PORTING OPEN SOURCE ETHERCAT MASTER SOFTWARE STACK TO ZEDBOARD RUNNING FREERTOS

[1]CHUAN-CHING SUE, [2]JHONG-WEI SYU

[1,2]Dept. of CSIE, National Cheng Kung University, Tainan, Taiwan.
E-mail: [1]suecc@mail.ncku.edu.tw, [2]40043120@gm.nfu.edu.tw

Abstract - Real-Time Ethernet was developed to address the numerous shortcomings of conventional fieldbus standards. EtherCAT is the most widely applied Real-Time Ethernet protocol, due to its real-time operability, synchronization performance, and high bandwidth utilization. However, most existing open-source EtherCAT master software stacks are on the Linux platform, and therefore require specific extensions to enable the precision scheduling of periodic real-time tasks. Implementation of the EtherCAT embedded master in this paper is achieved by modifying interfaces between SOEM software stack and FreeRTOS RTOS and between FreeRTOS RTOS and ZedBoard hardware. We also discuss various issues encountered in the control and synchronization of multiple high-precision EtherCAT slave motors. The implemented EtherCAT embedded master enables the scheduling of periodic real-time tasks during every 125μs and reduce maximum scheduling jitter to just 1.622μs.

Keywords - Real-Time Ethernet, EtherCAT, Embedded, RTOS.

I. INTRODUCTION

Since 1980, the trend in industrial communication protocols has been on the development of fieldbus standards in accordance with the applicationsdeveloped by various companies[1] (e.g., P-NET, PROFIBUS, WorldFIP, Foundation Fieldbus, ControlNet, Interbus and CANOpen). Conventional fieldbus standards have numerous shortcomings [2-3], such as closed networks, a lack of interoperability between devices in different companies, limitations on the number of nodes in each network segment, and a lack of precision control. Fieldbus standards are being gradually replaced by Ethernet standards [4-5], which enable open system connectivity (OSI), a high degree of interoperability, high-speedtransmission, support for TCP/UDP/IP applications, and resistance to interference.

The fact that Ethernet systems based on CSMA/CD mechanism cannot guarantee deterministic real-time control led to the development of Real-Time Ethernet in 2008 [6], under the IEC 61158 fieldbus standard (e.g. PROFINET, Ethernet Powerlink (EPL), SERCOS III, Modbus/TCP, Ethernet/IP and EtherCAT).

Among the Real-Time Ethernet (RTE) [7], EtherCAT is the most widely applied because it provides the best performance with regard to real-time operability and bandwidth utilization [8-10]. EtherCAT networks comprise multiple high-precision slave motors and an EtherCAT master. The EtherCAT master requires thescheduling of periodic real-time tasks [11-12] to enable the synchronization of multiple high-precision slave motorsin order to achieve real-time control. Periodic real-time tasks deal with two main issues: (1) Management of control signals based on state information of EtherCAT slaves and (2) Communications processes including the transmission of EtherCAT frames via Network Interface Card (NIC). Real-Time Operating Systems (RTOSs) make it possible to schedule the execution of periodic real-time tasks and estimate the worst-case execution time, which makes them suitable for EtherCAT master.

Nevertheless, most open-source EtherCAT master software stack (e.g. IgH [13] and SOME [14]) was developed on the Linux platform, which requires specific extensions to enable the precision scheduling of periodic real-time tasks. Our proposed development of the EtherCAT embedded master was based on the modification of interfaces between SOEM Software Stack and FreeRTOS RTOS [15]as well as between FreeRTOS RTOS and ZedBoard hardware [16] in accordance with the EtherCAT Master Architecture.This scheme achieves maximum scheduling jitter of only 1.622μs (only 6.4% of a 250μs scheduling period).

In this paper,we designed a novel EtherCAT embedded master aimed at synchronizing multiple high-precision EtherCAT slave motors to enable multi-axis motion control using a 125μs scheduling period and discussed issues encountered in the control and synchronization of multiple high-precision EtherCAT slave motors.

The remainder of this paper is outlined in the following. Section 2 discusses our motivation behind this research. Section 3 introduces the proposed EtherCAT embedded master based on the EtherCAT Master Architecture, integrating SOEM Software Stack, FreeRTOS RTOS, and ZedBoard hardware. Section 4 presents the tests used to verify the efficiency of the EtherCAT embedded master. Finally, Section 5 concludes the paper and discusses future work.

II. MOTIVATION

Cereia, Cibrario, and Scanzio [17-18]executed periodic real-time tasks on the open-source IgH EtherLab EtherCAT master component under Linux. Periodic real-time tasksareresponsible for sending the EtherCAT frame to digital I/O terminal slaves. A

1000µs scheduling period results in maximum scheduling jitter of 1434µs without interference from other best-effort applications. They also set a 1000µs scheduling period and use real-time extensions (e.g. RT Patch and RTAI) to reduce maximum scheduling jitter to just 2.489µs without interference from other best-effort applications. They also set a 250µs scheduling period with maximum scheduling jitter of 5.549µs. Nonetheless, IgH contains a great deal of code running on the kernel space, which makes it hard to understand, modify, and maintain.

Andreas [19] surveyed all existing EtherCAT master software stacks in an attempt to implement an EtherCAT master; however, the resulting solution lacks many of the functionalities of previous schemes. The difficulties encountered in the development of EtherCAT masters can be divided as follows: (1) The time-consuming task of studying many specifications and datasheets; (2) Tracing an open-source EtherCAT master software stack to elucidate implementation in accordance with the specifications; and (3) Defining the interfaces between the open-source EtherCAT master software stack and operating system as well as between the operating system and hardware platform.

To the best of our knowledge, no existing EtherCAT master is able to schedule periodic real-time tasks every 125µsto enable high-precision control over EtherCAT slave motors. We also developed rapidly an EtherCAT embedded master based on an EtherCAT Master Architecture, without the need for additional real-time extensions to minimize scheduling jitter.

III. EtherCAT MASTER ARCHITECTURE

EtherCAT Master Architectures come as a general architecture (4 layers including PC, Linux Operating System, EtherCAT Master Software Stack and Application) or embedded system architecture (5 layers including Embedded Platform, Ethernet NIC Driver, Real-Time Operating System, EtherCAT Master Software Stack and Application), as shown in Fig. 1.

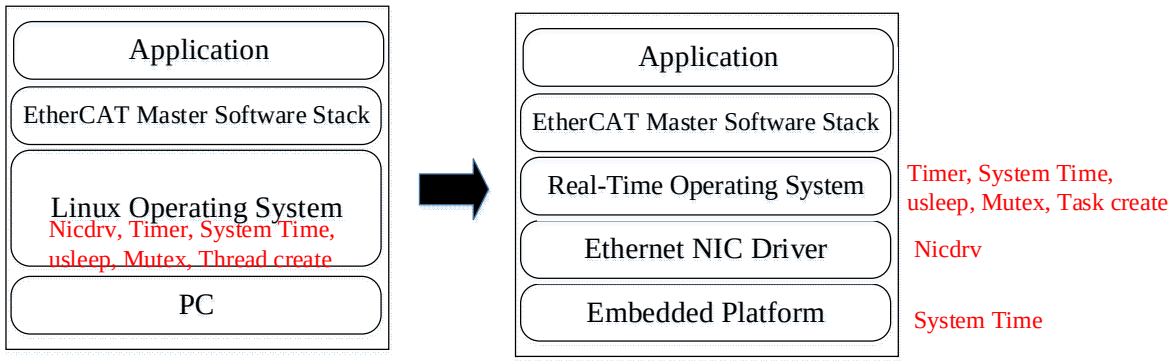

Fig. 1: EtherCAT Master Architecture from PC to Embedded Platform

In the general architecture, the Linux Operating System provides a Timer, System Time, usleep, Mutex, Thread create and Nicdrv. The embedded system architecture provides the functions of an Embedded Platform, Ethernet NIC Driver, and Real-Time Operating System, as shown in Fig. 1.

A. Embedded Platform

Layer 1, the ZedBoard hardware [16],is used to replace the PC because it is more easily combined with FreeRTOS to enable the scheduling of periodic real-time tasks. The ZedBoard hardware is also better suitedto special applications, such as motor control and software acceleration.

The ZedBoard hardware includes a Dual-core ARM Cortex A9 and board support package (BSP). The ARM Cortex A9 uses a decreasing counter as a 3ns resolution hardware timer (hTimer) to provide a reference to System Time. The BSP provides an Ethernet NIC Driver, and real-time operating system.This makes it possible to modify the interfaces among the Ethernet NIC Driver, real-time operating system (RTOS), and EtherCAT master software stack quickly and easily.

B. Ethernet NIC Driver

BSP is in Layer 2, providing Ethernet NIC driver (Nicdrv) to replace the Ethernet NIC driver within the Linux kernel. We modified LWIP ICMP example code to make it possible to send a 64~1518-byte EtherCAT frame and receive EtherCAT response frames from the slave.

In the LWIP ICMP example code, the DMA receiving handler inserts ping request packets into the packet queue of the main memory, when a ping request packet is received by the NIC RX Ring Buffer. Mailbox task delivers the ping request packet to ICMP task to be copied to the RX Buffer in main memory using memcpy, whereupon the buffer in the packet queue is de-allocated.

LWIP ICMP example code is used to ensure the availability of the TX Ring Buffer in NIC to send the ping response packet in the TX Buffer (i.e. main memory) via NIC. As long as the buffer is available, the DMA sending handler transmits the ping response packet to the TX Ring in the NIC, and request the NIC to send a ping response packet, as indicated by the blue block in Fig. 2.

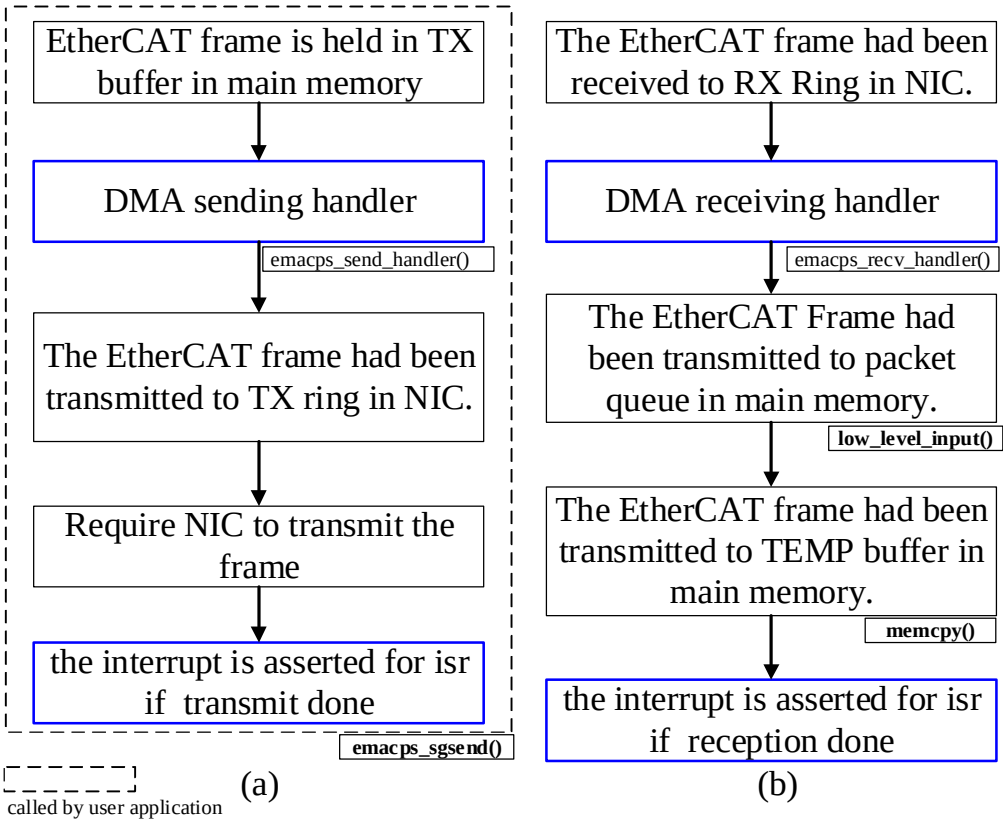

Fig. 2: Procedure used in sending and receiving EtherCAT Frame

We refer to a process flow of the ping response packet to send an EtherCAT frame in the TX Buffer. The TCP/IP header is removed to produce a Raw Socket to send a 64~1518-bytes EtherCAT frame, as shown inFig. 2(a).
We refer a process flow of the ping request packet to receive the EtherCAT response frame in RX Buffer. We must continue monitoring the packet queue in the main memory for the exchange of process data. In the event of an incoming packet, it is copied (in packet queue) to a Temp Buffer (in main memory) using memcpy, and the buffer in packet queue is de-allocated. The packet is then copied to the RX Buffer in main memory using memcpy if the packet in Temp Buffer is an EtherCAT Frame, as shown inFig. 2(b).

C. Real-time Operating System

Layer 3, FreeRTOS RTOS [15] v8.2.3 replaces the Linux Operating System to provide Timer, System Time, usleep, Mutex and Task create.
The Timer provides a software tick count (sTick) as a timeout reference. Until the timeout is exceeded, the master continues sending the frame until it is received. In tick ISR, sTick increases by 1 if a timeout occurs. In cases where the timeout threshold is low, the tick ISR is triggered more frequently, which generally gives FreeRTOS RTOS high overhead. To prevent excessive overhead, we set the timeout to 2ms. The resolution of the hardware timer is 3ns; therefore, the hardware initial counter value (hIcnt) is set at 666666, i.e., 2ms (≅ 3ns × hIcnt).
The EtherCAT Master must synchronize the system time of all slaves; therefore, we obtain system time using sTcik and hTimer.

System Time is calculated as follows:

$$\text{System Time} = [\text{sTick} \times \text{hIcnt} + (\text{hIcnt} - \text{hTimer})] \times 3\text{ns}$$

In the configuration of a slave, the usleep function provides a time interval between frames complied in accordance with EtherCAT specifications.
Mutex is used for the allocation of frame buffer resources (i.e. TX Buffer and RX Buffer), where TX Buffer and RX Buffer are pre-allocated in main memory (i.e. static memory space). In seeking EtherCAT frame transmission, the EtherCAT master must obtain two buffers from TX Buffer and RX Buffer. EtherCAT master must ensure that two buffers are not allocated repeatedly.
The Task create function is used for the creation of periodic real-time tasks. The software tick handler (i.e. the tick ISR) increases sTick and is responsible for the scheduling of periodic real-time tasks. If all tasks have the same priority, then time-sharing is used for scheduling; otherwise, higher priority tasks are executed first.

D. EtherCAT Master Software Stack

Layer 4is theSimple Open EtherCAT Master (SOEM) v1.3.1. Developing the EtherCAT embedded master requires a comprehensive understanding of the interfaces between "SOEM and FreeRTOS" and

"FreeRTOS and ZedBoard".
SOEMrequires Nicdrv, Timer, System Time, usleep, Mutex, and Thread create, provided by the System call of Linux (general architecture); however,it would be preferable to implement these functions by modifying the FreeRTOS application interface (API), as shown in Table 1.

Description	System call of Linux	FreeRTOS API
Nicdrv	send()	**emacps_sgsend()**
	recv()	**low_level_input() and memcpy()**
Timer	gettimeofday()	**xTaskGetTickCount()**
System time	gettimeofday()	**xGetCurrentTime()**
usleep	nanosleep()	**usleep()**
Mutex	pthread_mutex_lock() pthread_mutex_unlock()	xSemaphoreTake() xSemaphoreGive()
thread create	pthread_create()	xTaskCreate()

Table 1:System call of Linux map to the FreeRTOS API

E. Application

The Layer 5, application deals with CoE access to the PDO to control the slave.
For example, the 2nd RxPDO includes Controlword and Target position. Controlword refers to the control signal and Target position refers to the rotation of the slave motor into a specific position, as shown in Table 2.

Index	Sub	Name	Default value (IndexData Length)
1601_h	0	Number of mapped objects	3
	1	Controlword	60400010_h
	2	Target position	$607A0020_h$

Table 2:2nd Receive PDO for Motion Control

In another example, the 2nd TxPDO includes Statusword and the Actual position. Statusword refers to the current status signal of the slave, whereas Actual position refers to the(required) current position of the slave motor, as shown in Table 3**Error! Reference source not found.**.

Index	Sub	Name	Default value (IndexData Length)
$1A01_h$	0	Number of mapped objects	3
	1	Statusword	60410010_h
	2	Actual position	60640020_h

Table 3: 2nd Transmit PDO for Motion Control

IV. RESULTS

In order to confirm the correctness of the system behavior and to determine whether the system meets he specifications, we will test and verify the system in the following subsections.

A. Testing and Verification

Testing is to ensure that the correctness of the EtherCAT protocol related application. We adopted two-axis motion control to demonstrate the efficiency of the EtherCAT embedded master based on an embedded system architecture. Our aim was to check the main functionalities, including Network configuration, Mailbox protocol CoE, Distributed Clocks with master synchronization, and Cyclic PDO.
Verification is to ensure that the system meets the expected specification. The verification in the paper is a measure of the scheduling jitter for the embedded master. We measured the maximum scheduling jitter with real-time task period of 1000µs, 500µs, 250µs, 125µs and 60µs. The maximum scheduling jitter was less than the real-time period, which means that our system conforms to the specification and it can control the slaves properly.
In these experiments, we used two Yaskawa SGDV-2R8AE1A SERVOPACK[20]respectively connected to a SGMAV-04ADA61 motor as two EtherCAT Slaves. Fig. 3 presents a flowchart showing the EtherCAT embedded master used for the configuration and control of each EtherCAT slave.
Each slave provides four sets of PDO mappings. We employed multi-axis motor synchronous control; therefore, all of the slaves use 2nd PDO mapping, as previously shown in Tables 2 and 3.
The minimum SYNO0 Cycle Time of the slaves is 125µs; therefore, the EtherCAT embedded master must collaborate with the slaves for real-time control using a 125µs real-time period. As shown in Fig. 4, the real-time periodic tasks involve sending a 100-byte EtherCAT frame for the exchange of PDO data. The time remaining in the real-time period is calculated as follows: 125µs real-time period minus the time used for the management of control signals and communication processes. For the sake of simplicity, the remaining available period is used to execute usleep() to replace the transmission of aperiodic data.

Scheduling jitter is measured as follows: currentsystemtime – the lastsystemtime - real-time period. We used 10^6 samples to measure scheduling jitter associated with periodic real-time tasks.When using a real-time period of 125µs, the maximum scheduling jitter was 1.885µs. This should be acceptable for most real-time control applications, as shown in Table 4.

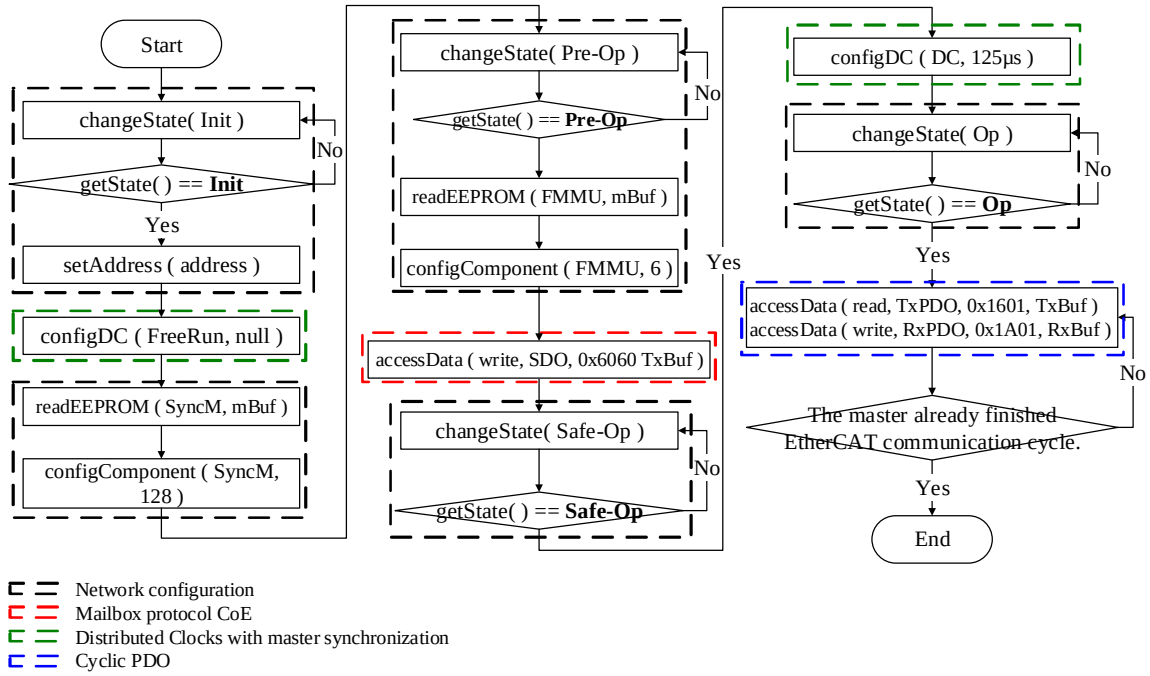

Fig. 3: Flowchart illustrating servo synchronization for each EtherCAT slave

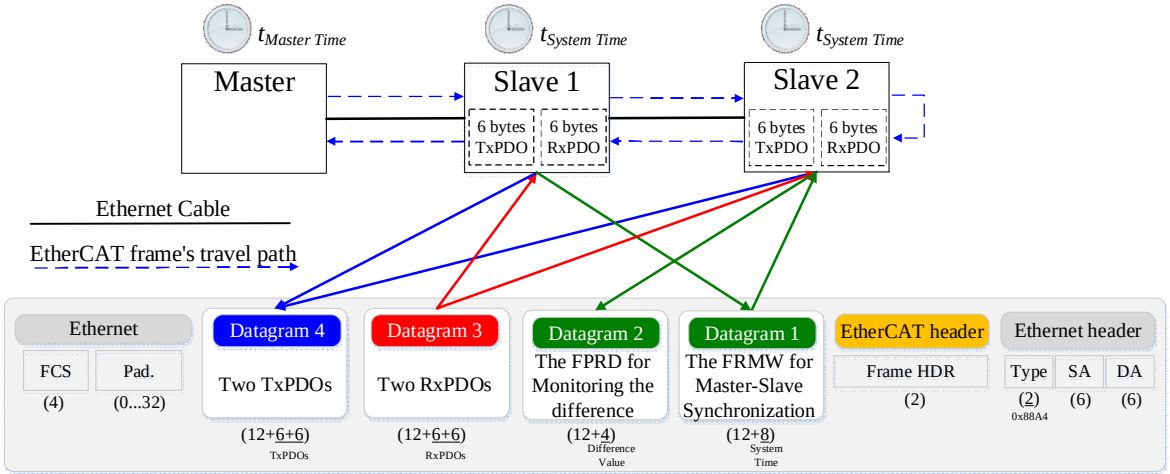

Fig. 4: Cyclic process data communication between master and two slaves

	1000µs	500µs	250µs	125µs	60µs
μ_J	-0.516	-0.056	0.174	0.301	0.360
σ_J	0.234	0.238	0.233	0.230	0.228
Min_J	-1.156	-0.659	-0.421	-0.281	-0.219
Max_J	0.968	1.432	1.733	1.885	1.929

Table 4: Statistics of scheduling jitter under real-time periods

The frequency distribution of scheduling jitter with a 125µs real-time period is shown in Fig. 5.

To verify the time synchronization between the EtherCAT master and the EtherCAT slave, the EtherCAT embedded master continuously sends the EtherCAT frame using varying periods to correct the system time difference value between the slave and the Reference Clock to less than 100ns. The experimental results show that the slave converges to 100ns in a 110ms period, as shown in Fig. 6.

For comparison, we installed Ubuntu 16.04 on a PC with an Intel Core2Duo E6500 dual-core CPU running at 2.93 GHz with 1964 MB of dual-channel DDR2 RAM and ranSOEM v1.3.1 on a standard Linux PC, finding that scheduling jitter appeared to be too high.

As shown in Table 5, when the real-timeperiod was1000µs, the maximum scheduling jitter was 1732.0µs (far

exceeding 1000µs). Such high scheduling jitter would cause the slaves to generate an error signal if the SYNC0 Cycle Time were set to 1000µs.

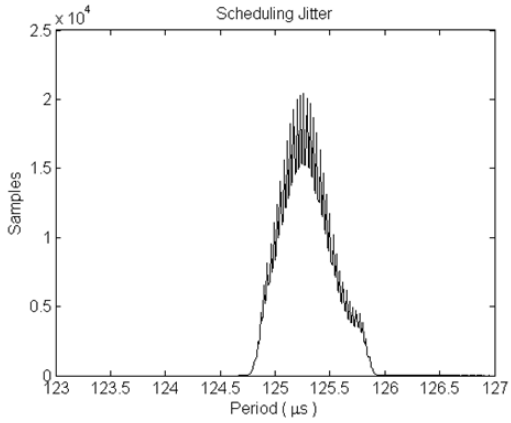

Fig. 5: Frequency distribution of real-time period of 125 µs

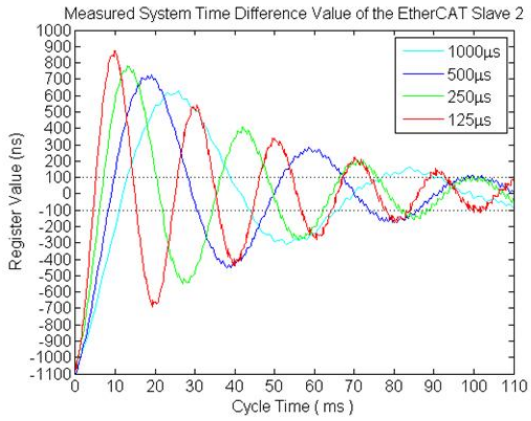

Fig. 6:System Time Difference Value of EtherCAT Slave 2

	μ_J	σ_J	Min_J	Max_J
1000µs	5.944µs	18.696µs	3.0µs	1732µs

Table 5: Statistics of scheduling jitter J under real-time period of 1000µs

CONCLUSIONS

The EtherCAT protocol provides highly satisfactory real-time operations, excellent synchronization, and maximum bandwidth utilization. We developed a novel EtherCAT Master Architecture to overcome restrictions involved in using real-time extensions, using an EtherCAT master embedded through the modification of interfaces between SOEM Software Stack and FreeRTOS RTOS and between FreeRTOS RTOS and ZedBoard hardware. The maximum scheduling jitter of the proposed scheme is 1.929µs, which should be acceptable for any real-time control applications.

This paper makes contributions inthree areas. We developed an EtherCAT embedded master that enables the scheduling of periodic real-time tasks using a 125µs real-time period. We were able to integrate the three fundamental layers: SOEM software stack, FreeRTOS RTOS and ZedBoard hardware. We also reduced scheduling jitter.

Our future work will focus on two areas of research: (1) Implementing the other functionalities that are not supported by SOEM(EoE and SoE); also include the redundancy functionality because the redundancy functionality provided by SOEM was removed during the development of the EtherCAT embedded master. In the future, the ZedBoard hardware will be used to extend multi-port Ethernet to recover the redundancy functionality and (2) we will also seek to accelerate communication processing through the manipulation of EtherCAT frames and cyclic executions in FPGA. The FPGA provided by the ZedBoard hardware can be used to accelerate frame generation and frame disassembly.

ACKNOWLEGEMENT

This work was supported in part by the Ministry of Science and Technology, Taiwan, R.O.C., under Grant MOST106-2221-E-006-008

REFERENCES

[1] M. Felser, T. Sauter, "The fieldbus war: history or short break between battles?," in Proceedings, IEEE International Workshop Factory Communication Systems, pp. 73-80, 2002.

[2] J. Kay, R. Entzminger, and D. Mazur, "Industrial ethernet-overview and best practices," in Pulp and Paper Industry Technical Conference, pp. 18-27, Jun. 2014.

[3] J. Kay, R. Entzminger, D. Mazur, "Industrial ethernet: Overview and application in the forest products industry," IEEE Industry Application Magazine, pp. 54-63, 2015..

[4] B. Galloway, G. Hancke, "Introduction to industrial control networks," IEEE Communications Surveys and Tutorials, vol. 15, no. 2, pp. 860-880, 2013.

[5] M. Felser, T. Sauter, "Standardization of industrial Ethernet—The next battlefield?," in Proceedings, IEEE International Workshop Factory Communication Systems, pp. 413-421, Sep. 2004.

[6] M. Rostan, "Industrial Ethernet Technologies: Overview and Comparison," ETG Industrial Ethernet Seminar Series, Nuremberg, Nov. 2008.

[7] J.-D. Decotignie, "Ethernet-based real-time and industrial communications," in Proceedings, IEEE International, vol. 93, no. 6, pp. 1102-1118, Jun. 2005.

[8] M.Felser, "Real-time Ethernet – Industry prospective," in Proceedings, IEEE International, vol. 93, no. 6, pp. 1118-1129, Jun. 2005.

[9] H. Büttner, D. Janssen and M. Rostan, "EtherCAT - the Ethernet fieldbus," PC Control Magazine, pp.14-19, March 2003.

[10] D. Jansen, H. Buttner, "Real-time ethernet the EtherCAT solution," Computing and Control Engineering, vol. 15, pp. 16-21, Feb. 2004.

[11] S. Potra and G. Sebestyen, "EtherCAT protocol implementation issues on an embedded linux platform," in Proceedings,IEEE International Conference, vol. 1. pp. 420-425, 2006.

[12] M. Felser, "Real time Ethernet: Standardization and implementations,"in Proc. IEEE Int. Symp. Ind. Electron., pp. 3766–3771, 2010.
[13] IgH EtherCAT master Reference Manual [Online]. Available: http://www.etherlab.org/en/ethercat/
[14] Open EtherCAT Society [Online]. Available: http://openethercatsociety.github.io/
[15] FreeRTOS [Online]. Available: http://www.freertos.org/
[16] Zynq-7000 All Programmable SoC Technical Reference Manual [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf
[17] M. Cereia, I. Cibrario Bertolotti, and S. Scanzio, "Performance evaluationof an EtherCAT master using Linux and the RT Patch," in Proc.IEEE Int. Symp. Ind. Electron., pp. 1748–1753, Jul. 2010.
[18] M. Cereia, I. Cibrario-Bertolotti, and S. Scanzio, "Performance of a real-time EtherCAT master under Linux," IEEE Transactions on Industrial Informatics, vol. 7, no. 4, pp. 679-687, Nov. 2011.
[19] Andreas Tågerud, "Implementation of an EtherCAT Master," Master Thesis, pp. 1-76, Sep. 2011.
[20] AC Servo Drives Σ-V Series USER'S MANUAL EtherCAT (CoE) Network Module Model: SGDV-OCA01A [Online]. Available:http://www.innovativeidm.com/ResourceManager.aspx?FileName=TM.YMOT.SIEPC72082904.pdf&FileType=7

★★★