저작자표시-비영리-변경금지 2.0 대한민국

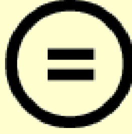
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

Read-Copy-Update 메커니즘을 이용한 AUTOSAR IOC의 통신 지연 개선

Reducing the Communication Latency of IOC in AUTOSAR using Read-Copy-Update Mechanism

2017 년 2 월

서울대학교 융합과학기술대학원

융합과학부 지능형융합시스템전공

김 학 범

Read-Copy-Update 메커니즘을 이용한 AUTOSAR IOC의 통신 지연 개선

Reducing the Communication Latency of IOC in AUTOSAR using Read-Copy-Update Mechanism

지도 교수 홍 성 수

이 논문을 공학석사 학위논문으로 제출함
2017 년 1 월

서울대학교 융합과학기술대학원
융합과학부 지능형융합시스템전공
김 학 범

김학범의 공학석사 학위논문을 인준함
2017 년 1 월

위 원 장 ___박 재 홍___ (인)

부위원장 ___홍 성 수___ (인)

위　　원 ___곽 노 준___ (인)

초 록

최근 자동차 산업에서 자율주행차량, 친환경 차량 등의 등장으로 전기 전자 분야에 대한 중요성이 대두되고 있다. 이에 따라 차량 내의 전기 전자 시스템은 늘어나고 이를 제어 하기 위한 소프트웨어 양과 복잡도 역시 함께 증가하고 있다. 이렇게 거대해지고 복잡한 차량용 소프트웨어를 효과적으로 개발하기 위해 고안이 된 것이 차량용 소프트웨어 플랫폼인 AUTOSAR(AUTomotive Open Software Architecture) 이다. 특히, AUTOSAR v4.1.1부터는 시스템의 성능을 높이고 기능안전을 만족하기 위해 멀티코어를 지원한다. 하지만, AUTOSAR에서 지원하는 core간의 통신방식인 IOC(Inter OS-application communicator)에서는 예기치 못한 통신 지연이 발생하여 전체 시스템의 성능이 저하되는 문제를 초래한다. 이 논문에서는 IOC의 통신 지연문제를 밝히고 Read-Copy-Update 메커니즘을 이용하여 이를 해결하고자 한다.

주요어 : AUTOSAR, IOC, RCU, OS-application, RTE
학 번 : 2015-26043

목 차

제 1 장 서 론 1

제 2 장 배 경 6
제 1 절 AUTOSAR 개요 6
제 2 절 AUTOSAR 통신 메커니즘 9
제 3 절 Read-Copy-Update 15

제 3 장 문제 정의 19
제 1 절 시스템 모델 19
제 2 절 IOC의 한계점 20

제 4 장 개선된 AUTOSAR의 IOC 통신 26
제 1 절 RCU를 적용한 개선된 IOC 통신 26
제 2 절 개선된 IOC 통신 구현 27

제 5 장 실험 및 검증 30
제 1 절 실험 환경 30
제 2 절 실험 구성 31
제 3 절 실험 평가 33

제 6 장 결 론 38

참고문헌 39

Abstract 40

표 목차

[표 1] Linux에서 사용되는 RCU API Family 18
[표 2] RCU Library 구현.. 28
[표 3] RTE 코드에서 RCU API 호출을 통한 구현 29
[표 4] 실험 환경 구축을 위한 하드웨어와 소프트웨어 상세........... 30
[표 5] Task 수행 시간 측정을 위한 코드 구현 32
[표 6] 실험 결과 비교 (기존 시스템과 개선된 시스템) 34

그림 목차

[그림 1] AUTOSAR 협력 파트너 7
[그림 2] AUTOSAR 표준의 구성 7
[그림 3] AUTOSAR의 계층적 구조 8
[그림 4] 포트를 이용한 AUTOSAR 통신 9
[그림 5] Sender-Receiver 인터페이스 10
[그림 6] Client-Server인터페이스 10
[그림 7] Parameter 인터페이스 11
[그림 8] Non Volatile Data 인터페이스 11
[그림 9] 포트와 인터페이스를 이용한 소프트웨어 컴포넌트간 통신 11
[그림 10] AUTOSAR의 3가지 통신 경로 12
[그림 11] OS-Application 내에서의 통신 13
[그림 12] 서로 다른 OS-Application간의 통신 13
[그림 13] IOC에서 Sender-Receiver 통신 14
[그림 14] IOC에서 Client-Server 통신 14
[그림 15] 서로 다른 ECU간의 통신 15
[그림 16] RCU를 이용한 데이터 Read/Write 동작 과정 17
[그림 17] RCU의 Grace Period 18
[그림 18] Runtime Environment 20
[그림 19] 전자 브레이크 시스템에서의 통신 경로 25
[그림 20] 레이더 모듈에서의 통신 지연 23
[그림 21] 브레이크 모듈에서의 통신 지연 24
[그림 22] AUTOSAR에서의 1: N 통신 25
[그림 23] Spinlock과 RCU 비교 27
[그림 24] 브레이크 시스템 구성도 31
[그림 25] 실험 평가 모델 32
[그림 26] 변수 계측 장비 CANape 33
[그림 27] Read Access Tasks의 수와 개선율의 상관관계 34
[그림 28] Task 수행시간 결과 (Read access Tasks 수: 5) 35
[그림 29] Task 수행시간 결과 (Read access Tasks 수: 6) 35
[그림 30] Task 수행시간 결과 (Read access Tasks 수: 7) 35
[그림 31] Task 수행시간 결과 (Read access Tasks 수: 8) 36
[그림 32] Task 수행시간 결과 (Read access Tasks 수: 9) 36
[그림 33] Task 수행시간 결과 (Read access Tasks 수: 10) 36
[그림 34] 기존 시스템에서 CAN 데이터 전송 주기 측정 37
[그림 35] 개선된 시스템에서 CAN 데이터 전송 주기 측정 37

제 1 장 서 론

자동차 산업에서는 전기 자동차와 자율 주행 자동차의 개발이 한창이다. 차량의 배기가스로 인한 온실효과, 미세먼지 등 환경문제가 심각한 사회 문제로 대두되고 있다. 유럽에서는 CO2 규제량을 2015년 130g/km에서 2021년 95g/km 로 5.1%의 개선을 목표로 하고 있고 미국 4.3%, 중국 5.2%, 일본 3.3%로 CO2 배출을 줄이는 것을 목표로 하고 있다. 이를 달성하기 위해서는 전기차 도입이 필수 적이다.

또한 폭스바겐 디젤스캔들 이슈로 인하여 디젤차량의 규제가 강화되고 있어 많은 OEM들은 전기차 도입 확대 및 차별화된 전기 자동차 출시 등 전기 자동차 중심의 전략 변경 움직임을 보이고 있다. 이러한 자동차 산업의 변화는 전기 자동차 판매 실적에도 직접적인 영향을 미치고 있다. 2015년 주요 국가별 전기 자동차와 플러그인 하이브리드 차량의 판매 실적을 살펴 보면, 중국 250%, 유럽 90%, 미국 3%, 기타 15% 전년대비 증가하였다. 앞으로의 전기 자동차 시장 전망을 살펴 보면, 2015년 55만대에서 2020년 까지 289만대로 확대될 것으로 예상된다.

또한, BMW, Mersedes Benz와 같은 세계 자동차의 선두 업체들은 인공지능을 기반으로한 자율주행차량을 개발하고 있다. 2025년까지 완전한 자율주행 자동차를 목표로 개발 중이고 이미 1단계인 부분 자율주행 기술이 양산 적용된 차량이 도로를 누비고 있다.

특히, 핀란드 헬싱키에서는 일반도로에서 자율주행 미니버스를 시험적으로 주행한다. 또한 싱가포르에서는 일반인 대상으로 자율주행 택시를 시범적으로 주행하고 있다. 우리나라에서도 자율주행차의 8대

핵심 부품 개발을 국가과제로 선정하여 2025년 까지 선진국 수준을 목표로 진행하고 있다. 여기서 자율 주행차의 핵심 부품은 카메라, 레이더, V2X, 복합측위, 디지털맵, 차량-운전자 인터페이스, 자동주행기록장치, 통합 제어장치 이다.

앞에서 설명한 전기자동차와 자율주행자동차의 공통 핵심 기술은 소프트웨어를 기반으로 한 전자제어장치(Electronic Control Unit)이다. 다수의 전자제어장치가 Network로 연결되어 각 제어기에 필요한 데이터를 주고 받으며 전기자동차와 자율주행자동차의 기술을 완성시킨다. 이렇듯 이제 차량 기술의 핵심은 전자제어장치이다. 차량용 전자제어 장치의 종류는 ESC (Electronic Stability Control), ECM (Engine Control Unit), TCM (Transmission Control Module), BCM (Body Control Module), LKAS (Line Keeping Assistance System), SPAS (Smart Parking Assistance System), AEB(Automatic Emergency Brake) 등이 있다. 각 전자제어 장치는 각기 다른 역할을 수행하지만, 공통적으로 하드웨어와 소프트웨어로 구성된다. 하드웨어는 MCU (Micro Controller Unit), Sensor, 통신 모듈, IC를 포함한 ECU와 Actuator로 구성된다. 또한 소프트웨어를 통해 하드웨어의 동작을 제어한다.

차량 내, 전자제어 장치가 늘어나고 이를 제어 하기 위한 소프트웨어의 양과 복잡도 또한 기하급수적으로 늘어나고 있다. 이에 따라 운전자의 안전과 편의성을 위한 소프트웨어가 의도하지 않은 오동작으로 오히려 운전자를 위험하게 만드는 상황에 이르고 있다. 대표적인 사례가 바로 도요타의 엔진 ECU 소프트웨어의 결함으로 인한 급발진 사건이다. 이런 소프트웨어 결함으로 인한 시스템의 오작동이 빈번하게 발생하게 되고 이에 대한 대책으로 차량용 소프트웨어의

표준화를 하고자 하는 움직임이 생겨났다. 2005년에 BMW, 다임러, 보쉬, 콘티넨탈 등의 자동차 완성체 업체와 부품업체가 협력하여 개방형 자동차 표준 소프트웨어 구조인 AUTOSAR (AUTomotive Open System Architecture)를 최초로 개발하여 배포하였다. 그때부터 현재까지 AUTOSAR의 버전이 갱신되고 있으며, AUTOSAR를 기반으로 하여 전기전자 시스템이 개발되고 있다. AUTOSAR는 소프트웨어를 여러 개의 계층화된 소프트웨어로 나누고 각 계층의 인터페이스 영역의 경계 표준화를 통해 구조화된 소프트웨어 플랫폼을 제공하고 이를 통해 소프트웨어의 복잡도를 해결하고 신뢰성을 높이고 있다. 또한 하드웨어에 의존적인 Basic Software와 시스템의 기능에 의존적인 Application Software를 구분함으로써 개발된 소프트웨어가 하드웨어에 유연한 통합, 전이, 교체가 이뤄진다.

AUTOSAR 버전 v4.1.1 이후부터는 Multi-core를 지원하는 기능들이 추가되었다. 차량용 소프트웨어에서 Multi-core 의 도입은 시스템 성능을 높이고 안전성을 확보하기 위함이다. Multi-core의 시스템에서는 양이 많고 복잡한 소프트웨어의 수행시간을 단축시켜서 시스템 전체 성능을 향상시키게 된다. 또한 차량 안전 규제인 ISO26262에서는 차량용 전기전자의 시스템의 안전한 설계를 위해서 Multi-core를 이용한 이중 설계를 요구하고 있다. 이러한 요구사항을 만족하기 위해서 AUTOSAR에서는 Multi-core OS 개념을 도입한다. Multi-core OS에서는 Core간의 메모리 분리를 위해서 OS-Application을 사용한다. OS Application이란, OS의 objects인 task, isr, alarm, schedule table, counter의 집합을 의미한다. AUTOSAR에서는 OS-Application을 Core간의 메모리 영역을 나누는데 사용한다. 하나의 Core에서 동작하는 소프트웨어 컴포넌트는 공유된 메모리를 통해

데이터를 공유하기 때문에 자원의 충돌 문제가 발생하지 않는다. 하지만 서로 다른 Core간의 데이터를 공유하기 위해서는 메모리 영역을 나눠서 사용하기 때문에 직접적으로 메모리에 접근해서 데이터를 주고 받을 수 없다. 따라서 AUTOSAR에서는 IOC (Inter OS-Application Communication)을 통해 core간 데이터를 교환할 수 있다. IOC를 이용하여 데이터를 송수신 할 때, IOC 버퍼에 데이터를 저장하고 읽는다. IOC 버퍼에 접근할 때, 동시에 두 개 이상의 Task 가 접근하게 되면 데이터 왜곡이 발생할 수 있다. 이를 보호하기 위해 동기화 메커니즘인 Spinlock을 사용한다. Spinlock은 한 Task가 데이터에 접근 할 때, 다른 Task의 접근을 차단해 데이터 동기화를 성공하지만, 접근이 차단되는 경우 그 시간 동안 spinning을 하며 지연된다. 이런 spinning으로 인한 지연 때문에 Single-core에서 보다 Multi-core에서 시스템 전체 성능이 떨어지는 경우도 발생하고 있다. 실제 Multi-core AUTOSAR를 기반으로 개발 중인 전자식 브레이크 시스템의 메인 Task의 수행 시간이 Single-core 기반일 때보다 더 길어지는 현상이 발생하였다. 이는 하나의 데이터를 여러 Task들이 동시에 접근하는 경우가 빈번하고 이때마다 spinning으로 인한 전달 지연이 발생하였고 이로 인해 전체 시스템의 성능이 떨어지는 결과를 초래하였다. 이 문제를 해결하기 위해 본 논문에서는 RCU (Read-Copy-Update)라는 동기화 메커니즘을 이용하고자 한다. RCU는 Read access를 할 때, waiting free여서 오버헤드가 적다. 또한 Read access하는 Task가 많을수록 좋은 성능을 발휘한다. 자동차 시스템은 하나의 데이터를 여러 Task가 공유하는 경우가 빈번하여 RCU를 이용한 동기화 메커니즘을 적용하기에 적합하다. 본 논문에서는 Multi-core AUTOSAR의 IOC를 통한 core간 통신에서 발생할 수 있는 통신 지연 현상을 밝히고 RCU를

적용해서 이 문제를 해결하고자 한다.

제 2 장 배 경

이 장에서는 본 논문의 이해를 돕기 위한 배경 지식을 설명한다. 제 1절에서는 AUTOSAR의 개요에 대해서 설명하고 2절에서는 AUTOSAR의 통신 메커니즘을 설명한다. 3절에서는 Read-Copy-Update를 설명한다.

제 1절 AUTOSAR 개요

AUTOSAR는 자동차 제조 회사, 자동차 부품 회사, 차량용 소프트웨어 회사와 반도체 회사 등 자동차 산업과 관련한 업체가 참여하여 만든 차량용 소프트웨어 플랫폼 표준이다. AUTOSAR는 그림1과 같이 주요 완성차 업체 및 주요 부품회사로 이뤄진 core partners와 기타 완성타 업체 및 자동차 부품 회사, 반도체 회사로 이뤄진 premium partners와 기타 업체로 이뤄진 Associate partners의 협력으로 2002년 8월 최초 배포되었고 지금까지 개발되고 있다. 이렇게 많은 자동차 업체들이 AUTOSAR 개발에 참여하고 있는 이유는 차량용 소프트웨어의 개발, 실행, 관리를 용이하게 제공하는 플랫폼을 개발함으로써, 차량용 전기 전자 기능 증가에 따른 복잡성 관리와 품질 및 신뢰성을 향상 시키는데 있다. 자동차 경쟁 업체들은 플랫폼에서 협력하고 그 밖에서 경쟁하자는 취지로 AUTOSAR 컨소시엄에 참여하고 있다.

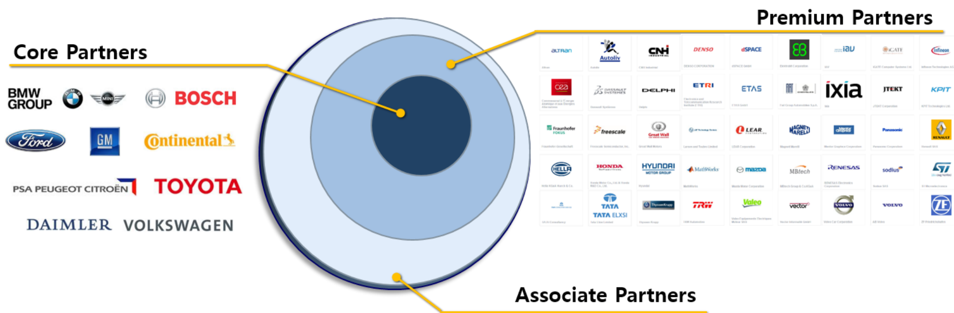


그림1. AUTOSAR 협력 파트너

AUTOSAR를 이루는 표준은 그림 2과 같이 세 가지로 나눌 수 있다. 첫째, 기본 소프트웨어(Basic Software, BSW)와 런타임 환경(Run Time Environment, RTE)에 대한 명세를 포함하는 소프트웨어 아키텍처(Software Architecture), 둘째, 소프트웨어 개발 방법론과 각 개발 방법에 사용되는 입력 문서와 출력 문서의 양식을 정의한 방법론과 템플릿(Methodology and Templates), 다양한 차량용 응용 소프트웨어를 AUTOSAR 기반으로 구현 시, 참조 모델을 정의한 응용 인터페이스(Application Interfaces)가 있다.

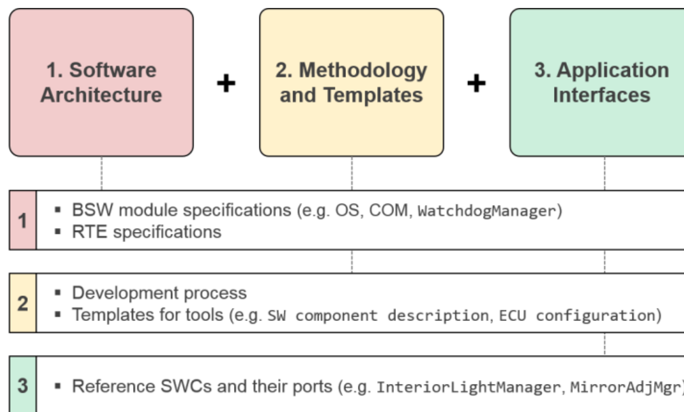


그림2. AUTOSAR 표준의 구성

AUTOSAR는 아래 그림3에서 보는 것과 같이 계층화된 구조로 되어 있고, 각 계층은 가장 하위 단인 Microcontroller에서 가장 상위 단인 Application layer까지 존재한다.

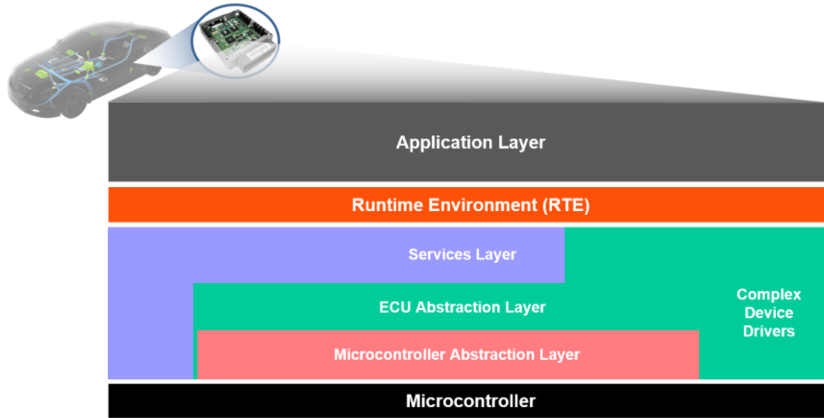


그림3. AUTOSAR의 계층적 구조

각 계층을 하위 단부터 설명 하고자 한다.

우선, Microcontroller Abstraction Layer (MCAL)은 Basic Software (BSW) 중 가장 하위 단의 계층으로 소프트웨어 모듈이 직접 μC 또는 내부 peripheral에 접근할 수 있도록 하는 내부 Driver를 담고 있다. 이를 통해 μC과 독립적인 상위의 소프트웨어 Layer를 제공한다.

둘째, ECU Abstraction Layer이다. 이는 MCAL위에 존재하며 외부 Flash 메모리, Watchdog에 대한 Driver를 제공한다. 또한 해당 Peripheral과 Driver에 접근할 수 있는 표준화된 API를 제공한다. 이를 통해 외부 ECU 하드웨어로부터 독립적인 상위 소프트웨어 Layer를 제공한다.

셋째, Complex Device Drivers (CDD)은 AUTOSAR 인터페이스를 통해 접근하거나 접근될 수 있는 표준화된 소프트웨어가 아닌 소프트웨어 Entity이다. AUTOSAR에서 규정하지 않은 특정한 목적의 기능을 가진 소프트웨어를 통합시키는 역할을 한다.

넷째, Services Layer는 BSW의 가장 상위 Layer로써, OS의 기능, 네트워크 통신, Management 서비스, Diagnostic 서비스, ECU 상태 관리 등을 제공한다. RTE, BSW, Application 소프트웨어에 이와 같은 서비스를 제공한다.

다섯째, Run Time Environment (RTE)는 Application 소프트웨어에게 통신 서비스를 제공한다. AUTOSAR의 소프트웨어 컴포넌트는 RTE를 통해 다른 소프트웨어 컴포넌트와 통신을 한다. 특정 ECU 하드웨어로부터 독립적인 소프트웨어가 될 수 있도록 한다.

마지막으로 Application Layer는 응용 소프트웨어 컴포넌트로 구성되고 시스템의 기능을 제공한다.

제 2절 AUTOSAR 통신 메커니즘

제1절에서 AUTOSAR에서 통신을 담당하는 것은 RTE라고 설명했다. 이 절에서는 AUTOSAR의 통신 메커니즘을 좀더 자세히 설명하고자 한다. 그림 4에서 보는 것처럼 소프트웨어 컴포넌트는 다른 소프트웨어 컴포넌트 또는 BSW와 Ports를 통해 통신을 한다. 여기서 각 Port들은 Virtual Functional Bus를 통해 연결된다. Virtual Functional Bus (VFB)는 상호 연결된 컴포넌트 간의 데이터 및 서비스 교환을 가능하게 하는 통신 메커니즘이다. RTE는 VFB의 인터페이스를 특정 ECU내에서 구현한 통신 메커니즘이다.

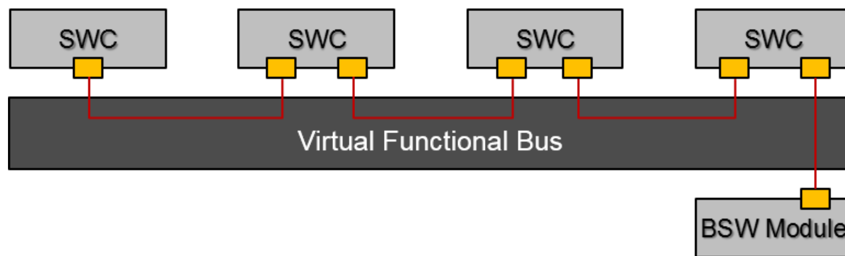


그림4. 포트를 이용한 AUTOSAR 통신

여기서 Port란, 데이터가 소프트웨어 컴포넌트에 들어오고 나갈 수 있는 부분을 뜻한다. Port에는 세가지 타입이 있다. PPort (provide port), RPort (require port), PRPort (provide-require port) 이다. Port간의 연결을 하기 위해서는 어떤 인터페이스로 연결해야 할지를 결정해야 하는데,

AUTOSAR에서 제공하는 인터페이스의 종류는 다음과 같다. Sender-Receiver, Client-Server, Parameter, Non Volatile Data, Trigger, Mode Switch 이다. 각 인터페이스에 대해서 자세히 살펴보겠다.

우선, Sender-Receiver interface는 VFB를 통해 송신/수신 할 수 있는 데이터 요소의 집합이다. 1:n, m:1의 Sender-Receiver 통신을 지원한다. 이 인터페이스를 사용하기 위해서는 아래 그림5와 같이 해당 아이콘의 Port를 이용하여 소프트웨어 컴포넌트를 연결하면 된다.

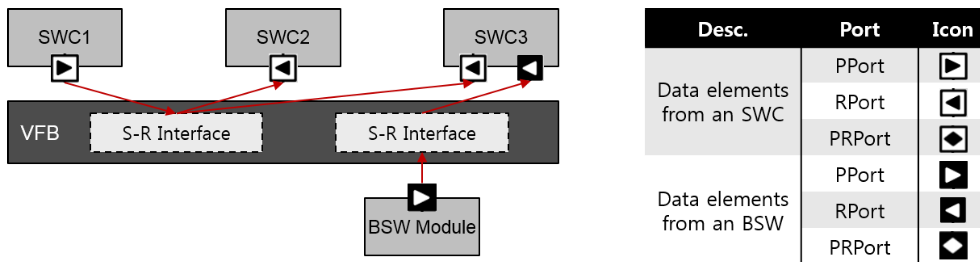


그림5. Sender-Receiver 인터페이스

둘째, Client-Server interface는 Server에서 제공되고 Client에서 호출되는 기능의 집합이다. 이를 통해 Server 측에서 제공하는 기능의 함수를 Client 에서 호출해서 사용할 수 있다. 아래 그림6와 같이 해당 아이콘의 Port를 이용하여 소프트웨어 컴포넌트를 연결하면 된다.

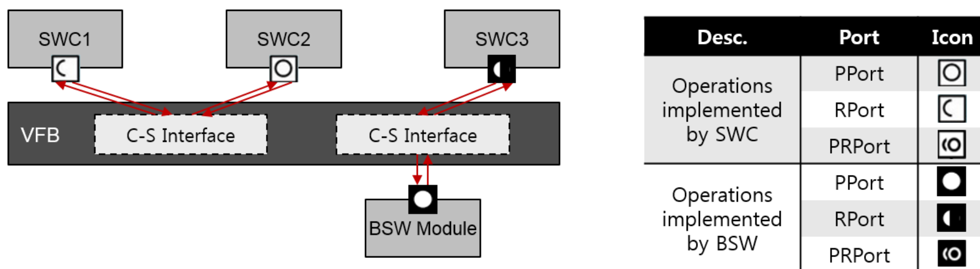


그림 6. Client-Server 인터페이스

셋째, Parameter interface는 *ParameterSwComponentType* 컴포넌트를 제공함으로써 컴포넌트가 Calibration data, Constant data에 접근할 수 있게 한다. 그림 7에서의 아이콘의 Port를 사용해서 연결하면 된다.

Desc.	Port	Icon
Parameters from an SWC	PPort	
	RPort	
Parameters from an BSW	PPort	
	RPort	

그림7. Parameter 인터페이스

넷째, Non Volatile Data interface는 *NvBlockBwComponentType*을 이용하여 컴포넌트가 Rom 영역에 데이터를 저장하거나, 읽어들일때 사용한다. 그림 8에서의 아이콘의 Port를 사용해서 연결하면 된다.

Desc.	Port	Icon
Data elements provided by SWC	PPort	
	RPort	
	PRPort	
Data elements provided by BSW	PPort	
	RPort	
	PRPort	

그림8. Non Volatile Data 인터페이스

그 외에도 컴포넌트 또는 BSW의 상태를 전달하는 Mode Switch interface가 있고, 한 컴포넌트가 다른 컴포넌트의 수행을 시작하게 하는 Trigger interface가 있다.

위에서 설명한, Port와 Interface의 조합으로 소프트웨어 컴포넌트간의 통신이 이뤄진다. 아래 그림9는 두 개의 ECU에서 서로 다른 소프트웨어 컴포넌트간에 Port와 Interface로 연결되어 통신을 하는 예이다.

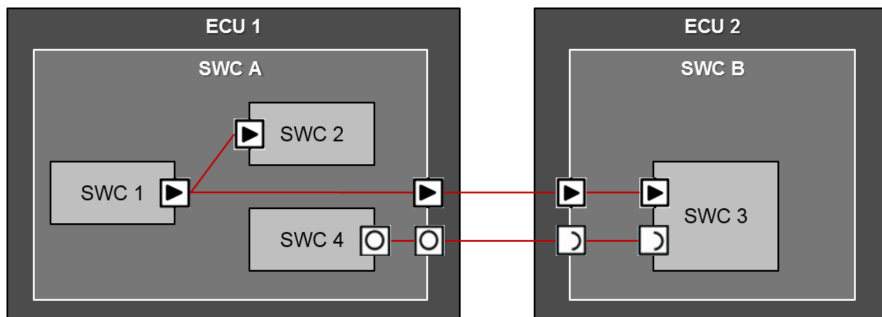
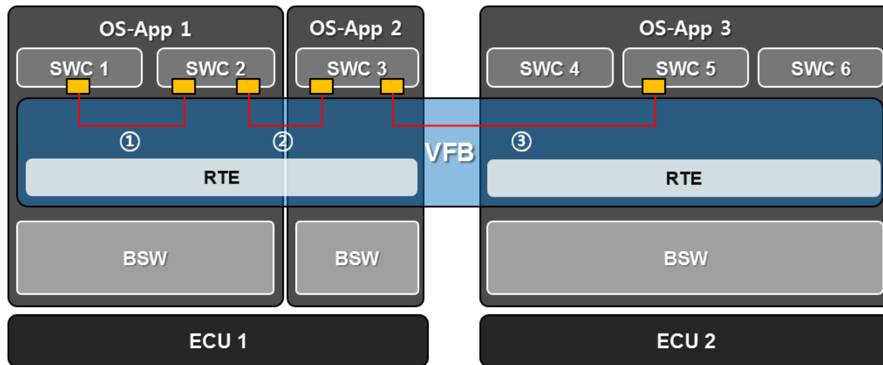


그림9. 포트와 인터페이스를 이용한 소프트웨어 컴포넌트간 통신

다음으로는 AUTOSAR내에서의 통신 경로에 대해서 설명한다. 아래 그림10에서 보는 것과 같이 세가지 통신 경로가 존재한다. 여기서 OS-Application이란, 상호간에 동작하는 유닛을 형성하기 위한 OS의 objects인 task, isr, alarm, schedule table, counter의 집합을 의미한다. AUTOSAR에서는 OS-Application을 Core간의 메모리 영역을 나누는데 사용한다. 즉, 서로 다른 Core는 서로 다른 OS-Application을 할당 받아 사용한다. 하지만 한 Core에서 설계자의 의도에 의해 다수의 OS-Application으로 나눠서 사용할 수 있다.



① OS-Application 내에서의 통신

② 서로 다른 OS-Application간의 통신

③ 서로 다른 ECU간의 통신

그림10. AUTOSAR의 3가지 통신 경로

우선, OS-Application 내에서의 통신을 살펴보면 그림 11 처럼 소프트웨어 컴포넌트들이 같은 메모리 영역을 공유하기 때문에 자원공유 충돌이 없다. 공유된 메모리를 통해 Sender-Receiver interface로 데이터를 송수신 할 수 있고, Client-Server interface로 Operation을 제공하고 받을 수 있다.

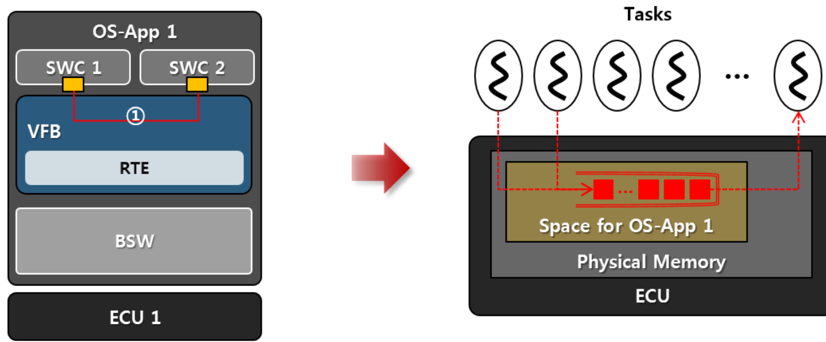


그림11. OS-Application 내에서의 통신

둘째, 서로 다른 OS-Application 간의 통신이다. 이를 위해 AUTOSAR에서는 IOC (Inter OS-Application Communication)을 제공한다. IOC는 OS-Application간의 통신이나, Core의 영역이나 Memory Protection 영역을 넘나드는 통신을 위한 서비스를 제공한다.

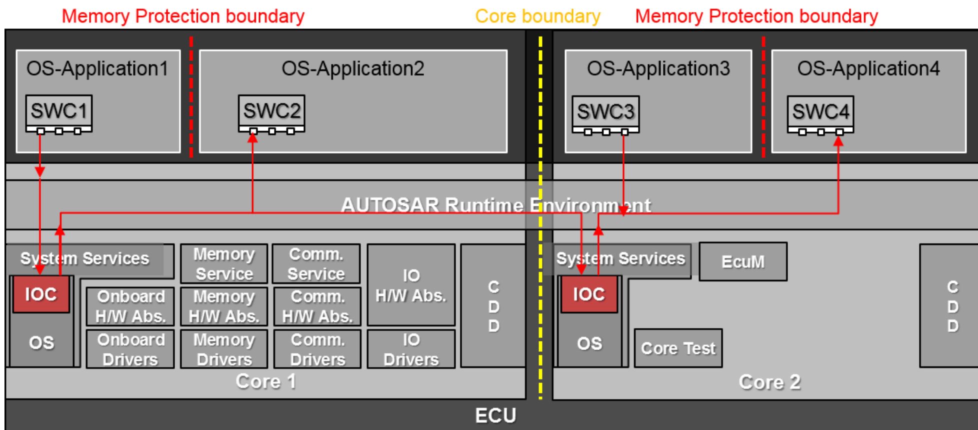


그림 12. 서로 다른 OS-Application간의 통신

IOC에서 Sender-Receiver interface를 이용한 통신은 그림 13과 같다. 우선 소프트웨어 컴포넌트가 *RTE_Send()* API를 호출하여 데이터를 RTE로 보낸다. RTE에서는 *IocSend()* API를 호출하여 해당 데이터를 IOC Buffer에 저장한다. 다른 OS-Application 내의 소프트웨어 컴포넌트가 해당 데이터를 읽기 위해 *Rte_Receive()*를 호출하고 RTE에서는 *IocReceive()*를 호출하여 IOC Buffer내의 데이터에 접근하게 된다. 단, ICO Buffer에 저장된 데이터는

쓰기 또는 읽기 동작을 할 때, 데이터가 수정되는 것을 방지 하기 위해 Spinlock이라는 동기화 메커니즘으로 보호된다.

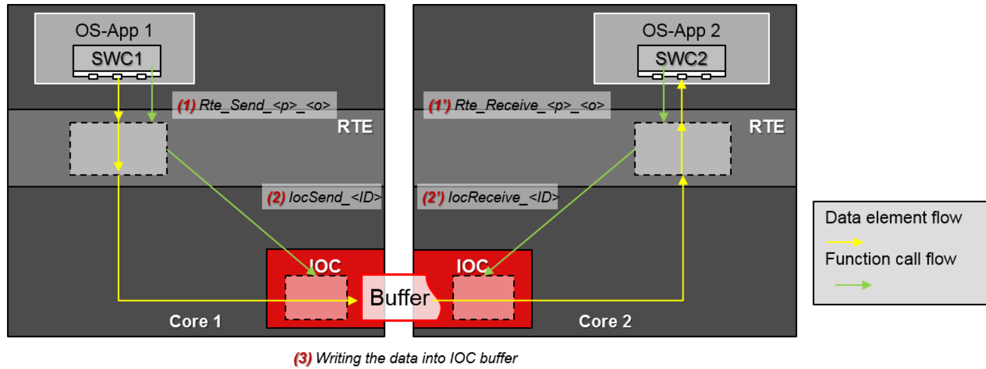


그림 13. IOC에서 Sender-Receiver 통신

다음은 IOC에서 Client-Server interface를 이용한 통신이다. 그림 14와 같이 Operation을 제공 받고자 하는 소프트웨어 컴포넌트는 *RTE_Call_<p>_<o>*를 호출하여 RTE에 알리고 RTE에서는 *IocSend_<id>*를 이용하여 제공받고자 하는 operation을 Server 컴포넌트에 알린다. Server 쪽 RTE는 IocReceive_<id>를 이용해 어떤 Operation이 제공되어야 하는지를 해당 소프트웨어 컴포넌트에게 전달하고 수행된 결과를 IOC Buffer에 기록하여 Client쪽에 전달한다. IOC는 Client-Sever 통신을 Sender-Receiver 통신과 notification을 이용해서 지원한다.

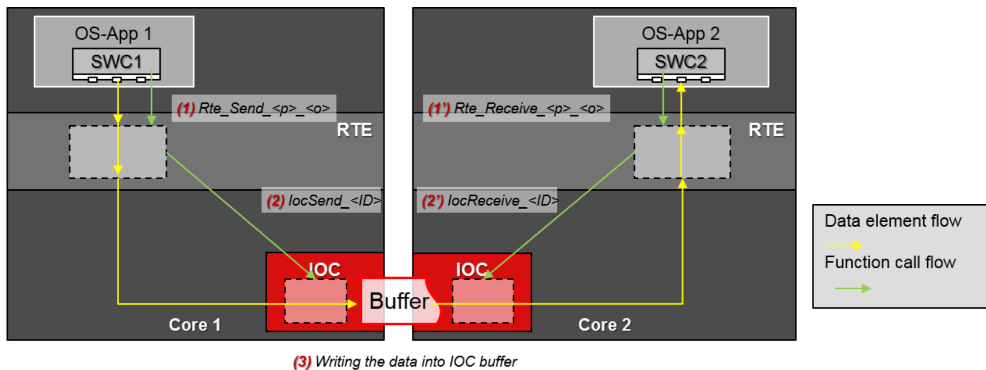


그림14. IOC에서 Client-Server 통신

마지막으로 서로 다른 ECU간의 통신이다. BSW에는 통신기능을 담당하는

COM Stack이 있다. Com Stack 은 주기, Message ID, 통신 방식 (CAN, LIN, FlexRay) 등을 설정하여 ECU 간의 통신을 지원하는 모듈이다. 다른 ECU로 데이터를 보내고자 하는 소프트웨어 컴포넌트는 COM Stack 에서 제공하는 API를 호출하여 ECU 밖으로 데이터를 송신하고 수신할 수 있다.

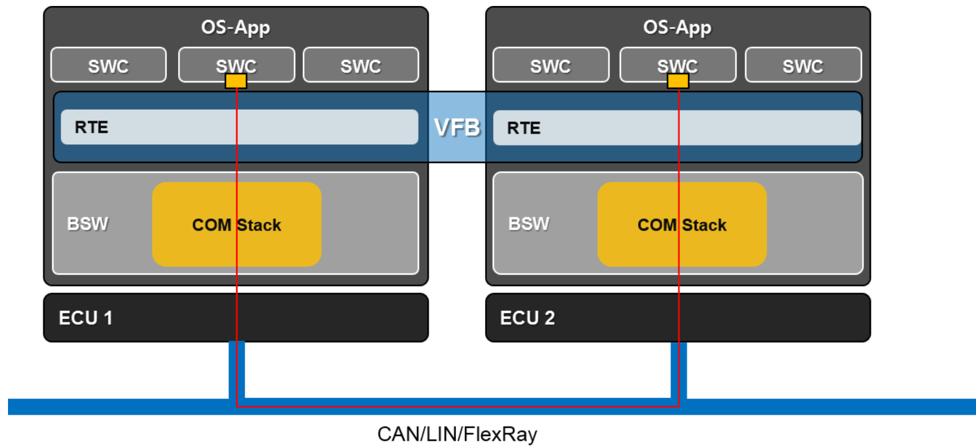


그림 15. 서로 다른 ECU간의 통신

제 3절 Read-Copy-Update

Read-Copy-Update는 읽기 동작에서 블러킹이 되지 않는 read/write 동기화 메커니즘이다. 2002년 리눅스 커널 버전 2.5.43에서 소개되었고 그 후 2005년 PREEMPT_RCU 가 추가되었고, 2009년 user-level RCU도 소개되었다.

RCU의 주된 아이디어는 다른 버전의 데이터에 접근 가능하도록 하여 critical section에 진입하기 전의 waiting 시간을 없애는 것이다. RCU는 read-side overhead를 최소화하는데 목적이 있기 때문에 동기화 로직이 읽기 동작에 많은 비율로 사용되는 경우에만 사용한다. 쓰기 동작이 10% 이상인 경우 다른 동기화 로직을 선택하는 것이 합당하다. RCU는 쓰기 동작에서는 기존과 같은 동기화 기법을 사용해야 한다. PREEMPT_RCU을 사용하면 read-side critical section에서

preemption 될 수 있다.

RCU의 장단점을 살펴보면, 장점으로는 성능향상을 들 수 있다. 특히, 읽기 동작에서 zero wait, zero overhead이기 때문에 성능이 향상된다. 또한 구현된 RCU library를 API 형태로 적용할 수 있다는 측면에서 확장성이 좋다. 둘 이상의 프로세스가 서로 다른 프로세스가 가진 자원을 요구하면서 양쪽 모두 작업 수행을 할 수 없이 대기 상태로 놓이는 Deadlock 이슈가 없다. 또한 낮은 우선순위의 프로세스가 더 높은 우선순위의 프로세스를 블럭하는 priority inversion의 이슈가 없다. 또한 전달 지연 시간을 바운드 시킬 수 있다.

단점으로는 RCU 구현과 사용이 복잡하다는 측면과 쓰기 동작에는 개선 사항이 없다는 점이다. 또한 데이터의 freshness를 보장할 수 없다는 단점이 있다. 여기서 freshness란, 읽기 동작이 항상 데이터의 가장 최근의 정보를 참조 할 수 있다는 것이다.

RCU가 동작하기 위해서 읽기 동작에서는 *Lock* (critical section을 시작함), *Unlock* (critical section을 끝냄), *Dereference* (포인터로 참조 하고자 하는 데이터 구조를 읽어드림)의 API가 필요하다. 쓰기 동작에서는 *Assign* (포인터를 할당해서 새로운 데이터 구조를 생성함), Synchronize (이전 reader가 critical section을 끝마치길 기다림)의 API가 필요하다.

RCU를 이용한 동기화 메커니즘의 일련의 과정을 살펴 보면 아래 그림16과 같다. 우선 Reader1은 critical section을 시작하기 위해 lock을 얻고 read하고자 하는 데이터가 저장된 메모리 주소 값을 읽어 드린다. 여기서의 lock은 spin lock과 달리 다른 Reader, Writer가 데이터에 접근하는 것을 막지 않는다. Writer가 데이터를 적고자 할 때, Assign() API를 호출하여 저장할 새로운 데이터 구조를 생성한다.

Reader2가 데이터에 접근하고자 할 때 lock을 얻고 생성된 새로운 데이터가 저장된 주소 값을 읽어드린다. Reader2가 데이터를 읽어드린 후 unlock을 통해 critical section 수행을 종료한다. Writer는 Reader가 critical section을 종료하면 Reader가 참조한 데이터를 free하여 데이터의 Synchronize를 한다.

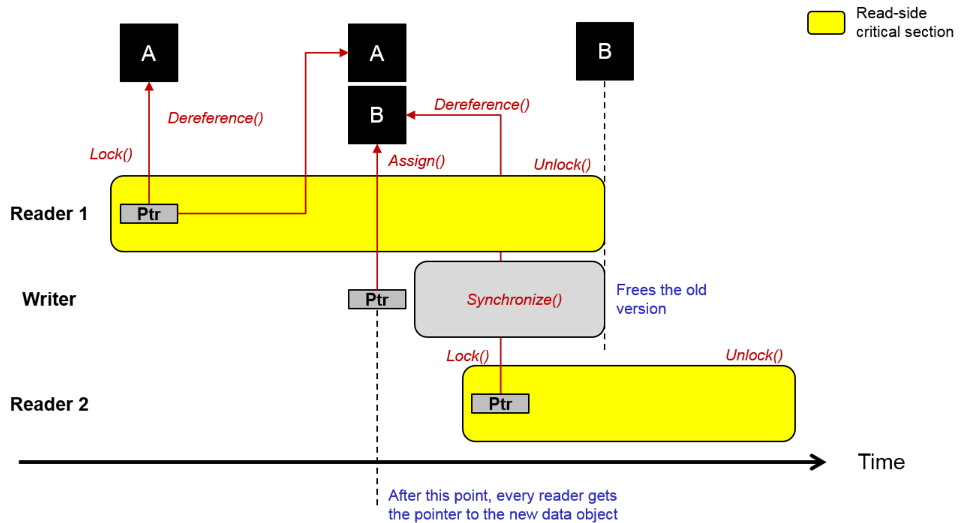


그림 16. RCU를 이용한 데이터 Read/Write 동작 과정

여기서 하나의 의문점은 writer가 어떻게 reader가 critical section을 종료했는지를 아느냐는 것이다. 이를 설명하기 위해서는 quiescent state와 grace period의 개념이 도입된다. Quiescent state는 task가 critical section을 수행하지 않는 상태를 의미한다. Grace period란, 그림 17에서 보는 것처럼 모든 thread가 적어도 하나의 quiescent state에 있는 기간을 말한다. 다시 말해 writer가 생성한 데이터를 reader가 critical section에 진입하여 해당 데이터를 모두 참조할 때까지의 기간이다. 모든 read−side critical section이 종료되는 시점을 알기 위해서 CPU가 context switch가 완료되었는지를 확인한다. 모든 CPU context switch가 완료되면 RCU read−side critical section

period가 완전히 끝났다고 보장할 수 있게 된다. 이때야 비로소 old data를 free하여 제거한다.

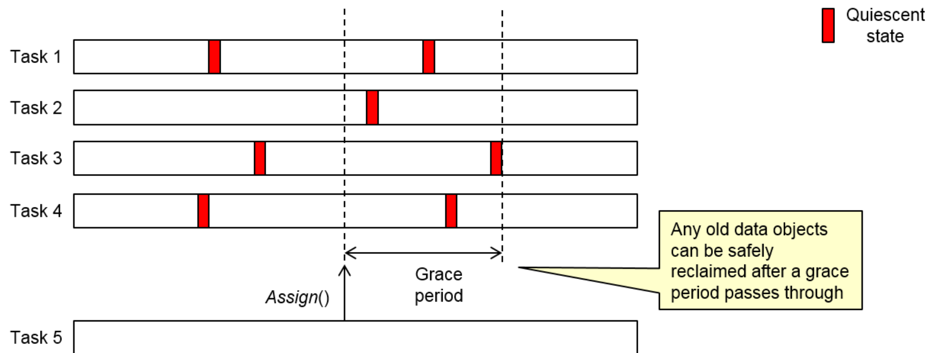


그림 17. RCU의 Grace Period

유저 입장에서 RCU 동기화 메커니즘을 사용하기 위해서는 아래와 같은 API가 필요하다. Reader operation을 위해서는 Lock, Unlock, Dereference의 API가 필요하고 Writer operation을 위해서는 Assign, Synchronize의 API가 필요하다. 그림 18은 Linux에서 제공하는 RCU API를 정리한 것이다.

		Basic RCU with PREEMPT!=NO	Basic RCU with PREEMPT=NO	RCU-BH
Convention (in a read-side critical section)		No blocking and preemption	No blocking and preemption	BH disabled
Observed quiescent state		Context switch	Voluntary context switch	Completion of BH handler
Reader operations	Lock	rcu_read_lock() Disable preemption	rcu_read_lock() Do nothing	rcu_read_lock_bh() Disable BH
	Unlock	rcu_read_unlock() Enable preemption	rcu_read_unlock() Do nothing	rcu_read_unlock_bh() Enable BH
	Dereference	rcu_dereference()	rcu_dereference()	rcu_dereference_bh()
Writer operations	Assign	rcu_assign_pointer()	rcu_assign_pointer()	rcu_assign_pointer()
	Synchronize	synchronize_rcu()	synchronize_rcu()	synchronize_rcu_bh()

표 1. Linux에서 사용되는 RCU API Family

제 3 장 문제 정의

이 장에서는 앞서 제 2장에서 설명한 Multi-Core AUTOSAR에서 제공하는 IOC가 지닌 한계점을 설명한다. 이를 위해 본 논문에서 설명하고자 하는 대상 시스템을 설명하고 Multi-core AUTOSAR의 통신 방식인 IOC의 한계점에 대해 설명한다.

제 1절 시스템 모델

이 장에서 설명하고자 하는 시스템 모델은 앞서 제 2장에서 설명한 AUTOSAR 차량용 소프트웨어 플랫폼이다. AUTOSAR는 계층형 소프트웨어 구조를 가지고 있고 각각의 계층은 MCAL, ECU Abstraction Layer, Service Layer, Complex Device Driver, RTE, Application Software로 이뤄져 있다. 여기서 설명하고자 하는 IOC는 AUTOSAR의 통신을 담당하고 있는 RTE에 속한다. 그림 19와 같이 RTE는 AUTOSAR 계층에서 ASW (Application Software Layer)와 BSW (Basic Software)의 중간에 속한다.

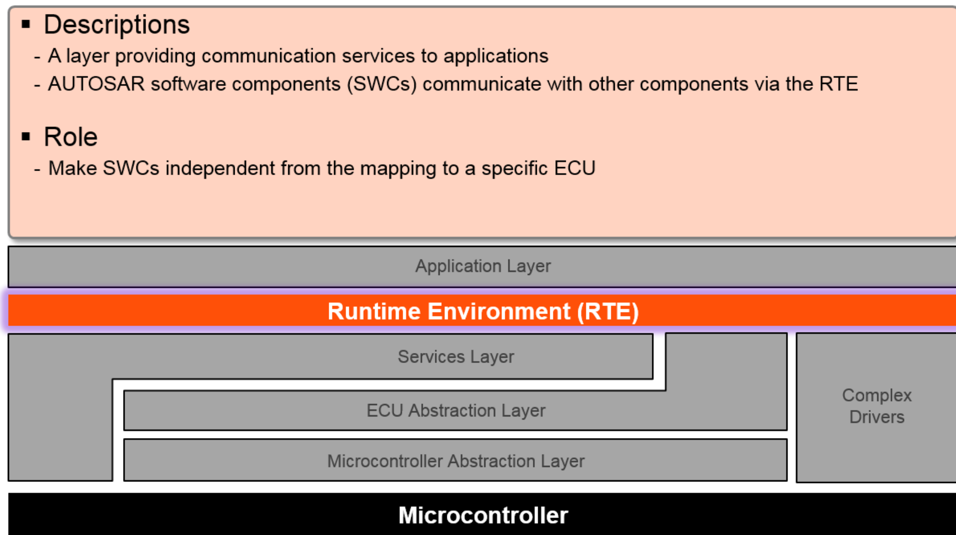


그림18. Runtime Environment

제 2절 IOC의 한계점

AUTOSAR를 기반으로 한 시스템에서의 통신 루트는 3가지이다. 같은 OS application에 존재하는 소프트웨어 컴포넌트간의 통신, 서로 다른 OS application에 존재하는 소프트웨어 컴포넌트간의 통신, 서로 다른 ECU에 존재하는 소프트웨어 컴포넌트간의 통신이다. IOC의 한계점은 두 번째 통신 루트인 서로 다른 OS application에 존재하는 소프트웨어 컴포넌트간의 통신에서 발생한다. 앞서 말한 대로, OS application은 Core간의 메모리 분할을 가능하게 하여 다른 Core에서 동작하는 프로세스가 다른 Core에 할당된 메모리 영역에 침범하는 것으로부터 보호한다. IOC는 서로 분할된 메모리 영역을 이어주는 역할을 하며 이를 통해 서로 다른 Core에 존재하는 소프트웨어 컴포넌트간에 통신을 가능하게 한다. 하지만 IOC를 이용한 통신에서는 여러 Core에서 동작하는 Task들이 동시에 데이터에 접근하는 일이 발생할 수 있으며, 이에 따라 데이터의 동기화 문제가 발생할 수 있다.

이에 대한 해결책으로 AUTOSAR에서는 Spinlock 메커니즘을 이용하여 한 프로세스가 데이터에 접근하고 있을 때, 다른 프로세스의 접근을 차단한다. 이를 통해 데이터의 동기화 문제를 해결하고 있다. Spinlock은 데이터 동기화 문제를 해결하는 강력한 방법이지만, 다른 프로세스가 Lock을 획득하여 데이터를 점유하고 있는 중에는 나머지 프로세스는 Spinning을 하며 해당 데이터가 Unlock되기를 기다려야 한다. 이러한 문제는 소수의 프로세스가 동작 할 때는 큰 문제가 되지 않지만, 다수의 프로세스가 하나의 데이터를 점유하기 위해 경쟁하는 경우에는 프로세스의 수행시간에 지연을 줄 수 있다. 특히, AUTOSAR와 같이 여러 개로 분할된 소프트웨어 컴포넌트들이 다수의 데이터를 교환하는 시스템에서는 spinning으로 인한 프로세스의 수행 지연이 시스템 전체의 성능에 영향을 줄 수 있다.

다음은 AUTOSAR를 기반으로 한 전자식 브레이크 시스템을 예를 들어 IOC의 통신 지연 문제를 밝히고자 한다. 자율주행자동차 개발을 위해서는 자동차 스스로 제동할 수 있는 지능형 브레이크가 필요하다. 특히, 긴급상황에서 장애물을 감지 하여 충돌을 회피하기 위한 AEB (Autonomous Emergency Brake)는 자율 주행 자동차뿐만 아니라 일반 차량에도 많이 사용된다. AEB를 구현하기 위해서는 장애물을 감지한 시점부터 300ms 안에 80Bar 이상의 제동압력을 생성시키는 요구사항을 만족해야 한다. AUTOSAR를 기반으로 한 AEB 시스템에서의 데이터 흐름도를 분석하여 IOC의 한계점을 밝히고자 한다.

그림20에서 보는 것처럼 AEB 시스템은 장애물을 감지하여 이 정보를 보내주는 Radar 모듈과 필요한 제동압력을 계산하여 실제 압력을 생성하는 Brake 모듈로 구성된다. 두 개의 모듈은 Multi-core AUTOSAR로 소프트웨어 구조가 설계되어 있고 서로 모듈간의 통신은

CAN (Controller Area Network)으로 이뤄진다. 전체 데이터 흐름도를 살펴보면 Radar 정보를 BSW의 모듈인 MCAL과 ECU Abstraction Layer에서 읽어 들이고 해당 데이터를 RTE로 송부한다. RTE에 저장된 데이터는 소프트웨어 컴포넌트3 (SWC3)가 읽어 들여서 전기적 데이터를 물리적 데이터로 변환시키고 가공된 데이터를 다른 Core에서 동작하는 SWC1,2로 보내기 위해 IOC 통신을 이용한다. IOC 버퍼에 저장된 데이터를 SWC1,2에서 읽어 드려서 긴급제동을 위한 목표 제동압력과 제동제어명령을 생성한다. SWC1,2에서 생성된 데이터는 BSW의 Com (Communication Stack)을 통해서 Brake모듈로 전달된다. Brake 모듈의 Com에서 이를 전달 받아 SWC4로 넘겨주고 SWC4 에서는 목표제동압력을 생성하기 위해 필요한 세부 제어 정보를 결정하고 이 정보를 IOC 를 통해 SWC5,6으로 전달한다. 해당 정보를 통해 브레이크 압력을 생성시키기 위한 모터와 밸브 구동명령을 생성하고 이를 BSW의 MCAL로 전달하여 실제 브레이크를 구동한다.

그림. 19 전자 브레이크 시스템에서의 통신 경로

각 모듈 별로 구성된 소프트웨어 컴포넌트와 이들을 스케줄링하는 각 Task의 동작을 세부적으로 살펴보면 그림 21과 같다. 각 SWCi의 메인 Runnable은 Taski에 Mapping되어 동작함을 가정한다. 즉, SWC1,2,3은 각각 Task1,2,3에 수행되어 동작한다. Task3은 Radar의 데이터를 BSW로부터 전달 받아 이를 물리적 신호로 변환한다. 변환된 데이터를 다른 코어에서 동작하는 소프트웨어 컴포넌트로 보내기 위해 IOC통신을 사용하며 IOC를 이용해 데이터를 Write할 때 데이터의 동기화를 위해 Lock을 획득하여 다른 Task의 접근을 차단한다. 데이터의 Write Operation이 끝이나면 Unlock을 통해 다른 Task의 접근을 허락한다. 해당 데이터에 접근을 원하는 Task1와 Task2는 그림21에서 보는 것과 같이 Lock을 획득하기 위해서 Spinning 하며 수행시간이 지연된다. 특히, Task1의 경우는 Task3과 Task2 의 수행이 모두 종료되어야만 데이터에 접근할 수 있기 때문에 지연시간이 더욱더 길어진다. 결론적으로 IOC 통신 시, 데이터 동기화를 위해 사용하는 Spinlock이 데이터 전달 지연을 야기한다.

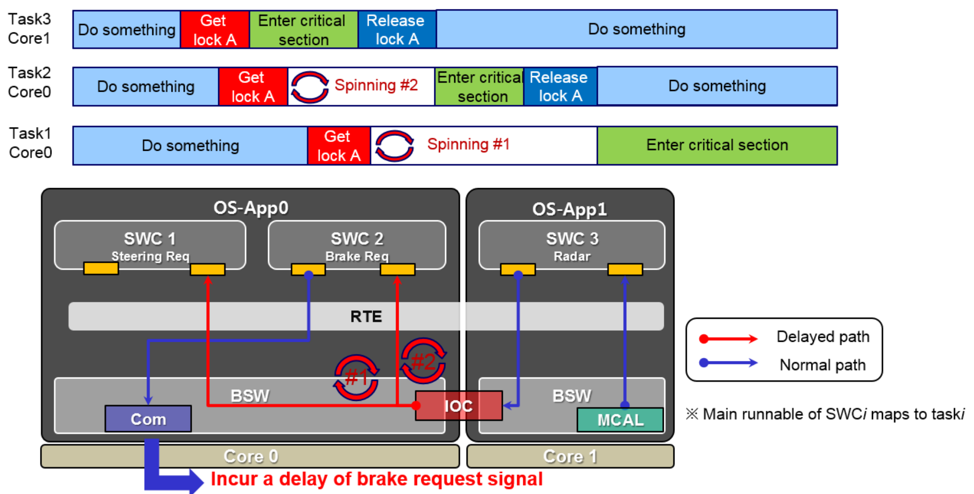


그림 20. 레이더 모듈에서의 통신 지연

다음은 브레이크 모듈에서의 통신 지연에 대해서 설명하고 자 한다. 그림 22에서 보는 것과 같이 Radar 모듈에서 전달 받은 데이터를 Communication Stack을 통해 SWC4로 전달되고 제동을 위해 필요한 액츄에이션 정보를 생성하여 이를 IOC 통신을 통해 다른 Core에서 동작하는 SWC5,6 으로 보낸다. 여기서도 마찬가지로 SWC4가 Mapping되어 동작하는 Task4가 Lock을 잡고 데이터에 접근하는 동안 Task5,6은 해당 데이터가 Unlock될때까지 Spinning하며 수행이 지연된다. Brake 모듈에서 역시 마찬가지로 IOC 통신시에 발생하는 Spinning이 전체 데이터 전송 지연을 초래한다.

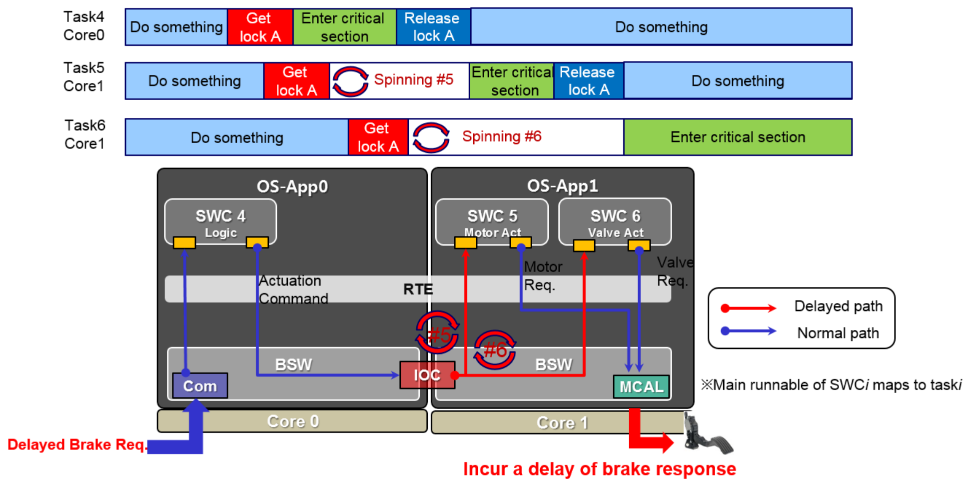


그림 21. 브레이크 모듈에서의 통신 지연

앞에서 살펴본 각 Radar, Brake 모듈의 통신 지연은 AEB의 요구사항인 80Bar/300ms 를 만족시키는데 큰 걸림돌로 작용한다. 하나의 데이터를 접근하고자 하는 Task의 수가 적으면 Spinlock으로 인한 spinning문제가 소소할 수 있겠지만, 그림 23과 같이 AUTOSAR 시스템에서는 하나의 데이터를 여러 개의 소프트웨어 모듈이 참조하는 경우 즉, 1 : N (Writer : Reader) 통신이 많기 때문에 Spinning으로

인한 통신 지연이 누적되어 전체 시스템의 성능을 떨어뜨린다.

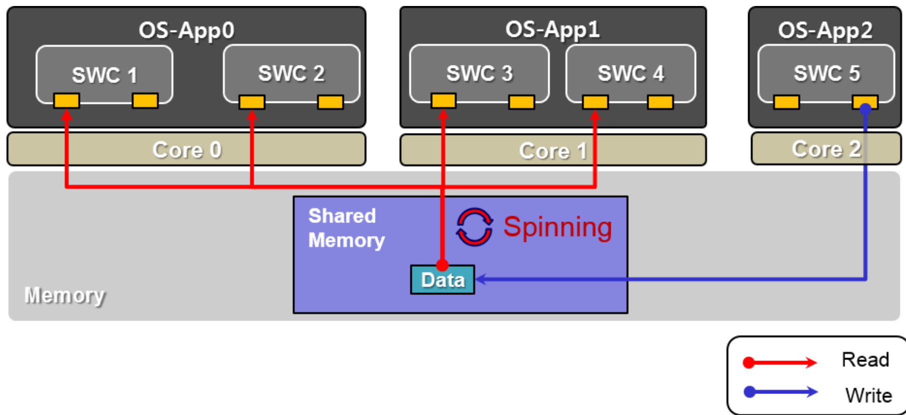


그림 22. AUTOSAR에서의 1: N 통신

제 4 장 개선된 AUTOSAR의 IOC 통신

제 1절 RCU를 적용한 개선된 IOC 통신

앞에서 설명한 AUTOSAR의 IOC 통신 지연 문제를 해결하기 위해서 기존 Spinlock 동기화 메커니즘이 아닌 RCU를 적용하여 개선된 동기화 메커니즘을 구현하고자 한다. RCU는 앞 2절에서 설명한 것과 같이 Read 동작을 하는 Task에게 동시 접근을 가능하게 하는 Mutual exclusion을 제공하는 동기화 메커니즘이다. 이를 통해 Read 동작을 하는 Task간의 경쟁이 사라짐으로써, 그 만큼의 수행 시간을 단축 시킬 수 있다. 다만, 여전히 Write 동작을 하는 Tasks 간에는 서로 Block되어 spinning 시간이 발생한다. 따라서 RCU는 1 : N (Write : Reader)의 통신 방식에 적합하다.

자동차 내에서의 데이터 흐름을 보면, 하나의 모듈에서 생성한 데이터를 다수의 모듈에서 동시에 참조하는 경우가 많다. 예를 들면, Brake ECU에 장착된 Wheel Speed Sensor의 값을 처리하는 모듈에서 생성한 데이터를 자동차 속도를 계산하는 모듈, ABS 제어를 하는 모듈, ESC 제어를 하는 모듈, CBC 제어를 하는 모듈, 자동차 속도를 모니터링 하는 모듈, EPB 제어를 하는 모듈 등등에서 참조한다. 이렇듯 하나의 데이터 정보를 다수의 모듈이 동시에 참조하는 경우가 많은 자동차 시스템에서는 RCU 동기화 메커니즘을 도입하는 것이 성능측면에서 적합하다.

RCU의 동작 메커니즘과 특성 등은 제 2장에서 설명했으니, 생략하고 제 3장에서 설명한 IOC의 한계점으로 발생하는 통신 지연 문제를 RCU를 적용했을 때 얼마나 통신 지연이 해결되는지를 살펴 보고자 한다.

아래 그림 24는 제 3장에서 언급한 IOC의 동기화 메커니즘은

Spinlock을 사용했을 때와 RCU를 적용하여 개선된 IOC 동기화 메커니즘을 사용했을 때의 Task 수행 시간을 비교한 칸트 차트이다. 그림에서 보는 것과 같이 Read Operation을 하는 Task는 데이터에 접근할 때 waiting 이 없기 때문에 Spinlock에서 발생하는 spinning 지연시간이 없다. 따라서 그만큼의 시간을 세이브 할 수 있다. 또한 이러한 성능 개선 효과는 Read Operation을 하는 Task의 수가 늘어날수록 더욱 두드러진다.

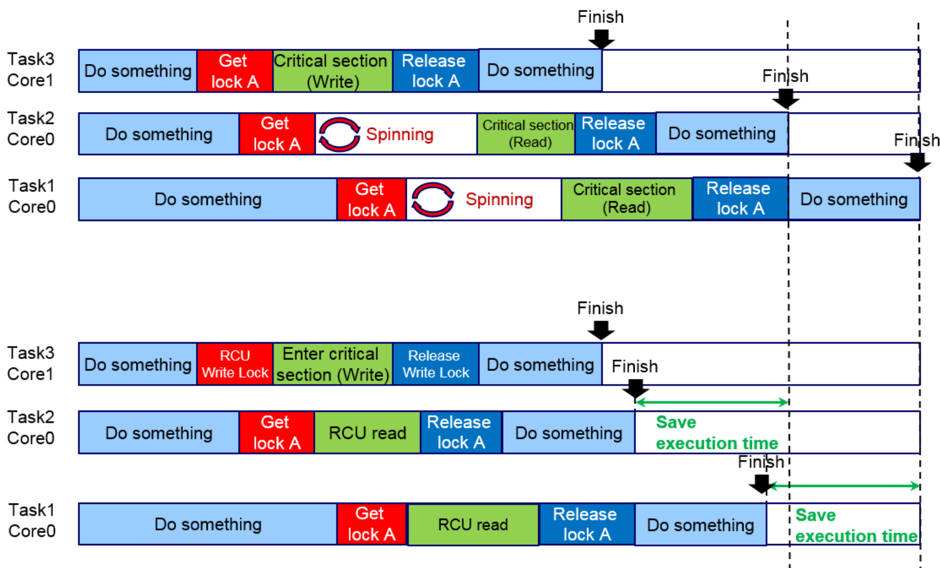


그림 23. Spinlock과 RCU 비교

제 2절 개선된 IOC 통신 구현

이 절에서는 앞에서 설명한 RCU 를 적용한 개선된 IOC 통신을 구현하는 것을 설명하고자 한다. IOC의 동기화 메커니즘인 Spinlock의 경우 AUTOSAR에서 API 형태로 제공이 된다. 하지만, RCU 통신을 위한 API는 제공이 되지 않기 때문에 사용자가 직접 구현을 해야 한다. 구현

해야 하는 RCU API는 Read Operation을 위한 Lock, Unlock, Dereference 3개가 필요하고 Write Operation을 위해서는 Assign, Synchronize 2개의 API가 필요하다. 아래 그림은 AUTOSAR Builder로 생선된 RTE code 에 RCU library source code, header 파일을 구현한 것이다. 앞서 설명한 API를 정의하였고 Data를 저장할 Buffer 생성을 위해 data structure를 정의하였다.

```
**                                                    **
** Description : Read- Copy Update                     **
*********************************************************************

/*********************************************************************
/*RCU function definition*/
/*********************************************************************
void rcu_assign_pointer(uint32 *ptr, uint32 value);
void rcu_dereference(uint32 value, uint32 *ptr);
void rcu_read_lock(void);
void rcu_read_unlock(void);
void Synchronize(void);


/*********************************************************************
/*RCU function description*/
/*********************************************************************
void rcu_assign_pointer(uint32 *ptr, uint32 value)
{
   __imaskldmst(ptr, value, 0, 32)
}

void rcu_dereference(uint32 value, uint32 *ptr)
{
   __imaskldmst(ptr, value, 0, 32);
}

void rcu_read_lock(void)
{
     return rcu_read_lock(&LockResourceKey);
}

void rcu_read_unlock(void)
{
     return rcu_read_unlock(&LockResourceKey);
}

void Synchronize(void)
{
   return rcu_syncronization();
}
```

assign() for writer

deference() for writer

lock() for reader

unlock() for reader

Synchronize() for writer

<Rculib.c>

```
   uint32      RCU_Buffer14;
   uint32      RCU_Buffer15;
   uint32      RCU_Buffer16;
   uint32      RCU_Buffer17;
   uint32      RCU_Buffer18;
   uint32      RCU_Buffer19;
   uint32      RCU_Buffer20;
   uint32      RCU_Buffer21;
   uint32      RCU_Buffer22;
   uint32      RCU_Buffer23;
   uint32      RCU_Buffer24;
   uint32      RCU_Buffer25;
   uint32      RCU_Buffer26;
   uint32      RCU_Buffer27;
   uint32      RCU_Buffer28;
   uint32      RCU_Buffer29;
   uint32      RCU_Buffer30;
   uint32      RCU_Buffer31;
   uint32      RCU_Buffer32;
   uint32      RCU_Buffer33;
   uint32      RCU_Buffer34;
   uint32      RCU_Buffer35;
   uint32      RCU_Buffer36;
   uint32      RCU_Buffer37;
   uint32      RCU_Buffer38;
   uint32      RCU_Buffer39;
   uint32      RCU_Buffer40;
   uint32      RCU_Buffer41;
   uint32      RCU_Buffer42;
   uint32      RCU_Buffer43;
   uint32      RCU_Buffer44;
   uint32      RCU_Buffer45;
   uint32      RCU_Buffer46;
   uint32      RCU_Buffer47;
   uint32      RCU_Buffer48;
   uint32      RCU_Buffer49;
   uint32      RCU_Buffer50;
   uint32      RCU_Buffer51;
   uint32      RCU_Buffer52;
   uint32      RCU_Buffer53;
   uint32      RCU_Buffer54;
   uint32      RCU_Buffer55;
} ? end {anonSync_RCU_Local_ConfigType} ? Sync_RCU_Local_ConfigType;

Sync_RCU_Local_ConfigType *local_ptr, *old_ptr;

#define rcu_read_lock(pLockkey) \
   do { \
         _rcu_read_lock();
   } while(0)

#define rcu_read_unlock(pLockkey) \
   do { \
         _rcu_read_lock();
   } while(0)

#define rcu_syncronization() \
   do { \
        ;
   } while(0)
```

RCU data structure

Macro for *lock()*

Macro for *unlock()*

<Rculib.h>

표 2. RCU Library 구현

위에서 구현한 API를 이용하여 IOC 통신을 이용한 Data Exchange Code 에 적용하면 아래 그림 26과 같다. 우선 Core0에서 수행 중인 Task1ms에서 데이터를 Write하는 operation을 보면 우선 RCU Local Pointer에 동적 메모리를 할당한다. 그리고 앞에서 선언한 RCU Buffer Structure에 쓰고자 하는 데이터를 Copy한다. Data Copy가 끝나면, Global 포인터의 값을 Old 포인터로 넘기고 Local 포인터의 값을 Global 포인터로 할당한다. Synchronize()를 통해 이전 데이터의 Read 접근이 끝났음을 알게 되면, Old 포인터에 할당된 데이터는 free를 통해

삭제된다.

 Writer Operation을 보면 다른 Core1에서 수행 중인 Task5ms의 Data Exchange Code에서 우선 Lock을 통해 Data의 접근 권한을 얻는다. 하지만 이 Lock은 Spinlock의 Lock과는 달리 다른 Task의 접근을 Block 시키지 않는다. 데이터가 저장된 Global 포인터를 Local 포인터로 읽어 드리고 Local 포인터의 데이터를 읽어드린다. 데이터를 모두 읽어 드린 후 Unlock을 통해 Read Operation을 맞힌다.

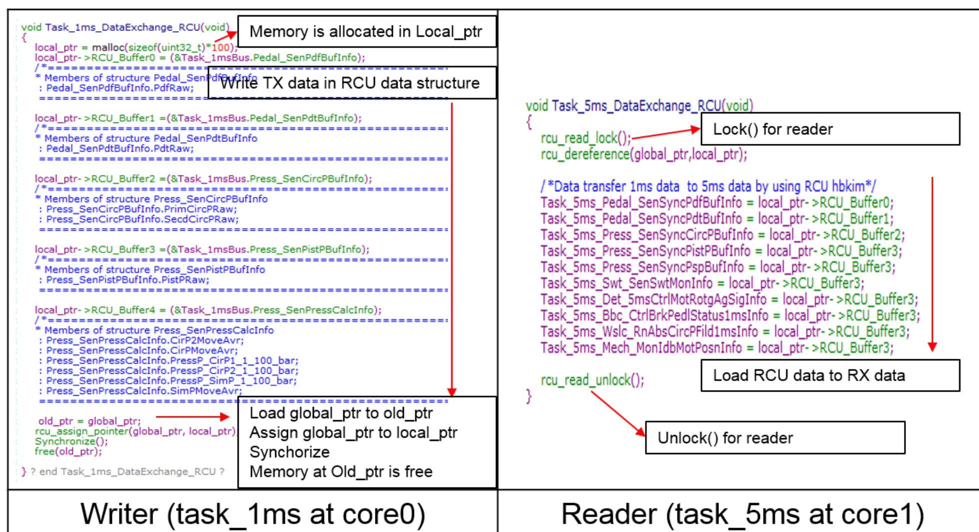


표 3. RTE 코드에서 RCU API 호출을 통한 구현

제 5 장 실험 및 검증

이 장에서는 앞서 설명한 RCU를 적용하여 개선된 AUTOSAR의 IOC 통신 메커니즘이 3장에서 설명한 AUTOSR IOC 통신 메커니즘의 한계점을 보완하는지 실험환경을 구축하고 검증한다.

제 1절 실험 환경

RCU를 적용한 개선된 AUTOSAR의 IOC 메커니즘을 구현하고 검증하기 위해 아래 표1과 같이 하드웨어와 소프트웨어 실험 환경을 구성하였다. 실험에 사용한 MCU는 AURIX 제품인 TC277 시리즈이며, 4MB의 RAM과, 2 CH의 CAN Transceivers, 2 FlexRay Transceivers의 통신 모듈을 지원한다. 소프트웨어는 EB사에서 배포한 v4.1.1 버전의 AUTOSAR 플랫폼을 사용하였고, 응용 소프트웨어 모델링은 AUTOSAR Builder를 이용하였고 BSW는 EBtresos 10.0을 이용해서 설정하였다. 응용 소프트웨어는 Mando에서 제공한 전자식 브레이크 소프트웨어를 사용하였다.

Hardware			Software	
ECU	MCU	TC277 (32bit Microcontroller)	AUTOAR version	Release v4.1.1
	Memory	4MB flash memory	SWC modeling	AUTOSAR Builder
	Comm.	2 CAN transceivers 2 FlexRay transceivers	BSW Configuration	EBtresos 10.0
Actuator	HCU	Solenoid valve, check valve	Application SW	Mando brake SW
	Motor	3 phase induction motor		

표 4. 실험 환경 구축을 위한 하드웨어와 소프트웨어 상세

또한 ECU와 소프트웨어로 컨트롤하는 Actuator는 아래 그림25와 같이 유압으로 제동압력을 생성하기 위한 3상모터, Solenoid 밸브, Caliper로 구성된다.

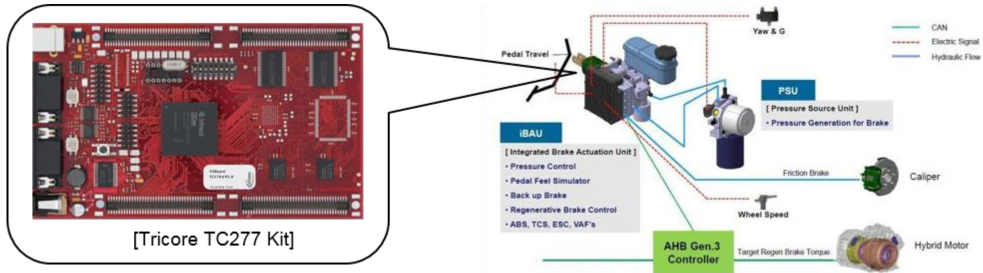


그림 24. 브레이크 시스템 구성도

제 2절 실험 구성

실험은 기존의 AUTOSAR IOC 통신 메커니즘과 RCU를 도입하여 개선된 AUTOSAR IOC 통신 메커니즘의 수행시간을 비교하여 진행한다. IOC의 통신 수행시간을 비교하기 위해 실제 IOC를 통해 데이터를 주고받는 Task의 수행시간을 측정한다. IOC의 통신 수행시간이 Task의 수행시간에 직접적으로 영향을 주기 때문에 Task의 수행시간을 측정하여 이를 평가하고자 한다.

수행시간을 측정 하고자 하는 Task 아래 그림 26과 같이 Core0에서 Sensor 신호와 ASIC 정보를 처리하여 전달하기 위해 수행 중인 Task_1ms와 Core1에서 이 정보를 이용해서 Sensor 신호를 가공하고 제어 로직을 수행하는 Task_5ms로 구성된다. Read access하는 Task의 수를 늘려가면서 수행시간을 측정한다. 이유는 Read access하는 Task의 수가 늘어날수록 RCU를 적용한 개선된 IOC 통신의 효과가 두드러질 것으로 판단되기 때문이다.

아래 그림에서 빨간색 화살표는 소프트웨어 컴포넌트가 Task에 Mapping되어 있는 것이고 파란색 화살표는 Task간의 데이터 전달 흐름을 나타낸 것이다. 또한 녹색 화살표는 ECU 간의 통신을 위해 Comm Stack 모듈로 데이터가 전달되는 경로이다.

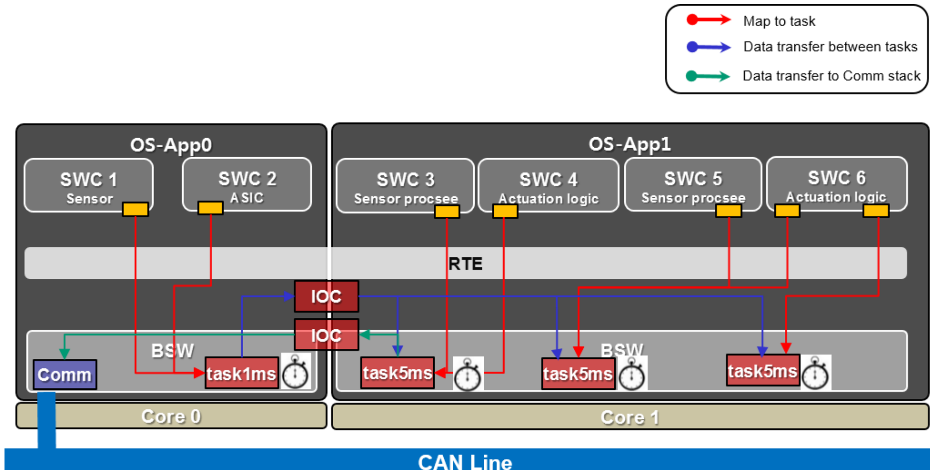


그림 25. 실험 평가 모델

Task의 수행시간을 측정하기 위해 Task의 시작과 종료 시간을 time capture하여 수행시간의 평균값, 최대값, 최소값을 계산하였다. 표 4에 구현한 함수를 Task가 종료되는 시점에 호출하여 Task의 수행시간을 측정하였다.

```
static void Task_5ms_Exit(void)
{
#if TASK_5MS_MEAS_EXEC_TIME == STD_ON
   Task_5ms_Timer_Elapsed = STM0_TIM0.U - Task_5ms_Timer_Start;

   if(Task_5ms_Timer_Elapsed > Task_5ms_Timer_Elapsed_Max)
   {
      Task_5ms_Timer_Elapsed_Max = Task_5ms_Timer_Elapsed;
   }
   if(Task_5ms_Timer_Elapsed < Task_5ms_Timer_Elapsed_Min)
   {
      Task_5ms_Timer_Elapsed_Min = Task_5ms_Timer_Elapsed;
   }
   Task_5ms_Timer_Elapsed_Avg = (Task_5ms_Timer_Elapsed_Avg * 0.9F) + (Task_5ms_Timer_Ela
#endif
}
```

표 5. Task 수행 시간 측정을 위한 코드 구현

또한 측정한 Task 수행시간이 저장된 변수 값을 읽어드려 Graph로 보이고 이를 분석하기 위한 Tool로 CANape를 이용하였다. CANape는 메모리에 존재하는 전역변수를 실시간으로 모니터하고 저장할 수 있다.

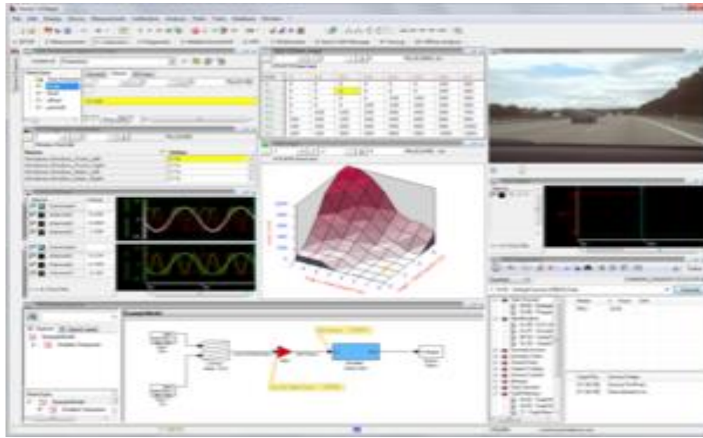


그림. 26 변수 계측 장비 CANape

제 3절 실험 평가

실험 평가는 앞서 설명한 것과 같이 기존 AUTOSAR IOC 통신을 이용하여 수행한 Task의 수행시간을 측정하고 같은 조건에서 RCU를 적용한 AUTOSAR IOC 통신을 이용하여 수행한 Task의 수행시간을 측정하여 이를 비교한다. 또한 Read access하는 Task의 수를 늘려가며 Task의 수행시간을 평균하여 이를 비교한다. 개선된 AUTOSAT IOC통신의 개선율을 나타내기 위해 아래와 같이 개선율을 개산하였다.

개선율(Improvement Rate) =

$$\frac{(Task\ execution\ time\ of\ existing\ system\ - Task\ execution\ time\ of\ system\ with\ RCU)}{Task\ execution\ time\ of\ existing\ system} * 100$$

측정한 Task의 수행시간은 아래 표5와 같이 Read access task의 수를 늘려가며 (Number of Tasks = 5, 6, 7, 8, 9, 10) 측정하였고 이를 평균하여 정리한 것이다.

Increasing task's number	Existing System (Average of Task Execution Time)	System with RCU (Average of Task Execution Time)	Improvement Rate
Number of Tasks : 5	3.251768 ms	3.204995 ms	1.4%
Number of Tasks : 6	3.365519 ms	3.302687 ms	1.8%
Number of Tasks : 7	3.512452 ms	3.42826 ms	2.3%
Number of Tasks : 8	3.617712 ms	3.462431 ms	4.2%
Number of Tasks : 9	3.758253 ms	3.496844 ms	6.9%
Number of Tasks : 10	3.798927 ms	3.50357 ms	7.7%

표 6. 실험 결과 비교 (기존 시스템과 개선된 시스템)

실험 결과를 살펴보면, RCU를 적용하여 개선된 시스템이 Task의 수가 늘어날수록 그 개선율이 뛰어남을 알 수 있다. 10개의 Read access tasks의 경우 약 7.7% 개선율을 보임을 알 수 있었다. Task의 수가 1~4개인 경우 기존 시스템과 개선된 시스템간의 결과 차이가 크지 않아 데이터를 첨부 하지 않았다. 이를 결과를 통해 RCU를 이용하여 개선한 시스템은 Read access하는 Task가 많은 환경에서 개선율이 더 뛰어남을 알 수 있다. 이를 Graph로 다시 그려보면 아래 그림 28과 같다.

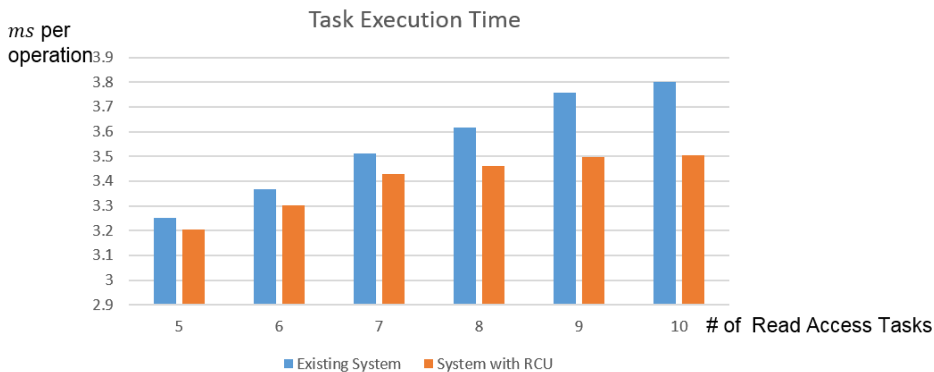


그림 27. Read Access Tasks의 수와 개선율의 상관관계

실험의 세부 데이터는 아래 그림과 같다.

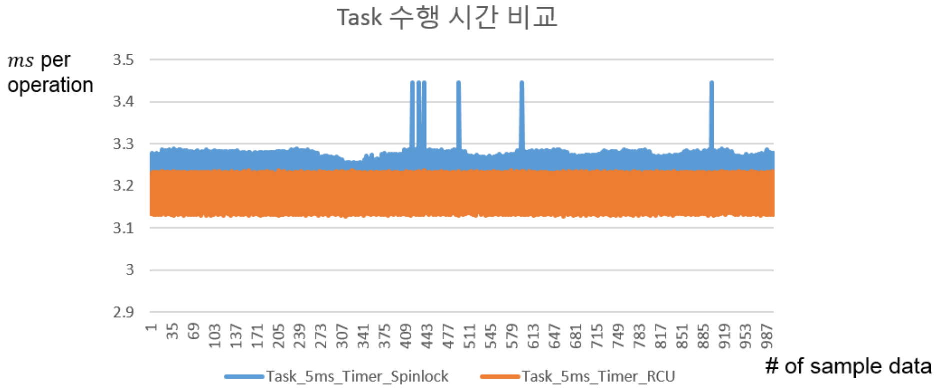


그림 28. Task 수행시간 결과 (Read access Tasks 수: 5)

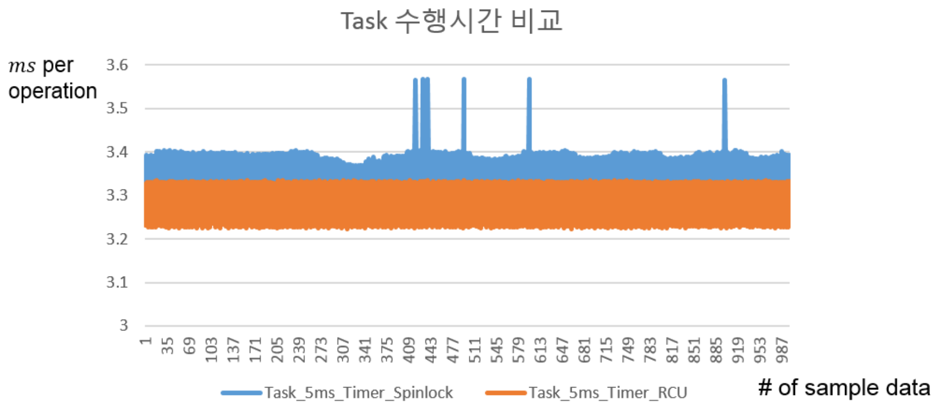


그림 29. Task 수행시간 결과 (Read access Tasks 수: 6)

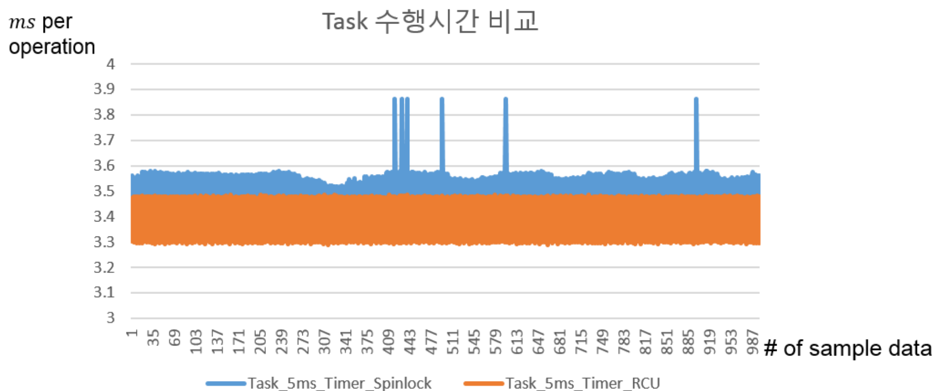


그림 30. Task 수행시간 결과 (Read access Tasks 수: 7)

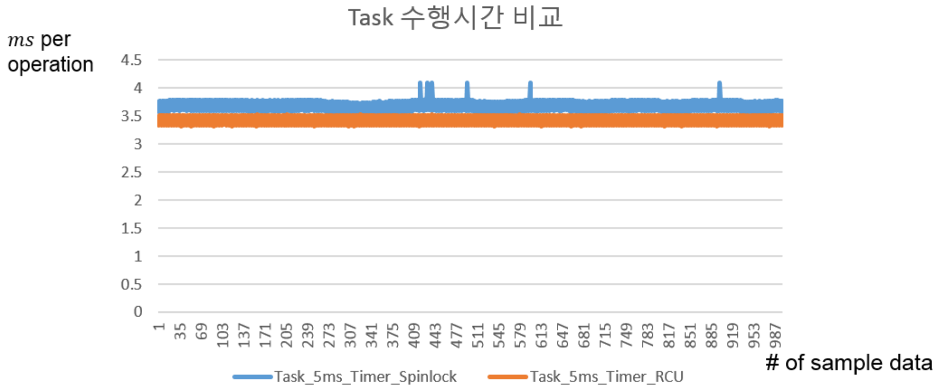


그림 31. Task 수행시간 결과 (Read access Tasks 수: 8)

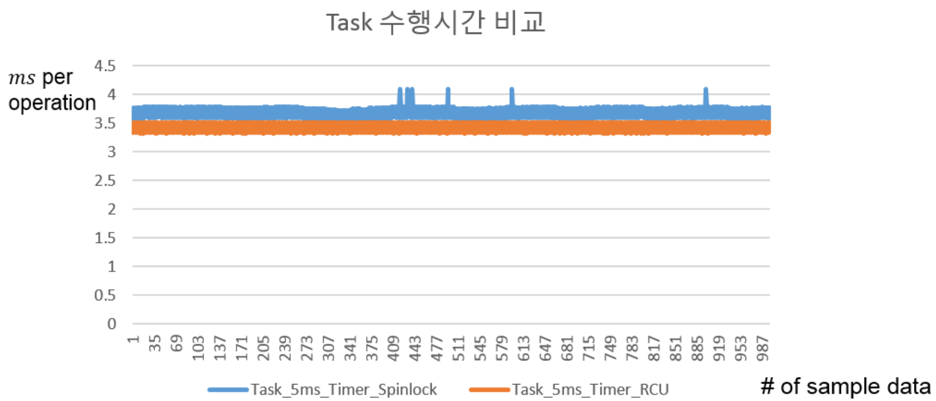


그림 32. Task 수행시간 결과 (Read access Tasks 수: 9)

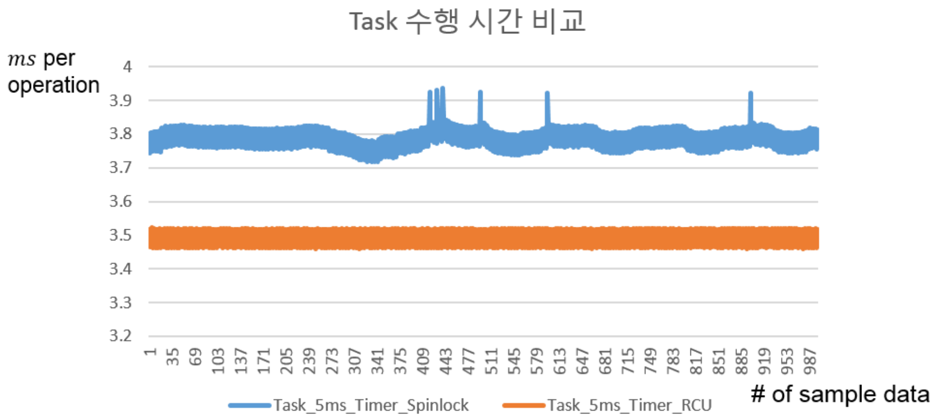
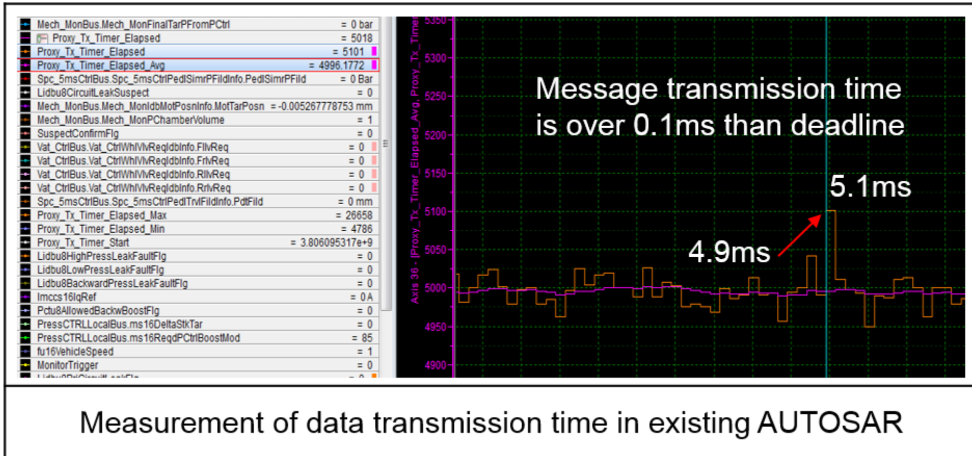


그림 33. Task 수행시간 결과 (Read access Tasks 수: 10)

멀티 코어 AUTOSAR를 기반으로 브레이크시스템 개발 시, CAN 통신

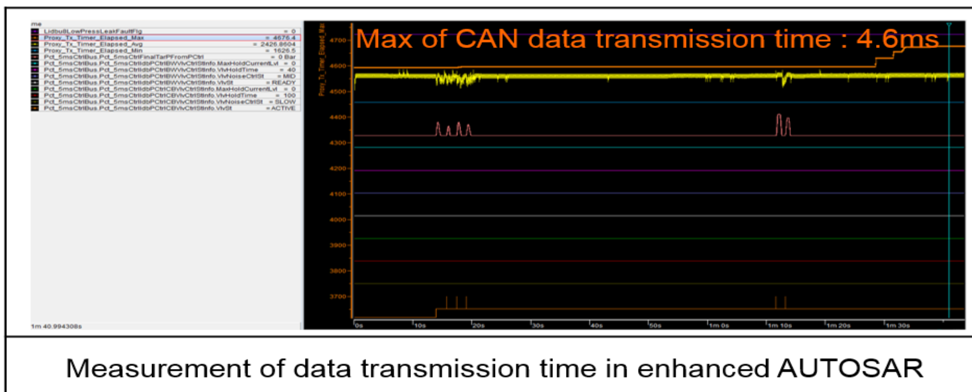
을 이용한 데이터 전달 시간이 Deadline인 5ms를 간헐적으로 초과했던 문제의 개선 여부를 확인하기 위해 아래와 같은 실험을 진행하였다. 그림 26의 실험 모델에서 ECU 외부로 통신하기 위한 CAN 통신 주기를 비교하였다. 기존 시스템에서는 간헐적으로 CAN 통신 주기가 Deadline인 5ms를 넘어서는 경우도 발생하였다.



Measurement of data transmission time in existing AUTOSAR

그림 34. 기존 시스템에서 CAN 데이터 전송 주기 측정

반면에 개선된 시스템에서는 그림 36과 같이 CAN데이터 전송 주기가 Deadline인 5ms 이내로 수렴하는 것을 볼 수 있었다. 아래 그림은 개선된 시스템으로 100초 이상 동작하였을 때, 최대 CAN 데이터 전송주기를 측정하였고 이 값(4.6ms)이 Deadline 5ms 이내임을 확인하였다.



Measurement of data transmission time in enhanced AUTOSAR

그림 35. 개선된 시스템에서 CAN 데이터 전송 주기 측정

제 6 장 결론

이 본문에서는 AUTOSAR가 Multi core를 지원하기 위해 도입한 IOC 통신의 한계점을 밝히고 이를 개선하기 위한 방법과 그 결과를 다뤘다. AUTOSAR IOC에서 통신 지연이 발생하는 이유는 데이터의 동기화 문제를 해결하기 위해 Spinlock이라는 동기화 메커니즘을 채택하였고 이 동기화 기법은 데이터의 동기화를 완벽히 맞추나 한 Task가 자원을 점유 하고 있을 때, 그 자원을 기다리는 동안 Spinning을 하며 다른 Task의 수행이 지연된다는 한계점을 가지고 있다. 특히 이러한 전달 지연은 하나의 데이터를 여러 Tasks 에서 참조하는 환경에서 두드러진다.

대부분의 자동차 시스템은 하나의 데이터를 여러 제어 모듈에서 참조하여 제어 로직을 완성한다. 예를 들어 전자식 브레이크 시스템에서는 하나의 Wheel speed sensor 데이터를 자동차 속도를 계산하는 모듈, ABS 제어를 하는 모듈, ESC 제어를 하는 모듈, AEB 제어를 하는 모듈 등에서 동시에 참조한다. 이런 다수의 Read Access Tasks가 존재하는 자동차 시스템에서는 Spinlock을 이용하는 IOC의 통신으로 지연현상이 극명하게 발생한다. 이런 IOC의 통신 지연을 개선하기 위해 Read Access에 no wait 그리고 low overhead인 RCU (Read-Copy-Update) 동기화 방식을 적용하였다. RCU는 Write Access하는 Task간에는 Block이 발생하지만, Read Access하는 Task간에는 Block없이 데이터를 참조할 수 있다는 측면에서 전달 지연을 개선할 수 있다.

RCU를 적용한 개선된 시스템에서 수행한 Task의 수행시간은 이전의 시스템에서 수행한 Task의 수행시간 보다 약 7.7%(Read Access Tasks 10개 기준) 줄어들었다. 또한 Read Access Tasks의 수가 늘어날수록 개선 비율이 점점 좋아짐을 알 수 있었다.

참고 문헌

[1] SDI-한국리서치 전기차 7개국 소비자 조사, 2016
[2] IHS, CAND 15년 지역별 전기차 판매 실적, 연도별 전기차 시장 전방, 2016
[3] AUTOSAR, "Layered Software Architecture," http://www.autosar.org, 2015.
[4] AUTOSAR, Specifications, http://www.autosar.org.
[5] ISO, ISO 26262 Road vehicles –Functional safety-, http://www.iso.org
[6] AUTOSAR, "Technical Safety Concept Status Concept Report," http://www.autosar.org, 2015
[7] AUTOSAR, Specifications, "AUTOSAR_SWS_OS," http://www.autosar.org, 2015.
[8] AUTOSAR, Specifications, "AUTOSAR_SWS_RTE," http://www.autosar.org, 2015.
[9] AUTOSAR, Specifications, "AUTOSAR_SWS_Com," http://www.autosar.org, 2015.
[10] AUTOSAR, Specifications, "AUTOSAR_SWS_CANNetworkManagement," http://www.autosar.org, 2015
[11] Wang Dafang, "Communication Mechanisms on the Virtual Functional Bus of AUTOSAR,", 2010

Abstract

Reducing the Communication Latency of IOC in AUTOSAR using Read-Copy-Update Mechanism

Hakburm Kim
Graduate School of Convergence Science and Technology
Seoul National University

Recently, electrical and electronic sectors in vehicle are of growing importance because autonomous vehicle, eco-friendly vehicle has emerged in automotive industry. Therefore electric and electronic systems in vehicle is increasing and software complexity also increases with the amount for controlling them. For developing huge and complex software in vehicle, the AUTOSAR (AUTomotive Open Software Architecture) which is an automotive software platform was designed. In particular, a support of muti-core ECU is emerged from the AUTOSAR v4.1.1 in order to meet the functional safety and increase system performance. However, unexpected transmission delay in the IOC (Inter OS-application communicator) which is the communication method in the AUTOSAR occurs resulting in a problem that the performance of the entire system decreases. This paper reveals the problem of communication delay in the IOC and solve it adopting a Read-Copy-Update mechanism in IOC

Keywords : AUTOSAR, IOC, RCU, OS-application, RTE
Student Number : 2015-26043