Journal of Institute of Control, Robotics and Systems (2017) 23(1):60-66
https://doi.org/10.5302/J.ICROS.2017.16.0181 ISSN:1976-5622 eISSN:2233-4335

공개 임베디드 하드웨어의 실시간 메커니즘 성능 분석

Performance Evaluation of Real-time Mechanisms on Open Embedded Hardware Platforms

신 욱 철, 최 병 욱*
(Uk Cheol Shin[1] and Byoung Wook Choi[1,*])
[1]Dept. of Electrical and Information Engineering, Seoul National University of Science and Technology

Abstract: Open hardware platforms are gaining wide popularity in various fields of technology for their accessibility and inexpensiveness. Although availability of these platforms is undeniable, the performance of these machines in handling tasks which require execution within critical time constraints is still in question. Moreover, technical documentation and support for open hardware are scarce in comparison to expensive commercially distributed platforms. In this paper, we present a detailed process to implement embedded Linux and the real-time extension of Xenomai to two popular open embedded hardware, BeagleBone Black and i.MX6Q SABRELite. Also, we analyze the response time of real-time mechanisms and its performance for inter-task communication (ITC) between real-time tasks. Response time of real-time mechanisms is an important measure to predict the real-time performance of each platform. The performance evaluation is performed both in kernel space and user space using the metrics of jitter, to estimate the periodicity of the cyclic task, and execution time of three real-time mechanisms including semaphore, message queue, and message pipe. The results of this study would serve as a guideline in developing real-time systems on low cost open embedded hardware platforms using the real-time Linux extension of Xenomai.

Keywords: xenomai, real-time embedded Linux, real-time mechanisms, ITC, open hardware platforms

I. 서론

임베디드 시스템은 마이크로 프로세서를 기반으로 특정한 목적을 수행하는 시스템으로 많은 제품 개발과 연구가 진행되어왔다. 최근에는 사물인터넷(IoT: Internet of Things) 등 빅 데이터 기술, 인공지능의 출현으로 인해 그 기반이 되는 임베디드 시스템에 대한 관심이나 중요도가 더욱 증가하고 있다[1,2].

많은 사용자 유도를 위하여 하드웨어와 소프트웨어를 공개하는 오픈 하드웨어와 오픈 소프트웨어 개념이 최근의 일반적인 경향이 되고 있다. 제품의 쉬운 사용과 함께 개발비의 절감을 통하여 개발자는 본연의 목적에 충실할 수 있도록 도와준다. 이와 같은 목적으로 BeagleBone 프로젝트, Raspberry 프로젝트 및 Freescale 하드웨어 등이 있다[3-5].

최근의 제어장치는 하드웨어 추상화와 함께 스케쥴링 및 부가적 서비스를 활용하기 위하여 운영체제 위에서 구현 된다. 실시간 성능 보장을 위하여 다중 태스크 구현이 필수적이며 실시간 메커니즘이 지원되어야 한다[6,7].

실시간 운영체제는 상용 RTOS (Real-Time Operating System)와 함께 오픈 RTOS가 있다. 최근에는 오픈 실시간 패치인 Xenomai를 이용하여 실시간 시스템 구현에 대한 연구가 많이 진행되고 있다[8-10].

실시간 성능 분석에 관한 논문으로는 다양한 관점에서 진행된 결과가 있으나, 임베디드 시스템의 중요성과 함께 개방 하드웨어의 사용이 증가한다는 측면에서 이에 대한 연구는 부족한 실정이다. 또한 임베디드 하드웨어 플랫폼에 대한 최신의 커널과 최신의 실시간 임베디드 리눅스 패치인 Xenomai 환경에서의 실시간 메커니즘에 대한 성능 분석은 보고된 바가 없다[6,7].

일반적으로 실시간 운영체제의 성능 분석은 인터럽트 지연이나 컨텍스트 스위칭에 대한 지연의 문제를 포함하고 있다[11]. 그러나 본 연구는 일반적인 실시간 성능 분석보다는 오픈 하드웨어를 이용하여 응용 프로그램을 작성할 때 제일 중요한 문제를 다루고자 한다. 즉, 최신의 리눅스와 Xenomai를 이용할 경우에 타스크 간의 통신 방식인 다양한 메커니즘에 있어서의 사용자 모드와 커널 모드의 성능 분석을 수행함으로써 개발자들이 응용 시스템 개발에 중요한 지표를 제시하는 것을 목적으로 한다.

본 논문에서는 최근에 실시간 제어 시스템으로의 사용이 많이 시도되고 있는 BBB (BeagleBone Black), Freescale i.MX6 series의 SABRELite를 이용하여 최근에 가장 일반화된 실시간 임베디드 리눅스인 Xenomai를 이용하여 실시간 시스템을 구현한다. 또 하나의 오픈 하드웨어 프로젝트인 Raspberry 보드는 사용 목적상 멀티미디어 및 게임용으로 사용되고 있으며 Raspberry Pi 2,3에서는 커널 버전 및 Xenomai 버전 등이 상이하여 본 연구에서는 제외하였다[11].

본 논문은 II장에서 하드웨어 특징에 대하여 간단히 설명하며, III장에서는 실시간 임베디드 시스템 구현 과정을 설명한다. IV장에서는 실시간 태스크의 주기성에 관한 실험 결과를 나타내며, V장에서 실시간 시스템 구현에 사용되는 실시간 메커니즘의 시간적 성능 분석 결과를 나타낸다. 이러한

* Corresponding Author
Manuscript received September 25, 2016 / revised December 7, 2016 / accepted December 26, 2016
신욱철, 최병욱: 서울과학기술대학교 전기정보공학과
(shinwc159@hanmail.net/bwchoi@seoultech.ac.kr)
※ 본 연구는 서울과학기술대학교 교내 학술연구비 지원에 의하여 연구되었음.

결과를 이용하여 실시간 제어 장치로 사용에 필요한 결론을 VI장에서 맺도록 한다.

II. 오픈 하드웨어 플랫폼

1. BeagleBone Black

Beagle 보드는 단일 보드 컴퓨터의 일종으로 저전력 오픈 하드웨어이며 TI사와 Digi-key의 협력으로 개발되어 많은 사용자와 함께 응용 예제가 발표 되었다[3].

BBB는 2013년 초 출시되었으며 TI사의 1GHz Sitara AM335x ARM Cortex-A8 프로세서, 4GB의 내장 메모리, 512MB DDR3 램 등을 지원한다. 65개의 디지털 I/O와 7개의 아날로그 입력은 물론 다양한 아날로그 혹은 디지털 장비를 지원한다.

현재 BBB 오픈 하드웨어는 3D 프린터, 로봇분야 등의 다양한 분야에서 사용되고 있다[12].

2. SABRELite

i.MX6 시리즈는 Freescale에서 제작하는 ARM Cortex 아키텍쳐 기반의 싱글, 듀얼, 그리고 쿼드-코어 패밀리를 포함하는 멀티코어 플랫폼이며 저전력 기반의 응용을 위한 전용 제어 오픈 하드웨어 플랫폼이다[5].

SABRELite i.MX6Q는 2011년 초 출시되었으며 i.MX6Q ARM Cortex A9 쿼드-코어 플랫폼, Vivante 시리즈의 GPU를 지원하며 2D, 3D, Vector Graphics 처리가 가능한 강력한 컴퓨팅 성능과 멀티미디어 성능을 제공한다. 그 외에도 1 GB RAM, 1GB Ethernet, 그 외의 주변장치 등을 제공한다.

현재 i.MX 시리즈의 제품은 자동차, 산업 및 제어 장치로 시장 전반에 적용되어 사용되고 있다.

본 연구에서 사용한 BBB 및 SABRELite의 대략적인 사양을 표 1에 정리하였다[3,5,14].

III. 실시간 임베디드 리눅스 제어 장치 구현

Xenomai는 실시간 운영체제가 아닌 실시간 시스템 환경을 제공하는 인터페이스이다. 즉, Xenomai를 사용하기 위해서는 그 기반이 되는 커널을 필요로 한다. 본 논문에서는 리눅스 커널 버전과 Xenomai의 호환성 검토와 포팅 시도를 통해 Xenomai 2.6.x 버전에서 구현 가능한 최신 리눅스 커널을 기반으로 하여 그림 1과 같은 개발환경을 구현하였다. 개발을 진행한 Host-Target 교차 개발 환경은 표 2와 같다.

표 1. 각 보드에 대한 하드웨어 사양.
Table 1.Hardware Specifications of each platforms.

	BeagleBone Black	i.MX6Q SABRELite
제품 공개	2013.4	2011.1
가격	US$49	US$200
SoC	AM3358/9	I.mx6Quad
CPU	32-bit Cortex-A8 + 2xPRU(200MHz)	32-bit quad-core ARM Cortex-A9
주파수(Mhz)	1000	1200
GPU	PowerVR SGX530	Vivante GC2000
이더넷	10/100 Ethernet	10/100/1Gb Ethernet
USB 지원	1 x USB 2.0 1 x Mini USB	2 x USB 2.0 1x OTG
메모리	512MiB DDR3	1GB DDR3
크기	86.4mm x 53.3mm	82mm x 82mm

본 논문에서의 최신의 실시간 임베디드 리눅스를 구현하는 과정에 있어서 두 가지 하드웨어에는 동일한 환경을 구현하였다. SABRELite의 임베디드 리눅스 구현 과정은 본 연구팀의 연구 결과인 [14]의 일부분에서 다뤘고 여기서는 BBB에서의 임베디드 리눅스의 간략한 구현 과정을 다룬다.

리눅스 커널 3.x 버전 이후에서는 리눅스 커널과 하드웨어 간의 의존성을 없애기 위하여 DTB (Device Tree Binary)를 필요로 한다. 이는 하드웨어 초기화를 담당하는 하드웨어 종속적인 부분만을 분리하여 별개의 하드웨어 구성에 대한 데이터 구조 파일인 DTB로 만든 것이다.

이에 따라 커널 부트과정에서 커널 이미지와 DTB 파일을 읽어올 수 있는 부트로더가 필요하다. 이를 지원하는 U-boot 버전으로는 2012년 이후 버전에서만 지원된다.

포팅 과정에서는 microSD card(MMC device)를 통해 파일시스템을 구축하였으며, 이를 위해 그림 2와 같이 microSD 메모리 파티션을 구성하였다.

부팅 과정에서 u-boot의 환경변수를 서술한 부트 스크립트를 이용하여 커널 이미지와 DTB를 읽어오게 하였다. rootfs 파티션에는 부트 스크립트, 커널 이미지, DTB, Xenomai를 설치한 RFS (Root File System)를 위치시킨다.

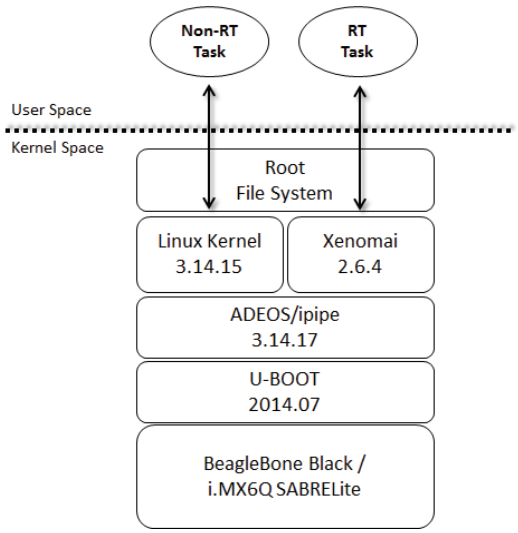
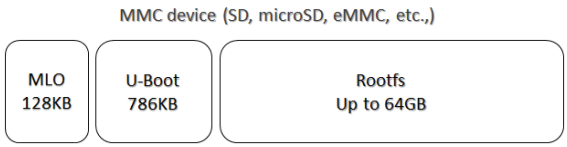


그림 1. 실시간 임베디드 리눅스 구조.
Fig. 1. Real-time embedded Linux architecture.

표 2. Host-Target 교차 개발 환경.
Table 2.Host-Target cross development environment.

항목		규격
Host	Processor	Intel Core i5-4460 @ 3.20GHz
	OS	32-bit Ubuntu 14.04 LTS
	Kernel	Linux 3.16.0-45-generic
Target	Board	BeagleBone Black
		Freescale i.MX6Q SABRELite
	Toolchain	arm-linux-gnueabihf-4.8.3
	Bootloader	U-Boot 2014.07
	Kernel	Linux 3.14.15 ARMv7 Multiplatform
	ADEOS	ipipe-core-3.14.17-arm-2
	Real-time Skin	Xenomai 2.6.4
	RFS	Minimal Ubuntu 14.04.02



Area Name	Format	Size	End	Size	Partition Name
MLO	-	128KB	0	255	mmcblk0boot0
U-BOOT	-	786KB	256	1792	mmcblk0boot1
ROOTFS	EXT4	UP to 64GB	2048	remaining	mmcblk0p1

그림 2. 메모리 디바이스 도표 및 파티션 테이블.
Fig. 2. Memory device diagram and partition table.

수행 과정은 다음과 같다.

U-boot는 Boundary Devices의 Git repository에서 u-boot-imx6 tree에 v2014.07 branch를 다운로드 받은 후 BBB의 프로세서에 맞는 패치를 받아 패치를 실행한다[15,16]. 그런 이후에 해당 보드에 맞는 설정을 해준 후 컴파일을 실행한다.

```
# cd u-boot-imx6
# make am335x_evm_defconfig
# make
```

생성된 MLO와 u-boot이미지를 dd 명령어를 통해 microSD 그림에 나타낸 메모리에 적재한다.

BBB에 사용할 최신 리눅스 커널을 다운로드를 받기 위해 RobertCNelson의 Git repository 에서 armv7-multiplatform tree의 v3.14.x branch를 받은 후 build_kernel 이란 shell script를 실행한다. 다운로드 된 리눅스 커널 3.14.15 디렉터리의 경로는 환경변수로 설정한다[17].

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

그 후 am33x tree에서 BBB 프로세서의 Power Management에 관한 펌웨어를 받아 KERNEL/firmware의 디렉터리로 옮겨준다[18].

Xenomai 플랫폼을 구축하기 위해서는 Xenomai 소스와 ADEOS I-Pipe 패치를 받은 후 Xenomai 소스의 prepare-kernel.sh 스크립트를 실행하여 리눅스 커널을 확장하여야 한다[19].

```
# 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
```

그 후 BBB의 기본적인 설정 파일을 기반으로 추가적인 옵션을 수정한다.

```
# cd $KERNEL
# cp configs/beaglebone .config
# make menuconfig
```

RTOS는 전력 변화에 크게 영향을 받기 때문에 스위치와 같은 Suspend/Resume 동작은 RTOS에서는 허용하지 않는다. 따라서 다음과 같은 설정은 비활성화 시켜야 한다.

- CONFIG_CC_STACKPROTECTOR_NONE
- CONFIG_CPU_FREQ
- CONFIG_KGDB

그리고 최적의 실시간성 보장을 위하여 환경설정은 다음과 같이 설정해주었다.

- CONFIG_SCHED_MC = Y
- CONFIG_PREEMPT = Y

설정 완료 후 커널, 모듈 및 DTB를 컴파일한다.

```
# make zImage dtbs modules
```

RFS는 RobertCNelson의 서버에서 가장 최신의 Minimal Ubuntu의 파일시스템을 사용하며 다운로드를 받은 후 압축을 풀어주고 RFS의 경로를 환경변수로 설정한다[20].

```
# export RFS=$(pwd)/rootfs
```

그 후 커널 모듈을 설치한다.

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

커널 모듈을 설치한 이후 Xenomai 라이브러리를 RFS에 설치해준다. Target에 맞는 최적의 플랫폼 구성을 위하여 ARM Cortex-A8 프로세서에 맞는 최적화 옵션과 FPU (Floating Point Unit)을 사용하기 위한 옵션을 추가하여 환경 설정을 해주었다.

```
# 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
```

본 과정에서는 부팅 과정에서 부트 스크립트를 이용한다. 스크립트의 경로에 맞게 생성한 커널 이미지와 DTB를 위치시킨다[21]. 마지막으로 RFS를 microSD의 rootfs 파티션으로 옮겨준다. 그 후 microSD를 board에 삽입 후 부팅시킨다.

Xenomai가 올바르게 설치됨을 확인하기 위하여 Xenomai가 제공하는 latency 테스트 프로그램을 BBB와 SABRELite에서 실행해 보았다. 그 결과 BBB에서는 1ms의 샘플링 주기에서 지연 시간이 7us 이내로, SABRELite에서는 지연시간이 2us이내로 측정된다. 이것으로 BBB, SABRELite에서 Xenomai가 올바르게 설치되어 실시간 서비스를 제공함을 알 수 있다.

IV. 주기적 태스크의 주기성 연구

1. 실험환경

실시간 시스템의 실시간성에 대한 기본 요건으로는 지정한 주기에 맞춰서 정밀한 태스크의 수행이 이루어져야 한다. 실시간 시스템은 다양한 주기로 구현되며 정밀 제어 프로그램의 경우에는 1ms의 주기부터, 일반적인 응용프로그램의 경우에는 10ms를 기본으로 하여 정수 배의 주기로 실시간 태스크를 구현한다.

무 부하(idle) 상태와 최고 부하 상태(full load)를 이용하여 통계적 성능 분석을 통하여 성능 분석을 수행하고 있으나, 본 논문의 경우 실험 대상 타스크는 최고 높은 우선 순위를 가지고 있어 영향이 적고 지면의 제한으로 기술하지 않았다.

본 실험 환경에서는 각 하드웨어의 유저/커널 공간에서 1ms, 10ms의 주기의 태스크를 작성하여 실험을 진행한다. 커널 공간의 프로그램은 모듈 형태로 구현하여 실험하였고, 성능 분석에 필요 없는 데이터 처리 등은 일체 제외하였다. 또한, 스케줄링으로 인한 지연을 방지하기 위해 각각의 태스크에 Xenomai의 가장 높은 우선순위인 99를 부여하여 실험하였다. 실험의 정확성을 위해 각각의 실험은 5000번의 반복을

표 3. 다양한 주기의 태스크에서의 실시간 성능 측정 결과 [ms].
Table 3. Real-Time Performance Measurement Results in Different Periodic Tasks [ms].

		1ms				10ms			
		User		Kernel		User		Kernel	
		T_{period}	T_{jitter}	T_{period}	T_{jitter}	T_{period}	T_{jitter}	T_{period}	T_{jitter}
Beagle Bone Black	Avg.	1.0000	0.0033	1.0000	0.0014	10.000	0.0015	10.000	0.0006
	Max	1.0324	0.1068	1.0132	0.0133	10.022	0.0061	10.014	0.0018
	Min	0.8931	0	0.9874	0	9.9394	0	9.9822	0
	St.D	0.0012	0.0011	0.0005	0.0005	0.0023	0.0018	0.0012	0.0011
SABRE Lite	Avg.	1.0000	0.0003	1.0000	0.0001	10.000	0.0011	10.000	0.0002
	Max	1.0087	0.0087	1.0061	0.0061	10.020	0.0020	10.012	0.0012
	Min	0.9937	0	0.9956	0	9.9910	0	9.9956	0
	St.D	0.0009	0.0009	0.0004	0.0004	0.0019	0.0016	0.0005	0.0005

통해 충분한 샘플을 획득하게 하였으며 실시간 태스크에서의 수행시간 측정은 Xenomai의 RT-Time API를 이용하여 시간을 측정하였다.

그림 3은 주기적 태스크의 실험 구조를 의사코드로 표현하였다. 태스크의 실행 시 특정 주기가 설정되고 주기적으로 실행되게 한다.

그 후 측정된 시간을 기반으로 각 주기에서의 태스크의 주기성(periodicity)과 지연시간(jitter) 그리고 그에 대한 최고(max), 최소(min), 평균(average), 표준정규분포(standard normal distribution)를 기준으로 실시간성을 분석한다.

주기적인 태스크의 실제주기 T_{period}와 지터 T_{jitter}의 관계는 다음과 같이 정의 된다.

$$T_{jitter} = \left| T_{cycle} - T_{period} \right| \quad (1)$$

여기서 T_{cycle}은 태스크 생성할 때 설정한 주기를 의미한다.

2. 실험결과

실험 결과는 표 3과 같고 각 주기마다 5000번의 실험을 반복하였으며 결과는 ms로 나타내었다.

실험을 진행한 1ms 주기의 태스크에서 T_{period}는 BBB, SABRELite 모두 예상된 Cycle 값에 근접한다. T_{period}의 표준편차가 BBB의 경우 유저 공간에서 0.0012, 커널 공간에서 0.0005의 편차 값을 가지며 SABRELite에서는 유저 공간에서 0.0009, 커널 공간에서 0.0004의 편차 값을 가진다.

T_{jitter}는 BBB에서는 평균적으로 유저 공간 3.3us, 커널 공간 1.4us 이며 SABRELite에서는 유저 공간 0.3us, 커널 공간 0.1us로 나타나며 표준편차도 위와 같이 두 개의 하드웨어 모두 커널 공간에서 적은 값을 가지며 SABRELite가 BBB보다 더 적은 편차를 보인다. 이러한 전체적인 경향은 10ms 주기의 태스크에서도 동일하게 나타난다.

종합적으로 주어진 주기에 대해 두 개의 하드웨어가 모두 유저 공간보다 커널 공간에서 더 정밀하고 안정적인 정주기성을 발휘하며 1ms의 짧은 주기에서 더 정밀한 값을 가진다. 상대적으로 SABRELite가 BBB보다 높은 성능을 가지지만 두 개의 하드웨어 모두 주어진 주기에 대한 정밀한 태스크의 수행이 이루어짐으로 실시간 제어기로서의 사용의 기본요건을 충족함을 알 수 있다.

```
void Periodic_Task()
{
   Set period and make periodic
   ...
   While(count is not 5000)
   {
      Start time measurement
      Wait period
      Stop time measurement
      Increase count
      Calculate time and Save in buffer
   }
}
```

그림 3. Periodic_Task 의사 코드.
Fig. 3. Pseudocode of periodic_Task.

V. 실시간 메커니즘 성능 분석

1. 실험환경

Xenomai는 실시간 시스템 환경을 제공하며 실시간 시스템은 다중 태스크 작업 환경을 지원하며 태스크 간의 통신(ITC: Inter Task Communication)을 통해 태스크 간의 동기화, 자원관리, 활동을 조정한다. Xenomai에서의 ITC 메커니즘으로는 세마포어(semaphore), 메시지 큐(message queues), 메시지 파이프(message pipes)가 존재하며 이들은 유저, 커널 공간에 상관 없이 모두 동일한 구조를 가진다.

본 장에서는 ITC 메커니즘에 대한 성능 분석으로 각 메커니즘의 수행 시간에 대한 시간적 성능을 분석하고 그에 대한 결과 분석을 통해 각 하드웨어의 실시간 제어기로서의 성능을 평가해보고자 한다. 실험을 진행한 환경은 IV장과 동일하며 각각의 태스크들은 동일한 우선순위를 가진다.

2. 세마포어(semaphore)

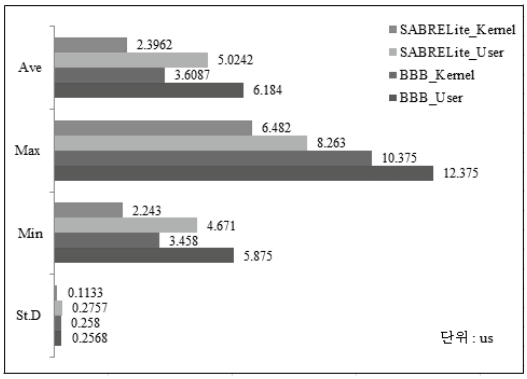
세마포어는 멀티 태스크 응용프로그램의 활동을 조정하기 위한 도구이다. Xenomai에서 세마포어는 빠른 태스크 간 통신을 제공하며 상호배제와 태스크 동기화를 해결하기 위한 주요 수단으로 이용된다.

그림 4는 세마포어의 시간적 성능을 분석하기 위한 의사코드이다. Taks1이 세마포어를 양도한 시점부터 Task2가 세마

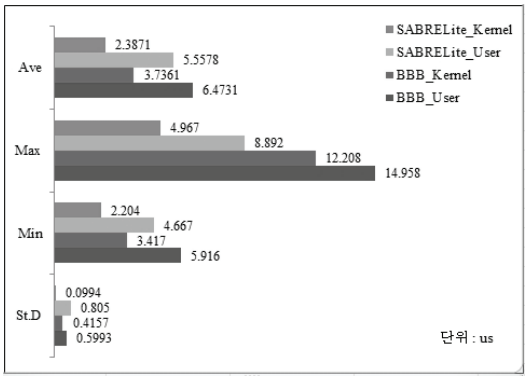
```
void Task1()
{
   Set   period and make periodic
   ...
   While(count is not 5000)
   {
      Release semaphore
      Start time measurement
      Increase count
   }
}
```

```
void Task2()
{
   ...
   While(count is not 5000)
   {
      Get semaphore
      Stop time measurement
      Calculate time and Save in buffer
   }
}
```

그림 4. 세마포어 획득 시간 측정을 위한 의사 코드.
Fig. 4. Pseudo-code for Measuring time to acquire semaphore.



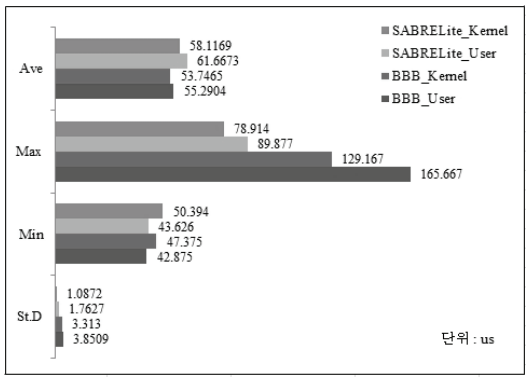
(a) 1ms.



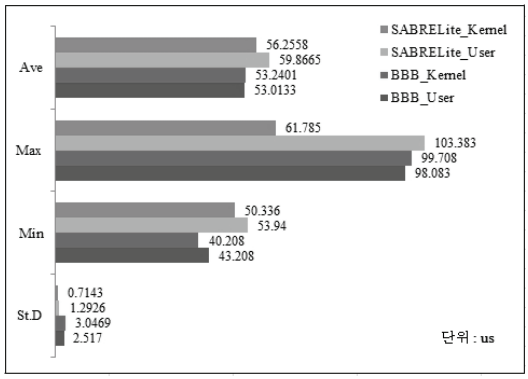
(b) 10ms.

그림 5. 다양한 주기에서의 세마포어 획득 결과.

Fig. 5. Semaphore Acquisition Result in Different period.



(a) 1ms.



(b) 10ms.

그림 7. 다양한 주기에서의 메시지 파이프 실험 결과.

Fig. 7. Message Pipe Experiment Results in Different period.

포어를 획득한 시점까지 걸린 시간을 측정하였다.

실험에 대한 결과는 그림 5와 같다. 세마포어를 양도하고 다른 태스크가 이를 획득하는데 소요되는 시간이 1ms주기의 태스크에서 BBB 유저 공간 평균 6,184us, 커널 공간 평균 3.6087us가 걸리며 SABRELite 유저 공간 평균 5.0242us, 커널 공간 평균 2.3962us의 시간이 소요된다. 두 개의 하드웨어 모두 유저 공간보다 커널 공간에서 적은 시간을 소요하는 것을 알 수 있고 SABRELite가 BBB보다 빠른 수행시간을 가짐을 알 수 있다. 이러한 경향은 10ms 주기의 태스크에서도 동일하게 나타난다.

실험을 수행한 2가지 주기의 표준편차에서 커널 공간 1ms의 주기에서 더 밀집된 분포도를 보이고 2가지 주기에서 모두 BBB보다 SABRELite가 작은 편차를 보임으로 미루어 보아 SABRELite가 BBB보다 안정적인 시간적 응답 특성을 발휘하고 높은 신뢰성을 가진다는 것을 알 수 있다.

```
void Task1()
{
  Set period and make periodic
    ...
  While(count is not 5000)
  {
   Put data to Pipe
   Start time measurement
   Get   data from Pipe
   Stop time measurement
   Increase count
   Calculate time and Save in buffer
  }
}
```

```
void Task2()
{
    ...
  While(count is not 5000)
  {
   Get data from Pipe
   Put data to Pipe
  }
}
```

그림 6. Message Pipe 실험을 위한 의사 코드.

Fig. 6. Pseudocode for testing Message Pipe.

3. 메시지 파이프(message pipe)

메시지 파이프는 Xenomai 태스크와 가상장치(/dev/rtpX)의 파일 I/O동작을 이용한 표준 리눅스 프로세서의 태스크와의 양방향 통신 채널이다. 즉, 실시간 태스크와 비실시간 태스크 사이의 통신수단이다.

그림 6은 커널 공간/유저 공간의 실시간 Task1이 Pipe를 통해 사용자 공간의 리눅스 태스크로 데이터를 전달하고 다시 리눅스 태스크에서 실시간 태스크로 데이터를 전달하는 구조를 의사코드로 나타내었다. 실험은 위의 과정을 수행하는데 걸린 시간을 측정하였다.

실험에 대한 결과는 그림 7과 같다. 실시간 태스크와 리눅스 태스크 간의 메시지 송수신 시간이 1ms주기의 태스크에서 BBB 유저 공간 평균 53.0133us, 커널 공간 평균 53.2401us가 걸리며 SABRELite 유저 공간 평균 59.8665us, 커널 공간 평균 56.2558us의 시간이 소요된다. BBB가 SABRELite보다 빠른 수행 시간을 가지지만 BBB의 커널 공간에서의 평균 수행시간과 편차가 유저 공간에 비해 높게 나오고 이러한 경향은 10ms의 주기에서도 동일하다.

두 가지 주기에서의 실험에서 BBB는 메시지 파이프의 수행이 불안정한 반면 SABRELite는 안정적인 성능을 보인다. 이러한 메시지 파이프의 불안정성의 원인으로는 리눅스 태스크에서는 비 실시간 메커니즘인 리눅스 API를 사용하기 때문에 실시간성을 보장할 수 없어 이러한 현상이 발생하는 것으로 보인다.

4. 메시지 큐(message queue)

메시지 큐는 Xenomai에 의해 관리되는 큐를 통하여 실시

```
void Task1()
{
  Set period and make periodic
  ...
  While(count is not 5000)
  {
   Send message to queue
   Start time measurement
   Increase count
  }
}
```

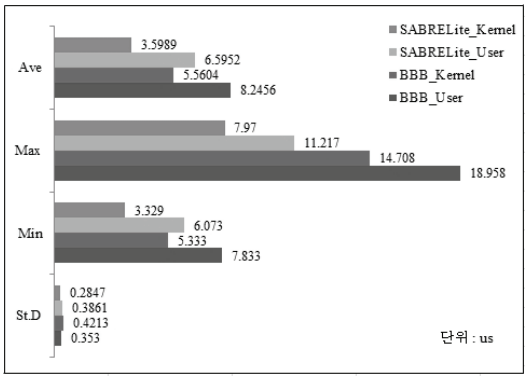
```
void Task2()
{
  ...
  While(count is not 5000)
  {
   Get message from queue
   Stop time measurement
   Calculate time and Save in buffer
  }
}
```

그림 8. Message queue 실험을 위한 의사 코드.
Fig. 8. Pseudocode for testing Message queue.

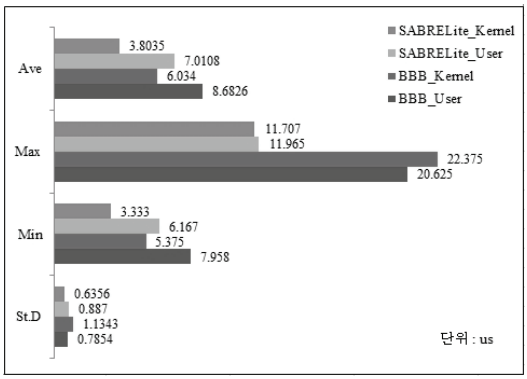
간 태스크 사이에서 데이터를 교환, 전달하는 통신수단이다. 메시지 큐는 메시지의 개수, 가변 길이, 가변 데이터 타입을 허락한다. 큐는 FIFO (First in, First Out) 방식과 우선순위 기반의 방식이 존재한다.

데이터 전송에 사용되는 메시지 큐의 경우 그림 8과 같은 구조를 이용하여 데이터 전송에 사용되는 실시간 메커니즘의 성능을 분석하였다. 메시지 큐는 N:1 구조의 통신이 가능하지만 여기서는 두 개의 태스크만을 이용하여 Task1이 Task2에 메시지를 전달하는데 소요된 시간을 측정하였다.

실험에 대한 결과는 그림 9와 같다. 1ms 주기의 태스크에서 메시지 큐를 통한 메시지 송수신 소요 시간이 BBB 유저 공간 평균 8.2456us, 커널 공간 평균 5.5604us, SABRELite 유저 공간 평균 6.5952us, 커널 공간 평균 3.5989us가 소요된다. 이전의 메커니즘 분석 결과와 동일하게 짧은 주기, 커널 공간에서 적은 시간이 소요됨을 알 수 있고 10ms의 주기에서도 같은 경향을 보인다.



(a) 1ms.



(b) 10ms.

그림 9. 다양한 주기에서의 메시지 큐 실험 결과.
Fig. 9. Message queue Experiment Results in Different period.

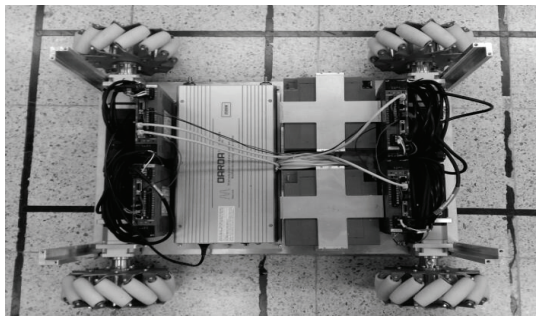


그림 10. 이동로봇 플랫폼.
Fig. 10. Mobile Robot Platforms.

그림 11. 다축 모션 제어.
Fig. 11. Multi-Axis Motion Control.

실험을 수행한 모든 결과에서 SABRELite가 BBB보다 더 적은 시간을 소요함을 알 수 있고 표준편차로 미루어보아 SABRELite가 더 밀집된 분포도를 이루고 있어 안정적인 시간적 응답 특성을 발휘함을 알 수 있다. 다만 BBB의 경우 커널 공간이 유저 공간에 비해 평균 소요 시간은 짧지만 분포도는 더 높은 값을 가짐으로 보아 약간의 불 안정성을 가지고 있음을 알 수 있다.

5. 응용 예

그림 10과 그림 11은 본 연구 결과를 바탕으로 저가의 공개된 하드웨어를 SABRELite, BBB 보드를 이용하여 구성 중인 실시간 응용 시스템이다.

그림 10은 실시간 제어기와 서보 드라이버간의 통신 프로토콜로 EtherCAT을 사용한 전방향 이동 로봇이며 레이저 센서를 이용한 주행 연구[22]와 이동로봇의 물리적 제한을 고려한 궤적 생성 방법에 대한 연구[23]을 기반으로 자율 주행에 대한 연구를 진행 중이다. 그림 11은 EtherCAT 프로토콜 기반으로 다축 모터에 대한 모션 제어를 진행 중이다[24]. 이러한 응용 예에서의 실시간 제어를 위한 프로그램에서 다양한 실시간 메커니즘이 필수 불가결하다. 또한 이러한 응용 예 이외에도 실시간 시스템에서의 실시간 메커니즘은 무기 제어, 발전소 제어, 철도 제어, 산업체 제조 공정 제어, 로봇 제어 등의 수많은 응용 시스템 구성이 가능할 것이다.

VI. 결론

본 논문에서는 선행 연구 결과를 바탕으로 현재 다양한 분야에서 활발히 사용되고 있는 2개의 임베디드 하드웨어를 대상으로 하여 실시간 제어기로서의 실시간 성능을 평가하기 위해서 각 하드웨어에 Xenomai를 구현하고 실시간 성능의 분석을 위해 태스크의 주기에 따른 주기성, 실시간 메커

니즘의 수행속도에 대한 시간적 분석하였다. 이를 통해 다양한 분야에서의 2가지 하드웨어가 실시간 제어기로서 요구되는 실시간성을 만족함을 검증하였다.

실험 결과, i.MX 시리즈의 SABRELite는 매우 안정적이고 우수한 실시간 성능을 발휘함을 알 수 있었다. 그에 따라 산업, 의료, 방산 등의 고 정밀제어가 요구되는 제품의 실시간 제어기로서도 충분히 활용 가능하다. 또한 BeagleBone Black 또한 SABRELite보다는 실시간 성능에 있어서 아쉬운 부분이 있지만 가격대비 우수한 성능을 발휘함으로 교육용, 산업, 상업 등의 적정한 수준의 실시간 성능이 요구되는 제품의 실시간 제어기로서 활용 가능하다.

현재 본 연구팀에서 SABRELite, BeagleBone Black 등의 하드웨어에 공개된 실시간 이더넷 프로토콜인 IgH EtherCAT을 구현하고 상위 제어기로서의 성능 분석에 관한 연구를 진행하고 있으며 향후 이에 대한 자세한 성능 평가에 관한 내용을 서술하고자 한다.

REFERENCES

[1] K. Delic and J. Riley, "Current and future trends in AI," *Information, Communication and Automation Technologies (ICAT), 2013 XXIV International Symposium on*. IEEE, 2013.

[2] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3-9, Feb. 2014.

[3] Beagle board Project, *https://beagleboard.org/*

[4] Raspberry Pi, *https://www.raspberrypi.org/*

[5] Freescale Semiconductor, *http://opensource.freescale.com/*

[6] J. H. Koh and B. W. Choi, "Real-time performance of real-time mechanisms for RTAI and Xenomai in various running conditions," *Internal Journal of Control and Automation*, vol. 6, no. 2, pp. 139-151, Feb. 2013.

[7] J. H. Koh and B. W. Choi, "On benchmarking of real-time mechanisms in various periodic tasks for real-time embedded linux," *Journal of Korea Robotics Society*, vol. 7, no. 4, pp. 292-298, Dec. 2012.

[8] W. S. Liu, *Real-Time Systems*, Prentice Hall, 2000.

[9] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application," *IEEE Transactions on Nuclear Science*, vol. 55, no. 1, pp. 435-439, Feb. 2008.

[10] M. D. Marieska, A. I. Kistijantoro, and M. Subair, "Analysis and benchmarking performance of real time patch Linux and Xenomai in serving a real time application," *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*. IEEE, 2011.

[11] T. Straumann, "Open source real time operating system overview," *International Conference on Accelerator & Large Experimental Physics Control Systems*, 2001.

[12] Xenomai Support Hardware, *https://xenomai.org/embedded-hardware/*

[13] I. Siradjuddin, S. P. Tundung, A. S. Indah, and S. Adhisuwignjo, "A real-time model based visual servoing application for a differential drive mobile robot using Beaglebone Black embedded system," *2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 2015.

[14] boundarydevices, *https://boundarydevices.com/product/*

[15] Boundary Devices, *github.com/boundarydevices/u-boot-imx6*

[16] u-boot patch, *https://rcn-ee.com/repos/git/u-boot-patches/*

[17] ARM Multiplatform, *github.com/RobertCNelson/*

[18] Am33x firmware, *http://arago-project.org/git/projects*

[19] Xenomai Download Server, *xenomai.org/downloads/*

[20] Minimal Ubuntu Filesystem, *rcn-ee.com/rootfs/eewiki/minfs/*

[21] Boot script, *https://eewiki.net/display/linuxonarm/BeagleBone+Black*

[22] J. K. Park and T. H. Park, "Autonomous navigation system of mobile robot using laser scanner for corridor environment," *Journal of Institute of Control, Robotics and Systems*, vol. 21, no. 11, pp. 1044-1049, Nov. 2015.

[23] G. J. Yang and B. W. Choi, "Maximum velocity trajectory planning for mobile robots considering wheel velocity limit," *Journal of Institute of Control, Robotics and Systems*, vol. 21, no. 5, pp. 471-476, May 2015.

[24] R. Delgado, C. H. Hong, W. C. Shin, and B. W. Choi, "Implementation and performance analysis of an EtherCAT master on the latest real-time embedded linux," *International Journal of Applied Engineering Research*, vol. 10, no. 24, pp. 44603-44609, 2015.

신 욱 철

2011년~현재 서울과학기술대학교 전기정보공학과 재학 중. 관심분야는 실시간 시스템 설계, 임베디드 리눅스, 지능형 로봇 소프트웨어.

최 병 욱

1988년~1992년 한국과학기술원 전기 및 전자공학과 석사 및 박사 졸업. 1988년~2000년 LG산전 중앙연구소 책임연구원. 2000년~2005년 선문대학교 제어계측공학과 부교수. 2003년~2005년 임베디드웹 대표이사. 2007년~2008년 Nanyang Technological University, Senior Fellow. 2005년~현재 서울과학기술대학교 전기정보공학과 교수. 관심분야는 실시간 시스템 설계, 임베디드 시스템, 임베디드 리눅스, 지능형 로봇 소프트웨어.