논문 2017-12-01

산업용 로봇 제어를 위한 Preempt-RT 기반 멀티코어 모션 제어기의 구현 및 성능 평가

(Implementation and Performance Evaluation of Preempt-RT Based Multi-core Motion Controller for Industrial Robot)

김 익 환, 안 효 성, 김 태 현*
(Ikhwan Kim, Hyosung Ahn, Taehyoun Kim)

Abstract : Recently, with the ever-increasing complexity of industrial robot systems, it has been greatly attention to adopt a multi-core based motion controller with high cost-performance ratio. In this paper, we propose a software architecture that aims to utilize the computing power of multi-core processors. The key concept of our architecture is to use shared memory for the interplay between threads running on separate processor cores. And then, we have integrated our proposed architecture with an industrial standard compliant IDE for automatic code generation of motion runtime. For the performance evaluation, we constructed a test-bed consisting of a motion controller with Preempt-RT Linux based dual-core industrial PC and a 3-axis industrial robot platform. The experimental results show that the actuation time difference between axes is 10 ns in average and bounded up to 689 ns under 1000 μs control period, which can come up with real-time performance for industrial robot.

Keywords : Real-time performance, Preempt-RT, Multi-core motion controller, Network-based industrial control system, Open-source software

Ⅰ. 서 론

최근 제조업과 IT 기술의 융합에 대한 관심이 날로 커짐에 따라, 이를 현실화하기 위한 노력들이 미국, 유럽연합과 같은 선진국들을 중심으로 이루어지고 있다. IT 융합을 통한 제조업 경쟁력 향상의 핵심 키워드는 IIoT (Industrial IoT)와 사이버 물리 시스템 (CPS: Cyber Physical System) 개념에 근거한 전체 생산 시스템의 수직/수평적인 통합이다 [1, 2]. 이러한 흐름의 대표적인 예로 스마트 팩토리의 핵심 요소인 모션 제어시스템의 구현 방식 변화를 들 수 있다.

모션 제어시스템의 핵심 컴포넌트인 모션 제어기 구현은 시스템의 기능 확장성과 소프트웨어 개발의 유연성 확보를 위해 기존의 전용기 혹은 하드웨어 PLC (Programmable Logic Controller) 형태에서 범용 운영체제, 범용 하드웨어 조합으로 구성된 Soft PLC 형태의 구현방식으로 급격한 전환이 이루어지고 있다 [3-6]. 또한 제어기에 생산 관리 시스템 (MES: Manufacturing Execution System)과 전사적 자원관리 시스템 (ERP: Enterprise Resource Planning)과의 연동, 그리고 모바일 장치를 활용한 제어 및 모니터링 등과 같이 다양한 응용계층 서비스가 요구됨 [1, 2]에 따라 제어기에 탑재되는 소프트웨어 복잡도가 점차 높아지고 있다. 기존에 널리 사용되고 있는 싱글코어 기반의 모션 제어기로는 점점 높아지는 응용 복잡도를 수용하면서 동시에 산업체 요구 수준인 1000 μs 이하의 실시간 제어주기를 보장하는 데 한계가 있다. 따라서 최근에 높은 가격 대 성능비를 가진 멀티코어 프로세서를 활용하는 제어기에 대한 관심이 높아지고 있으며, 이를 실제로 적용하고자 하는 노력들

*Corresponding Author (thkim@uos.ac.kr)
Received: Sep. 26 2016, Revised: Nov. 7 2016, Accepted: Nov. 15 2016.
I. Kim, H. Ahn, T. Kim: University of Seoul
※ 본 논문은 2015 년도 정부 (교육부)의 재원으로 한국 연구재단의 지원을 받아 수행된 기초연구사업임 (No. NRF-2015R1D1A1A09057762)

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2017.12.1.1

[7-9]이 활발히 이루어지고 있는 상황이다.

본 논문에서는 멀티코어 환경을 고려한 모션 제어기 소프트웨어 아키텍처와 이 아키텍처에 근거한 모션 제어 소프트웨어 런타임의 자동 생성을 위한 통합 개발환경의 기능 확장을 제안한다. 제안하는 아키텍처는 I/O 와 Computation 작업이 혼합된 단일쓰레드 형태의 모션 응용을 각각 I/O 쓰레드와 Computation 쓰레드로 구분하여 각각 별도의 프로세서 코어에 할당하고, 쓰레드간 통신을 위해 공유 메모리를 사용하도록 하였다. 공유 메모리에는 CiA (CAN in Automation) 402 서보 드라이브 응용 프로파일 [10]에 정의된 드라이브의 동작 모드들을 포함하는 필수 데이터 집합이 매핑된다. 제안한 아키텍처의 동작을 검증하기 위해 산업용 개방형 표준을 준수하는 오픈소스 통합 개발환경인 Beremiz [11]의 기능을 확장하여 멀티코어 기반 제어기 소프트웨어를 개발하고, Preempt-RT 리눅스 기반 듀얼코어 제어기와 EtherCAT 3축 산업용 로봇으로 구성된 테스트베드에서 기본 성능 테스트를 진행하였다. 또한, 동기화 성능을 평가하고자 각 축별 동작 시점의 편차를 측정한 결과 1000 μs 통신 제어주기에서 평균 10 ns, 최대 689 ns 정도의 좋은 동기화 성능을 보임을 확인하였다.

본 논문의 구성은 다음과 같다. Ⅱ장은 연구 배경을 소개하고, Ⅲ장에서는 제안하는 소프트웨어 아키텍처의 설계 내용과 오픈소스 통합 개발환경에 통합한 결과를 보인다. Ⅳ장에서는 구현된 제어기의 성능 평가 결과를 보이고 마지막으로 Ⅴ장에서 결론과 향후 연구를 제시한다.

Ⅱ. 연구 배경

모션 제어시스템은 모션 제어기와 모터 드라이브, 액츄에이터, 그리고 I/O 모듈들로 구성되며, 시스템 운용에 필요한 티칭 펜던트, HMI (Human Machine Interface) 등과 같은 하드웨어 장치들이 추가될 수 있다. 최근에는 각 컴포넌트간 데이터 교환을 위한 산업용 네트워크로 실시간성을 보장하면서도 높은 대역폭을 제공하는 실시간 이더넷 기반 기술들 [12]이 주로 사용되고 있다. 그 중에서도 EtherCAT의 경우, Ethernet 기술과의 호환성 및 대용량 데이터 전송에 적합한 대역폭을 제공하면서 각 장치별 데이터 교환에 소요되는 시간이 결정적인 특징으로 인해 주목받고 있다. 한편, 모션 제어기에 탑재되는 프로그램 개발 및 유지보수의 효율성을 높이기 위해 IEC 61131-3, PLCopen Motion Control 등과 같은 개방형 표준을 준수하는 통합 개발환경의 사용이 필수적으로 간주되며 [11], 최근에는 기능 추가 및 커스터마이징이 용이한 오픈소스 기반 솔루션에 대한 관심과 연구들 [3, 4, 11]이 지속적으로 이루어지고 있다.

IEC 61131-3 기반 모션 응용은 기본적으로 다수의 주기 태스크들로 구성된다. 이들 태스크들은 모터 드라이브와의 통신을 담당하는 태스크와 위치, 속도, 토크 등과 같은 상태 값을 토대로 모션 궤적을 생성해내는 모션 생성 태스크, 그리고 HMI 및 장치의 상태 모니터링 감시 등을 위한 태스크들 등과 같이 주로 I/O 및 계산 작업을 수행한다. 태스크의 구현은 IEC 61131-3에서 정의한 언어들을 사용해 이루어지며, 다양한 플랫폼으로의 이식성을 제공하기 위해 C 언어와 같은 상위 수준 언어로 1차 변환된 후 최종적으로 타겟 의존적인 최종 런타임 코드로 빌드된다. 하지만 IEC 61131-3 표준에서는 상위 수준 언어로 작성된 코드 수준에서의 실제 구현과 관련된 세부 사항은 명시하고 있지 않다.

일반적으로 가장 간단한 형태의 모션 응용 구현 방식은 다수의 태스크들을 동일 제어주기를 갖는 단일 쓰레드 형태로 구현하는 것으로, 싱글코어 기반에서 동작하는 낮은 복잡도를 갖는 모션 응용 구현에 사용된다. 하지만 이러한 구현방식은 기능 확장성과 효율적인 자원 사용면에서 한계를 가진다. 한편, 태스크들을 각각의 쓰레드로 매핑하고 스케쥴링되도록 하는 멀티쓰레드 구현방식의 경우에는 병렬성을 활용한 효율적인 프로세서 자원 사용을 목표로 한다.

그러나 각 쓰레드 간의 데이터 의존성 및 데이터 일관성을 보장하기 위한 동기화 기법의 사용으로 인한 상대적으로 높은 구현 복잡도와 이로 인한 실시간 성능저하 가능성이 존재할 수 있으며, 싱글코어 기반의 수행환경에서는 이러한 특징이 두드러질 수 있다. 따라서 모션 응용에 대한 실시간 요구사항을 만족할 수 있는 멀티코어 아키텍처를 활용한 멀티 쓰레드 기반의 모션 응용 구현에 대한 연구가 필요하며, 본 연구는 멀티코어를 활용하여 종단간 지연시간을 줄이는 것을 목표로 하는 기존 선행 연구들 [7-9]과 달리 높아지는 응용 복잡도에 대응하여 표준에 근거한 구현을 통한 높은 기능 확장성과 실시간성이 동시에 보장될 수 있는 소프트웨어 아키텍처를 제안하고, 이를 반영한 통합 개발환경의 확장 구현 결과를 제시한다.

Ⅲ. 멀티코어를 고려한 소프트웨어 아키텍처의 설계 및 구현

1. 제안하는 소프트웨어 아키텍처

본 연구에서는 IEC 61131-3 표준과 EtherCAT 통신을 지원하는 통합 개발환경인 Beremiz를 활용한다. Beremiz는 PLC 프로그램 작성을 위한 에디터와 통신 프로파일 외에도 C 또는 Python 언어로 작성된 외부 함수들과의 인터페이스를 위한 플러그인 형태의 프로그래밍 Extension 들을 제공하여 기능 확장이 유연한 장점이 있다.

Beremiz를 이용해 자동 생성하는 모션 런타임 코드는 단일 쓰레드 구현을 기본으로 채택하고 있다. 그림 1은 리눅스를 타겟 플랫폼으로 하는 EtherCAT 기반 모션 응용 런타임의 구조를 나타낸다. 모션 런타임은 모터 드라이브에서 전달된 위치, 속도, 토크 값 등을 바탕으로 새로운 모션을 생성하는 모듈 (이하 M.T.G.: Motion Trajectory Generator)과 모터 드라이브의 상태전이를 위한 제어변수들을 다루는 모듈 (이하 D.S.M.: Drive State Machine), 그리고 PLC 프로그래밍 변수들과 모터 드라이브와의 통신 변수들과의 인터페이스를 제공하는 I/O 변수 매핑 모듈 (이하 I/O V.M.: I/O Variable Mapping)로 구성된다. 이들 모듈들은 일반적인 PLC 제어 시퀀스를 따라 정해진 제어주기 내에서 모터 드라이브로부터 송신한 데이터에 대한 값을 수신하여 새로운 모션 궤적에 대한 계산을 수행하고, 이에 대한 EtherCAT 프레임을 생성하여 모터 드라이브들에게 전송하는 과정을 반복한다 [4]. 모션 런타임 쓰레드의 제어주기는 I/O V.M.과 동일하게 설정된다.

EtherCAT 서보 드라이브의 제어명령은 주기적 실시간 데이터인 PDO (Process Data Object) 형태로 매핑되며 모터 드라이브 기준으로 송수신하는 PDO의 방향에 따라 RxPDO, TxPDO로 구분된다. RxPDO 영역은 모터 드라이브의 상태머신 제어 및 모터 드라이브 구동에 필요한 변수 값들이며, TxPDO 영역은 다음 모션의 궤적을 계산하기 위해 모터 드라이브가 회신한 변수 값들이다. 일반적으로 드라이브의 동작 모드 (위치, 속도, 토크)에 따라 CiA 402 프로파일에 정의된 각기 다른 PDO 집합을 사용하도록 되어있다.

본 논문에서 제안하는 멀티코어 지원을 위한 확장 구조는 그림 2와 같다. 기존 모션 런타임 구현에서 모터 드라이브들과의 통신을 수행하는 I/O V.M.과

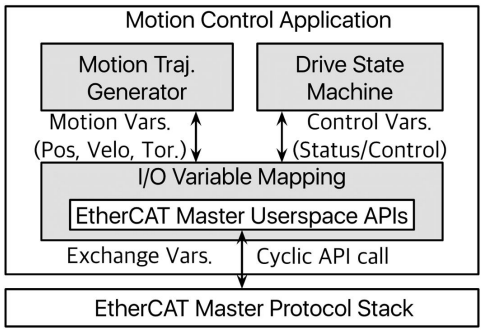


그림 1. 단일 쓰레드 기반의 모션 응용 구조
Fig. 1 Structure of single-thread based motion control application

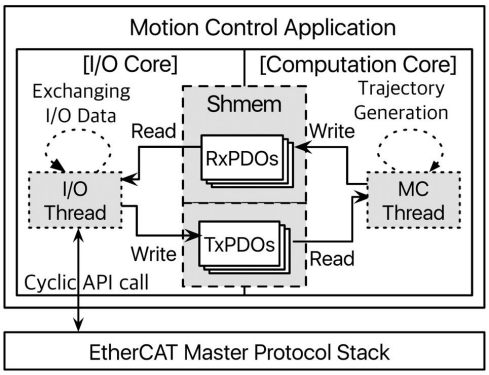


그림 2. 제안하는 소프트웨어 아키텍처
Fig. 2 Proposed software architecture

모션 생성을 위한 계산 및 드라이브 상태머신 제어를 수행하는 모듈들 (M.T.G., D.S.M.)을 분리하여 각각 I/O 쓰레드와 MC (Motion Computation) 쓰레드로 분할, 매핑한다. 그리고 쓰레드 간에 공유되어야 하는 변수들은 프로세스간 통신 메커니즘 (IPC) 중의 하나인 공유 메모리를 사용하여 매핑하되, RxPDO, TxPDO 영역으로 구분하고, 각 영역에 모터 드라이브 별 PDO 변수들이 매핑된다. 앞서 언급한 대로 PDO 영역은 드라이브 상태머신 관련 변수 및 모터 드라이브의 모션 생성을 위한 변수들로 구분되며, EtherCAT 모터 드라이브의 기본 동작 모드들을 포괄하는 코어간 연동을 위한 최소한의 필수 PDO 집합은 표 1과 같다. 마지막으로 쓰레드들의 특성을 토대로 프로세서를 각각 I/O, Computation 코어로 구분, 수행환경을 분리하여 각 쓰레드 간 수행 간섭을 최소화 할 수 있도록 한다. 이와 같이 쓰레드의 수행환경을 분리할 경우 싱글

표 1. 코어간 연동을 위한 PDO 변수 집합
Table 1. PDO variables for the interplay between cores

Category	Name	Index
RxPDO	ControlWord	0x6040
	TargetPosition	0x607A
	TargetVelocity	0x60FF
	TargetTorque	0x6071
	DigitalOutputs	0x60FE
	ModeofOperation	0x6060
TxPDO	StatusWord	0x6041
	ActualPosition	0x6064
	ActualVelocity	0x606C
	ActualToque	0x6077
	ModeofOperationDisp	0x6061

쓰레드 구현 대비 다음과 같은 두 가지 이점을 얻을 수 있다. 첫 번째, 모션 궤적을 계산하는 수행시간이 제외되어 I/O 쓰레드의 제어주기를 줄일 수 있으며 이로 인해 모션의 정밀도 (Precision)를 확보할 수 있다. 두 번째는 모션 계산에 소요되는 시간의 편차로 인한 I/O 쓰레드 내 제어 명령 전송 시점의 편차가 축소되어 향상된 전송 주기성을 제공할 수 있다는 점이다.

2. 통합 개발환경과의 연동방안 설계 및 구현

제안한 소프트웨어 아키텍처를 구현하기 위해 먼저 Beremiz에서 생성하는 런타임 수행 구조를 분석하였다. 런타임 수행구조는 초기화 과정 (Initialization Phase), 드라이브 데이터 수신 과정 (Retrieve Phase), 수신된 데이터를 이용하여 모션을 생성하는 과정 (Computation Phase), 그리고 이를 EtherCAT 프레임의 형태로 생성 및 전송하는 과정 (Publish Phase)으로 구성된다. 초기화 과정에서는 pthread API 들을 이용한 모션 런타임 쓰레드의 생성 및 쓰레드 수행주기 설정, 그리고 EtherCAT 마스터 스택의 초기화 함수들이 호출된다. 그리고 Retrieve Phase와 Publish Phase는 IgH EtherCAT 마스터 스택에서 제공되는 API 들이 사용되며, Computation Phase는 모션 계산을 위한 루틴들이 호출된다. III. 1절에서 제안한 아키텍처에 따르면 Retrieve Phase와 Publish Phase는 I/O 쓰레드, Computation Phase는 MC 쓰레드가 담당하게 된다.

그림 3은 본 논문에서 제안하는 아키텍처가 통합된 통합 개발환경의 구조와 이를 이용한 멀티코어 환경에서 동작하는 런타임 코드의 생성과정을 나타낸다. 기존 Beremiz에서 생성하는 리눅스 타겟 런타임은 주기 쓰레드 구현을 위해 리눅스 타이머를 사용하며, 쓰레드 스케쥴링 정책 및 우선순위 역시 일반 쓰레드로 설정된다. 이로 인해 다른 비 실시간 쓰레드 및 시스템 쓰레드들에 의한 선점으로 실시간성 보장이 어렵다. 따라서 본 연구에서는 일반적으로 잘 알려진 실시간성 확보를 위한 기법들 [13]을 적용하였으며, 주요 내용은 다음과 같다.

우선 타겟 제어기 운영체제에 의존적인 코드 템플릿에 대해 쓰레드 생성 전 초기화 과정에서 mlockall 함수를 사용하여 생성되는 쓰레드들이 페이지 스와핑에 의한 쓰레드 수행 간섭으로 인한 실시간 성능저하를 겪지 않도록 하였다. 이후 런타임 수행과정에서 다른 쓰레드들의 수행으로 인한 선점 효과가 최소화되도록 pthread_attr_setschedpolicy 함수와 pthread_attr_schedparam 함수를 사용하여

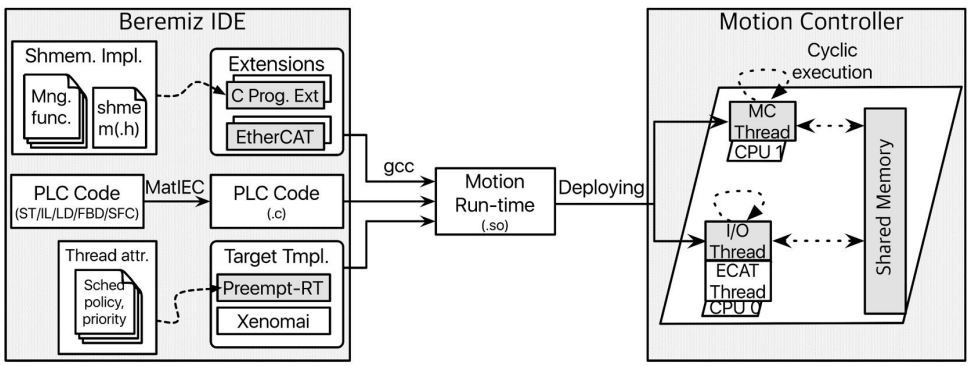


그림 3. 멀티코어 활용을 위한 Beremiz 통합개발환경 상에서의 런타임 코드 생성과정
Fig. 3 Runtime code generation procedure on the Beremiz IDE for multi-core controller

스케쥴링 정책 및 쓰레드 우선순위 설정을 하도록 Wrapper 함수를 작성하였다. 스케쥴링 정책과 우선순위는 각각 FIFO 스케쥴링 정책과 시스템 관리 쓰레드들보다 한 단계 낮은 98로 설정하였다. 그리고 멀티코어 환경에서 비 실시간 태스크로부터의 수행 간섭을 최소화하기 위해 사용되는 CPU shielding 기법을 CPU_SET 매크로와 pthread_attr_setaffinity_np 함수로 구현하여 각 쓰레드들이 모션 제어기의 특정 코어에서만 수행되도록 하였다. 예를 들면, 듀얼코어 기반 모션 제어기 상에서 I/O 쓰레드는 Core 0, MC 쓰레드는 Core 1에서 각각 수행된다. 마지막으로 쓰레드 주기성을 확보하기 위해 기존 리눅스 타이머를 사용한 주기 쓰레드 구현방식을 ns 단위의 해상도를 제공하는 clock_nanosleep 함수를 이용한 one-shot 타이머 구현으로 변경하였다.

한편, 공유 메모리 관련 기능들을 추가하기 위해 Beremiz에서 제공하는 C Extension을 사용하였다. C Extension은 프로그램 코드 수정이 가능한 에디터 및 C 언어 변수와 PLC 프로그래밍 변수간 연동을 위한 테이블 형태의 사용자 인터페이스를 제공한다. 에디터에는 헤더파일 및 전역변수, 런타임 수행 구조와 대응되는 함수별 섹션이 정의되어 있으며 각각 모션 런타임 빌드과정에서 해당하는 런타임 수행구조에 자동으로 삽입된다. 먼저 표 1의 PDO 변수들을 C 구조체를 사용하여 별도의 헤더파일 (shmem.h)로 정의하였다. 그리고 System V IPC 형식의 공유 메모리 API 들을 사용하여 C Extension 에디터 내 함수별 섹션에 생성 (shmem_create), 삭제 (shmem_remove), 읽기 (shmem_read), 그리고 쓰기 (shmem_write) 함수들을 구현하였다. 구현된 함수들은 향후 Beremiz 런타임 코드 수행구조와 대응되는 부분에서 호출되게 된다. 예를 들면, 공유 메모리 생성은 EtherCAT 마스터 스택 초기화 함수 호출 이후에 이루어지며 공유 메모리 읽기 및 쓰기 과정은 모터 드라이브 구동을 위한 PDO 교환과정에서 이루어진다.

그림 4는 초기화 과정 이후 모션 런타임의 주기적인 수행 흐름을 나타낸다. 먼저 I/O 쓰레드가 Retrieve Phase에서 EtherCAT 마스터 스택을 통해 모터 드라이브가 회신한 TxPDO를 처리한다. 그 후 MC 쓰레드가 이전 제어주기에서 계산한 새로운 경로값 (RxPDO)을 공유 메모리로부터 읽어온 후 TxPDO를 공유 메모리에 쓰기 작업을 하여 MC 쓰레드 내의 trj_gen 함수를 이용해 새로운 경로값을 업데이트할 수 있도록 한다. 그리고 공유 메모리로부터 읽어온 RxPDO를 이용하여 Publish Phase

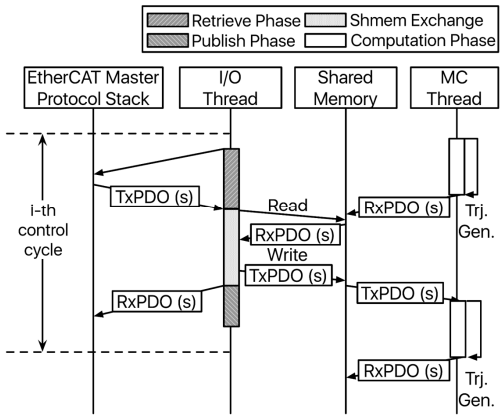


그림 4. 모션 런타임의 수행 시퀀스
Fig. 4 Execution sequence of motion runtime

에서 EtherCAT 프레임을 생성하고 이를 모터 드라이브에게 전송한다. 전체 모션 시스템 관점에서 볼 때 I/O 쓰레드에서의 프레임 전송 주기성이 매우 중요하며, 올바른 제어 명령을 포함한 EtherCAT 프레임이 전송되어야 한다. 따라서 I/O 쓰레드가 공유 메모리에 접근할 때 항상 최신의 RxPDO 변수값을 기반으로 제어 명령을 생성할 수 있도록 MC 쓰레드는 I/O 쓰레드의 수행주기의 절반으로 수행된다. 그리고 MC 쓰레드의 수행을 I/O 쓰레드와 겹치지 않게 설정하여 I/O 쓰레드가 항상 공유 메모리를 참조하여 EtherCAT 프레임을 전송하며, MC 쓰레드가 항상 새로운 경로값을 공유 메모리에 업데이트 할 수 있도록 했다. 또한, 프레임 전송 주기성을 보장하기 위해서는 I/O 쓰레드에서의 공유 메모리 읽기/쓰기 함수의 수행시간과 편차가 최대한 작아야 한다. 멀티쓰레드 환경에서는 공유되는 자원의 데이터 일관성을 보장하기 위해 동기화 매커니즘을 사용하는 것이 일반적이지만, 동기화 기법 사용에 따른 오버헤드를 줄여 I/O 쓰레드에서의 프레임 전송 주기성 보장이 용이하도록 본 연구에서는 두 개의 쓰레드들이 공유 메모리 자원에 접근하여 읽기/쓰기를 수행하는 메모리 영역을 분리하였다.

Ⅳ. 성능 평가

1. 실험 환경

본 논문에서 제안한 소프트웨어 아키텍쳐 기반 모션 런타임의 성능 평가는 그림 5와 같이 3대의 EtherCAT 모터 드라이브가 각각 x1, x2, z축을 제어하는 3축 산업용 로봇 플랫폼 상에서 진행하였다.

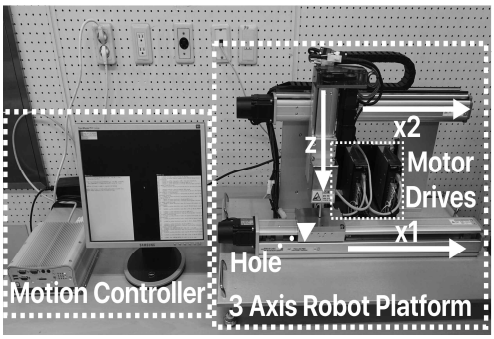


그림 5. 성능평가 테스트베드
Fig. 5 Test-bed system for performance evaluation

로봇 플랫폼의 동작 시나리오는 다음과 같다. x1 축이 먼저 특정 위치로 이동한 후 x2, z 축이 x1 축에 부착된 Hole을 따라 이동하는 시퀀스가 반복된다. 모션 제어기는 듀얼 코어 Intel Atom N2800 1.86 GHz, DDR3 4GB 메모리와 128GB SSD, Intel e1000e 1Gbps 네트워크 카드를 탑재하고 있는 산업용 PC 를 사용하였다. 제어기 플랫폼의 운영체제와 개발환경을 포함한 주요 소프트웨어는 Preempt-RT 패치가 적용된 리눅스 3.8.13-rt16 버전과 IgH EtherCAT Master 1.5.2, Beremiz 1.1 버전이 각각 사용되었다. 또한, 런타임 코드 수준에서의 실시간성 보장을 위한 설정사항 외에도 플랫폼 수준에서의 실시간성 보장을 위한 설정 사항들 [14]을 적용하였다. 모션 제어기 구성 시 불필요한 커널 디버깅 관련 기능 및 장치 드라이버 등을 제외하였으며, 멀티코어 환경에서 사용되는 isolcpus 커널 부팅 파라미터를 사용하여 두 번째 코어 (Core 1)를 리눅스 스케쥴러 도메인에서 분리시킨 런타임 환경으로 설정하였다. 더불어 BIOS에서 제공되는 시스템 온도 모니터링 및 동적 팬 관리기능 등과 같은 NMI (Non Maskable Interrupt), SMI (System Management Interrupt) 들을 비활성화 하였다.

2. 성능 평가 결과

EtherCAT 기반 제어 시스템의 일반적인 성능 평가 지표는 제어기에서 명령을 전송하는 간격을 나타내는 제어주기와 EtherCAT 슬레이브 장치인 모터 드라이브가 동작하는 시점 간의 편차다 [15, 16]. 제어주기가 짧을수록 생성되는 모션의 정밀도 (Precision)는 향상되며, 각 슬레이브 장치들 간의 동작 시점의 편차가 작을수록 모션의 동시성 (Synchroneity)은 높아진다 [15]. EtherCAT 분산 시계 (이하 DC: Distributed Clock) 기반의 동기화 기법을 사용하는 경우 모션의 동시성이 1 μs 이내로 보장될 수 있음이 실험적으로 알려져 있으나 [16], DC 기법의 정상 동작을 위해서는 두 가지 전제 조건을 만족해야 한다. 첫 번째는 EtherCAT 프레임이 설정한 제어주기와 동일한 간격으로 모터 드라이브에 도착해야 하며, 두 번째는 모든 제어주기에 대해 드라이브의 EtherCAT 프레임 도착 시점이 DC 이벤트 발생시점 이전이어야 한다.

모터 드라이브 내 소프트웨어 지연시간과 통신매체에 의한 지연시간은 예측 가능하므로 [15, 16],

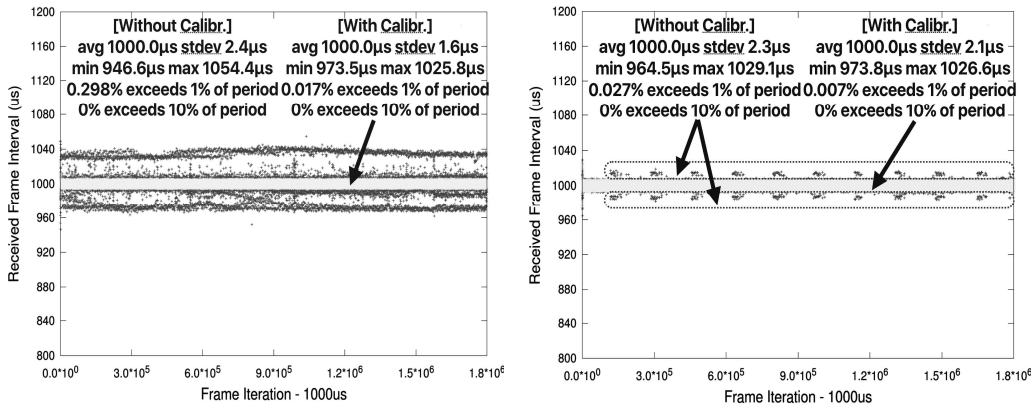
표 2. I/O 쓰레드 내부 측정 결과 (단위: μs)
Table 2. Internal measurement result of I/O Thread (Unit: μs)

Core Assign.	Item	Control Cycle	Min.	Max.	Diff. (Max.-Min.)	Stdev
Core 0	Release Jitter	1000	973.0	1037.0	64.0	0.9
		500	472.2	524.6	52.4	0.7
		250	228.2	270.0	41.8	0.5
	Frame Interval Cycle	1000	956.9	1065.2	108.3	2.3
		500	455.7	542.5	86.8	1.5
		250	217.9	280.9	63.0	0.9
Core 1	Release Jitter	1000	990.0	1008.0	18.0	0.5
		500	487.4	511.1	23.7	0.4
		250	230.5	268.9	38.4	0.3
	Frame Interval Cycle	1000	982.4	1011.9	29.5	0.5
		500	469.6	527.4	57.8	0.5
		250	232.6	266.4	33.8	0.4

모터 드라이브에 EtherCAT 프레임이 도착하는 간격은 프레임 전송주기가 얼마나 균일한지에 달려있다. 먼저 I/O 쓰레드를 어떤 코어에 매핑하느냐에 따른 차이와 제어주기별 성능 차이를 확인하기 위해 I/O 쓰레드를 Core 0과 1에 각각 매핑했을 경우에 대해 제어주기를 각각 1000 μs, 500 μs, 250 μs로 변경하면서 태스크 릴리즈 지터와 EtherCAT 프레임 전송 명령이 시작되는 주기를 측정하는 방식으로 실험을 진행하였다. 릴리즈 지터는 I/O 쓰레드의 예정된 시작 시점과 실제로 시작된 시점의 차이로 실시간 성능을 나타내는 지표가 되며 지터 값의 편차가 작을수록 좋은 실시간 성능을 나타낸다. 프레임 전송 시점은 태스크가 실제 릴리즈되어 Retrieve Phase와 공유 메모리 읽기/쓰기 과정 이후 시점을 나타내며, 모션 동시성을 보장하기 위해서는 전송 시점의 편차가 작아야 한다. 측정은 리눅스 시스템에서 ns 단위의 시간 해상도를 제공하는 clock_gettime 시스템 콜을 런타임 코드에 추가하여 사용하였으며, 각 실험별로 10분씩 이루어졌다.

모든 실험 셋에 대해 측정된 평균값들은 설정한 제어주기와 동일했으나, 실제 편차는 각각 다르게 나타났다. 표 2에 제시된 측정결과와 같이, I/O 쓰레드를 Core 0에 매핑한 경우에는 제어주기와 상관없이 ±40 μs 정도의 릴리즈 지터가 발생했으며, 제어주기가 작을수록 최대편차 (Diff.)와 표준편차가 감소하는 경향을 보였다. 이는 선행 연구들 [4, 15, 16]에서 제시한 바와 같이 제어주기가 작아질수록 커널 내 백그라운드 태스크들에 의한 간섭이 발생할 확률이 낮아져 캐시 메모리 활용도가 높아지는 것이 원인으로 생각된다. 한편, 프레임 전송 시점 주기성은 릴리즈 지터의 경향을 그대로 반영하고 있으며, 편차가 소폭 증가한 것을 확인할 수 있다. 이러한 결과의 원인으로는 EtherCAT 마스터 커널 모듈로부터 I/O 쓰레드에 데이터를 가져올 때 발생하는 데이터 복사 시간과 공유 메모리 참조를 위한 함수들의 수행시간 편차에 의해 누적된 지터의 영향으로 판단된다. 반면, I/O 쓰레드를 Core 1에 매핑한 경우 릴리즈 지터는 최대 ±20 μs 정도로 매우 좋은 주기성을 나타내고 있으며, 제어주기별 릴리즈 지터의 편차가 Core 0 실험 결과 대비 각각 71.9%, 54.8%, 61.6% 정도 감소함을 알 수 있다. 이전 실험결과와는 달리 제어주기와 편차의 관련성은 없어 보이지만, 주기가 작을수록 표준편차가 소폭 감소하는 경향이 나타났다. 프레임 전송 시점의 주기성 측면에서도 이전 실험 결과 대비 각각 72.8%, 43.8%, 46.3% 정도의 편차가 감소한 결과를 보이고 있다. 측정 결과로 볼 때, I/O 쓰레드를 Core 1에 매핑했을 때는 core shielding 기법과 isolcpus 옵션이 더욱 유효하게 작용한 것을 알 수 있다. 이는 Core 0에서 동작하는 커널 쓰레드들과 Core 1에서 동작하는 I/O 쓰레드 간 수행 경합 상황이 감소한 것이 주요 원인으로 생각된다.

또한, 시스템의 실시간 성능을 실제 드라이브에서 평가하기 위하여 선행 연구 [4, 15]에서 구축한 것과 같은 외부 측정 시스템을 이용하여 첫 번째 모터 드라이브에서 관찰된 프레임 수신 주기성을



(a) Mapping I/O thread to Core 0 (b) Mapping I/O thread to Core 1

그림 6. 프레임 수신 간격 측정 결과 (듀얼 코어)

Fig. 6 Measurement result of frame reception period (Dual-core)

추가로 측정하였다. I/O 쓰레드의 제어주기는 모션 제어를 위한 기준 제어주기인 1000 μs로 설정하였으며, 각 실험은 30분씩 이루어졌다. 그림 6에 제시된 측정 결과로 볼 때 내부 측정 결과와 비슷한 경향을 보임을 알 수 있다. I/O 쓰레드를 Core 0과 Core 1에 할당 시 프레임 수신 주기의 편차가 각각 107.8 μs, 74.6 μs, 표준 편차는 2.4 μs, 2.3 μs 정도로 나타났다. 그러나 I/O 쓰레드를 Core 1에 할당하는 경우 주기의 1%인 10 μs를 초과하는 측정 샘플의 비율은 0.298%에서 0.027%로 대폭 감소하는 것을 확인할 수 있다. 그러나 그림 6. (b)에 나타난 것과 같이 프레임 수신 주기성이 소폭 흐트러지는 경향을 보이는데, 이는 Core 1의 로컬 타이머 인터럽트에 의한 간섭이 주요 원인으로 판단된다.

프레임 전송의 주기성 확보는 EtherCAT DC 기능을 사용하기 위한 선결조건이다. 프레임 주기성을 향상하기 위해 선행연구 [15]에서 제안한 프레임 전송 시점 조절 프레임워크를 적용하였으며, 런타임 코드 내부 측정 결과를 바탕으로 I/O 쓰레드에서의 전송시점을 제어 주기 내 60% 시점으로 조절한 후 프레임 전송 주기를 다시 측정하였다. 그림 6에 제시된 측정 결과를 보면 프레임 수신 주기의 최대편차가 각각 52.3 μs, 52.8 μs로 감소했으며, 표준편차도 각각 1.6 μs, 2.1 μs로 감소하였다. 기존 실험 대비 프레임 수신 주기의 편차가 각각 51.5%와 29.2%, 주기의 1% 인 10 μs를 초과하는 샘플의 비율은 각각 0.017%와 0.007%로 감소하였다. I/O 쓰레드를 Core 0에 매핑한 경우와 같이 수행 간섭의 확률이 높은 상황에서는 전송 시점 조절 기법 적용에 따른 주기성 향상 효과가 크며, Core 1에 매핑한 경우에도 약간의 성능이 향상됨을 알 수 있었다.

외부 성능평가 결과를 토대로 3 대의 드라이브 간 모션 구동의 동시성을 평가하기 위해 런타임 코드에 DC 기능을 적용하고, 성능을 평가하였다. DC 동기화 이벤트 발생주기는 프레임 전송주기와 동일하게 1000 μs로 설정하였다. 성능평가 항목으로 전역시계와 각 모터 드라이브에서 생각하는 전역 시계 값의 차이인 System Time Difference 를 선정하고, 측정은 EtherCAT 마스터 스택에서 제공하는 관련 API를 런타임 코드에 추가하여 10분간 진행하였다. 그림 7에 제시된 성능 평가 결과에서 볼 수 있듯이 런타임 코드 수행 초기에는 모터 드라이브들간 최대 1 s 가량의 편차가 발생하나 7.4 s 이후 안정화되어 그 이후 평균 10 ns, 최대 689 ns, 표준편차 34.4 ns의 동기화 편차를 보여 높은 수준의 동기화 성능을 제공할 수 있음을 알 수 있다.

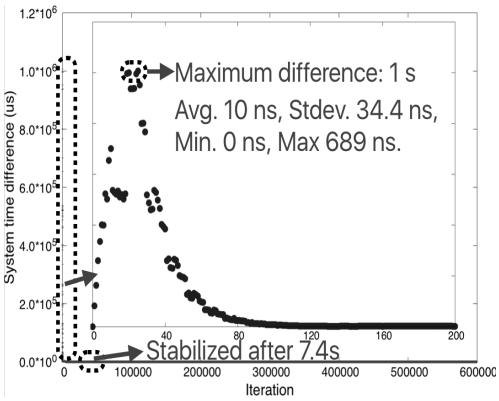


그림 7. 시스템 시간 편차 측정 결과
Fig. 7 Measurement result of system time difference

V. 결 론

본 논문에서는 기존 싱글 쓰레드 구현 방식의 모션 응용을 공유 메모리를 이용한 멀티 쓰레드 구현 방식으로 전환하여 멀티코어 환경에 대응할 수 있는 소프트웨어 아키텍처를 제안하였다. 제안한 아키텍처는 CiA 402 표준을 기반으로 하여 확장성을 제공할 수 있으며, 멀티코어 기반 리눅스를 대상으로 실시간 성능 보장을 위한 다양한 설정사항들을 포함한다. 아키텍처의 동작 검증을 위해 오픈소스 통합 개발환경에 확장 구현하였으며, 사례 연구로 3축 로봇 플랫폼에서 성능평가를 진행하였다. 성능평가 결과 1000 μs 제어주기 상에서 프레임 수신 주기의 편차가 최대 53 μs 가량, 각 드라이브 간 동작 편차가 평균 10 ns, 최대 689 ns 이내로 상당히 좋은 동기화 성능을 보임을 알 수 있었다.

향후 연구로는 타겟 모션 제어기의 코어 수와 코어별 CPU 자원 사용량 등의 정보를 토대로 최적의 쓰레드-코어 할당 알고리즘과 관련 정보를 제공하는 프레임워크 개발 및 멀티코어 임베디드 제어기로의 적용을 고려하고 있다.

References

[1] M. Riedl, H. Zipper, M. Meier, C. Deidrich, “Cyber-physical systems alter automation architectures,” Annual Reviews in Control, Vol. 38, No. 1, pp. 123-133, 2014.

[2] P. Leitao, A.W. Colombo, S. Karnouskos,

"Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges," Computers in Industry, Vol. 81, pp. 11-25, 2015.

[3] I. Kim, S. Park, M. Sung, T. Kim, "Design and implementation of a real-time motion controller using open source software," Journal of KIISE: Computer Systems and Theory, Vol. 39, No. 2, pp. 84-95, 2012 (in Korean).

[4] C. Kim, I. Kim, T. Kim, "Xenomai-based embedded controller for high-precision, synchronized motion applications," Journal of KIISE: Computing Practice, Vol. 21, No. 3, pp. 173-182, 2015 (in Korean).

[5] I.K. Jung, J.H. Kim, "Real-time centralized soft motion control system for high speed and precision robot control," IEMEK J. Embed. Sys. Appl., Vol. 8, No. 6, pp. 295-301, 2013 (in Korean).

[6] H. Son, D. Kang, J. Lee, "User-oriented controller design for multi-axis manipulators," IEMEK J. Embed. Sys. Appl., Vol. 3, No. 2, pp. 49-56, 2008 (in Korean).

[7] A. Canedo, M.A. Al-Faruque, "Towards parallel execution of IEC 61131 industrial cyber-physical systems applications," Proceedings of the Conference on Design, Automation Test in Europe, pp. 554-557, 2012.

[8] A. Canedo, H. Ludwig, M.A. Al-Faruque, "High communication throughput and low scan cycle time with multi/many-core programmable logic controllers," IEEE Embedded Systems Letters, Vol. 6, No. 2, pp. 21-24, 2014.

[9] J. Choi, H. Kang, K. Kim, "Real-time pipelining scheduling in multi-core motion controllers," Journal of KIISE: Transactions on Computing Practice, Vol. 20, No. 5, pp. 291-295, 2014 (in Korean).

[10] EtherCAT Technology Group Std., "ETG 6010: EtherCAT implementation directive for CiA402 drive profile," 2013.

[11] I. Kim, T. Kim, M. Sung, E. Tisserant, L. Bessard, C. Choi, "An open-source development environment for industrial automation with EtherCAT and PLCopen motion control," Proceedings of the IEEE 18th International Conference on Emerging Technologies and Factory Automation, pp. 1-4, 2013.

[12] P. Danielis, J. Skodzik, V. Altmann, E.B. Schweissguth, F. Golatowski, D. Timmermann, J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," Proceedings of the 19th IEEE International Conference on Emerging Technologies and Factory Automation, pp. 1-8, 2014.

[13] H. Yoon, J. Song, J. Lee, "Real-time performance analysis in linux-based robotic systems," Proceedings of the 11th Linux Symposium, pp. 331-339, 2009.

[14] K. Erwinski, M. Paprocki, L. Grzesiak, K. Karwowski, A. Wawrzak, "Application of ethernet powerlink for communication in a linux RTAI open CNC system," IEEE Transactions on Industrial Electronics, Vol. 60, No. 2, pp. 628-636, 2013.

[15] I. Kim, T. Kim, "Guaranteeing isochronous control for networked control systems," Sensors, Vol. 15, No. 6, pp. 13945-13965, 2015.

[16] M. Sung, I. Kim, T. Kim, "Toward a holistic delay analysis of EtherCAT synchronized control process," International Journal on Computer, Communication and Control, Vol. 8, No. 4, pp. 608-621, 2013.

Ikhwan Kim (김 익 환)

He received B.S. and M.S. degrees in Mechanical and Information Engineering from University of Seoul, Korea, in 2008 and 2011, respectively. He is currently a Ph. D. candidate. His current research interests focus on cyber physical systems (CPS), real-time systems, and industrial automation.
Email: ihkim@uos.ac.kr

Hyosung Ahn (안 효 성)

He received B.S. degree in Mechanical and Information Engineering from University of Seoul in 2014. He is currently a master student. His current research interests focus on industrial automation and real-time systems.
Email: hyosung88@uos.ac.kr

Taehyoun Kim (김 태 현)

He received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Korea, in 1994, 1996, and 2001, respectively. He is currently a professor with the Department of Mechanical and Information Engineering, University of Seoul, Korea. His current research interests include embedded systems, real-time systems, and industrial automation.
Email: thkim@uos.ac.kr