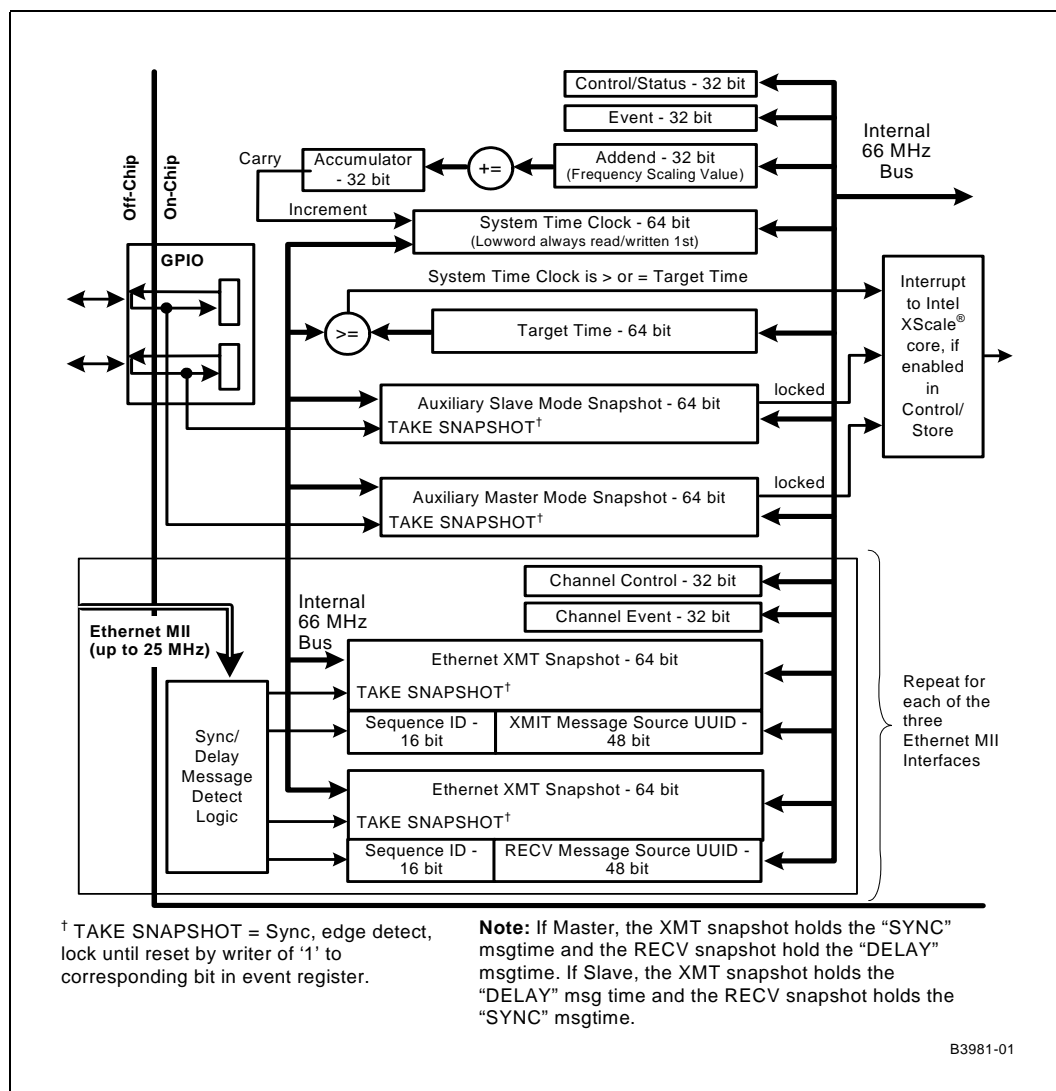
A 32-bit Addend and 32-bit Accumulator register exist to allow the software synchronization algorithm to throttle the frequency of the System time clock. This enables the means of accurate clock adjustment.

The hardware takes system time clock snapshots or timestamps when triggered to do so by detecting specified 1588 'SYNC' or 'DELAY_REQ' messages ^[4] on any of the three monitored MII interfaces. Two 64-bit timestamp registers are used, one for transmit and one for receive, per MII interface.

An Auxiliary pair of 64-bit registers also exist to capture timestamps of the System time clock when triggered to do so by two GPIO pins. This enables support for a non-MII interface, such as a global positioning satellite (GPS) clock source.

The logic supports both Master and Slave modes of operation, per interface, as directed by software.

Figure 3. 1588* Hardware Logic in Intel® IXP46X Network Processors



4.1.2 Taking Timestamps

1588 specifies four messages that form the framework of the protocol: Sync, Follow_up, Delay_Req, and Delay_Resp. For further details of their usage, refer to the 1588 specification [4].

Taking a timestamp means that the System time clock register captures its current value in a second 'snapshot' register. Separate transmit and receive snapshot registers and control state machines exist for each of the three support MII interfaces and a pair for the Auxiliary interface.

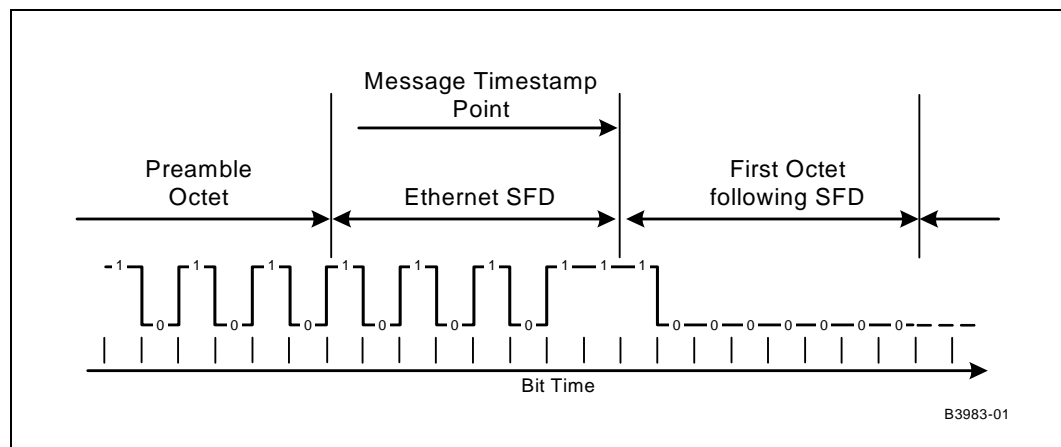
A software-configurable Interface Control register determines if an interface is in master or slave mode. This is used by an interface to know which 1588 messages to detect and capture into the transmit and receive snapshot registers.

On the MII interfaces, the hardware generates timestamps only upon detection of the Sync and Delay_Req messages. A master interface timestamps Sync messages transmitted and Delay_Req messages received. Conversely, a slave interface timestamps Delay_Req messages transmitted and Sync messages received. The hardware does not timestamp for Follow_up and Delay_Resp messages.

As shown in Figure 4, the hardware captures the timestamp immediately after the Start of Frame Delimiter (SFD) of these messages. The 1588 protocol specifies the message timestamp point. Ethernet messages use UDP/IP, which means that messages can be lost and software must compensate for this. Therefore, the captured timestamp is not ‘locked’ into the snapshot register until the last nibble of the frame’s CRC is transmitted or received. Once locked, the hardware sets a unique indication of the snapshot in an appropriate Event register. It can’t be unlocked to allow another timestamp to be taken until acknowledged by software by writing a ‘1’ to the appropriate bit in the Event register. The hardware allows disabling the locking feature in the Control register.

The hardware can determine the last nibble of the CRC, and the byte offset and value in the Ethernet frame to identify the message, because of the Sync and Delay_Req messages fixed length and location.

Figure 4. Timestamp Reference Point



The PHY bit locking delay and binary clock cross-over synchronization delay, from MII 10 or 100 speed to 66-MHz bus, causes the timestamp to be slightly later than the reference point. The software adjusts for this deterministic delay and acquires the MII speed used via the PHY’s Management Data Interface (MDI).

The master-configured MII interface transmits a multicast Sync message periodically over the network at 1, 2, 8, 16, or 64 second intervals. A Sync message defines a value of 0x00 in byte 74 of the Ethernet frame. The Delay_Req message defines a value of 0x01 in byte 74 of the Ethernet frame, after the SFD.

In addition to the timestamp, the logic captures the Sequence ID and Source UUID when a Sync message is received by a slave-configured interface, or a Delay_Req message is received by a master-configured interface.

For the Auxiliary interface, inputs received from either of the two GPIO pins each trigger one timestamp snapshot register, and these inputs are cross-over synchronized to the 66-MHz domain so that raw inputs may be applied. Configuring these two GPIO pins as outputs allows the Intel XScale core to trigger the internal snapshot, and set an external event, simultaneously. Section 4.3, “Target Applications”, on page 16 describes an example of this auxiliary interface usage.

The message-detection logic supports ‘Tagged MAC Frames’ from the IEEE 802.3* standard, which defines priority-based messages. Only the IPv4 message format is supported.

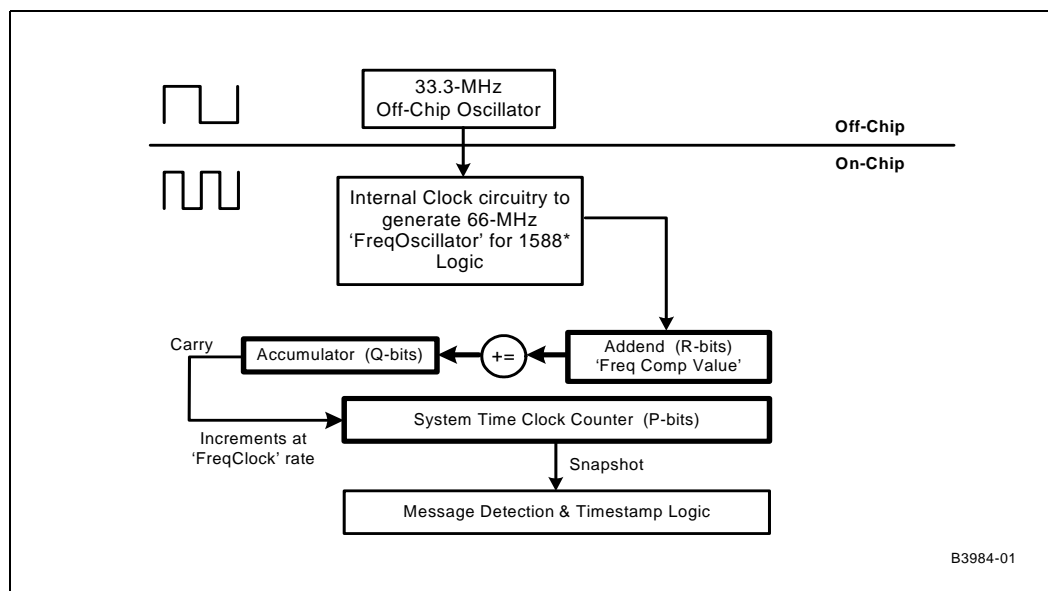
For debug purposes, the hardware implements a ‘Traffic Analyzer Support’ feature that disables timestamp locking, thus every message on the network triggers a timestamp capture. The hardware requires the Intel XScale core to read the timestamp before the next SFD, or the timestamp is overwritten.

4.1.3 Frequency-Compensated Clock

Simply updating the System time with the new calculated synchronized time value could disrupt linear time in applications such as real-time motion control. It is necessary, therefore, to manage clock drifts and maintain linear time between Sync message periods. One effective method is to use a frequency-compensated clock, which also maintains a high degree of clock accuracy using standard, inexpensive oscillators [1].

Generically, frequency-compensated clock logic, shown in Figure 5, consists of a P-bit System clock counter register, a Q-bit Accumulator register as a high-precision frequency divider, and an R-bit Addend register that holds the frequency compensation value ‘FreqCompValue’. All the above logic operates at the frequency of the input oscillator ‘FreqOscillator’. The hardware adds the FreqCompValue set in the Accumulator register once every 1/FreqOscillator. The System time clock is incremented by the overflow pulse signal from the Accumulator; therefore, the rate at which the System clock time is incremented, the ‘FreqClock’, is determined by the FreqCompValue.

Figure 5. Frequency Compensated Clock Logic



The ratio of FreqOscillator to FreqClock, called the ‘FreqDivRatio’, is a design constant initially chosen to be a number greater than 1.0001, if using a 0.01% stability oscillator (+/- 100 PPM). The frequency drift in this case may be an infinite precision real number between 0.9999 and 1.0001. As it is not possible in hardware to compensate for infinite precision, a ‘CompensationPrecision’ constant is defined that represents the highest precision desired for frequency compensation. Also a ‘SyncInterval’ is defined as the interval of time, in multiples of seconds, between 1588 Sync message issuance.

Using the following mathematical relationships, the designer can determine the widths of the Addend, Accumulator, and System time clock registers that deliver the required precision for frequency compensation.

- a. $\text{FreqDivRatio} = \text{FreqOscillator} / \text{FreqClock}$
- b. $\text{CompensationPrecision} \leq 1 / (\text{SyncInterval} * \text{FreqClock})$
- c. $2^Q \geq \text{FreqDivRatio} / \text{CompensationPrecision}$
- d. $2^R \geq 2^Q / \text{FreqDivRatio}$
- e. $2^P \geq 2^Q$

In the case of the implemented 1588 logic, for example:

$\text{FreqOscillator} = 66 \text{ MHz}$

$\text{FreqClock} = 50 \text{ MHz}$

In the above example, the rate of the System clock time is set to 75% of the max FreqOscillator rate. This is typical in applications to allow headroom for the system clock time rate to be increased, thereby speeding up the clock, as well as decreasing for slowing the clock down, when adjusting the System time clock for synchronization.

$\text{FreqDivRatio} = 1.32,$

$\text{SyncInterval} = 1 \text{ second}$

$\text{CompensationPrecision} = 1 \times 10^{-9}$

Width of Addend register, $R = 32$

Width of Accumulator register, $Q = 32$

Width of System clock time register, $P = 64,$ to provide a clock resolution of 20 nanoseconds.

The following shows the equation used to calculate the FreqCompValue (if the Accumulator is 32 bits) that's written to the Addend register to adjust the System time clock rate:

$\text{FreqCompValue} = 2^{32} / \text{FreqDivRatio}$

The FreqCompValue is written in hexadecimal into the 32-bit Addend register. [Table 1](#) shows three examples.

Table 1. FreqCompValue Hexadecimal Examples

Frequency Oscillator	Frequency Clock	Frequency Division Ratio	Frequency Compensation Value
66 MHz	40 MHz	1.65	0x9B26C9B3
66 MHz	50 MHz	1.32	0xC1F07C1F
66 MHz	60 MHz	1.1	0xE8BA2E8B

This Accumulator-Addend register pair implements a high-precision frequency divider that is at least as accurate as CompensationPrecision. It is expected that the frequency-compensated clock method, using standard oscillators, will satisfy the accuracy requirements for many target applications.

Section 4.2, “1588* Software Implementation in Intel® IXP46X Product Line of Network Processors”, on page 14 discusses a detailed description of the synchronization algorithm that controls the frequency-compensation logic.

4.1.4 Schedule Interrupts to the Intel XScale® Core

The hardware generates an interrupt signal to the Intel XScale core when any of the following conditions occur:

- Target Time Expiration
- Auxiliary Master Snapshot is taken
- Auxiliary Slave Snapshot is taken

Target time expiration occurs when the System time clock equals or exceeds the set Target time clock, thereby allowing a method for software to accurately schedule an interrupt event off of the synchronized System time clock.

4.1.4.1 Message Errors

The logic ignores messages flagged with an error signal from the PHY or 1588 messages not properly formatted per 1588 specifications. No hardware provisions exist for error detection except the ‘locking’ feature described earlier. During CRC or other errors detected by the Ethernet MAC or higher levels of the network protocol stack, the software responsibilities include deleting the message and ignoring any timestamps incorrectly captured by the logic.

Software may also soft-reset this logic to the same state that occurs after power-on reset.

4.1.4.2 1588* Logic Design

The logic has two main interfaces: the 25-MHz MII bus, and the 66-MHz Internal bus. The logic is designed to scale with each generation of future Intel network processor, in terms of number and type of supported interfaces and features.

4.2 1588* Software Implementation in Intel® IXP46X Product Line of Network Processors

In the software realm, with the standard Ethernet data path going to memory, the programmer may implement any desired feature set of the 1588 specification, even a full software implementation of 1588. Control of messages such as Management, Follow_up, and Delay_Resp, implementation of data sets to determine if a node operates as a slave, master or grandmaster^[4], etc., become user-defined code. For the highest accuracy, hardware assistance provides precise timestamp capture based on Sync and Delay_Req messages and also logic to precisely adjust the System time clock rate. The following describes the software algorithm that utilizes this hardware to deliver clock synchronization accuracy in the order of nanoseconds to sub-microseconds.

4.2.1 Clock Synchronization Algorithm

Figure 6 shows master and a slave clock nodes. At time MasterSyncTime (n-1) the master sends a Sync message to the slave clock(s). The slave receives this message after a propagation delay ‘MasterToSlaveDelay’ at MasterClockTime(n-1), and the slave sets its clock as follows:

$$\text{SlaveClockTime}(n-1) = \text{MasterClockTime}(n-1) = \text{MasterSyncTime}(n-1) + \text{MasterToSlaveDelay}$$

Note: MasterSyncTime(n-1) is sent to the slave in the Sync message and 'n' denotes the Sync message count.

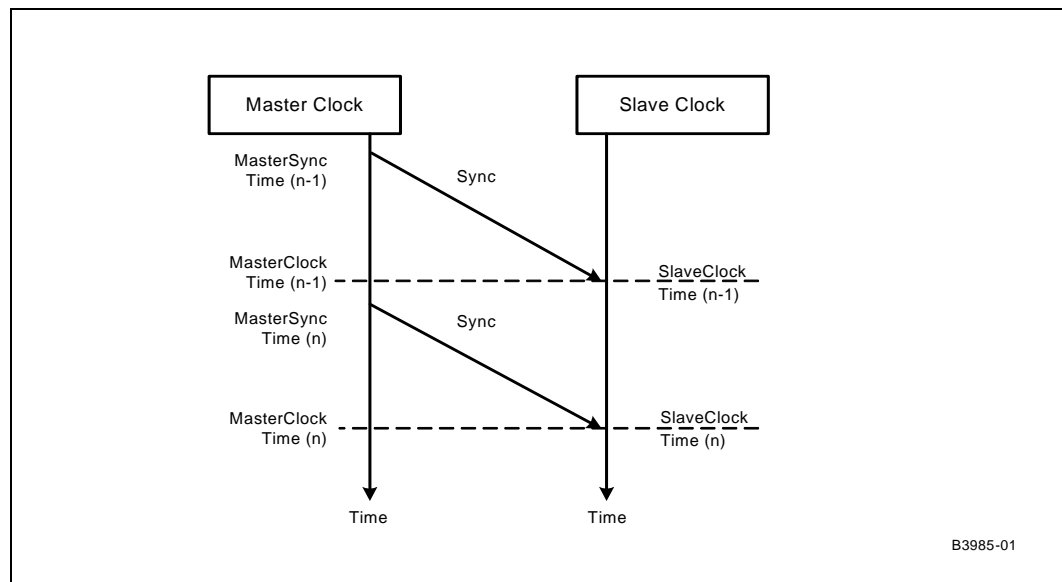
At time MasterSyncTime(n) the master sends the next Sync message to the slave. The slave receives this message at MasterClockTime(n) but the SlaveClockTime(n) has drifted apart by SlaveClockDrift. Therefore:

$$\begin{aligned} \text{MasterClockTime}(n) &= \\ &\text{MasterSyncTime}(n) + \text{MasterToSlaveDelay} \\ \text{SlaveClockTime}(n) &= \\ &\text{MasterSyncTime}(n) + \text{MasterToSlaveDelay} + \text{SlaveClockDrift} \end{aligned}$$

Hence, the slave clock has to be corrected such that:

- a. The frequencies of the master and slave clocks are equal to each other.
- b. The difference in value of the MasterClockTime and the SlaveClockTime should be eliminated or minimized.

Figure 6. Sync Message Timing



Initially the slave clock's Addend register is set with the appropriate FreqCompValue(0), as per the desired FreqClock, and let the MasterToSlaveDelay be set to zero. The synchronization algorithm described below will be applied, and after a few Sync cycles, a frequency lock will occur. The slave will then determine the precise value for the MasterToSlaveDelay and resynchronize with the master using a new FreqCompValue(1).

At time MasterSyncTime(n) the master sends a Sync message to the slave. The slave receives this message at local time SlaveClockTime(n) and calculates MasterClockTime(n) as follows:

$$\begin{aligned} \text{MasterClockTime}(n) &= \\ &\text{MasterSyncTime}(n) + \text{MasterToSlaveDelay} \end{aligned}$$

The master clock count for the current Sync cycle, MasterClockCount(n), is:

$$\text{MasterClockCount}(n) = \text{MasterClockTime}(n) - \text{MasterClockTime}(n-1)$$

The slave clock count for the current Sync cycle, SlaveClockCount(n), is:

$$\text{SlaveClockCount}(n) = \text{SlaveClockTime}(n) - \text{SlaveClockTime}(n-1)$$

The delta between master and slave clock counts for the current Sync cycle, ClockDiffCount(n), is:

$$\text{ClockDiffCount}(n) = \text{MasterClockTime}(n) - \text{SlaveClockTime}(n)$$

The frequency scaling factor for the slave clock, FreqScaleFactor(n), is:

$$\text{FreqScaleFactor}(n) = (\text{MasterClockCount}(n) + \text{ClockDiffCount}(n)) / \text{SlaveClockCount}(n)$$

And so the FreqCompValue for the slave's Addend register for this Sync cycle, n, is:

$$\text{FreqCompValue} = \text{FreqScaleFactor}(n) * \text{FreqCompValue}(n-1)$$

Theoretically, this algorithm would achieve lock in one Sync cycle; however, it may take several cycles due to changing network propagation delays and operating conditions.

This algorithm is self-correcting, meaning if for any reason the slave clock is initially set to a value that is incorrect from the master, the algorithm will correct it at the cost of more Sync cycles.

The frequency-compensated clock provides a simple and highly accurate method for clock synchronization on 1588, using inexpensive, standard oscillators. It also maintains time and manages drift accurately, even in the presence of lost synchronization messages.

An implementation of this algorithm, using an off-chip FPGA hardware assist, yielded slave clock synchronization accuracy in a best-case range of 25 to 100 nanoseconds ^[1].

4.3 Target Applications

Intel's hardware-assisted 1588 implementation in the IXP46X product line is designed to fit into a wide range of control and measurement systems that require network connectivity.

Using the IXP46X product line allows various applications requiring reliable sub-microsecond clock synchronization accuracy, such as real-time industrial automation and telecommunications, to achieve its goals with minimal design effort along with significant cost and PCB-area savings. Applications requiring a greater level of accuracy, in the nanoseconds range, are also possible with further design effort. This effort is both on- and off-chip in terms of controlling network-component and propagation delays as well as environmental and clock drift issues.

An example application that utilizes both the MII and Auxiliary interfaces would be a master or grandmaster node configuration that uses a highly accurate GPS clock. The GPS clock would issue a 'pulse per second' GPIO input to trigger Auxiliary snapshots of the local System time clock,

which then gets synchronized with the GPS clock. This master or grandmaster node will then issue Sync messages to the MII-based distribution to allow slave clock nodes to synchronize to the GPS clock.

5.0 Conclusions

The Intel 1588 implementation pioneers 1588 time-synchronization integration with IXP46X network processors. The co-hardware and software implementation is designed to deliver a high-accuracy clock synchronization solution — in the nanosecond to sub-microsecond range — for Ethernet-based applications, with the added benefits of reduced bill of materials cost and printed circuit board area.

This page is intentionally left blank.