논문 2011-06-19

OSEK/VDX 기반 차량용 RTOS의 구현과 성능 테스트

(An Implementation and Performance Test of Automotive RTOS Based on OSEK/VDX)

조 성 래*, 김 병 준, 진 성 호, 이 준 호
(Sung-Rae Cho, Byung-Joon Kim, Sung-Ho Jin, Jun-Ho Lee)

Abstract : As the architecture of automotive software is complicated and LOC of software increases, there is an increasing demand for standard operating system. OSEK/VDX defines an industry standard for automotive operating system and middleware. This paper introduces an implementation of RTOS(OSCAR-OSEK) based on OSEK/VDX standard. For better speed of scheduling, we suggest a new method for ready queue implementation considering the characteristic of automotive application software. Also we introduce a method for improving the portability of RTOS on various MCUs. And, we show OSCAR-OSEK implementation and performance test results.

Keywords : Automotive embedded system, Standard operating system, OSEK/VDX, RTOS, Performance test, OSCAR-OSEK

Ⅰ. 서 론

현재의 자동차는 사람 및 화물 수송을 위한 기계장치일 뿐만 아니라, 첨단 IT 기술이 적용되어 운전자의 편의성, 안전성 및 각종 인포테인먼트 기능을 제공하는 하나의 큰 전자제품으로 볼 수 있다. 에너지 효율 개선 및 저탄소 정책 추진에 따라 내연기관 엔진을 주동력으로 하는 기존의 자동차 시장보다 전력을 주동력으로 하는 전기자동차 시장이 확대될수록 자동차의 전자화는 가속화될 전망이다. 이러한 차량들은 다양한 전자제어장치(ECU, Electronic Control Unit)를 탑재하고 있으며 그 수는 날로 증가하고 있다. 이에 따라 ECU에서 동작하는 소프트웨어의 LOC(Lines of Code)가 급증하고 있으며, 효율적인 소프트웨어 개발과 ECU 공급기업간의 소프트웨어 호환성 문제가 대두되고 있다.

OSEK/VDX와 AUTOSAR는 차량용 실시간 운영체제 및 미들웨어에 대한 대표적인 산업 표준이다[1-2]. 그 중에서 OSEK(Open Systems, and the Corresponding Interfaces for Automotive Electronics)은 차량용 분산 제어장치의 개방형 소프트웨어 아키텍처를 위한 산업표준을 제정하기 위해서 1993년에 독일 자동차업계들이 주도하여 만든 공동 프로젝트로 시작되었다. 초기에 BMW, Bosch, DaimlerChrysler, Opel, Siemens 등이 참여하였고 프랑스 자동차 업계의 유사한 프로젝트인 VDX(Vehicle Distributed eXecutive)의 PSA와 Renault가 함께 참여함으로써 OSEK/VDX 프로젝트가 구성되었다.

OSEK/VDX의 규격 명세(Specification)는 크게 ECU 소프트웨어의 실시간 실행을 위한 운영체제(Operating System), ECU 및 태스크 간의 데이터 교환을 위한 통신(Communication), 차량내부 통신의 설정, 관리 및 모니터링을 위한 네트워크 관리(Network Management)로 구성되어 있다. 주로 유럽에서 표준화를 주도하고 있으며 독일의 Vector, Electrobit, 및 미국의 Freescale 등의 임베디드 소프트웨어 전문 기업들이 OSEK OS 규격 명세에 따라 RTOS를 개발하여 공급하고 있다. 이에 따라 해

* 교신저자(Corresponding Author)
논문접수 : 2011. 04. 15., 수정일: 2011. 05. 06.,
채택확정 : 2011. 06. 02.
조성래, 김병준, 진성호 : 대구경북과학기술원
이준호 : (주)와이즈랩
※ 본 연구는 교육과학기술부의 대구경북과학기술원 기관고유사업의 일환으로 수행하였음.[11-RS-03, 고신뢰성 임베디드시스템 핵심기술 개발]

외의 자동차 부품기업 및 OEM 들은 이를 이용한 전장부품을 개발하고 있다. 그러나 국내 자동차 부품 기업들은 아직 전장부품 개발에 RTOS를 적극적으로 활용하고 있지 않다. 대부분 펌웨어 기반의 소프트웨어를 사용하고 있는 실정이며, 소프트웨어 재사용 및 관리의 어려움과 표준화 요구에 대응하기 위해 RTOS 및 소프트웨어 플랫폼의 도입이 절실히 요구되고 있다.

본 논문에서는 OSEK OS 규격 명세에 기반한 RTOS(OSCAR-OSEK) 구현 및 성능 테스트에 대한 내용을 기술한다. Ⅱ장에서는 OSEK OS 규격 명세 및 기존의 관련 연구에 대한 내용을 살펴보고, Ⅲ장에서는 OSEK OS의 구현 및 성능 테스트 결과에 대해서 기술하고 Ⅳ장에서 결론을 맺는다.

Ⅱ. 관련 연구

1. OSEK/VDX 규격 명세

그림 1은 OSEK OS와 Communication, Network Management 모듈간의 관계를 간략히 보여준다.

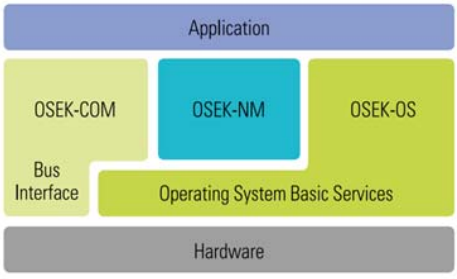

그림 1. OSEK/VDX 모듈 구조
Fig. 1. OSEK/VDX modules

OSEK OS 모듈은 차량용 소프트웨어의 실시간 수행을 위한 핵심모듈이며 애플리케이션(Application) 뿐만 아니라, OSEK NM, 및 OSEK COM의 일부 모듈이 OSEK OS 상에서 실행된다.

OSEK OS 규격 명세는 버전 2.2.3이 최신문서로 태스크, 인터럽트, 이벤트, 리소스, 알람 등으로 구성된다 [3]. OSEK OS는 OIL(OSEK Implementation Language)을 지원하는 환경설정 및 코드생성 도구를 통해 OS 실행 전에 애플리케이션(application)이 필요로 하는 OS 오브젝트(Object)를 OIL 파일로 기술한다 [4]. 코드생성도구는 OIL 파일을 기반으로 C로 작성된 OS 오브젝트 정보를 포함하는 소스코드를 생성하고 이를 사용해서 OS의 커널이 동작하게 된다. 그리고 OS 실행 후에는 OS 오브젝트 정보와 관련된 사항은 추가, 삭제 및 변경이 되지 않는다. 본 절에서는 OSEK OS의 대표적인 기능을 살펴본다.

1.1 OS의 태스크 관리

OSEK OS의 태스크는 기본 태스크(Basic Task) 또는 확장 태스크(Extended Task)의 타입을 가지며 정적으로 부여된 우선순위 및 선점(Preemption) 가능여부에 대한 속성을 가지고 있다. 그림 2는 태스크 상태 전이 모델을 보여준다.

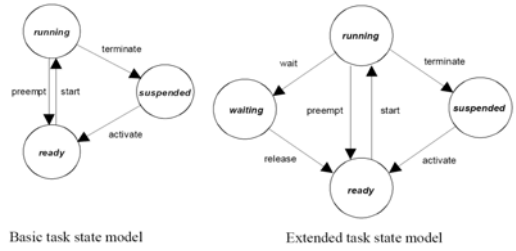

그림 2. 태스크 타입별 상태 전이 모델
Fig. 2. Task transition model

기본 태스크는 suspended, ready, running 중 한 가지 상태(state)로 존재할 수 있다. 각 상태간의 전이는 activate, start, preempt, terminate와 같은 네 가지 이벤트들에 의해서 발생된다. running상태에 있는 기본 태스크는 더 높은 우선순위를 가진 태스크에게 선점(preemption)을 당하지 않으면 태스크 내의 모든 실행코드의 수행을 완료한다. 확장 태스크는 기본 태스크의 상태전이 모델에 waiting 상태가 추가되어 있어 이벤트를 받기 위해 이 상태로 전이할 수 있다. waiting 상태에 있는 태스크가 기다리는 이벤트를 받으면 ready 상태로 전이하여 스케줄러로부터 CPU 할당을 기다리게 된다.

1.2 스케줄러의 동장 방식

OSEK OS의 스케줄러는 우선순위에 기반한 FIFO(first in first out) 알고리즘을 사용한다. 우선순위는 최소 8개를 지원하여야 하며 우선순위 값이 클수록 스케줄러는 해당 태스크가 CPU를 우선적으로 할당 받을 수 있도록 한다. 그림 3은 OSEK OS 스케줄러가 다음 실행할 태스크를 선택하는 과

정을 보여준다.

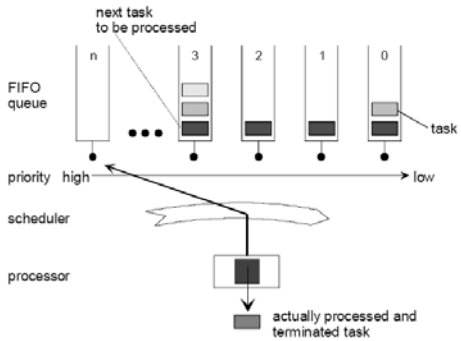

그림 3. 스케줄러의 동작 방식
Fig. 3. Principles of scheduler operation

OSEK OS는 우선순위 개수와 동일한 FIFO큐를 가지고 있으며 각 큐에 해당 우선순위의 태스크가 대기한다. 태스크가 suspend 상태이거나 waiting 상태에서 ready 상태로 전이 될 때 해당 태스크는 FIFO 큐의 맨 뒤에 들어가지만, 높은 우선순위 태스크에 의해서 running 상태에서 선점되는 태스크는 해당 큐의 맨 앞에 들어가서 다음 스케줄 과정에서 우선적으로 선택될 수 있도록 보장을 받는다.

1.3 코드 생성 방법

OIL(OSEK Implementation Language)은 OSEK OS의 오브젝트와, OSEK COM의 메시지를 정의하는 규약이다. OSEK OS는 메모리 사용의 최소화 및 최적실행을 위해서 오브젝트의 동적 생성을 지원하지 않고, OIL에 의해서 개발초기 단계에 정적으로 생성하도록 하고 있다.

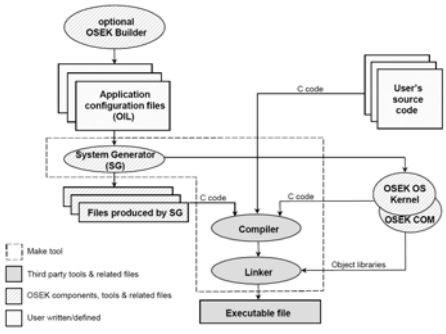

그림 4. 애플리케이션 개발 과정
Fig. 4. Development process for applications

그림 4는 OIL을 이용한 OSEK OS 의 애플리케이션 개발 과정을 보여준다.

먼저 개발자는 OSEK Builder를 사용하여 애플리케이션이 사용할 태스크, 이벤트, 자원, 메시지들을 정의한다. 각 태스크의 우선순위나 메시지의 크기, 사용될 통신 옵션의 종류 등을 정하여 OIL 파일로 저장하게 된다. 다음으로 저장된 OIL 파일을 기반으로 System Generator를 통해 OS 오브젝트와 관련된 C 코드를 생성한다. 마지막으로 생성된 C 코드, OSEK OS 커널 및 애플리케이션 C 코드를 통합 컴파일 하여 최종적인 이미지를 생성하게 된다 [4].

1.4. 적합성 테스트와 인증

OSEK OS의 적합성 테스트 관련 표준화 작업은 MODISTARC(Methods and tools for the validation of OSEK/VDX based DISTributed ARChitectures) 프로젝트가 담당을 하고 있다 [5].

OSEK OS의 적합성 테스트는 블랙박스 테스트 방법을 취하며 OSEK OS 규격 명세에서 정의된 OS API가 정상적으로 동작하는지를 적합성 테스트용 애플리케이션을 통해서 확인한다. 그림 5는 OSEK OS의 적합성 테스트 구조를 보여준다.

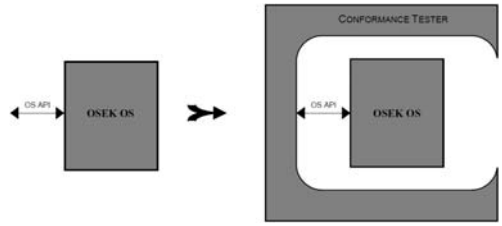

그림 5. OSEK OS 적합성 테스트
Fig. 5. OSEK OS conformance test

OSEK/VDX는 OSEK OS의 적합성 테스트 및 인증을 독일의 MBTech사에 위임하여 수행한다. OSEK OS의 인증 절차는 그림 6과 같다.

MBTech사는 MODISTARC 규격에 따라 구현된 testsuite를 제공한다. 개발된 OSEK OS의 인증을 원하는 기관은 testsuite를 구매한 후, 해당 testsuite를 OS가 개발된 타겟보드에 포팅 한다. 포팅이 완료되면 testsuite를 적용하여 OS의 적합성 테스트를 수행한다. 성공적으로 테스트가 완료되면 테스트 결과 파일, 포팅된 testsuite, OS Implementation Document, Requested Information for OSEK Certification Approval 문

서를 MBTech사에 송부한다. MBTech사에서는 문서 및 결과파일을 검토한 후 인증 승인 문서(Certification approval Document)를 발행한다 [6].

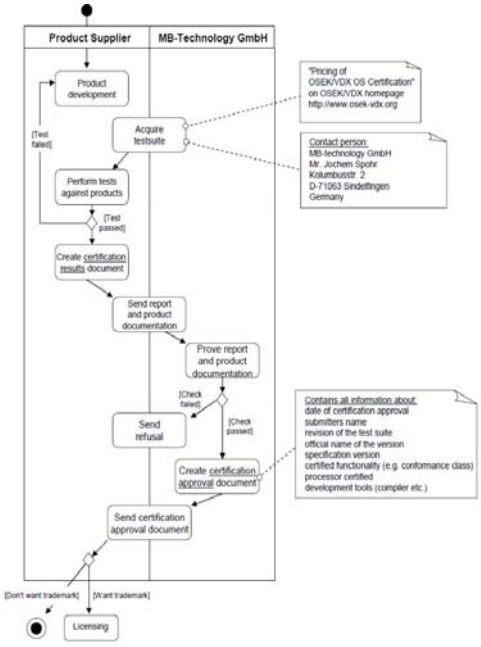

그림 6. 적합성 테스트 및 인증 절차
Fig. 6. Conformance test and certification process

2. OSEK OS 개발 및 관련 연구

OSEK OS 규격은 2005년에 버전 2.2.3이 배포되어 있어 국내외의 다양한 기업, 학교, 연구소가 개발을 진행하고 있다. 본 절에서는 국내외 OSEK OS 개발 현황 및 관련연구에 대해서 살펴본다.

2.1 OSEK OS 개발

OSEK OS는 대표적으로 독일의 Vector, Electrobit 및 미국의 Freescale사에서 관련 제품을 출시하고 있다. Vector 및 Freescale사는 OSEK OS 버전 2.2 기반으로 인증을 획득한 제품을 공급중이고, Electrobit사는 ORTI(OSEK Runtim Interface) 및 CAN 프로토콜 스택을 포함한 제품을 출시하고 있다. 또한 오픈 소스 프로젝트로 프랑스의 Trampoline이 있으며 대학 등에서 연구 목적으로 사용 중이다.

국내에서는 대학 및 연구소 등에서 연구 및 선행 개발을 위해 OS 개발하고 있으며 DGIST와 ETRI에서 OS 개발을 완료하여 OSEK/VDX 인증을 획득한 상태이다.

2.2 국외 관련 연구

Trampoline은 프랑스의 연구기관인 IRCCyN에서 주도하고 있는 OSEK OS 오픈 소스 프로젝트이다 [7]. 리얼타임 커널, 코드생성 도구 및 애플리케이션 실행을 위한 가상환경으로 구성되어 있으며 GPL 라이센스 정책을 따르고 있기 때문에 누구나 소스 코드 분석 및 이를 이용한 애플리케이션 개발이 가능하다.

그림 7은 Trampoline의 대기 태스크 집합의 리스트 구조이다. Task Descriptor는 이중 연결리스트 구조를 가지고 있으며 task set의 맨 앞의 노드는 다음 우선순위의 노드를 가리키는 포인터를 함께 가지고 있다 [8].

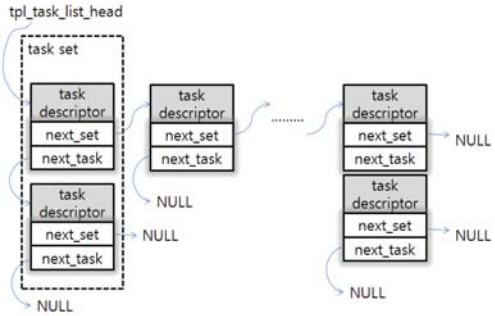

그림 7. Trampoline의 대기큐 구조
Fig. 7. Ready queue structure of Trampoline

이 구조는 스케줄러가 가장 우선순위가 높은 태스크를 찾을 때 tpl_task_list_head가 지칭하는 태스크를 참조하면 되므로 태스크 선택 속도가 빠르다는 장점이 있다. 또한 실제 태스크가 할당된 우선순위에 대한 태스크셋(task set)을 가지므로 메모리를 절약할 수 있다. 그러나 우선순위가 낮은 태스크가 다른 상태에서 대기상태로 진입할 때, 해당 우선순위의 서브셋으로 진입하기 위해 연결 리스트(Linked List) 구조의 서브셋을 따라 탐색을 해야 한다. 또한 낮은 우선순위의 태스크가 waiting 상태에서 해제된 경우에는 해당 태스크셋의 태스크 리스트를 따라 탐색해서 맨 뒤쪽에 연결해야 하므로 탐색시간이 오래 거릴 수 있다.

Trampoline은 별도의 OIL 파일 생성을 위한 GUI 기반의 환경설정도구를 제공하지 않는다. OIL

은 비교적 간단한 문법이라 텍스트 편집방식으로 OIL 파일을 기술하는 것이 가능하다. 기술된 OIL파일을 이용해서 C로 작성된 소스파일을 생성하기 위해서 GNU M4 매크로로 작성된 System Generator를 사용하고 있다. M4 매크로는 대부분의 UNIX 기반 시스템에 소스코드 패키지 환경설정을 위해서 널리 사용되는 도구로 윈도우즈 시스템에서도 포팅 되어 있기 때문에 오픈소스 프로젝트에 자주 이용된다.

2.3 국내 관련 연구

OSEK OS 및 자동차 전장용 실시간 운영체제 관련 국내 논문은 2009년에 발표된 OSEK/VDX 기반 차량 전장용 응용개발도구와 관련된 논문이 있다 [9]. [9]의 연구는 Eclipse CDT 기반으로 OIL 편집, EEPROM 관리 및 디버깅 기능을 지원하는 통합개발환경의 구조와 현재 개발 진행 상황을 소개하였다. 2010년에는 자동차 전장용 실시간 태스크 스케줄링 알고리즘을 소개한 논문이 발표되었다 [10]. [10]에서는 uC/OS-II 실시간 운영체제의 준비 리스트 모델을 개선하여 동일한 우선순위의 태스크를 최대 8개까지 둘 수 있는 시간결정성 스케줄링 알고리즘을 제안하였다.

3. RTOS의 성능 평가

RTOS는 차량, 로봇, 항공, 자동제어와 같이 다양한 산업분야에 사용되고 있으며 해당분야에 따라 다른 환경과 속성을 가지고 있다. 따라서 해당 산업분야별로 RTOS의 성능 평가에서 중요시하는 척도가 다를 수 있다. 예를 들어 항공 분야와 같은 곳은 개발에 소요되는 비용도 중요하지만 안전성이 무엇보다도 중요시 되고 있다.

일반적으로 RTOS에서 중요하게 분류하는 성능 평가의 척도는 다음과 같다 [11].

- 성능(Performance) : RTOS가 주어진 명령어들을 얼마나 빠르게 수행하는가? 이 척도는 주어진 명령어를 정해진 시간 내에 수행하는지 또는 다른 RTOS와의 수행시간을 비교하여 평가 할 수 있다.
- 안전성(Safety) : 실시간 시스템(Realtime System), 특히 Safety-critical System은 그 시스템이 항상 결정적 방식(Predeterminable way)으로 동작하는 것을 보장해야 한다.
- 융통성(Flexibility) : 하나의 RTOS를 다양한 하드웨어 환경에서 사용하기 위해서 타겟보드에 쉽게 적용할 수 있는가는 중요한 요소이다. 특히 디버거, 컴파일러 및 소스코드의 활용성 또한 중요한 척도가 된다.
- 비용(Cost) : RTOS 선택의 모든 경제적인 요인이 포함된다. 비용은 OS의 라이센스를 구매하기 위한 직접적인 비용 외에도 기술지원과 같은 간접비용도 포함된다.

이 중에서 RTOS의 성능(Performance)을 평가하기 위해서 주로 사용하는 척도는 다음과 같다

- 태스크전환 성능(Task-switch performance) : RTOS의 커널이 실행중인 하나의 태스크에서 다른 태스크로 전환하는데 소요되는 시간을 측정
- 메모리할당 성능(Memory allocation performance) : 메모리 공간의 동적 할당 및 해제를 수행하는데 소요되는 시간을 측정
- 인터럽트 지연성(Interrupt latency) : 인터럽트가 발생하고 ISR이 실행될 때까지 소요되는 최대 지연 시간
- 초기화 시간(Initialization time) : 시스템이 리셋하고 난후 초기화를 마칠 때 까지 소요되는 시간

III. OSEK OS 구현 및 성능 테스트

1. 스케줄링 속도 개선 방안

1.1 차량용 애플리케이션 소프트웨어의 특징

차량용 애플리케이션 소프트웨어는 대부분 센서 또는 타이머를 통해 산발적 또는 주기적인 이벤트에 반응하여 액추에이터를 동작하는 로직을 가지고 있다. CAN이나 FlexRay와 같은 차량 네트워크 프로토콜 스택을 고려하더라도 인포테인먼트(Infortainment) ECU를 제외하면 실행파일 크기가 수십 킬로바이트에서 1메가바이트를 넘지 않는 비교적 간단한 태스크 구조를 가지고 있다.

1.2 우선순위 재정의 방법

OSEK OS는 최소 8개 이상의 우선순위를 요구하고 있으나 대부분의 상용 제품이 16 개 이상의 우선순위를 지원한다. 그러나 지원하는 우선순위에 비해 실제 사용되는 우선순위 개수는 크게 많지 않기 때문에, 효율적인 대기큐 관리를 위해서 우선순위를 재정의해서 사용 할 수 있다. 사용자 태스크를 위해서 지원하는 우선순위가 1~255 라고 가정

할 때, 수식 (1)과 같이 우선순위를 재정의 한다.

$$RP(x) = PP(x) - n(NPBOP(x)) \quad (1)$$

where n(NPBOP(x)) : the number of NPBOP(x),
NPBOP(x) : {1..PP(x)}} && NUP

수식 (1)에서 PP(x)는 OIL 문법을 통해 사용자가 지정한 태스크의 우선순위이고, RP(x)는 태스크가 RTOS에서 실행될 때 사용하기 위해서 코드생성 도구가 재정의한 우선순위이다. NUP는 운영체제가 지원하는 우선순위 중에서 태스크가 할당되지 않아 실행 중에 사용하지 않는 우선순위들을 나타내므로 n(NPBOP(x))는 1~PP(x)사이에서 사용하지 않은 우선순위의 개수를 나타낸다. 표 1은 PP에 코드생성도구가 재정의 하는 RP의 예를 나타낸다.

표 1. 우선순위 재정의 예
Table 1. Example of priority redefinition

Task ID	PP	RP
Task1	2	1
Task2	2	1
Task3	5	2
Task4	5	2
Task5	7	3

1.3 대기큐의 구조

우선순위 재정의 방법을 사용할 경우, 그림 8과 같이 코드생성단계에서 실제 태스크가 할당된 우선순위를 기반으로 대기큐-태스크 포인터 테이블을 동적으로 생성한다. RTOS는 재정의 된 우선순위인 RP를 사용하여 태스크 우선순위에 따라 대기큐 관리를 수행한다.

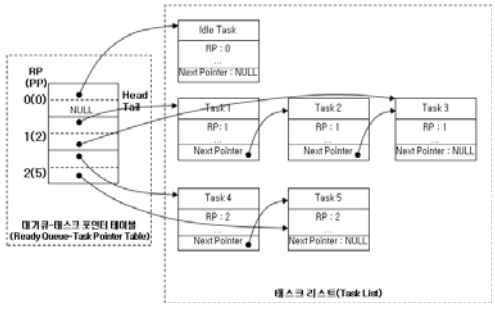

그림 8. 대기큐의 구조
Fig. 8. Ready queue structure

대기큐를 위한 태스크 포인터 테이블의 각 인덱스는 각각 RP에 해당하는 태스크의 리스트를 탐색하기 위해 Head와 Tail에 대한 포인터를 가지고 있다. 따라서 태스크가 ready 상태로 전이 되어 대기큐에 삽입할 때 태스크의 우선순위 값이 인덱스 정보로 사용되므로 해당 위치를 바로 찾아갈 수 있다. 또한 리스트의 Tail 정보를 가지므로 태스크 리스트를 탐색할 필요 없이 해당 큐의 맨 뒤쪽에 태스크를 신속하게 연결할 수 있다.

2. 이식성 향상 방안

OSEK OS의 표준 규격은 애플리케이션 소프트웨어의 이식성(Portability)에 초점이 맞춰져 있다. 소프트웨어 개발 생산성을 높이기 위해서는 애플리케이션의 이식성뿐만 아니라, RTOS 커널 소프트웨어도 다양한 MCU에 쉽게 이식되어야 한다. 본 논문에서 소개하는 RTOS는 이식성 향상을 위해 다음과 같이 AUTOSAR 버전 3.0 규격 명세의 세가지 추상화 계층을 적용하였다 [12-14].

- 하드웨어 플랫폼 추상화 계층
- 컴파일러 추상화 계층
- 메모리 추상화 계층

```
/* The compiler abstraction shall define the FUNC macro for the declaration
   and definition of functions, that ensures correct syntax of function
   declarations as required by a specific compiler. - used for API functions
   rettype      return type of the function
   memclass     classification of the function itself
 */
#define FUNC(rettype, memclass) rettype memclass

/* Pointer to variable data
   ptrtype      type of the referenced data
   memclass     classification of the pointer's variable itself
   ptrclass     defines the classification of the pointer's distance
 */
#define P2VAR(ptrtype, memclass, ptrclass) ptrtype memclass * ptrclass

/* Pointer to constant data
   ptrtype      type of the referenced data
   memclass     classification of the pointer's variable itself
   ptrclass     defines the classification of the pointer's distance
 */
#define P2CONST(ptrtype, memclass, ptrclass) const ptrtype memclass * ptrclass
```

그림 9. 컴파일러 추상화 계층
Fig. 9. Compiler abstraction layer

하드웨어 플랫폼 추상화 계층은 운영체제 소프트웨어에 대해서 CPU 독립적인 데이터 타입 관련 정보들을 제공하며, boolean, uint8~32, sint8~32, float, 32, float64 등의 데이터 타입과 Byte Ordering 정보가 이에 해당한다. 컴파일러 추상화 계층은 다양한 컴파일러가 사용하는 컴파일 키워드를 처리하기 위한 매크로를 제공하며 변수, 상수 및 함수에 대한 포인터 등이 이에 해당한다.

메모리 추상화 계층은 운영체제 커널의 소스코드와 데이터의 메모리 매핑정보를 MCU 독립적으로 적용하기 위한 추상화 계층이다.

그림 9는 컴파일러 추상화계층을 구현하는 헤더 파일의 일부분이다.

“#define FUNC(rettype, memclass) rettype memclass” 라는 매크로에서 rettype은 함수의 리턴타입을, memclass는 함수가 호출될 때 사용되는 메모리 클래스, 예를 들면 “far”, “near”와 같은 컴파일러 키워드를 의미한다. 커널 소스코드 내에서 함수를 작성할 때 그림 9의 매크로를 사용함으로써 컴파일러의 종류가 바뀌어서 함수 기술 방법이 달라지더라도 매크로의 수정만으로 기존의 커널 소스코드를 그대로 재사용할 수 있도록 한다.

그림 10은 컴파일러 추상화 계층이 적용된 OS 소스코드로 알람을 처리하기 위한 모듈이다.

```
#include "Os.h"

extern P2CONST(alarm_ctrl_type, OS_APPL_CONST, OS_CONST) AcbTbl[];

#define OS_START_SEC_CODE
#include "Memmap.h"

FUNC(void, OS_CODE) InitAlarmList
    (
    void
    )
{
    VAR(alarm_id_type, AUTOMATIC) i;
    P2VAR(alarm_ctrl_type, OS_APPL_DATA, AUTOMATIC) alarm;

    for (i = 0; i < NUM_ALARM; i++)
    {
        alarm(P2VAR(alarm_ctrl_type, OS_APPL_DATA, AUTOMATIC))AcbTbl[i];
        if (alarm->State == ALARM_STATE_AUTO_RUN)
        {
            InsAlarmList(alarm);
        }
    }
}
```

그림 10. 추상화 계층이 적용된 소스 코드
Fig. 10. Source codes applied abstraction layer

3. 구현 환경 및 지원 사양

OSEK OS 규격 명세 버전 2.2.3 및 OSEK COM 규격 명세 버전 3.0.3을 기반으로 차량용 임베디드시스템을 위한 RTOS(OSCAR-OSEK)를 구현하였다. OSCAR-OSEK은 1, 2절에서 소개한 우선순위 재정의 방법 및 추상화 계층을 적용하여 경량 태스크들을 빠르게 스케줄링 할 수 있을 뿐만 아니라, 이식성이 높게 구현하였다.

OSCAR-OSEK에서 제공하는 OS 오브젝트 관련 지원 사양은 그림 11과 같다. 그림에서 회색배경으로 표시된 부분은 OSEK OS 규격 명세에서 요구하는 최소 사양이고 흰색 부분은 OSCAR-OSEK의 지원사양이다.

	BCC1	BCC2	ECC1	ECC2
Multiple requesting of task activation	no	yes	BT: no ET: no	BT: yes ET: no
OSCAR-OSEK	no	yes	BT: no ET: no	BT: yes ET: no
Number of tasks which are not in the *suspended* state	8		16 (any combination of BT/ET)	
OSCAR-OSEK	256		256 (any combination of BT/ET)	
More than one task per priority	no	yes	no (both BT/ET)	yes (both BT/ET)
OSCAR-OSEK	no	yes	no (both BT/ET)	yes (both BT/ET)
Number of events per task	-		8	
OSCAR-OSEK	-		16	
Number of task priorities	8		16	
OSCAR-OSEK	256			
Resources	RES_SCHEDULER		8(including RES_SCHEDULER)	
OSCAR-OSEK	256(including RES_SCHEDULER)			
Internal resources	2			
OSCAR-OSEK	256			
Alarm	1			
OSCAR-OSEK	256(per task), 65536(per system)			
Application Mode	1			
OSCAR-OSEK	1			

그림 11. OSCAR-OSEK 지원 사양
Fig. 11. Support features of OSCAR-OSEK

그림 12. OSCAR-OSEK 지원 MCU
Fig. 12. Support MCUs of OSCAR-OSEK

OSCAR OSEK은 Freescale사의 MC9S12-XDP512(16비트 MCU) 뿐만 아니라 비교적 개발환경 구축이 저렴한 초소형 MCU인 ATMEL사의 ATmega128A(AVR 8비트 MCU) 및 ST-Microelectronics사의 STM32F103RBT6(ARM Cortex-M3 32비트 MCU) 상에도 적용을 완료하였다.

4. 성능 테스트 결과

RTOS는 기본적으로 태스크들이 정해진 시간내에 실행될 수 있도록 스케줄링을 보장하여야 한다. 이를 위해서 우선순위 기반의 선점형 커널을 필수적으로 제공하여야 하며 OSCAR-OSEK은 이를 지원한다.

Ⅱ장의 3절에서 기술한 바와 같이 일반적으로 RTOS 성능 평가와 관련한 4가지 척도는 태스크전환(Task-switch), 메모리할당(Memory-allocation), 인터럽트 지연성(Interrupt latency), 초기화 시간(Initialization time)이다. OSEK OS는 실행 중에 동적으로 메모리할당을 수행하지 않으므로 이는 평가항목에서 제외한다. 또한 초기화 시간은 사용한 태스크 및 OS 오브젝트의 수에 따라 달라지고, OSEK OS와 같은 경량 OS에서는 매우 짧은 시간내에 수행이 완료되므로 성능평가에 큰 영향을 주지 않는다.

인터럽트 지연시간은 인터럽트 발생시점을 타겟보드에서 정확하게 측정하기 어렵고, 인터럽트가 발생된 시점의 실행코드에 따라 결과가 달라지므로 성능 평가가 쉽지 않다. 마지막 평가항목인 태스크 전환 시간은 RTOS 성능평가에 가장 중요한 요인이 될 수 있다. 태스크전환 시간이 짧을수록 애플리케이션 태스크의 실시간 실행 성능이 높아진다.

OSCAR-OSEK은 태스크의 우선순위 재정의 방법을 이용하여 대기큐의 구조를 단순화 하였다. OSCAR-OSEK의 태스크전환 성능 테스트를 위해 3개의 사용자 태스크가 멀티태스킹으로 동작하는 테스트 애플리케이션을 작성하였다. 테스트에 사용된 타겟보드는 Freescale사의 MC9S12XDP512 이며 적용 클럭은 40MHz 이다.

그림 13은 OIL 환경설정 도구를 이용하여 테스트 애플리케이션을 설정한 화면이다.

3개의 태스크 중 Task1은 확장 태스크(Extended Task)로 정의하여 이벤트를 수신할 수 있게 설정하였다. 3개의 태스크는 그림 14과 같은 순서로 동작한다.

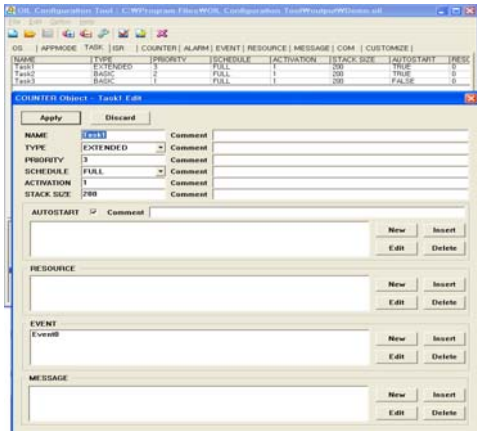

그림 13 OIL 환경설정 도구 및 설정
Fig. 13. OIL Configurator and configuration

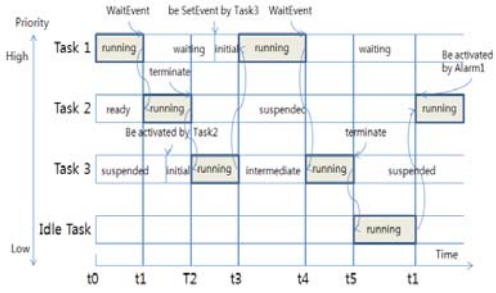

그림 14. 테스트 애플리케이션의 실행 순서
Fig. 14. Execution sequences of test application

OSCAR-OSEK이 초기화 된 후, 우선순위가 가장 높은 Task1이 먼저 실행을 하다가 *WaitEvent* 시스템서비스(System service)를 실행하여 ready 상태로 전이한다. 이때 태스크전환이 발생하게 된다. Task1은 Task3가 *SetEvent* 시스템서비스를 통해 전달한 이벤트를 받아 wait 상태에서 ready 상태로 전이하게 되고 우선순위가 Task3보다 높으므로 Task3을 선점하여 실행한다.

그림 15는 태스크전환 시간을 측정한 결과이다.

첫 번째 열은 태스크전환에 소요된 시스템 Tick의 개수이고 두 번째 열은 태스크전환 시간을 us 단위로 환산한 시간이다. 측정 결과, 평균 21.67 System Tick이 소요되었고 타겟보드가 40MHz로 동작하였으므로 평균 0.542 usec에 태스크전환 기능 수행이 완료되었다.

Task Switch	The Number of SystemTick(40MHz)	Context Switching Time(us)
Task1 -> Task2	23	0.575
Task2 -> Task3	22	0.550
Task3 -> Task1	27	0.675
Task1 -> Task3	22	0.550
Task3 -> IdleTask	18	0.450
IdleTask -> Task2	18	0.450
Average	21.67	0.542

그림 15. 태스크전환 시간 측정 결과
Fig. 15. Results of task switch time

IV. 결 론

본 논문은 OSEK/VDX 표준에 기반한 차량용 RTOS의 구현 및 성능 테스트에 대해서 기술하였다. 차량용 애플리케이션의 특성을 이용한 우선순위 재정의 방법과 스케줄링 속도 향상을 위한 대기큐의 구조를 제안하고 이를 구현에 활용하였다. 제안한 방법은 실제 애플리케이션에서 사용하는 태스크의 우선순위만을 고려하여 코드생성단계에서 대기큐를 동적으로 생성하므로 태스크전환 성능을 향상시킬 수 있다. 또한, 이식성 향상을 위해 AUTOSAR에서 제안하는 추상화 계층을 적용하여 RTOS 커널 코드의 이식성을 향상시키는 방향으로 구현을 하였다. 개발된 RTOS의 기능 검증 및 성능 테스트를 위해 간단한 테스트 애플리케이션을 적용하여 실제 타겟보드에서 태스크전환 속도를 측정하였다. 향후 제안한 대기큐 구조를 Trampoline 및 기타 상용 RTOS와 비교하여 성능 평가를 진행할 예정이고 지원하는 MCU 추가 및 안정화 작업을 수행할 예정이다.

참고문헌

[1] OSEK/VDX, http://www.osek-vdx.org.
[2] AUTOSAR, http://www.autosar.org.
[3] OSEK/VDX, "Operating System Version 2.2.3", http://portal.osek-vdx.org.
[4] OSEK/VDX, "OIL : OSEK Implementation Language Version 2.5", http://portal.osek-vdx.org.
[5] OSEK/VDX, "MODISTARC", http://portal.osek-vdx.org.
[6] OSEK/VDX, "Certifications", http://portal.osek-vdx.org.
[7] Trampoline, http://trampoline.rts-software.org.
[8] Jean-Luc B´echennec, Mika¨el Briday, S´ebastien Faucou and Yvon Trinquet, "Trampoline an opensource implementation of the OSEK/VDX RTOS specification". Conference on Emerging Technologies and Factory Automation, pp. 62-69, 2006.
[9] 안성호, 김재영, 김광수, "OSEK/VDX 기반의 차량 전장용 응용개발도구 설계 및 구현", 대한임베디드공학회논문지, Vol.4, No.2, pp. 84-89, 2009.
[10] 권규호, 이정욱, 김기석, 김재영, 김주만, "자동차 전장용 실시간 태스크 스케줄링 알고리즘", 대한임베디드공학회논문지, Vol.5, No.2, pp. 103-110, 2010.
[11] Gu Yang, Wu Zhaohui, Yue Long, "AlphaOS, an automotive RTOS based on OSEK/VDX: design and test", Networking, Sensing and Control, 2005. Proceedings, 2005.
[12] AUTOSAR, "Specification of Platform Types Version 2.2.1", http://www.autosar.org.
[13] AUTOSAR, "Specification of Compiler Abstraction 2.0.1", http://www.autosar.org.
[14] AUTOSAR, "Specification of Memory Mapping 1.1.1", http://www.autosar.org.

저 자 소 개

조 성 래(Sung-Rae Cho)

1998년 : 경일대학교 컴퓨터공학과 학사.
2000년 : 영남대학교 컴퓨터공학과 석사.
현재, 대구경북과학기술원 선임연구원.

관심분야 : 임베디드 소프트웨어, Real-time OS, 임베디드 소프트웨어 테스팅.
Email : srcho@dgist.ac.kr

김 병 준(Byung-Joon Kim)

2007년 : 계명대학교 전자공학과 학사.
2009년 : 계명대학교 전자공학과 석사.
현재, 대구경북과학기술원 연구원.

관심분야 : 임베디드 소프트웨어, Real-time OS.
Email : bjkim@dgist.ac.kr

진 성 호(Sung-Ho Jin)

1989년 : 경북대학교 전자공학과 학사.
1991년 : 경북대학교 전자공학과 석사.
현재, 대구경북과학기술원 선임연구원.

관심분야 : 임베디드 소프트웨어, 스마트 센서 / 액추에이터.
Email : sungho@dgist.ac.kr

이 준 호(Jun-Ho Lee)

1997년 : 경일대학교 전자공학과 학사.
2010년 : 동의대학교 컴퓨터공학과 석사.
현재, (주)와이즈랩 기술개발실장.

관심분야 : 임베디드 하드웨어, Real-time OS.
Email : jhlee@yslab.co.kr