International Journal of Fuzzy Logic and Intelligent Systems, vol. 9, no. 3, September 2009 pp. 244-248

Real-time EtherCAT Master Implementation on Xenomai for a Robot System

Yongseon Moon*, †Nak Yong Ko, Kwangseok Lee***, Youngchul Bae **** and Jong Kyu Park*****

***School of Information Communication, Sunchon National University, Korea, moon@ sunchon.ac.kr**
****Dept. Control, Instrumentation, and Robot Engineering, Chosun University, Korea, nyko@chosun.ac.kr**
*****Robotics Institute, Redone Technologies, Ltd., lks@urc.kr**
******Division of Electrical Electronic Communication, Chonnam National University, Yeosu, Korea, ycbae@chonnam.ac.kr**
*******Korea Institute of Science and Technology Information**

Abstract

This paper describes a real-time EtherCAT Master library. The library is developed using Xenomai. Xenomai is a real-time development framework. It cooperates with the Linux kernel, in order to provide a pervasive, interface-agnostic, hard real-time support to user-space applications, seamlessly integrated into the GNU/Linux environment. The proposed master library implements EtherCAT protocol for master side, and supports Application Programming Interfaces(APIs) for programming of real-time application which controls EtherCAT slave.

Key Words: EtherCAT, Xenomai, real-time, master, slave

1. Introduction

EtherCAT implements master/slave architecture over standard Ethernet. The master controls a chain of EtherCAT devices. There have been some EtherCAT masters developed on several different platforms[1]. Some of the masters support real-time operations while some others only do non-real time operations[2].

In some applications, real-time performance is vital to control an intelligent and dynamic system. Especially in robotics, real-time performance is critical because each component of the system should work in corporation and coordination. As an example, time-critical coordination of motions of joints and sensors is indispensible for a humanoid robot which has many degrees of freedom(over 30) to function successfully[3]. In this case, a protocol which supports fast transfer and guarantees strict synchronization is required. EtherCAT is one of the promising protocols for this purpose[4,5,6,7]. The master library developed in this research provides APIs enabling application developers to communicate with and control EtherCAT Slaves. The library uses Xenomai to provide real-time support with the order of micro seconds of latency.

Generally, EtherCAT master plays an intermediary role between application and EtherCAT slave[6]. Most of the masters work in transparency to application layer. Master services run in back-ground and are responsible for managing slave states, processing data and responding to application requests.

The main characteristic of a slave device is its state. In each state, slave device just can execute some specific services. Most of services can be executed in operational state. Therefore, before allowing application layer to communicate with slave, master must setup the proper state for the slave device.

The paper begins with introduction in section I, followed by discussions on Xenomai in section II. Section III explains system structure of a robot in which the master is implemented. Section IV describes how the master is designed. The proposed system is implemented and tested. The result of the test is detailed in section V. Section VI closes the paper with some remarks on the proposed master design.

2. Real-Time Operating System

A robot system is required to be real time system. Especially for a humanoid robot which is highly dynamic system to work in robust and stable manner, real time performance with the servo frequency of kilo hertz is strongly recommended[3]. There are some operating systems or platforms which support real-time operation: Windows CE, INtime, RTLinux, RTAI, Xenomai, BlueCat Linux, MontaVista Linux, and TimeSys Linux [8,9,10].

With a licensed product, we can obtain a good service but they only support in the limited range. Even if the solution is not the best design for our system, we can not modify it. On the other hand, for an open source product, we can modify the source code to make it best suitable for our system. Linux is open source product which is widely in use.

Use of RTAI and Xenomai endows the Linux with real time performance. Both the RTAI and Xenomai use the same ideas

Manuscript received Jul. 17. 2009; revised Sep. 9. 2009.
This research was supported by the "GRRC" project of Gyeonggi Provincial Government, Republic of Korea.
†Corresponding author: Nak Yong Ko, Chosun Univ., nyko@chosun.ac.kr

and support RTDM layer. However, they have some differences in the goals and the way for implementation. RTAI is focused on improving performance to get the lowest latency, while Xenomai considers extensibility, portability and maintainability as well as low latency[11,12]. Table 1 compares RTAI and Xenomai.

Table 1. Comparison of RTAI and Xenomai

RTAI	Xenomai
Architecture support	
x86, x86-64, PowerPC, ARM, MIPS	X86, x86-64, PowerPC, ARM, PowerPC64, Blackfin
Space	
Kernel, user	Kernel, user
APIs	
RTAI, POSIX 1003.1b, RTDM	Xenomai Native, Posix 1003.1b, RTAI, VxWorks, pSos+, VRTX uITRON, RTDM

Xenomai supports a lot of CPU architectures and APIs. This feature enables users to port applications from one to another architecture. Besides, the mailing list of Xenomai project provides good support for the developers with its prompt and active responses. In these respects, Xenomai is used to develop real time EtherCAT master library in this research.

3. System Architecture

The robot system in this work consists of the four layers: Applications, EtherCAT master, Xenomai, and real-time driver. Each layer has the following functions. Fig.1. depicts the architecture.

(1) Application layer: Users use the master library to develop their own applications.

(2) EtherCAT Master: We use APIs which Xenomai supports to develop real-time EtherCAT master Library in user space. It includes two main parts: XML parser and EtherCAT master core. XML parser is needed because the configuration information of slaves is obtained from XML file. EtherCAT master core is responsible for managing slave states and enables data communication between application and slaves. It begins to work when user application calls the function "Start()," and runs until the function "Stop()" is called. User application accesses EtherCAT master services by calling EtherCAT master APIs.

(3) Xenomai-Linux platform: The platform supports real-time system calls which can be used in user space or kernel space. Native and POSIX APIs are used to develop EtherCAT Master Library. Xenomai includes various layers. The Adeos plays the role of virtualizing hardware interrupts; this layer provides basic service of enabling the Linux to run in real-time which is the attribute of Xenomai. The Adoes is directly exposed to the hardware abstraction layer. Most of the requests for Adeos services are issued from HAL layer. The nucleus layer realizes real-time functionality.

(4) RTnet – Real time driver: After installing Xenomai to endow the standard Linux with real-time capability, all standard Linux services still run normally without updating new driver. Non real-time task can still use devices with old (non real-time) drivers. However, if a real-time task use non real-time services it will be changed to secondary mode and run as normal Linux tasks. To obtain hard real-time response, real-time driver must be used. For this purpose, RTnet is the best solution[9,12,13]. RTnet supports several popular NIC adapters including Gigabit Ethernet. In case of EtherCAT master, RTnet provides POSIX socket APIs enabling real-time communication with slaves.

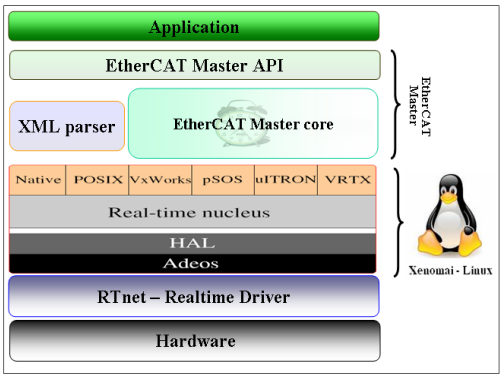

Fig. 1 System architecture

4. EtherCAT Master Design

It is desirable for the real-time EtherCAT master library to be simple, efficient and user-friendly to provide users with convenient tool for slave control. In addition, portability to other architectures is very important for the master. At present, C and C++ are supported in many architectures and they are familiar to most of the developers. So using Native and POSIX APIs is a good choice to satisfy these requirements. The proposed architecture is shown in Fig. 2.

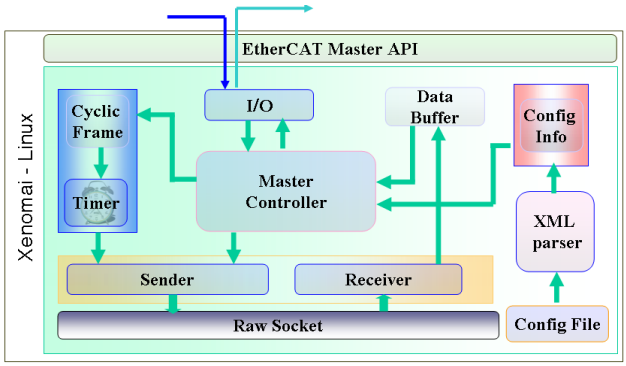

Fig. 2 EtherCAT master library

There are four major components in the master library: Config info, sender/receiver, master controller, and Cyclic.

They have the following functions.

(1) Config info stores the information about slaves.

(2) Sender/receiver is responsible for sending and receiving packets in real-time.

(3) Master controller which is the heart of master library manages all operations.

(4) Cyclic maintains the communication between master and slave.

To obtain the best performance of the tasks, each component should have suitable priority. XML parser gets information of slaves from configure file and then saves the information into Config info object. The response time for this task impose no restriction on the overall performance because it just run only once when user starts the library. So the XML parser has the lowest priority.

To read information from XML file, one can also use an XML parser library available from GNOME such as libxml2. However, in case of EtherCAT master, there are some features which impede application of libxml2. One of the disadvantageous features is the size of the library. Big size of the library causes some problems when porting it to an embedded system[14,15]. To solve the problem, this paper proposes developing a simple XML parser module which fits to EtherCAT master implementation.

EtherCAT frame header does not include any headers of transport layer or network layer. To send and receive packets, the proposed master uses raw socket with real-time network driver supported by RTnet. The priority of the sender can be adjusted depending on the importance of the data. If it is required to send a data urgently, sender will be assigned higher priority(TASK_PRIO 99). Cyclic frame is sent by the sender with high priority(TASK_PRIO 97). Sender always runs to get response packets from the slave with the same priority.

While a slave is running, most of constructions which control slave are included in some fields of cyclic frame. It means that cyclic frame format will not change. Therefore, to improve the performance, cyclic frame is cached in memory and sent to slaves after a period of time. Some fields of cyclic frame will be updated if needed.

Master controller is the most important component. It controls other components, processes data and issues suitable action to control slaves. As we discussed above, EtherCAT slaves have different characteristics from other types of devices. One of the characteristics is the state of slaves. There are several states of slave: Init state, Pre-Operational, Safe-Operational, Operational, and Bootstrap(optional).

Each state has its required services. Master should provide all services required after slaves confirm the state change. The transition from Init state to Operational state follows the sequence: Init, Pre-Operational, Safe-Operational, and Operational. In order to change the slave state to Operational, a developer can change the state in sequence, or change the state at the same time.

Though the time needed to create one EtherCAT frame header is small, it increases with the number of packets. To optimize the payload of each packet sent, it is required to change all slave states at the same time. Basically, we have three state changes for a slave: Init to Pre-Operational (I2P), Pre-Optional to Safe-Operational(P2S), and Safe-Operational to Operational (S2O).

The proposed solution to I2P stage will be explained. Similar approach is used for the solution to other stages. In this stage, master must provide all the services which the slave requires. Information about these services stored in init command with transition field is “IP.” We create a frame containing all services required and send the frame to slaves, and in turn, create other frames until all the slaves change to Operational state.

On master side, we can not change the time between sending a packet and receiving the response from slave for the packet, because it depends on the number of slaves and the process time of each slave. To minimize the time between two packet sending, we assign high priority(TASK_PRIO 98) to the master controller task. Though TASK_PRIO 98 is not the highest priority, controller task always runs until EtherCAT master stops. As we discuss above, a sender task has the highest priority (TASK_PRIO 99). However, the task is created by master controller task, and stops itself after finishing data sending. Therefore, only one master controller task runs with the highest priority, so that the master provides real-time services to users.

The APIs should be easy to use to make the master library as simple as possible to the user. The API has the five functions.

(1) LoadConfig() function: This function should be called before any other function call. XML parser will parse the information stored in XML file and save it into the Config Info object.

(2) Start() function: The function Start() lets the master controller use information stored in Config Info object to change all slaves’ state to Operational. In addition, Cyclic frame is created and sent to slaves after a period of time. Then, a user can use WriteData() or ReadData() function.

(3) WriteData() function: When this function is called, master updates cyclic frame and then send data to slaves immediately.

(4) ReadData() function: Master does not know when a user calls ReadData() function. It takes some micro seconds for the packets to be sent from master to slave. Master cannot store all packets. After receiving a packet, Receiver will store it in a buffer. If next packet arrives, the next packet will be overwritten to the buffer. Therefore, the ReadData() function returns the latest packet received from slaves.

(5) Stop() function: The function Stop() stops all master services and frees memory.

All the above functions are built as shared library so that a user can use these functions with no build time. But it can cause some complexity when a user compiles an application because Xenomai environment can have some new options in the shared library.

5. Experiments

Using the proposed approach, the authors have developed a real-time EtherCAT master library that provides several APIs to users. As an application which uses these APIs, a slave system which turns on and off LEDs is built. The master and slave system include a Xenomai – Linux PC and two EtherCAT slaves. The Xenomai – Linux PC has CPU Intel core 2 dual 2.8 GHz, 2 GB RAM, and 250 GB HDD.

In this application, we set Xenomai task to use a 200 microsecond tick. To capture real-time packet transfer, Wireshark with RTnet plug-in is used. It is found that in most of the cases it takes 200 microseconds to transfer a packet to and from the master and slave. A package has 60 bytes of data. The packet transfer time is shown in the Fig. 3. Unlike other data transfer protocols, the extra time required to transfer data to the additional slaves is much less than 200 microseconds. The extra time is 1~2 microseconds per one slave system. However, in many other protocols, the data transfer time is multiplication of transfer time per slave and the number of slaves.

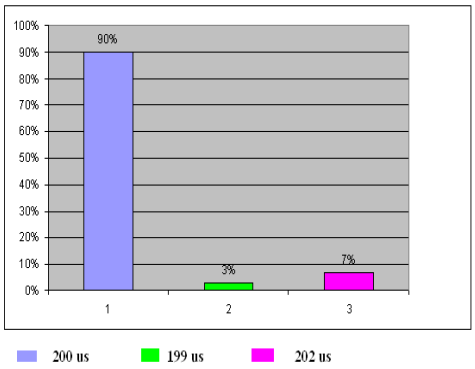

Fig. 3 Packet transfer time

6. Conclusion and Future Work

The proposed EtherCAT master library runs on Xenomai-Linux PC. With one master and two slaves, the average transfer time is 200 microseconds per packet of 60 bytes. It is expected that on an embedded system with limited memory and processing power, performance of Xenomai based real time EtherCAT may deteriorate. It is suggested for further research that EtherCAT master be implemented on embedded Xenomai system and the performance of the EtherCAT master be analyzed for improvement[14,15].

References

[1] D. Jansen, H. Buttner, "Real-time Ethernet, the EtherCAT Solution," *Computing and Control Engineering Journal*, vol. 15, no. 1, pp. 16-21, Feb.-March 2004.

[2] Max Felser, "Real-Time Ethernet - Industry Prospective," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1118-1129, June 2005.

[3] M. Konyev, F. Palis, Y. Zavgorodniy, A. Melnikov, A. Rudskiy, A. Telesh, "Walking Robot "ANTON": Design, Simulation, Experiments," *in Proc. Eleventh International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, 08–10 September 2008, Coimbra, Portugal.

[4] Qian-Liang Xiang, Zhi-Yuan Xin, Ji-Ru Lin, Guo-Jie Li, Hao Liang, Juan Li, "Application of the Real-time EtherCAT Technology in Power Systems," *Relay*. vol. 36, no. 11, pp. 42-45, 1 June 2008, Relay Press.

[5] S.G. Robertz, K. Nilsson, R. Henriksson, A. Blomdell, "Industrial Robot Motion Control with Real-time Java and EtherCAT," *in Proceedings of 2007 IEEE Conference on Emerging Technologies and Factory Automation*, pp. 1453-1456, Sept. 2007.

[6] B. Finkemeyer, T. Kröger, D. Kubus, M. Olschewski, and F. Wahl, "MiRPA: Middleware for Robotic and Process Control Applications," *Proc. 2007 IEEE International Conference on Intelligent Robots and Systems, San Diego*, USA, pp. 76-90, October 2007.

[7] Beckhoff website, http://www.beckhoff.com/

[8] A. Macchelli, C. Melchiorri, R. Carloni, M. Guidetti, "Space Robotics: an Experimental Set-up Based on RTAI-Linux," *Proc. 4th Real Time Linux Workshop*, Boston, USA, 6-7 December 2002.

[9] J. Kiszka, B. Wagner, "RTnet - a Flexible Hard Real-time Networking Framework," *in Proc. 2005 IEEE Conference on Emerging Technologies and Factory Automation*, pp. 449-456, Sept. 2005.

[10] RTAI website, http://www.rtai.org

[11] Xenomai website, http://www.xenomai.org

[12] Xenomai and RTnet website, http://www.xenomai.org /index.php/RTnet:Main

[13] RTnet website, http://www.rtnet.org/

[14] S. Potra, G. Sebestyen, "EtherCAT Protocol Implementation Issues on an Embedded Linux Platform," *in Proceedings of 2006 IEEE International Conference on Automation*, Quality and Testing, Robotics, pp. 420-425, May 2006.

[15] Jens Onno Krah, Christoph Klarenbach, "FPGA Based Field Oriented Current Controller for High Performance Servo Drives," *in Proc. Power Conversion Intelligent Motion, Power Quality*, Nürnberg, 27-29 May 2008.

Yongseon Moon received the B.S. degree, M.S. degree, and Ph.D. degree from the Division of Electronics Engineering, Chosun University, Korea. He is Professor of the school of information communication, Sunchon National University, Korea, from 1992. His research interests include industry communication, robot internal network technology by EtherCAT, middleware and nerves network.

Phone : +82-61-753-1133
Fax : +82-62-373-2022
E-mail : moon@sunchon.ac.kr

Nak Yong Ko received the B.S. degree, M.S. degree, and Ph.D. degree from the Department of Control and Instrumentation Engineering, Seoul National University, Korea, in the field of robotics. He is Professor of the department of Control, Instrumen-tation, and Robot Engineering, Chosun University, Korea, from 1992. During 1996~1997 and 2004~2005, he worked as a visiting research scientist at the Robotics Institute of Carnegie Mellon University. His research interests include autonomous motion of mobile robots(collision avoidance, localization, map building, navigation, and planning), manipulator force/torque control, and incorporation of mobile robot technology into GIS.

Phone : +82-62-230-7108
Fax : +82-62-233-6896
E-mail : nyko@chosun.ac.kr

Kwangseok Lee is a Research Engineer in Robotics Institute at REDONE Tech. He earned B.S. degree, M.S. degree from division of Electrincs Engineering, Sunchon National University, Korea in 2006, 2008. since 2005, he worked in Robotics Institute at REDONE Tech. his research interests are ubiquitous sensor network for robot location, embedded system for intelligent robot, Robot network by EtherCAT, middle ware and nerves network.

Phone : +82-062-375-2003
E-mal : lks@urc.kr

Young-Chul Bae received his B.S degree, M.S and Ph. D. degrees in Electrical Engineering from Kwangwoon University in 1984, 1986 and 1997, respectively. From 1986 to 1991, he joined at KEPCO, where he worked as Technical Staff. From 1991 to1997, he joined Korea Institute of Science and Technology Information (KISTI), where he worked as Senior Research. In 1997, he joined the Division of Electron Communication and Electrical Engineering, Yosu National University, Korea. In 2006, he joined the School of Electrical • Electronic Communication and Computer Engineering of Chonnam National University, where he is presently a professor. His research interest is in the area of Chaos Nonlinear Dynamics that includes Chaos Synchronization, Chaos Secure Communication, Chaos Crypto Communication, Chaos Control and Chaos Robot, Robot control etc.

Jong Kyu Park received his B.S and M.S degrees in Electronics Engineering from Chungang University, Seoul Korea, in 1984 and 1990 respectively. From 1986 to 1988, he joined at Changyongsik Patent & Law office, where he worked as Patent Analysis Staff. From 1990 to 1991, he joined at LGE, where he worked as Technical Staff. In 1991, he joined Korea Institute of Science and Technology Information (KISTI), where he is presently a Senior Researcher. His research interest is in the area of Information Analysis that searches a Future Promising Technology.