Received March 9, 2020, accepted March 19, 2020, date of publication March 23, 2020, date of current version April 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2982704

Accuracy Improvement of Master–Slave Synchronization in EtherCAT Networks

SUNG-MUN PARK[1], HYOUNG-WOO KIM[1], HONG-JU KIM[2], AND JOON-YOUNG CHOI[1], (Member, IEEE)
[1]Department of Electronics Engineering, Pusan National University, Busan 46241, South Korea
[2]Korea Electrotechnology Research Institute, Changwon 51543, South Korea

Corresponding author: Joon-Young Choi (jyc@pusan.ac.kr)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2018R1D1A3B07043053.

ABSTRACT In the study, we develop a method to improve the accuracy of master–slave synchronization in EtherCAT networks. The method involves two key compensations that are not considered in the EtherCAT protocol. First, the propagation delay between the master and reference slave is measured, and the system time of the reference slave is then compensated for the measured propagation delay. Second, the bias component of the synchronization error between the master and reference slave is periodically estimated, and system time of each slave is then compensated for the estimated bias component. The entire method is implemented as part of the master application without modifying the EtherCAT protocol or requiring excessive computation load or large memory space. Thus, the developed method is advantageous because it can be immediately applied at low cost to existing EtherCAT networks. By performing extensive experiments based on a Linux-based EtherCAT master, we demonstrate that the developed method significantly improves the accuracy of the master–slave synchronization in EtherCAT networks.

INDEX TERMS Distributed clock, EtherCAT, master–slave synchronization, propagation delay, slave–slave synchronization.

I. INTRODUCTION

Industrial Ethernet networks are widely used for automation applications in place of traditional fieldbus networks because of their various benefits such as cost-effective implementation, flexible topologies, high bandwidth, and wide compatibility [1]–[3]. EtherCAT is a prominent real-time protocol among industrial Ethernet protocols with a desirable feature that each slave processes the addressed data on the fly as the frame moves downstream. Thus, the maximum effective data rate even exceeds 100 Mbps in the full-duplex mode of Fast Ethernet, and fast operation is enabled with cycle times shorter than 100 μs. This property of EtherCAT is employed for very fast control loops in high-end motion control or power electronics applications [4]–[6].

In addition to exhibiting a short cycle time, EtherCAT provides synchronization among the master and slaves based on the distributed clocks (DC) mechanism where the reference slave's clock is optionally adjusted to the master's clock, and other non-reference slaves' clocks are adjusted to the reference slave's clock. Although the DC mechanism enables synchronous operation of devices in distributed system applications, a recent demand has arisen for more precise synchronization performance via distributed real-time precision control applications such as multi-axis motion control systems including robots and computerized numerical control machines [7]–[9].

The slave–slave time synchronization in EtherCAT networks is realized with a high degree of accuracy, as demonstrated by a slave–slave jitter of less than 12 ns in [10]. Additionally, an accuracy improvement method is proposed for slave–slave synchronization in EtherCAT networks, and the performance is demonstrated via experiments in [11].

However, it is observed that EtherCAT suffers from poor accuracy in master–slave synchronization, and two methods have been proposed to improve master–slave synchronization accuracy in [10], [12]. In [12], the cycle period of the EtherCAT master application is adjusted to synchronize each start time of the master and reference slave cycle periods, but the system times of the master and reference slave are not

The associate editor coordinating the review of this manuscript and approving it for publication was Bohui Wang.

considered to be synchronized. In [10], a prototype circuit is developed to achieve high-quality time synchronization between the master and slave devices using precision time control protocol (PTCP) embedded in EtherCAT frames. The approach requires modification of master and slave architectures in EtherCAT networks; thus, it is not feasible to immediately apply the approach to off-the-shelf EtherCAT master and slave devices.

Therefore, it is necessary to develop a new method to overcome the disadvantages of the methods proposed in [10], [12], and obtain a high degree of master–slave synchronization accuracy in EtherCAT networks. Accordingly, we initially investigate the causes of poor accuracy in master–slave synchronization as follows. The EtherCAT protocol does not consider the propagation delay between the master and reference slave, and this causes a significant error in master–slave synchronization [13]. The industrial PC used as the computer platform for running the master degrades synchronization performance via the jitter of critical-frame transmission [10]. Only EtherCAT slaves are equipped with special hardware for time synchronization that is typically not provided for the master [14], [15]. The time control loop (TCL) in the reference slave compensates for the master–slave synchronization error by only a fixed 1 ns correction of the reference slave clock every 10 ns without considering the error magnitude [11].

We carefully consider the characteristic of each investigated cause, and it is observed that the negligence of master–slave propagation delay and master–slave synchronization error magnitude in TCL is directly related to the EtherCAT protocol itself, and they are recognized as two critical disadvantages of the EtherCAT protocol for poor master–slave synchronization. In the study, we focus on the two critical drawbacks of the EtherCAT protocol and develop a new synchronization method to solve the disadvantages and obtain a high degree of synchronization accuracy.

The contribution of the study is summarized as follows. In the initialization state, the master measures the propagation delay between the master and reference slave and then compensates the system time of the reference slave for the measured propagation delay. In the normal operation state, the master periodically measures the synchronization error between the master and reference slave and estimates the bias component in the synchronization error via the exponential moving average (EMA) filter. Subsequently, the master adjusts the system clock of each slave by the amount of the estimated error bias, and this, in turn, compensates for the master–slave synchronization error. The entire method is implemented as part of the master application without requiring an excessive computational load or large memory space. Thus, the developed method exhibits an advantage that it can be immediately applied to existing EtherCAT networks at a low cost.

We perform extensive experiments using EtherCAT networks that consist of the Linux-based master and commercial off-the-shelf EtherCAT slaves. The experimental results demonstrate that the master–slave synchronization performance significantly improves when compared to the performance of the original EtherCAT network, and this verifies the validity and usefulness of the developed method.

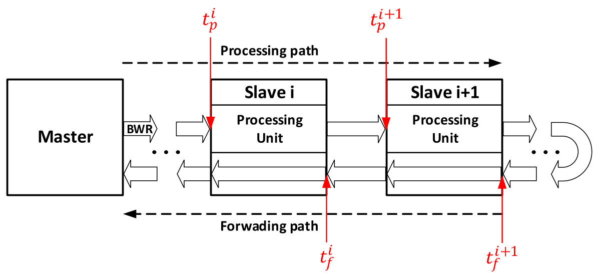


FIGURE 1. Timestamps for propagation delay measurement in EtherCAT networks.

The remainder of the paper is organized as follows. In Section II, we introduce master–slave and slave–slave synchronization of the DC mechanism in the original EtherCAT protocol. Section III describes our development of a new method to improve master–slave synchronization performance. In Section IV, we describe the experiments performed and discuss the experimental results. Finally, we present conclusions in Section V.

II. SYNCHRONIZATION OF EtherCAT DC MECHANISM

In the section, we describe the detailed operation of the DC synchronization mechanism in the EtherCAT protocol, and determine the causes for performance degradation of master–slave synchronization in EtherCAT networks. First, we designate the clocks and times employed by the master and slaves in the DC mechanism. The master is assigned the *system time* by the system clock of the operating system on which the master program runs. The slave is assigned the *local time* by an internal clock, and the slave manages the global clock, the *system time*, that has elapsed since January 1, 2000 and is held in the *System_Time* register in 1 ns units. Specifically, the *reference time* is used to denote the *system time* of the reference slave. The reference slave is typically selected by the DC-enabled slave that is closest to the master.

The synchronization between the master and slaves is performed in three phases as follows: propagation delay measurement, offset compensation, and drift compensation.

A. PROPAGATION DELAY MEASUREMENT

The master triggers propagation delay measurement by sending a broadcast write (BWR) datagram to each slave. When the datagram is received at the port of a slave, the slave records the timestamp and separately stores it in the corresponding *Receive_Time_Port* 0 to 3 slave registers. As shown in a simple network topology depicted in Fig. 1, t_p^i denotes the time at which the BWR datagram arrives at port 0 of the i-th slave on the processing path, and t_f^i denotes the time at which it returns to port 1 of the i-th slave on the forwarding path. The processing direction denotes the direction involving passage

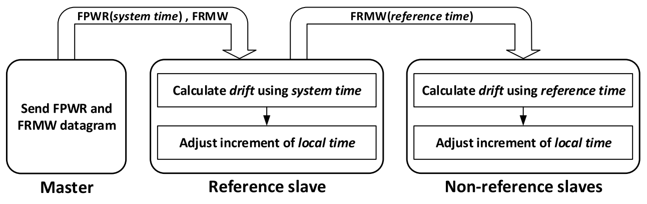

FPWR(*system time*) , FRMW
FRMW(*reference time*)
Send FPWR and FRMW datagram
Calculate *drift* using *system time*
Adjust increment of *local time*
Calculate *drift* using *reference time*
Adjust increment of *local time*
Master
Reference slave
Non-reference slaves

FIGURE 2. **Drift compensation procedure in EtherCAT networks.**

through the EtherCAT processing unit, and the forwarding direction denotes the direction not involving passage through the EtherCAT processing unit.

After each slave completes writing of the timestamps to the *Receive_Time_Port* registers, the master reads the *Receive_Time_Port* register of each slave and collects all the timestamps. Based on the collected timestamps and identified network topology, the master then calculates the propagation delay of each slave between the reference slave and each non-reference slave, and the calculated delay is then written to the *System_Time_Delay* register of each slave [16].

It is noted that the *System_Time_Delay* register of the reference slave is set to zero because the propagation delay of each slave is defined by the data transmission delay between the reference slave and each slave. The propagation delay measurement is performed only once when the network is initialized.

B. OFFSET COMPENSATION

The *offset* of each slave is defined as the difference between the master's *system time* and each slave's *local time* and manifests when each slave is powered up at a different time. The master reads the *local time* of each slave and calculates its *offset* by determining the difference between the master's *system time* and each slave's *local time*, while considering the propagation delay of each slave. After calculating each slave's *offset*, the master writes this value into the *System_Time_Offset* register of each slave.

The process is performed only once when the network is initialized. Based on the *local time* and *offset* stored in the *System_Time_Offset* register, each slave compensates for its offset by calculating the *system time* every 10 ns as follows:

$$t_{sys\_time} = t_{loc\_time} + t_{offset}, \tag{1}$$

where t_{sys_time}, t_{loc_time}, and t_{offset} denote *system time*, *local time*, and *offset* of each slave, respectively.

C. DRIFT COMPENSATION

The *drift* of the reference slave is defined as the difference between the master's *system time* and *reference time*, and the *drift* of a non-reference slave is defined as the difference between the *reference time* and each slave's *system time*. The *drift* of each slave is caused by the jitter of the master clock and frequency differences of oscillators among the slaves. As shown in the *drift* compensation procedure in Fig. 2, the reference slave's *local time* is adjusted to follow the master's *system time*, while the *local time* of each non-reference slave is adjusted to follow the reference slave's *system time*.

To compensate for the *drift*, the master periodically sends configured address physical write (FPWR) and configured address physical read multiple write (FRMW) datagrams. While sending the FPWR and FRMW datagrams, the master also writes its own *system time* into the FPWR datagram, and the reference slave reads the master's *system time* from the FPWR datagram. Simultaneously, the reference slave writes the *reference time* into the FRMW datagram, and each non-reference slave reads the *reference time* from the FRMW datagram. Whenever each slave receives the FPWR and FRMW datagrams, it calculates the *drift* Δ_t using the *offset*, propagation delay, and time received from the FPWR (the master's *system time* for the reference slave) or FRMW (the *reference time* for the non-reference slaves) datagram as follows:

$$\Delta t = (t_{loc\_time} + t_{offset}) - (t_{recv\_time} + t_{prop\_delay}), \tag{2}$$

where t_{recv_time} denotes the time received from the FPWR or FRMW datagram, and t_{prop_delay} denotes the propagation delay stored in the *System_Time_Delay* register of each slave. The calculated *drift* Δt is written into the *System_Time_Difference* register of each slave.

Each slave's *local time* is updated by the TCL every 10 ns based on the value of the *drift*. When the calculated *drift* Δt corresponds to zero, each slave's *local time* is increased by 10 ns every 10 ns. When the calculated *drift* Δt is not zero, based on the sign of each slave's Δt, the non-zero *drift* is compensated as each slave's *local time* is increased by 9 ns every 10 ns for positive Δt and by 11 ns every 10 ns for negative Δt.

The frequency of each slave's *drift* compensation is set by the *Speed_Counter_Start* register, and this determines the time period for the execution cycle of the *drift* compensation and is typically set as significantly higher than 10 ns [11], [16]. It is noted that the *drift* compensation of the reference slave is optional as a configuration condition of the EtherCAT DC synchronization mechanism, and the *drift* compensation of the reference slave must be turned on to ensure that the time synchronization works between the master and all the slaves.

D. CAUSES FOR SYNCHRONIZATION PERFORMANCE DEGRADATION

Based on the analysis results of the previous subsections, we observe two primary causes for the performance degradation of the master–slave synchronization in EtherCAT networks.

First, the propagation delay between the master and reference slave is not measured and is set to zero as stated in Subsection II-B; thus, the reference slave's *offset* is calculated without considering the propagation delay between them. The negligence results in a significant synchronization error between the master and reference slave because the reference slave applies (1) and (2) with zero propagation delay. Furthermore, the master–reference slave synchronization error inevitably leads to another significant synchronization error between the master and each non-reference slave because non-reference slaves attempt to synchronize with the *reference time* via *drift* compensation.

Second, as stated in Subsection II-C, the *drift* compensation of each slave considers only the sign of the *drift*, and this can decrease its effect. The effect of the *drift* compensation is maximized by considering the size as well as the sign of the *drift* such that the correction size of each slave's *local time* should not be fixed at 1 ns every 10 ns but a variable time size every 10 ns based on the size of its *drift*.

III. METHOD FOR IMPROVING MASTER–SLAVE SYNCHRONIZATION ACCURACY

First, we observe the actual experimental results in Figs. 8(a) and 8(b) that demonstrate the distribution of the synchronization errors between the master and reference slave in conventional EtherCAT networks with two and six slaves. As shown in Figs. 8(a) and 8(b), it is clearly observed that the sample mean is not near zero, and samples exhibit significant bias components although the DC mechanism of the EtherCAT protocol works normally, and this motivates the development of a method for improving the master–slave synchronization accuracy.

Based on the analysis results in Subsection II-D, it is concluded that the two main causes of the bias components in Figs. 8(a) and 8(b) are related to the negligence of the propagation delay between the master and reference slave and size of each slave's *drift* in the *drift* compensation. We develop two compensation methods in the following subsections to solve the aforementioned problems and improve master–slave synchronization performance.

A. COMPENSATION FOR PROPAGATION DELAY BETWEEN MASTER AND REFERENCE SLAVE

We develop a method to compensate for the propagation delay between the master and reference slave that initially requires the propagation delay to be measured. In the master application, we consider the timestamp, t_{p0}^{m}, immediately after sending a datagram to the reference slave, and the timestamp, t_{p1}^{m}, immediately after datagram is returned to the master. Subsequently, the master reads the values of the *Receive_Time_Port* 0 register, t_{p0}^{r}, and *Receive_Time_Port* 1 register, t_{p1}^{r}, from the reference slave and calculates the propagation delay, $t_{pd}^{m,r}$, as follows:

$$t_{pd}^{m,r} = \frac{(t_{p1}^{m} - t_{p0}^{m}) - (t_{p1}^{r} - t_{p0}^{r})}{2}, \tag{3}$$

the derivation of which is illustrated in Fig. 3. The propagation delay between the master and reference slave is not measured by a constant value and instead varies at each measurement; thus, it is necessary to appropriately deal with the time-varying characteristic of the propagation delay in the measurement [17], [18]. Hence, the procedure of the

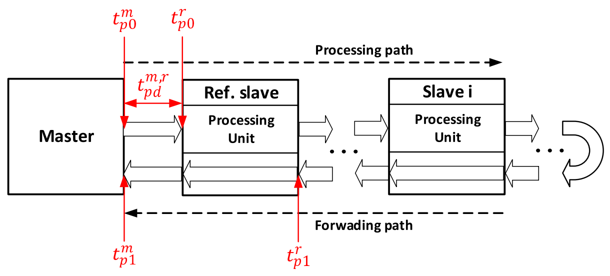


FIGURE 3. Measurement of master–reference slave propagation delay.

propagation delay measurement is repeated 1,000 times, and the averaged propagation delay is selected as the propagation delay measurement between the master and reference slave.

Subsequently, the measured propagation delay between the master and reference slave is compensated for in the normal operation state of the master application as follows. Whenever the FPWR datagram containing the master's *system time* is periodically sent to the reference slave, the master calculates the compensated *system time* by adding the present *system time* and measured propagation delay and sends the compensated *system time* as opposed to the present *system time*. While receiving the FPWR datagram, the reference slave reads the master's compensated *system time* from the FPWR datagram as t_{recv_time} in (2). Hence, when *drift* compensation is performed in the reference slave, compensation for the propagation delay is simultaneously performed because the reference slave attempts to compensate for the *drift* calculated from (2), where t_{recv_time} already contains the propagation delay.

In summary, without modification of the original EtherCAT protocol, compensation for the propagation delay is realized between the master and reference slave via sending of the compensated *system time* of the master to the reference slave. The overall compensation procedure is shown in the flow chart in Fig. 4. Furthermore, another desired result is realized wherein the synchronization performance between the master and each non-reference slave also improves because non-reference slaves perform *drift* compensation based on the *reference time* that has been already compensated for the propagation delay between the master and reference slave.

B. REDUCTION IN SYNCHRONIZATION ERROR

We develop a method to overcome the disadvantage of *drift* compensation in EtherCAT networks and decrease the synchronization error between the master and each slave. Based on the observation in Figs. 8(a) and 8(b), the bias component of the synchronization error is considered as a measurement of the synchronization error and used for feedback information on the synchronization error. With the selection of the bias as the feedback signal to represent the synchronization error, we consider an approach where the master periodically estimates the bias of the synchronization error between the master and reference slave and uses the estimated bias for

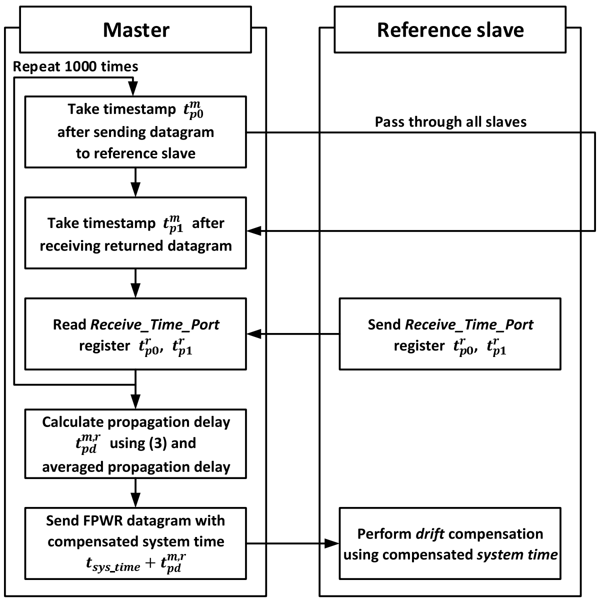

Master
Reference slave
Repeat 1000 times
Take timestamp t_{p0}^{m} after sending datagram to reference slave
Pass through all slaves
Take timestamp t_{p1}^{m} after receiving returned datagram
Read *Receive_Time_Port* register t_{p0}^{r}, t_{p1}^{r}
Send *Receive_Time_Port* register t_{p0}^{r}, t_{p1}^{r}
Calculate propagation delay $t_{pd}^{m,r}$ using (3) and averaged propagation delay
Send FPWR datagram with compensated system time $t_{sys\_time} + t_{pd}^{m,r}$
Perform *drift* compensation using compensated *system time*

FIGURE 4. Compensation procedure for master–reference slave propagation delay.

bias compensations of the non-reference slaves and reference slave.

The approach is justified by the well-known property that the non-reference slaves synchronize with the reference slave in EtherCAT networks with a high degree of accuracy via the DC mechanism [10]. The aforementioned property prompts the concept wherein the bias estimate of only the master-reference slave synchronization error is sufficient to compensate for synchronization errors of the non-reference slaves as well as the reference slave without performance degradation. The procedure to estimate the bias of the master–reference slave synchronization error is outlined as the master periodically measures synchronization error between the master and reference slave; a low pass filter (LPF) is applied to the measured errors; and the output of the LPF is selected as the bias estimate for bias compensations of all the slaves.

The detailed procedure of the overall developed method is described as follows. The master measures the synchronization error between the master and reference slave by reading the *System_Time_Difference* register of the reference slave because it is periodically updated with a new time difference between the master and reference slave based on the DC mechanism. We suppose that Δt^k is the master–slave synchronization error at the k-th sampling time, i.e., the value of the *System_Time_Difference* register of the reference slave read at the k-th sampling time.

We employ the following EMA filter as an appropriate LPF to estimate the bias component from the master–reference slave synchronization error samples:

$$\Delta T^k = \alpha \Delta t^k + (1-\alpha)\Delta T^{k-1}, \tag{4}$$

where ΔT^k denotes the output of the EMA filter for the master–reference slave synchronization error at the k-th sampling time, the initial condition is set as $\Delta T^0 = \Delta t^0$, and α is a designed constant smoothing factor such as $0 < \alpha < 1$.

Subsequently, we design a method to send the estimated bias back to each slave and correct its *system time* by exploiting the role of each slave's *offset* that is used in the periodic correction of the *system time* at each slave as described in the *offset* compensation (1). Hence, the master reads the *offset* stored in the *System_Time_Offset* register of each slave and modifies it as follows:

$$t_{new\_offset} = t_{offset} - \Delta T^k, \tag{5}$$

where t_{new_offset} denotes the new *offset* compensated for the bias of the master–slave synchronization error. The master then writes the new *offset* back to the *System_Time_Offset* register of each slave, and the *system time* of each slave is effectively compensated for the bias component whenever the *offset* compensation in each slave is performed based on (1).

By periodically repeating the procedure, the developed method compensates for the bias of the synchronization error between the master and each slave only using the bias estimate of the master–reference slave synchronization error. Thus, the compensation for the bias of each slave's synchronization error significantly decreases the master–slave synchronization error and improves synchronization accuracy between the master and each slave. The overall procedure of the developed method is shown in the flow chart in Fig. 5.

IV. EXPERIMENT

A. SETUP

We perform extensive experiments to verify the synchronization performance of the developed method using the IgH EtherCAT master stack [19], which is a Linux-based open-source EtherCAT master. The experimental setup is constructed using commonly available commercial off-the-shelf devices, as shown in Fig. 6. The EtherCAT master runs on a SeeedStudio's BeagleBone Green board based on Linux kernel 4.9.59 with the RT patch, and EtherCAT slaves run on TI's TMDSICE3359 boards and Infineon's XMC4800 EtherCAT kits.

We consider two network configurations for the experiments: one is configured with a master and two TI slaves, and the other is configured with a master, two TI slaves, and four Infineon slaves. A function generator is connected to each general purpose input (GPI) of the master, reference slave, and last slave to measure the synchronization error between the master and each slave. We measure synchronization errors of only the reference and last slaves while excluding intermediate slaves because the synchronization error between the master and last slave is the worst [13]. Thus, the intermediate slaves always guarantee better synchronization performance than the last slave. A PC is used to analyze the experimental results.

B. MEASUREMENT OF SYNCHRONIZATION ERRORS

With respect to the synchronization error measurement, it is necessary to simultaneously obtain the *system times* of the

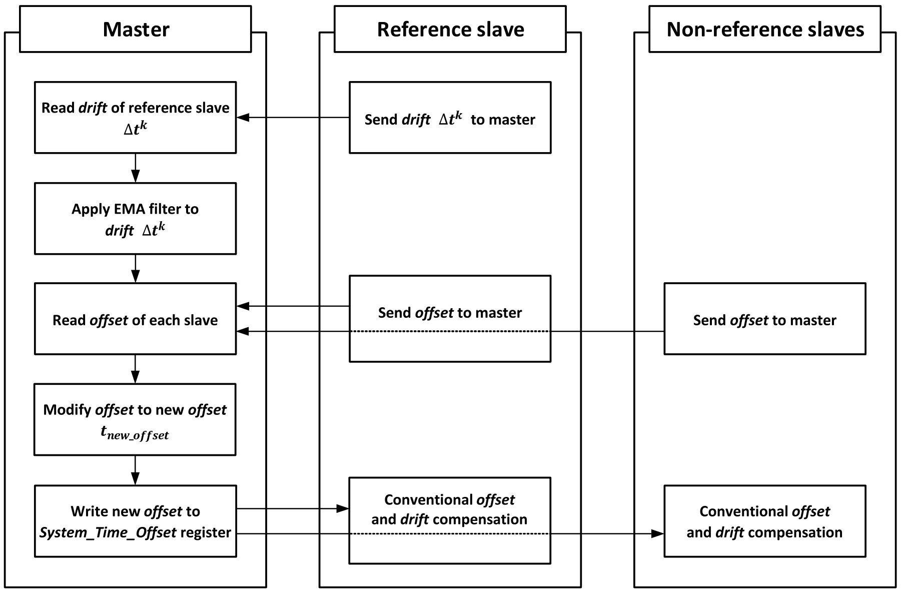

Master
Reference slave
Non-reference slaves
Read *drift* of reference slave Δt^k
Send *drift* Δt^k to master
Apply EMA filter to *drift* Δt^k
Read *offset* of each slave
Send *offset* to master
Send *offset* to master
Modify *offset* to new *offset* t_{new_offset}
Write new *offset* to *System_Time_Offset* register
Conventional *offset* and *drift* compensation
Conventional *offset* and *drift* compensation

FIGURE 5. Procedure for synchronization error compensation.

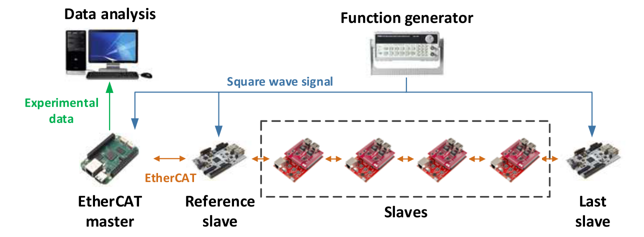


FIGURE 6. Setup for experiments.

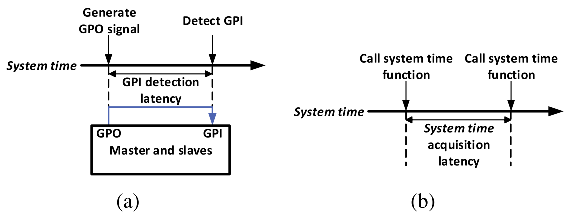


FIGURE 7. Measurement of latencies: (a) GPI detection latency (b) *system time* acquisition latency.

master, reference slave, and last slave and calculate the *system time* differences between the master and reference slave and between the master and last slave, which are considered as synchronization errors of the reference slave and last slave, respectively.

The measurement procedure is described in detail as follows. The *system times* are obtained by calling the system time functions of Linux and TI-RTOS for the master and slaves, respectively, and the simultaneous acquisition is obtained by calling the system time function in synchronization with the rising edge GPI interrupt from the external square wave signal provided by the external function generator. Subsequently, as soon as the system time function returns with each *system time* of the master and slave, the master stores the master *system time* in memory while the slaves send the slave *system times* to the master at each operation cycle, and the sent slave *system times* are stored in memory by the master. After completion of each experiment round, the *system times* of the master, reference slave, and last slave stored at each operation cycle are saved in an SD card, and the saved experimental results are transferred and analyzed on the analysis PC [11], [13].

In the procedure, in order to more precisely measure the synchronization errors, we consider two types of latencies: the GPI detection latency and *system time* acquisition latency, which represent the time required from the GPI signal input to the GPI interrupt occurrence, and time required from the call of the system time function to its return with *system time*, respectively. Hence, we estimate and compensate for the two latencies when analyzing the experimental results.

As shown in Fig. 7(a), the GPI detection latency is estimated by connecting a general purpose output (GPO) and a GPI in each master and slave and measuring the time difference from the GPO signal generation to the GPI interrupt. The measurement operation is repeated 1,000 times, and the average *system time* difference is selected as the estimate of the GPI detection latency. The GPI detection latencies of the master on Linux and slaves on TI-RTOS are estimated as 13.71 µs and 0.95 µs, respectively.

As shown in Fig. 7(b), the *system time* acquisition latency is estimated by measuring the time difference between two consecutive calls of the system time function in each master and slave. The system time function is called 1,000 times, and the average time difference of 999 time differences between the two consecutive calls of the system time function is selected as the estimate of the *system time* acquisition latency. The *system time* acquisition latencies of the master on Linux

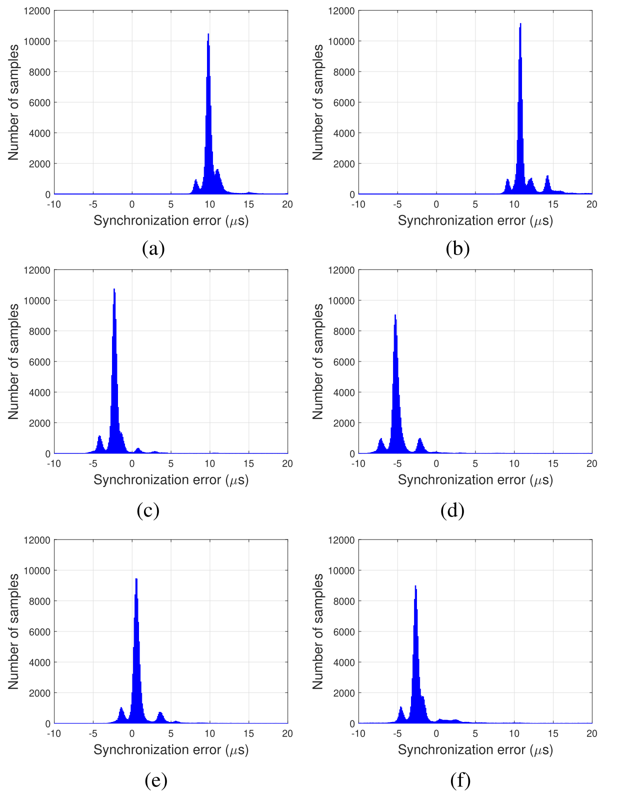


FIGURE 8. **Distribution of synchronization errors between master and reference slave: (a) Conventional method (two slaves) (b) Conventional method (six slaves) (c) Compensation for propagation delay (two slaves) (d) Compensation for propagation delay (six slaves) (e) Compensation for propagation delay and bias (two slaves) (f) Compensation for propagation delay and bias (six slaves).**

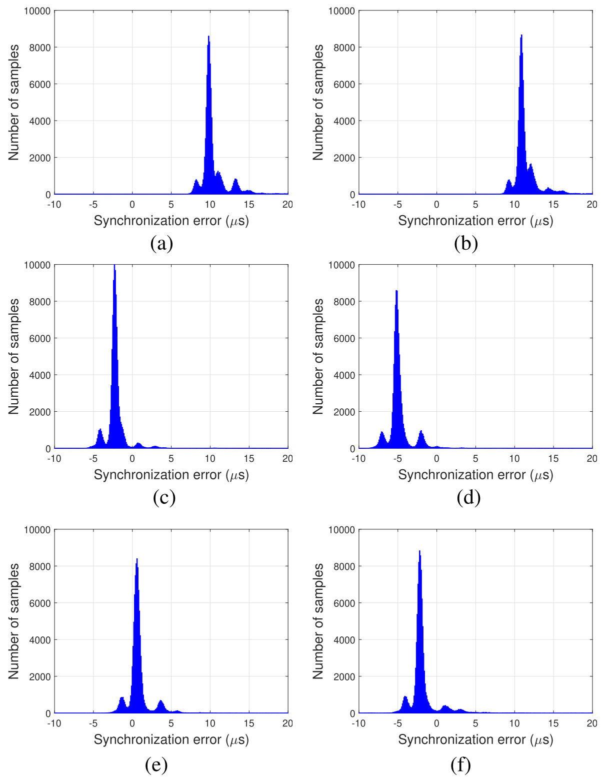


FIGURE 9. **Distribution of synchronization errors between master and last slave: (a) Conventional method (two slaves) (b) Conventional method (six slaves) (c) Compensation for propagation delay (two slaves) (d) Compensation for propagation delay (six slaves) (e) Compensation for propagation delay and bias (two slaves) (f) Compensation for propagation delay and bias (six slaves).**

and slaves on TI-RTOS are estimated as 0.28 μs and 0.49 μs, respectively.

In the step of analyzing the experimental results on the analysis PC, the two estimated latencies are subtracted from the obtained *system times* of the master, reference slave, and last slave, and this results in a more precise measurement of synchronization errors between the master and each slave.

The master operation cycle is set to 1 ms, which implies that the master sends one packet every 1 ms. The period of the external square wave signal is set to 2 ms. For an experiment round, 100,000 samples are obtained after waiting for 5 min for the system to stabilize, and cold reset is initially performed to minimize the effect of the previous experiment. The smoothing factor of the EMA filter for the synchronization error estimation is set as $\alpha = 0.25$, and the operating cycle of the developed method for the synchronization error reduction is set to 100 ms.

C. RESULT

We perform the experiments for three cases as follows: conventional method, compensation for only the propagation delay between the master and reference slave, and compensation for both the propagation delay and bias. We use the experimental results and compare their synchronization performances and verify the improvement in accuracy owing to the developed method.

Figs. 8 and 9 show the distributions of all the measured samples of synchronization errors between the master and reference slave and between the master and last slave, respectively. We compare Figs. 8(a), 8(c), with Figs. 8(b), 8(d) and Figs. 9(a), 9(c) with Figs. 9(b), 9(d), and it is observed that the distribution center in the case of the propagation delay compensation is closer to zero. This verifies that the synchronization accuracy is significantly improved between the master and each slave even when applying the compensation for only the propagation delay. However, it is also observed that an appreciable deviation from zero still remains. Additionally, we compare Figs. 8(c), 8(e), with Figs. 8(d), 8(f) and Figs. 9(c), 9(e) with Figs. 9(d), 9(f), and it is observed that the distribution center in the case of both the propagation delay and bias compensation is significantly closer to zero, which verifies that the bias compensation considerably improves synchronization accuracy and is essential to improve the fine synchronization performance.

Figs. 10 and 11 show the mean and root mean square (RMS) of the synchronization errors up to each sampling instant between the master and reference slave and between the master and last slave, respectively. Comparing Figs. 10(a), 10(e) with Figs. 10(b), 10(f) and

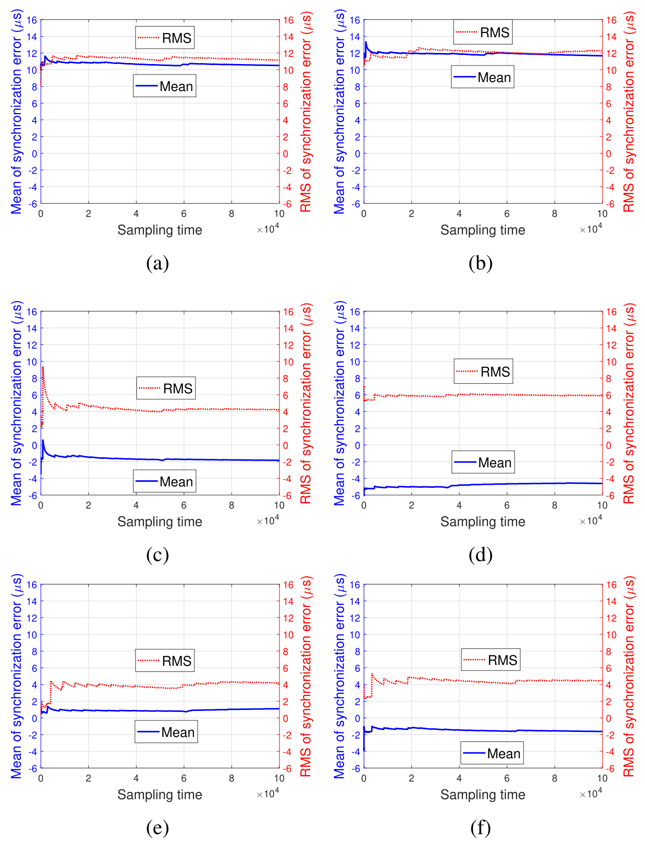


FIGURE 10. Mean and RMS of synchronization errors between master and reference slave up to each sampling time: (a) Conventional method (two slaves) (b) Conventional method (six slaves) (c) Compensation for propagation delay (two slaves) (d) Compensation for propagation delay (six slaves) (e) Compensation for propagation delay and bias (two slaves) (f) Compensation for propagation delay and bias (six slaves).

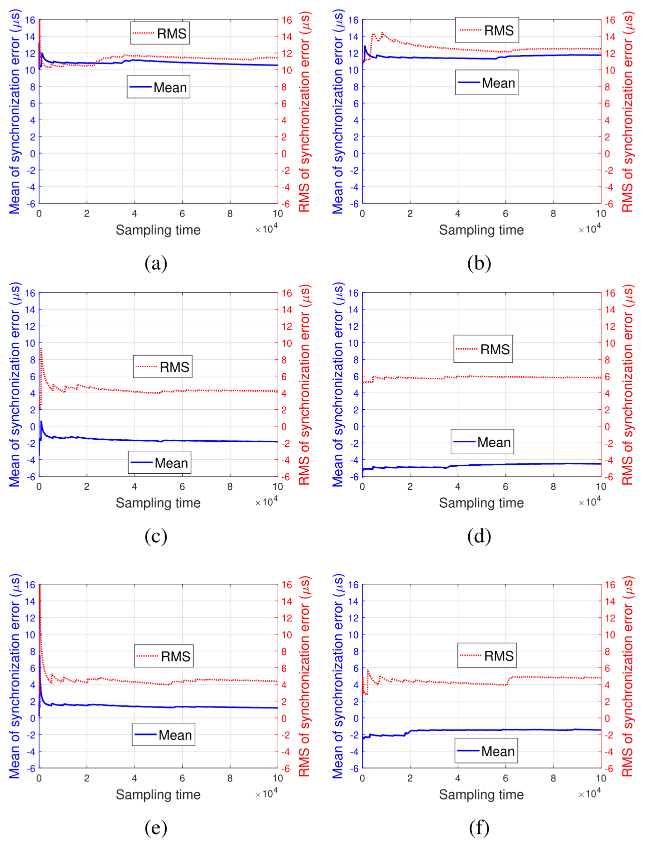


FIGURE 11. Mean and RMS of synchronization errors between master and last slave up to each sampling time: (a) Conventional method (two slaves) (b) Conventional method (six slaves) (c) Compensation for propagation delay (two slaves) (d) Compensation for propagation delay (six slaves) (e) Compensation for propagation delay and bias (two slaves) (f) Compensation for propagation delay and bias (six slaves).

TABLE 1. Statistical properties of synchronization errors between master and slaves.

Number of slaves	Statistics	Synchronization errors (μs)			
		Master—Reference slave		Master—Last slave	
		Conventional	Developed	Conventional	Developed
2	GMEAN	10.58	1.06	10.57	1.08
	MRMS	11.23	4.19	11.24	4.24
6	GMEAN	11.67	-1.71	11.74	-1.59
	MRMS	12.29	4.55	12.35	4.59

Figs. 11(a), 11(e) with Figs. 11(b), 11(f), it is observed that the mean and RMS of the synchronization errors sharply decrease for the developed method when compared to that of the conventional method, and this also verifies that the synchronization accuracy significantly improves between the master and each slave. Additionally, Figs. 10 and 11 demonstrate a stable dynamic operation of the developed method with a fast time response because the mean and RMS rapidly reach steady states through the initial transient states.

To further demonstrate the developed method, we repeat each round of experiment nine more times for the conventional method and compensation for both propagation delay and bias and calculate statistics, as summarized in Table 1. In Table 1, GMEAN denotes the grand mean of synchronization errors, i.e., the mean of all the means of the 10 experimental results and MRMS denotes the mean of the RMS values of the 10 experiments. The statistics also demonstrate that the developed method significantly improves synchronization performance when compared to the conventional method.

V. CONCLUSION

In the study, we developed a method to improve synchronization accuracy between the master and each slave in EtherCAT networks. The propagation delay was measured between the master and reference slave, and the *system time* of the reference slave was then compensated for the measured propagation delay. The bias component of the synchronization error between the master and reference slave was estimated, and the estimated bias was then used to compensate for the bias of the synchronization error between the master and each slave. The compensations for both the propagation delay and bias component of the synchronization error effectively decreased the synchronization error and significantly improved synchronization accuracy between the master and each slave.

The developed method was implemented as part of the master application without modification of the original

EtherCAT protocol or requirement of an excessive computational load or large memory space. Thus, the developed method exhibits an advantage that it can be immediately applied to existing EtherCAT networks at a low cost. We constructed the experimental setup using commercial off-the-shelf devices for EtherCAT networks, and the experimental results verified that the developed method significantly improves synchronization accuracy between the master and each slave when compared to the conventional method.

On the other hand, the synchronization performance of the developed method inevitably depends on the jitter of the master *system time* because each slave *system time* is synchronized to the master *system time*. Therefore, in addition to the developed method, proper management to reduce jitter in the system time of the operating system running the master will also be effective in improving master–slave synchronization in EtherCAT networks.

REFERENCES

[1] P. Ferrari, A. Flammini, D. Marioli, A. Taroni, and F. Venturini, "Experimental analysis to estimate jitter in PROFINET IO class 1 networks," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom.*, Sep. 2006, pp. 429–432.

[2] G. S. Sestito, A. C. Turcato, A. L. Dias, M. S. Rocha, M. M. da Silva, P. Ferrari, and D. Brandao, "A method for anomalies detection in real-time Ethernet data traffic applied to PROFINET," *IEEE Trans. Ind. Informat.*, vol. 14, no. 5, pp. 2171–2180, May 2018.

[3] K. O. Akpinar and I. Ozcelik, "Analysis of machine learning methods in EtherCAT-based anomaly detection," *IEEE Access*, vol. 7, pp. 184365–184374, Dec. 2019.

[4] G. Prytz, "A performance analysis of EtherCAT and PROFINET IRT," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Autom.*, Sep. 2008, pp. 408–415.

[5] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "On the accuracy of the distributed clock mechanism in EtherCAT," in *Proc. IEEE Int. Workshop Factory Commun. Syst.*, May 2010, pp. 43–52.

[6] H. Kang, K. Kim, and H.-W. Jin, "Real-time software pipelining for multidomain motion controllers," *IEEE Trans. Ind. Informat.*, vol. 12, no. 2, pp. 705–715, Apr. 2016.

[7] L. Li, P. Cong, K. Cao, J. Zhou, T. Wei, M. Chen, S. Hu, and X. S. Hu, "Game theoretic feedback control for reliability enhancement of EtherCAT-based networked systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1599–1610, Sep. 2019.

[8] D. Macii, D. Fontanelli, and D. Petri, "A master-slave synchronization model for enhanced servo clock design," in *Proc. Int. Symp. Precis. Clock Synchronization Meas., Control Commun.*, Oct. 2009, pp. 1–6.

[9] X. Xu, Z. Xiong, J. Wu, and X. Zhu, "High-precision time synchronization in real-time Ethernet-based CNC systems," *Int. J. Adv. Manuf. Technol.*, vol. 65, nos. 5–8, pp. 1157–1170, Mar. 2013.

[10] D. Ganz, S. Leschke, and H. Dermot Doran, "Improving EtherCAT master-slave synchronization precision using PTCP embedded in EtherCAT frames: A proof-of-concept," in *Proc. IEEE World Conf. Factory Commun. Syst. (WFCS)*, May 2015, pp. 1–7.

[11] S.-M. Park, H. Kim, H.-W. Kim, C. N. Cho, and J.-Y. Choi, "Synchronization improvement of distributed clocks in EtherCAT networks," *IEEE Commun. Lett.*, vol. 21, no. 6, pp. 1277–1280, Jun. 2017.

[12] B. Li, H. Lin, S. Sun, and L. Zheng, "A synchronization method for local applications of EtherCAT master-slave in open CNC system," in *Proc. IEEE Int. Conf. Inf. Autom.*, Aug. 2018, pp. 527–533.

[13] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluation of EtherCAT distributed clock performance," *IEEE Trans. Ind. Informat.*, vol. 8, no. 1, pp. 20–29, Feb. 2012.

[14] D. Orfanus, R. Indergaard, G. Prytz, and T. Wien, "EtherCAT-based platform for distributed control in high-performance industrial applications," in *Proc. IEEE 18th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2013, pp. 1–8.

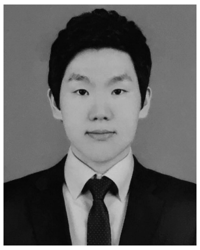
[15] X. Chen, D. Li, S. Wang, H. Tang, and C. Liu, "Frequency-tracking clock servo for time synchronization in networked motion control systems," *IEEE Access*, vol. 5, pp. 11606–11614, Jul. 2017.

[16] *Hardware Data Sheet Section I-EtherCAT Slave Controller, Ver. 2.2*, Beckhoff, Verl, Germany, 2014, pp. 50–55.

[17] C. Deng, M. J. Er, G.-H. Yang, and N. Wang, "Event-triggered consensus of linear multiagent systems with time-varying communication delays," *IEEE Trans. Cybern.*, early access, Aug. 12, 2019, doi: 10.1109/TCYB.2019.2922740.

[18] B. Wang, W. Chen, B. Zhang, and Y. Zhao, "Regulation cooperative control for heterogeneous uncertain chaotic systems with time delay: A synchronization errors estimation framework," *Automatica*, vol. 108, Oct. 2019, Art. no. 108486.

[19] IGH. *IgH EtherCAT Master for Linux*. Accessed: Mar. 24, 2020. [Online]. Available: http://etherlab.org



SUNG-MUN PARK received the B.S. and M.S. degrees in electronic and electrical engineering from Pusan National University, Busan, South Korea, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree in electrical and computing engineering. His research interests include embedded systems and motor drive control.

HYOUNG-WOO KIM received the B.S. and M.S. degrees in electronic and electrical engineering from Pusan National University, Busan, South Korea, in 2012 and 2015, respectively, where he is currently pursuing the Ph.D. degree in electrical and computing engineering. His research interests include nonlinear control, motor drive control, and embedded systems.

HONG-JU KIM received the B.S., M.S., and Ph.D. degrees in electronic and electrical engineering from the Pohang University of Science and Technology (POSTECH), Pohang, South Korea, in 1996, 1998, and 2003, respectively. He is currently a Senior Researcher with the Korea Electrotechnology Research Institute, Changwon, South Korea. His research interests include machining tools and smart manufacturing.

JOON-YOUNG CHOI (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronic and electrical engineering from the Pohang University of Science and Technology (POSTECH), Pohang, South Korea, in 1994, 1996, and 2002, respectively. From 2002 to 2004, he worked as a Senior Engineer with the Electronics and Telecommunication Research Institute (ETRI), Daejeon, South Korea. From 2004 to 2005, he worked as a Visiting Associate with the Department of Computer Science and Electrical Engineering, California Institute of Technology (CALTECH), Pasadena, CA, USA. He is currently a Professor with the Department of Electronics Engineering, Pusan National University. His research interests include nonlinear control, embedded systems, and automation.

• • •