LETTER

Time Synchronization Method for ARM-Based Distributed Embedded Linux Systems Using CCNT Register*

Young-Woo KWON†, Sung-Mun PARK†, Nonmembers, and Joon-Young CHOI†, Member

SUMMARY We propose a system time synchronization method between ARM-based embedded Linux systems. The master Linux with reference clock sends its own system time to the slave Linux via Transmission Control Protocol communication along with a general-purpose input/output (GPIO) signal, and then the slave Linux corrects its own system time by the difference between its own system time at receiving the GPIO signal and the received reference time. The synchronization performance is significantly improved by compensating for the GPIO signal detection latency and the system time acquisition and setting latencies in Linux. These latencies are precisely measured by exploiting the function of Cycle Counter register in ARM coprocessor. Extensive experiments are performed with two ARM-based embedded Linux systems, and the results demonstrate the validity and performance of the proposed synchronization method.

key words: ARM processor, distributed system, embedded Linux, time synchronization

1. Introduction

Embedded systems are becoming increasingly complicated and distributed to meet the requirements of various fields, such as industrial automation, measurement, and control. Embedded systems in distributed environments are required to operate in real time under strict timing constraints and need to operate accurately at a given time just as a single system operates. To satisfy this requirement, a reference time is established for all distributed systems in common, and the system time of each distributed system must be accurately synchronized to the reference time [1], [2].

The existing time synchronization methods for Linux systems are found with the Network Time Protocol (NTP) and Precision Time Protocol (PTP). The NTP is widely used for the time synchronization of personal and server computers on the Internet, but supports only millisecond synchronization accuracy, which is not sufficient for the precise industrial applications [3]. On the other hand, the PTP theoretically guarantees synchronization accuracy from microseconds to nanoseconds, and is used for industrial Ethernet such as EtherCAT and TTEthernet. However, the PTP shows the disadvantages of requiring the time-stamping function of event messages and dedicated PTP Hardware Clock (PHC). Furthermore, because the PHC cannot be used directly as the operating system clock, an additional synchronization process is required between the operating system clock and the PHC in each distributed system in order to synchronize the operating system clock of each distributed system with the reference clock. This additional synchronization process may a considerable deterioration of synchronization accuracy [4], [5].

In order to overcome the disadvantages of the existing synchronization methods, we propose a synchronization method for ARM-based embedded Linux systems by using general-purpose input/output (GPIO) device and Transmission Control Protocol (TCP) communication. The master Linux with reference clock triggers the synchronization by sending its own system time to the slave Linux via TCP along with a GPIO signal. Then, the slave Linux synchronizes with the master Linux by correcting its own system time by the difference between its own system time at receiving the GPIO signal and the received reference time. The synchronization performance is significantly improved by compensating for the general-purpose input (GPIO) detection latency and the system time acquisition and setting latencies in Linux that are precisely measured by exploiting the Cycle Counter (CCNT) register of ARM coprocessor.

We perform extensive experiments using a setup consisting of two ARM-based embedded Linux systems. The experimental results verify the validity of the proposed method and demonstrate that the synchronization accuracy is achieved with mean error sizes of 0.33 μs, which is a sufficient performance for practical applications. Furthermore, compared to the existing PTP, the proposed method shows the advantages of requiring no additional dedicated PHC and directly synchronizing the main system times of distributed Linux systems.

2. Latency Measurements

The overall system architecture for the proposed method is depicted in Fig. 1, where the master Linux with reference clock is connected to the slave Linux via GPIO ports and Ethernet for TCP connection. To improve synchronization performance, it is necessary to measure and compensate for the GPIO detection latency and system time acquisition and setting latencies in distributed embedded Linux systems. The GPIO detection latency is defined by the time required from the GPIO signal input to the GPIO interrupt generation, and the system time acquisition and setting latencies are de-

*a) E-mail: jyc@pusan.ac.kr
DOI: 10.1587/transinf.2020EDL8094
Fig. 1 System architecture consisting of master Linux and slave Linux

Fig. 2 Timing diagram to measure system call latency

Fig. 3 Timing diagram to measure GPI detection latency

2.1 Latency Measurement for Reading CCNT Register

Although the CCNT register is a high-frequency clock source, it also brings about the reading latency, which must be compensated for to achieve an accurate time measurement. Hence, it first needs to measure the reading latency of the CCNT register, and we develop the following procedure. Sequentially executing the reading of the CCNT register 100,000 times, we collect the 100,000 consecutive CCNT register samples and calculate the 99,999 CCNT register difference samples between each two consecutive CCNT register samples. Then, the average of the 99,999 samples is selected as the reading latency of the CCNT register, which is denoted as \( \text{AVG}(t_C) \).

2.2 Latency Measurements for System Time Acquisition and Setting

The system calls, \texttt{gettimeofday} and \texttt{do_settimeofday}, are used to acquire and set the system time of Linux, respectively. To improve the synchronization accuracy, it needs to measure and compensate for the system time acquisition latency, \( t_A \), and the system time setting latency, \( t_S \), and we develop the following procedure to measure these latencies using the CCNT register.

As shown in Fig. 2, the system time acquisition and setting latencies are measured by reading the CCNT register immediately before and after the system calls and calculating

\[
\begin{align*}
 t_A &= (t_2 - \text{AVG}(t_C)) - t_1 \quad \text{(1)} \\
 t_S &= (t_2 - \text{AVG}(t_C)) - t_1, \quad \text{(2)}
\end{align*}
\]

where \( t_1 \) and \( t_2 \) denote the times obtained from the CCNT register immediately before and after the system calls, respectively. This procedure is repeated 100,000 times for each system call and the averaged values, \( \text{AVG}(t_A) \) and \( \text{AVG}(t_S) \), are selected as the measurements of the system time acquisition and setting latencies.

2.3 Latency Measurement for GPI Detection

Figure 3 shows the timing diagram to measure the GPI detection latency, \( t_G \), using the CCNT register. The slave measures the time, \( t_1 \), by reading the CCNT register, and immediately sends the general-purpose output (GPO) signal to the master. Then, when the master receives the GPI signal, an interrupt is generated and immediately sends the GPO signal back to the slave. When the slave receives the GPI signal, an interrupt is generated and the time, \( t_2 \), is read from the CCNT register. Finally, the GPI detection latency is calculated by

\[
 t_G = \frac{(t_2 - \text{AVG}(t_C)) - t_1}{2}. \quad \text{(3)}
\]

This procedure is repeated 100,000 times, and the averaged value, \( \text{AVG}(t_G) \), is selected as the measurement of the GPI detection latency.

3. Time Synchronization Method

The basic idea of the proposed synchronization method is shown in Fig. 4. The master acquires the reference time, the system time of the master Linux, at the \( k \)-th operation of synchronization, \( t_{ma} \), and sends the reference time via TCP to the slave along with a GPIO signal. Then, when the slave receives the GPIO signal, an interrupt is generated and immediately sends the GPO signal back to the master. When the master receives the GPIO signal, an interrupt is generated and immediately sends the GPO signal back to the slave. When the slave receives the GPIO signal, an interrupt is generated and the time, \( t_2 \), is read from the CCNT register. Finally, the GPI detection latency is calculated by

\[
 t_G = \frac{(t_2 - \text{AVG}(t_C)) - t_1}{2}. \quad \text{(3)}
\]

This procedure is repeated 100,000 times, and the averaged value, \( \text{AVG}(t_G) \), is selected as the measurement of the GPI detection latency.
Δtk = tma_k - (tksl_1 - AVG(tG) - AVG(tA)). (4)

This measured time difference between the master and slave can be directly used to correct the slave system time for synchronization, but we apply the following Exponential Moving Average (EMA) filter to the measured time difference in order to effectively eliminate the high-frequency noise component and make the operation stable and smooth:

\[ \Delta T^k = \alpha \Delta t^k + (1 - \alpha) \Delta T^{k-1}, \]

where \( \Delta T^k \) denotes the EMA filter output at the \( k \)-th operation, that is, the filtered time difference between the master and slave, the initial condition is set as \( \Delta T^0 = \Delta t^0 \), and \( \alpha \) is a smoothing factor selected as \( 0 < \alpha < 1 \). [6]

Subsequently, the current slave system time, \( t_{sl}^k \), is updated with new system time, \( T^k_{sl} \), by compensating for the filtered time difference considering the system time setting latency as

\[ T^k_{sl} = t_{sl}^k + \Delta T^k + AVG(tS). \]

which achieves the synchronization between the master and slave. This update of the slave system time is illustrated in Fig. 5. Furthermore, this procedure is periodically performed to maintain an accurate synchronization even in the presence of sudden system parameter variations due to clock jitters or oscillator frequency deviations in the master and slave Linux. The overall procedure of the proposed synchronization method is shown in the flow chart in Fig. 6 [7].

### 4. Experiment

#### 4.1 Setup

We perform extensive experiments to verify the synchronization performance of the proposed method. The experimental setup is built with two BeagleBone Black boards based on the AM3358 processor and Linux kernel 4.9.59 with RT patch. Two BeagleBone Black boards are used for the master and slave, and connected via GPIO ports and Ethernet for TCP. The master periodically sends the acquired master system time to the slave along with a GPO signal every 1 ms cycle, i.e., the operation cycle of the proposed method is set to 1 ms.

The synchronization performance is accurately evaluated by using an oscilloscope measuring GPO signals from each board. Each board is programmed to generate a GPO signal every 100 ms by a timer interrupt based on each system time, and the oscilloscope simultaneously measures the GPO signals from two boards and the time differences represent the synchronization errors. In each round of experiment, each board is completely initialized with a random system time.

First, in order to find the optimal smoothing factor of the EMA filter, we perform the experiments for various values of \( \alpha \), where 5,000 samples of synchronization errors are collected. Second, ten rounds of experiments are performed for the proposed method with the optimal smoothing factor and the existing PTP, where 1,000 samples of synchroniza-
tion errors are collected in each round of experiment.

4.2 Result

By using the method developed in Sect. 2, the reading latency of the CCNT register, system time acquisition and setting latencies, and GPI detection latency are measured as 0.05 $\mu$s, 0.29 $\mu$s, 2.9 $\mu$s and 14.5 $\mu$s, respectively.

The experimental results for various $\alpha$ values are shown in Table 1, and the best performance is achieved when $\alpha = 0.35$, from which the optimal smoothing factor can be estimated as $\alpha = 0.35$. Additionally, the results in Table 1 are analyzed as follows. As $\alpha$ decreases below 0.35, the response speed of the EMA filter becomes so slower that the output of the EMA filter cannot accurately estimate the low-frequency component of time difference samples, which causes the mean value of the synchronization errors to increase. On the other hand, as $\alpha$ increases above 0.35, the response speed of the EMA filter becomes so faster that the output of the EMA filter is affected by the high-frequency component of the time difference samples, which causes some oscillations in the filter output and leads to the large size of mean and RMS values.

The experimental results for the proposed method and PTP are shown in Figs. 7, 8, and Table 2. Figures 7 and 8 show the distributions of 1,000 synchronization error samples from the first round of experiments for the PTP and proposed synchronization method, respectively. The synchronization error statistics of all the ten rounds of experiments for the PTP and proposed method are summarized in Table 2, where GMEAN denotes the grand mean of synchronization errors, i.e., the mean of all the means of the ten experimental results and MRMS denotes the mean of the RMS values of the ten experimental results.

As observed in the distributions of Figs. 7 and 8, the distribution centers are close to zero and the distribution widths are narrow, which demonstrates that the two methods operate normally for time synchronization. Additionally, it is observed that the distribution width of Fig. 8 is narrower than that of Fig. 7, which demonstrates that the proposed method shows better performance in terms of the RMS value of synchronization errors. This better RMS value of the proposed method is also verified from Table 2, but in terms of GMEAN, the PTP shows better performance than the proposed method. However, the synchronization error of the proposed method amounts to only 0.33 $\mu$s without requiring a dedicated PHC and additional synchronization process between the PHC and system time of Linux. This remarkable result ensures that the proposed method is sufficiently useful for practical application and can be a competitive alternative of the PTP when the PHC is not available.

5. Conclusion

We propose a time synchronization method for distributed embedded Linux systems using the CCNT register, GPIO signals, and TCP communication. Synchronization performance is significantly improved by compensating for system time acquisition and setting latencies and GPI detection latency measured by the fine CCNT register; and implementing a recursive synchronization based on the EMA filter. Compared to the existing PTP, the proposed method shows the advantages of requiring no PHC and synchronizing directly between the Linux system times. The experimental results show that the proposed method achieves a remarkable synchronization error of 0.33 $\mu$s in the mean, which is sufficient performance for practical applications. Consequently, the proposed method can be used for a competitive alternative of the existing PTP when the PHC is not available.

References

[1] R. Cochran, C. Marinescu, and C. Riesch, “Synchronizing the Linux System Time to a PTP Hardware Clock,” Proceedings of IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, pp.87–92, 2011.
[2] M. Leng and Y. C. Wu, “Distributed Clock Synchronization for Wireless Sensor Networks Using Belief Propagation,” IEEE Trans. Signal Process., vol.59, no.11, pp.5404–5414, Nov. 2011.
[3] M. Akhlaq and T. R. Sheltami, “RTSP: An Accurate and Energy-Efficient Protocol for Clock Synchronization in WSNs,” IEEE Trans. Instrum. Meas., vol.62, no.3, pp.578–589, March 2013.
[4] T. Kovachzay, T. Tusori, and D. Vincze, “Prototype Implementation and Performance of Time-based Distributed Scheduling on Linux for
Real-Time Cyber-Physical Systems.” 2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication, Nov. 2018.

[5] R.L. Scheiterer, C. Na, D. Obradovic, and G. Steindl, “Synchronization Performance of the Precision Time Protocol in Industrial Automation Networks,” Journal of IEEE Trans. Instrum. Meas., vol.58, no.6, pp.1849–1857, 2009.

[6] S.-M. Park, H. Kim, H.-W. Kim, C.N. Cho, and J.-Y. Choi, “Synchronization Improvement of Distributed Clocks in EtherCAT Networks,” Journal of IEEE Communications Letter, vol.21, no.6, pp.1277–1280, Feb. 2017.

[7] M. Laner, S. Caban, P. Svoboda, and M. Rupp, “Time synchronization performance of desktop computers,” 2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, Sept. 2011.