특집 02

임베디드 소프트웨어 테스트 기술 동향 및 계층 통합 유형에 따른 테스트 방안

성아영 (삼성전자)

목 차 »

1. 서 론

오늘날 소프트웨어가 내장되지 않은 제품이 거의 없을 만큼 '스마트 폰, 스마트 TV, 의료, 통신, 항공, 자동차' 와 같은 광범위한 분야로 임베디드 소프트웨어(embedded software)가 사용되고 있으며, 임베디드 제품의 소형화 및 고성능화가 진행되면서 제품 경쟁력의 핵심이 하드웨어에서 소프트웨어에 의해 좌우되는 기술집약적 고부가가치 산업으로 발전하고 있다. 특히 소프트웨어의 비중이 높아지면서 임베디드 제품 결함의 80%이상이 하드웨어가 아닌 임베디드 소프트웨어로부터 야기되기 때문에[1], 임베디드 소프트웨어의 결함이 제품 전체 품질에 매우 큰 영향을 미친다. 최근 급격히 증가하는 시스템의 기능 요구 사항과 복잡도를 지원하기 위해, 임베디드 소프트웨어는 운영체제, 미들웨어(middleware)를 포함하여 각종 서비스 개발을 위한 라이브러리(library) 함수와 같은 다양한 형태로 존재한다. 임베디드 소프트웨어 테스팅이 일반 데스크탑 기반의 소프트웨어에 비해 용이하지 않은 이유는 다음과 같다.

첫째, 임베디드 소프트웨어는 하드웨어 계층, 하드웨어 추상화 계층 (hardware abstraction layer, 이하 HAL)[2], 운영체제(operating system)가 맞춤화되어 개발된다. 임베디드 소프트웨어의 상당 부분은 타겟 보드에 의존적은 어셈블리 및 탑재된 커널에 맞춤화된 라이브러리를 포함하기 때문에, 각 계층 간의 인터페이스 및 사용되는 코드는 표준화되어 있지 않다.

둘째, 하드웨어 계층, 하드웨어 추상화 계층, 운영체제는 서로 긴밀하게 연관되어 있어, 이 계층들을 기반으로 한 임베디드 소프트웨어에 대한 테스트 결과를 분석하는 것이 용이하지 않다. 즉, 하드웨어, 운영체제, 어플리케이션에 대한 전문적인 지식과 경험 없이는, 입출력 장치의 특성 및 시간에 의존적인 테스트 결과를 제대로 분석하는 것이 용이하지 않다.

셋째, 임베디드 소프트웨어는 서로 각 계층 간

의 상호 작용을 토대로 실행되는 소프트웨어로서, 오류가 발생하였을 때, 오류의 발생 위치가 어디인지, 오류의 원인은 무엇인지, 또한 발생된 오류는 정확한지에 대한 판단이 용이하지 않다.

이에 커널(kernel)을 중심으로 각 계층 간의 유기적인 상호작용을 토대로 임베디드 시스템이 동작한다는 점에 착안하여, 계층 간 통합 유형을 고려한 임베디드 소프트웨어의 테스트 방안에 대해 소개한다.

구성은 다음과 같다. 2장에서는 임베디드 소프트웨어 테스트의 동향에 대해 기술하고, 3장에서는 임베디드 시스템의 계층별 통합 유형에 대해 기술하고, 4장에서는 계층 간의 통합 유형을 고려한 테스트 방안에 대해 기술하고, 5장에서는 결론을 기술한다.

2. 임베디드 소프트웨어 테스트의 동향

임베디드 시스템 테스트에 대한 연구는 어플리케이션, 커널, 하드웨어에 대하여 정형 명세 기반의 테스트 케이스 생성, 전통적인 black-box 및 white-box기반의 테스트 기법, 모니터링 기법과 같이 다양한 기법이 도입되고 있다.

정형 명세(formal specification)를 활용한 임베디드 시스템 테스트를 위한 테스트 케이스 생성에 대한 많은 연구가 존재한다. [3]은 실시간 시스템의 입출력 시그널에 초점을 두고 timed Wp-method를 제안하였다. Wp-method는 non-deterministic timed automata로부터 시간적 속성을 반영한 테스트 케이스를 생성함으로써, 임베디드 시스템의 주요 특징인 실시간 속성(property)에 초점을 둔 테스트 방안이다. UPPAAL[4]은 integer, structured data types, channel synchronization과 같은 확장된 기능을 가진 timed automata로 모델링이 가능하며, TRON과 같은 실시간 운영체제 및 다양한 임베디드 어플리케이션을 검증하기 위한 검증도구로서 널리 사용되고 있다. 이러한 정형 명세에 기반 한 방법은 시스템 정확히 이해하고 시간 속성과 같은 임베디드 시스템에 특화된 핵심 특징들을 파악하고 모델링해야 하기 때문에 많은 경우 적절한 specifications가 존재하지 않는다.

[5]는 class diagram이나 state diagrams와 같은 임베디드 어플리케이션에 대한 informal specification을 중심으로 테스트 하였으며, [6]은 어플리케이션 소스 코드에 존재하는 커널 API 및 전역 변수와 같은 인터페이스를 중심으로 실시간 운영체제를 테스트하였다.

전통적인 테스트 방법은 커널 테스트에 주로 사용되고 있다. [7], [8]은 오류 기반(fault-injection based)의 테스트 방법을 사용하였다. [9]는 black-box 기반의 boundary value 분석 기법과 white-box기반의 basis-path 기법을 사용하였으며, Linux 커널은 gcov [10]나 lcov [10]는 statement coverage나 function coverage처럼 코드 기반의 테스트 자동화 도구가 존재한다. 테스트 기법은 아니지만 [11]은 Linux 커널의 유지보수를 위하여 커널 내 각 모듈들에 대해 전역 변수의 definition 및 use 유형을 분석하여, 각 모듈간의 위험성을 분석하였다.

[12]는 하드웨어 설계 언어인 HDL (hardware description language)를 중심으로 data-flow 분석 기법을 사용하여 하드웨어 circuits의 정확성(correctness)를 테스트하였다.

[13]은 light-weight 모니터링 프레임웍(framework)을 구축하여 실시간 시스템을 모니터링하고 디버깅하는데 활용하였다.

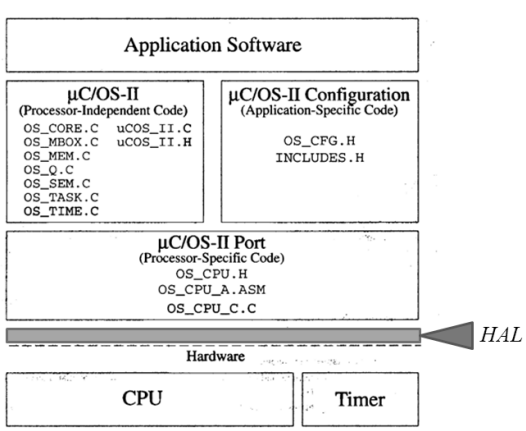
그러나 이러한 기법들은 독립적인 계층에 대한 테스트이거나 어플리케이션 계층에 치중된 커널과의 인터페이스만을 고려하였다. 임베디드 시스

템이 커널을 중심으로 유기적인 상호 작용을 한다는 것을 착안한다면, 각 계층 간의 통합 유형을 분석하고 계층 간 정보를 모니터링하기 위한 테스트 방안이 필요하다.

3. 임베디드 시스템의 계층별 통합 유형

(그림 1)은 microC/OS-II에 대한 시스템 구조를 나타내며[14], 이 커널은 commercial-grade로 항공 및 의료 분야에서 널리 사용되는 실시간 커널이다[15]. (그림 1)에서 보는 것과 같이 어플리케이션, 커널, HAL, 하드웨어 계층으로 나눈다. (그림 1)은 microC/OS-II 특화된 내용으로, 모든 운영체제의 커널이 파일 이름이나 함수의 이름은 다르게 표기될 수 있으나 유사한 기능을 가지고 있다.

HAL 계층은 타겟 하드웨어 벤더(vendor)가 자신의 하드웨어에 맞춤하여 작성하는 소프트웨어로 하드웨어에 매우 의존적일 뿐 아니라, HAL이 시스템 내부에 숨겨져 있기 때문에 대부분의 어플리케이션 프로그래머들은 HAL계층의 소프트웨어의 중요성을 간과하기 쉽다. 특히, HAL은 USB, UART, LED, LCD, switch와 같이 타겟 보드(target board) 상에 존재하는 하드웨어 디바이스들을 관리하기 위한 소프트웨어로, 하드웨어 디바이스들을 사용하는 모든 어플리케이션 및 커널은 반드시 HAL 계층과의 상호 작용이 존재한다.



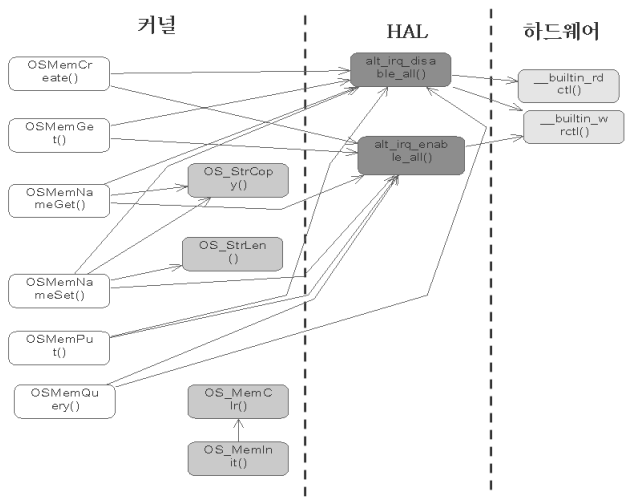
(그림 1) MicroC/OS-II의 시스템 구성도 [14]

각 계층 간의 통합 유형은 커널과 HAL의 통합, HAL과 하드웨어의 통합, 어플리케이션과 HAL의 통합, 어플리케이션과 커널의 통합, 어플리케이션과 하드웨어의 통합, 커널과 하드웨어의 통합과 같이 6개의 유형이 있다. 각 유형별 설명은 세부 절에서 기술한다. 각 세부 절에서는 Altera[16]에 의해 제공되는 하드웨어, HAL, 커널 및 어플리케이션 컴포넌트들을 이용하여 구축한 rapid prototyping system를 예제로 하여, 각 계층 간의 통합 유형을 기술한다. 때문에 일부 API나 함수의 이름이 Altera에 맞춤되어져 있으나, 이름만 다를 뿐 대부분의 시스템에 공통적으로 내장되는 기능이다.

3.1 커널과 HAL의 통합

커널과 HAL이 통합되는 주요 범위는 임계영역(critical section)으로의 진입 및 해제 시다. 이는 커널 내에 존재하는 임계영역 진입 시에 반드시 IRQ(interrupt request)를 비활성화(disabled)하여 다른 프로세스가 해당 영역을 침범하지 못하도록 해야 하기 때문이다. 즉, 커널 내부의 임계영역 진입 또는 해제 시에 HAL은 IRQ와 직접적으로 HAL API (application program interface)를 이용하여 해당 레지스터의 값을 읽고 씀으로써 IRQ를 비활성화 또는 활성화 한다.

(그림 2)는 커널[특히 (그림 1)의 os_mem.c]과 HAL에 대한 통합을 나타낸다. OSMemGet() 함수의 경우 일정 메모리를 얻기 위해 우선 alt_irq_disable_all()함수를 호출하여 IRQ를 비활



(그림 2) 커널(메모리 관리 부분)과 HAL의 통합 예

성화 한다. 이때 alt_irq_disable_all() 함수는 __builtin_rdtcl과 __builtin_wrctl 이라는 레지스터 제어용 매크로를 이용하여 현재 콘트롤 레지스터(control register)의 값을 읽은 후, 그 값이 활성화 상태이면 비활성화 상태로 전환한다. 이후 실제 메모리를 얻은 후 alt_irq_enable_all()이라는 함수를 통해 다시 컨트롤 레지스터의 값을 인터럽트 활성화 상태로 전환한다. 이때 alt_irq_disable_all() 함수는__builtin_wrctl을 내부적으로 호출한다. 단, 대상 시스템이 Altera [16] 타겟 보드에 의해 제공받는 HAL API를 사용하기 때문에 'alt' 라는 prefix가 붙는다.

3.2 HAL과 하드웨어의 통합

일반적으로 HAL은 하드웨어 디바이스들을 일반화하여 추상화한 소프트웨어로서, 하드웨어 디바이스가 바뀔 때마다 얼마든지 수정이 가능하다. HAL은 하드웨어 벤더에 의해 제공되어야 하는 부분이기 때문에, 하드웨어 장치가 바뀔 때마다 바뀔 수 있다.

3.3 어플리케이션과 HAL의 통합

어플리케이션 계층은 HAL API 를 호출함으로써 각종 타이머, 알람, LED, LCD와 같은 각종 하드웨어 디바이스들을 제어 할 수 있다. 예를 들어, alt_alarm_start()라는 HAL API를 이용하여 HAL 계층에서 정의한 알람을 시작한다. 어플리케이션은 onTick()이라고 불리는 틱 함수를 정의한 후, HAL은 TIMER_INTERVAL과 onTick()함수의 정보를 사용하고, 그 결과를 다시 어플리케이션 프로그램에게 알려줌으로써 알람 동작을 시작한다.

3.4 어플리케이션과 커널의 통합

어플리케이션 계층은 커널 API 를 호출함으로써 커널의 내부에 접근 할 수가 있다. 이때, 커널은 대상 시스템에 맞춤화되어 탑재되기 때문에, (그림 1)의 OS_CPU_C.c와 같이 운영체제를 포팅(porting)하는 목적으로도 존재한다. 하지만 대부분의 API가 어플리케이션 소프트웨어에 대한 스케줄링 및 자원 관리 목적으로 존재한다.

예를 들면, 어플리케이션의 main() 함수는 OSTasCreateExt()[어플리케이션 태스크를 생성하는 커널 API]를 통해 생성된 task1() 및 task2()를 생성하고, task1()과 task2()는 OSTimeDlyHMSM()[동기화를 위한 시간지연 커널 API]를 통해 각 태스크의 실행 간격을 조절한다. OSStart()[스케줄러를 시작하는 커널 API]라는 함수는 최상위 우선순위의 태스크를 찾아 실행시키는 함수로 커널에 의해 정의되고, 커널 내의 스케줄러에 의해 사용되는 함수이다. 이때 'OS'가 prefix로 붙은 함수명은 커널 API를 구별하기 위한 규약이다.

3.5 어플리케이션과 하드웨어의 통합

어플리케이션 계층은 변수, 어셈블리 언어, 매크로 함수 등을 사용함으로서, 메모리 블럭, 레지스터, 하드웨어 입출력 장치에 값을 직접적으로 읽거나 쓸 수 있다. 예를 들어, 어플리케이션에서 file operations, 파일 포인터(file pointer), 디바이스 입출력(I/O)를 위한 변수(variable)를 이용하여 각 장치를 제어할 수 있다.

3.6 커널과 하드웨어의 통합

커널 및 하드웨어 간의 직접적인 통합은 거의 존재하지 않으며, 존재한다고 하더라도 어플리케이션 계층을 경유하여 커널 및 하드웨어 간의 통합이 가능하다.

4. 계층 통합 유형에 따른 테스트 방안

본 장에서는 계층 통합 유형에 따른 테스트 방안에 대해 소개한다. 임베디드 시스템 내의 계층 간 통합은 서로 다른 함수간의 호출을 근간으로 이루어진다. 따라서 호출되는 함수 간에 주고받는 변수를 파악하여, 임베디드 시스템의 통합에 관계하는 데이터를 분석하는 것이 핵심이다. 하지만 HAL과 하드웨어의 통합 및 커널과 하드웨어의 통합은 하드웨어에 의존적일 뿐 아니라, 장치의 초기 장착 시에 주로 발생하므로 제외한다.

어플리케이션, 커널, HAL 소스코드 상에 존재하는 변수는 각 계층의 상태를 나타내고, 해당 변수가 레지스터(register)나 메모리와 같은 물리적인 주소 공간에 값이 써질 때 하드웨어와의 상호작용이 발생한다. 단, 실제 임베디드 시스템에는 여러 개의 어플리케이션이 지속적으로 실행이 되며 이로 인한 스케줄링 및 공유 자원 (shared variables)에 대한 문제가 늘 발생하고 있다. [17], [18], [19]와 같이 최근 기술 동향을 살펴보더라도 data-flow 분석 기법을 중심으로 2개 이상의 어플리케이션들이 실행되는 상황에서 테스트를 수행하였다. <표 1>은 3장에서 기술한 커널과 HAL의 통합, 어플리케이션과 HAL의 통합, 어플리케이션과 커널의 통합, 어플리케이션과 하드웨어의 통합 관점에서 [17], [18], [19]를 기술한다.

[17]은 어플리케이션을 임베디드 시스템의 진입점으로 간주하고, 커널 및 HAL API의 파라미터로 전달되는 변수, 전역 변수 등 메모리 및 레지스터에 표현 될 수 있는 변수들을 모니터링 한다. 때문에 한 개의 어플리케이션 태스크(task)를 중심으로 그 태스크가 커널, HAL, 하드웨어와 가질 수 있는 정보를 테스트 할 수 있다. 하나의 어

〈표 1〉 계층 통합 유형에 따른 테스트 방안 비교

	[17]	[18]	[19]
커널과 HAL 통합	O	X	X
어플리케이션과 HAL 통합	O	X	X
어플리케이션과 커널 통합	O	X	X
어플리케이션과 하드웨어 통합	O	O	O

플리케이션 태스크에 대해 계층 간의 분석이 완료 되었다면 태스크 간에 발생할 수 있는 문제점(예. 두 태스크 모두 공유하는 커널 변수)들을 분석 가능하다. [18]은 multi-threaded 환경에서 concurrent definition-use pairs를 식별하고, model checking을 통해 식별된 definition-use pairs을 검증함으로써 shared memory나 물리적인 장치에서 발생할 수 있는 concurrency faults를 발견하기 위한 방안을 제시하였다. [19]는 각 태스크간의 context-flow를 고려하여 어플리케이션과 하드웨어사이의 통합에 초점을 두고 테스트 하였다. 특히 inter context-flow는 어플리케이션 계층에 존재하는 shared variable에 특화된 definition-use pairs를 식별한다. [17]과 달리 [18]과 [19]가 커널 및 HAL 계층에서 발생 할 수 있는 오류는 발견할 수 없지만, 여러 개의 어플리케이션 태스크에 대한 실행 순서를 고려하여 shared variables만을 위한 definition-use pairs 식별함으로써, concurrency faults에 특화된 기법이다.

5. 결 론

smart phone 이나 smart TV에 탑재되는 수많은 어플리케이션 프로그램이 운영체제, 미들웨어, HAL 및 하드웨어와 연동하고 있다. 그러나 임베디드 시스템 테스팅의 현실은 아직도 많은 부분이 개발자의 경험에 의존한다. 특히 어플리케이션 개발자들은 운영체제(특히 커널)나 HAL과 같은 소프트웨어가 하드웨어 및 커널 내부의 세부 모듈의 데이터 구조까지(예. control register, task control block, event control block) 파악하고 테스팅을 수행하는 것이 쉽지 않다.

현재 많은 기법들이 단위(unit) 테스트 수준에서 각 계층에 대한 독립적인 테스트를 수행하고 있으며, 계층 간의 통합을 고려하더라도 어플리케이션 계층에 치중된 커널과의 인터페이스만 고려하는 실정이다. 현재 임베디드 시스템이 커널을 중심으로 유기적인 상호 작용을 한다는 것을 착안한다면, 각 계층 간의 통합 유형을 분석하고 계층 간 정보를 효과적으로 모니터링하기 위한 테스트 방안이 절실히 필요하다. 최근 이를 위해 data-flow 기반의 분석 기법에 초점을 둔 테스트 방안들이 제안되고 있으며, 이러한 기법들이 이론적인 수준에서만 머무르는 것이 아니라, 실제로 산업 현장에서 사용되도록 기법에 대한 검증 및 자동화가 필요하다.

참 고 문 헌

[1] 국내시장 분석-임베디드 소프트웨어 개발 도구 시장 동향, 한국소프트웨어 진흥원, 2005.10.

[2] A. Jerraya and W. Wolf, *Hardware/Software Interface Codesign for Embedded Systems*, IEEE Computer, Vol.8, Issue 2, pp.63-69, 2005.

[3] A. En-Nouaary and R. Dssouli and F. Khendek, *Timed Wp-method: Testing real-time systems*, IEEE Transactions on Software Engineering, Vol.28, No.11, pp.1203-1238, 2002.

[4] G. Behrmann and A. David and K.~G. Larson, *A tutorial on UPPAAL*, (http://www.uppaal.com), 2004.

[5] Tsai,W., Yu,L., Zhu,F., and Paul,R., *Rapid embedded system testing using verification patterns*, IEEE Software, Vol.22, No.4, pp.68-75, 2005.

[6] A. Sung, B. Choi, and S. Shin, *An Interface Test Model for Hardware-dependent Software and Embedded OS API of the Embedded System*, Elsevier Journal of Computer Standards and Interfaces, Vol.29, pp.430-443, 2007.

[7] J. Arlat, J. Favre, M.Rodriguez, and F. Salles, *Dependability of COTS Microkernel-based systems*, IEEE Transactions on Computers, Vol.51, pp.138-163, 2002.

[8] P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz, *Comparing operating systems using robustness benchmarks*, in the Proc. of International Symposium on Reliable Distributed Systems, pp.72-79, 1997.

[9] M. A. Tsoukarellas, V. C. Gerogiannis and K. D. Economides, *Systemically testing a real-time operating system*, IEEE Micro, Vol.15, pp.50-60, 1995.

[10] Yu,L., Schach,S.R., Cehn,K., and Offutt,J., *Categorization of common coupling and its application to the maintainability of the Linux kernel*, Vol.30, No.10, pp.694-705, 2004.

[11] A test coverage program, http://gcc.gnu.org/onlinedocs/gcc/Gcov.html, 2010.

[12] Zhang,L. and Harris,I.G., A *data flow fault coverage metric for validation of behavioral HDL description*, in the Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp.369-372, 2000.

[13] Junior, J. C. and Renaux, D., *Efficient monitoring of embedded real-time systems*, in the Proc. of Information Technology: New Generations (ITNG), pp.651-656, 2008.

[14] J.J. Labrosse, *MicroC/OS-II the Real-time kernel*, CMP Books, 2001.

[15] μC/OS-II Kernel, http://www.micrium.com/products/rtos/kernel/rtos.html, 2010.

[16] FPGA, CPLD, and Structured ASIC, http://www.altera.com, 2010.

[17] A. Sung, W. Srisa-an, G. Rothermel, and T. Yu, *Testing Inter-layer and Inter-task interactions in RTES applications*, in the Proc. of IEEE Asia-Pacific Software Engineering Conference (APSEC), 2010.

[18] Lai,Z., Cheung,S.C., and ChanW.K., *Inter-context control-flow and data-flow test adequacy criteria for nesC applications*, in the Proc. of ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), pp.94-104, 2008.

[19] Chen, Jun and MacDonald, Steve, *Testing concurrent programs using value schedules*, in the Proc. of IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE), pp.313-322, 2007.

저 자 약 력

성 아 영

이메일 : ahyoung.sung@samsung.com

- 1996년 3월~2000년 2월 이화여대 컴퓨터학과(학사)
- 2000년~2002년 2월 이화여대 컴퓨터학과(석사)
- 2002년 3월~2007년 8월 이화여대 컴퓨터학과(박사)
- 2007년 9월~2010년 3월 Univ. of Nebraska–Lincoln (Post Doc.)
- 2010년 5월~현재 삼성전자 영상디스플레이 사업부
- 관심분야 : embedded software testing, static analysis, test automation