J. of the Korean Sensors Society
Vol. 16, No. 1 (2007) pp. 52 – 61

저비용 네트워크 기반 임베디드 시스템을 위한 시간동기 기술

이 동 익†

Fault-tolerant clock synchronization for low-cost networked embedded systems

Dongik Lee†

Abstract

Networked embedded systems using the smart device and fieldbus technologies are now found in many industrial fields including process automation and automobiles. However the discrepancy between a node's view of current time and the rest of the system can cause many difficulties in the design and implementation of a networked system. To provide a networked system with a global reference time, the problem of clock synchronization has been intensively studied over the decades. However, many of the existing solutions, which are mainly developed for large scale distributed computer systems, cannot be directly applied to embedded systems. This paper presents a fault-tolerant clock synchronization technique that can be used for a low-cost embedded system using a CAN bus. The effectiveness of the proposed method is demonstrated with a set of microcontrollers and DC motor-based actuators.

Key words : clock synchronization, networked systems, embedded systems, fault-tolerance, CAN bus

1. 서 론

전통적으로 폐루프 제어 시스템은 하나의 중앙 컴퓨터가 아날로그 방식을 통해서 모든 센서 및 액츄에이터와 직접 연결되는 중앙집중 구조로 구현되었다. 그러나 전자공학의 발전에 힘입어, 1990년대 이후 디지털 필드버스(fieldbus)와 지능형 센서 및 엑츄에이터를 이용한 네트워크 기반 임베디드 제어 시스템 구조(그림 1)가 산업계 전반에 걸쳐 광범위하게 적용되고 있다. 이러한 네트워크 기반 제어기술은 이제 고도의 안전성과 신뢰도가 요구되는 안전중요 시스템(safety-critical systems) 분야에까지 확대적용되기에 이르렀으며, 대표적인 사례로써 네트워크 기반 제트엔진[1], 차세대 지능형 자동차[2] 등을 들 수 있다.

그러나 통신 네트워크가 시스템에 포함됨으로써 중앙집중 구조에서는 고려할 필요가 없었던 새로운 문제점들이 발생한다. 특히 통신 네트워크에 의한 전송지연과, 동일한 변수가 노드 마다 서로 다른 값으로 인식되는 불일치 현상은 시스템의 성능을 저하시키거나 오류를 유발하는 요인이 된다. 특히 불규칙하게 변하는 지연시간은 결과적으로 제어시스템이 시변(time-varying) 특성을 갖게 함으로써, 설계 및 구현에 큰 어려움으로 작용한다. 이를 극복하기 위해서 많은 연구들이 진행되고 있으며, 지금까지의 주요 연구내용들은 참고문헌[3-5]에서 찾아볼 수 있다. 그러나 대부분의 연구 결과들은

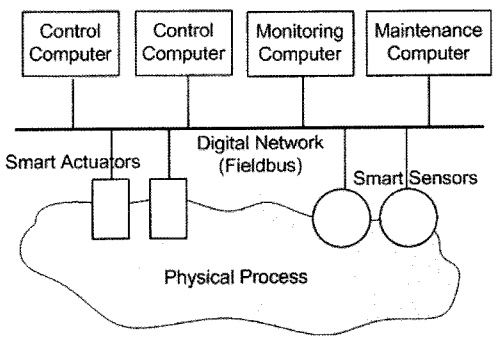


그림 1. 네트워크 기반 임베디드 시스템
Fig. 1. A networked embedded system.

경북대학교 전자전기컴퓨터학부 (School of Electrical Engineering and Computer Science, Kyungpook National University)

†Corresponding author: dilee@ee.knu.ac.kr
(Received : August 29, 2006, Accepted : September 14, 2006)

버퍼나 관측기 등을 이용하여 지연시간의 불규칙성을 최소화하는 방법에 주안점을 두고 있다[6,7]. 그런데 이러한 접근법은 과도한 계산량, 복잡한 소프트웨어 구조, 모델링 오차 등으로 인해서 저비용 임베디드 시스템에 적용하기 어렵다.

전송지연과 데이터 불일치 문제를 함께 해결하기 위한 또다른 접근법은 전체 시스템을 시간분할(time-triggered) 방식으로 구현하는 것이다[8]. 시간분할 방식이란 TDMA(time division multiple access) 기법에 기초한 시스템 구현 방식을 가리킨다. 즉 모든 연산과 통신은 시스템 설계 단계에서 설정된 시간 스케줄에 따라 주기적으로 수행된다. 시간분할 방식을 적용할 경우 시스템을 구성하는 센서, 액츄에이터, 제어기, 통신 등의 작업들이 일정한 시점에 일어나기 때문에 지연시간을 최소화할 수 있을 뿐 아니라 시스템의 시간특성을 미리 예측할 수 있게 된다. 그 결과 시스템의 설계/분석 및 구현이 용이하고 시스템의 신뢰도도 향상되는 장점이 있다. 예를 들면, 네트워크로 연결된 다중 액츄에이터 시스템의 경우, 시간분할 방식을 적용함으로써 모델기반 관측기와 복잡한 제어 알고리즘을 이용하지 않고, 단순한 비례제어만으로도 요구성능을 만족시킬 수 있음을 알 수 있다[9].

그러나 시간분할 방식을 적용하기 위해서는 반드시 시스템을 구성하는 모든 노드들이 공통으로 인식할 수 있는 기준시간이 반드시 필요하다. 이 때 각 노드의 클럭을 이용하여 전역 기준시간을 추정하는 기술을 시간동기(clock synchronization) 라고 한다. 대규모 분산 컴퓨터 시스템을 위한 시간동기 기술에 관한 연구는 이미 컴퓨터 과학 분야를 중심으로 20년 이상 광범위하게 진행되었으며, 그 결과 매우 다양한 방법들이 개발되었다[10-12]. 그러나 기존 방법들은 대부분 고속 컴퓨터 통신 네트워크로 연결된 워크스테이션 등에 적용하기 위해서 개발되었으며, 일반적으로 복잡한 연산과 알고리즘 구조, 대규모 메시지 전송 등이 포함된다. 반면에, 임베디드 시스템의 경우 통신 대역폭, 메모리, 연산처리속도, 클럭의 품질, 동작환경 등에 있어서 많은 제약이 따르기 때문에 이들을 임베디드 시스템에 직접 적용하기는 어렵다.

본 논문에서는 먼저 네트워크 기반 임베디드 시스템에 있어서 정밀하고 신뢰도가 높은 시간동기 기술의 중요성과 효용성을 살펴본다. 이어서 저비용 임베디드 제어시스템에 많이 이용되고 있는 CAN버스 기반의 시간동기 알고리즘을 제안한다. 이 방법은 저비용 프로세서 및 필드버스를 이용한 시스템에 적용할 수 있도록 단순한 구조, microseconds 수준의 동기 정밀도, 확정성, 유연성, 고장대처 능력 등을 고려하여 설계하였다. 제안한 방법은 마스터-슬레이브(master-slave) 구조를 갖는 소프트웨어 방식 알고리즘이며, 마스터 클럭의 고장에 대처할 수 있도록 3개의 클럭을 마스터 그룹으로 사용하였다. 또한 메시지 전송지연으로 인한 낮은 동기 정밀도를 개선하기 위해서 Gergleit & Streich[13]가 제안한 방법을 활용하였다. 끝으로, 마이크로컨트롤러와 CAN 버스로 구성된 steer-by-wire 모델 시스템을 이용한 실험을 통해서, 제안한 방법이 메시지 전송지연과 데이터 불일치로 인한 문제점들을 효과적으로 개선시킬 수 있음을 확인하였다.

2. 시간동기의 필요성

네트워크 기반 시스템의 성능 및 특성은 각 노드에서 처리되는 태스크들의 실행결과에 의해 결정된다. 이때 각 노드의 태스크 스케줄은 프로세서에 내장된 클럭을 이용하여 추정된 시간을 기준으로 처리된다. 따라서 클럭의 정확도가 전체 시스템의 성능에 중대한 영향을 미친다. 그러나 컴퓨터에 내장된 클럭의 특성은 진동, 온도, 습도, 기압 등 외부 환경에 의해 비선형적으로 변하며, 일반적으로 10^{-4}~10^{-6} sec/sec의 드리프트(drift)를 갖는다[14]. 드리프트에 의해 발생한 클럭오차는 실시간 태스크의 데드라인을 만족시키지 못하는 등 매우 다양하고 심각한 시스템 오류들을 유발시키며, 이로 인한 실제 사고 사례들도 보고된 바 있다[15]. 시간동기 기술이란, 이처럼 불완전한 각 노드의 클럭(local clock)을 이용하여 근사적인 전역 기준시간(approximate system-wide time reference)을 추정함으로써, 시스템에 포함된 임의의 두 클럭 사이의 동기 오차를 허용범위 이내로 유지하는 기술을 가리킨다. 이 장에서는 임베디드 시스템에 있어서 시간동기 기술의 필요성에 대해서 간략히 살펴본다.

시간분할 통신(time-triggered communications): "Steer-by-wire"와 같은 안전중요 시스템에서는 불규칙하게 변하는 메시지 전송 지연시간은 안전성과 신뢰도 측면에서 매우 큰 장애요소로 작용한다. 이 문제를 극복하기 위해서 최근 자동차 산업을 중심으로 엄격한 사전 스케줄을 적용하는 시간분할 통신 프로토콜이 개발되었다[16,17]. 그러나 시간분할 통신 프로토콜이 정상적으로 동작하기 위해서는 각 노드의 실시간 태스크 스케줄 관리를 위한 기준이 되는 전역 기준시간이 반드시 필요하다.

실시간 제어 시스템: 네트워크 기반 제어 시스템에 있어서 불규칙하게 변하는 지연시간은 시스템의 특성이

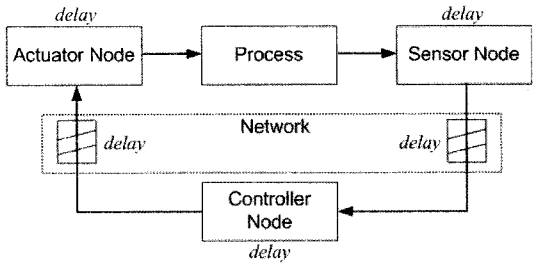


그림 2. 네트워크 기반 임베디드 시스템에서 시간지연 발생 요인들
Fig. 2. Various sources of delays in a networked embedded system.

시간에 따라 변하는 것과 같은 효과를 유발함으로써, 시불변 시스템의 설계분석을 위한 다양한 제어이론과 기법들을 적용할 수 없게 되며, 시스템의 특성이 불안정해질 수도 있다. 이러한 불규칙한 지연시간은 그림 2에서 보는 것과 같이 통신 네트워크 뿐 아니라, 센서와 엑츄에이터, 제어 컴퓨터에 이르기까지 시스템을 구성하는 모든 연산노드에 의해서 발생된다. 이 가운데서도 메시지 전송지연과 센서의 출력을 읽는 시점이 일정하지 않은 점이 가장 심각한 원인이다[18]. 다행스럽게도 메시지 전송지연은 시간분할 통신 프로토콜을 사용하여 개선할 수 있으며, 센서 측정값을 읽는 시점 역시 엄격한 시간분할 스케쥴을 적용함으로써 일정하게 유지할 수 있다. 그 결과 기존의 모델 기반 관측기나 버퍼를 이용하는 복잡한 방법 대신에 매우 간결한 제어기를 사용하더라도 네트워크 기반 시스템의 제어성능을 만족시킬 수 있게 된다. 이는 곧 상대적으로 저렴한 프로세서와 간결한 구조를 갖는 시스템을 의미하며, 그 결과 전체 시스템의 신뢰도 향상과 비용절감 효과로 이어질 수 있다.

데이터 측정: 전역 기준시간은 네트워크 기반 시스템을 위한 데이터 측정시 반드시 필요하다. 예를 들면 대부분의 데이터 측정 장치에서는 데이터의 인과관계를 추정할 수 있도록 각 데이터의 측정시점을 나타내는 시간우표(timestamp)를 이용한다. 그런데 전역 기준시간이 없거나 정확하지 않을 경우 각 노드에서 제공하는 시간우표의 일관성을 잃게 됨으로써, 마치 미래의 데이터가 과거의 데이터에 영향을 미치는 것과 같은 오류가 발생한다.

고장대처 기법: 여유도(redundancy)는 센서나 엑츄에이터 고장에 대처하기 위해 공통적으로 사용하는 방법이다. 그러나 전역 기준시간이 제공되지 않을 경우, 다중 여유 부품 사이의 고장진단을 위한 비교 알고리즘(voting algorithm)이 서로 다른 시점에 생성된 데이터를 기반으로 판단을 내림으로써 정확한 고장진단이 불가능하게 된다. 비동기 방식을 적용한 초기 F16 전투기의 시험비행에서, 세 대의 비행제어 컴퓨터가 모두 정상적으로 동작하고 있었음에도 불구하고, 각 컴퓨터가 나머지 컴퓨터를 서로 고장상태로 인식함으로써 시험비행을 중단한 사고는 고장진단 시스템에서 시간동기의 필요성을 보여주는 대표적 사례이다[15].

3. 소프트웨어 기반 시간동기 알고리즘의 구성요소

컴퓨터 과학 분야를 중심으로 수많은 시간동기 알고리즘이 개발되었으며, 이들 가운데 대부분은 소프트웨어 기반 알고리즘으로 분류된다. 일반적으로 소프트웨어 기반 알고리즘은 다음과 같은 세 가지 단계로 구성된다[10,12]:

- 재동기 검출(Resynchronization detection): 사전에 설정된 재동기 주기(resynchronization period)에 따라서 시간동기 알고리즘을 기동시키는 단계
- 클럭정보 수집(reading remote clock values): 시스템에 포함된 각 클럭노드가 추정한 현재시간 정보를 수집하는 단계
- 클럭교정(clock correction): 수집한 클럭 데이터를 이용하여 근사적인 전역 기준시간을 추정하고, 이를 기준으로 각 클럭의 오차를 보정하는 단계

이러한 3단계로 구성된 시간동기 알고리즘은 그림 3과 같이 순환적인 프로그램으로 구현될 수 있다. 여기서 F_c는 수렴함수(convergence function) 라고 불리며, 수집된 클럭정보를 이용하여 전역 기준시간을 추정한다. adj_p^{k+1}은 $(k+1)$번째 재동기 시점에서, 기준시간과 클럭 p의 오차를 제거하기 위해서 필요한 교정량을 가리킨다.

k=0;
adj_p^0=0;
do forever
  detect resynchronisation event at time R^{k+1};
  $adj_p^{k+1} = F_c(p, x_1^{k+1}, \ldots, x_N^{k+1}) - p$;
  k=k+1;
end

그림 3. 소프트웨어 방식 시간동기 알고리즘 구현
Fig. 3. Implementation of blocks for software-based clock synchronization.

4. 임베디드 시스템을 위한 시간동기 기술

4.1. 요구사양 분석

워크스테이션과 Ethernet 등으로 구성되는 대규모 분산 컴퓨터 시스템과는 달리, 네트워크 기반 임베디드 시스템은 지능형 센서/엑츄에이터 및 제어기에 내장된 프로세서와 저비용 필드버스를 이용하므로 연산처리능력, 메모리, 전원, 통신 대역폭 등에 있어서 많은 제약이 따른다. 따라서 대규모 분산 컴퓨터 시스템을 위해 개발된 기존의 시간동기 방법들은 알고리즘의 복잡성과 대량 메시지 전송 등으로 인해서 임베디드 시스템에 직접 적용하기는 어렵다. 이 장에서는 임베디드 시스템의 일반적인 특성을 고려하여, 저비용 네트워크 기반 임베디드 시스템을 위한 시간동기 알고리즘의 요구사양을 도출하고 기존에 연구된 방법들의 문제점을 분석한다. 먼저 일반적인 임베디드 시스템의 특성을 살펴보면 다음과 같이 요약할 수 있다[19]:

- 실시간 시스템: 임베디드 시스템은 기본적으로 실시간 시스템으로써, 시스템을 구성하는 모든 노드는 주어진 시간(deadline) 이내에 정확한 연산과 태스크를 처리할 수 있어야 한다.
- 소형 및 경량화: 이동성, 제한된 설치공간, 연료/전원 효율성 등의 이유로 소형 및 경량화가 요구된다. 이로 인하여 프로세서와 통신 네트워크의 사양도 제약을 받게 된다.
- 안전성과 신뢰도: 자동차, 항공기, 정밀가공기기, 화학플랜트 등 임베디드 시스템의 고장은 인명손상이나 환경피해로 이어질 위험성이 높기 때문에 고도의 신뢰도 수준을 만족할 필요가 있다. 제품의 신뢰도는 경제적 측면에서도 중요한 요소이다.
- 열악한 설치환경: 임베디드 시스템은 차량이나 제트엔진과 같이 진동, 습도, 온도, 압력 등 매우 열악한 환경 속에 노출되어 동작하는 경우가 많다.
- 비용과 효율성: 대부분의 임베디드 시스템은 대량생산 제품으로써, 제품의 가격과 효율에 매우 민감하다.
- 전원 소모: 소형/경량화 및 이동성 등이 요구되므로 전원이나 연료 효율은 매우 중요한 요구사양이다.
- 빠른 제품 주기 및 업그레이드: 제품 주기가 수 개월 이내인 경우가 많으므로, 전체 시스템을 재설계하지 않고 부분적인 수정보완이 가능해야 하며, 노드의 추가 또는 제거도 용이해야 한다.

이러한 특성들을 고려할 때, 임베디드 시스템을 위한 시간동기 알고리즘은 다음과 같은 요구사양을 만족시킬 수 있어야 한다:

표 1. 응용 시스템에 따른 시간동기 정밀도 요구수준 사례
Table 1. Synchronization requirements in typical applications

Application Area	Precision
Low speed sensors	Miliseconds
Electro-mechanical actuators	Miliseconds
Process control	100 μsec~1 msec
Automotive on-board control	10 μsec

- Microseconds 수준의 동기 정밀도: 정확한 실시간 처리는 시간동기 정밀도에 의해서 직접적으로 영향을 받으며, 동기 정밀도 요구수준은 적용 시스템의 특성에 따라서 달라진다. 임베디드 시스템을 위한 시간동기 프로토콜인 IEEE-1588[20]의 사용자 그룹에서 예시한 주요 응용분야별 시간동기 정밀도 요구수준[21]을 표 1에 요약하였다. 일반적으로 임베디드 시스템을 위한 시간동기는 microseconds 수준의 동기 정밀도를 제공할 수 있어야 할 것으로 판단된다.
- 고장대처: 시간동기 알고리즘은 전체 시스템의 동작특성과 안전 및 신뢰도에 직접 영향을 미치므로, 열악한 주변 환경과 일부 클럭의 고장에 대처할 수 있는 기능이 포함되어야 한다.
- 시스템 자원 사용 최소화: 소형, 경량화, 전원 소모량 등을 고려할 때, 시간동기 알고리즘을 수행하는데 필요한 연산량, 메모리 사용량, 메시지 전송량 등이 최소화되어야 한다.
- 확정성: 시스템의 안전성과 신뢰도를 확보하기 위해서는 반드시 시간동기 알고리즘의 확정성(determinism)이 만족되어야 한다. 확정성이란 알고리즘의 설계 단계에서 동작특성을 예측 및 분석 가능하며, 주어진 입력에 대해서 알고리즘 수행 결과는 유일하게 결정됨을 의미한다.
- 유연성: 효율적인 시스템 수정 및 업그레이드를 위해서, 시간동기 알고리즘은 노드를 추가하거나 제거하더라도 전체 알고리즘을 수정하지 않고 처리할 수 있는 유연성을 지원해야 한다.

4.2. 고장대처 및 동기정밀도 문제

이와 같은 요구사양을 고려할 때, 저비용 임베디드 시스템을 위한 시간동기 알고리즘은 마스트-슬레이브 구조를 갖는 소프트웨어 방식으로써, 고장대처 능력과 충분한 동기 정밀도를 제공해야 한다. 그 결과, 산업계

에서 일반적으로 이용되는 시간동기 방법들은 하나의 마스터 클럭이 주기적으로 동기 신호를 전송하고, 나머지 모든 클럭들은 이 신호를 기준으로 동기를 유지하는 단순한 프로그램으로 구현되는 경우가 많다. 그러나 앞서 제시한 요구사양과 비교할 때, 기존의 방법들은 고장대처 기능과 동기 정밀도 측면에서 만족되기 어렵다.

먼저 고장대처 관점에서 살펴보면, 기존 방법들은 단순한 구조를 유지하기 위해서 마스터 클럭의 고장에 대처할 수 있는 기능을 전혀 고려하지 않거나, 여러 개의 마스터 후보 가운데서 고장난 클럭을 제거하고 정상적인 마스터 클럭을 선택하기 위한 비교(voting) 알고리즘[22] 자체가 너무 복잡하여서 오히려 신뢰도를 저하시키는 것을 볼 수 있다. 일부 연구에서는 마스터-슬레이브 방식 대신에 구조적으로 고장대처가 용이한 분산 알고리즘[23]을 적용한 경우가 있으나, 이는 과도한 연산과 메시지 전송이 요구되므로 저비용 임베디드 시스템에 적용하기 어렵다.

소프트웨어 방식 시간동기 기술의 가장 큰 단점은 클럭 정보를 교환할 때 발생하는 메시지 전송지연으로 인해서 동기 정밀도 수준이 낮다는 점이다. 일반적으로 소프트웨어 기반 알고리즘의 동기 정밀도는 수 십 msec 이상이므로, 표 1에서 제시한 응용분야에 적용하기 어렵다. 마스터-슬레이브 구조를 갖는 소프트웨어 기반 시간동기 알고리즘의 동기오차(δ)는 다음과 같이 주어진다:

$$\delta = 2\rho R + \xi \quad (1)$$

여기서 ρ와 R는 클럭의 드리프트 속도와 재동기 주기를 나타낸다. ξ는 클럭 측정오류(clock reading error)라고 불리는 항으로써, 드리프트를 제외한 다른 원인에 의해서 발생하는 동기오차를 나타낸다. 클럭해상도, 연산지연, 전송지연 등이 주요 원인이다. 따라서 높은 수준의 클럭과 짧은 재동기 주기를 적용하더라도, 불규칙한 전송지연으로 인해서 동기 정밀도 수준은 크게 제약을 받게 된다. 최근에 제정된 임베디드 시스템을 위한 시간동기 프로토콜 표준인 IEEE-1588[20]은 nanoseconds 수준의 동기 정밀도를 제공하지만, 이는 Ethernet 기반의 고성능 대규모 임베디드 시스템에 주로 적용될 것으로 예상되며, 표준 프로토콜을 수행하기 위한 전용 칩과 정밀한 오실레이터 등 하드웨어의 지원이 필요하므로 저비용 임베디드 시스템에 적용하기는 어려울 것으로 판단된다.

Gergleit & Streich[13]는 그림 4에 제시한 것처럼 CAN 기반 시스템에서 전송지연으로 인한 동기오차를 줄일 수 있는 방법을 제안하였다. 즉 송신노드(마스터 클럭)

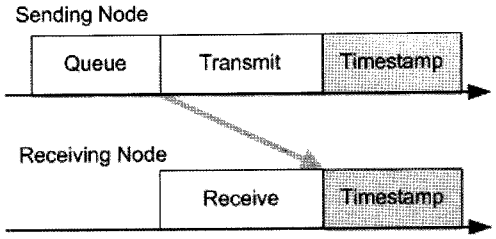


그림 4. 메시지 전송지연을 극복하기 위해서 Gergleit & Streich[13]가 제안한 방법

Fig. 4. A clock synchronization method proposed by Gergelit & Streich[13] to reduce the effect of message latency.

에서 현재 시간을 측정하여 그 정보를 메시지로 전송하는 것이 아니라, 메시지 전송이 끝나는 시점에 송신노드와 수신노드(슬레이브 클럭)가 동시에 현재 클럭 시간을 읽는 방법이다. 이는 CAN 프로토콜의 특성상 버스에 연결된 모든 노드들은 현재 버스에서 전송 중인 신호를 동시에 읽을 수 있으며, 메시지 프레임의 끝을 가리키는 EOF(end of frame) 비트가 송신/수신되는 시점을 동시에 인식할 수 있기 때문에 가능하다. 마스터 클럭의 시간은 다음 재동기 주기에 슬레이브 클럭으로 전송된다. 따라서 클럭 교정은 한 주기 이전에 측정한 마스터 클럭 시간을 기준으로 이루어진다. 이 방법을 적용하면 소프트웨어 방식임에도 불구하고 CAN의 1비트 시간(예를들면, 1 Mbit/sec일 경우 1 μsec)에 해당하는 동기정밀도를 얻을 수 있다. 그러나 이 방법은 마스터 클럭의 고장에 대처하는 방법은 고려하지 않았다.

5. 제안한 시간동기 알고리즘

본 논문에서는 Gergleit & Streich[13]가 제안한 방법을 이용하여 동기 정밀도 요구사양을 만족시키고, 마스터 클럭의 고장에 대처할 수 있도록 제한적인 분산 알고리즘을 적용하였다. 제안한 시간동기 알고리즘은 그림 5에 나타내었으며, 각 단계별 수행내용은 다음과 같다.

- [단계-1] 재동기 검출(detect resynchronization): 마스터 클럭으로 설정된 세 개의 클럭은 자신의 지역시간을 기준으로 n번째 재동기 시점을 검출한다. 이때 가장 먼저 재동기 시간 $T=nR$에 도달한 클럭이 동기 메시지를 전송한다. 여기서 T는 각 클럭에서 측정한 지역시간을 가리킨다. 그림 5에서는 클럭 C_1이 가장 먼저 재동기 시점에 도달했다고 가정하

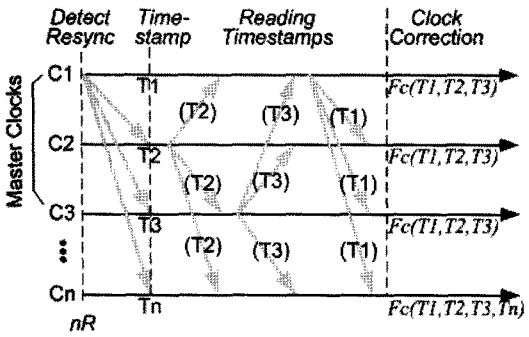


그림 5. 본 논문에서 제안한 시간동기 방법
Fig. 5. The proposed synchronization method.

였다. 동기 메시지는 특정한 데이터를 포함하지 않는 가장 짧은 메시지로 정의한다. 동기 메시지는 재동기 과정을 기동시키는 역할만 수행하며, 세 개의 마스터 클럭 가운데서 어느 클럭에 의해서 전송되더라도 동기 정밀도에 영향을 미치지 않는다.

- [단계-2] 도착시간 기록(timestamping): 마스터 클럭 뿐 아니라 시스템을 구성하는 모든 노드들이 동기 메시지가 전달되는 시점에 자신의 클럭을 기준으로 도착시간(T_2, T_3,..., T_n)을 측정한다. 동기 메시지를 보낸 클럭도 전송이 끝나는 시점에 시간(T_1)을 측정한다.
- [단계-3] 클럭 정보 교환(reading timestamps): 세 개의 마스터 클럭은 자신이 측정한 동기 메시지 도착시간을 즉시 전송한다. 시스템에 연결된 모든 노드들은 세 개의 마스터 클럭으로부터 차례대로 전송된 클럭 정보를 저장한다. 이 단계를 거친 후 각 마스터 클럭은 세 개의 클럭 정보(T_1, T_2, T_3)를 확보하게 되며, 나머지 노드들은 마스터 클럭의 시간정보와 함께 자신의 클럭 시간을 포함하여 모두 네 개의 클럭 정보를 확보하게 된다. 고장 등으로 인해서 전송되지 않은 마스터 클럭값은 0으로 가정한다.
- [단계-4] 클럭 교정(clock correction): 각 노드에서는 단계 3에서 확보한 4개(마스터 클럭은 3개)의 클럭정보를 수렴함수 Fc에 적용하여 클럭 교정량을 계산한다. 본 논문에서 사용한 수렴함수는 중간값(midpoint value)[24]을 기준시간으로 선정하도록 하였다. 각 클럭은 기준시간과 자신의 클럭값 사이의 차이를 0으로 만들기 위한 오프셋을 수정함으로써 클럭을 교정한다.

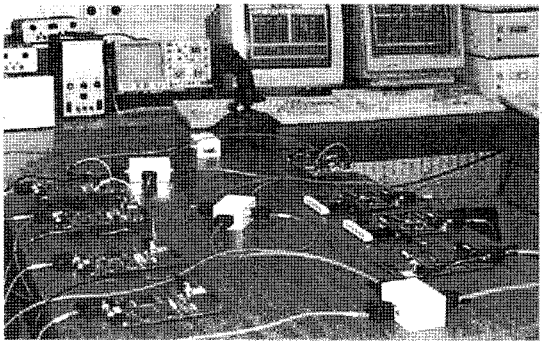
본 논문에서 제안한 동기방식은 세 개의 마스터 클럭값 가운데서 중간값을 선택하므로, 임의의 한 개 마스터 클럭에서 고장이 발생하거나 메시지가 전달되지 않더라도 전역 기준시간을 추정하는데 영향을 미치지 않고 시간동기를 유지할 수 있음을 알 수 있다. 제안한 방법은 4장에서 제시한 요구사양과 비교하여 다음과 같이 요약할 수 있다:

- 알고리즘 구조: 마스터-슬레이브 방식이므로 구조적으로 단순하며 연산이 간단하고 실제 구현과 성능분석도 간편하다. 시간동기를 위해서 전송되는 메시지는 시스템에 연결된 노드 개수와 무관하게 4개로써 일정하다.
- 비용 및 유연성: 소프트웨어 방식이므로 저렴한 비용으로 구현이 가능하다. 또한 마스터-슬레이브 구조이므로 마스터 클럭을 제외한 나머지 시스템의 변경시 시간동기 알고리즘을 수정할 필요가 없다.
- 동기 오차: 마스터-슬레이브 방식의 최대 동기오차는 식(1)과 같이 결정된다. 여기서 Gergleit & Streich[13]의 방법을 적용함으로써 전송지연으로 인한 동기오차는 CAN 버스의 1비트 시간에 불과하다. 따라서 버스 속도와 클럭의 품질, 재동기 주기 등을 적절히 조절함으로써 표 1에서 예시한 대부분의 정밀도 요구수준을 만족시킬 수 있음을 알 수 있다.
- 고장대처: 세 개의 마스터 클럭을 설정함으로써 마스터 클럭 고장에 대처할 수 있다. 비교 알고리즘이 필요치 않으므로 단순한 구조를 유지할 수 있다.
- 확정성: 동기오차 및 고장대처 능력을 설계 단계에서 예측할 수 있다.

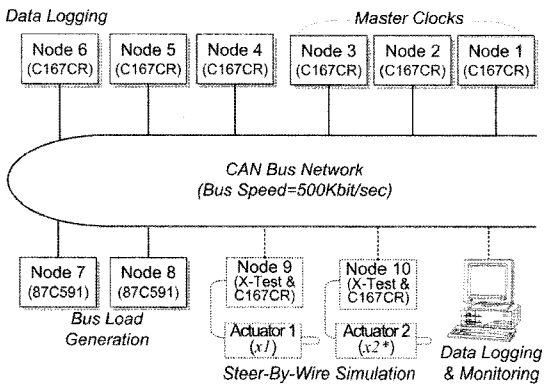
6. 실험 및 고찰

6.1. 실험장치 구성

시간동기 기술의 효용성을 보이기 위해서, (i) 메시지 전송지연 비교 실험과, (ii) 제어성능 비교실험으로 나누어서 수행한다. 이를 위해서 그림 6과 같이 8~11개의 CAN 노드로 구성된 실험환경을 설정하였다. 먼저 메시지 전송지연 측정 실험은 그림 6(a)에 보인 것처럼 8개의 마이크로컨트롤러 보드만을 이용하여 수행하였다. 이어서 시간동기가 제어성능에 미치는 영향을 분석하기 위한 실험은 그림 6(b)와 같이 CAN 기반의 steer-by-wire 모델 시스템[9]이 추가된 실험환경을 이용하였다. PC를 제외한 CAN 노드들은 모두 Infineon C167CR 또는 Philips 87C591를 이용하였다. 시간동기를 수행하는 마스터 클럭은 노드 1번~3번까지이며, 6



(a)



(b)

그림 6. 실험 장치 구성: (a) CAN 버스로 연결된 마이크로 컨트롤러 보드, (b) steer-by-wire 모델 시스템이 포함된 실험 장치 구성

Fig. 6. Experimental setup: (a) Photograph of microcontrollers interconnected through a CAN bus, and (b) experimental setup including the steer-by-wire simulator.

번 노드에서는 데이터 저장 기능을 수행한다. 그 외 엑츄에이터 제어기(노드 9번 및 10번)와 PC를 제외한 나머지 노드들은 임의의 메시지를 전송하여 버스 통신량을 90 % 수준으로 유지하는 기능을 함께 수행한다. 버스 속도는 500 Kbit/sec를 적용하였다.

6.2. 메시지 전송지연 및 마스터클럭 고장대처성능 분석

이 실험에서는 제안한 알고리즘의 고장대처 성능을 확인하고, 시간동기가 이루어진 경우와 그렇지 않은 경우에서 메시지 전송지연을 비교한다. 실험을 위해서 그림7에 나타낸 것과 같이 노드 1~3을 마스터클럭으로 설정하였으며, 재동기 주기는 R=1 sec로 하였다. 메시지 전송지연을 측정하기 위해서, 1번 노드부터 5번 노

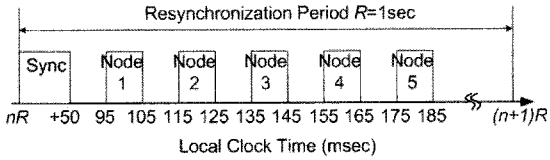


그림 7. 시간동기 정밀도 분석을 위한 시험 메시지가 전송되는 시간슬롯 스케쥴

Fig. 7. Predetermined timeslots for sending test messages to demonstrate the synchronization precision.

드까지 차례대로 정해진 시간슬롯(각 슬롯의 길이는 10 msec) 동안에 하나의 메시지를 전송하고 노드 6번에서는 각 메시지가 도착한 시간을 측정한다.

먼저 그림 8에서는 임의로 선택한 두 개의 클럭(노드 1번과 6번) 사이의 오차를 나타내었다. 클럭 오차는 1번 노드에서 전송한 메시지의 도착 예정시간($T=nR+$ 100 ms)과 실제 도착시간 사이의 오차를 6번 노드의 지역시간을 기준으로 측정하였다. 이를 위해서 1번 노드는 매 주기마다 지정된 시간슬롯에 한 개의 메시지를 전송한다. 이 시간슬롯 동안 다른 메시지는 전혀 전송되지 않으므로 메시지 충돌로 인한 전송지연은 무시할 수 있다. 1번과 6번 노드 사이의 동기오차가 충분히 작을 경우 메시지는 매 주기마다 지정된 시간 $T=nR+$ 100 msec에 도착해야 하며, 그렇지 않은 경우는 두 클럭 사이에 동기 오차가 있음을 의미한다. 그림 8에 나타낸 것처럼, 제안한 시간동기 알고리즘을 적용한 경우 예정시간에 메시지가 도착함을 알 수 있다. 그러나 t=100 sec 이후부터 시간동기 알고리즘을 실행하지 않은 경우, 메시지 도착시간이 점차 늦어짐을 알 수 있다(즉 클럭오차(C_1-C_6)가 음수). 이 때 두 클럭 사이의 드리프트 속도는 약 0.7 msec/sec임을 알 수 있다. 마스터클럭의 고장 대처능력을 확인하기 위해서, 그림 8과 같이 마스터클럭으로 설정된 노드 2와 노드 3을 각각 t=40 sec 및 t=60 sec에 차례대로 리셋시킴으로써 고장상황을 인가하였다. 그러나 이와 같은 고장 인가 시에도 고장이 발생하지 않은 나머지 2개의 마스터클럭에 의해서 시간동기가 적절히 유지됨을 확인할 수 있다.

제안한 시간 동기 알고리즘이 메시지 전송지연 개선에 효과적임을 확인하기 위해서, 그림 7에 나타낸 노드별 지정 시간슬롯 이외 시간 동안은 모든 노드들이 임의의 메시지를 연속적으로 전송함으로써 버스 사용량이 90 % 이상이 되도록 설정하고, 각 메시지 도착시간을 6번 노드에서 측정하였다. 그림 9에 나타낸 것과 같이, 시간동기가 이루어진 경우(t=100 sec 이전), 각 노

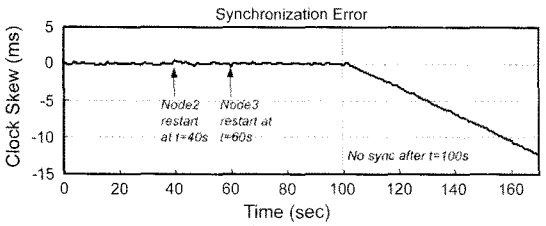


그림 8. 두 개의 클럭(노드 1번과 6번) 사이의 동기 오차 비교(마스터클럭 고장 대처 성능을 확인하기 위해서 t=40초 및 t=60초에서 노드 2 및 노드 3을 차례로 리셋함. t=100초 이후부터 제안된 시간동기 알고리즘 실행 중지)

Fig. 8. Comparison of clock skew between node 1 and node 6 (Nodes 2 & 3 are restarted to demonstrate the ability of tolerating a faulty master clock. The proposed synchronization method is disabled after t=100 sec).

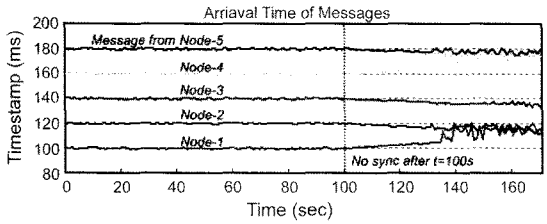


그림 9. 메시지 전송 지연 비교(t=100초 이후부터 제안된 시간동기 알고리즘 실행 중지)

Fig. 9. Comparison of message latency. The proposed synchronization method is disabled after t=100 sec.

드에 할당된 시간슬롯 동안 독점적으로 버스를 사용하여 메시지를 전송함으로써 충돌로 인한 지연이 발생하지 않으므로 정확한 시간에 메시지가 도착함을 알 수 있다. 반면에, t=100 sec 이후부터 시간동기를 실행하지 않은 경우, 각 클럭의 지역시간이 기준시간으로부터 벗어남으로써 예정시간과 다른 시간에 메시지가 도착함을 볼 수 있다. 특히 클럭 오차가 10 msec 길이의 독점 시간슬롯을 벗어난 경우, 다른 메시지와 충돌로 인한 전송지연이 발생하여 메시지 도착시간이 불규칙하게 큰 폭으로 변할 뿐 아니라, 심지어 먼저 보낸 메시지와 뒤에 보낸 메시지의 도착 순서가 바뀌는 경우까지 발생함을 확인할 수 있다.

6.3 제어 성능 비교

제어성능 비교 실험을 위해 사용된 steer-by-wire 모형 시스템은 두 개의 지능형 액츄에이터가 CAN 버스를 통해 각 스트로크의 현재 길이 정보를 서로 비교함으로써 조향각을 제어하도록 설계되었다[9]. 이 때, 각 액츄에이터 스트로크의 길이를 측정하는 시점이 서로 다르면 두 값을 비교할 때 정확한 결과를 얻을 수 없으며, 그 결과 제어성능이 저하되거나 불안정해질 수 있다.

일반적으로 디지털 제어기 구현을 위한 샘플링 주파수는 모델링 오차와 디지털화 오차 등을 고려하여 폐루프 시스템 대역폭의 20배 이상이 되도록 선택한다[25]. 이러한 기준을 적용할 경우, 본 실험에 이용된 액츄에이터 제어를 위한 샘플링 주파수는 40 Hz 이상이 되도록 설계하는 것이 바람직하다. 그러나 본 논문에서는 시간동기의 효용성, 즉 시간동기를 이용함으로써 샘플링 주파수를 낮출 수 있음을 보이기 위해서 10 Hz를 적용하였다.

먼저 그림 10은 시간동기가 잘 이루어진 상황에서 사인함수로 주어진 명령각을 추종하는 제어성능을 나타낸다. 두 액츄에이터 스트로크 사이의 최대 허용오차는 D=5 mm로 설정하였다. 그림 10(b)에 제시한 실험결과를 보면, 실제 오차가 1 mm 이내를 유지하고 있음을 알 수 있다. 즉 시간동기가 이루어진 경우 두 액츄에이터에서 거의 동일한 시점에 측정한 스트로크 길이를 이용하여 제어량을 계산함으로써 설계된 제어기가 적절히 동작함을 확인할 수 있다. 반면에, 그림 11은 시간동기 알고리즘을 적용하지 않음으로써 두 제어기

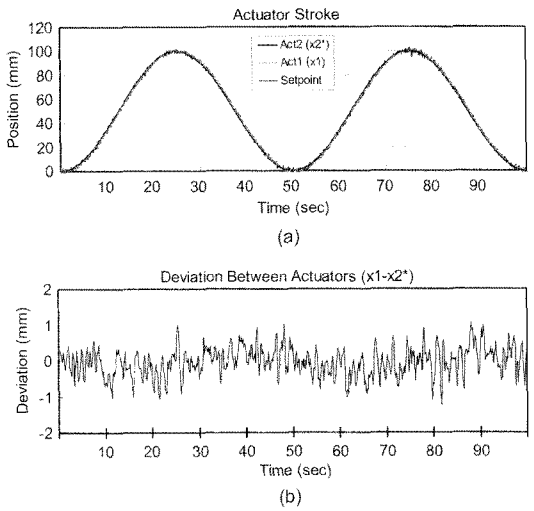


그림 10. 시간동기가 이루어진 상태에서 steer-by-wire 액츄에이터 제어 성능: (a) 액츄에이터 스트로크 x_1, x_2^*, (b) 두 액츄에이터 스트로크 사이 편차

Fig. 10. Control performance of the steer-by-wire actuators with synchronization clocks: (a) Actuator stroke trajectories, x_1, x_2^*, (b) Deviation between two strokes.

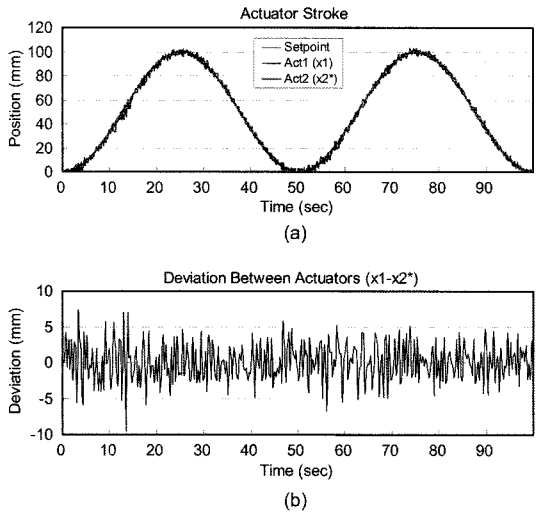


그림 11. 시간동기가 이루어지지 않은 상태에서 steer-by-wire 엑츄에이터 제어 성능: (a) 엑츄에이터 스트로크 x_1, x_2^*, (b) 두 엑츄에이터 스트로크 사이 편차

Fig. 11. Control performance of the steer-by-wire actuators without synchronization: (a) Actuator stroke trajectories, x_1, x_2^*, (b) Deviation between two strokes.

를 포함한 모든 CAN 노드들의 클럭들이 동기를 잃고 기준시간에서 벗어난 가운데 제어를 수행한 결과를 나타낸다. 이 경우 최대 허용오차(5 mm)를 벗어날 뿐만 아니라, 엑츄에이터의 움직임 빈도가 상대적으로 더 많음으로써 과도한 전원소모와 엑츄에이터 손상을 유발할 수 있음을 알 수 있다. 이는 각 엑츄에이터 스트로크의 길이를 측정하는 시점이 서로 다를 뿐 아니라, 메시지 충돌 때문에 발생한 불규칙한 시간지연으로 인해서 스트로크의 현재 길이를 정확하게 비교하지 못해서 발생한 문제이다. 즉 동기 오차로 인해서 두 엑츄에이터의 측정시점은 샘플링 주기의 2배에 해당하는 200 ms까지 차이가 날 수 있으며, 여기에 메시지 충돌로 인한 전송지연까지 추가됨으로써 제어성능이 저하되었음을 볼 수 있다. 시간동기 기술을 이용하지 않고 이 문제를 해결하기 위해서는 보다 높은 샘플링 주파수를 적용하거나 관측기 등 복잡한 제어 알고리즘을 이용하여야 한다. 즉 시간동기가 이루어진 시스템에서는 보다 저렴한 마이크로프로세서를 사용하더라도 충분히 원하는 성능을 만족할 수 있으나, 그렇지 않은 경우 보다 빠른 연산을 수행할 수 있도록 고가의 프로세서를 적용해야 함을 확인할 수 있다. 뿐만 아니라, 고비용 프로세서 사용은 곧 제품비용, 전원소모, 중량, 복잡성, 신뢰도 등에 직접적으로 영향을 미치기 때문에 시스템 구현시 큰 제약이 될 수 있다.

7. 결 론

센서와 엑츄에이터 및 제어컴퓨터 등이 필드버스로 연결된 네트워크 기반 임베디드 제어시스템에 있어서 정밀하고 신뢰도가 높은 시간동기 기술의 중요성과 효용성을 살펴보고, CAN 기반 임베디드 제어시스템을 위한 시간동기 알고리즘을 제안하였다. 제안된 시간동기 알고리즘은 임베디드 제어시스템을 위한 시간동기 기술의 요구사양을 만족할 수 있도록 구현 용이성, 최대 동기오차, 확정성, 고장대처 능력 등을 고려하여 설계하였다. 즉 저비용 프로세서와 필드버스 기반의 임베디드 시스템에 적합하도록 마스터-슬레이브 구조를 갖는 소프트웨어 방식으로 설계하였으며, 고장에 대처할 수 있도록 세 개의 마스터 클럭을 사용하였다. 그리고 Gergleit & Streich[13]가 제안한 방법을 적용하여 소프트웨어 기반 동기 방식의 단점인 낮은 정밀도를 개선하였다. 제안된 시간동기 알고리즘은 마이크로컨트롤러와 차세대 자동차용 steer-by-wire 모델 시스템으로 구성된 실험장치에 적용하여 그 성능과 효용성을 분석하였다. 그 결과, 제안한 방법을 이용함으로써 저비용 임베디드 시스템의 메시지 전송지연을 최소값으로 유지할 수 있을 뿐 아니라, 단순한 비례제어 알고리즘과 상대적으로 낮은 샘플링 주파수를 적용하더라도 제어성능을 만족시킬 수 있음을 확인하였다.

본 논문에서 제안한 시간 동기 방법은 CAN 버스의 독특한 다중전송(atomic broadcast) 특성을 이용하고 있으나, Ethernet 등에도 적용할 수 있도록 수정보완하는 연구를 진행 중이다. 아울러 마스터 클럭의 고장대처 요구 수준에 맞추어 시간동기 알고리즘을 유연하게 설계할 수 있는 방법에 관한 연구도 필요하다.

감사의 글

본 논문은 2단계 BK21 정보기술연구인력 양성사업단의 연구비에 의하여 연구되었음.

참고 문헌

[1] P. L. Shaffer, "Distributed control system for turbine engines", *Journal of Engineering for Gas Turbines and Power*, vol. 121, pp. 102-107, January 1999.

[2] G. Leen and D. Heffernan, "Expanding automotive electronic systems", *IEEE Computers*, vol. 35, no. 1, pp. 88-93, January 2002.

[3] J. Nilsson, "Real-time control systems with delays", *PhD Thesis*, Lund Institute of Technology, Sweden, 1998.

[4] T. C. Yang, "Networked control system: A brief survey", *IEE Proc. Control Theory Appl.*, vol. 153, no. 4, pp. 403-412, 2006.

[5] Y. Tipsuwan and M. Y. Chow, "Control methodologies in networked control systems", *Control Engineering Practice*, vol. 11, pp. 1099-1111, 2003.

[6] R. Luck and A. Ray, "An observer-based compensator for distributed delays", *Automatica*, vol. 26, no. 5, pp. 903-908, 1990.

[7] H. Chan and U. Ozguner, "Closed-loop control of systems over a communications network with queues", *International Journal of Control*, vol. 62, no. 3, pp. 493-510, 1995.

[8] H. Kopetz, *Real-time systems: Design principles for distributed embedded applications*, Kluwer, 1997.

[9] 이동익, "네트워크 기반 steer-by-wire 시스템을 위한 지능형 엑츄에이터 제어", 센서학회지, 제15권, 제6호, 2006.

[10] F. B. Schneider, "A paradigm for reliable clock synchronization", *Tech. Report TR-86-735*, Computer Science Dept., Cornell University, Ithaca, NY, USA, 1986.

[11] P. Ramanathan, K. G. Shin, and R. W. Butler, "Fault-tolerant clock synchronization in distributed systems", *IEEE Computer*, vol. 23, no. 10, pp. 33-42, 1990.

[12] E. Anceaume and I. Puaut, "Performance evaluation of clock synchronization algorithms", *Tech. Report N3526*, Unite de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, France, 1998.

[13] M. Gergeleit and H. Streich, "Implementing a distributed high-resolution real-time clock using the CAN bus", *Proc. 1st CiA International CAN Conference*, 1994.

[14] A. V. Schedl, "Design and simulation of clock synchronization in distributed systems", *PhD Thesis*, Technical University of Vienna, Austria, 1996.

[15] J. M. Rushby and F. Henke, "Formal verification of algorithms for critical systems", *IEEE Trans. Software Engineering*, vol. 19, no. 1, pp. 13-23, 1993.

[16] TTTech Computertechnik GmbH, *Time-triggered protocol TTP/C, High-Level Specification Document*, Available: http://www.tttech.com.

[17] *FlexRay Requirement Specification*, ver 2.0.2, April 2002, Available: http://www.flexray.com

[18] J. Eidson and W. Cole, "Ethernet rules closed-loop system", *InTech*, pp. 39-42, June 1998.

[19] P. Koopman, "Embedded system design issuesthe rest of the story", *Proc. International Conference on Computer Design*, Austin, USA, 1996.

[20] IEEE-1588, "Standard for precision clock synchronization protocol for networked measurement and control systems", 2002. Available: http://ieee1588.nist.gov/.

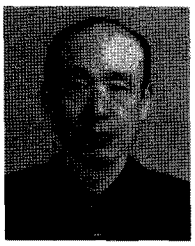
[21] J. C. Eidson and B. Hamilton, "IEEE-1588 node synchronization improvement by high stability oscillators", *Proc. Workshop on IEEE 1588 Standard*, Gaithersburg, USA, 2003.

[22] R. Gusella and S. Zatti, "An election algorithm for a distributed clock synchronization program", *Tech. Report CSD-86-275*, Computer Science Division, University of California, Berkeley, USA, 1985.

[23] L. Rodrigues, M. Guimaraes, and J. Rufino, "Fault-tolerant clock synchronization in CAN", *Proc. IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.

[24] J. Lundelius and N. Lynch, "A new fault-tolerant algorithm for clock synchronization", *Information and Computation*, vol. 77, pp. 1-36, 1988.

[25] G. F. Franklin, J. D. Powel, and A. Emami-Naeini, *Feedback control of dynamic systems (5th Ed.)*, Prentice Hall, 2006.



이 동 익

- 1987년 8월 경북대학교 전자공학과(공학사)
- 1990년 2월 경북대학교 대학원 전자공학과 (공학석사)
- 1990년 3월~1997년 8월 국방과학연구소 연구원
- 1997년 9월~2002년 4월 영국 셰필드대학교 자동제어시스템공학과(공학박사)
- 2002년 1월~2005년 3월 영국 DRTS Ltd 공동설립 및 CTO
- 2005년 4월~현재 경북대학교 전자전기컴퓨터학부 전임강사