Journal of KIIT. Vol. 18, No. 2, pp. 61-67, Feb. 28, 2020. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2020.18.2.61

ARM 기반 임베디드 리눅스를 위한 정밀한 시간 측정 방법

권영우*, 최준영**

Accurate Time Measurement Method for ARM-based Embedded Linux

Young-Woo Kwon*, Joon-Young Choi**

본 논문은 BK21플러스, IT기반 융합산업 창의인력양성사업단에 의하여 지원되었고 2019년도 정부 (교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-20181D1A3B07043053).

요 약

본 논문에서는 ARM 기반 임베디드 리눅스 시스템에서 코프로세서의 CCNT(Cycle Count) 레지스터를 이용하여 상대적인 시간 간격을 정밀하게 측정하는 방법을 제안한다. 측정하고자 하는 프로그램의 시작과 끝에 CCNT 레지스터 값을 읽고 저장하여 차이를 계산함으로써 프로그램의 실행시간을 측정한다. CCNT 레지스터는 프로세서의 고속 클럭에 동기하여 동작하기 때문에 고해상도의 시간을 정밀하게 측정할 수 있고, CCNT 레지스터 값을 읽는 함수를 어셈블리어로 작성하여 시간 측정 지연을 최소화함으로써 실시간 측정 성능을 향상시킨다. 임베디드 리눅스에서 시간 측정을 위해 일반적으로 사용되는 getnstimeofday 함수와 성능 비교 실험을 통해 제안한 방법이 10배 이상 정밀하고 신속한 시간 측정 성능을 달성하는 것을 검증한다.

Abstract

We propose a method to accurately measure relative time intervals in ARM-based embedded Linux systems using CCNT (Cycle Count) register of the coprocessor. The program execution time is measured by reading and storing CCNT register values at the beginning and end of the program to be measured and calculating the difference. Since the CCNT register counts synchronized with the processor's high-speed clock, it can accurately measure high-resolution time. Moreover, the real-time measurement performance is improved by implementing the function that reads the CCNT register value in assembly language to minimize the time measurement latency. The performance comparison experiment with the getnstimeofday function, which is commonly used for time measurement in embedded Linux, verifies that time measurement performance of the proposed method is 10 times faster and more precise.

Keywords
measurement of execution time, embedded Linux, getnstimeofday, ARM processer

* 부산대학교 전자전기컴퓨터공학과 석사과정
- ORCID: http://orcid.org/0000-0001-5121-4314
** 부산대학교 전자공학과 교수(교신저자)
- ORCID: http://orcid.org/0000-0002-5160-3739

• Received: Jan. 03, 2020, Revised: Jan. 20, 2020, Accepted: Jan. 23, 2020
• Corresponding Author: Joon-Young Choi
Dept. of Electronic Engineering, Pusan National University, San 30 Jangjeon-dong, Geumjeong-gu, Pusan, 609-735, Korea.
Tel.: +82-51-510-3978, Email: jyc@pusan.ac.kr

Ⅰ. 서 론

대부분의 임베디드 시스템은 제한된 시간 안에 프로그램의 실행이 요구되고 더욱이 하드 리얼타임 임베디드 시스템에서는 프로그램이 오류 없이 실행될 뿐 아니라 정해진 시간에 정확하게 실행되어야 한다[1][2]. 이러한 프로그램 실행의 적시성은 특히 공장자동화, 항공기, 무기 제어, 레이더 신호처리, 메디컬 시스템 분야에서 매우 중요한 성능요인이 된다[3][4]. 한편 이러한 시간 제약조건을 만족하면서 스케줄링을 최적화하기 위해서는 프로그램의 실행시간을 정확하게 측정하는 것이 반드시 필요하다[5]-[7].

인텔 프로세서 기반 컴퓨팅 시스템의 경우 프로그램 실행시간을 정밀하게 측정하기 위해 RDTSC (Read Time Stamp Counter) 레지스터를 이용하는 방법을 사용한다. 즉 인텔 프로세서의 RDTSC 레지스터는 프로세서 클럭 사이클에 동기하여 동작하는데 측정하고자 하는 프로그램의 시작과 끝에 RDTSC 레지스터 값을 읽고 저장하여 차이를 계산하면 프로그램의 실행시간을 정밀하게 측정할 수 있다. 또한 최근 프로세서의 동작 주파수가 3GHz이상 인 것을 감안하면 매우 높은 해상도의 시간 단위로 정밀한 측정이 가능하다[8][9].

반면에 임베디드 리눅스 시스템의 경우 대부분 ARM 프로세서 기반으로 설계되므로 RDTSC 레지스터가 존재하지 않는다. 따라서 프로그램 실행시간을 측정하기 위해 리눅스에서 제공하는 gettimeofday, clock_gettime, getnstimeofday 시스템 호출 함수를 일반적으로 사용한다. gettimeofday의 경우 매우 오래전부터 많이 사용되던 함수이나 마이크로초 단위까지의 시간만 반환하므로 나노초 단위의 시간을 측정할 수 없는 단점이 있다[10][11]. 또한 유닉스 표준인 SUSv4-2008에서 제거 함수로 지정이 되어 가까운 장래에는 퇴출될 수 있어 가능한 사용을 지양해야 할 함수이다. clock_gettime, getnstimeofday 함수는 현재 시스템 시간을 나노초 단위까지 고해상도 시간을 반환하고 clock_gettime은 사용자 영역에서 getnstimeofday는 커널 영역에서 사용할 수 있다[12][13]. 그러나 시간 측정을 위해 커널 영역에서 제공하는 서비스를 이용해야 하므로 리눅스 스케줄러 알고리즘에 영향을 받고 예측이 어려운 상당한 크기의 함수 실행 지연이 발생할 수 있어 정밀하고 신속한 시간 측정이 불가능하다[14].

이러한 문제점을 극복하기 위해 본 논문에서는 ARM 기반 임베디드 리눅스 시스템에서 코프로세서의 CCNT(Cycle Count) 레지스터를 이용하여 어셈블리어로 작성된 함수로 레지스터 값을 읽어 정밀한 프로그램의 실행시간을 측정하는 방법을 제안한다. 전체 논문의 구성은 다음과 같다. 2장에서는 프로그램 실행시간을 측정할 때 발생하는 문제점을 설명하고, 3장에서는 정밀한 시간 측정을 위해 제안된 방법을 설명한다. 4장에서는 제안된 방법의 성능을 검증하기 위해 비교 실험을 수행하고 실험결과를 분석한다. 5장에서는 결론을 도출한다.

Ⅱ. 프로그램 실행 시간 측정

일반적인 임베디드 시스템에서 특정 프로그램의 실행시간을 측정하는 방법은 그림 1과 같이 프로그램을 실행하기 전에 시간(t_1)을 측정하고 프로그램이 끝나는 지점에 한 번 더 시간(t_2)을 측정하여 저장하고 차이를 계산하여 프로그램 실행시간을 측정한다.

그림 1의 t_P는 특정 프로그램의 실행시간을 나타내고 t_R은 시스템 시간 값을 얻는데 소요되는 시간 측정 지연이다. 즉 그림 1과 같이 실제로 프로그램 실행시간을 측정할 때 얻은 값은 프로그램의 실행시간과 시간을 측정하는데 소요되는 시간 측정 지연이 포함된 값이며 각 시간들의 관계는 다음과 같은 식으로 표현된다.

$$t_2 - t_1 = t_P + t_R \tag{1}$$

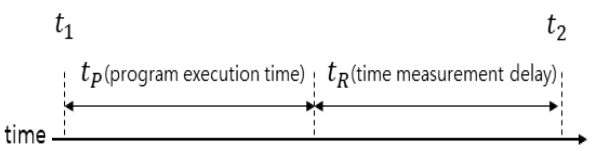

그림 1. 프로그램 실행시간을 측정하기 위한 타이밍 도표
Fig. 1. Timing diagram to measure program execution time

따라서 정확한 실행시간을 측정하기 위해서는 현재의 시스템 시간을 읽어오는 함수 자체의 실행 지연이 작거나 없는 함수를 사용하여 t_R을 최소화하는 것이 반드시 필요하다.

III. 시간 측정 방법

3.1 getnstimeofday 함수

임베디드 리눅스에서 일반적으로 시간을 측정하기 위해 gettimeofday 함수를 많이 사용하나 현재 시스템 시간을 마이크로초 단위 시간까지만 지원하므로 본 논문에서는 나노초 단위까지 측정 가능한 getnstimeofday 함수를 사용한다. 이 함수를 사용하면 임베디드 리눅스의 timespec 구조체에 현재 시스템 시간의 초와 나노초를 나누어서 저장하여 반환한다.

3.2 CCNT 레지스터

CCNT 레지스터는 ARM 기반 프로세서에서 System Control Coprocessor의 레지스터이며 프로세서의 클럭 사이클에 따라 레지스터의 값이 1씩 증가하는 카운터이다. 따라서 CCNT 레지스터를 이용하여 시간을 측정하기 위해서는 프로세서의 클럭 사이클을 정확히 알고 있어야 하고 클럭 사이클의 주파수가 높을수록 고해상도의 시간을 측정할 수 있다. 그러나 CCNT 레지스터는 모든 ARM 기반 프로세서에서 지원하는 것이 아니며 Cortex-A 및 Cortex-R 시리즈의 경우에는 Cycle Count Register라는 이름으로 지원하며 Cortex-M 시리즈의 경우 DWT_CYCCNT(Data Watchpoint and Trace Cycle Count) 레지스터라는 CCNT와 동일한 역할을 하는 레지스터를 지원한다.

CCNT 레지스터를 사용하기 위해서는 먼저 PMNC(Performance Monitor Control) 레지스터에서 모든 카운터를 활성화, 모든 performance 카운터 초기화, CCNT를 리셋 한다. 그 다음 Count Enable Set (CNTENS) 레지스터에서 CCNT를 활성화하고 마지막으로 Overflow Flag Status(FLAG) 레지스터에서 기록되어 있는 오버플로우 값을 모두 0으로 초기화한다.

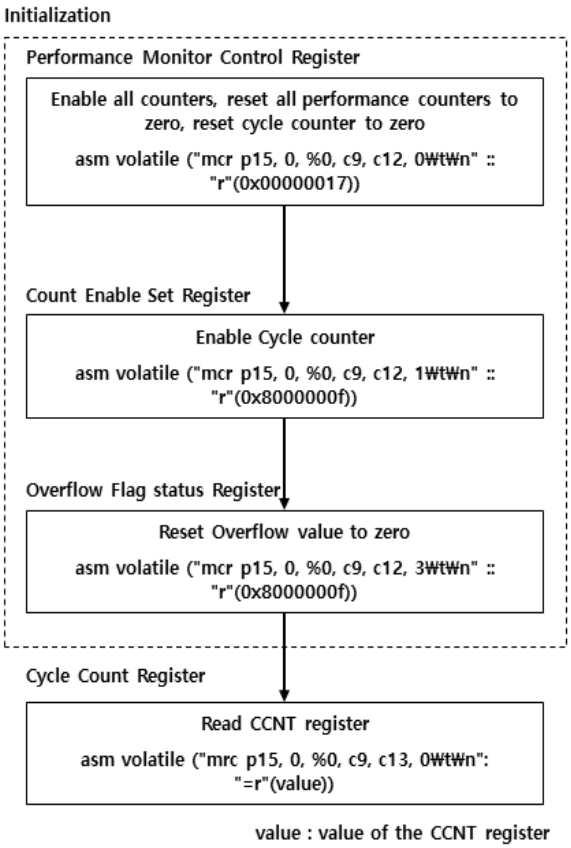

그림 2. CCNT 레지스터를 사용하기 위한 전체 과정
Fig. 2. Overall process for using the CCNT register

이러한 초기화 과정이 완료되면 CCNT 레지스터는 프로세서 클럭과 동기하여 동작하게 되고 CCNT 레지스터 값을 어셈블리어로 작성한 함수를 사용하여 읽어 시간을 측정하면 고해상도의 시간을 측정 지연이 거의 없이 얻을 수 있다. 제안한 CCNT 레지스터를 사용하기 위한 전체 절차와 각 단계에서 사용된 어셈블리어 코드를 정리하여 나타내면 그림 2와 같다.

IV. 실 험

4.1 실험 환경 및 방법

제안된 시간 측정 방법의 성능을 검증하기 위해 두 종류의 실험을 수행한다. 첫 번째는 임베디드 리눅스에서 시간 측정 실험을 수행하고 두 번째는 공장자동화에 많이 사용되는 이더캣 네트워크에서 시간 측정 실험을 수행한다.

실험에서는 TI사의 ARM 기반 AM3358 프로세서를 장착한 비글본 블랙보드를 사용하고 임베디드 리눅스 커널은 RT패치를 수행한 4.9.59 버전을 설치하였다. 프로세서의 동작 주파수는 1GHz이므로 CCNT 레지스터의 카운터는 1 나노초마다 증가한다. 그리고 각 종류의 실험을 수행하기 전에 CCNT 레지스터를 초기화하여 리셋을 완료한 후 실험을 진행하고, 실험 도중 CCTN 레지스터의 오버플로우가 발생하면 오버플로우를 고려하여 레지스터 값을 인식한다. 한편 이더캣 네트워크의 시간 측정 실험에서 마스터는 앞에 언급한 비글본 블랙보드에서 동작하고 슬레이브는 TI사의 ARM 기반 AM3359 프로세서를 장착한 icev2 보드에서 동작하며 비글본 블랙 보드와 icev2 보드의 이더넷 장치를 랜선으로 연결하여 이더캣 네트워크를 구성한다.

임베디드 리눅스에서의 실험은 비글본 블랙보드 1대를 사용하고 getnstimeofday 함수와 CCNT 레지스터를 사용할 경우 각각 시간을 얻기 위해 발생하는 시간 측정지연을 비교하기 위해 각각 3종류의 실험을 수행한다. 첫 번째 실험은 연속으로 시간을 1000번 측정하고 그 각각의 차이 999개를 계산하여 시간 측정 지연을 추정하고 비교한다. 두 번째 실험은 1초주기마다 시간을 연속으로 2번 측정하고 이 과정을 1000번 반복하여 각 1000개의 시간 차이 값을 계산하여 시간 측정 지연을 추정하고 비교한다. 세 번째 실험은 1초 간격과 5초 간격을 교대로 시간을 연속으로 2번 측정하고 이 과정을 1000번 반복하여 1000개의 시간 차이 값을 계산하여 시간 측정 지연을 추정하고 비교한다. 이렇게 다양한 간격으로 시간을 측정하는 실험을 수행하는 이유는 각 시간 측정 함수가 호출 되었을 때 임베디드 리눅스의 스케줄링 알고리즘이 시간 측정 지연에 미치는 영향도 파악할 수 있기 때문이다.

한편 이더캣 네트워크에서 시간 측정 실험은 다음과 같이 수행된다. 이더캣 네트워크는 마스터 1대와 슬레이브 1대로 구성되고 마스터의 동작 주기를 100μs와 1ms로 설정하여 이더캣 네트워크를 동작시킨다. 이때 이더캣 마스터 응용프로그램의 시작부분에 매주기마다 시간을 측정하여 저장하고 1000개의 저장된 시간의 차이 값 999개를 계산하여 시간 측정 지연을 추정하고 비교한다.

4.2 실험 결과

첫 번째 임베디드 리눅스에서 수행한 시간 측정 실험 결과를 그림 3, 4, 5와 표 1에 나타낸다. 그림 3, 4, 5에서 왼쪽 그림은 getnstimeofday 함수를 오른쪽 그림은 CCNT 레지스터를 사용한 실험 샘플의 분포이다.

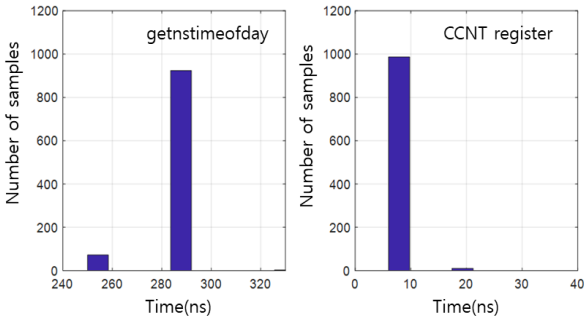

그림 3. 연속 시간 측정에 대한 측정 지연 분포
Fig. 3. Distributions of measurement latencies for consecutive time measurements

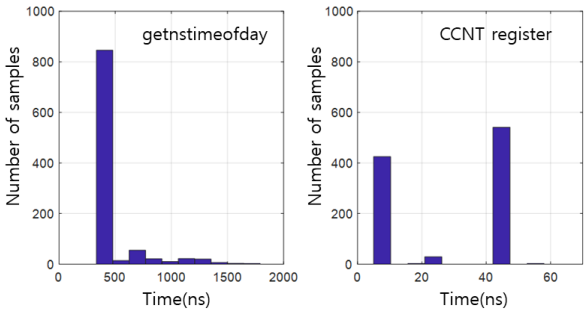

그림 4. 1초마다 연속 2번 시간 측정에 대한 측정 지연 분포
Fig. 4. Distributions of measurement latencies for consecutive two time measurements every 1 second

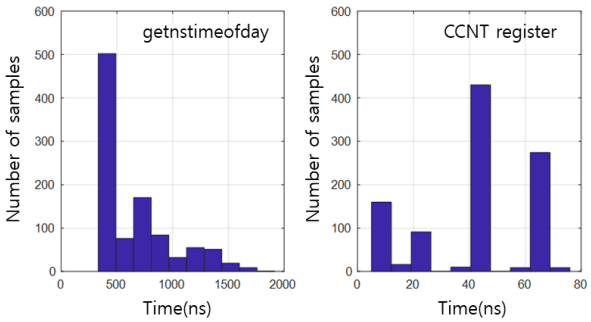

그림 5. 1초, 5초 마다 연속 2번 시간 측정에 대한 측정 지연 분포
Fig. 5. Distributions of measurement latencies for consecutive two time measurements every 1, 5 seconds

표 1. 임베디드 리눅스에서 시간 측정 지연 평균
Table 1. Average of time measurement latencies in embedded linux

Experiment \ Method	getnstimeofday	CCNT register
Consecutive time	288ns	6ns
Every 1 second	485ns	26ns
Every 1, 5 seconds	666ns	40ns

연속으로 시간을 측정할 때 측정지연의 분포는 그림 3과 같으며 getnstimeofday 함수를 사용하여 얻은 샘플의 평균은 288ns이며 CCNT 레지스터를 사용하여 얻은 샘플의 평균은 6ns로 측정되었다. 1초 주기마다 연속 2번 측정할 때 측정지연의 분포는 그림 4와 같고 getnstimeofday 함수를 사용하여 얻은 샘플의 평균은 485ns이며 CCNT 레지스터를 사용하여 얻은 샘플의 평균은 26ns로 측정되었다. 1초 간격과 5초 간격을 교대로 시간을 연속으로 2번 측정할 때 측정지연의 분포는 그림 5와 같고 getnstimeofday함수를 사용하여 얻은 연속 2개의 시간 차이값 평균은 666ns이며 CCNT 레지스터를 사용하여 실험한 경우 40ns로 측정된다.

측정 지연에 대한 평균값들을 표 1에 나타내고 이러한 결과로부터 제안하는 시간 측정 방법이 기존 함수를 사용하는 것보다 시간 측정지연이 10배 이상 작은 것을 확인할 수 있고 이는 제안하는 방법이 프로그램 실행 시간을 더욱 정밀하고 신속하게 측정하는 것을 검증한다. 또한, 표 1의 연속 시간 측정에서 제안된 방법의 경우 6ns 라는 매우 짧은 측정지연 결과를 얻었는데 이러한 결과는 1GHz의 클록에서 Cortex-A8의 2,000 MIPS 성능과 시간 측정 함수가 단일 명령어로 구성되는 상황을 고려하면 충분히 실현 가능하고 타당한 것으로 판단된다.

두 번째 이더캣 네트워크에서 수행한 실험결과를 그림 6과 표 2에 나타낸다. 그림 6은 100μs 주기로 이더캣 네트워크를 동작 시켰을 때 매 주기 사이의 측정 시간 차이에 대한 측정 분포를 나타낸다. 그림 6에서 왼쪽 그림은 getnstimeofday 함수를 사용한 결과이고 평균은 100.383μs로 계산되고 오른쪽 그림은 CCNT 레지스터를 사용한 결과이고 평균은 100.039μs로 계산된다.

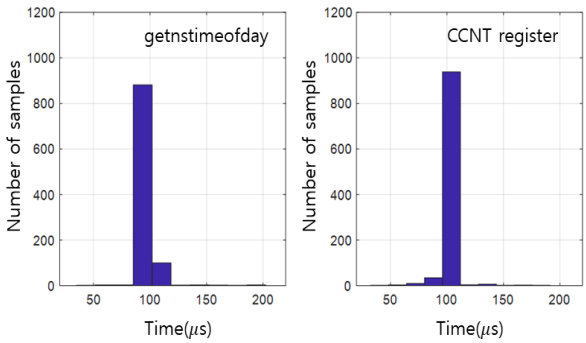

그림 6. 100μs 주기 이더캣 동작에 대한 주기 측정 분포
Fig. 6. Distributions of period measurement for 100μs period EtherCAT operation

표 2. 이더캣 네트워크에서 주기 측정 오차 평균
Table 2. Average of period measurement errors in EtherCAT network

Experiment \ Method	getnstimeofday	CCNT register
100μs period error	383ns	39ns
1ms period error	72ns	6ns

이렇게 구한 평균값과 원래 이더캣 네트워크의 주기와 차이는 주기 측정 오차로 추정할 수 있고 오차가 작을수록 시간 측정 방법의 성능이 우수한 것으로 판단할 수 있다. 이더캣 네트워크 동작주기 100μs, 1ms에 대한 주기 측정 오차를 표 2에 나타내며 모든 이더캣 동작 주기에서 CCNT 레지스터를 사용하는 제안한 방법이 getnstimeofday 함수를 사용한 경우 보다 10배 이상 작은 오차 값을 나타내는 것을 확인할 수 있고 이 결과 또한 제안하는 방법이 프로그램 실행 시간을 더욱 정밀하고 신속하게 측정하는 것을 검증한다.

V. 결 론

본 논문에서는 ARM 기반의 임베디드 리눅스 시스템에서 코프로세서의 CCNT 레지스터를 이용하여 상대적인 시간 간격을 정밀하게 측정하는 방법을 제안하였다. 다양한 실험을 통하여 CCNT 레지스터로부터 시간을 측정할 때 측정지연이 평균적으로 40ns 미만으로 관찰되었으며 프로그램 주기 측정 오차 또한 39ns 미만으로 관찰되었으므로 기존 함수로 측정하는 것보다 성능이 10배 이상 우수한 것

을 검증하였다. 또한 CCNT 레지스터는 프로세서의 고속 클럭 사이클에 동기하여 동작하는 카운터이기 때문에 고속 클럭 사이클에 대하여 고해상도의 시간을 측정할 수 있다. ARM 기반 임베디드 프로세서가 시장 점유율 90% 이상을 차지하고 있는 상황을 고려했을 때 본 논문에서 제시한 시간측정 방법은 유용하게 사용될 것으로 판단되며, 특히 분산 시스템의 시간 동기화 방법을 위해 제안된 시간측정 방법을 적용하면 기존 방법보다 시간 오차를 정밀하게 측정하고 이를 기반으로 더욱 정밀한 시간 동기화 성능을 달성 할 수 있을 것이다.

References

[1] D. B. Stewart, "Measuring Execution Time and Real-Time Performance", Embedded Systems Conference, Boston, USA, pp. 341-361, Sep. 2006.

[2] P. E. McKenney, "Real Time' vs. 'Real Fast': How to Choose?", RTLWS 2009, Dresden, Germany, pp. 57-65, Sep. 2009.

[3] L. Abeni, A. Goel, C. Krasic, J. Snow, and J. Walpole, "A measurement-based analysis of the real-time performance of linux", Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, San Jose, CA, USA, pp. 133-142, Sep. 2002.

[4] S. T. Dietrich and D. Walker, "The Evolution of Real-Time Linux", 7th RTL Workshop 2005, Lille University of Science and Technology, Nov. 2005.

[5] A. M. Chirkin and A. S. Z. Belloum, et al., "Execution time estimation for workflow scheduling", Future Generation Computer System, Vol. 75, pp. 376-387, Oct. 2017.

[6] G. A. Elliott, "Real-time Scheduling for GPUS with Applications in Advanced Automotive Systems", PhD thesis, University of North Carolina at Chapel Hill, USA, Jan. 2015.

[7] M. A. Iverson, F. Ozguner, and L. C. Potter, "Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment", Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99), San Juan, Puerto Rico, USA, pp. 99-111, Apr. 1999.

[8] C. Song and W. Yongqing, "Detection and research of RTX timer actual computing workload for ONC system based on RDTSC", The International Journal of Advanced Manufacturing Technology, Vol. 57, No. 1-4, pp. 257-264, Nov. 2011.

[9] C. Walbourn, "Game Timing and Multicore Processors", SDE, Windows Gaming & Graphics, Microsoft Corp, Nov. 2005.

[10] N. Bonelli, S. Giordano, G. Procissi, and R. Secchi, "Brute: A high performance and extensible traffic generator", Proc. of International Symposium on Performance of Telecommunication Systems (SPECTS'05), Philadelphia USA, pp. 222-227, Jul. 2005.

[11] P. Beckman, K. lskra, K. Yoshii, and S. Coghlan, "The Influence of Operating Systems on the Performance of Collective Operations at Extreme Scale", 2006 IEEE International Conference on Cluster Computing, Barcelona, Spain, pp. 1-12, Sep. 2006.

[12] C. J. AN, H. C. Yi, H. W. Kim, S. M. Park, and J. Y. Choi, "Improving Packet Loss Rate of EtherCAT Master Dependent on Hardware Performance", Journal of KIIT, Vol. 13, No 4, pp. 77-83, Apr. 2015.

[13] P. Ferrari, A. Flammini, S. Rinaldi, A. Bondavlli and F. Brancati, "Evaluation of timestamping uncertainty in a software-based IEEE1588 implementation", 2011 IEEE International Instrumentation and Measurement Technology Conference, Binjiang, China, pp. 1-6, May 2011.

[14] S. A. Finney, "Real-time data collection in Linux: A case study", Behavior Research Methods, Instruments & Computers, Vol. 33, No. 2, pp. 167-173, May 2001.

저자소개

권 영 우 (Young-Woo Kwon)

2019년 2월 : 부산대학교
전자공학과(공학사)
2019년 3월 ~ 현재 : 부산대학교
전자전기컴퓨터공학과 석사과정
관심분야 : 임베디드 시스템,
제어시스템

최 준 영 (Joon-Young Choi)

1994년 2월 : 포항공과대학교
전기전자공학과(공학사)
1996년 2월 : 포항공과대학교
전자전기공학과(공학석사)
2002년 : 포항공과대학교
전자전기공학과(공학박사)
2005년 3월 ~ 현재 : 부산대학교
전자공학과 교수
관심분야 : 임베디드 시스템, 제어 시스템