API 및 OIL 변환을 이용한 POSIX 기반 코드의 OSEK/VDX 코드로의 변환

A Transformation from POSIX Based Source Code to OSEK/VDX Source Code Based on API and OIL Translation

송 영 호, 이 태 양, 이 종 덕, 문 찬 우, 정 구 민*, 안 현 식

(Young-Ho Song, Tae-Yang Lee, Jong-Deok Lee, Chan-Woo Moon, Gu-Min Jeong, and Hyun-Sik Ahn)

Abstract: In this paper, we present a transformation method of source code from a POSIX (Portable Operating System Interface) based source code into an OSEK/VDX (Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen/Vehicle Distributed eXecutive) source code. As the electronic parts of automobile systems increase, the use of embedded software in automobile systems is also growing. Accordingly, many electronic systems are designed in automobile system with OSEK/VDX. Otherwise, one of the major problems of embedded software would be portability to other OS's. To enhance the portability and interoperability of embedded software, we propose a source code transformation method from POSIX to OSEK/VDX based on API (Application Programming Interface) translation method. Considering the characteristics of the OSEK/VDX which uses OIL (OSEK/VDX Implementation Language) standard, transformation process is performed with source code transformation and OIL code generation. For the validity of the proposed method, the transformation experiment is given using Micro-C OS II and OSEK/VDX with XC167CI micro-controller.

Keywords: OSEK/VDX, POSIX, RTOS (Real Time Operating System), API Translator (TransPI), CIC (Common Intermediate Code)

1. 서론

자동차의 성능향상을 위한 ECU (Electronic Control Unit)는 오일쇼크 및 배기가스 규제에 대응하기 위하여 1979년 GM (General Motors)에서 연료분사엔진에 처음 적용한 이후로 나날이 그 수가 증가하고 있다[1,2]. ECU의 구동을 위한 소프트웨어 또한 1980년대 말 2천 라인 수준에서 2001년 2백만 라인, 2006년 4백만 라인으로 증가하여 자동차 내 전자시스템의 복잡성이 급증하게 되었다[3]. 이러한 변화에 의하여 소프트웨어 개발에 필요한 시간 및 비용이 늘어나고 있으며 이를 해결하기 위해 소프트웨어 플랫폼의 도입이 증가하는 추세이다.

대표적인 소프트웨어 플랫폼으로 RTOS (Real Time Operating System)가 있다. RTOS는 다양한 작업을 시간에 따라 정확히 관리하고, 시스템 자원을 관리하며 응용프로그램 개발을 위한 일관된 환경을 제공한다. 따라서 복잡한 소프트웨어를 효과적으로 개발할 수 있으며, 시스템의 신뢰성을 향상시킬 수 있다.

차량용 RTOS인 OSEK/VDX (Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen/Vehicle Distributed eXecutive) OS는 차량 내 전자시스템의 표준을 목표로 설계되었으며 이미 국내외를 비롯한 여러 자동차 업계에서 사용하고 있거나 도입을 고려하고 있다. 앞으로 차량

내 소프트웨어 개발 부분에 있어서 OSEK/VDX OS가 차지하는 비중이 증가할 것으로 예측되며, 이에 따라 기존의 프로그램은 OSEK/VDX OS 환경에 맞게 구현해야 하는 경우가 생길 것으로 예상된다[4,5].

예를 들어 POSIX (Portable Operating System Interface) 기반의 운영체제를 기반으로 작성된 차량용 소프트웨어를 OSEK/VDX OS 환경에 맞게 구현하는 경우를 고려해 볼 수 있다. POSIX는 서로 다른 OS간의 이식을 고려하여 IEEE에서 규정한 API (Application Programming Interface) 규격으로 embedded Linux, uC/OS-II 등이 이 규격을 따르고 있으며, 특히 uC/OS-II는 FAA (Federal Aviation Administration)가 항공전자공학 장치에 uC/OS-II를 적용할 수 있다는 승인을 한 이후로 다수의 임베디드 시스템에서 사용되고 있다.

임베디드 SW의 OS 의존도는 매우 크기 때문에 프로그램의 코드를 재사용할 수 있다면 엔지니어링 비용을 약 60%까지 절감할 수 있으며, 품질 향상 및 개발 기간 단축까지 확보할 수 있다[6].

이러한 경우를 고려하여 이중 소프트웨어 플랫폼간의 API 자동 변환을 통해 코드를 재사용하기 위한 연구가 진행되었다[7-9]. 하지만 POSIX 기반 OS들과 달리 OSEK/VDX에서는 특정 응용에 대한 객체(object) 설정 방법을 OIL 표준에서 정의하고 있으며, OIL 파일을 통하여 스케줄링 정책, 동일 우선 순위 허용 여부 및 서비스에 대한 설정을 수행한다[10]. 따라서 프로그램 코드 재사용을 위해서는 API 변환뿐만 아니라 객체의 특성을 고려한 OIL 파일 생성과정이 필요하다[11].

본 논문에서는 이러한 점을 고려하여 POSIX 기반 RTOS 코드를 OSEK/VDX 기반 코드로 변환 시의 고려사항에 대하여 논하고, 기존 발표[12]를 바탕으로 OIL (OSEK/VDX Implementation Language) 파일 생성 과정을 포함한 코드 변환 방법을 제안한다. 또한 uC/OS-II 기반 코드를 OSEK/VDX 기

* 책임저자(Corresponding Author)

논문접수: 2010. 3. 15., 수정: 2010. 4. 15., 채택확정: 2010. 4. 30.

송영호, 이태양, 문찬우, 정구민, 안현식: 국민대학교 전자공학과

(syhf2000@nate.com/babo2sun@naver.com/mcwnt@kookmin.ac.kr/gm1004

@kookmin.ac.kr/ahs@kookmin.ac.kr)

이종덕: LIG넥스원(jdlee81@lignex1.com)

※ 이 논문은 2008년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2008-331-D00456).

반 코드로 변환하여 차량용 마이크로 컨트롤러에 적용하는 사례연구를 통하여 제안 방법의 타당성을 검증한다.

본 논문의 구성은 다음과 같다. II 장에서는 관련연구에 대해 살펴보고, III 장에서 POSIX 기반 소스코드를 OSEK/VDX 기반 코드로 변환하는 방법에 대하여 논한다. IV 장에서는 OSEK/VDX 기반 코드로 변환시 생성되어야 하는 OIL 파일을 CIC (Common Intermediate Code) 파일을 통하여 생성하는 방법에 대하여 알아보고 V 장에서 제안된 방법을 확인하기 위한 사례연구를 수행하고 마지막으로 VI 장에서 결론을 맺는다.

II. 관련 연구

1. OSEK/VDX

OSEK은 차량용 실시간 운영체제로서 BMW, bosch, daimler chrysler, opel, siemens, VW 그리고 IIIT of the university of karlsruhe 가 개발에 참여하였으며 1994년에 프랑스의 차량 제조업체인 PSA와 renault 사가 개발한 차량 내 분산시스템 (VDX: Vehicle Distributed eXecutive)과 병합하면서 OSEK/VDX 로 명명되었다. 현재 OSEK/VDX는 종래의 OS (Operating System), COM (COMmunication), NM (Network Management), OIL 표준규격 이외에 OSEKtime OS, FTCOM (Fault Tolerant Communication) 등을 정의하고 있다.

OSEK/VDX OS는 OSEK/VDX에서 정의하는 실시간 운영 체제로서, 태스크, 이벤트(event), 리소스(resource), 알람(alarm) 등의 객체를 제공하며 각 객체의 속성이 정적으로 고정되어 있는 것이 큰 특징이다. 즉, 어플리케이션 안에서 각 객체의 생성 및 소멸, 그리고 속성의 변경이 불가능하다. 스케줄링 방식은 우선순위 기반으로 선점형, 비 선점형 그리고 두 가지가 결합된 형태의 스케줄링 방법이 제공된다. 또한 동일 우선순위를 허용하며, 동일 우선순위의 태스크는 FIFO 구조를 이용하여 스케줄링 한다.

OSEK/VDX 개발 과정은 OIL 파일을 이용한 OSEK/VDX 어플리케이션의 설정에서부터 시작한다. OIL 파일은 해당 어플리케이션에서 사용되는 객체에 대해 기술한 것으로서, 기술되는 객체는 OS, ISR (Interrupt Service Routine), 태스크, 리소스, 이벤트, 알람 등이 있다. 시스템 생성기(system generator)

그림 1. OSEK/VDX의 개발 절차.

Fig. 1. Development process of OSEK/VDX.

는 OIL 파일을 이용하여 OSEK OS 커널(kernel) 소스를 포함한 다수의 파일을 생성하며 개발자는 태스크, ISR 등의 소스 코드를 작성하여 시스템 생성기로부터 생성된 커널 소스 등과 합친 후 빌드(build)과정을 통해 최종 실행 가능한 이미지를 만들 수 있다. 그림 1은 OSEK/VDX 개발 과정을 보여준다.

2. TransPI

TransPI는 임베디드 소프트웨어 개발을 위한 툴로써 HOPES (Hopes for Parallel Embedded Software) 프로젝트 결과물의 일부분이다[13]. TransPI는 C언어로 구현되어 있으며 그림 2와 같은 구조를 가진다. 입력 파일 구문 분석기, XML로 기술된 변환 규칙 파일 구문 분석기, 패턴, 심벌 그리고 변환 규칙을 이용한 API 변환 툴, 마지막으로 변환 구문을 이용하여 출력파일을 생성하는 툴로 구성된다.

TransPI는 POSIX 표준을 따르는 RTOS에서 일반적으로 제공하는 서비스들을 추상화 시켜 Generic API 형태로 정의하고, 이를 이용하여 이종 RTOS 간 소스코드 변환을 수행한다[14]. 따라서 기존에 만들어진 임베디드 소스코드를 이종 플랫폼 환경에 맞게 다시 개발해야 하는 경우, 소스코드 변환에 소요되는 시간을 줄이고 소스코드의 재 사용성을 증가 시킨다.

TransPI는 역변환과 정변환 기능을 통해 코드변환을 수행한다. 역변환은 특정 RTOS 기반의 C-코드 파일을 추상화된 형태의 CIC 파일로 변환하는 것을 의미한다. 반대로 정변환은 역변환 기능을 사용해 생성된 CIC 파일을 특정 RTOS에서 실행 가능한 C-코드 형태로 변환하는 것을 나타낸다.

TransPI는 아래와 같은 4단계의 순서로 동작한다.

1. 초기화: 사용할 자료구조 정의 및 메모리 할당 등의 초기화 루틴을 수행하며, 입/출력 파일 목록을 확보하여 변환 준비를 마친다.
2. 관련 정보 수집: 입력 태스크 코드를 구문 분석하여 변환 대상이 되는 API를 분류하고, API의 부가정보들을 수집한다.
3. 코드 변환: 패턴을 참조하여 변환 대상이 되는 API들을 분류하고 심벌 테이블, 변환 규칙을 참조하여 변환 구문을 생성한다.
4. 출력 파일 생성: 기존 코드와 앞 단계에서 생성된 변환 구문을 이용하여 최종 출력 코드를 생성하고 메모리 정리 및 종료 코드를 호출한다.

그림 2. TransPI의 구조.

Fig. 2. Structure of TransPI.

III. POSIX 기반 소스 코드의 OSEK/VDX 기반 코드 변환

본 장에서는 TransPI를 이용하여 POSIX 기반의 소스 코드를 OSEK/VDX 기반 코드로 변경 시의 고려 사항에 대하여 논한다. TransPI가 RTOS 간의 소스코드를 변환하는 과정은 다음과 같다. XML 역변환 룰 파일에 정의된 규칙에 따라 RTOS의 API들을 Generic API로 변환하여 추상화된 형태의 CIC 소스코드를 만들어낸다. 그 후 미리 정의된 특정 RTOS에 대한 정변환 룰 파일을 참조하여 CIC 소스코드를 대상 RTOS 기반 소스코드로 변환한다.

따라서 POSIX 기반 소스코드를 OSEK/VDX OS기반 코드로 변환하는 과정은 크게 세 단계로 구분된다.

먼저 기존 RTOS가 제공하는 서비스와 OSEK/VDX OS가 제공하는 서비스 사이에 유사한 기능들을 찾아야 한다. POSIX 기반의 RTOS인 uC/OS-II의 경우 태스크, 세마포어(semaphore), 메시지큐(message queue) 관리 등 8가지의 기능을 제공하고 있으며, OSEK/VDX OS에서는 태스크, 리소스, 알람, 이벤트, 인터럽트 관리 등 5가지의 기능을 제공하고 있다. 따라서 그림 3처럼 uC/OS-II가 제공하는 서비스들 중 OSEK/VDX OS용으로 변환될 수 있는 기능은 5가지 기능으로 한정된다.

다음으로 고려되어야 할 것은 변환 가능한 기능 중에서 실제 각 기능을 사용하는데 쓰이는 API를 분류하여야 한다. RTOS가 제공하는 대부분의 기능별 API는 크게 Create, Set, Get등과 관련되어 있다. 하지만 API가 사용하는 인자의 종류 또는 개수에 차이가 있으므로 모든 API의 변환은 불가능하다. 더욱이 OSEK/VDX는 POSIX 표준을 따르지 않기 때문에 POSIX API와 형태상으로 일치하지 않는 부분이 많다.

마지막으로 변환과정에 사용되는 API들을 Generic API 형태에 맞게 분류하고, TransPI가 참조하는 XML 형태의 변환 규칙 파일을 만든다. 그림 4는 uC/OS-II 코드를 OSEK/VDX 기반 소스코드로 변환시키기 위해 TransPI가 참조하는 역변환 룰 파일이다.

<PATTERN> 태그는 소스코드를 구문 분석하여 변환 가능

그림 3. OSEK/VDX OS와 uC/OS-II 제공 서비스 간 관계.
Fig. 3. Related object between OSEK/VDX OS and uC/OS-II.

```
<PATTERN name="THREAD_CREATE0">
  <API name="OSTaskCreateExt" ret="0" cast="0" param="9">
    <rule dir="REPLACE">THREAD_CREATE(@VAR[0],@VAR[3],@VAR[6],@VAR[11])</RULE>
  </API>
</PATTERN>
<PATTERN name="EVENT0">
  <API name="OSFlagCreate" ret="0" cast="0" param="4">
    <rule dir="REPLACE">EVENT_CREATE(@VAR[0],@VAR[1],@VAR[2],@VAR[3])</RULE>
  </API>
</PATTERN>
<PATTERN name="THREAD_COND_SIGNAL0">
  <API name="OSFlagPost" ret="0" cast="0" param="4">
    <rule dir="REPLACE">THREAD_COND_SIGNAL(@VAR[0],@VAR[1],@VAR[2],@VAR[3])</RULE>
  </API>
</PATTERN>
```

그림 4. uC/OS-II API를 CIC로 변환하기 위한 XML 규칙 파일.
Fig. 4. XML Rule File for uC/OS-II to CIC.

그림 5. POSIX 기반 소스코드의 OSEK/VDX 기반 코드 변환.
Fig. 5. Transformation from POSIX based source code to OSEK/VDX source code.

한 패턴이 존재하는지 검사한다. 패턴은 API 사용시 특정한 API 들이 순차적으로 호출되는 것을 하나의 태그로 지정한 것으로서 메모리 할당 후 할당한 메모리를 반환하는 등의 예를 들 수 있다. 이후 해당 패턴이 있다면 <API> 태그를 통하여 패턴 내의 각 API가 반환값 및 형변환, 규정된 인자의 수 등 변환에 적절한 형태를 지니고 있는지 검사한다. <rule> 태그는 변환 규칙을 기술한다. REPLACE 인자는 변환 시 기존의 코드를 새로운 코드로 대체하라는 의미이다. 이 때 기존 코드의 인자를 VAR[]인자를 이용하여 수집한다.

그림 5에 POSIX 기반의 소스 코드를 OSEK/VDX 기반 코드로 변환하는 과정을 도식화하여 나타내었다.

OSEK/VDX의 경우는 POSIX 표준을 따르지 않고, OIL 파일과 시스템 생성기를 이용한 코드 생성 과정이 있기 때문에 프로그램 코드의 변환만으로는 완전한 변환이 이루어질 수 없다. 그러므로 POSIX에서 OSEK/VDX OS로의 변환시에는 프로그램 코드의 API 변환뿐만 아니라 OIL 파일 생성까지 포함하여야 한다.

IV. CIC 코드를 이용한 OIL 파일 생성

OSEK/VDX 개발을 위해 사용되고 있는 주요 상용 제품으로는 Freescale사의 OSEKturbo, WindRiver사의 OSEKWorks, LiveDevices사의 RTA 그리고 3Soft사의 proOSEK 등이 있으며 openOSEK, TOPPERS/OSEK 등과 같은 다양한 공개 소프트웨어가 존재한다[15]. OIL 표준 2.5 버전에서는 CPU, OS 등의 14개의 객체에 대하여 다양한 속성을 정의하고 있으나 실제 개발에 사용되는 OSEK/VDX 호환 OS에서는 개발의 편의를 위하여 표준 이외의 추가 속성을 사용하고 있다. 또한 변환 대상인 RTOS의 특성에 의해 정의되는 속성이 존재한다. 따라서 POSIX를 따르는 RTOS의 API를 추상화한 CIC 코드를 통하여 모든 객체의 속성을 자동으로 생성할 수는 없다.

본 논문에서는 자동으로 생성하는 속성의 범위를 OIL 표준에 정의된 속성으로 제한하고, 변환 대상인 RTOS에 의존

적인 속성은 범용으로 사용되는 속성을 따르도록 하여 14개 객체 중 OS, TASK, EVENT 세 개의 객체에 대한 OIL 파일을 자동으로 생성할 수 있게 하였다.

1. OS

OS 객체의 속성은 다음과 같이 설정할 수 있다.

CC= AUTO: CC는 Conformance Class를 의미한다. 태스크 모델과 동일우선순위 허용 여부에 따라 4가지로 구분된다. AUTO로 설정할 경우 태스크의 설정에 따라 결정된다. 타 RTOS의 경우 태스크 모델 및 동일 우선순위 허용 여부가 한 가지로 정의된 경우가 보편적이며, 태스크 생성과정에서 이러한 정보를 간접적으로 확인할 수 있다. 따라서 태스크의 정보를 참조하여 설정할 수 있도록 AUTO로 설정한다.

SCHEDULE= AUTO : 스케줄링 방식을 설정하는 속성이다. OSEK/VDX OS는 선점형, 비 선점형 방식과 병행방식을 모두 지원한다. 선점형과 비 선점형 방식은 태스크의 SCHEDULE 속성을 모두 FULL 또는 NON으로 설정하여야 하며, 병행방식의 경우 각 태스크의 SCHEDULE 속성을 각각 지정해야 한다. 따라서 태스크의 SCHEDULE 속성에 의해 설정되도록 AUTO로 설정한다.

STAUTS= EXTENDED: OSEK/VDX OS는 BASIC과 EXTENDED 상태 모델을 지원한다. EXTENDED 상태 모델은 BASIC 상태 모델에 WAIT 상태를 추가 지원하며 BASIC 상태 모델의 프로그램을 완벽히 호환한다. 따라서 호환성을 고려하여 EXTENDED로 설정한다.

STARTUPHOOK= TRUE: OSEK/VDX OS는 StartupHook, ErrorHook, ShutdownHook, PreTaskHook, PostTaskHook의 5가지 Hook함수를 지원한다. 또한 OSEK/VDX OS는 별도의 main함수를 사용하지 않으며, 초기화 과정은 StartupHook에서 이루어진다. 따라서 기존 POSIX 기반 RTOS의 초기화 코드를 사용하기 위하여 StartupHook을 활성화 시킨다. 기타 Hook 함수는 비 활성화 시켜 코드 크기를 경량화 시킨다.

2. TASK

TASK 객체의 속성은 다음과 같이 설정할 수 있다.

SCHEDULE= FULL: OS 객체의 SCHEDULE 속성을 AUTO로 설정하였기 때문에 OSEK/VDX OS의 스케줄링 정책은 태스크 객체의 SCHEDULE 속성에 의하여 정의된다. 하지만 각각의 태스크가 어떠한 스케줄링 정책을 따를 것인지를 판단할 수 있는 기준은 없다. 따라서 이 속성은 적합한 값으로 자동 설정이 불가능 하여 선점형을 기준으로 설정되게 하였다. 스케줄링 정책에 대한 기준을 TASK 객체에서 모두 설정하게 함으로서 다른 스케줄링 정책을 사용하는 프로그램을 이식하더라도 OS 객체에는 신경 쓰지 않고 TASK 객체만을 수정하면 되도록 하였다.

ACTIVATION= 1: 이 속성은 해당 태스크가 설정 값 만큼 수행되면 SUSPENDED 상태로 진입하게 한다. 이 속성을 사용하는 이유는 BASIC 상태 모델이 WAIT 상태를 지원하지 않기 때문에 가상으로 WAIT 상태를 지원하기 위한 것이다. 본 논문에서는 상태모델을 항상 EXTENDED 상태로 유지하기 때문에 기본값으로 1을 설정한다.

PRIORITY: POSIX를 따르는 RTOS에서는 태스크를 생성하는 과정에서 우선순위를 설정한다. 따라서 이 속성값은 태스크 설정과 관계된 API를 구문 분석하여 자동으로 설정할

그림 6. OIL 파일 생성을 포함한 POSIX 기반 소스코드의 OSEK/VDX 기반 코드 변환 과정.

Fig. 6. Transformation process from POSIX based source code to OSEK/VDX source code with OIL file generation.

수 있게 하였다. OS 객체의 CC 속성을 AUTO로 설정하였으므로 이 속성에 의하여 CC 속성이 설정되게 된다.

AUTOSTART= TRUE: OSEK/VDX OS에서 태스크를 READY 상태로 설정하기 위해서는 다른 태스크에서 ActivateTask() API를 호출하거나 알람 서비스를 이용하여야 한다. 따라서 OS 시작 전에 하나 이상의 태스크가 READY 상태에 있지 않으면 어떠한 태스크도 동작할 수 없다. POSIX 기반의 RTOS는 태스크 생성 시점에서 태스크가 준비 상태가 된다는 점을 고려하여 모든 태스크에 대하여 이 속성 값을 참으로 설정한다. 이 속성 값이 참일 경우 APPMODE라는 추가속성을 정의하여야 한다. 이 속성은 POSIX 기반 RTOS에서는 사용되지 않기 때문에 모든 AUTOSTART 태스크에 대하여 OSDEFAULTAPPMODE값을 정의하여 사용하도록 하였다.

3. EVENT

EVENT 객체는 MASK라는 단 하나의 속성만 가지며 이는 객체에 대한 식별자이다. AUTO로 설정하여 식별자를 자동 할당하도록 하였다.

4. TransPI를 이용한 OIL 파일 생성

TransPI는 2장에서 소개하였듯이 초기화, 관련 정보 수집, 코드 변환, 출력 파일 생성의 4단계를 거쳐 API 변환을 수행한다. OIL 파일을 자동으로 생성하는 과정 또한 이와 동일하나 OIL 파일 생성시 별도의 코드 변환은 필요하지 않기 때문에 아래와 같은 3단계로 구성된다.

1. 초기화: 초기화 과정에서는 TransPI가 기존에 사용하는 초기화 과정 이외에 OIL 파일 생성에 필요한 자료 구조 정의 및 메모리 할당 등의 초기화 루틴을 추가 수행한다.

2. 관련 정보 수집: 기존의 TransPI가 수행하는 관련 정보 수집 과정에서 획득한 정보를 OIL 파일 생성을 위한 구조체에 전달하는 루틴을 수행한다.

3. 출력 파일 생성: 수집된 정보를 바탕으로 OIL 파일을 생성한다.

그림 6은 OIL 생성 과정을 포함한 POSIX 기반 소스 코드의 OSEK/VDX 기반 소스 코드로 변환과정을 나타낸다.

V. 사례 연구: uC/OS-II 기반 코드의 OSEK/VDX 기반 코드 변환

본 장에서는 제안된 변환 방법을 이용하여 예제 소스 코드를 변환하고 XC167CI 마이크로 컨트롤러에 이식하여 변환된 소스 코드의 실행 여부를 확인하였다. 본 연구에서 사용된

그림 7. 이벤트 기반 동기화 예제의 동작.

Fig. 7. Operation of synchronization example using event.

```

void Increase_go(void* data)
{
    char buff[20] = "Increase_go : ";
    while (1)
    {
        int i;
        INT8U Error;

        OSFlagPend(my_flag_grp, ENGINE_START,
                   OS_FLAG_WAIT_SET_ANY, 1000, &Error);
        for (i = 0; i < 4; i++)
        {
            SendSerial(buff, number++);
            delay();
        }
        OSFlagPost(my_flag_grp, ENGINE_START, OS_FLAG_CLR, &Error);
    }
}

```

그림 8. uC/OS-II 기반의 Increase_go 소스 코드.

Fig. 8. Increase_go Source Code based on uC/OS-II.

POSIX 기반 RTOS는 uC/OS-II이며, 이를 TransPI를 이용하여 proOSEK으로 변환하였다. 예제 소스 코드는 이벤트를 이용한 동기화이다. 동기화는 차량 ECU의 제어루프를 구성하는 센싱, 제어 알고리즘, 액추에이터 제어와 같은 일련의 작업의 기초가 되기 때문에 이벤트를 이용한 동기화 예제 소스코드에 대한 변환을 수행하였다. 그림 7은 각 태스크 간 동작을 나타낸다.

Increase_go 태스크와 Decrease_go 태스크는 이벤트를 이용하여 동기를 맞추고, 공유하는 숫자를 각각 증가시키거나 감소시키는 동작을 반복적으로 수행한다. OSEK/VDX OS 기반으로 생성된 소스 코드의 정상 동작 여부를 확인하기 위하여 증감된 숫자를 PC로 전송하도록 구성하였다. 그림 8에 uC/OS-II의 Increase_go 태스크의 소스코드를 나타내었다.

uC/OS-II 기반으로 작성한 예제코드를 제안된 변환 방법을 이용하여 변환하였다. TransPI를 통해 생성된 OSEK/VDX OS 기반의 소스 코드와 OIL 파일이 그림 9, 10과 같이 생성됨을 확인할 수 있다.

생성된 OIL 파일은 표준에 정의된 속성만을 포함하고 있기 때문에 실험에 사용한 proOSEK에서 정의하는 속성을 추가하여야 한다. 최종 생성된 OIL 파일의 일부를 그림 11에 나타내었다.

추가된 속성은 proOSEK에서 개발의 편의를 위하여 제공하는 속성과 마이크로 컨트롤러에 대한 속성으로 구분된다.

상용 제품은 차량용 마이크로 컨트롤러로 주로 사용되는 일부 제품에 대한 드라이버를 제공하며, 이를 통해 별도의 이식과정 없이 OSEK/VDX OS를 사용할 수 있다. proOSEK은 연구에 사용한 XC167CI 마이크로 컨트롤러에 대한 드라이버를 지원하며 OS 객체에서 이를 제어하기 위한 다양한 속성을 정의한다. 이러한 속성은 하드웨어에 의존적인 코드를 제

```

TASK(Increase_go)
{
    char buff[20] = "Increase_go : ";
    while(1)
    {
        int i;

        WaitEvent(EVENT_1);
        for(i = 0; i < 4; i++)
        {
            SendSerial(buff, number++);
            delay();
        }
        ClearEvent(EVENT_1);
    }
    TerminateTask();
}

```

그림 9. OSEK/VDX OS 기반의 Increase_go 소스 코드.

Fig. 9. Increase_go source code based on OSEK/VDX OS.

```

CPU OSEK_DEFAULT_CPU
{
    OS OSEK_DEFAULT_OS
    {
        CC = AUTO;
        SCHEDULE = AUTO;
        STATUS = EXTENDED;
        STARTUPHOOK = TRUE;
        ERRORHOOK = FALSE;
        SHUTDOWNHOOK = FALSE;
        PRETASKHOOK = FALSE;
        POSTTASKHOOK = FALSE;
    }
};

TASK increase_go
{
    TYPE = EXTENDED;
    AUTOSTART = TRUE;
    {
        APPMODE = OSDEFAULTAPPMODE;
    }
    SCHEDULE = FULL;
    PRIORITY = 10;
    EVENT = EVENT_1;
    ACTIVATION = 1;
};

```

그림 10. TransPI를 통해 생성된 OIL 파일.

Fig. 10. Created OIL file using TransPI.

```

#include<XC16X_OS.oil>
CPU OSEK_DEFAULT_CPU
{
    OS OSEK_DEFAULT_OS
    {
        CC = AUTO;
        SCHEDULE = AUTO;
        STATUS = EXTENDED;
        STARTUPHOOK = TRUE;
        ERRORHOOK = FALSE;
        SHUTDOWNHOOK = FALSE;
        PRETASKHOOK = FALSE;
        POSTTASKHOOK = FALSE;

        MICROCONTROLLER = XC164CS;
        XC16X_MEMORYMODEL = LARGE;
        XC16X_REALTIMECLOCK = NOT_USED;
        XC16X_VECTORTABLE = SIZE_4;
        XC16X_MAC_SUPPORT = DONTKNOW;
        STACKCHECK = TRUE;
        EXTRA_RUNTIME_CHECKS = TRUE;
        USERMAIN = FALSE;
        PREISRHOOK = FALSE;
        POSTISRHOOK = FALSE;
        USEGETSERVICEID = FALSE;
        USEPARAMETERACCESS = FALSE;
        SERVICETRACE = FALSE;
        USELASTERROR = FALSE;
        USERESSCHEDULER = FALSE;
    }
};

TASK increase_go
{
    TYPE = EXTENDED;
    AUTOSTART = TRUE;
    {
        APPMODE = OSDEFAULTAPPMODE;
    }
    SCHEDULE = FULL;
    PRIORITY = 10;
    EVENT = EVENT_1;
    ACTIVATION = 1;

    CALLSCHEDULER = DONTKNOW;
    XC16X_SYSTACKSIZE = 256;
};

TASK decrease_go
{
    TYPE = EXTENDED;
    AUTOSTART = TRUE;
    {
        APPMODE = OSDEFAULTAPPMODE;
    }
    SCHEDULE = FULL;
    PRIORITY = 11;
    EVENT = EVENT_1;
    ACTIVATION = 1;

    CALLSCHEDULER = DONTKNOW;
    XC16X_SYSTACKSIZE = 256;
};

```

그림 11. proOSEK 정의 속성을 추가한 OIL 파일.

Fig. 11. Attribute added OIL file for proOSEK.

어하기 위해 사용되므로 각 마이크로 컨트롤러에 적합하게 자동 생성할 수 없으므로 프로그래머가 직접 기술해야 한다.

실험 결과 그림 12와 같이 두 태스크가 동기화를 이루어 동작함을 확인할 수 있었다.

그림 12. 태스크 동기화 예제의 스크린 샷.

Fig. 12. Screen shot of synchronization example of task.

VI. 결론

본 논문에서는 POSIX 기반 RTOS의 소스코드를 OSEK/VDX OS기반으로 변환시 고려사항을 알아 봄으로서 기존의 프로그램을 OSEK/VDX OS 환경에 맞게 구현해야 할 경우 손쉽게 기존의 프로그램 코드를 재 사용할 수 있게 하도록 하였다.

변환에 사용한 TransPI는 POSIX 표준을 근거로 정의 한 Generic API를 기반으로 RTOS간의 API 변환을 수행한다. 따라서 POSIX 표준을 따르지 않는 RTOS로의 변환을 수행시에는 변환에 어려움이 따른다. 본 논문에서는 비(非) POSIX 기반의 OS인 OSEK/VDX OS로의 변환을 위하여 OSEK/VDX OS로의 변환을 위한 XML 규칙 파일을 작성하고, 이를 이용하여 API 변환을 수행하였다. 또한 OIL 표준에 의거하여 OIL 파일을 자동으로 생성하는 방법을 제안하고, 이를 기반으로 POSIX 기반 OS인 uC/OS-II에서 작성된 동기화 예제를 OSEK/VDX 코드로 변경하여 XC167CI 보드에 변환된 코드를 이식하였다. 이러한 동기화는 차량에서 연료분사제어 및 점화제어 등에 사용될 수 있다.

하지만 실제 개발에 사용되는 OSEK/VDX 호환 OS에서는 개발의 편의를 위하여 표준 이외의 다양한 추가 속성을 사용하고 있기 때문에 이러한 추가 속성을 모두 고려하기는 어렵다. 그러므로 제안한 변환 방법을 사용하여 POSIX 기반의 RTOS를 OSEK/VDX OS 기반으로 변환하기 위해서는 개발환경에 따른 OIL 파일의 수정이 필요하다. 또한 향후 소스코드의 변환 범위를 확장하기 위한 추가 연구가 필요하다.

참고문헌

- [1] 이귀영, 오희근, “엔진 ECU 개발과 타 제어장치와의 통합,” 대한기계학회지, 제32권 제10호, pp. 830-836, 1992.
- [2] http://en.wikipedia.org/wiki/Electronic_control_unit
- [3] 이수영, “자동차와 IT 융합기술의 전망,” 오토저널, 30권 제4호, pp. 88-94, 2008.
- [4] J. Dirk, “OSEK/VDX history and structure,” *OSEK/VDX Open System in Automotive Networks(Ref. No.1998/523)*, IEE Seminar, pp. 2/1-2/4, Nov. 1998.
- [5] OSEK/VDX Operating System, V2.2.3, [http:// portal.osek-idx.org](http://portal.osek-idx.org), Feb. 2005.
- [6] Frost & Sullivan, “Strategic Analysis of the European Market for Software in Passenger Cars,” M03B-26, 2007(<http://www.frost.com>)
- [7] 박병률, 맹지찬, 이종범, 유민수, 안현식, 정구민, “임베디

드 S/W 개발을 위한 RTOS API 변환기의 설계 및 구현,” 정보 및 제어 학술대회 논문집, pp. 443-445, Oct. 2006.

- [8] 박병률, 맹지찬, 이종범, 유민수, 안현식, 정구민, “MDA에 기반한 실시간 운영체제 API 정변환/역변환기의 개발,” 전기학회 논문지 제56권 제12호, pp. 2247-2250, 2007.
- [9] 박병률, “모델기반 접근 방법을 이용한 이중 RTOS간 API 변환기의 설계에 관한 연구,” 국민대학교 전자공학과, 석사학위 논문.
- [10] A. Zahir, “OIL-OSEK implementation language,” *OSEK/VDX Open System in Automotive Networks(Ref. No. 1998/523)*, IEE Seminar, pp. 8/1-8/3, Nov. 1998.
- [11] OSEK/VDX System Generation OIL, V2.5, <http://portal.osek-idx.org>, 2004.
- [12] 이종덕, 허명선, 박병률, 유민수, 안현식, 정구민, “POSIX 기반 태스크 동기화 코드의 OSEK/VDX 기반 코드 변환에 대한 연구,” 한국자동차공학회 2008년 학술대회 자료집, p. 513, Nov. 2008.
- [13] HOPES, <http://www.peace.snu.ac.kr/hopes/>, 2005.
- [14] P. Koopman, J. Devalle, “The exception handling effectiveness of POSIX operating systems,” *IEEE Transactions on Software Engineering*, vol. 26, no. 9, pp. 837-848, 2000.
- [15] 박미룡, 이성훈, 박경민, 한태만, “자동차 기술과 IT기술 융합 SW 표준화 동향,” 한국통신학회지 제25권 제6호, pp. 3-11, 2008.

송 영 호

2009년 국민대학교 전자공학부 졸업.
2009년~현재 국민대학교 대학원 석사과정 재학중. 관심분야는 임베디드 시스템, 실시간 운영체제.

이 태 앙

2009년 선문대학교 전자공학과 졸업.
2009년~현재 국민대학교 대학원 석사과정 재학중. 관심분야는 임베디드 시스템, 차량 전자제어.

이 종 덕

2007년 국민대학교 전자공학부 졸업.
2009년 동 대학원(공학석사). 2009년~현재 LIG 넥스원 연구원. 관심분야는 임베디드 시스템.

문 찬 우

1989년 서울대학교 제어계측공학과(공학사). 1991년 서울대학교 제어계측공학과(공학석사). 2001년 서울대학교 전기컴퓨터공학부(공학박사). 2006년~현재 국민대학교 전자공학과 조교수. 관심분야는 모바일 로봇, 정밀제어, 모터 제어.

정 구 민

1995년 서울대학교 제어계측공학과(공학사). 1997년 서울대학교 제어계측공학과(공학석사). 2001년 서울대학교 전기컴퓨터공학부(공학박사). 2005년~현재 국민대학교 부교수. 관심분야는 임베디드 시스템, 차량 전자제어.

안 현 식

1982년 서울대학교 제어계측공학과(공학사). 1984년 서울대학교 제어계측공학과(공학석사). 1992년 동 대학원(공학박사). 1993년~현재 국민대학교 전자공학과 교수. 관심분야는 임베디드 시스템, 차량 전자제어.