Synchronization Improvement of Distributed Clocks in EtherCAT Networks

Sung-Mun Park, Hongju Kim, Hyoung-Woo Kim, Chang Nho Cho, and Joon-Young Choi

***Abstract*—The aim of this study is to improve the performance of clock synchronization in EtherCAT networks. Although EtherCAT synchronizes the slave clocks to the reference clock using a distributed clock synchronization mechanism, a synchronization error still exists between them. We propose a method to estimate and significantly reduce this error. In addition, we implement the method within the existing synchronization mechanism without adding an excessive computational load. The experimental results for synchronization performance verify that the proposed method improves the accuracy of clock synchronization in EtherCAT networks.**

***Index Terms*—EtherCAT, distributed clock, drift compensation, clock synchronization.**

I. Introduction

In automation applications, industrial Ethernet networks have been widely employed because of their various benefits such as high bandwidth, high transfer rate, flexible topologies, and multiple protocols [1]. EtherCAT is one of the industrial network protocols based on Ethernet, and it uses an IEEE 802.3 standard Ethernet frame containing one or more datagrams, which are the basic message units of EtherCAT. A desirable feature that distinguishes EtherCAT from other industrial protocols using the Ethernet frame is that each slave processes the addressed data "on the fly" as the frame moves downstream. Consequently, the EtherCAT frame is delayed only by hardware propagation, which makes the theoretical maximum effective data rate even higher than 100 Mbps using the full duplex feature of Ethernet [2].

In communication systems, the performance of asynchronous networks is known to degrade due to non-negligible clock deviations [3], [4]. In order to reduce these clock deviations, EtherCAT provides a synchronization solution using a distributed clock (DC) mechanism. In this mechanism, the clocks of all the non-reference slaves are adjusted to the reference clock, which is the clock of the first DC-capable slave in the EtherCAT network [5]. However, the internal oscillators of slaves always have a small but non-negligible tolerance, which may cause a synchronization error between the reference clock and the clocks of the other slaves [6]. Moreover, in motion-control systems based on distributed networks, higher synchronization precision is required for precision performance [7].

S.-M. Park, H.-W. Kim and J.-Y. Choi are with the Department of Electronic Engineering, Pusan National University, Busan, 609-735 Korea (e-mail: {psm2689, hwkim0314, jyc}@pusan.ac.kr).

H. Kim and C. N. Cho are with the Korea Electrotechnology Research Institute, Changwon, 642-120 Korea (e-mail: {hjkim0429, cncho}@keri.re.kr).

This research was supported by the KERI Primary research program of MSIP/NST (No. 17-12-N0101-22)

In this letter, we propose a new drift compensation method to improve the EtherCAT synchronization performance. In contrast to the conventional DC synchronization method, the proposed method actively estimates the drift of each non-reference slave at the master and compensates for both the sign and size of the drift without an excessive additional computational load. As a result of the new method, the synchronization error in an EtherCAT network is significantly reduced. This is verified by experimental results using accurate clock deviation measurement.

This letter is organized as follows. In Section II, we introduce the conventional EtherCAT DC mechanism. In Section III, we describe the new drift compensation method for improving clock synchronization performance. In Section IV, the results of the experiments are presented and discussed. We draw the conclusions in Section V.

II. EtherCAT DC mechanism

In this section, we introduce the DC synchronization mechanism of EtherCAT. There are two main causes of clock deviations among the slaves in EtherCAT networks: the mismatch of start-up times among the slaves and frequency differences in the oscillators on the slaves [4]. In order to synchronize clocks between the reference slave and other slaves, EtherCAT uses the DC synchronization mechanism.

EtherCAT slaves manage three time variables for clock synchronization: *system_time*, *local_time*, and *reference_time* [8]. Variable *system_time* is the global clock, which begins on January 1, 2000 with a base unit of 1 ns and is stored in an internal 64-bit register in each slave. Variable *local_time* is an internal clock for each slave and represents the time that starts from zero when each slave powers up. The DC-capable slave that is closest to the master is usually chosen as the reference slave, and *reference_time* is defined as *system_time* of the reference slave.

The DC synchronization procedure consists of three phases: offset compensation, propagation delay measurement, and drift compensation. The offset and drift compensation procedures are outlined in the flowchart in Fig. 1.

The offset of each slave is defined as the time difference between *reference_time* and *local_time* of each slave. To determine the offset of each non-reference slave, the master reads *local_time* of each slave and calculates the difference between it and *reference_time*; the master writes this difference into the *System_Time_Offset* register of each slave. The offset is calculated only once since the network is initialized. Using

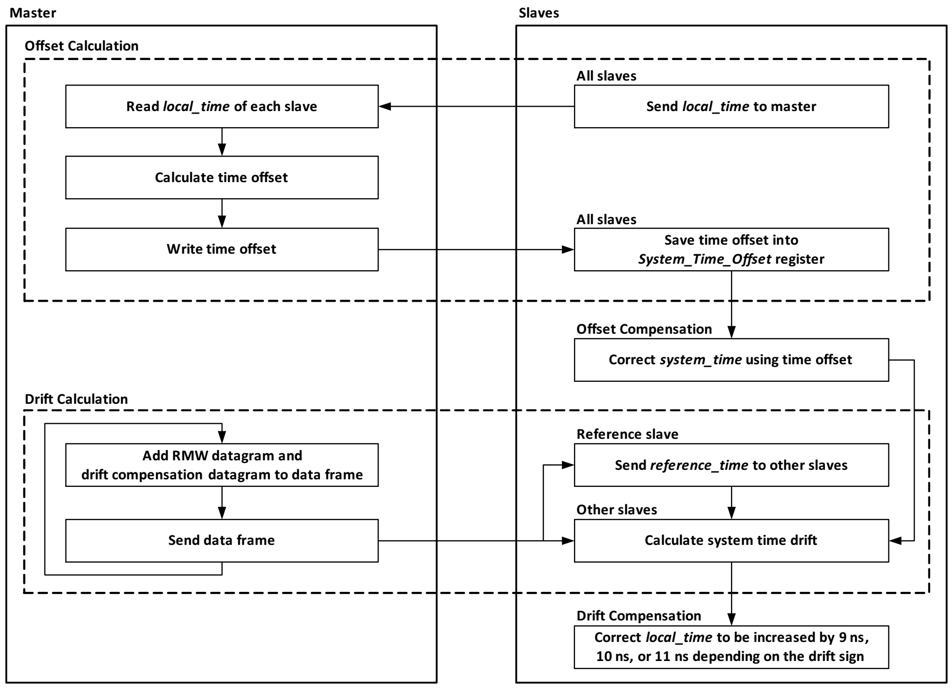

Master
Slaves
Offset Calculation
All slaves
Read local_time of each slave
Send local_time to master
Calculate time offset
All slaves
Write time offset
Save time offset into System_Time_Offset register
Offset Compensation
Correct system_time using time offset
Drift Calculation
Reference slave
Add RMW datagram and drift compensation datagram to data frame
Send reference_time to other slaves
Other slaves
Send data frame
Calculate system time drift
Drift Compensation
Correct local_time to be increased by 9 ns, 10 ns, or 11 ns depending on the drift sign

Fig. 1: DC synchronization procedure in EtherCAT

the calculated offset, each slave compensates for its offset every 10 ns as

$$\textit{system\_time} = \textit{local\_time} + \textit{offset}. \tag{1}$$

The propagation delay of a non-reference slave, which is used for drift compensation, is defined as the time taken to transmit a message to it from the reference slave. The master sends a broadcast write (BWR) datagram to all the slaves and measures the times when the packet first arrives at each slave and is returned to the same slave around the network. Based on the measured times, the master calculates the propagation delay time between the reference slave and other slaves and writes this delay into the *System_Time_Delay* register of each slave. The master measures the propagation delay only once since the network is initialized.

The drift is defined as the time difference between *reference_time* and *system_time* of each slave considering the propagation delay, and is mainly due to the frequency differences of the oscillators in the reference slave and other slaves. Although all the slaves are precisely synchronized at an instance in time, the drift always causes synchronization errors between the reference slave and other slaves as time progresses. To calculate the system time drift of each non-reference slave, the master sends a read multiple write (RMW) datagram and drift compensation datagram to each slave every user-defined cycle time. The reference slave writes *reference_time* into the RMW datagram, and the other slaves read *reference_time* from the RMW datagram. Then, in response to the drift compensation datagram, each non-reference slave calculates the system time drift Δt using the offset time, propagation delay, and *reference_time* as

$$\begin{aligned}\Delta t &= (\textit{local\_time} + \textit{offset}) - (\textit{reference\_time} + pd)\\ &= \textit{system\_time} - (\textit{reference\_time} + pd),\end{aligned} \tag{2}$$

where *pd* denotes the propagation delay between the reference slave and each non-reference slave.

Depending on the sign of the drift Δt calculated for each slave, the time control loop (TCL) compensates for the drift as *local_time* of each slave is increased by 9 ns for positive Δt and 11 ns for negative Δt for 10 ns with the period that is set by the *Speed_Counter_Start* register [8]. We note that the size of the drift is not considered in the drift compensation. Under this drift compensation, even though the master periodically sends a drift compensation datagram to each slave, it is observed that a considerable synchronization error still exists [6].

III. New drift compensation method

As discussed in Section II, each slave still suffers from clock deviation caused by the drift. To achieve a more precise synchronization, we propose a method in which the master estimates and compensates for the drift of each slave. For this purpose, it is necessary to measure the system time drift of each slave at the master. However, because the drift is compensated only in the binary protocol of each slave, the master cannot access the system time drift directly. Instead, we need to estimate the system time drift of each slave using data that is available to the master.

Because the system time drift of each slave is defined as the difference between *reference_time* and *system_time* of each slave, as shown in (2), we need *reference_time* and *system_time* of each slave at the master, which can be easily obtained by reading *system_time* of each slave at the master. Let $\textit{reference\_time}_m^k$ and $\textit{system\_time}_{i,m}^k$ denote *reference_time* and *system_time* of the i-th slave measured by the master at the k-th sampling time, respectively. The following relationship

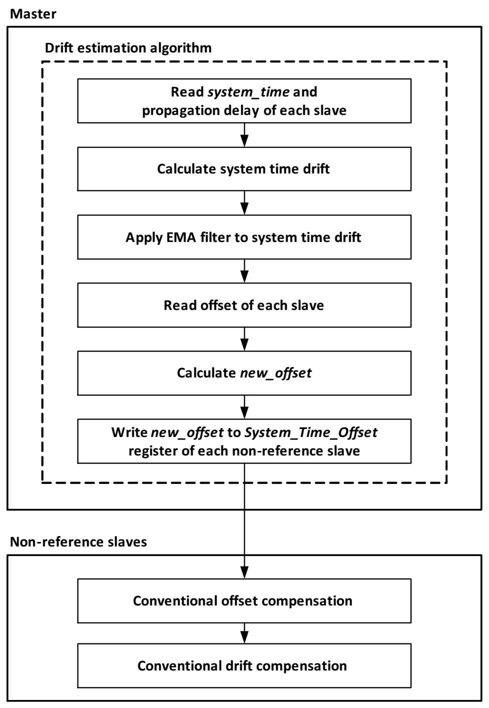

Master
Drift estimation algorithm
Read system_time and propagation delay of each slave
Calculate system time drift
Apply EMA filter to system time drift
Read offset of each slave
Calculate new_offset
Write new_offset to System_Time_Offset register of each non-reference slave
Non-reference slaves
Conventional offset compensation
Conventional drift compensation

Fig. 2: Proposed drift compensation method

then holds:

$$\mathit{reference\_time}_m^k = \mathit{reference\_time}^k - pd_{(m,r)} \quad (3)$$

$$\mathit{system\_time}_{i,m}^k = \mathit{system\_time}_i^k - pd_{(m,r)} - pd_{(r,i)}, \quad (4)$$

where $\mathit{reference\_time}^k$ is *reference_time* at the k-th sampling time, $\mathit{system\_time}_i^k$ is *system_time* of the i-th slave at the k-th sampling time, $pd_{(m,r)}$ is the propagation delay from the master to the reference slave, and $pd_{(r,i)}$ is the propagation delay from the reference slave to the i-th slave. Subtracting (3) from (4) yields the system time drift of the i-th slave at the k-th sampling time Δt_i^k as

$$\begin{aligned}\Delta t_i^k &= \mathit{system\_time}_i^k - \mathit{reference\_time}^k \\ &= \mathit{system\_time}_{i,m}^k - \mathit{reference\_time}_m^k + pd_{(r,i)}, \end{aligned} \quad (5)$$

where $pd_{(r,i)}$ can be measured by reading the *System_Time_Delay* register of the i-th slave at the master using the configured address physical read (FPRD) datagram. Therefore, all the terms of the right hand side of (5) can be measured at the master, and we can estimate the system time drift of each slave at the master at each sampling time k. When the measured system time drift at each sampling time is regarded as a time series, it can be observed that the system time drift is highly oscillatory with a bias component, which is shown in Section IV.

Given the observed dynamic properties of the system time drift, we adopt an exponential moving average (EMA) filter in order to eliminate high frequency noise and extract low frequency components, which includes the bias of the measured system time drift. It is well known that an EMA filter can effectively filter out high frequency components and noise given a small number of samples.

Let ΔT_i^k denote the filtered drift of the i-th slave at sampling time k. The EMA filter for Δt_i^k in (5) is then described as

$$\begin{aligned}\Delta T_i^k &= \alpha \Delta t_i^k + (1-\alpha)\Delta T_i^{k-1} \\ &= \alpha \sum_{j=0}^{k-1} (1-\alpha)^j \Delta t_i^{k-j} + (1-\alpha)^k \Delta t_i^0, \end{aligned} \quad (6)$$

where ΔT_i^k is the estimated system time drift of the i-th slave at the k-th sampling time, the initial condition is set as $\Delta T_i^0 = \Delta t_i^0$, and α is a constant smoothing factor selected between 0 and 1.

In order to compensate for the estimated system time drift of the i-th slave, we need a way to send the estimated drift back to the i-th slave and correct *system_time* of the i-th slave. To do this, we exploit the offset of the i-th slave offset_i, which is used in the periodic correction of *system_time* at the i-th slave, as shown in the offset compensation (1). Hence, the master reads the offset stored in the *System_Time_Offset* register of the i-th slave and modifies the offset as

$$\mathit{new\_offset}_i = \mathit{offset}_i - \Delta T_i^k. \quad (7)$$

In addition, it writes $\mathit{new\_offset}_i$ back to the *System_Time_Offset* register of the i-th slave. Then, whenever the offset compensation is conducted according to (1), the new drift compensation method is applied and compensates for the system time drift in terms of both sign and size, which is in contrast to the conventional method, where the size of the system time drift is not considered.

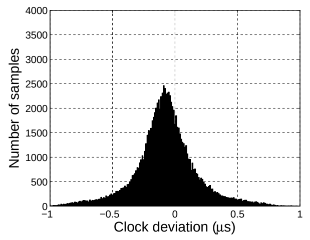
Consequently, applying $\mathit{new\_offset}_i$ to the corresponding slave has the effect of eliminating the low frequency component of the system time drift, which significantly reduces the bias component of synchronization error. The overall procedure of the new drift compensation method is depicted in Fig. 2.

IV. Experiment

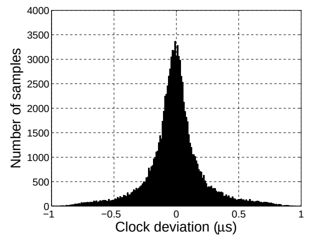
A. Setup

We conducted accurate experiments to evaluate the synchronization performance of the proposed method using the IgH EtherCAT master stack [9], which is a Linux-based open source EtherCAT master, based on a Linux kernel 3.10.32 with the Xenomai patch [10]. The EtherCAT master PC is equipped with an Intel Core i5-2500 Processor, three 1 GB DDR3 RAMs, and an Intel 82574L Gigabit Ethernet Controller. For the EtherCAT slaves, we choose TI's TMDSICE3359 and Panasonic's MADHT1507BA1, which are commercial off-the-shelf EtherCAT slaves. We consider two network configurations: one consists of two TI slaves, and the other three TI and three Panasonic slaves. A square wave generator is connected to the first and last slaves to accurately measure the synchronization performance.

The basic idea of the synchronization performance measurement is to measure *system_time* in response to an external input signal [4]. The square wave generator provides the first and last slaves with an independent periodic external signal, and each slave stores *system_time* to an extra register at each rising edge of the external signal. The master reads

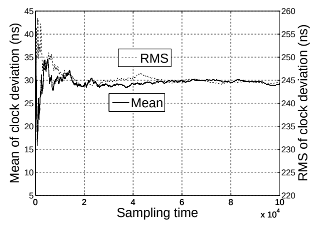


(a) Conventional method

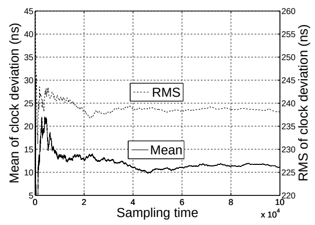


(b) Proposed method

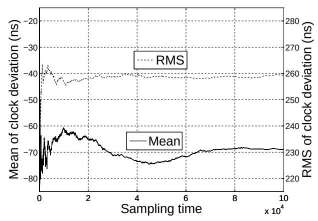
Fig. 3: The distribution of clock deviations between the first and last slaves when the number of slaves is six



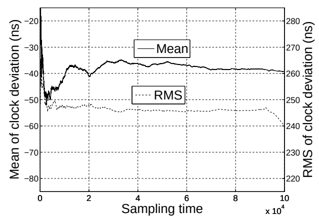
(a) Conventional method (two slaves)



(b) Proposed method (two slaves)



(c) Conventional method (six slaves)



(d) Proposed method (six slaves)

Fig. 4: The mean and RMS of clock deviations between the first and last slaves up to each sampling time

system_time stored in the extra register of each slave and calculates the difference in *system_time* between the first and last slaves.

We turned off the master function to synchronize *reference_time* to *system_time* of the master in order to exclude the influence of the master clock. The master was set to send out one packet every 1 ms, and the periods of the external square wave signal and the proposed algorithm were set to 2 ms and 100 ms, respectively. For every experiment, we first conducted a cold reset in order to eliminate any influence from the previous experiment. After waiting 5 min for the system to stabilize, $100,000$ samples were measured. The constant smoothing factor of the EMA filter for the system time drift estimation was set to $\alpha = 0.05$.

B. Results

We conducted the experiment for both the proposed and conventional methods in order to verify the improvement of the proposed method compared to the conventional method. Fig. 3 shows the distributions of all measured samples of clock deviation between the first and last slaves in a round of experiment. Comparing Fig. 3(a) and 3(b), it is obvious that the proposed method performs better than the conventional method because the center of the proposed method is closer to the zero deviation and the width of the proposed method is narrower.

TABLE I: Statistical properties of clock deviations between the first and last slaves

Statistics		Number of slaves			
		2		6	
		Conventional	Proposed	Conventional	Proposed
GMEAN	(ns)	26	13	-63	-38
MAX	(ns)	1140	1098	1409	1110
MIN	(ns)	-1060	-1057	-1384	-1207
MRMS	(ns)	241	236	255	243

Fig. 4 shows the mean and root mean square (RMS) of clock deviations between the first and last slaves up to each sampling time for the two and six slave networks. As shown in Fig. 4, the magnitude of the means and RMS of clock deviations are reduced when applying the proposed method, which verifies that the bias component of the deviation is significantly removed by using the proposed method.

Moreover, we repeat the same experiments 19 times more, and calculate statistics and list the results in TABLE I, where GMEAN denotes the grand mean of clock deviations, that is, the mean of the means of 20 experiment results; MAX, MIN, and MRMS denote the mean of the maximums, minimums, and RMS's of 20 experiments, respectively. These statistics imply that the performance of the proposed method is better than that of the conventional method. We note that all the results from 20 experiments for the same setup show the same sign of the mean values.

V. Conclusion

We proposed a method to compensate for system time drift in order to improve the performance of the DC synchronization in EtherCAT networks. The proposed method actively estimates the drift of each non-reference slave at the master and compensates for both the sign and size of the drift while the conventional method for the drift compensation does not use the information on the drift size. Consequently, the synchronization error in the EtherCAT network is significantly reduced by the proposed method. This was verified by accurately measured experiments, in which the synchronization error was measured at the rising edge of a square wave generated by an external function generator.

References

[1] D. Orfanus, R. Indergaard, G. Prytz, and T. Wien, "EtherCAT-based platform for distributed control in high-performance industrial applications," *IEEE Conf. Emerging Technol. and Factory Autom.*, pp. 1–8, 2013.
[2] G. Prytz, "A performance analysis of EtherCAT and PROFINET IRT," *IEEE Conf. Emerging Technol. and Factory Autom.*, pp. 408–415, 2008.
[3] P. Ferrari, A. Flammini, D. Marioli, and A. Taroni, "Experimental analysis to estimate jitter in PROFINET IO class 1 networks," *IEEE Conf. Emerging Technol. and Factory Autom.*, pp. 429–432, 2006.
[4] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluation of EtherCAT distributed clock performance," *IEEE Trans. Industrial Informatics*, Vol. 8, No. 1, pp. 20–29, 2012.
[5] M. Sung, I. Kim, and T. Kimo, "Toward a Holistic Delay Analysis of EtherCAT Synchronized Control Processes," *Int. Journal of Computers Communications and Control*, Vol. 8, No. 4, pp. 608–621, 2013.
[6] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Performance Evaluation of the EtherCAT Distributed Clock Algorithm," *IEEE Int. Symposium on Industrial Electronics*, pp. 3398–3403, 2010.
[7] X. Xu, Z. Xiong, J. Wu, and X. Zhu, "High-precision time synchronization in real-time Ethernet-based CNC systems," *Int. Journal of Advanced Manufacturing Technology*, Vol. 8, pp. 1157–1170, 2012.
[8] Hardware Data Sheet Section I - ET1100: EtherCAT Slave Controller Beckhoff, 2014.
[9] IgH EtherCAT Master for Linux. [Online] http://www.etherlab.org.
[10] Xenomai. [Online] http://www.xenomai.org.