Implementation and performance analysis of an etherCAT master on the latest real-time embedded linux

Article *in* International Journal of Applied Engineering Research · December 2015

CITATIONS
3

READS
915

4 authors, including:

Raimarius Delgado
Seoul National University of Science and Tec…
9 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Byoungwook Choi
Seoul National University of Science and Tec…
69 PUBLICATIONS **367** CITATIONS
SEE PROFILE

International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 24 (2015) pp 44603-44609

Implementation and Performance Analysis of an EtherCAT Master on the Latest Real-time Embedded Linux

Raimarius Delgado
Graduate Student, Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, South Korea raim223@seoultech.ac.kr

Chang Hwi Hong
Student, Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, South Korea hokofoho@naver.com

Wook Cheol Shin
Student, Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, South Korea shinwc159@hanmail.net

Byoung Wook Choi*
Professor, Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, South Korea bwchoi@seoultech.ac.kr

Abstract
This paper presents a performance analysis of an open source EtherCAT Master on a real-time embedded Linux extension, Xenomai. The system is implemented on an embedded target with the latest Linux kernel. Xenomai is a dual kernel approach that provides real-time characteristics to user-space applications integrated with the standard Linux kernel. IgH EtherCAT Master is an open source solution that supports real-time Ethernet protocol on top of real-time Linux extensions. However, as opposed to PC-based Linux, implementation of toolchains and patches is difficult on an embedded platform due to lack of systematic documents and technical support regarding kernel revisions. Therefore, this paper provides comprehensive procedures and detailed information to deal with latest embedded Linux kernels. We also evaluated the real-time performance in terms of cyclic task, jitter, and transaction time. Transaction time is defined as the time interval between the master sending a command to a slave and receiving the corresponding feedback. Finally, the system is tested by connecting the EtherCAT Master to a slave with CANopen protocol and the results are shown using a control client that exhibits on-the-fly processing. The results signify high potential of utilizing open source EtherCAT-based embedded control devices for commercial and industrial control systems.

Keywords: High Curvature Path, Physical Limits, Convolution, Bézier Curve, Path Planning, Obstacle Avoidance.

Introduction

Nowadays, ratification of Ethernet-based fieldbus system as the standard physical communication layer in the fields of automation and control technology is radically increasing worldwide. Ethernet is already an established command-level technology for both factory networking and inter-control communication [1].

After adaptation of the fieldbus standard, Ethernet-based network simplifies, thoroughly reduces wires, and makes maintenance of the system as convenient as possible. Moreover, the move towards Ethernet as the basic communication platform is also based on the excellent price over performance relationship of the technology [2]. However, Ethernet is not optimized to send subsequent short messages and requires microprocessors at each node that entirely slows down the whole system. In comparison with other fieldbuses, it could not achieve typical automation requirements regarding performance and being deterministic. Some applications require real-time performance that is vital in controlling intelligent and dynamic systems.

Real-time Ethernet protocols have been developed to ensure determinism over standard Ethernet such as, EtherNet/IP, Powerlink, PROFINET, and EtherCAT [2]. EtherCAT (Ethernet for Control Automation) is a real-time Ethernet protocol that is gaining popularity in factory automation and process automation. It offers various appealing features such as higher performance of optimal usage of the Ethernet bandwidth for data transfers, more flexible topology and lower costs than other Ethernet fieldbus technology. [3-5].

At present, a common EtherCAT system uses structured environment that provides high quality, top-performance, and technical support on high-end hardware [6]. However, these systems are not flexible in terms of development especially in solving the black box problem. In addition, another important issue is the cost of acquisition and maintenance where a single unit is considered expensive, bulky, and not fully optimized for its purpose.

Thus, this paper presents an implementation of an open-source EtherCAT Master solution provided by IgH EtherLab [7],

* Corresponding Author

under real-time dual-kernel approach of Xenomai [8] and embedded Linux. This solution gives more power and freedom to the user in developing within the applications, data, and physical layers of the whole system. Moreover, ARM-based processors such as BeagleBoard, ODROID, and Freescale i.MX Series, have found its way to the field as it offers optimal performance for an EtherCAT Master while being operated in low power and relatively low-cost compared to high-end computers.

In order to ensure practical and proper design of a motion control system based on EtherCAT, benchmarking the time characteristics of the protocol using real-time mechanisms is required to verify the expected timing accuracy between EtherCAT Master and Slave communication.

Currently, there are different performance analysis researches for EtherCAT systems but they lack either actual automation workload, or inadequate attention to the performance of real-time mechanisms on the master side. Cereia et.al evaluated the performance of a PC-based real-time EtherCAT Master using the fully pre-emptible Linux kernel, RT_PREEMPT and RTAI in terms of the accuracy of periodic control tasks of the master, without automation workload [9]. Sung et.al addressed the performance analysis of an EtherCAT control system in terms of end-to-end delay of synchronized processes using older versions of both Xenomai and IgH EtherLab, 2.6.0 and 1.5.0 respectively [10].

The contributions of this paper are divided into two main parts: First, the suitable and latest working environment is developed and implemented for an ARM-based embedded board, Freescale i.MX6Q SABRELite. Unlike working environments for PC-based Linux, formulation of compatible versions of the Linux kernel, Xenomai, and EtherCAT has many issues for embedded Linux. These issues include lack of updates for hardware dependent parts of the kernel code, inadequate support of the Adeos patch [11] that is required for dual kernel approach, and the introduction of the Device Tree Binary (DTB) [12] for Linux kernel version 3.x which requires newer version of the bootloader. Next, a performance analysis between the ARM-based EtherCAT Master and a slave communicated using CANopen protocol [13] was conducted. The feasibility of the system was evaluated in three different categories. The first one is the periodicity of the real-time control task in different cyclic periods. The second one is the jitter that occurs for each cycle of the control task, and the final one is the transaction time that is defined as the end-to-end delay for a Master to send a command and receive the corresponding feedback while in connection with the CANopen-based slave.

The second section briefly introduced the EtherCAT protocol and the CANopen-over-EtherCAT (CoE) protocol that employs CANopen mechanisms on the EtherCAT Data layer. Next, we describe the porting procedures for step-by-step implementation of the development environment for the embedded EtherCAT Master. The fourth section shows the experimental results and performance evaluation of the EtherCAT system. The final section concludes this paper.

EtherCAT and CoE

EtherCAT is a protocol offering very high real-time performance and determinism developed by Beckhoff Automation [6]. The typical EtherCAT network supports single-master network configurations, where the master communicates with the slaves by sending them suitable telegrams.

The EtherCAT master sends a telegram that passes through each node. Each EtherCAT slave device reads the data addressed to it "on the fly", and inserts its data in the frame as the frame is moving downstream. The frame is delayed only by hardware propagation delay times. The last node in a segment (or branch) detects an open port and sends the message back to the master using Ethernet technology's full duplex feature. The telegram's maximum effective data rate increases to over 90 %, and due to the utilization of the full duplex feature, the theoretical effective data rate is even higher than 100 Mbit/s.

CANopen is a communication protocol and device profile specification for embedded systems used in automation. CANopen implements the layers above and including the network layer. The CANopen standard consists of an addressing scheme, several small communication protocols and an application layer defined by a device profile. The communication protocols have support for network management, device monitoring and communication between nodes, including a simple transport layer for message segmentation/desegmentation. The lower level protocol implementing the data link and physical layers is usually Controller Area Network (CAN). Fig. 1 shows the sequence diagram in developing a control application for a CANopen over EtherCAT (CoE) with IgH EtherCAT Master [7].

First, CoE protocol enables the complete CANopen profile family to be utilized via EtherCAT. The SDO protocol is used directly, so that existing CANopen stacks can be used practically unchanged. Optional extensions are defined that lift the 8-byte limit and enable complete readability of the object list. The process data are organized in process data objects (PDO), which are transferred using the efficient means of EtherCAT - naturally without 8-byte limit. The application can register the PDOs' entries for exchange during cyclic operation. The sum of all registered PDO entries defines the PDOs.

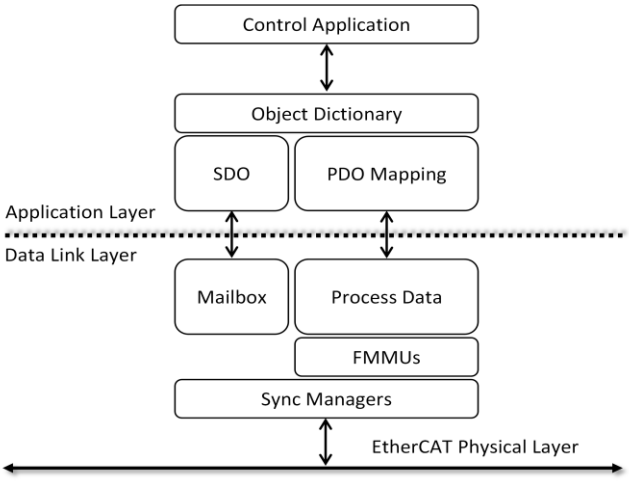


Fig. 1 CoE Device Architecture

Implementation of the Embedded EtherCAT Master

In this paper, the EtherCAT Master uses i.MX6Q embedded board manufactured by Freescale Semiconductors [14]. The summary of the software architecture of the EtherCAT Master is shown in Fig. 2. For an embedded board without its own compiler, host-target development environment is required which in this case, described by Table 1.

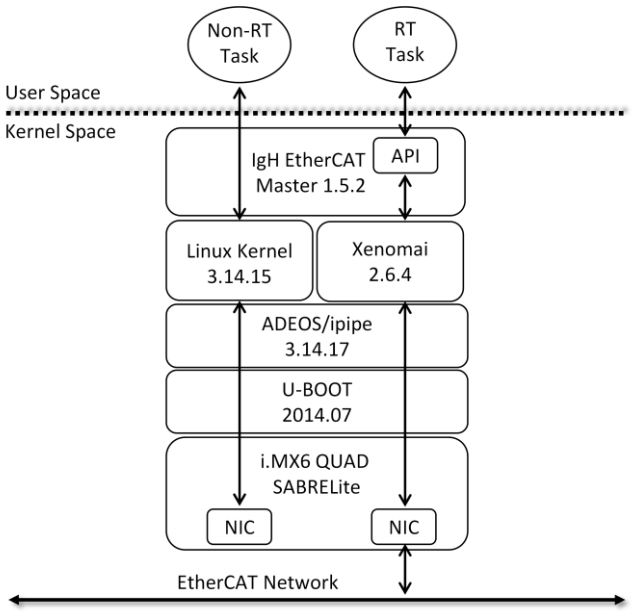


Fig. 2 Software Architecture of the Embedded EtherCAT Master

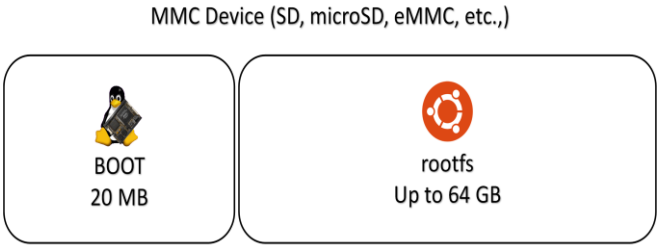
TABLE. 1. Host-Target Development Environment

Item		Description
Host	Processor	Intel Core i5-4460 @ 3.2 GHz
	OS	32-bit Ubuntu 14.04 LTS
	Kernel	Linux 3.16-0-45-generic
Target	Board	Freescale i.MX6Q SABRELite
	Toolchain	arm-linux-gnueabihf-4.8.3
	Bootloader	U-Boot 2014.07
	Kernel	Linux 3.14.15 ARMv7 Multiplatform
	Adeos Patch	ipipe-core-3.14.17-arm-2
	Real-time Skin	Xenomai 2.6.4
	RFS	Minimal Ubuntu 14.04.02
	EtherCAT Master	IgH EtherCAT Master 1.5.2

The latest version of embedded Linux that can be implemented on the i.MX6Q is Linux kernel 3.14.15 offered by ARMv7 Multiplatform [15]. Moreover, the compatible Xenomai version is 2.6.4 with the Adeos patch 3.14.17-arm-2. The real-time system is stacked with IgH EtherCAT Master 1.5.2 which is the latest stable version.

Depending on the manufacturer, most of embedded Linux kernel versions above 3.0 introduce the DTB, depending on the manufacturer. DTB is a data structure that describes the hardware and initializes the devices for the board during kernel boot time. In case of the i.MX6Q, all Linux kernel versions above 3.8 require DTB when the kernel starts to know all the devices within the board and initialize useful drivers.

The factory version of the bootloader, U-boot 2009.08, could not read DTB files therefore; an updated version offered by Boundary Devices [15], U-boot 2014.07 is implemented instead.



Area Name	Format	Size	Memory Block		Partition Name
			Start	End	
BOOT	FAT	20 MB	2048	43007	mmcblk0p1
rootfs	EXT4	Up to 64GB	43008	remaining	mmcblk0p2

Fig. 3 Memory Device Diagram and Partition Table

According to the boot commands and arguments of the latest bootloader, the Multimedia Card (MMC) memory device where the environment should be stored, is separated into two partitions as shown in Fig. 3. The first part is called the BOOT partition, which is a FAT formatted partition where the Xenomai-stacked kernel image and the board defining DTB are stored. It usually requires a size of 20 MB. The remaining memory is formatted in EXT4 to serve as the Root Filesystem (RFS) partition of the Linux kernel where the libraries and APIs of both Xenomai and EtherCAT Master is located.

Latest bootloader is downloaded from the Boundary Devices repository [15]. Default settings for the i.MX6Q SABRELite are configured and the bootloader binary is built.

```
# cd u-boot-imx6
# make mx6qsabrelite_config
# make
```

The latest Linux kernel ported for the i.MX6Q SABRELite board is downloaded by running the shell script, build_kernel within the cloned armv7-multiplatform tree [15] from the v3.14.x branch. The kernel path is stored as an environment variable.

```
# cd armv7-multiplatform
# ./build_kernel.sh
# export KERNEL=$(pwd)/KERNEL
```

To implement the dual-kernel approach using Xenomai and the embedded Linux kernel, the kernel is prepared and patched with Adeos I-pipe.

```
# tar xvjf xenomai-2.6.4.tar.bz2
# export XENO=$(pwd)/xenomai-2.6.4
```

```
#./xenomai-2.6.4/scripts/prepare-kernel.sh
--linux=$KERNEL
--adeos=ipipe-core-3.14.17-arm-2.patch
--arch=arm
# cd $KERNEL
# make distclean
# make imx_v6_v7_defconfig
# make menuconfig
```

When configuring the kernel of menuconfig, some configurations should be defined. Real-time Operating Systems (RTOS) are sensitive with voltage level changes such as switches. These features are disabled to avoid bugs and errors in real-time performance. The following configurations are strictly disabled:

- CONFIG_CC_STACKPROTECTOR_NONE
- CONFIG_CPU_FREQ
- CONFIG_KGDB

Moreover, task pre-emption and real-time scheduler within the Linux kernel are enabled to obtain optimal performance.

- CONFIG_SCHED_MC
- CONFIG_PREEMPT

Compilation of the Xenomai-stacked Linux kernel image, modules, and DTB are described below. The kernel image is loaded by the bootloader in a specific memory address of the RAM. Thus, the kernel image should be built accordingly. The resulting kernel image and DTB are stored in the BOOT partition of the MMC.

```
# make uImage LOADADDR=0x12000000
# make modules
# make dtbs
```

The RFS is downloaded from RobertCNelson's server [16] which is an image of a Minimal Ubuntu Filesystem. After downloading the image, it is uncompressed and the path of the RFS is stored as an environment variable.

```
# tar xvf ubuntu-14.04.2-minimal-armhf-2015-6-9.tar.xz
# cd ubuntu-14.04.2-minimal-armhf-2015-6-9
# tar xvf armhf-rootfs-ubuntu-trusty.tar
# export RFS=$(pwd)/rootfs
```

The kernel modules are installed inside the RFS with the following steps:

```
# cd $KERNEL
# export INSTALL_MOD_PATH=$RFS
# make modules_install
```

After the modules are all installed, the Xenomai library is also installed in the RFS. To optimize Xenomai with the target board, suitable configurations and FPU settings are implemented for ARMv7 architecture as follows:

```
# cd $XENO
# ./configure CFLAGS="-march=armv7-a -mfpu=vfp3"
LDFLAGS="-march=armv7-a -mfpu=vfp3"
--host=arm-linux-gnueabihf
--prefix=$RFS/usr/xenomai
# make
# make install
```

Latest version of the EtherCAT Master offered by IgH EtherLab is cloned from the IgH Mercurial repository found in [7]. Identical to the board optimization settings in building the Xenomai library, same configurations are implemented in building the EtherCAT Master API.

```
# cd igh_ethercat
#./configure
--prefix=$RFS --with-linux-dir=$ KERNEL
--with-xenomai-dir=$RFS/usr/xenomai
--enable-generic=yes --enable-rtdm=yes
CFLAGS="-march=armv7-a -mfpu=vfp3 -g -O2"
CXXFLAGS="-march=armv7-a -mfpu=vfp3
-g -O2"
LDFLAGS="-march=armv7-a-mfpu=vfp3"
--host=arm-linux-gnueabihf
# make
# make install
# make EXTRA_CFLAGS="-march=armv7-a
-mfpu=vfp3" modules
# make modules_install
```

After completion of all the steps, the RFS is moved to the rootfs partition of the MMC. The MMC is attached to the target board and the booting sequence is initialized.

Fig. 4 shows the Latency test results to check real-time characteristics of the Xenomai-stacked Linux kernel. The test mode is done in the user-space and the sampling period of the dummy task is 1000 microseconds. As shown in the figure, the maximum latency for a running time of 5 seconds is approximately 5 microseconds, which is less than the minimum viable latency of 50 microseconds. Fig. 5 checks EtherCAT Master connection with a slave.

```
== Sampling period: 1000 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 1000 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|     -0.298|     -0.026|      5.823|       0|     0|     -0.298|      5.823
RTD|     -0.309|     -0.026|      5.868|       0|     0|     -0.309|      5.868
RTD|     -0.263|     -0.036|      4.219|       0|     0|     -0.309|      5.868
RTD|     -0.298|     -0.041|      4.959|       0|     0|     -0.309|      5.868
---|-----------|-----------|-----------|--------|------|-----------------------
RTS|     -0.309|     -0.031|      5.868|       0|     0|    00:00:05/00:00:05
```

Fig. 4 Xenomai Latency Test

```
root@arm:/# /etc/init.d/ethercat start
Starting EtherCAT master 1.5.2 [ 1555.360661] ec_generic: Binding socket
 done
root@arm:/# /opt/etherlab/bin/ethercat slaves
0  0:0  PREOP  +  L7N
root@arm:/# /etc/init.d/ethercat stop
Shutting down EtherCAT master 1.5.2  done
```

Fig. 5 IgH EtherCAT Operation Check

Experiment Results and Performance Analysis

For this study, we constructed an EtherCAT system with the controller based on the embedded EtherCAT Master discussed in the previous section, a slave operated using CANopen protocol, and a control client that send and acquire data on the fly. The experimental environment is shown in Fig. 6. In addition, Table 2 describes detailed specifications of the system.

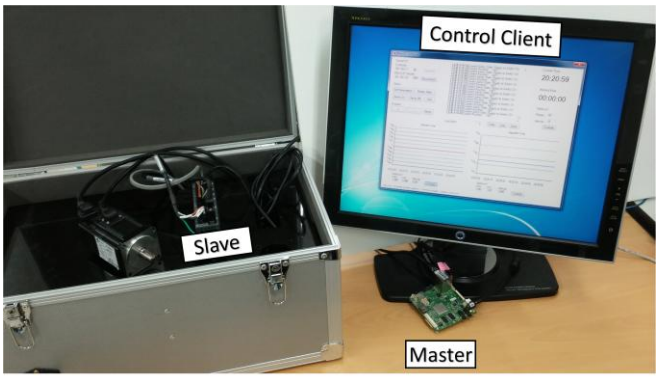


Fig. 6 Experiment Environment

TABLE. 2 Specifications of the EtherCAT System

Item		Description
Control Client	Software	C#-based Application
	Connection	User Datagram Protocol
	Features	State Switch, Position Control
Slave	Board	LS Mecapion L7NA004B
	PDO	Tx 12 Bytes, Rx 12 Bytes

Fig. 7 shows our developed control client application that is connected to the EtherCAT topology that controls servo state of the slave, send position commands, and monitors feedback from the slaves and timing characteristics. The application is developed using C# and uses UDP communication to preserve real-time characteristics.

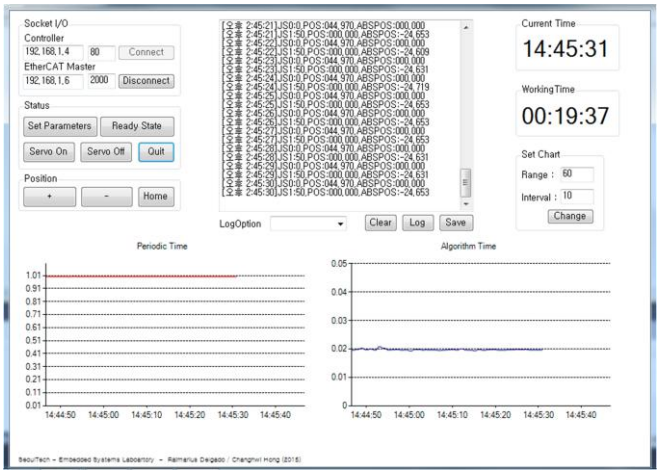


Fig. 7 EtherCAT Control Client

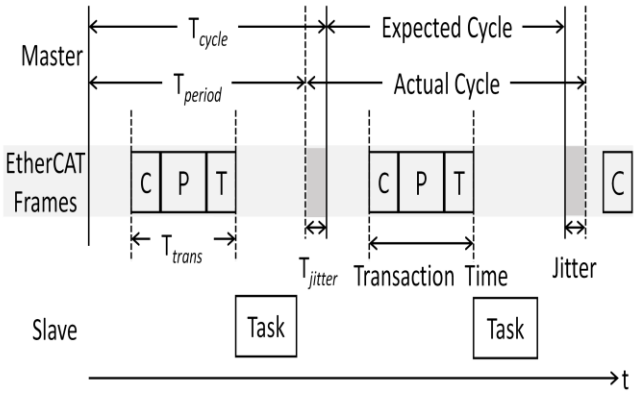


Fig. 8 Timing Diagram of the Real-time Control Task

```
procedure Control_Task
set period and make periodic task;
    while(1)
        read start time;
        collect process image from slave;
        process and compute next command;
        transmit process image to slave;
        read end time;
    end
wait period;
end
```

Fig. 9 Pseudocode of EtherCAT *Control_Task*

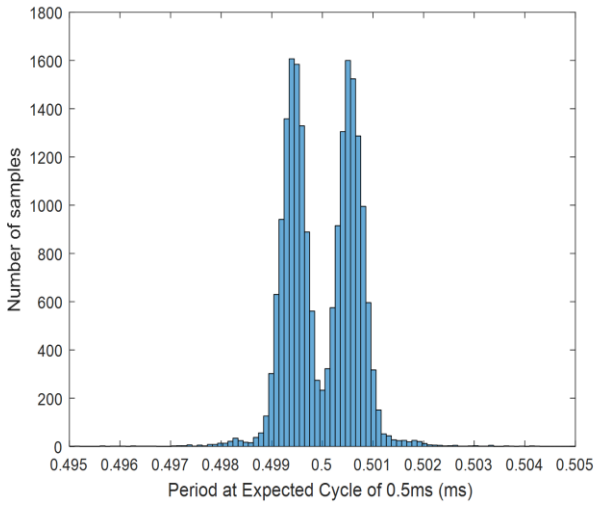
The real-time cyclic control task operating within the EtherCAT Master is expressed in a timing diagram shown in Fig. 8. In this figure, current state and data in process image from the slave is represented by C, which stands for collect. Processing or P denotes the current process where the next control command is generated. Transmit or T is the period where the generated command is transferred to the process image that is sent back to the slave. Fig. 9 shows the pseudocode for the real-time control task operated in the EtherCAT Master that expresses the timing diagram in a different manner. The transaction time of an EtherCAT control task is denoted as T_{trans} which is defined as:

$$T_{trans} = k_n P + k_s (C+T) \quad (1)$$

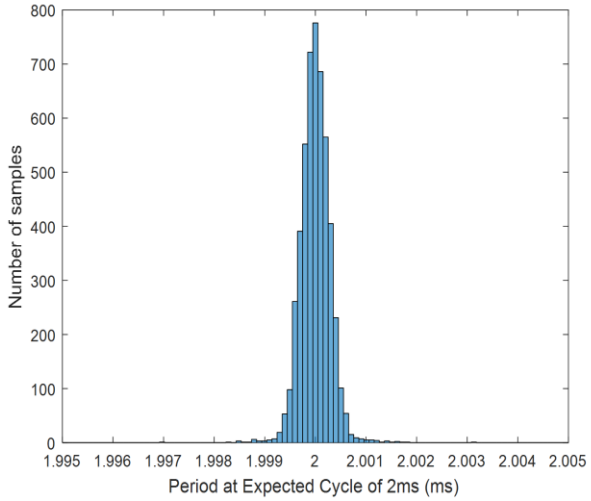
where, k_n is the delay that occurs depending on the number of slave. k_s denotes the delay that depends on the size of the process image that affects the Control and Transmit periods. T_{period} represents actual time that it takes for one cycle of the control task. Its relationship with the jitter is defined by the following equation:

$$T_{jitter} = \left| T_{cycle} - T_{period} \right| \quad (2)$$

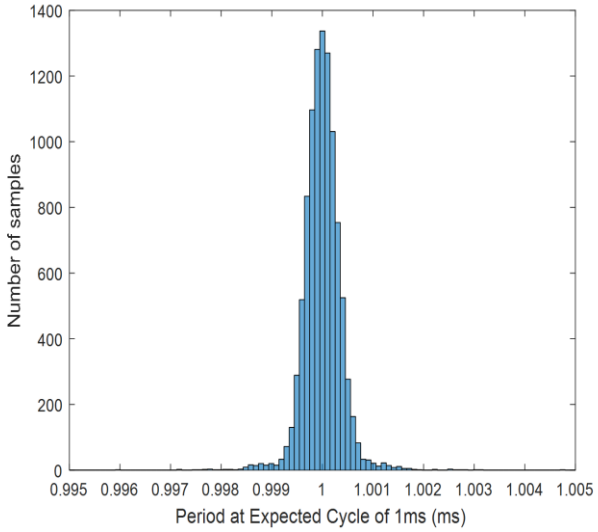
where, T_{cycle} is the expected cycle of the real-time control task.



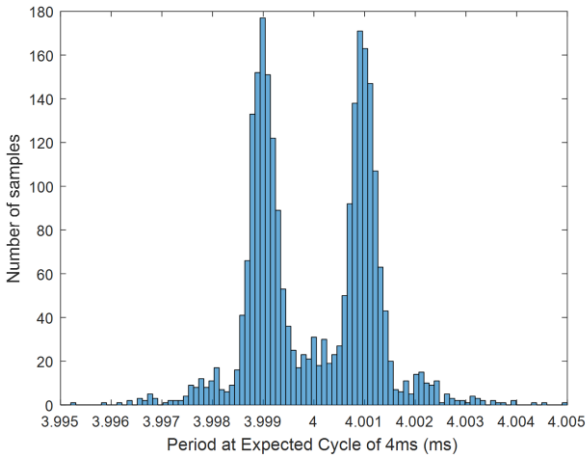
(a) 0.5 *ms*



(b) 2 *ms*



(c) 1 *ms*



(d) 4 *ms*

Fig. 10 Distribution Plot of Actual Period Results in Different Expected Cycle Time

The experiment is performed in varying expected cycle as shown by the distribution plot in Fig. 10. For each task with expected cycle time of 4 ms, 2 ms, 1 ms, and 0.5 ms, test metrics such as the actual period, jitter, and overall transaction time are tabulated in terms of average, maximum, minimum, and standard deviation as shown in Table 3.
The control task was run continuously for 10 seconds for one slave with 24 bytes of process image. The data are acquired and calculated using the control client.

TABLE. 3 Real-tiime Performance Measurement Results (ms)

T_{cycle}	Index	Avg	Max	Min	St. D
4 ms	T_{period}	4.0000	4.0049	3.9953	0.0012
	T_{trans}	0.0796	0.2993	0.0696	0.0070
	T_{jitter}	0.0011	0.0049	0.0000	0.0005
2 ms	T_{period}	2.0000	2.0031	1.9969	0.0003
	T_{trans}	0.0732	0.3005	0.0642	0.0080
	T_{jitter}	0.0002	0.0031	0.0000	0.0002
1 ms	T_{period}	1.0000	1.0047	0.95959	0.0004
	T_{trans}	0.0769	0.7551	0.0685	0.0090
	T_{jitter}	0.0003	0.0047	0.0000	0.0003
0.5 ms	T_{period}	0.5000	1.0747	0.1735	0.0058
	T_{trans}	0.0885	0.9923	0.0666	0.0157
	T_{jitter}	0.0007	0.5747	0.0000	0.0057

As shown by the experiment results, the average actual period in all expected cycle time was able to meet the corresponding target.
The highest average jitter at 0.0011 ms is found in the task running at 4 ms expected cycle. The overall highest jitter occurs when the control task is running at 0.5 ms expected cycle at 0.5747 ms. Moreover, the transaction time that is supposed to be within the boundary of the actual period shows questionable results where it is more than the actual and expected cycle time at 0.9923 ms.

As expected, the distribution plot above shows that majority of the data samples are not equal to the expected cycle time when the real-time control task is in 4 ms and 0.5 ms cyclic operation.

Based on the results, the most optimized period for an EtherCAT control task is advised to be within 1 and 2 ms periodic cycle. Moreover, the experiment was operated using the generic driver that comes with the IgH EtherCAT Master in user space. Less end-to-end delay could be accomplished if a native driver is implemented. In case of the jitter, the results show relatively low values compared to an established work in [9]. This proves that i.MX6Q SABRELite is a viable embedded real-time controller for industrial control systems using an open-source EtherCAT Master solution.

Conclusion

In this paper, an open-source EtherCAT Master was implemented on an embedded board using dual-kernel approach with the latest versions of Xenomai and embedded Linux. Performance analysis was conducted for the system in terms of periodicity, the jitter, and the overall transaction time of commands sent and received from a slave communicated with CANopen protocol.

The open-source EtherCAT Master, IgH EtherLab was stacked on top of an i.MX6Q SABRELite board updated to the latest bootloader version to accommodate Linux kernel versions that require DTB.

Using the developed master, experiments were conducted in various expected cycle time to test its performance with a single slave and the data are acquired with a C# based application using UDP for control and monitoring functions.

The experiment results show that the embedded EtherCAT Master mostly stable in cyclic task with period between 1 and 2 ms. Nonetheless, average jitter and transaction time validates that the EtherCAT Master constructed from open source software is applicable to industrial control systems in comparison to established researches in [9-10].

In our future research, we will extend our analysis by studying on the overall time constraints on the slave, which is essential in rigorous and synchronized control especially in advanced robotic systems. In addition, communication between EtherCAT masters and application of the EtherCAT Automation Protocol in cloud environment is another interest on hand.

Acknowledgment

This study was financially supported by Seoul National University of Science and Technology.

References

1. J.-D. Decotignie, “Ethernet-based real-time and industrial communications,” Proceedings of the IEEE, vol. 93, no. 6, pp. 1102-1117, 2005.
2. G. Prytz, “A performance analysis of EtherCAT and PROFINET IRT,” Proc. 13th IEEE Int’l Conf. Emerging Technologies and Factory Automation (ETFA), 2008.
3. L. Seno, and C. Zunino, “Real-Time Ethernet networks evaluation using performance indicators,” Proc. 14th IEEE Int’l Conf. Emerging Technologies and Factory Automation (ETFA), 2009.
4. P. Ferrari, A. Flammini, D. Marioli, and A. Taroni, “ A distributed instrument for performance analysis of real-time Ethernet networks,” IEEE Transactions on Industrial Informatics, vol. 4, no. 1, pp. 16-25, 2008.
5. M. Cereia, I.C. Bertolotti, and S. Scanxio, “Performance Evaluation of an EtherCAT Master using Linux and the RT Patch,” Proc. 19th IEEE Int’l Symp. Industrial Electronics (ISIE), 2010.
6. EtherCAT Group, https://www.ethercat.org/default.htm.
7. IgH EtherLab, http://www.etherlab.org/en/index.php.
8. Xenomai Project, http://xenomai.org/.
9. M. Cereia, I.C. Bertolotti, and S. Scanxio, “Performance of a real-time EtherCAT Master under Linux,” IEEE Transactions on Industrial Informatics, vol. 7, no. 4, pp. 679-687, 2011.
10. M. Sung, I. Kim, and T. Kim, “Toward a holistic delay analysis of EtherCAT synchronized control processes,” International Journal of Computers, Communications and Control, vol. 8, no. 4, pp. 608-621, 2013.
11. Adeos Project, http://home.gna.org/adeos/.
12. Embedded Linux Wiki, http://elinux.org/Device_Tree.
13. M. Rostan, J.E. Stubbs, and D. Dzilno, “EtherCAT enabled advanced control architecture,” Proc. 21st IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC), 2010.
14. Freescale Semiconductors, http://www.freescale.com/.
15. Boundary Devices Repository, https://github.com/boundarydevices/u-boot-imx6/.
16. Minimal Ubuntu, http://rcn-ee.com/rootfs/eewiki/minfs/.