SoC(System-on-Chip) 설계와 검증을 지원하는 실시간운영체제*

김지민, 유민수
한양대학교 정보통신공학과
e-mail:{jmkim,msryu}@rtcc.hanyang.ac.kr

A Real-Time Operating System for System-on-Chip Design and Verification

Jimin Kim, Minsoo Ryu
Graduate School of Information and Communications,
Hanyang University

요약

최근 SoC를 포함하는 대부분의 임베디드시스템에서는 RTOS가 선택이 아닌 필수적인 구성요소가 됨에 따라 SoC 개발의 초기단계에서부터 RTOS를 도입하는 것이 바람직하다. 하지만, 기존의 범용 RTOS가 제공하는 기능은 대부분 응용 소프트웨어의 개발과 수행을 위한 것으로 SoC 개발 및 검증에는 적합하지 않은 문제점을 가지고 있다. 본 연구에서는 SoC 개발을 위해 운영체제가 만족시켜야할 요구사항을 제시하고, 소프트웨어의 재사용성과 SoC의 검증을 함께 지원하는 VPOS(Verification-Purpose OS)를 개발하였다. VPOS는 초경량의 단순한 계층적 구조(layered structure)를 가지는 RTOS로서 소프트웨어 재사용을 위해 POSIX 표준 API, 유닉스 호환 디바이스 드라이버 인터페이스, HAL 등을 제공한다. 또한 SoC 설계의 검증을 위해 RT 수준의 통합시뮬레이션에 적합한 커널 구조 및 최적화된 코드, 하드웨어 테스트를 위한 쉘 명령어, 응용 소프트웨어의 디버깅을 위한 KREM(kernel resource and event monitoring) 등의 특징을 함께 제공한다.

1. 서론

최근 SoC를 포함하는 대부분의 임베디드시스템에서는 RTOS가 선택이 아닌 필수적인 구성요소가 되었다. RTOS를 사용하면 임베디드시스템에 요구되는 실시간성을 보장하는 것은 물론 멀티태스킹, 메모리 관리 및 주변장치의 효과적인 제어가 가능해지기 때문이다. 특히 RTOS가 제공하는 하드웨어 추상화 기능을 통해 H/W와 S/W의 개발에 요구되는 비용과 시간을 크게 단축시킬 수 있다. 따라서 SoC 개발의 초기단계에서부터 RTOS를 도입하는 것이 바람직하며, RTOS를 이용하면 SoC 설계의 복잡성을 줄이는 동시에 H/W와 S/W를 병렬적으로 개발하는 것이 가능해진다.

하지만 SoC 개발을 위해 기존의 범용 RTOS를 도입하는 것은 많은 문제점을 제기한다. 범용 RTOS는 복잡한 커널 구조와 인터페이스를 제공하기 때문에 전문적인 지식이 부족한 SoC 개발자가 RTOS를 다루는 것이 쉽지 않다. 또한 HW/SW 통합시뮬레이션이나 IP(Intellectual Property:설계자산) 및 응용 S/W의 검증 등, SoC 개발에 요구되는 기능을 기대할 수 없어 RTOS의 도입이 실효를 거두지 못하는 경우가 많다.

본 연구에서는 위와 같은 문제를 해결하기 위해 SoC 설계와 검증에 적합한 VPOS(Verification-Purpose OS)를 개발하였다. VPOS는 통상적인 RTOS에서 제공되는 우선순위 기반의 스케줄링 알고리즘과 실시간동기화 기법 등을 지원함은 물론, RT(Register Transfer) 수준의 통합시뮬레이션(cosimulation)을 지원하고 IP의 상태를 확인하고 테스트할 수 있는 텍스트 기반의 쉘 등, SoC 검증에 필요한 기능을 함께 제공한다. 또한 VPOS는 응용 소프트웨어, 디바이스 드라이버, 커널의 재사용성을

* 본 논문의 연구는 서울대학교 SoC 설계기술사업단과 산업자원부의 산업기술기반조성사업의 지원을 받았음 (03나05)

고려하여 설계되어 개발초기에 사용된 소프트웨어를 개발 최종단계에서 사용할 수 있도록 이식(porting)하는 것이 용이하다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 SoC 개발에 사용되는 운영체제에 요구되는 특징과 VPOS의 검증 기능에 대해 기술하고, 3장에서는 VPOS의 설계와 구현에 대한 내용을 설명한다. 4장에서는 본 논문의 연구결과를 요약하고 향후 연구 과제를 제시한다.

2. SoC 설계 및 검증을 고려한 운영체제의 요구사항

이 절에서는 SoC 개발에 있어 기존 범용 RTOS의 사용에 따른 문제점을 제시하고, SoC 개발에 사용되는 운영체제가 갖추어야할 요구사항을 도출한다.

2.1 소프트웨어 재사용

최근 임베디드 시스템의 복잡도가 비약적으로 증가하면서 SoC 개발에 있어서 소프트웨어의 개발, 이식(porting), 테스트가 대단히 중요해졌다. 통상적으로 SoC의 개발에 있어서 소프트웨어의 비중이 하드웨어에 비해 2~3배에 달하는 것으로 알려져 있다. 따라서 SoC에 사용되는 소프트웨어의 개발에 요구되는 비용과 시간을 줄이기 위하여 소프트웨어의 재사용성을 극대화하는 것이 요구된다.

SoC 개발에 있어서 재사용될 수 있는 소프트웨어로는 최종 시스템에 사용될 응용 S/W, 운영체제를 포함하는 시스템 S/W 그리고 설계 단계에서 하드웨어 검증에 사용되는 테스트 S/W 등이 있다.

응용 S/W는 보통 하드웨어에 비의존적인 코드로서 운영체제가 제공하는 API를 사용함에 따라 운영체제가 표준적인 API를 제공하는 것이 필요하다. 1990년대 초반에 IEEE에서는 유닉스 계열 운영체제에서 제공되는 API의 표준으로서 POSIX(Portable Operating System Interface for UNIX)를 정의한 바 있으며, 최근에는 The Open Group에서 POSIX 표준을 관리하고 있다 [6]. 현재 POSIX 최신 표준은 1003.1-2004로서 리눅스를 포함하는 대부분의 유닉스 계열 운영체제는 물론 일부 범용 RTOS에서 POSIX 표준 API를 제공하고 있다. 따라서 응용 S/W의 재사용성을 극대화하기 위해서는 SoC 개발에 사용되는 운영체제에서 POSIX 표준 API를 제공하는 것이 바람직하다.

시스템 S/W 및 테스트 S/W는 주로 하드웨어 의존적인 코드를 포함하는데, 이들을 재사용하기 위해서는 운영체제를 다양한 프로세서 및 IP를 포함하는 시스템으로 이식하는 것이 용이하여야 한다. 운영체제에서 하드웨어 의존적인 코드로는 CPU 아키텍처, 메모리, 보조프로세서, 캐시 관련 부분 및 각종 I/O 주변장치를 제어하는 디바이스 드라이버 등이 있다. 따라서 운영체제가 프로세서를 포함하는 하드웨어구조에 의존적인 부분을 분리하여 관리하고, 설계단계에서 작성된 디바이스 드라이버를 최종시스템에서 재사용할 수 있도록 호환성이 높은 디바이스 드라이버 인터페이스를 제공하는 것이 필요하다.

2.2 SoC 설계의 검증

소프트웨어 입장에서는 하드웨어 내부적인 특징을 파악할 수 없기 때문에 운영체제를 이용하여 하드웨어 설계의 오류를 검증하는 것은 제한적일 수밖에 없다. 하지만 HW/SW 통합시뮬레이션(HW/SW cosimulation), 단순한 형태의 하드웨어 IP의 테스트, 그리고 응용 소프트웨어 디버깅 등의 경우에 운영체제가 중요한 기능을 제공할 수 있다.

일반적으로 SoC 설계검증은 알고리즘 수준, TL(Transaction Level), RT(Register Transfer) 수준 등 다양한 추상수준(abstraction level)에서 이루어진다. 알고리즘 수준의 수행속도에 비해 TL의 수행속도는 대략 1/10, RT 수준의 수행속도는 대략 1/10000이다. 추상수준이 높을수록 수행속도는 빨라지지는 반면 검증의 정확도는 낮아진다. 따라서 정확한 검증을 위해서는 추상수준이 낮은 RT 수준의 검증이 중요하며, 특히 H/W와 S/W를 함께 RT 수준에서 통합시뮬레이션하는 것이 중요한 검증기법으로 널리 활용되고 있다. 하지만, 커널이 크고 내부구조가 복잡한 기존의 범용 OS를 사용하면 통합시뮬레이션의 속도가 크게 저하되어 현실적으로 RT 수준의 검증이 어렵게 된다. 따라서 통합시뮬레이션에 요구되는 시간을 최소화하기 위해서는 SoC 검증에 사용되는 운영체제 커널의 크기와 실행 오버헤드가 최적화되어야 한다.

또한 운영체제는 개별적인 IP의 테스트와 응용 소프트웨어의 디버깅을 효과적으로 지원할 수 있어야 한다. 운영체제는 하드웨어와 응용 소프트웨어의 인터페이스 역할을 함에 따라 각각에 대해 특화된 검증기능을 지원하는 것이 가능하다. 예를 들면 운영체제의 수준에서 특정 IP의 입출력 결과를 테스트하여 하드웨어 검증을 도와주거나 응용 소프트웨어의 디버깅에 필요한 다양한 정보를 제공하여 소프트웨어 검증을 수월하게 하도록 할 수 있다. 현재까지

알려진 대부분의 RTOS에서는 이러한 측면에서 요구되는 검증기능을 제공하지 않으며, 따라서 SoC 설계 및 검증에 사용되는 운영체제에서는 위에서 언급된 기능을 제공하는 것이 바람직하다.

3 VPOS의 설계 및 구현

본 절에서는 2절에서 기술된 요구사항을 충족시킬 수 있도록 개발된 VPOS의 설계 및 구현에 대하여 설명한다. VPOS는 ARM920T를 탑재한 S3C2410 보드에서 동작하도록 개발되었으며, 현재 다양한 기능이 추가로 확장되고 있다 [9].

3.1 VPOS의 개발목표 및 설계

VPOS의 가장 중요한 목표는 소프트웨어의 재사용을 가능하게 하고 SoC 검증에 필요한 요구를 충족시키는 것이다. 이러한 목표를 위해 VPOS는 초경량의 단순한 계층적 구조(layered structure)를 가지도록 설계되었다.

그림 1은 VPOS의 계층적 구조를 보여준다. VPOS의 최상층에는 시스템 콜 계층(system call layer)이 존재하여 커널과 응용 소프트웨어간의 인터페이스 역할을 하며, 최하층에는 하드웨어 추상화 계층(hardware abstraction layer)이 존재하여 운영체제를 쉽게 이식할 수 있도록 해준다.

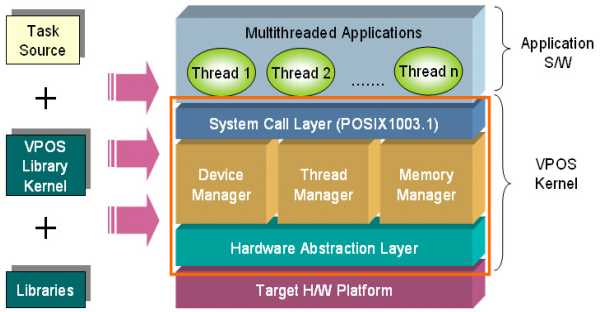

그림 1. VPOS 커널 구조

VPOS는 커널의 내부구조가 단순할 뿐만 아니라 커널의 크기가 50Kbytes 이내로서 RT 수준의 HW/SW 통합시뮬레이션에 적합하다. 또한 RTOS에 대한 전문지식이 부족한 SoC 개발자가 커널을 이해하고 필요에 따라 커널의 기능을 수정하거나 확장하는 것이 쉽게 되어 있다. 이를 위해 VPOS 내부적으로 사용되는 함수 상호간의 의존성을 최소화하고 함수 호출 트리(function call tree)의 깊이(depth)를 최소화하였다.

3.2 소프트웨어의 재사용

VPOS는 응용 소프트웨어, 디바이스 드라이버, 그리고 커널의 재사용을 고려하여 다음과 같은 특징을 제공한다.

표준 API의 제공: VPOS는 The Open Group에서 관리하는 POSIX 1003.1-2004 표준과 ANSI C 표준 라이브러리 형식을 따르는 API의 제공을 목표로한다. 현재는 pthread_create, pthread_join, pthrea_exit 등의 쓰레드 관련 함수, sem_wait, sem_post 등의 동기화(synchronization) 관련 함수, 문자열 처리, 동적 메모리 관리 등을 위한 API를 제공한다.

디바이스 드라이버 재사용: VPOS는 리눅스를 포함하는 유닉스 계열 운영체제와 유사한 디바이스 드라이버 등록구조 및 인터페이스를 제공한다 [3]. 최종 시스템에서 유닉스 계열 운영체제를 사용할 경우 별도의 작업을 수행하지 않고서도 SoC 개발 초기에 작성된 디바이스 드라이버를 재사용할 수 있다.

Device Driver Operation	Description
open	장치 초기화
close	장치 해제
read	장치로부터 데이터 입력
write	장치로 데이터 출력
ioctl	입출력 제어

표 1. VPOS·의 디바이스 드라이버 인터페이스

VPOS는 파일시스템 인터페이스를 이용한 디바이스 드라이버를 정의하는데, 표 1은 응용 소프트웨어를 위한 인터페이스를 보여주며 표 2는 디바이스 의존적인 드라이버 함수의 인터페이스를 보여준다. 디바이스 드라이버 개발자는 표 2에서 정의되는 vk_open, vk_release, vk_read, vk_write, vk_ioctl, vk_interrupt 함수포인터에 해당하는 각각의 드라이버 함수를 구현하여 등록해야 한다.

```
struct file_operations{
    int (*vk_open)(char *file_name);
    int (*vk_release)(int fd);
    int (*vk_read)(int fd, char *buf, size_t count);
    int (*vk_write)(int fd, char *buf, size_t count);
    int (*vk_ioctl)(int fd, unsigned int cmd, unsigned long arg);
    int (*vk_interrupt)(void);
} vk_file_operations_t;
```

표 2. VPOS의 디바이스 드라이버 등록 구조

HAL(Hardware Abstraction Layer)의 도입: 모든 하드웨어 의존적인 부분을 HAL로 정의하여 커널의

이식성을 향상시켰다. VPOS에서는 SoC의 특성을 고려하여 다음과 같이 세 가지 종류의 HAL을 정의한다 [1].

- CPU Core HAL API: CPU에 의존적인 코드 (context switching, CPU startup 코드 등)
- Variant HAL API: CPU와 I/O 디바이스를 제외한 장치 의존적인 코드 (MMU, FPU, DMA, Interrupt Controller 등에 의존적인 코드)
- I/O HAL API: I/O 장치에 의존적인 코드 (UART, Timer, GPIO 등에 의존적인 코드 등)

HAL의 도입은 VPOS 커널의 이식성을 높이는 것 뿐만 아니라 SoC 개발의 유연성과 생산성에도 도움을 준다. 전통적인 시스템 개발 과정에서는 하드웨어가 먼저 개발되고 이 후에 소프트웨어가 개발된다. 하지만 HAL을 도입하면 하드웨어와 소프트웨어를 병렬적으로 개발하는 것이 가능해진다 [7].

3.3 SoC를 위한 VPOS의 검증기능

VPOS는 하드웨어와 소프트웨어가 통합된 SoC를 효과과적으로 검증하기 위해 검증용 쉘 명령어와, 시스템의 자원사용과 커널 내부에서 발생하는 이벤트를 모니터링 할 수 있는 KREM(Kernel Resource and Event Monitor)을 제공한다.

쉘: Memory mapped I/O 방식에서 I/O 장치는 메모리 주소로 접근 할 수 있다. 즉, 해당 주소에 데이터를 읽고 쓰는 것으로 I/O를 제어할 수 있으며 데이터를 읽음으로써 I/O의 상태를 알 수 있다. VPOS는 memory mapped I/O 방식을 사용하는 하드웨어를 검증할 수 있도록 특정 메모리 주소를 read/write할 수 있는 쉘 명령어를 제공한다.

KREM: VPOS는 시스템 자원(resource)과 내부 이벤트를 모니터링 할 수 있는 KREM(Kernel Resource and Event Monitor) 기능을 제공한다. KREM은 응용 소프트웨어가 사용하는 CPU 시간, 세마포어, 메모리 등의 커널 리소스에 대한 정보와 인터럽트 및 컨텍스트 스위칭 등의 커널 이벤트에 대한 정보를 제공한다. KREM을 사용하면 실행시(run-time)에 별도에 장비를 사용하지 않고 응용 소프트웨어의 디버깅을 효과적으로 할 수 있다.

3.4 실시간성 보장 기법 지원

VPOS는 실시간성이 요구되는 응용 소프트웨어를 위하여 32단계의 정적우선순위(static priority)를 지원하는 CPU 스케줄러를 제공한다 [8]. 또한 우선순위계승 프로토콜(Priority Inheritance Protocol)을 사용하는 세마포어 및 뮤텍스를 제공한다 [2, 4, 5].

4. 결론 및 향후 연구 과제

본 연구에서는 SoC 개발을 위한 운영체제의 요구사항을 도출하고, 소프트웨어의 재사용과 SoC 설계 검증을 효과적으로 지원할 수 있는 VPOS(Verification-Purpose OS)를 개발하였다. VPOS는 소프트웨어 재사용을 위해 POSIX 표준 API, 유닉스 호환 디바이스 드라이버 인터페이스, HAL 등을 제공한다. 또한 SoC 설계 검증을 위해 RT 수준의 통합시뮬레이션에 적합한 커널 구조 및 최적화된 코드, 하드웨어 테스트를 위한 쉘 명령어, 응용 소프트웨어 디버깅을 위한 KREM 등의 기능을 함께 제공한다.

향후에는 현재 VPOS에 포함되지 않은 파일시스템 및 네트워크 프로토콜 스택 등을 추가로 확장할 계획이며, 아울러 다양한 추상수준(abstraction level)에서 수행되는 검증을 고려한 기능 등을 지원하고자 한다.

참고문헌

[1] Redhat, http://sources.redhat.com/ecos/

[2] Hideyuki Tokuda, Tatsuo Nakajima and Prithvi Rao, “Real-Time Mach: Towards a Predictable Real-Time System”, *USENIX Mach Workshop*, pages 73-82, October 1990.

[3] Daniel P. Bovet and Marco Cesati “Understanding the LINUX KERNEL” 2nd Ed. O'REILLY

[4] Mark Heuser, “An implementation of real-time thread synchronization”, in *Proceedings of Usenix Summer Conference*, June, 1990.

[5] L. Sha, R. Rajkumar, and J. P. Lehoczky, “Priority Inheritance Protocols: An Approach to Real-Time Synchronization”, in *IEEE Transactions on Software Engineering*, Volume 39, Pages 1175 - 1185, September, 1990.

[6] POSIX, http://www.opengroup.org/

[7] Sungjoo Yoo and Ahmed A.Jerraya “Introduction to Hardware Abstraction Layers for SoC Design”, Automation and Test in *Europe Conference and Exhibition*, pp.10336, March 2003.

[8] John A. Stankovic, “Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems”, *IEEE Computer*, 21(10):10-19, October 1988.

[9] SAMSUNG Electronics, “USER'S MANUAL S3C4210X”, February, 2003.