공학석사 학위논문

AUTOSAR 기반 차량 시스템의 성능 최적화를 위한 러너블-태스크 매핑 규칙

Optimizing the Performance of AUTOSAR-based Automotive System via Runnable-to-Task Mapping Rules

2019 년 2 월

서울대학교 대학원

전기·정보공학부

민 우 영

AUTOSAR 기반 차량 시스템의 성능 최적화를 위한 러너블-태스크 매핑 규칙

Optimizing the Performance of AUTOSAR-based Automotive System via Runnable-to-Task Mapping Rules

지도 교수 홍 성 수

이 논문을 공학석사 학위논문으로 제출함
2018 년 11 월

서울대학교 대학원
전기·정보공학부
민 우 영

민우영의 공학석사 학위논문을 인준함
2018 년 12 월

위 원 장 김 태 환 (인)

부위원장 홍 성 수 (인)

위 원 심 규 석 (인)

초 록

자동차가 점차 전장화됨에 따라 차량용 소프트웨어의 크기와 복잡도가 크게 증가하고 있다. 이 때문에 차량용 소프트웨어의 개발에 소요되는 시간과 비용 또한 증가하여 유럽의 주요 자동차 회사들은 개발의 효율성을 높이고자 AUTOSAR(AUTomotive Open System ARchitecture) 표준을 제정하였다. AUTOSAR 표준은 차량용 소프트웨어의 아키텍처와 개발 과정을 정의한 표준으로써 현재 많은 자동차 회사들에서 이를 준수하여 제품을 개발하고 있다.

AUTOSAR 표준에 따른 응용 소프트웨어는 소프트웨어 컴포넌트(software component) 단위로 모듈화되어 설계되며 각각의 소프트웨어 컴포넌트는 자신의 기능을 구현하는 러너블(runnable)을 1개 이상 갖는다. 개발자는 러너블을 동작시키기 위해 운영체제의 스케줄링 단위인 태스크에 매핑하는데, 러너블-태스크 매핑에 따라 시스템 오버헤드 발생량이 크게 달라지므로 이는 시스템 성능 측면에서 매우 중요한 작업이다.

본 학위 논문에서는 자율주행을 수행하는 타겟 응용의 성능 최적화를 위해 기존 연구에서 제안한 6개의 러너블-태스크 매핑 규칙을 적용하며, 추가적인 성능 향상을 위해 기존 규칙을 개선한 매핑 규칙을 제안한다. 제안된 규칙을 적용하여 매핑했을 때와 개발자가 임의로 매핑했을 때 타겟 응용의 성능을 실험을 통해 비교하며, Infineon 사의 AURIX 보드와 ETAS 사의 AUTOSAR 플랫폼 상에서 타겟 응용을 구현하여 실험하였다. 실험 결과 제안된 규칙을 적용하여 매핑했을 때 타겟 응용의 종단 간 응답 시간(end-to-end response time)이 개발자가 임의로 매핑했을 때의 기댓값보다 약 1.49배 짧은 것으로 확인되었다.

주요어 : AUTOSAR, 차량용 소프트웨어, 러너블-태스크 매핑, 매핑 규칙, 종단 간 응답 시간
학 번 : 2017-26663

목 차

표 목차

그림 목차

제 1 장 서 론

최근 자동차가 점차 전장화됨에 따라 고급 자동차에는 100개 가량의 전자 제어 장치(Electronic Control Unit. 이후 ECU)가 탑재되고 있으며 차량 내 많은 기능들이 ECU 상에서 소프트웨어적으로 구현되고 있다[1]. 이에 따라 차량용 소프트웨어의 중요도가 높아지고 있고 코드의 크기와 복잡도가 또한 크게 증가하고 있다. 기존의 차량용 소프트웨어 개발은 각각의 자동차 부품 회사들이 독자적으로 자신의 ECU 규격에 맞추어 소프트웨어를 구현하는 방식으로 이루어졌는데, 이러한 방식으로는 하드웨어에 대한 의존성 때문에 한 ECU에서 개발한 소프트웨어를 다른 ECU에서 재사용할 수 없으며 소프트웨어를 부분적으로 교체하거나 유지보수하는 데에 많은 시간과 비용이 소요된다.

따라서 유럽의 주요 자동차 회사들은 차량용 소프트웨어의 아키텍처와 개발 과정을 표준화하여 차량용 소프트웨어의 개발 기간과 비용을 절감하고 효율성을 극대화시키고자 하였다. 이를 위해 2003년에 BMW, Bosch, Continental, Daimler, Ford, Volkswagen 등을 중심으로 협력체가 형성되었고 AUTOSAR(AUTomotive Open System ARchitecture)라는 차량용 소프트웨어 아키텍처 표준을 제정하였다. AUTOSAR는 계층적 구조를 통해 소프트웨어를 하드웨어로부터 독립적으로 개발할 수 있도록 하며 소프트웨어를 모듈화하여 재사용성, 확장성, 그리고 독립성을 높인다[2]. 현재 70여 개의 자동차 회사들이 AUTOSAR의 파트너 사로 참여하고 있고 세계의 주요 자동차 회사들이 AUTOSAR 표준을 준수하여 제품을 개발하고 있다.

AUTOSAR에 따르면 응용 소프트웨어는 소프트웨어 컴포넌트(software component) 단위로 모듈화되어 설계되며 소프트웨어 컴포넌트의 기능은 각 소프트웨어 컴포넌트가 가지는 러너블(runnable)을 통해 구현된다. 러너블은 개발자가 입력하는 일련의 코드를 가진 함수로써 운영체제의 스케줄링 단위인 태스크에 매핑되어 실행된다. 이때, 러너블-태스크 매핑은 AUTOSAR 기반 차량 시스템의 성능을 좌우하는 매우 중요한 작업이다. 몇 개의 태스크를 생성하고 각각에 어떤 러너블을 매핑하는지에 따라 시스템 오버헤드의 발생량이

달라지며 과중한 오버헤드는 시스템의 성능 저하로 직결된다. 하지만 이 작업은 온전히 개발자의 역할이며 AUTOSAR 표준에서는 매우 기본적인 가이드[3]만 제공하므로 개발자가 무수히 많은 매핑의 경우 중 시스템 성능 측면에서 효율적인 매핑을 찾기란 매우 어렵다.

이러한 문제를 해결하고자 개발자가 효율적인 러너블-태스크 매핑을 찾을 수 있도록 지원하는 연구들이 많이 진행되었다. 기존 연구는 개발자에게 효율적인 매핑을 제공하기 위한 연구[4, 5, 6, 7]와 개발자가 직접 효율적인 매핑을 찾을 수 있도록 가이드하는 연구[8, 9, 10, 11]로 분류된다. 전자의 연구들은 개발자가 타겟 응용의 정보와 제약사항을 입력하면 이를 반영하여 오버헤드가 적은 러너블-태스크 매핑을 찾는 매핑 알고리즘을 제안하며, 후자의 연구들은 적은 오버헤드를 갖기 위해 개발자가 따라야 하는 매핑 규칙을 제안한다. 타겟 응용이 복잡하지 않다면 매핑 알고리즘을 설정, 실행하는데 필요한 준비 과정 때문에 매핑 규칙을 사용하는 것이 적합하며, 타겟 응용이 복잡하다면 개발자가 직접 매핑하기 어렵기 때문에 매핑 알고리즘을 사용하는 것이 적합하다.

본 학위 논문에서는 운전자의 개입 없이 차량이 주행할 수 있도록 제어하기 위한 ADAS(Advanced Driver Assistance System) 응용에 대하여 시스템 성능 측면에서 효율적인 매핑을 찾고자 하며, 타겟 응용의 복잡도가 높지 않으므로 매핑 규칙을 활용한다. 매핑 규칙을 제안한 연구 중 [11]은 AUTOSAR 표준에 명시된 러너블의 속성에 따라 태스크에 매핑하도록 6개의 규칙을 제안하며 이는 [8, 9, 10]에서 제안한 규칙을 일부 포함하고 있다. 따라서 본 학위 논문에서는 [11]의 규칙을 적용하여 타겟 응용의 러너블-태스크 매핑 작업을 수행하며 추가적인 성능 향상을 위해 기존 규칙을 개선한 매핑 규칙을 제안한다.

제안된 규칙을 적용하여 매핑한 타겟 응용을 Infineon 사의 AURIX 보드와 ETAS 사의 AUTOSAR 플랫폼 상에 구현한 결과 종단 간 응답 시간(end-to-end response time)이 개발자가 임의로 매핑한 경우의 기댓값보다 약 1.49배 단축됨을 확인하였다.

본 학위 논문은 앞으로 다음과 같이 진행된다. 2장에서는 논문의 이해를 돕기 위해 AUTOSAR에 대한 배경지식을 설명한 후 본 학위

논문과 관련된 기존 연구들을 소개한다. 3장에서는 본 학위 논문에서 기저로 삼은 러너블-태스크 매핑 규칙에 대해 정리한 후 이를 바탕으로 확장된 규칙과 추가된 규칙을 제안한다. 4장에서는 타겟 응용에 대해 자세히 설명하고 매핑 규칙을 적용하여 러너블-태스크 매핑 작업을 수행한다. 5장에서는 개발자가 임의로 매핑한 경우와 매핑 규칙을 따른 경우에 대해 타겟 응용의 종단 간 응답 시간을 실험을 통해 비교하며 평가한다. 6장에서는 본 학위 논문의 내용을 요약하고 결론을 맺는다.

제 2 장 배경지식과 관련 연구

이 장에서는 본 학위 논문의 이해를 돕기 위해 AUTOSAR에 대한 배경지식을 설명한 후 본 학위 논문과 관련된 기존의 연구들을 정리한다.

제 1 절 AUTOSAR

이 절에서는 1.1에서 AUTOSAR 전반에 대해 개괄적으로 설명한 후 1.2에서 AUTOSAR의 수행 모델에 대해 설명하고 1.3에서 AUTOSAR의 통신에 대해 설명한다.

1.1 AUTOSAR 개괄

AUTOSAR는 자동차 내부의 전자 제어 장치인 ECU에 탑재되는 차량용 소프트웨어의 표준이다. 이는 자동차가 전장화됨에 따라 복잡해지고 양적으로 증가하는 차량용 소프트웨어의 재사용성을 높이고 하드웨어 구조로부터 독립적으로 개발될 수 있게 함으로써 개발 기간과 비용은 축소하고 효율성을 향상시키기 위해 제정되었다. 2003년에 유럽의 주요 자동차 회사인 BMW, Bosch, Continental, Daimler, Ford, Volkswagen 등을 주축으로 AUTOSAR 협력체가 처음 형성되었으며 이 협력체에서 차량용 소프트웨어의 아키텍처, 개발방법론, 인터페이스 등을 담은 AUTOSAR 표준을 정의하였다. 2004년 표준의 첫 버전을 발표한 이후 2006년 버전 2.1에서 AUTOSAR를 구성하는 주요 표준들이 완성되었으며, 버전 4.0에 멀티코어 지원 등 새로운 요소들이 다수 추가된 후 계속된 유지보수 및 개선을 통해 2018년 현재 4.4 버전이 공개되어 있다.

AUTOSAR는 하드웨어 구조와 독립적인 소프트웨어 개발을 위해 계층적 구조(layered architecture)를 사용한다. AUTOSAR의 계층적 구조는 그림 1과 같이 세 계층으로 구분된다.

- 응용 계층(application layer)
- 런타임 환경(run-time environment. 이후 RTE)
- 기본 소프트웨어(basic software. 이후 BSW)

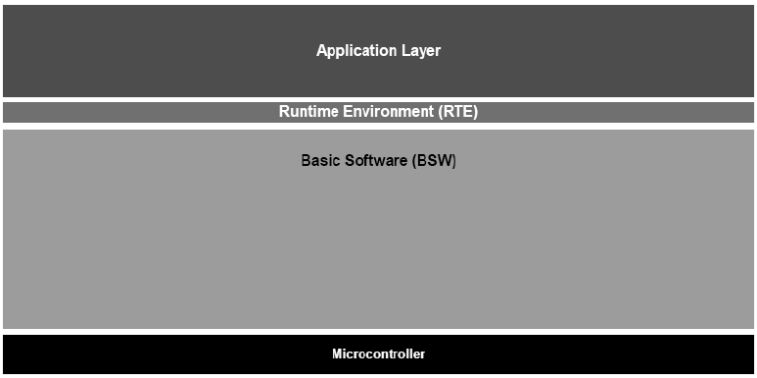

그림 1. AUTOSAR의 계층적 구조

먼저 응용 계층은 개발자가 원하는 차량용 소프트웨어의 기능을 수행하는 응용 소프트웨어를 구현하는 계층이다. 응용 소프트웨어는 소프트웨어 컴포넌트 단위로 모듈화되어 설계되는데, 소프트웨어 컴포넌트 간의 독립적인 개발을 위해 내부 기능은 드러내지 않은 채 포트(port)를 통해 통신 인터페이스(interface)만 드러낸다. 소프트웨어 컴포넌트끼리는 VFB(Virtual Functional Bus)라는 가상의 통신 버스를 통해 미리 정의된 통신 인터페이스의 내용대로 통신할 수 있다는 가정 하에 개발된다.

RTE는 위 계층인 응용 계층에 존재하는 응용 소프트웨어가 수행할 수 있는 런타임 환경을 제공하며, 특히 소프트웨어 컴포넌트 간의 통신 서비스를 제공한다. 앞서 설명한 VFB의 구현 형태가 바로 RTE이다. 이 계층을 이용하여 각 소프트웨어 컴포넌트는 통신하고자 하는 소프트웨어 컴포넌트가 어느 ECU에 있는지 고려할 필요없이 개발할 수 있다.

BSW는 다시 세 개의 세부 계층으로 구분된다. 이는 그림 2과 같이 서비스 계층(service layer), ECU 추상 계층(ECU abstraction layer), 마이크로컨트롤러 추상 계층(Microcontroller abstraction layer)이다. 서비스 계층은 응용 계층으로 운영체제의 기능, 통신, 메모리, ECU 상태 관리 등의 기능을 제공한다. ECU 추상 계층은 마이크로컨트롤러와 주변 장치에 접근하기 위한 인터페이스로써, 표준화된 API를 제공한다. 마이크로컨트롤러 추상 계층은 마이크로컨트롤러와 주변 장치에 대한 디바이스 드라이버를 포함한다. BSW는 BSW 모듈로 구현되며 특히 서비스 계층의 모듈은 소프트웨어 컴포넌트 형태로 응용 계층에 제공되어 다른 소프트웨어 컴포넌트에서 접근할 수 있다.

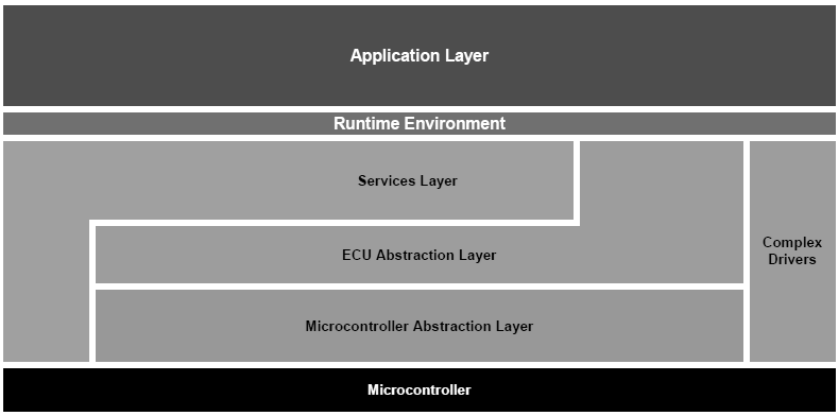

그림 2. AUTOSAR의 세부 계층적 구조

AUTOSAR의 개발 과정을 간략하게 설명하면 그림 3과 같다. 먼저 개발자가 VFB 상에서 소프트웨어 컴포넌트를 설계한다. 이후 소프트웨어 컴포넌트들을 어느 ECU에 수행할 지 결정하고 AUTOSAR 개발 도구에 소프트웨어 컴포넌트의 명세와 ECU, 시스템 설정의 명세를 입력하여 각 ECU마다 할당된 소프트웨어 컴포넌트들을 위한 RTE를 생성한다. BSW 또한 개발자가 AUTOSAR 개발 도구를 통해 각 ECU마다 어떤 BSW 모듈을 사용할지 결정하고 각 모듈을 설정한 후 생성한다.

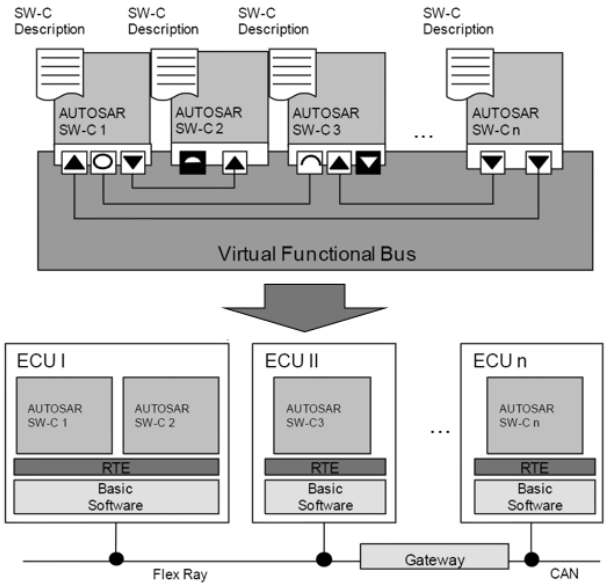

그림 3. 소프트웨어 컴포넌트의 ECU 매핑과 RTE 생성

1.2 AUTOSAR의 수행 모델

1.1장에서 설명한 것처럼 AUTOSAR의 응용 소프트웨어는 VFB 상에서 소프트웨어 컴포넌트 단위로 모듈화하여 설계된다. VFB 상에서 소프트웨어 컴포넌트는 외부와의 통신을 위해 포트를 정의하고 포트에 인터페이스를 설정하며, 내부 기능을 위해 러너블(runnable)과 RTE 이벤트(RTE event)를 설정한다. 포트는 소프트웨어 컴포넌트의 데이터 출입구를 나타내며 각 포트에는 통신 규약을 나타내는 인터페이스가 설정된다. 러너블은 소프트웨어 컴포넌트의 수행 주체로써 RTE 이벤트에 의해 작동하는 일련의 코드이다. 러너블은 WaitPoint의 유무에 따라 카테고리 1 러너블과 카테고리 2 러너블로 구분되는데, WaitPoint가 없는 카테고리 1 러너블은 수행 도중 수행을 멈출 수 없으며, WaitPoint를 가진 카테고리 2 러너블은 수행 도중 특정 RTE 이벤트가 발생하기를 기다리기 위해 수행을 멈출 수 있다. RTE 이벤트는 특정 상황에 발생하여 러너블을 작동시키는 주체으로써, 대표적으로 설정된 주기마다 발생하는 TimingEvent, 설정된 포트로 데이터가 들어왔을 때 발생하는 DataReceivedEvent, 설정된 포트로 서비스 요청이 들어왔을 때 발생하는 OperationInvokedEvent 등이 있다. 이러한 RTE 이벤트는 러너블을 작동시킬 수도 있고 WaitPoint에서 깨울 수도 있다.

이렇게 VFB 상에서 설계된 사항들은 소프트웨어 컴포넌트가 ECU에 매핑된 후 각 ECU에 필요한 부분을 반영하여 RTE로 구현된다. 러너블은 함수로 구현되어 개발자가 내부 코드를 작성하고 VFB에 설계된 통신은 RTE API로 생성되어 러너블의 코드 내에서 호출된다. 러너블은 운영체제의 스케줄링 단위인 태스크에 매핑되어 함수 호출 형태로 실행된다. 태스크는 가질 수 있는 상태에 따라 두 종류로 구분되는데, 먼저 기본 태스크(basic task)는 정지 상태(suspended state), 준비 상태(ready state), 실행 상태(running state)를 가지며 확장 태스크(extended task)는 여기에 추가로 대기 상태(waiting state)를 갖는다. 정지 상태에 있는 태스크는 내부의 러너블을 작동시키는 RTE 이벤트가 발생하면 활성화되어 준비 상태가 된 후 운영체제에 의해 스케줄링되면 실행 상태가 된다. 이후 수행을 모두 마치면 다시 정지 상태가 된다. 확장 태스크는 WaitPoint를 가진 러너블을 포함하는 태스크로써, 실행 상태에서 특정 RTE 이벤트를

기다리기 위해 대기 상태가 될 수 있다. 이후 기다리던 RTE 이벤트가 발생하여 태스크가 깨어나면 준비 상태로 돌아온다. 그림 4는 기본 태스크와 확장 태스크가 갖는 상태를 나타낸 그림이다.

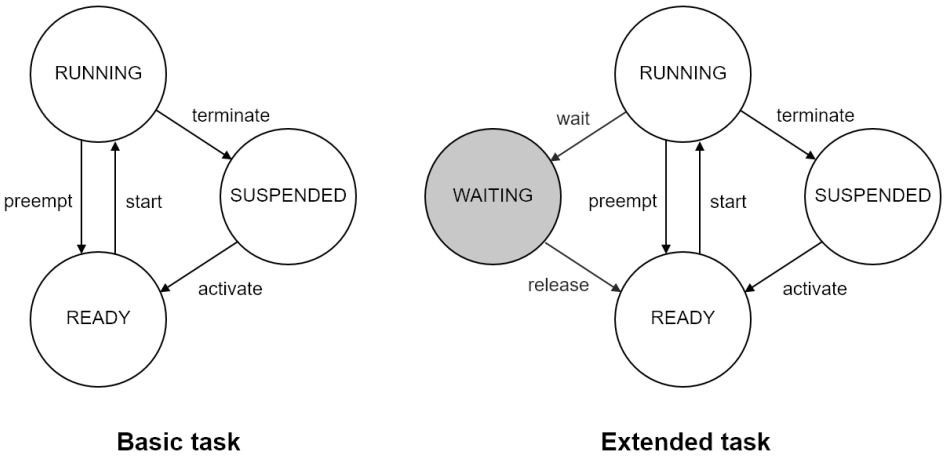

그림 4. 기본 태스크와 확장 태스크가 갖는 상태 비교

한 태스크에 여러 러너블이 매핑될 수 있으며, 이 경우 수행 순서를 설정해야 한다. 한 태스크에 매핑된 러너블들은 RTE 이벤트의 발생에 따라 선택적으로 호출된다. 또한 각 태스크는 우선순위(priority)를 가지며 운영체제는 준비 상태의 태스크 중 우선순위가 가장 높은 태스크를 스케줄링하여 수행시킨다. 현재 수행 중인 태스크가 정지 상태가 되거나 대기 상태가 되면 그 다음 높은 우선순위를 가진 태스크를 수행시킨다. 같은 우선순위의 태스크들은 먼저 준비 상태가 된 태스크부터 수행된다.

1.3 AUTOSAR의 통신

AUTOSAR에서는 소프트웨어 컴포넌트 간 통신을 표준화하며 두 종류로 구분된다. 이는 데이터를 단방향으로 전달하기 위한 송수신 통신(sender-receiver communication)과 미리 정의된 서비스를 요청하고 결과를 전달받기 위한 클라이언트-서버 통신(client-server communication)이다. AUTOSAR는 송수신 통신에 두 가지 방법을 지원하는데 이는 명시적(explicit) 통신과 암시적(implicit) 통신이다. 명시적 통신은 러너블 내에서 데이터를 송수신하는 RTE API를

호출하는 시점에 데이터가 전달되는 방법이며, 암시적 통신은 RTE API의 호출 시점과 상관없이 데이터의 송신은 러너블의 수행이 끝난 후 일어나고 데이터의 수신은 러너블의 수행 직전에 일어나는 방법이다. 클라이언트-서버 통신 또한 두 가지 방법이 있는데, 동기적(synchronous) 통신과 비동기적(asynchronous) 통신이다. 동기적 통신은 클라이언트 러너블에서 서비스를 요청한 뒤 결과가 오기까지 수행을 멈춘 후 기다린다. 이후 서버 러너블에서 서비스에 대한 결과를 반환하면 다시 수행을 재개한다. 비동기적 통신은 클라이언트 러너블에서 서비스를 요청한 후 수행을 멈추지 않고 이어나가며 서비스에 대한 결과가 필요할 때 RTE API를 호출하여 결과를 반환 받는다.

VFB 상에서 소프트웨어 컴포넌트 간에 어떤 통신을 하는지는 인터페이스를 통해 결정되지만 통신의 방법은 러너블의 통신 동작을 나타내는 속성을 부여함으로써 결정된다. 송수신 통신을 하는 러너블일 경우 DataAccessPoint를, 클라이언트-서버 통신을 하는 러너블일 경우 ServerCallPoint를 설정해야 한다. DataAccessPoint에는 DataSendPoint, DataReceivePoint, DataWriteAccess, DataReadAccess 4가지 속성이 있다. 명시적 통신의 송신 러너블은 DataSendPoint를, 수신 러너블은 DataReceivePoint를 가져야 하며 암시적 통신의 송신 러너블은 DataWriteAccess를, 수신 러너블은 DataReadAccess를 가져야 한다. ServerCallPoint에는 SynchronousServerCallPoint, AsynchronousServerCallPoint, AsynchronousServerCallResultPoint 3가지 속성이 있다. 동기적 통신의 클라이언트 러너블은 SymchronousServerCallPoint를 가져야 하며 비동기적 통신의 클라이언트 러너블은 AsynchronousServerCallResultPoint를 가지고 서버 러너블은 AsynchronousServerCallPoint를 가져야 한다.

WaitPoint는 특정 RTE 이벤트를 받기 위해 러너블의 수행을 멈추게 하는 속성이다. 예를 들어 송수신 통신의 수신 러너블에 DataReceivePoint와 WaitPoint를 함께 부여하면 데이터를 수신하기 위한 RTE API를 호출했을 때 데이터가 들어오면 발생하는 DataReceivedEvent가 발생할 때까지 수행을 멈춘다. 또 클라이언트 러너블에 AsynchronousServerCallResultPoint와 WaitPoint를 함께

부여하면 서비스의 결과를 반환 받고자 RTE API를 호출했을 때 서버 러너블에서 결과를 아직 처리하지 못했다면 결과를 받을 때 발생하는 AsynchronousServerCallReturnsEvent가 발생할 때까지 수행을 멈춘다. AUTOSAR 개발 도구는 이렇게 소프트웨어 컴포넌트에 설정된 인터페이스와 러너블에 부여된 속성을 참조하여 RTE API를 구현한다.

제 2 절 관련 연구

이 절에서는 AUTOSAR 기반 응용 소프트웨어의 러너블-태스크 매핑을 지원하기 위한 기존 연구들을 소개한다. 기존 연구는 개발자에게 효율적인 매핑을 제공하기 위해 매핑 알고리즘을 제안하는 연구[4, 5, 6, 7]와 개발자가 직접 효율적인 매핑을 찾을 수 있도록 매핑 규칙을 제안하는 연구[8, 9, 10, 11]로 분류된다.

2.1 매핑 알고리즘을 제안하는 연구

[4]의 저자는 AUTOSAR 기반 응용 소프트웨어의 응답 시간을 최소화하고 메모리와 버스 사용량을 최소화하기 위한 러너블-태스크 매핑 알고리즘을 제안한다. [4]의 알고리즘은 ILP(Integer Linear Program)를 이용하며 모든 러너블의 정보, 고려해야 하는 제약 사항, 목적 함수(objective function)를 수식화한 후 CPLEX라는 ILP 해결 알고리즘과 유전 알고리즘을 통해 목적 함수를 최소화하면서 제약 사항을 모두 만족하는 매핑을 탐색한다. [5]의 저자는 응용의 메모리 사용량을 최소화시키기 위해 [4]와 마찬가지로 ILP를 이용하고 CPLEX 알고리즘을 통해 매핑을 찾는다. [4]와의 차이점은 태스크 내 러너블의 수행 순서, 태스크의 우선순위, 태스크 간 동기화 기법 등을 추가로 고려한다. [6]의 저자는 주어진 하드웨어에서 최대한 많은 러너블을 수용할 수 있도록 매핑하는 알고리즘을 제안한다. 이는 모든 러너블이 주기적이고 태스크는 한 개만 있다는 가정 하에 러너블의 주기와 모든 러너블의 최소공배수, 최대공약수를 입력으로 받아 러너블들의 주기와 로드를 고려하여 태스크 내에서 수행될 순서를 결정하는 휴리스틱 알고리즘이다. [7]의 저자는 러너블들의 통신 지연시간을 최소화하는 러너블-태스크 매핑을 제안한다. 이는 각 러너블이 노드가 되고 서로 통신하는 러너블끼리 연결한 러너블들의 상호작용 그래프(runnable interaction graph)를 입력으로 받아 그래프 내에서 사이클(cycle)이

생기지 않는 한 서로 통신하는 러너블들을 함께 매핑하는 휴리스틱 알고리즘이다.

매핑 알고리즘을 사용하면 많은 경우의 매핑을 탐색할 수 있고 알고리즘이 대신 매핑을 찾아준다는 장점이 있지만 개발자가 주어진 형식에 맞추어 입력을 넣고 실제로 알고리즘을 실행해야 한다는 단점이 있다. 이 때문에 타겟 응용이 개발자가 직접 매핑할 수 없을 만큼 복잡할 경우에는 효과적이지만 그렇지 않은 타겟 응용에 적용하기에는 적합하지 않다.

2.2 매핑 규칙을 제안하는 연구

[8, 9]의 저자는 응용의 응답 시간을 최소화하기 위해 같은 주기로 실행되는 러너블들을 같은 태스크에 매핑해야 한다는 규칙을 제안한다. 이는 가장 간단한 매핑 규칙으로써 산업계에서 실제로 많이 사용되는 규칙이다. [10]의 저자는 응용의 종단 간 응답 시간을 최소화하기 위해 러너블 간 통신 지연을 최소화하고자 두 개의 규칙을 제안하는데 첫 번째는 종단 간 응답 시간에 제약을 가지고 있으면서 서로 데이터를 주고 받는 일련의 러너블들을 한 태스크에 매핑하는 규칙이고 두 번째는 태스크들 중 같은 주기로 수행되는 태스크가 서로 통신한다면 이를 합병하는 규칙이다. [11]의 저자는 응용의 종단 간 응답 시간을 최소화하기 위해 러너블 간 통신 지연을 최소화하고자 6개의 매핑 규칙을 제안한다. [11]의 규칙은 러너블의 통신 동작을 나타내는 속성을 기준으로 제시되어 개발자가 적용하기 용이하며, [8, 9, 10]의 규칙을 일부 포함한다. 따라서 본 학위 논문에서는 [11]를 기저로 삼아 타겟 응용의 러너블-태스크 매핑에 적용한다.

제 3 장 러너블-태스크 매핑 규칙

이 장에서는 본 학위 논문에서 기저로 삼은 [11]의 6개 매핑 규칙에 대해 설명한 뒤 이를 개선한 매핑 규칙을 제안한다.

제 1 절 6가지 매핑 규칙 설명

[11]의 저자는 러너블-태스크 매핑으로 인해 응용의 응답 시간을 지연시키는 3가지 요소를 고려하여 6개의 매핑 규칙을 제시한다. 3가지 요소는 러너블들이 많은 수의 태스크로 분산되어 생기는 문맥 교환 오버헤드와 동기화 오버헤드, 그리고 러너블들이 같은 태스크에 매핑됨으로써 발생 가능한 러너블의 수행 지연이다. 문맥 교환 오버헤드란 CPU에서 수행되는 태스크가 변경되어 기존의 태스크의 문맥을 저장하고 변경될 태스크의 문맥을 불러오는데 소요되는 오버헤드이고, 동기화 오버헤드란 태스크 간 공유되는 데이터에 대한 일관성(consistency)을 보장하고자 데이터에 접근하는 코드 영역을 상호 배제적(mutually exclusive)으로 만들기 위해 소요되는 오버헤드이다. AUTOSAR 운영체제는 코어 내 태스크 동기화를 위해 세마포어(semaphore)를, 코어 간 태스크 동기화를 위해 스핀락(spinlock)을 제공한다.

6개의 규칙은 아래와 같다.

> R.1) 한 러너블이 하나의 DataSendPoint를 가지며 이에 상응하는 수신 러너블이 하나의 DataReceivePoint를 가진다면, 이 둘은 같은 태스크에 매핑.

R.1은 그림 5와 같이 데이터를 송신하는 러너블과 수신하는 러너블을 한 태스크에 매핑한다. 이를 통해 두 러너블이 다른 태스크에 매핑됨으로써 발생하는 문맥 교환 오버헤드와 두 러너블 간에 전달되는 데이터에 접근하기 위한 동기화 오버헤드를 제거한다.

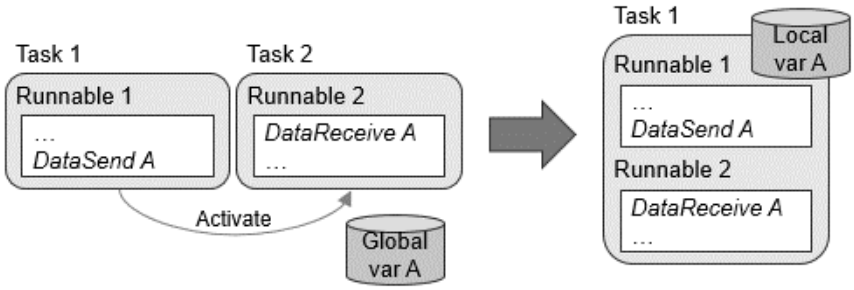

그림 5. R.1의 예시

R.2) 둘 이상의 러너블들이 같은 데이터를 받기 위해 하나의 DataReceivePoint만을 가진다면, 이들은 같은 태스크에 매핑.
R.3) 둘 이상의 러너블들이 같은 데이터를 받기 위해 하나의 DataReadAccess만을 가진다면, 이들은 같은 기본 태스크에 매핑.

R.2와 R.3은 그림 6과 같이 같은 데이터를 수신하는 러너블들을 한 태스크에 매핑한다. 이를 통해 같은 데이터를 수신하는 러너블들이 서로 다른 태스크에서 수행되어 발생하는 문맥 교환 오버헤드를 제거한다. R.2와 R.3의 차이점은 DataReceivedPoint는 WaitPoint와 함께 사용될 수 있어 기본 태스크 또는 확장 태스크에 매핑될 수 있지만 DataReadAccess는 WaitPoint와 함께 사용될 수 없어 기본 태스크에 매핑하도록 한정한 것이다.

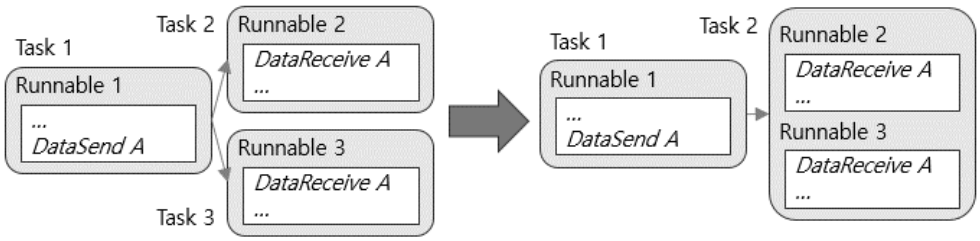

그림 6. R.2와 R.3의 예시

R.4) 같은 주기의 TimingEvent로 인해 작동하는 둘 이상의 러너블들이 WaitPoint 없이 모두 동일한 러너블에게 데이터를 보낸다면, 모두 같은 기본 태스크에 매핑.

R.4는 그림 7과 같이 같은 주기로 실행되며 같은 데이터를

송신하는 러너블들에 대하여, 문맥 교환 오버헤드 때문에 데이터 송신이 늦어지는 것을 막기 위해 이들을 같은 태스크에 매핑하는 규칙이다.

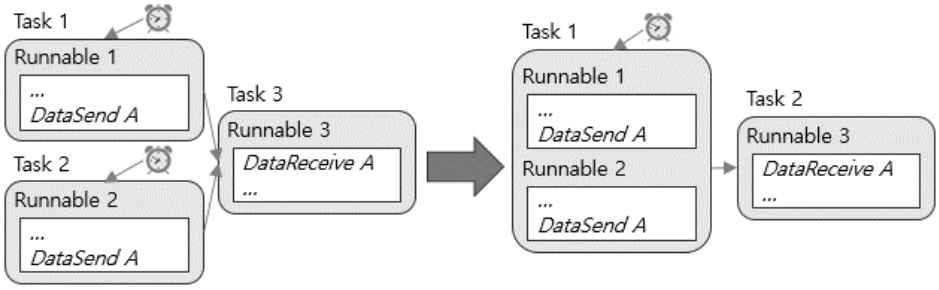

그림 7. R.4의 예시

R.5) canBeInvokedConcurrently 속성이 true이고 WaitPoint가 없는 서버 러너블은 클라이언트 러너블들에서 직접 호출될 수 있도록 태스크에 매핑하지 않음.

R.5는 여러 클라이언트 러너블에서 서비스를 요청했을 때 이를 병렬적으로 처리해도 무관한 서버 러너블에 대하여, 그림 8과 같이 클라이언트 러너블에서 서버 러너블로 서비스 요청을 보낸 후 결과를 받는 것이 아니라 클라이언트 러너블 내에서 직접 서버 러너블을 호출할 수 있도록 하여 문맥 교환 오버헤드를 줄이는 규칙이다.

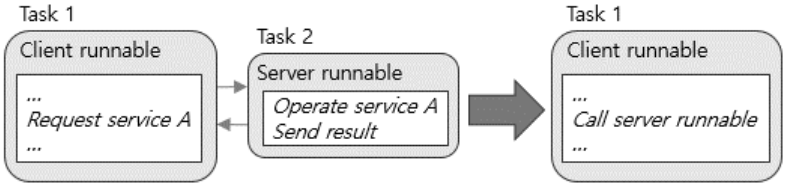

그림 8. R.5의 예시

R.6) 서로 다른 WaitPoint를 가진 러너블들은 같은 태스크에 매핑하지 않음.

서로 다른 WaitPoint를 가진 러너블들이 같은 태스크에 매핑될 경우 한 WaitPoint로 인해 태스크의 수행이 멈추면 다른 WaitPoint를 가진 러너블은 기다리는 이벤트가 발생하여도 수행을 재개할 수 없고

수행이 지연된다. R.6은 이러한 지연을 막고자 하는 규칙이다.

제 2 절 매핑 규칙의 개선

이 절에서는 응용의 시스템 오버헤드를 더욱 줄이기 위해 기존 규칙 중 R.4와 R.6을 확장하고 R.7을 추가한다.

R.4는 같은 주기로 같은 데이터를 송신하는 러너블이 다수 존재할 때 문맥 교환 오버헤드로 인한 데이터 송신의 지연을 막기 위해 이들은 한 태스크에 매핑하는 규칙이다. 하지만 이 문맥 교환 오버헤드는 데이터를 송신하는 러너블이 아니라 같은 주기로 실행되는 모든 러너블에도 발생하며, 이 때문에 러너블의 수행 시작이 지연된다. 따라서 아래와 같이 R.4를 확장한 R.4′ 을 제안한다. 만일 R.4′ 은 TimingEvent에 의해 작동되는 모든 러너블에 적용되는 규칙이므로 다른 규칙들과 상충될 수 있는데, 이 경우 다른 규칙을 우선적으로 적용한다.

> R.4′) 둘 이상의 러너블이 같은 주기로 발생하는 TimingEvent에 의해 작동한다면, 이들은 모두 같은 태스크에 매핑.

R.6은 한 WaitPoint에서 기다리는 이벤트가 발생하더라도 다른 WaitPoint 때문에 러너블이 수행을 재개할 수 없어 생기는 지연을 막기 위한 규칙이다. 하지만 이러한 지연은 한 WaitPoint에서 기다리는 이벤트가 반드시 다른 WaitPoint 이후에 발생한다면 생기지 않는다. 이러한 경우 이들을 다른 태스크에 매핑하면 불필요한 문맥 교환 오버헤드가 생기므로 아래와 같이 R.6을 확장하여 R.6′ 을 제안한다.

> R.6′) 서로 다른 WaitPoint를 가진 러너블들은 같은 태스크에 매핑하지 않음. 단, 한 러너블의 WaitPoint에서 기다리는 이벤트가 반드시 다른 러너블의 WaitPoint 이후 발생한다면 함께 매핑 가능.

또한 태스크의 수행이 WaitPoint 때문에 멈춰 있다면, 이미 자신을 작동시키는 이벤트가 발생하여 수행 가능한 상태인 러너블들은

모두 지연된다. 따라서 수행 가능한 러너블이 있음에도 불구하고 WaitPoint에 의해 태스크가 대기 상태에 빠지지 않도록 아래와 같이 R.7을 제안한다.

> R.7) WaitPoint를 가진 러너블이 같은 태스크 내 존재하는 다른 러너블보다 선행될 필요가 없다면 수행 순서를 마지막으로 배치.

제 4 장 타겟 응용에 대한 매핑 규칙 적용

이 장에서는 본 학위 논문에서 러너블-태스크 매핑 작업을 수행할 타겟 응용에 대해 설명한 후 [11]에서 제안한 매핑 규칙과 이를 개선한 매핑 규칙을 타겟 응용에 적용한다.

제 1 절 타겟 응용 설명

본 학위 논문에서 선정한 타겟 응용은 운전자의 개입없이 자동차가 주행할 수 있도록 종방향, 횡방향 제어를 수행하는 ADAS 응용이다. 이는 여러 센서들로부터 주변 상황과 차량 상태를 입력 받아 전방의 차량과의 안전거리 유지하고 차선 내에서 주행할 수 있도록 제어한다. 그림 9와 같이 12개의 소프트웨어 컴포넌트로 구성되며 크게 입력, 추정, 연산, 출력 4단계로 진행된다. 각 소프트웨어 컴포넌트는 하나의 러너블을 가지며 러너블의 이름은 소프트웨어 컴포넌트의 이름과 같다.

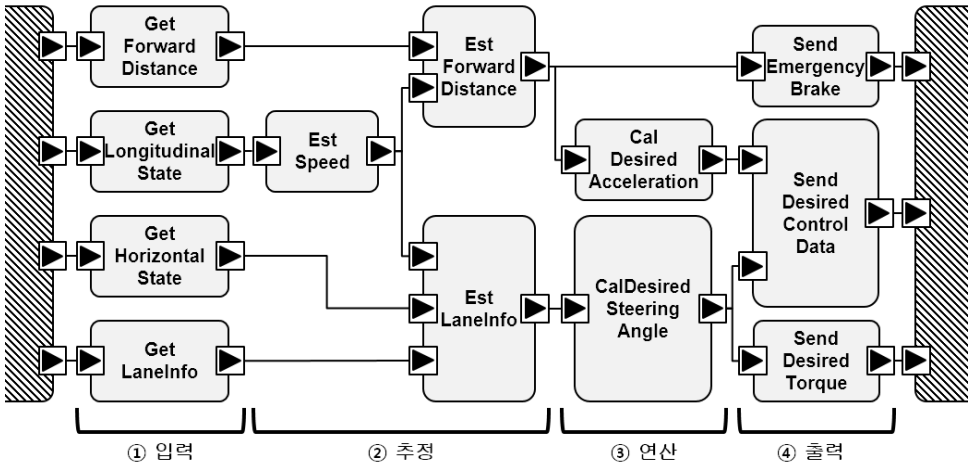

그림 9. 타겟 응용의 소프트웨어 컴포넌트 설계

입력 단계의 4개 러너블은 여러 센서로부터 주변 상황과 차량의 상태를 CAN(Controller Area Network) 통신을 통해 입력 받아 CAN 메시지를 전처리한다. 가장 왼쪽에 있는 빗금으로 표시된 소프트웨어 컴포넌트가 센서로부터 들어온 CAN 메시지를 전달하는 역할을 한다. GetForwardDistance는 전방 차량과의 거리를 레이더 센서로부터 입력 받고 GetLongitudinalState는 차량의 속도와 가속도를 차량 내

센서로부터 입력 받는다. GetHorizontalState는 차량의 조향각과 요레이트(yaw rate)를 차량 내 센서로부터 입력 받고 GetLaneInfo는 차선의 정보를 영상 센서로부터 입력 받는다. 4개의 러너블은 TimingEvent를 통해 주기적으로 실행되며 센서의 주기에 맞춰 각각 50ms, 20ms, 10ms, 100ms의 주기를 갖는다. CAN 메시지를 수신한 뒤 전처리된 데이터를 송신하기 위해 DataReceivedPoint와 DataSendPoint를 가지며, CAN 메시지가 일정 시간 동안 들어오지 않을 경우 오류를 출력하기 위해 DataReceivedEvent를 timeout만큼의 시간 동안 기다리는 WaitPoint를 가진다.

추정 단계의 3개 러너블은 10ms마다 제어 신호를 출력하기 위해 센서로부터 입력된 데이터를 10ms 주기로 추정한다. EstForwardDistance는 전방 차량과의 거리를, EstSpeed는 차량의 속도를, EstLaneInfo는 차선 정보를 추정하며 multi-rate kalman filter를 사용한다. 3개의 러너블 모두 10ms마다 발생하는 TimingEvent에 의해 주기적으로 실행되며 EstSpeed는 DataReceivedPoint와 DataSendPoint를 갖고 EstForwardDistance는 2개의 DataReceivedPoint와 1개의 DataSendPoint를 가지며 EstLaneInfo는 3개의 DataReceivedPoint와 1개의 DataSendPoint를 가진다. EstForwardDistance와 EstLaneInfo는 전방 차량과의 거리와 차선 정보를 추정하기 위해 EstSpeed로부터 입력되는 속도 정보가 반드시 필요하므로 EstSpeed로부터 데이터를 수신했을 때 발생하는 DataReceivedEvent를 기다리도록 WaitPoint를 갖는다.

연산 단계의 2개 러너블은 센서로부터 입력된 데이터를 바탕으로 목표 제어 값을 연산한다. CalDesiredAcceleration은 전방 차량과의 거리를 입력 받아 안전 거리를 유지할 수 있도록 목표 가속도 값을 연산하고 CalDesiredSteeringAngle은 차선 정보를 입력 받아 차선에서 벗어나지 않도록 목표 조향각 값을 연산한다. 이들은 필요한 입력 값이 수신되었을 때 작동할 수 있도록 DataReceivedEvent에 의해 작동하며 DataReceivedPoint와 DataSendPoint를 갖는다.

출력 단계의 3개 러너블은 제어 값을 CAN 통신을 통해 제어기로 전달하기 위해 CAN 메시지로 만들어 출력한다. 가장 오른쪽에 있는 빗금으로 표시된 소프트웨어 컴포넌트가 출력된 CAN 메시지를

제어기로 전달하는 역할을 한다. SendEmergencyBrake는 전방 차량과의 거리가 매우 가까울 때 긴급 제동 신호를 출력하며 SendDesiredControlData는 목표 가속도와 조향각 값을 입력 받아 하나의 CAN 메시지로 합병하여 출력하고 SendDesiredTorque는 목표 조향각 값을 입력 받아 차량 핸들의 목표 토크 값을 계산하여 출력한다. SendEmergencyBrake는 전방 차량과의 거리를 입력 받을 때 작동하기 위해 DataReceivedEvent에 의해 작동되며 SendDesiredControlData와 SendDesiredTorque는 10ms마다 제어 신호를 출력하기 위해 TimingEvent에 의해 작동된다. SendEmergencyBrake와 SendDesiredTorque는 DataReceivedPoint와 DataSendPoint를 가지며 SendDesiredControlData는 2개의 DataReceivedPoint와 1개의 DataSendPoint를 가지며 목표 가속도와 조향각 값이 반드시 필요하므로 두 데이터가 수신될 때 발생하는 DataReceivedEvent를 기다리도록 WaitPoint를 가진다.

표 1은 12개 러너블을 명세한 표이다.

표 1. 타겟 응용의 러너블 명세

단계	이름	속성	이벤트
입력	GetForwardDistance	DRP & WP, DSP	TE 50ms
	GetLongitudinalState	DRP & WP, DSP	TE 20ms
	GetHorizontalState	DRP & WP, DSP	TE 10ms
	GetLaneInfo	DRP & WP, DSP	TE 100ms
추정	EstSpeed	DRP, DSP	TE 10ms
	EstForwardDistance	2 DRP & WP, DSP	TE 10ms
	EstLaneInfo	3 DRP & WP, DSP	TE 10ms
연산	CalDesiredAcceleration	DRP, DSP	DR
	CalDesiredSteeringAngle	DRP, DSP	DR
출력	SendEmergencyBrake	DRP, DSP	DR
	SendDesiredControlData	2 DRP & WP, DSP	TE 10ms
	SendDesiredTorque	DRP, DSP	TE 10ms

DRP: DataReceivePoint, WP: WaitPoint, DSP: DataSendPoint

TE: TimingEvent, DR: DataReceivedEvent

제 2 절 규칙 적용

이 절에서는 1절에서 설명한 타겟 응용의 12개의 러너블에 대해 [11]에서 제안한 6가지 러너블-태스크 매핑 규칙과 본 학위 논문에서 개선한 매핑 규칙을 적용하여 태스크에 매핑한다.

2.1 기존 규칙 적용

12개의 러너블에 6개의 기존 매핑 규칙을 적용하면, 먼저 R.1에 의해 서로 데이터를 송수신하고 하나의 DataSendPoint와 DataReceivePoint를 갖는 러너블인 GetLongitudinalState와 EstSpeed가 같은 태스크에 매핑되고 EstLaneInfo와 CalDesiredSteeringAngle과 SendDesiredTorque가 같은 태스크에 매핑된다. 또한 R.2에 의해 DataReceivedPoint를 가지며 같은 데이터를 수신하는 러너블인 SendEmergencyBrake와 CalDesiredAcceleration이 같은 태스크에 매핑되고 R.1에 의해 이들에게 데이터를 송신하는 EstForwardDistance까지 함께 매핑된다.

R.6에 의해 WaitPoint를 가진 7개의 러너블은 한 러너블의 WaitPoint로 인해 태스크의 수행이 멈추었을 때 다른 러너블의 WaitPoint에서 기다리는 이벤트가 발생하여도 수행을 재개하지 못하는 상황을 막기 위해 각자 다른 태스크에 매핑되어야 한다. 하지만 이들 중 EstForwardDistance와 EstLaneInfo의 WaitPoint는 같은 DataReceivedEvent를 기다리는 WaitPoint이므로 같은 태스크에 매핑할 수 있다.

위 사항들을 반영하면 두 가지 매핑의 경우가 나올 수 있으며 이는 그림 10, 11과 같다. 첫 번째 경우는 같은 태스크에 매핑되어야 하는 러너블들을 각자 하나의 태스크에 매핑한 경우이며 두 번째 경우는 이들 중 합병 가능한 두 태스크를 합병한 경우이다.

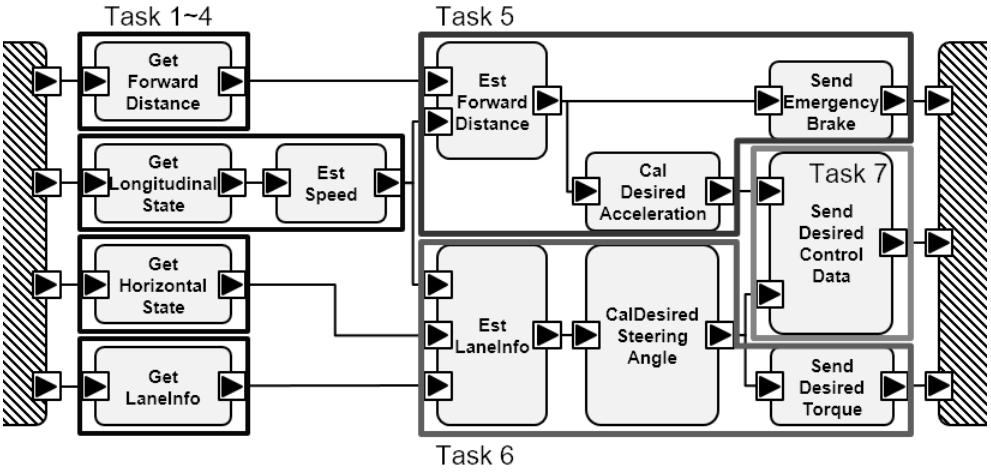

그림 10. 기존 매핑 규칙 적용 경우 1

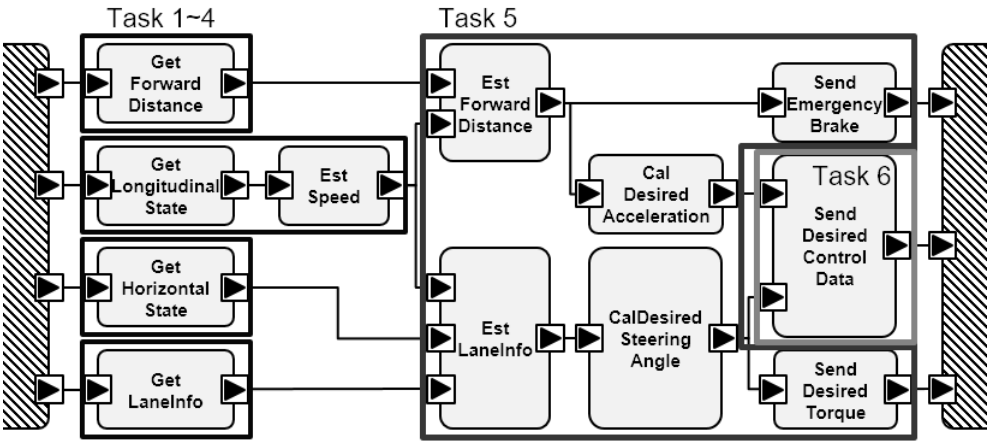

그림 11. 기존 매핑 규칙 적용 경우 2

2.2 개선된 규칙 적용

확장된 R.4′ 를 반영하면 10ms의 주기로 실행되는 5개의 러너블인 EstSpeed, EstForwardDistance, EstLaneInfo, SendDesiredControlData, SendDesiredTorque는 같은 태스크에 매핑되어야 한다. 하지만 EstSpeed는 R.1에 의해 GetLongitudinalState와 같은 태스크에 매핑되었으며 GetLongitudinalState는 WaitPoint를 가지므로 R.6′ 에 의해 다른 러너블들과 함께 매핑될 수 없다. 이외의 4개의 러너블은 다른 규칙과 상충되지 않아 함께 매핑될 수 있다. EstForwardDistance 또는 EstLaneInfo와 SendDesiredControlData는 서로 다른 WaitPoint를

가지기 때문에 기존의 R.6에 의해 같은 태스크에 매핑될 수 없었지만, EstForwardDistance, EstLaneInfo의 WaitPoint에서 기다리는 이벤트가 발생한 후 연산 단계의 CalDesiredAcceleration과 CalDesiredSteeringAngle이 수행되고 이들이 발생시키는 이벤트를 SendDesiredControlData의 WaitPoint가 기다린다. 따라서 SendDesiredControlData의 WaitPoint에서 기다리는 이벤트는 반드시 EstForwardDistance 또는 EstLaneInfo의 WaitPoint 이후에 발생하므로 R.6′ 에 의해 함께 매핑되어도 무관하다.

SendDesiredControlData는 다른 러너블보다 선행되어야 하는 조건이 없으며 WaitPoint를 가지므로 R.7에 의해 가장 마지막 순서로 배치되어야 한다. 그림 12는 본 학위 논문에서 개선한 매핑 규칙을 적용한 경우를 나타낸 그림이다.

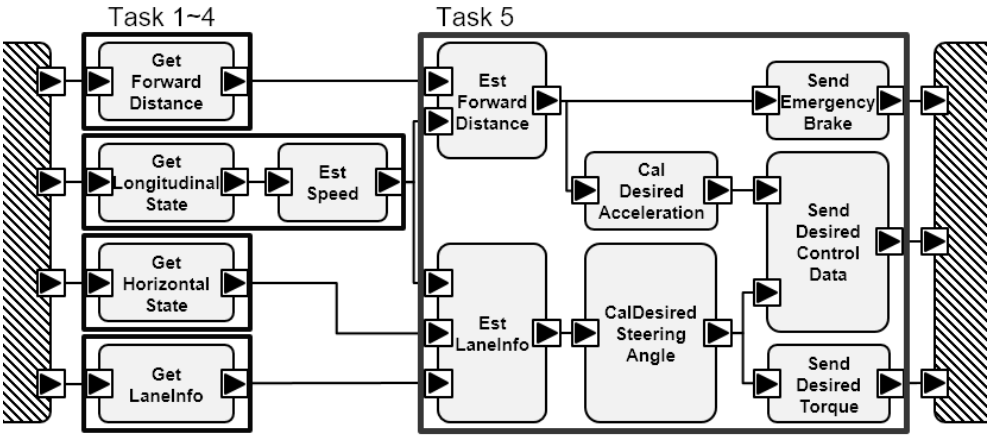

그림 12. 개선된 규칙 적용 경우

제 5 장 실험 및 검증

이 장에서는 개발자가 임의로 러너블-태스크 매핑을 수행한 경우와 기존의 6개 매핑 규칙을 따른 경우, 본 학위 논문에서 개선한 매핑 규칙을 따른 경우에 대해 ADAS 타겟 응용의 종단 간 응답 시간을 실험을 통해 비교한다. 먼저 실험 환경을 소개한 뒤 실험 구성을 설명하고 실험 결과를 평가한다.

제 1 절 실험 환경

본 학위 논문에서는 표 2와 같은 실험 환경 상에 타겟 응용을 구현하였다. 실험에 사용한 보드는 Infineon 사의 AURIX Starter Kit TC297이며 이는 MCU(Microcontroller Unit)로 TC297을 가진다. TC297은 프로세서로 TriCore™ 1.6P를 세 개 보유하지만 본 학위 논문에서는 한 개만 사용하며 주파수는 최저치인 80MHz로 설정하였다. MCU에 올라갈 AUTOSAR 소프트웨어 플랫폼은 ETAS 사의 제품인 RTA-RTE 6.2.1, RTA-OS 5.5.9를 사용하였고 AUTOSAR 버전 4.2.2에 따라 타겟 응용을 구현하였다. 개발된 소프트웨어는 HighTec 컴파일러 4.6.5.0을 통해 컴파일하여 MCU에서 구동하였다.

표 2. 실험 환경

Hardware	Test board		AURIX Starter Kit TC297
	MCU		TC297
	Processor		TriCore™ 1.6P @ 80MHz
Software	AUTOSAR SW platform	Version	Release 4.2.2
		RTE	RTA-RTE 6.2.1
		OS	RTA-OS 5.5.9
	Compiler		HighTec 4.6.5.0

제 2 절 실험 구성

실험은 타겟 응용의 12개의 러너블을 개발자가 임의로 태스크에

매핑한 경우, [11]에서 제시한 6개의 규칙에 따라 매핑한 경우, 본 학위 논문에서 개선한 규칙에 따라 매핑한 경우에 대해 각각 구현한 후 종단 간 응답 시간을 측정하여 비교한다. 타겟 응용의 종단 간 응답 시간은 12개의 러너블이 모두 작동될 때 첫 러너블이 시작되는 시간에서 모든 러너블이 수행을 마쳤을 때 시간의 차이다.

개발자가 임의로 매핑한 경우는 무수히 많은 경우의 수가 존재하므로 이를 모두 구현하여 종단 간 응답 시간을 측정하기란 불가능하다. 따라서 본 학위 논문에서는 매핑 규칙을 적용했을 때의 종단 간 응답 시간과 비교하기 위해 세 개의 값을 구하고자 한다. 첫 번째 값은 모든 러너블을 모두 한 태스크에 매핑했을 때의 종단 간 응답 시간이고 두 번째 값은 모든 러너블을 독자적인 태스크에 따로 매핑했을 때의 종단 간 응답 시간이다. 세 번째 값은 개발자가 임의로 매핑한 경우의 기댓값이며 이는 10개의 러너블-태스크 매핑을 표본으로 뽑아 이들의 종단 간 응답 시간 평균을 계산하여 구한다. 모든 러너블을 한 태스크에 매핑할 경우 문맥 교환 오버헤드와 동기화 오버헤드가 최소가 되지만 WaitPoint로 인해 해당 태스크가 대기 상태가 될 경우 모든 러너블의 수행이 불가능하다. 따라서 WaitPoint에서 기다리는 이벤트가 늘 WaitPoint 이전에 발생한다는 가정 하에서는 가장 종단 간 응답 시간이 짧겠지만, 만일 하나의 WaitPoint라도 태스크를 대기 상태로 만들면 모든 러너블의 수행이 불가능하므로 치명적인 수행 지연을 초래한다. 모든 러너블을 독자적인 태스크에 따로 매핑한다면 반대로 WaitPoint 때문에 다른 러너블들의 수행이 지연되는 일은 발생하지 않지만 문맥 교환 오버헤드, 동기화 오버헤드가 최대가 되어 가장 종단 간 응답 시간이 길게 된다. 이 두 극단의 케이스는 종단 간 응답 시간의 최소, 최대를 보여주기 위해 측정한다. 개발자가 임의로 매핑한 경우 종단 간 응답 시간의 기댓값을 구하기 위해 현실적으로 개발자가 선택할 만한 매핑 중에서 10개의 표본을 뽑아야 하는데, R.6′과 R.7을 만족하지 않는 매핑은 WaitPoint에 의해 치명적인 수행 지연이 발생할 수 있으므로[3] 10개의 표본은 이를 만족시키는 매핑에 한해서 선택한다.

[11]에서 제시한 6개의 규칙에 따라 매핑한 경우는 두 가지로써, 4장의 2.1절에서 설명하였고 이를 개선한 규칙에 따라 매핑한 경우는 2.2절에서 설명하였다. 이와 같이 실험에서 비교하고자 하는 종단 간

응답 시간의 값은 6가지이다.

제 3 절 실험 결과 및 평가

타겟 응용의 종단 간 응답 시간은 AUTOSAR OS에서 제공하는 시간 측정 함수를 이용하여 측정하였으며 실험 결과는 아래와 같다. 1번은 모든 러너블을 한 태스크에 매핑한 경우, 2번은 모든 러너블이 각자 따로 매핑된 경우, 3번은 개발자가 임의로 매핑한 경우의 기댓값이며 4, 5번이 [11]의 규칙을 적용한 두 가지 경우이고 6번이 개선된 규칙에 따라 매핑된 경우이다.

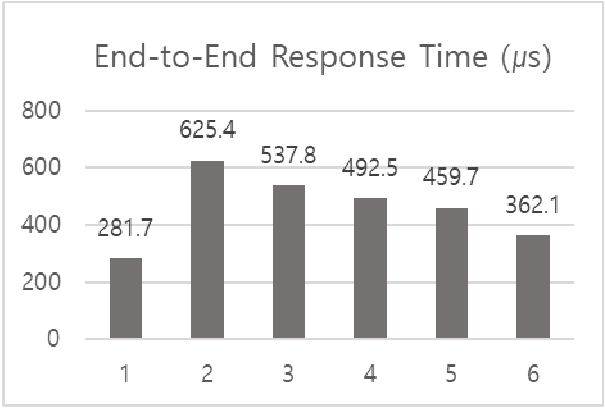

그림 13. 실험 결과

실험 결과를 보면 개발자가 임의로 매핑한 경우 종단 간 응답 시간의 최댓값은 625.4μs이고 기댓값은 537.8μs이다. 이에 반해 [11]의 6가지 규칙만을 적용하여 매핑한 경우 459.7μs이고 제안한 매핑 규칙을 적용하여 매핑한 경우 362.1μs이다. 따라서 제안한 매핑 규칙을 적용하여 매핑한 경우 타겟 응용의 종단 간 응답 시간이 개발자가 임의로 매핑한 경우에 비해 최대 1.73배, 평균 1.49배 단축되며 기존 6개의 규칙만을 사용한 경우에 비해 1.27배 단축됨을 확인하였다.

제 6 장 결 론

본 학위 논문에서는 AUTOSAR 기반으로 운전자의 개입 없이 자동차를 주행하도록 제어하는 ADAS 응용에 대하여 적은 시스템 오버헤드를 가짐으로써 최소의 종단 간 응답 시간을 갖는 러너블-태스크 매핑을 찾고자 한다. 이를 위해 [11]에서 제안한 6개의 매핑 규칙을 적용하였으며 추가적인 성능 향상을 위해 규칙 중 일부를 확장하고 새로운 규칙을 추가하여 개선된 매핑 규칙을 제안하였다.

이후 실험을 통해 제안한 매핑 규칙을 타겟 응용에 적용하여 러너블-태스크 매핑을 하였을 때 기존의 6가지 매핑 규칙만을 적용했을 때와 개발자가 임의로 매핑한 경우보다 얼마나 종단 간 응답 시간이 단축되는지 비교한다. 실험 결과, 타겟 응용의 종단 간 응답 시간이 기존 매핑 규칙만을 사용했을 때보다 1.27배 단축되었고 개발자가 임의로 매핑한 경우의 기댓값보다 1.49배 단축됨을 확인하였다.

본 학위 논문은 후속 연구로써, AUTOSAR 버전 4.0부터 지원하는 멀티코어 CPU를 활용하여 러너블-태스크 매핑 뿐만 아니라 태스크-코어 매핑까지 고려한 시스템 성능 최적화 연구를 통해 더욱 발전시킬 수 있다.

참고 문헌

[1] Abelein, Ulrich, et al. "Complexity, quality and robustness-the challenges of tomorrow's automotive electronics." Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012. IEEE, 2012.

[2] AUTOSAR Consortium, AUTOSAR Release 4.3, Specification of Layered Software Architecture, 2017.

[3] AUTOSAR Consortium, AUTOSAR Release 4.3, Specification of RTE Software, 2017.

[4] Wozniak, Ernest, et al. "An optimization approach for the synthesis of AUTOSAR architectures." Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on. IEEE, 2013.

[5] Zeng, H., and Di Natale, M. "Efficient implementa- tion of AUTOSAR components with minimal memory usage." Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on. IEEE, 2012.

[6] Monot, Aurélien, et al. "Multisource software on multicore automotive ECUs—Combining runnable sequencing with task scheduling." IEEE Transactions on Industrial Electronics 59.10 (2012): 3934-3942.

[7] Faragardi, Hamid Reza, Björn Lisper, and Thomas Nolte. "Towards a communication-efficient mapping of AUTOSAR runnables on multi-cores." Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on. IEEE, 2013.

[8] Zhang, Ming, and Zonghua Gu. "Optimization issues in mapping AUTOSAR components to distributed multithreaded implementations." Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on. IEEE, 2011.

[9] Sailer, Andreas, et al. "Optimizing the task allocation step for multi-core

processors within AUTOSAR." Applied Electronics (AE), 2013 International Conference on. IEEE, 2013.

[10] Faragardi, Hamid Reza, et al. "A communication- aware solution framework for mapping autosar runnables on multi-core systems." Emerging Tech- nology and Factory Automation (ETFA), 2014 IEEE.

[11] Long, Rongshen, et al. "An approach to optimize intra-ecu communication based on mapping of autosar runnable entities." Embedded Software and Systems, 2009. ICESS'09.

Abstract

Optimizing the Performance of AUTOSAR-based Automotive System via Runnable-to-Task Mapping Rules

Wooyoung Min

Department of Electrical and Computer Engineering

The Graduate School

Seoul National University

As automobiles become increasingly electric, the size and complexity of automotive software is greatly increasing. As a result, the time and cost of developing automotive software has also increased, leading European automotive companies have established the AUTOSAR (Automotive Open System Architecture) standard to improve development efficiency. The AUTOSAR standard is the standard for the architecture and development process of automotive software.

Application software according to the AUTOSAR standard is modularized in software components, and each software component has one or more runnables that implement its functions. The developer maps the runnables to the tasks, which is the scheduling unit of the operating system, in order to execute the runnable. Runnable-to-task mapping is very important process in terms of system performance, since the system overhead incurred greatly

depends on the runnable-to-task mapping.

In this thesis, I apply six runnable-to-task mapping rules proposed in the previous research to optimize the performance of the target application that performs autonomous driving, and propose mapping rules that improves the existing rules for further performance enhancement. I compare the performance of the target application when the runnables are mapped to the tasks according to the proposed mapping rules and when the developer arbitrarily mapped it. The target application is implemented on the Infineon AURIX board and ETAS AUTOSAR platform. Experimental results show that the end-to-end response time of the target application when mapped by applying the proposed rules is about 1.49 times shorter than the expected value when the developer arbitrarily mapped.

Keywords : AUTOSAR, Automotive software, Runnable-to-task mapping, Mapping rules, End-to-end response time
Student Number : 2017-26663