2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)
July 7-11, 2015. Busan, Korea

Performance Analysis of Linux-Based EtherCAT DC Synchronization

Hyun-Chul Yi[1] and Joon Young Choi[2]

Abstract— **We analyze the performance of Linux-based EtherCAT synchronization technique, namely the distributed clock (DC). The IgH EtherCAT master module is an open source program for the desktop Linux PC. In order to use the EtherCAT master in Linux system environment, we change and compile a Real-time patched kernel. As a result, we become able to use Linux-based EtherCAT master with the EtherCAT slave hardwares in the real experiments. Experiment results show that the system time differences between the reference clock and the slave clock are precisely measured in the real experiments; the DC mechanism appropriately compensates the time offsets; and the stability is achieved in the Linux-based real EtherCAT network.**

I. INTRODUCTION

The Ethernet-based network protocol is widely used in industrial environment. Its advantages include high speed, high performance, operation over long distances, and better interoperability. In particular, the Ethernet-based network has to simplify wiring and make maintenance convenient to take advantage of these merits for wide deployment in the industrial control system. EtherCAT is one of these network standards [1]. EtherCAT enables a complete networked control system to be built without even installing any custom interface card on the controller. Moreover, due to the availability of open-source EtherCAT master components like IgH EtherLab [2], this kind of systems can nowadays be built entirely from open-source components [3].

On the basis, the Linux systems are developing fast and have been used in controls and applications. As the system is of high performance and the cost is low, it has the tendency to play greater role of controls and systems. With the use of EtherCAT and the Linux system at low level of factory automation, a more flexible and convenient way for distributed control system to reality [4]. This paper begins with an introduction to EtherCAT master using a Linux system and EtherCAT slaves. The EtherCAT master protocol is sufficiently detailed to understand the remainder of the paper. Then, we explain the DC mechanism of the EtherCAT protocol. Finally, this paper discusses the experimental results using the Linux-based EtherCAT Master and the EtherCAT slaves. The Linux-based EtherCAT master system is implemented using the IgH EtherLab and Ubuntu. The EtherCAT slave systems are implemented using a TI AM3359 evaluation board with a sys/bios software stack.

[1]Hyun-Chul Yi is with Department of Electrical and Computer Engineering, Pusan National University, Busan, 609-735, Korea, hcy@pusan.ac.kr

[2]Joon Young Choi is Department of Electrical Engineering, Pusan National University, Busan, 609-735, Korea, jyc@pusan.ac.kr

TABLE I: Hardware Specification of the Linux System

List	Specification
CPU	Intel i5-3570(3.4Ghz)
RAM	8.00GB
HDD	Intel SSD 120GB
OS	Ubuntu 12.04
Kernel	3.2.0-rt10

II. LINUX-BASED ETEHRCAT

This article shows how the EtherCAT control system is designed in a Linux PC system. As Linux OS, we use Ubuntu 12.04 LTS with kernel 3.2.0 version. The system's hardware specification is depicted in Table I.

A. EtherCAT Master (EtherLab)

Basically EtherLab works as a Real Time kernel module [5], [6] attached to the open source operating system Linux communicating with peripheral devices by a special Ethernet technology, known as EtherCAT. Using the EtherLab, the Linux PC can act as the EtherCAT master. EtherLab has the following features.

- Open Source
- Hard Real Time
- Multi-Client, -User, -Server, -Tasking
- Industrial Grade I/O Stations
- Flexible

The practical feasibility of such a solution has been demonstrated by the results presented in [3], where the performance of the open-source EtherLab, executed on two different real-time extensions of Linux, has been evaluated both in an insulated environment than in the presence of interfering best effort loads. The EtherLab EtherCAT master has been designed for control applications developed as kernel modules: this choice optimizes performance, since avoids switching between kernel space and user space and allows a direct communication with the network driver [2], [7].

Fig. 1 shown that the architecture of the EtherLab EtherCAT master. In the Fig. 1, the control application uses the EtherCAT library, that provides a user space interface to communicate with the master module. This communication is realized by means of a character device, namely

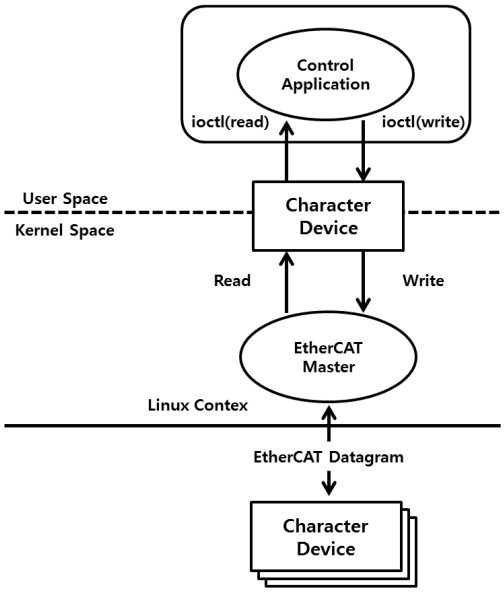

Fig. 1: Architecture of the EtherLab EtherCAT Master

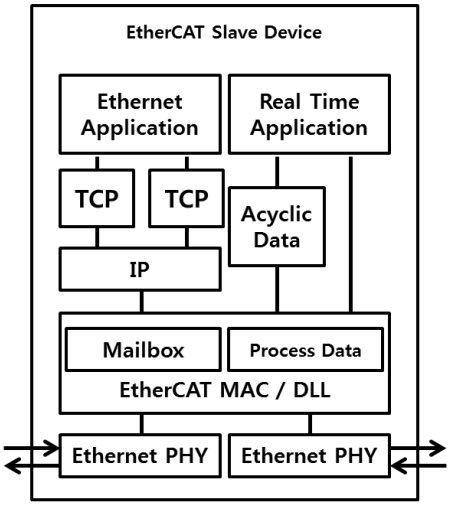

Fig. 2: Component of an EtherCAT node

$/dev/EtherCATx$, where x identifies the EtherCAT Master. The EtherCAT library maps the kernel functions (namely read, write or configuration commands) of the master API to user space through an *ioctl* interface, made available by this character device [2].

Even if the controller application is run in a hard real-time context, each time a master function is executed, the arguments and a function identifier are passed to the master kernel module by means of the *ioctl* system call, that in turns generates a trap and preempts the hard real-time context to serve the call. Moreover, since the character device is generated by the master module, the called function is executed by this component, that runs as a normal kernel thread. In this situation, non-real-time interfering tasks can be scheduled, causing interferences to the control application, for instance, in the case they have pending I/O operations. For these reasons, such a solution cannot be considered hard real-time and thus it seems that cannot fulfill the requirements in term of determinism needed by EtherCAT [2], [7].

B. EtherCAT Slave

The AM3359 microprocessors, based on the ARM Cortex-A8 processor, are enhanced with peripherals and industrial interface options such as EtherCAT and PROFIBUS. The Programmable Real-Time Unit Subsystem and Industrial Communication Subsystem (PRU-ICSS) is separate from the ARM core, allowing independent operation and clocking for efficiency and flexibility. The PRU-ICSS enables additional peripheral interfaces and EtherCAT real-time protocol [8], [9], [10].

Each EtherCAT node has three components, the physical layer, the data link layer and an application layer as shown in Fig. 2. The physical layer is implemented using 100BASE-TX copper, 100BASE-FX optical fiber or E-bus based on LVDS signaling. The MAC is implemented either in a specialized ASIC or an FPGA as per the EtherCAT standard specifications. Beyond the MAC is the industrial application that takes care of application-specific behavior and a standard TCP/IP and UDP/IP stack to support Ethernet-based device profiles. Depending on the complexity of the device, the EtherCAT node can be implemented in hardware or it could be a combination of hardware and software running on an embedded CPU [9].

III. ETHERCAT SYNCHRONIZATION

A. Distributed Clock

Distributed Clock (DC) is possible to synchronize the slave clocks on the bus to the $Reference\ Clock$ (which is the local clock of the first slave with DC support) and to synchronize the reference clock to the $Master\ Clock$ (which is the local clock of the master). This master clock is considered as a system clock. All other clocks on the bus (after the reference clock) are considered as $Slave\ Clocks$. This structure is presented in Fig. 3 [2].

One EtherCAT device will be used as a Reference Clock. Typically, the Reference Clock is the first EtherCAT slave controller with DC capability between master and all the slaves to be synchronized (DC slaves). The Reference Clock might be adjusted to a "global" reference clock. Each DC slave has a local clock, initially running independent of the Reference Clock. The difference between local clock and Reference Clock (offset) can be compensated, as well as clock drifts. The propagation delay, $t_{propagation}$, between

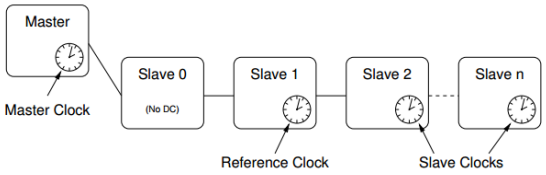

Fig. 3: Diagram of EtherCAT DC

Reference Clock and slave clock has to be taken into account when the System Time is distributed to the slaves. The offset, t_{off}, between local clock and Reference Clock has two reasons: the propagation delay from the slave controller holding the Reference Clock to the device with the slave clock, and initial differences of the local times resulting from different times at which the slave controllers have been powered up. This offset is compensated locally in each slave. Each slave store the time of its local clock, t_{recv}, when the first bit of the EtherCAT frame was received, separately for each port ($Receive\ Time\ Port$ 0-3 registers) [2], [9].

$$t_{sys} = t_{local} + t_{off} \quad (1)$$

$$\Delta t = t_{local} + t_{off} - t_{propagation} - t_{recv} \quad (2)$$

From (2), the small time offset error resulting from the different times of reading and writing the registers will be compensated by the drift compensation.The drift compensation is possible due to a special mechanism in each DC-capable slave. A write operation to the $System\ time$ register will cause the internal time control loop to compare the written time to the current system time. The calculated time error will be used as an input to the time controller, that will tune the local clock speed to be a little faster or slower, according to the sign of the error. If Δt is positive, the local time is running faster than the System time, and has to be slowed down. If Δt is negative, the local time is running slower than the System time, and has to be sped up. The time control loop adjusts the speed of the local clock [9].

B. DC Registers

The Registers used for DC compensation are listed in Table II. DC capable slaves provide the 32-bit "System time difference" register at address 0x092C, where the system time difference of the last drift compensation is stored in nanosecond resolution and in sign-and-magnitude coding. To check for EtherCAT synchronization, the system time difference registers can also be cyclically read via EtherCAT's register access commands e.g., BRD and FPRD commands.

BRD(Broadcast Read) command is that all slaves put logical OR of data of the memory area and data of the EtherCAT datagram into the EtherCAT datagram. All slaves increment position field. FPRD(Configured Address Read) command is that slave puts read data into the EtherCAT datagram if address matches with one of its configured addresses.

TABLE II: Registers for DC Compensation

Address	Name	Type	Access
0x0900	Receive Time Port 0	DWORD	R
0x0904	Receive Time Port 1	DWORD	R
0x0908	Receive Time Port 2	DWORD	R
0x090C	Receive Time Port 3	DWORD	R
0x0910	System Time	UINT64	RW
0x0918	Receive time Processing Unit	UINT64	RW
0x0920	System Time Offset	UINT64	RW
0x0928	System Time Transmission Delay	DWORD	RW
0x092C	System Time Difference	DWORD	RW

IV. EXPERIMENT

A. The hardware and software platform

The experiment is conducted with one PC master and three slaves. The network topology of EtherCAT is presented in Fig.4. The slave device's delay was calculated automatically by the EtherCAT protocol when the slave was connected. The EtherCAT system was implemented and tested on a general purpose PC equipped with Table I. The network interface card carrying EtherCAT traffic is based on the Intel gigabit network card. The PC has been set up with a Linux kernel version 3.2.0 patched with the RT version 10 components and calibrated to schedule and synchronize hard real-time tasks. The Linux kernel with RT patch offers real-time performance comparable with commercial RTOS as well as IgH EtherCAT master stack installed is optimally designed for Linux kernel module. As for the slave device, TMDSICE3359 provided by Texas Instrument was used as slave devices [3], [8].

B. DC Registers Access

To access the registers of EtherCAT, we use FPRD command. Using the FPRD command, we can select the EtherCAT slave's index number, register address, and data size. Fig. 5 shows that a pseudo code for DC register access. In Fig. 5, ADDRESS is the slave's index number. This value increased from 1 to 3 and resumed at 1. The second parameter value as 0x0900 is a start address of read data and the last parameter value as 48 is the total data size in bytes. 48bytes is the sum of 5 of DWORD registers and 3 of UINT64 registers.

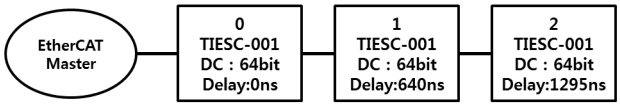

Fig. 4: Network Topology of EtherCAT

```
ADDRESS = 1
Loop {
        COMMAND_RECEIVE();
        COMMAND_PROCESS();

        DATA_PRINT();
        FPRD(ADDRESS++, 0x0900, 48);

        COMMAND_QUEUE();
        COMMAND_SEND();

        if (ADDRESS == 4)
                ADDRESS = 3;
}
```

Fig. 5: Pseudo Code for DC Register Access

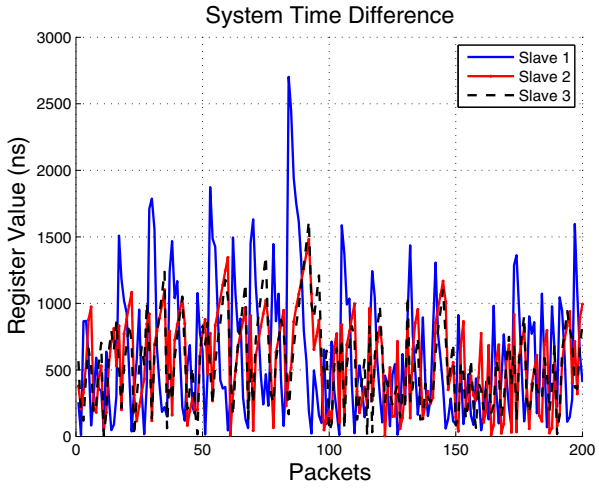

Fig. 6: Measured System Time Difference of EtherCAT Slaves

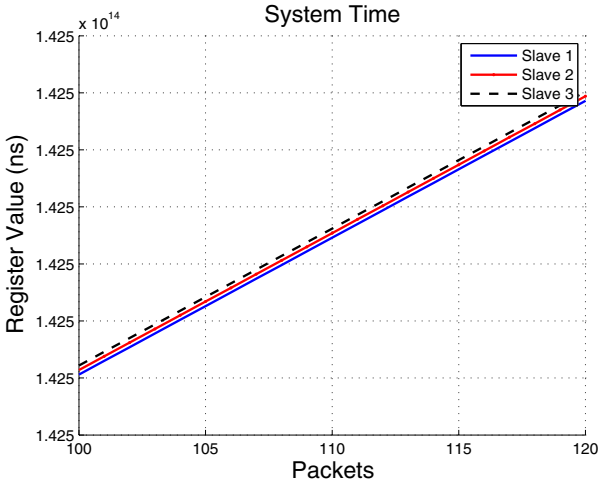

Fig. 7: Measured System Time of EtherCAT Slaves

C. Experimental Results

In order to let the DC mechanism settle, every test was preceded by a warm-up phase (about 5-min long), where about 75 thousand samples were acquired and discarded [1]. Fig. 6 shows that the measured system time difference of EtherCAT slaves. Because of a difference hardware clock between the master and the slave 1, the slave 1's System Time Difference is slightly larger than others. The slave 1, 2 and 3 were used same hardware clock. Therefore, the slave 2 and 3's System Time differece is smaller than another. The slaves All slave's values, however, have similar oscillating patterns. In Fig. 7, the difference between the slave 1 and the slave 2 is average 399.998ns and standard deviation 0.216ns. The difference between the slave 1 and the slave 3 is average 799.997ns and standard deviation 0.236ns. Fig. 8 shows that the system time difference and the cycle time are oscillated in a ranged boundary. The cycle times of EtherCAT master, especially, are oscillate between 5ns relative to 4ms.

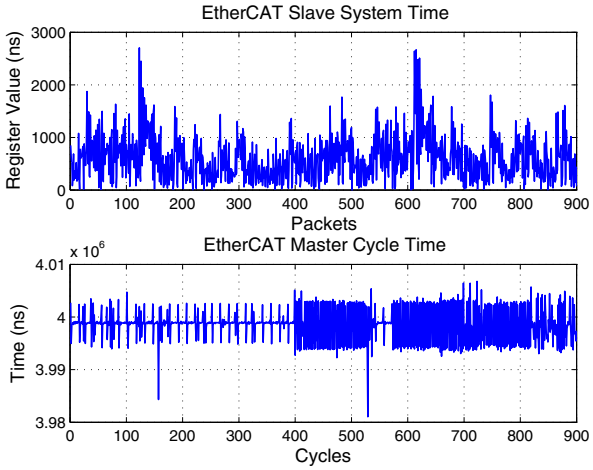

Fig. 8: Measured System Time of EtherCAT Slave and Cycle Time of EtherCAT Master

V. CONCLUSIONS

In this paper, the performance of Linux-based EtherCAT synchronization technique, namely the Distributed Clock, is analyzed with the Linux-based EtherCAT master and the slaves. For this purpose, the real-time patched Linux and real network setups in the experiments. As a consequence, we are obtained that the Linux-based EtherCAT is reliable and stable. Despite experimentation was necessarily carried out on small test-beds, results shows that DC performance on Linux-based is actually very good. All in all, DC seems to be able to support applications that require distributed synchronization (e.g., motion control and cnc) in the Linux-based system.

REFERENCES

[1] I.-S. Song, Y.-H. Jeon, J.-H. Kim, S.-H. Seo, K.-H. Kwon, J.-H. Chun, and J.-W. Jeon, "Implementation and analysis of the embedded master for EtherCAT," *Proc. Inst. Elect. Eng., in Proc. Int. Conf. Control Automation and Systems, ICCAS'10*, pp. 2418-2422, Oct. 2010.

[2] "IgH EtherCAT master Reference Manual," [Online]. Available: http://www.etherlab.org/en/ethercat/

[3] M. Cereia, I. Cibrario-Bertolotti, and S. Scanzio, "Performance of a real-time EtherCAT master under Linux," *IEEE Trans. Ind. Inf.*, vol. 7, no. 4, pp. 679-687, Nov. 2011.

[4] G.-P. Li, "Design of an embedded control and acquisition system for industrial Local Area Networks based on ARM," *Int. Conf. on Computer Science and Education (ICCSE)* pp. 35-39, Aug. 2010.

[5] P. Gerum, "XenomaiImplementing a RTOS emulation framework on GNU/Linux," 2004. [Online]. Available: http://www.xenomai.org/

[6] P. Mantegazza, E. Bianchi, L. Dozio, S. Papacharalambous, S. Hughes, and D. Beal, "RTAI: Real-time application interface," *Linux J.*, no. 72, pp. 142-148, Apr. 2000.

[7] M. Cereia and S. Scanzio, "A user space EtherCAT master architecture for hard real-time control systems," *IEEE Conf. on Emerging Technologies and Factory Automation (ETFA)* pp.1-8, Sept. 2012.

[8] "AM3359 industrial Communications Engine (ICE) Schematic," [Online]. Available: http://www.ti.com/tool/tmdsice3359.

[9] "Introduce of EtherCAT," [Online]. Available: http://www.ethercat.org/pdf/ethercat_e.pdf

[10] G. Cena, I.C. Bertolotti, S. Scanzio, A. Valenzano, C. Zunino, "Evaluation of EtherCAT Distributed Clock Performance," *IEEE Trans. Ind. Inf.*, vol. 8, no. 1, pp. 20-29, Feb. 2012.