논문 2018-13-32

AUTOSAR 기반 공유자원이용 스케줄링 구조

(Design of Scheduling on AUTOSAR OS With Shared Resource)

최 준 열*, 조 준 형, 최 윤 자
(Junyeol Choi, Joonhyung Cho, Yunja Choi)

Abstract : As a result of the technological advances in the E / E system, automotive system can provide advanced functions for safety and comfort. In addition, mechanical systems is changed to the electronic system. And the systems perform cooperative functions through communication. So the E / E system becomes more complicated as the size of the system increases. In order to secure the safety of complicated E / E system, ISO26262 standard require that Freedom from Interference and Sufficient Independence be met. In this paper, we propose a software scheduling method that can guarantee the independence between decomposed components after software decomposition and software development of ASIL D level EPB (Electronic Parking Brake) system

Keywords : AUTOSAR, Embedded, Scheduling, Execution time, Vehicle system, ISO 26262

Ⅰ. 서 론

차량을 구성하는 시스템의 전자화가 빠르게 증가되면서 소프트웨어가 차지하는 비중은 커지고 있다. 차량용 소프트웨어의 대부분은 임베디드 제어 소프트웨어로서 해당 소프트웨어의 장애는 시스템의 안전 목표의 달성을 방해하는 원인이 되어 심각한 손실 또는 재난을 일으키게 된다. 시스템의 안전 목표를 충족하기 위해 소프트웨어의 규모는 점차 증대 되었다. 앞으로도 안전 및 편의 제공을 위해 차량 시스템 간의 연계성은 매우 중요하게 될 것이며 이로 인해 소프트웨어 규모의 증대는 가속될 것으로 예상된다.

소프트웨어 구조 설계의 연구와 함께 복잡한 소프트웨어를 안전하고 신뢰성 있게 운용하기 위해 AUTOSAR (AUTomotive Open System Architecture)는 통합된 소프트웨어 구조 명세서를 만들었다.

AUTOSAR는 RTE (RunTime-Environment)를 사용하여 하드웨어와 소프트웨어를 연계시키고, 소프트웨어 간 독립성도 보장하여 소프트웨어의 재사용과 신뢰성을 모두 만족할 수 있도록 하였다 [1].

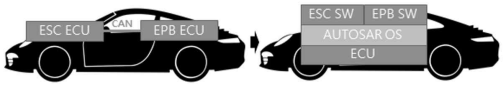

그림 1. 자동차 시스템 구성 경향
Fig. 1 Trend of vehicle systems

차량 시스템의 구조는 그림 1과 같이 기존의 독립형 시스템 구조에서 통합형 시스템 구조로 점차 변경되고 있으며, 이에 따라 시스템을 구성하는 구성요소의 크기, 무게, 전력소모에서 이득을 가지게 되었다.

하지만 그에 따라 동일한 하드웨어를 다수의 소프트웨어가 공유하게 되었으며 소프트웨어 간 스케줄링의 고려가 불가피하게 되었다. 스케줄링의 고려와 함께 목표실행시간 초과로 인한 연계 고장 (Cascading failure) 역시 고려하게 되었다. 지금까지는 독립형 시스템 구조였기 때문에 목표실행시간 초과로 발생하는 연계 고장은 고려하지 않아도 되었다.

제동 아이템은 EPB와 ESC 시스템으로 구성된다. 본 논문에서는 통합형 EPB/ESC시스템의 소프트웨어를 구성하는 태스크가 더 많은 자원을 사용하면서 목표실행시간 초과 시 의도하지 않은 시스템 동작을 막는 기법을 제안한다. 그리고 실제 개발

*Corresponding Author (trustcjy@gmail.com)
Received: Apr. 15 2018, Revised: July 9 2018, Accepted: Sep. 11 2018.
J. Choi, J. Cho: MANDO Brake Center, Y. Choi: Kyungpook National University

 2018 Dec. 13(6) 279-288
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2018.13.6.279

되는 ECU에서의 시뮬레이션을 통해 제안한 기법의 효율성과 타당성을 확인한다.

Ⅱ. 연구배경

1. AUTOSAR 플랫폼 운영체제

소프트웨어 구조와 안전성 및 신뢰성 관계에서 보면 AUTOSAR 플랫폼은 단순히 소프트웨어 플랫폼 표준화 이상의 의미를 가진다. 하드웨어와 소프트웨어의 분리를 통해 소프트웨어 재사용성 및 유지보수성을 높이고 소프트웨어 간 독립성을 보장해 하드웨어의 다중화 없이 안전을 보장하기 위한 시스템을 구성할 수 있다 [2]. AUTOSAR는 규격 시험 표준화를 통해 AUTOSAR 소프트웨어의 규격을 평가할 수 있는 시험 명세와 시험 프로세스를 명시하여 전장 소프트웨어의 신뢰성과 안전성 확보를 위한 가이드라인을 제공하고 있다.

AUTOSAR OS는 Partitioning OS종류의 하나로 일반적으로 사용되던 스케줄링 방식과는 다르다 [3]. 그림 2와 같이 Application 영역의 SWC (SoftWare Component)를 구성하는 Runnable과 BSW (Basic SoftWare) 영역의 BSW scheduled function을 태스크 단위로 선점형 스케줄링하여 구동한다.

SWC의 Runnable은 OS 스케줄러에 의해 관리되는 RTE (Runtime Environment)의 태스크와 맵핑되어 호출되며 BSW scheduled function은 BSW 스케줄러에 의해 태스크 단위로 스케줄링 된다. ASW 영역과 BSW 영역 간에는 RTE에 의해 분리되어 통신한다.

각 응용프로그램은 SWC로 구성되며 SWC를 구성하는 Runnable이 맵핑된 태스크의 스케줄링을 통해 소프트웨어가 구동된다. Runnable의 실행순서, 이벤트의 호출 주기, 호출 조건 등이 정의되면 각 Runnable별로 RTE의 태스크와 맵핑된다. OS 스케줄러는 태스크의 속성 및 스케줄 가이드에 따라 태스크를 운용하고, 태스크가 동작되면 태스크에 맵핑된 Runnable들이 실행된다. RTE는 태스크 운용을 위한 다양한 이벤트를 제공하며 일반적으로는 OS system clock에 따라 동작하는 Rte_Time_Task로 구현된다. RTE는 OS 스케줄러가 기능 단위의 Runnable을 구동할 수 있도록 태스크를 생성하여 Runnable을 호출하는 중재자 역할을 한다. 태스크는 OS 서비스를 이용하여 활성화, 재개, 종료 등의 제어를 하여 이를 통해 Runnable이 가지고 있는 고유 기능들이 서비스 된다.

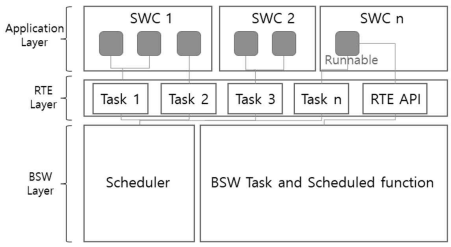

그림 2. AUTOSAR 스케줄링 구조
Fig. 2 AUTOSAR scheduling architecture

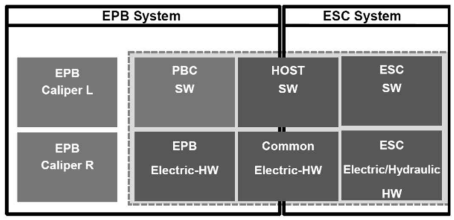

그림 3. 제동 아이템 구조
Fig. 3 Brake item architecture

2. 제동 아이템

제동 아이템은 차량의 속력, 가속도, 브레이크의 상태, 운전자의 의지 등의 정보를 종합하여 차량의 주차, 주행, 감속을 보조해 주는 장치이다. 제동 아이템은 그림 3과 같이 EPB와 ESC 시스템이 통합되어 기능을 수행한다 [4]. 각 시스템은 EPB Caliper 하드웨어를 공유하여 제동 아이템의 기능을 충족시킨다. ESC 시스템은 유압을 이용하며 EPB는 모터제어를 이용하여 제동력을 조절한다.

제동 아이템의 소프트웨어는 ESC 시스템을 담당하는 ESC 소프트웨어, EPB 시스템을 담당하는 PBC 소프트웨어 그리고 ESC와 EPB 2가지 시스템에 관련된 HOST 소프트웨어로 구성된다. 즉 ESC 시스템이 정상동작하기 위해서는 ESC 소프트웨어와 HOST 소프트웨어가 정상동작 되어야 하며, EPB 시스템이 정상동작하기 위해서는 PBC 소프트웨어와 HOST 소프트웨어가 정상동작 되어야 한다.

각 소프트웨어는 ISO26262 표준에 따라 개발되며, ESC와 HOST 소프트웨어는 최고위험에 해당하는 ASIL D등급의 요구사항을 가지며, PBC 소프트웨어는 그보다 낮은 ASIL B등급의 요구사항을 가진다. ISO26262 표준 측면에서 ASIL B에 해당하는 PBC 소프트웨어의 고장이 ASIL D에 해당하는 HOST 또는 ESC로 연계되지 않도록 독립성을 가지도록 설계되어야 한다 [5].

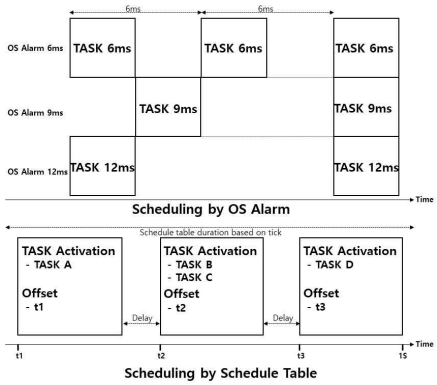

그림 4. AUTOSAR 운영체제 스케줄링
Fig. 4 Scheduling by AUTORSAR Operating System

3. 차량용 임베디드 소프트웨어 스케줄링

소프트웨어의 스케줄링 시에도 낮은 위험도를 가지는 태스크가 높은 위험도를 가지는 태스크에 영향을 주지 않도록 해야 한다. 즉, SWC를 구성하는 Runnable이 맵핑된 태스크를 관리하여 각각의 태스크가 가지고 있는 시간 제약성을 위반하지 않도록 설계한다. 주기적으로 정해진 시간마다 반복적으로 태스크가 동작하는데 이때 각각의 태스크들은 시간 제약성을 가진다. 차량용 임베디드 시스템과 일반 시스템의 비교 시, 가장 큰 차이점은 소프트웨어의 동작주기별로 기능의 종료 시점을 정확하게 보장해야 하는 것이다. 태스크의 종료 시점이 다음 태스크가 동작 될 시점보다 늦어진다면 시간 제약성을 보장하지 못하고 안전 목표를 충족하지 못하는 상황이 발생할 수 있다 [6].

기존에는 시간 제약성을 만족하기 위해 일반적으로 오프라인 스케줄링을 통해 스케줄링을 설계한다. 오프라인 스케줄링은 소프트웨어 구조 설계 단계에서 실행 순서 및 실행 시간에 대한 제약사항을 정한다. 이를 바탕으로 구현 후 운영체제 통합자가 태스크들의 스케줄링을 진행한다.

AUTOSAR 표준을 따르는 OS를 사용하는 경우 일반적으로 그림 4와 같이 2가지의 스케줄링이 가능하다.

AUTOSAR OS Alarm은 정해진 시간 주기로 할당된 태스크를 수행한다. 다수의 태스크가 동시에 실행되어야 하는 경우 태스크의 지연이 발생하므로 태스크가 중복되지 않도록 적절한 스케줄링이 필요하다.

AUTOSAR Schedule Table방법은 다수의 태스크가 중복되지 않도록 Expiry point별로 태스크를 분배하여 tick단위로 각 태스크를 실행하는 방법이다. Schedule table에 등록된 태스크가 목표실행시간을 초과하면 다음에 수행되어야 할 태스크의 지연이 발생할 수 있다.

일반적으로 제동 아이템은 구성되는 소프트웨어의 동작 순서가 중요하며, 각 소프트웨어 별 목표실행시간 초과에 대한 책임이 명확해야하므로 AUTOSAR Schedule Table 방법을 사용한다.

4. 연구의 필요성 및 범위

자동차 산업 양산프로젝트에서는 스케줄링 최적화를 위해 실행시간 (Execution Time) 기반의 동적 최적화 방법을 많이 사용하고 있었다. 하지만 AUTOSAR 플랫폼이 적용되고 통합형 시스템 구조가 생기면서 일반적인 동적 기반의 방법은 사용하기 어려워졌다 [7]. 태스크의 실행시간은 ECU가 동작하는 환경에 따라 달라지며, 태스크에 맵핑된 Runnable의 제어 흐름 경로에 따라서도 달라지는데 분리 개발되는 각 Runnable의 실행시간을 운영체제 통합자의 입장에서 측정하기가 어렵기 때문이다.

제동 아이템의 경우에도 서로 연관된 다수의 태스크가 독립적으로 동작하며, 각 태스크에 맵핑되는 소프트웨어는 분리개발 된다. 결국 운영체제 통합자의 입장에서 스케줄링의 최적화가 쉽지 않게 되었다.

다수의 시스템이 통합되는 구조로 인해 각 업체에서는 태스크의 실행시간 초과 시 다음 태스크의 실행이 지연되면서 발생하는 연계 고장을 고려하게 되었다. 결과적으로 시스템 개발 업체들은 시스템의 WCET (Worst Case Execution Time)를 매우 보수적으로 설계하여 태스크의 동작에 많은 자원을 요구하게 되었다.

운영체제 통합자는 요구하는 자원에 대한 적절성을 파악하기 힘들다. 각 업체가 만드는 소프트웨어의 최악실행경로를 알기 어렵기 때문이다. 각 소프트웨어가 상호작용을 하는 제동 아이템의 경우 소프트웨어 간 상호작용으로 만들어지는 최악 경로는 담당하는 업체에서도 파악하기가 어렵다. 즉 제한된 하드웨어 자원에서 동작하는 다수의 태스크를 최적화된 실행시간으로 스케줄링 하는 것은 실제 개발환경에서 거의 불가능하다.

또한 그림 5와 같이 소프트웨어의 유지보수가 진행되면서 기능의 변경 및 추가로 특정 태스크의 최대 실행시간이 증가되어야 하는 경우가 종종 생긴다. 이 경우 전체 주기의 실시간성을 충족하기 위해서는 다른 태스크의 최대 실행시간이 줄어들어야 한다. 하지만 기 양산된 프로젝트 또는 여러 업체가 분리하여 소프트웨어를 개발하는 경우에는 소프트웨어의 설계변경이 쉽지 않다.

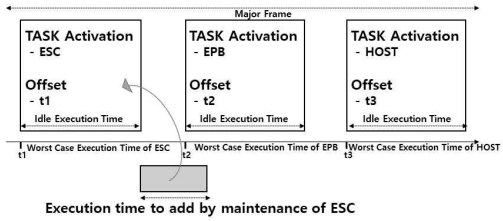

그림 5. 시스템 유지보수에 의한 스케줄링 장애
Fig. 5 Scheduling trouble by system maintenance

결국 유한한 하드웨어 자원은 Runnable 업체들의 자원요구량을 충분하게 충족시키지 못하고 태스크의 목표실행시간 초과로 이어진다. 목표실행시간을 초과하는 경우 시스템의 예상치 못한 동작이 야기될 수 있다. 선점형 스케줄링인 경우 마지막 태스크는 기능이 정상적으로 동작하지 못한 상태에서 문맥교환 (Context Switch)이 일어나며, 비선점형인 경우 시스템 동작주기의 목표실행시간을 넘어가면서 실시간성을 상실하게 된다.

따라서 제한된 프로젝트 기간과 고객의 요구사항 변경요구를 고려해야 되는 자동차산업에서는 스케줄링을 적절하게 설계하기 위해 간단하고 효율적인 방법이 필요하다.

5. Related Works for Scheduling

5.1 Scheduling Policy

전통적인 OS 스케줄링 알고리즘은 크게 선점형, 비선점형으로 구분된다 [8]. 최근 두 가지 기법을 실용적으로 사용하기 위해 제한적 선점형 기법이 연구되고 있다. PTS (선점형 임계치 스케줄링) 기법은 지명우선순위 (nominal priority)와 선점 임계치 (Preemptive Threshold), 두 가지 우선순위 수준을 가지며 태스크는 우선순위가 실행 중 태스크의 선점 임계치보다 높은 경우에 태스크를 선점할 수 있다 [9].

또한 스케줄링을 위해 사용되는 스택 공간을 더욱 줄이고 스케줄링 가능성을 높이기 위해 고정 우선순위 스케줄링 [10]과 선점 임계치 스케줄링을 통합한 PT-AMC (적응형 선점 혼합 임계치)도 연구되었다. PT-AMC는 우선순위 및 선점 임계 값 지정 기법과 WCRT (Worst Case Response Time)를 계산하여 적응형 혼합 임계치를 계산한다 [11].

OS 스케줄링 알고리즘과 스케줄링 가능성 분석의 전통적인 연구 범위를 벗어나 많은 연구자들은 소프트웨어 규모의 증가를 책임질 수 있는 정확한 알고리즘과 플랫폼에 관심을 가지게 되었다. 목표실행시간을 위반하지 않고 태스크의 실행 시간을 얼마나 늘릴 수 있는지 측정하는 확장성 설계와 소프트웨어 구조 최적화 연구가 진행되었다 [12]. 자동차분야에서는 AUTOSAR 모델의 설계에 관심을 가지게 되며 시스템 스케줄링 가능성을 최소화하고 스택 사용을 최소화하기 위해 작업 모델 정의, 맵핑, 임계 값 할당 등의 연구를 진행하고 있다 [13].

5.2 WCET Analysis

WCET 분석은 동적 기반의 분석 방법과 정적 분석 방법으로 진행된다. 동적 기반은 오프라인 스케줄링 시 또는 소프트웨어 검증 시, 소프트웨어를 실제 타겟에서 실행시켜 WCET를 찾는 방법이다.

반면 정적 분석은 다양한 기법으로 소프트웨어의 행위 또는 코드를 기반으로 소프트웨어 실행 없이 WCET를 찾는다. 정적 분석에서 사용하는 분석 기법들은 제어 흐름 그래프, 값 분석, 반복 횟수 분석, 경로 분석, 파이프라인 분석, 캐시 분석 등이 있다 [14-16].

각 분석 기법의 방법의 단점으로는 동적 기반에서는 최악의 경로를 확인할 수 있는 조합을 찾기가 어려우며 정적 분석 기반은 동적 기반에 비해 정확도가 떨어진다 [17].

최근에는 정적 분석기법보다는 동적 기반의 분석기법을 변형하여 사용하는 기법이 주로 연구된다 [18, 19].

관련연구들은 현재 산업에서 발생되는 스케줄링 문제를 해결하기에 충분한 내용을 담고 있다. 하지만 양산 프로젝트에 사용하기에는 현실적으로 어려움이 많다. 먼저 Runnable을 개발하는 집단에서 스케줄링 알고리즘에 대한 이해가 필요하다. 추가로 각 Runnable이 맵핑된 태스크의 속성을 운영체제 통합자가 명확히 이해해야 한다. 실제 자동차 산업에서는 다수의 기관 및 업체가 시스템을 나누어서 개발하기 때문에 운영체제 통합자가 최적화된 스케줄링을 하는 것이 쉽지 않다.

Ⅲ. 제안 기법

본 연구에서는 단일 프로세스 시스템을 다루며, 선점형 스케줄링과 고정 우선순위를 사용하여 목표실행시간 초과 문제로 발생하는 문제를 최소화 하고자 한다.

스케줄링의 효율적인 동작과 함께 결함허용 시스템을 위해 자원이 부족한 경우 시스템이 예측가능 한 동작을 하는 것을 목표로 한다.

1. Shared Spare Time

각 태스크별 실행시간은 $\tau.ID.ET_{idle}$, $\tau.ID.ET_{worst}$ 2가지로 구분된다. $\tau.ID.ET_{idle}$은 실제 타겟에서 동적 기반으로 측정된 태스크 중 기본 상태에서의 실행 시간이며, $\tau.ID.ET_{worst}$는 해당 태스크에 맵핑되는 Runnable 설계업체가 요구하는 최대 실행시간이다.

일반적으로 태스크는 보통의 실행시간 $\tau.ID.ET_{idle}$와 $\tau.ID.ET_{worst}$의 시간차이가 있으며 차량용 임베디드 소프트웨어의 스케줄링은 일반적으로 모든 $\tau.ID.ET_{worst}$기준으로 최대목표시간을 설계한다.

하지만 시스템 동작 시 항상 모든 태스크가 WCET가 되는 것은 아니다. 실제 양산되는 시스템에서도 각각의 태스크가 WCET에 도달하는 것은 흔치 않다. 소프트웨어 이러한 특성을 활용하여 식 (1)과 같이 $\tau.ID.ET_{idle}$과 $\tau.ID.ET_{worst}$의 시간차이를 자원여유시간 (Resource Spare Time)으로 정의한다.

$$\text{Resource Spare Time}: \tau.ID.ET_{worst} - \tau.ID.ET_{idle} \quad (1)$$

그리고 이를 이용하여 보다 더 효율적이고 안전한 스케줄링이 설계되도록 하고자한다. 스케줄링에서 사용할 공유자원 T_{spare}는 식 (2)와 같다. 즉 T_{spare}는 마지막 태스크를 제외한 모든 태스크의 자원여유시간의 합이다.

$$T_{spare} = \sum_{ID=1}^{n-1} (\tau.ID.ET_{worst} - \tau.ID.ET_{idle}) \quad (2)$$

2. Task Execution Blocking

차량용 소프트웨어에 사용되는 OSEK 기반 실시간 시스템은 제한된 선점형 스케줄로 Cooperative 스케줄링으로 구현되어 있다 [20]. Cooperative 스케줄링은 각각의 태스크는 동작중인 태스크를 선점할 수 없지만 스케줄링이 재결정되는 시점에 우선순위에 따라 자원을 선점할 수 있도록 한다. 본 연구에서는 태스크의 실행시간이 시스템 동작주기 *Frame.ET*을 넘어가는 경우 실시간성을 보장하기 위해 마지막 태스크의 실행은 Blocking이 되어야 한다.

일반적으로 정해진 *Frame.ET*을 초과하게 되면 목표실행시간을 초과하는 태스크 또는 인터럽트 루틴을 중단시켜 실행이 되지 않도록 할 수 있지만, 이런 경우 태스크가 어디까지 진행되었는지 알기 쉽지 않고 그림 6과 같이 연관되는 타 태스크와의 상호작용으로 시스템은 예기치 않은 동작을 할 수 있다. 따라서 안전 필수 시스템의 고장허용 설계는 효율성을 높이는 것보다 성능을 낮추고 시스템을 의도된 대로 동작시키는 것에 초점이 맞추어진다. 예를 들어 제동 아이템의 안전 상태는 “Actuating stop“이며 예기치 않은 동작을 하는 것보다 동작을 하지 않는 것이 더욱 안전한 상태가 된다.

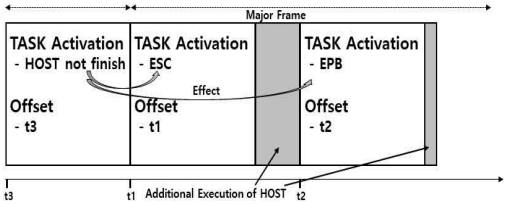

그림 6. 태스크 실행시간 초과에 따른 영향
Fig. 6 Affects related systems by execution timeout

따라서 제안되는 방법은 연관된 시스템의 안전상태 [21]를 위해 남아있는 자원 T_{remain}이 태스크의 동작에 부족하다고 판단되면 식 (3)과 같이 해당 태스크가 수행이 되지 않도록 한다.

$$\text{Task Block: } T_{remain} < \tau.ID.ET_{worst} \quad (3)$$

즉 고장 허용 시스템을 구현하기 위해 목표 *Frame.ET*을 초과하는 경우 태스크의 동작을 금지하는 것이 안전 상태라고 정의했다. 태스크의 경우 소프트웨어가 어디까지 실행되었는지에 따라 상호작용을 하는 연관된 후순위 태스크의 동작이 달라지기 때문이다. 예상하지 못하는 태스크의 동작은 전체 시스템 측면에서 의도하지 않는 동작의 원인이 된다.

3. Scheduling with Shared Resource

공유자원을 이용한 스케줄링은 각 태스크의 자원여유시간을 고려하여 설계한다. 제안기법은 그림 7과 같이 동작한다.

연관된 태스크들 순차적으로 동작시키며, 마지막 태스크가 동작해야 되는 시점에 남아있는 자원이 부족하다면 마지막 태스크의 동작이 시작되지 않도록 한다.

그림 8과 같이 태스크 간 간격을 두지 않으며 차례로 수행시키되, 자원이 부족하다고 판단되는 경

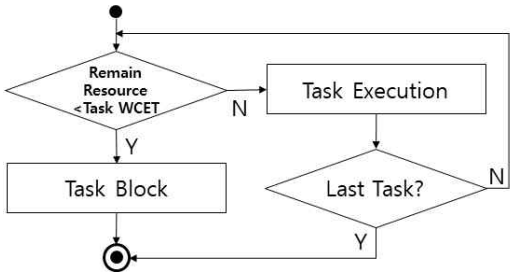

그림 7. 공유자원 기반 스케줄링 알고리즘
Fig. 7 Scheduling with shared resource

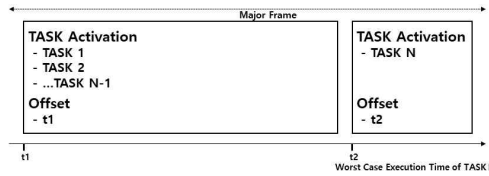

그림 8. 공유자원 기반 스케줄링 구조
Fig. 8 Architecture of shared resource scheduling

우 마지막 태스크의 실행을 금지하여 예상하지 못하는 시스템의 동작을 예방한다. 시스템은 다음동작 주기에서 재시작 되며 운영체제 통합의 측면에서 안전한 설계 개념을 구현할 수 있게 된다.

4. Design of the Scheduling with AUTOSAR

AUTOSAR 플랫폼의 운영체제에서 본 기법을 구현하기 위해서는 SC (Scalability Class) 2가 필요하다. SC2는 SC1의 OSEK OS를 포함하는 기본기능과 Timing protection등의 시간적인 기능을 지원한다. 본 기법의 적용을 위해서는 SC1의 Counter, Schedule table, SC2의 Protection Hook, Timing Protection의 기능들이 조합되어야 한다.

제동 아이템을 예로 들면 Schedule table을 이용하여 그림 9와 같이 설계한다. 먼저 수행되는 ESC 태스크와 EPB 태스크를 하나의 Expiry point 1에 할당하고 Expiry point 2에는 마지막에 수행되는 HOST 태스크를 배치한다.

다음으로 각 태스크를 설정할 때 Execution budget을 할당하여 $\tau.ID.ET_{worst}$를 초과하는 경우 ProtectionHook이 호출되도록 한다. ProtectionHook이 호출되면 PRO_TERMINATETASKISR 정책을 사용하여 목표실행시간을 초과하는 태스크가 종료되도록 한다.

또한 시스템 동작주기에서 $Frame.ET$ 초과가 예상되는 경우 마지막 태스크인 HOST 태스크의 동작

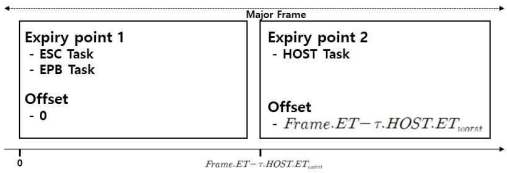

그림 9. 제동 아이템의 스케줄링 구조
Fig. 9 Scheduling architecture of brake item

을 막기 위해 OsSecondsPerTick을 이용한다. 그리고 Expiry point 1에 배치되는 Task들의 실행시간이 Expiry point 2의 Offset 값에 도달하면 Protection errorhook이 호출되도록 한다. Protection errorhook이 발생하면 API/StopScheduleTable를 사용하여 Schedule Table을 종료시킨다.

Ⅳ. 연구 평가

연구 및 평가를 위해 ASIL D를 할당받은 ESC Runnable, HOST Runnable, ASIL B를 할당받은 PBC Runnable 3가지를 사용한다. 또한 기존의 방법과 명확한 차이를 확인하기 위해 각 Runnable이 맵핑된 태스크의 목표실행시간은 특정 값으로 고정하여 사용한다. 또한 제동아이템의 기능을 정상적으로 동작시키기 위한 소프트웨어의 실행순서는 ESC, EPB 그리고 HOST 순서가 되어야한다.

태스크의 실행시간은 $\tau.ID.ET$ = 30ms ~ 70ms, 시스템은 반복주기는 $Frame.ET$ = 150ms를 가진다고 특정 한다. 기존 방법의 Scheduling Table은 3개의 Expiry point로 구성되어 있으며 각 Expiry point의 Offset은 50ms씩 차이가 있도록 설계되었다. 반면 제안된 기법을 사용하는 Scheduling Table은 2개의 Expiry point로 구성되며 0ms, 100ms의 Offset을 가진다.

Infineon TC27X core, 16MB의 RAM를 사용하는 ECU에서 시뮬레이션을 통해 연구 평가가 진행되었다. 연구평가 방법은 각 태스크에 맵핑되는 컴포넌트에 XCP를 통한 온라인 캘리브레이션을 사용하였다 [22]. 온라인 캘리브레이션을 통해 ESC 태스크의 실행시간을 조절하며 전체 태스크의 동작을 확인하였다. 캘리브레이션 및 모니터링을 위해 Vector CANoe. Opt AMD 툴을 사용하였다.

평가를 위해 산업에서 발생할 수 있는 상황을 가정한다. 가장먼저 수행되는 ESC 태스크의 기능이 변경되면서 실제실행시간이 70ms까지 증가될 수 있다고 가정한다. 즉 $\tau.ID.ET_{worst}$는 표 1과 같다.

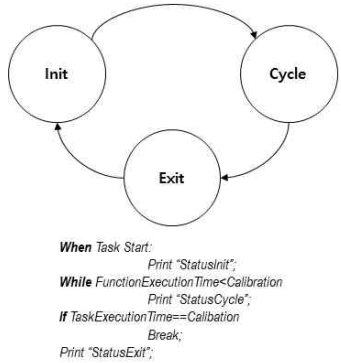

그림 10. 태스크 상태 다이어그램
Fig. 10 Relation of task status

표 1. 평가를 위한 WCET의 변경
Table 1. Change of WCET for evaluation

$\tau.ID.ET_{worst}$	ESC	EPB	HOST
Existing	50ms	50ms	50ms
Needed	Max70ms	50ms	50ms

기존에 사용하던 구조에서는 ESC 태스크의 Execution Budget변경을 위해 EPB 또는 HOST 태스크의 Execution Budget를 감소시켜야 했다. 만약 $\tau.ID.ET_{worst}$을 줄이지 못해 Execution Budget을 50ms로 유지하게 되면 기존 방법은 $Frame.ET$ 150ms을 충족하지 못하게 되며 시스템은 의도하지 않은 동작을 할 수 있다.

이와 같이 소프트웨어의 유지보수가 진행되면서 발생할 수 있는 상황을 예로 들어 기존방법과 제안기법의 효율성을 비교분석한다.

1. Task Composition and Test Environment

태스크의 기능은 태스크의 상태를 외부로 표출하여 CANoe 툴을 이용하여 모니터링 할 수 있도록 한다.

태스크의 상태는 그림 10과 같다. 상태는 Init, Cycle을 거쳐 Exit로 변경되며, 태스크가 시작되면 송출되는 변수 값은 'Init'이 된다. XCP를 이용하여 캘리브레이션이 된 목표시간까지 태스크는 동작하며, 이때 송출되는 변수의 값을 'Cycle'로 변경한다. 그리고 정상적으로 태스크가 종료될 때 송출되는 값이 'Exit'이 되도록 설계되어있다. 즉 정상적인 동작인 경우에는 송출되는 신호가 'Exit'가 되어야하며, 만약 'Init'또는 'Cycle'이 송출된다면 태스크는 정상 종료되지 못한 것이다.

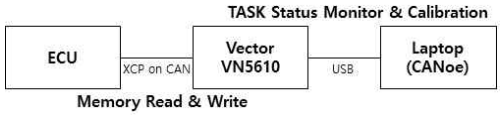

그림 11. 테스트 환경
Fig. 11 Test environment

시험환경은 그림 11과 같이 구성하였다. ECU와 VN5610장비는 XCP on CAN 프로토콜을 이용하여 연결하였으며 CANoe 툴을 이용하여 모니터링과 캘리브레이션을 진행하였다.

2. Comparison of methods

우리는 ESC 태스크의 실행시간을 증가시켜 제안기법의 효율성을 파악하였다. 먼저 각 태스크의 실행시간을 조정하기 위해 적정 캘리브레이션 값을 찾았다. 태스크의 실행시간별 캘리브레이션 값은 표 2와 같다.

먼저 ESC 태스크의 실행시간을 50ms, EPB 태스크의 실행시간을 30ms, HOST 태스크의 실행시간을 50ms로 설정한 뒤, XCP를 이용하여 ESC 태스크의 실행시간을 70ms으로 증가시켰다.

그리고 기존의 Schedule Table 방법, 그리고 제안된 방법에서의 스케줄링을 비교하였다. 기존의 Schedule Table은 기 언급된 것처럼 각 태스크의 WCET를 고려하여 Offset으로 ESC 0ms, EPB 50ms 그리고 HOST 100ms을 가지게 하였고. 제안된 구조는 ESC, EPB는 0ms의 Offset, HOST는 100ms의 Offset을 가진다.

ESC 태스크의 실행시간이 70ms으로 증가되자 그림 12와 같이 기존의 Schedule Table 방법에서는 HOST 태스크가 정상종료 되지 않고 문맥교환이 일어나는 것을 알 수 있다.

제안된 방법에서는 공유자원을 이용하여 정상적인 동작을 하는 반면, 기존의 방법에서는 ESC 태스크가 설계된 최대시간보다 20ms 더 동작함으로써 후순위의 태스크가 지연되어 결국 HOST 태스크는 동작을 정상적으로 완료하지 못하고 다음주기로 넘어가게 된 것이다.

각 태스크의 Offset 기준 50ms에서 ESC, EPB 태스크의 실행시간을 증가시킬 때, $Frame.ET$가 150ms 기준으로 정상동작할 수 있는 범위는 표 3과 같다.

제안된 방법은 공유자원을 사용하면서 시스템이 정상동작할 수 있는 자원을 기존의 방법보다 개선시킨다. 본 연구에서 사용한 것처럼 공유자원에 이용되는 태스크의 개수가 2개이며, 각 태스크의 실행

표 2. 태스크 실행시간 별 적정 캘리브레이션 값
Table 2. Calibration value of task execution time

Execution time	Calibration Value
≒30ms	120
≒50ms	200
≒70ms	280

표 3. 공유자원 활용 타당성
Table 3. Validity of using shared resource

ESC(ms)	EPB(ms)	Operation of HOST	
		Existing	Proposed
50	50	O	O
70	30	X	O
30	70	X	O
70	50	X	X
70	70	X	X

시간이 $\tau.ID.ET_{idle}$ = 30ms, $\tau.ID.ET_{worst}$ = 50ms인 경우 시스템은 최대 40ms (2×(50ms-30ms))의 공유자원을 가지게 된다. 한 번의 주기에서 하나의 태스크만 실행시간이 늘어난다면, 기존 20ms 대비 100%만큼 실행시간이 더 늘어나더라도 시스템은 정상 동작될 수 있다.

결론적으로 시스템은 마지막 태스크를 제외한 나머지 태스크들의 실행시간의 합 $\sum_{ID=1}^{n-1} \tau.ID.ET$이 $Frame.ET-\tau.HOST.ET_{worst}$인 100ms보다 크지 않다면 실시간성을 충족시키며 정상동작하는 것을 알 수 있다. 선점형 스케줄링의 구조를 가지되 각 태스크가 동작할 수 있는 최대시간이 늘어난 것이다.

ESC와 EPB 태스크의 실행시간을 각각 70ms, 70ms으로 변경한 경우에는 HOST는 기존의 방법, 제안된 방법 모두 정상동작하지 않는 것을 확인할 수 있었다.

이 경우 HOST의 동작을 살펴보면 기존의 방법은 그림 13과 같이 태스크의 문맥교환이 EPB와 HOST 태스크 두 군데에서 발생하는 것을 알 수 있다. HOST 태스크가 지연되면서 다음 주기의 자원 여유시간에 동작하며 이는 EPB 태스크의 동작에도 영향을 주게 된 것이다. EPB와 HOST 태스크 모두 실행시간이 부족하여 'Exit'를 송출하지 못하고 'Cycle'을 송출하고 있음을 확인할 수 있다.

반면 제안 기법의 구조에서는 HOST 태스크의 동작이 막혀 ESC와 EPB는 정상적으로 동작할 수 있었다. 즉 결함허용시스템을 구성할 때 소프트웨어를 의도된 대로 동작하게 하여 안전한 설계를 구현할 수 있다.

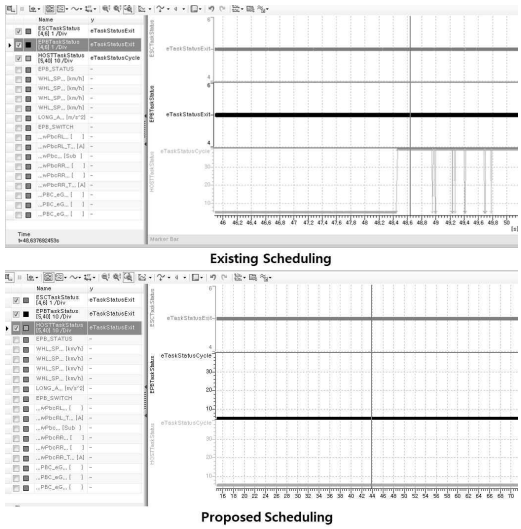

그림 12. 제안기법의 효율성
Fig. 12 Efficiency of the proposed scheme

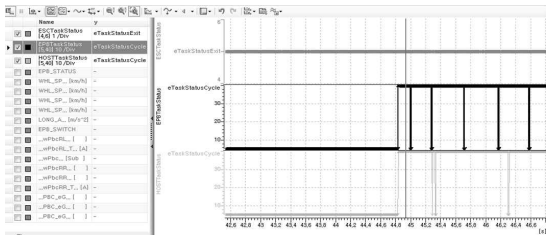

그림 13. 태스크 지연에 의한 영향성
Fig. 13 Issues of task delay

Ⅴ. 결 론

본 연구는 기존의 독립형 시스템 구조에서 통합형 시스템 구조로 변경되는 최근 환경에서 일어날 수 있는 문제점을 다루고 있다.

통합되는 구성요소의 제한된 정보와 한정된 하드웨어 자원 하에서, 시스템을 통합하며 직면하게 되는 스케줄링의 어려움을 해결하기 위한 스케줄링 방법을 연구하였다. 연구대상 아이템은 제동 아이템이며, 제동 아이템을 구성하는 EPB와 ESC 시스템을 통합하는 것으로 한정하여 연구를 진행하였다.

연구 평가를 통해 확인할 수 있듯이 제안기법은 실행시간 초과로 인해 일어나는 문제를 간단하게 개선할 수 있다. 공유자원을 통해 태스크의 기능상실 시점을 최대한 지연시킬 수 있으며, 기능상실이 불가피 하더라도 후순위의 태스크가 기능을 상실하도록 설계하여 시스템의 의도하지 않은 동작을 하

지 않도록 한다.

OSEK기반의 Cooperative 스케줄링의 개념을 가지고 가되, 마지막 태스크가 수행 중에 문맥 교환을 하지 않게 하여 안전한 선점형 구조를 만들 수 있다. 또한 Schedule table의 대기시간을 공유하여 태스크의 실행시간이 늘어나더라도 시스템의 고장이 발생하는 시간을 지연시킬 수 있다. 본 연구에서 가정한 시간기준으로 총 20ms의 추가 자원을 사용할 수 있음을 확인하였다.

본 연구의 단점으로는 마지막 태스크의 동작여부를 WCET기준으로 판단하는 것이다. 만약 마지막 태스크의 실제동작에서의 실행시간이 $\tau.ID.ET_{idle} < T_{remain} < \tau.ID.ET_{worst}$ 라면 마지막 태스크가 충분히 동작할 수 있는 자원이 있음에도 불구하고 동작이 금지된다.

하지만 제동시스템과 같은 안전관련 시스템은 시스템의 결함발생시 예측 가능한 범위 안에서 동작하는 것이 무엇보다 중요하고 생각한다. 소프트웨어의 스케줄링 타이밍에 대한 보호는 고장 허용 향상에 적절한 안전 메커니즘이기 때문이다.

앞으로 하드웨어와 운영체제가 발전하는 만큼 통합된 구조를 사용하는 많은 임베디드 시스템들이 생길 것이며 그로 인해 스케줄링 방법론은 통합 시스템의 구성에 따라 다양해 질 것으로 예상된다. 다른 기능을 수행하는 다수의 소프트웨어가 통합되어 운용되기에 소프트웨어 간 책임소재도 고려를 해야 한다. 점차 기존에 고려하지 않던 많은 정보들을 스케줄링 시 고려해야 할 것이다.

본 연구를 시작으로 차량용 소프트웨어 스케줄링의 고장허용 구조에 대한 많은 관심과 보다 효율적이고 실용적인 스케줄링 기법이 연구되기를 기대한다.

References

[1] H. Heinecke, KP. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Lefloure, J. LucMate, K. Nishikawa, T. Scharnhorst, “Automotive Open System Architecture-an Industry-wide Initiative to Manage the Complexity of Emerging Automotive E/E-architectures,” SAE Paper, No. 2004-21-004, 2004.

[2] AUTOSAR “Layered Software Architecture,” 2007.

[3] B Leiner, M Schlager, R Obermaisser, “A Comparison of Partitioning Operating Systems for Integrated Systems,” Proceedings of International Conference on Computer Safety, Reliability, and Security, pp. 342-355, 2007.

[4] VDA, “Recommendation for Integrating Actuators of Electric Parking Brakes Into ESC Control Units,” VDA 305-100, 2015.

[5] The International Organization for Standardization “Functional Safety”, ISO 26262, 2011.

[6] S. Schliecket, J. Rox, M. Negrean, K. Richter, M. Jersak, R. Emst. “System Level Performance Analysis for Real-Time Automotive Multicore and Network Architectures,” Proceedings of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 28, No. 7, pp. 979-992, 2009

[7] C. L. Liu, J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” Journal of the ACM, Vol. 20, No. 1, pp. 64-61, 1973.

[8] K. Kwon, J. Lee, K. Kim, J. Kim, J. Kim “Real-Time Task Scheduling Algorithm for Automotive Electronic System,” Journal of IEMEK J. Embed. Sys. Appl., Vol. 5, No. 2, pp. 103-110, 2010. (in Korean)

[9] G.C. Buttazzo, M. Bertogna, G. Yao, “Limited Preemptive Scheduling for Real-time Systems. a Survey,” Journal of IEEE Transactions on Industrial Informatics, Vol. 9, No. 1, pp. 3-15, 2013.

[10] S. Baruah, A. Burns, R. Davis, “Response-time Analysis for Mixed Criticality Systems,” Proceedings of IEEE Real-Time Systems Symposium, pp. 34-43, 2011.

[11] Q. Zhao, Z. Gu, H. Zeng, “Pt-amc: Integrating Preemption Thresholds Into Mixed - criticality Scheduling,” Proceedings of Design, Automation and Test in Europe, pp. 141-164, 2013.

[12] Q. Zhu, Y. Yang, M. Natale, E. Scholte, A. Sangiovanni-Vincentelli, “Optimizing the Software Architecture for Extensibility in Hard Real-time Distributed Systems,” Journal of IEEE Trans. Ind. Inf., Vol. 6, No.4, pp. 621-636, 2010.

[13] H. Zeng, M.D. Natale, Q. Zhu, “Minimizing

Stack and Communication Memory Usage in Real-time Embedded Applications", Journal of ACM Transaction on Embedded Computing System, Vol. 13, No. 5s, 2014.

[14] D. Sandell, A. Ermedahl, "Static Timing Analysis of Real-time Operating System Code", Proceedings of nternational Symposium On Leveraging Applications of Formal Methods, Verification and Validation, pp. 146-160, 2004.

[15] N. Holsti, T. Langbacka, S. Saarinen, "Using a Worst-case Execution-time Tool for Real-time Verification of the DEBIE Software", Proceedings of EUROPEAN SPACE AGENCY-PUBLICATIONS-ESA S, No. 457, pp. 307-312, 2000

[16] P. Puschner, A. Schedl, "Computing Maximum Task Execution Times - A Graph-based Approach", Journal of Real-Time Syst, Vol. 13, No. 1, pp. 67-91, 1997.

[17] R. Kirner, P. Puschner, "Measurement-Based Worst-Case Execution Time Analysis Using Automatic Test-Data Generation," Proceedings of WCET'04, 2004.

[18] S. Bunte, M. Zolda, M. Tautschnig, R. Kirner, "Improving the Confidence in Measurement-based Timing Analysis," Proceedings of IEEE International Symposium Object / Component / Service-oriented Real-time Distributed Computing, 2011.

[19] I. Wenzel, R. Kirner, B. Rieder, P. Puschner, "Measurement-based Timing Analysis, Leveraging Applications of Formal Methods, Verification and Validation," 2009.

[20] S. Anssi, S. Tucci, S. Kuntz, S. Gerard, F. Terrier, "Enabling Scheduling Analysis for AUTOSAR Systems," Proceedings of IEEE International Symposium on Object, Component, Service-Oriented Real-Time Distributed Computing, pp. 152-159, 2011

[21] J. Choi, Y. Kim, J. Cho, Y. Choi, "The Software FMEA guideline for Vehicle Safety," Journal of Korea Multimedia Society, Vol. 21, No. 9, pp.1099-1109, 2018.

[22] Lemon. K, "Introduction to the Universal Measurement and Calibration Protocol XCP." SAE Technical Paper, 2003.

Junyeol Choi (최 준 열)

He is received MS degree from Sungkyunkwan University in 2012. He has been working at the MANDO brake center since 2015.

Email: trustcjy@gmail.com

Joonhyung Cho (조 준 형)

He is received MS degree from Kyungpook National University in 1996. He has been the team leader of MANDO brake center since 2010.

Email: joonhyung.cho@halla.com

Yunja Choi (최 윤 자)

She is received Ph.D from Minesota University in 2003. She has been a professor at Kyungpook National University since 2010.

Email: yuchoi76@knu.ac.kr