임베디드 소프트웨어를 위한 오류 기반의 타이밍 테스트

성아영, 조나경°, 석문주, 최병주
이화여자대학교 컴퓨터학과
{aysung, nkcho}@ewhain.net, reflita@naver.com, bjchoi@ewha.ac.kr

Fault Based Timing Test for Embedded Software

Ahyoung Sung, Nakyung Cho°, Moonjoo Soek, Byoungju Choi
Dept. of Computer Science and Engineering, Ewha Womans University

요 약

시간과 관련된 테스트는 임베디드 소프트웨어뿐 아니라 전체 임베디드 시스템의 신뢰도에도 큰 영향을 미치기 때문에, 시간과 관련된 임베디드 소프트웨어 테스트는 필수적이다. 임베디드 소프트웨어는 실시간 운영체제와 대상 하드웨어와 유기적으로 연관되어 있어 테스트가 일반 패키지 소프트웨어에 비해 용이하지 않다. 본 논문에서는 시간과 관련된 임베디드 소프트웨어 테스트를 위해 필요한 항목들을 분석하였으며, 사례 수행을 통한 실험 결과를 제시한다.

1. 서론

최근 임베디드 소프트웨어 산업은 기술 개발 및 시장 경쟁에 의해 빠르게 변화하고 있다. 마이크로프로세서의 가격 저하, 소형화 및 고성능화가 진행되고, 제품 경쟁력의 핵심이 하드웨어 생산에서 소프트웨어 최적화 기술로 이동함에 따라 상품의 가치가 소프트웨어에 의해 좌우되고 있으며, 홈 네트워크 및 유비쿼터스 (ubiquitous) 산업과 같은 컴퓨팅 패러다임의 등장으로 임베디드 소프트웨어의 응용 분야는 더욱 넓고 다양해졌다.

임베디드 소프트웨어에 대한 관심 및 비중이 커짐에 따라 임베디드 소프트웨어 테스트 또한 중요한 이슈로 부각되고 있으며, 임베디드 소프트웨어는 다음과 같은 측면에서 일반 소프트웨어와 차별화된 특징을 갖는다.

첫째, 개발 환경과 실행 환경이 다르다. 개발은 주로 호스트 컴퓨터에서 하지만 타겟(target) 보드에서 실행하기 때문에, 소프트웨어를 탑재한 후 실행 상태에서 임베디드 소프트웨어의 오류를 발견하고 수정하는 것이 용이하지 않다. 따라서 탑재이전에 임베디드 소프트웨어에 대한 엄격한 테스팅이 필요하다.

둘째, 임베디드 시스템은 이벤트가 발생했을 때 그 이벤트에 대해 즉각적으로 응답을 해주어야 하는 실시간 시스템이기 때문에, 많은 임베디드 시스템에서 실시간 운영체제(Real-Time OS)를 사용하고 있다. 실시간 운영체제는 내/외부에서 이벤트가 발생했을 때, 이벤트 발생 시간과 그 이벤트 처리를 시작할 때까지의 지연시간이 미리 제시된 시간을 넘지 않는 시스템에 사용되는 운영체제이다[1]. 따라서 시간과 관련된 부분은 임베디드 소프트웨어뿐 아니라 전체 임베디드 시스템까지 영향을 미치기 때문에, 임베디드 소프트웨어 테스트에 있어 매우 중요하다.

특히 임베디드 소프트웨어는 실시간 운영체제와 대상 하드웨어와 유기적으로 연관되어 있어 테스트가 일반 패키지 소프트웨어에 비해 용이하지 않다. 본 논문에서는 시간과 관련된 임베디드 소프트웨어 테스트를 위하여 고려해야 할 항목들을 분석하고, 제안한 방안을 임베디드 시스템인 Programmable Logic Controller (PLC) Processor Module [2]에 적용한 실험 결과를 기술한다.

본 논문의 구성은 다음과 같다. 2장에서는 시간과 관련된 임베디드 소프트웨어 테스트 방안에 대해 기술하고, 3장에서는 본 논문에서 수행한 사례에 대해 기술하고, 4장에서는 결론 및 향후 과제를 기술한다.

2. 테스트 방안

본 장에서는 임베디드 소프트웨어의 타이밍 테스트를 위한 테스트 방안에 대해 기술하고자 한다.

2.1 임베디드 시스템의 구조

그림 1은 본 논문의 사례 대상이 된 임베디드 시스템의 구조이다.

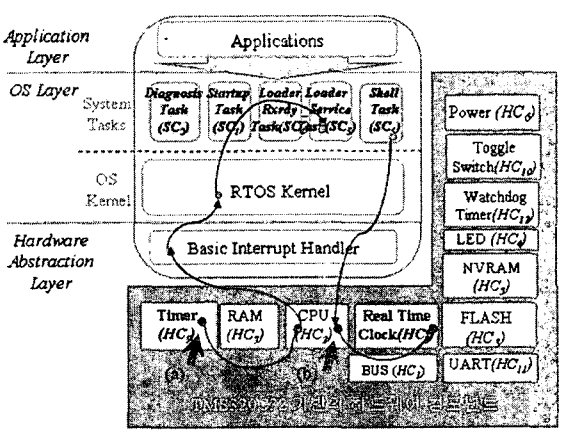

그림 1. PLC Processor Module의 구조

그림 1은 Texas Instruments사의 TMS320C32[3]를 기반으로 RS232C 통신을 사용하여 어플리케이션 프로그램에 대한 자가 진단 및 제어를 담당하는 PLC Processor Module의 구조이다.

PLC Processor Module의 하드웨어는 CPU, RAM, 타이머(Timer), Real Time Clock(RTC), LED, UART, FLASH 등과 같이 TMS320C32 DSP보드에 있는 물리적인 장치이다. PLC Processor Module의 소프트웨어는 대상 프로세서(processor)에 맞춤 된 인터럽트 핸들러(handler), uC/OS 커널[4], 커널과 맞물려서 돌아가는 시스템 태스크들로 구성한다.

본 논문에서 대상으로 하는 임베디드 소프트웨어는 시스템 태스크이다. 시스템 태스크는 CPU를 자기 자신이 독점하고 있다고 생각하는 프로그램으로, 독립적이면서도 실행 가능한 소프트웨어이다.

PLC Processor Module의 경우, 그림 1에서 보는 바와 같이, 시작 태스크(Startup task), 쉘 태스크(Shell task), 진단 태스크(Diagnosis task), 준비 태스크(LoaderRxrdy task), 서비스 태스크(Loader_Service task)와 같이 총 5개의 태스크가 존재한다[2].

2.2 오류 기반의 타이밍 테스트 방안

그림 1에서 보는 바와 같이 임베디드 시스템은 하드웨어와 어플리케이션 계층, 운영체제 계층, 대상 프로세서에 특화된 하드웨어 추상화 계층과 같이 서로 다른 종류의 소프트웨어 계층들이 맞물려서 존재하는 시스템이다[6].

일반적으로 오류 삽입 기법(fault injection technique)은 시스템의 신뢰도 (dependability)를 위하여, 하드웨어나 소프트웨어에 오류를 삽입하여 대상 하드웨어나 소프트웨어가 어떻게 동작(behavior)하는지를 관찰하는 기법[5]이다.

본 논문의 대상이 되는 시스템 태스크들의 경우, 실시간 운영체제, 타이머와 같은 하드웨어와 맞물려 실행되는 소프트웨어이기 때문에, 발생하는 오류는 그 원인을 판단하기가 용이하지 않다. 따라서 인위적으로 코드에 오류를 삽입하여 대상 시스템의 행동을 관찰기 위하여 오류 삽입 기법을 도입하였다[7].

오류 삽입 기법을 이용한 테스트를 수행하기 위해서는 다음과 같은 두 과정이 필요하다. 첫째, 어느 부분에 영향을 미치는지를 파악하여 오류를 인위적으로 삽입하는 위치(fault injection target. 이하 FIT)를 결정하는 과정이다. 둘 째, 오류 삽입 위치가 결정 되었다면 그 부분에 어떻게 오류를 심는지에 대한 과정이다. 각 과정에 대한 세부 설명은 다음과 같다.

2.2.1 오류 삽입 위치의 결정

오류 삽입 위치를 결정하기 위해서는 시스템 태스크의 특성을 파악해야 한다. 시스템 태스크는 탑재할 대상 프로세서(processor)에 맞게 구성되고, 실시간 운영체제와 유기적으로 연관(tightly coupled)된다.

PLC Processor Module의 경우 그림 1의 (a)와 (b)와 같이 시스템 태스크, 실시간 운영체제, 타이머 및 Real Time Clock 처럼 타이밍과 관련된 하드웨어에 대한 관계를 나타내고 있으며, (a)와 (b)에 대한 설명은 다음과 같다.

(a)의 경우, 시스템 태스크에서 시간과 관련된 시스템-콜(System Call)을 호출한 경우이다. PLC Processor Module 의 경우 'OSTimeDly(int timer_tick)'와 같이 시간과 관련된 시스템 콜은 타이머로부터 하드웨어 시그널을 인식하여, 이를 처리하기 위한 제어권을 운영체제로 넘긴다.

(b)의 경우, Inline assembly를 이용하여 시간과 관련된 하드웨어를 초기화 및 설정하는 경우이다. PLC Processor Module의 경우 RTC를 초기화하고 설정하기 위해, CPU에서 RTC 관련한 특정 레지스터(register)에 값을 쓰거나 읽는다.

따라서 타이밍을 테스트 하기 위한 임베디드 소프트웨어의 오류 삽입 위치는 시간과 관련된 시스템-콜이 호출되는 위치와 타이머 레지스터에 접근하기 위한 Inline assembly 코드의 위치가 된다.

2.2.2 오류 삽입 방법

오류 삽입 위치가 결정되면, 다음과 같은 방법으로 소스코드를 변경함으로써 오류를 삽입한다.

① 오류 삽입 위치가 시스템-콜인 경우

PLC Processor Module에서 사용된 uC/OS의 경우, 표 1에서와 같이 '시간 지연', '시간 지연 취소', '시간 값 읽기', '시간 값 설정'과 같은 4가지 형태의 시스템-콜이 존재하며 각 시스템 콜은 오류 삽입 위치가 된다. 표 1은 오류 삽입 위치에 해당하는 각 시스템-콜 별 오류 삽입 방법을 기술한 표이다.

표 1. 시간과 관련된 시스템-콜에 대한 오류 삽입 방법

번호	오류 삽입 위치 (해당 시스템-콜)	오류 삽입 방법	시스템-콜의 형태
1	OSTimeDly()	함수의 생성/삭제 및 타이머 틱 변경,	시간지연
	OSTimeDlyHMSM()	함수의 생성/삭제 및 시간 값 변경	
2	OSTimeDlyResume()	대상 태스크의 우선순위 변경	시간 지연 취소
3	OSTimeGet()	설정된 클럭 틱의 값 변경	시간 읽기
4	OSTimeSet()	클럭 틱 값 설정 변경	시간 설정 (쓰기)

② 오류 삽입 위치가 타이머 레지스터인 경우

오류 삽입 위치가 전역 변수로 선언된 타이머 레지스터인 경우, 해당 타이머 레지스터의 주소 및 주소의 값을 바꾼다.

2.2.3 테스트 데이터 선정

본 논문에서 대상으로 하는 PLC Processor Module의 경우 어플리케이션의 다운로드를 통해 각 시스템 태스크들이 운영체제 및 하드웨어와 맞물려서 무한 루프를 실행하기 때문에, 테스트 데이터 선정이 용이하지 않다.

특히 실시간 운영체제와 맞물려서 수행되는 시스템 태스크에 대한 타이밍 테스트의 경우, 테스트 시 2.2.1절과 2.2.2절에서 기술한 위치가 수행되어야 하며, 결과 값은 해당 위치에서의 값이 시스템의 설계 명세서에 기술된 값과 동일해야 한다.

본 논문에서 수행하는 테스트의 목적은 테스트 데이터 선정 자체가 아니며, 시간에 영향을 받는 시스템 태스크들의 기능이 의도한대로 수행되는지를 테스트하기 위함이다. 시스템 태스크들이 올바르게 동작하기 위해서는 어플리케이션들이 실행되어야 하며, 이때 어플리케이션으로부터 시스템 태스크들로 넘어오는 입력 데이터가 실제 테스트 데이터가 된다.

시스템 태스크를 위한 테스트 데이터를 선정하는 기준은 S와 S_{TN}'의 결과 값을 차별화 하는 입력 데이터를 선정한다[8]. S는 테스트 대상이 되는 임베디드 소프트웨어를 나타내고, S_{TN}'는 소스코드 상에서 ① 또는 ②에 대응하는 FIT의 해당 위치에 오

류를 삽입한 코드를 의미한다.

3. 사례 수행

본 장에서는 그림 1에서 기술한 PLC Processor Module의 시스템 태스크에 대한 타이밍 테스트를 위한 환경 및 사례 수행 절차를 기술하고자 한다.

3.1 테스트 환경 및 절차

PLC Processor Module의 시스템 태스크에 대한 타이밍 테스트를 위한 테스트 환경은 그림 2에서 보는 바와 같이, 호스트 컴퓨터와 테스트 대상 시스템인 TMS320C32 DSP (digital signal processor)기반의PLC Processor Module 및 에뮬레이터를 연결하고, RS-232C를 연결하여 시리얼 통신을 수행한다. 이때, 소스코드의 수정 및 테스트는 호스트 컴퓨터에 설치한 통합 개발 환경 지원 도구인 Code Composer[9]를 이용한다.

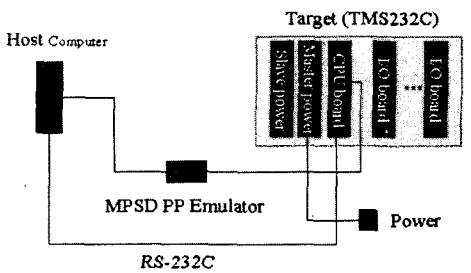

그림 2. 실험 환경

사례 수행 절차는 다음과 같다. 첫째, 대상 시스템 태스크에 오류 삽입 위치를 결정한 후, 2.2.2절에 기술한 방법과 같이 오류를 삽입한다. 이때, 오류를 삽입한 위치에 브레이크 포인트를 설정한다. 둘째, 시스템 태스크를 포함한 실시간 운영체제를 컴파일 한 후, 이를 PLC Processor Module로 다운로드 한다. 셋째, 호스트 컴퓨터에서 작성한 어플리케이션 프로그램을 다운로드 하고, 미리 설정한 브레이크 포인트의 값을 통해 시스템 태스크에 대한 테스트 결과를 확인한다.

3.2 사례 수행 결과

본 논문에서 수행한 사례 수행 결과는 표 2와 같다. S는 그림 1에서 기술한 5개의 각 시스템 태스크의 번호와 각 시스템 테스크를 구현한 코드의 전체 라인 수를 나타낸다.

표 2. 사례 수행 결과

S_i Name (Lines)	$n(S_{TN}^*)$	n(t)	n(T)
S_1 (995)	13	76	21
S_2 (162)	11	18	6
S_3 (432)	14	81	24
S_4 (953)	16	14	5
S_5 (830)	18	57	6

$n(S_{TN}^*)$은 생성된 S_{TN}^*의 수이며, n(t)는 S를 테스트 하기 위해 생성한 데이터의 수이며, n(T)는 t 중에서 선정된 데이터의 수를 나타낸다. S_{TN}^*은 S에 OSTimeDly(), OSTimeGet(), OSTimeSet()과 같은 시스템 콜이 사용된 위치와 타이머 레지스터를 제어하기 위해 전역 변수가 사용된 위치에 오류를 삽입함으로써 생성하였다.

표 2에서 보는 바와 같이 본 실험에서는 타이밍에 영향을 미치는 부분을 소스코드에서 찾아내고, 이 부분에 초점을 두고 데이터를 선정하였다. 선정된 테스트 데이터를 통해, 의도한 우선순위에 따른 태스크들의 동작, 잘못된 타이머 틱의 값, 레지스터 주소 값 충돌과 같은 오류를 발견하였다.

4. 결론 및 향후 과제

최근 임베디드 소프트웨어 산업 기술은 기술 개발과 시장 경쟁에 의해 빠르게 변화하고 있으며, 하드웨어의 소형화 및 고성능화가 진행됨에 따라 제품의 경쟁력이 하드웨어에서 소프트웨어 최적화 기술로 이동하고 있다.

임베디드 소프트웨어는 실시간 운영체제와 대상 하드웨어와 유기적으로 연관되어 있어 테스트가 일반 패키지 소프트웨어에 비해 용이하지 않으며, 특히 시간과 관련된 테스트는 임베디드 소프트웨어뿐 아니라 전체 임베디드 시스템의 신뢰도에도 큰 영향을 미치기 때문에, 시간과 관련된 임베디드 소프트웨어 테스트는 필수적이다.

본 논문에서는 시간과 관련된 임베디드 소프트웨어 테스트를 위해 필요한 항목들을 분석한 결과, 시간과 관련된 시스템 콜이나 클럭을 선언한 레지스터 변수 값이 올바르게 사용되었는지를 확인하기 위한 기능 중심의 타이밍 테스트에 초점을 두었다. 본 논문에서는 기능 중심의 타이밍 테스트를 위하여, 오류 기반의 임베디드 소프트웨어 테스트 방안을 제시하였으며, 제안한 방안을 TMS320C32 기반의 프로세서를 탑재한 PLC 시스템에 적용한 결과를 기술하였다.

본 논문에서 수행한 기능 위주의 타이밍 테스트를 위해 고려해야 할 항목들 중 하나인 시간과 관련된 시스템 콜의 클럭 값의 경우, 실제 시스템 태스크들의 우선순위 및 시스템의 성능에도 영향을 미친다. 향후, 본 논문에서 수행한 기능 위주의 타이밍 테스트를 기반으로 하여, 임베디드 소프트웨어의 성능 및 신뢰도와 같은 비기능 테스트로의 확장 연구를 진행할 예정이다.

5. 참고문헌

[1] Qung Li and Caroline Yao, "Real-Time Concepts for Embedded Systems", CMP Books, 2003.
[2] KNICS-PLC-SDS331-01, PLC Processor Module의 운영체제 설계 명세서, 한국 원자력 연구소, 2004.
[3] TMS320C32, Digital Signal Processor, http://www.ti.com/, Texas Instruments, 1998.
[4] Jean J. Labrosse, "MicroC/OS-II, The Real-Time Kernel", CMP Books, 1999.
[5] Mei-Chen Hsueh, T.K. Tsai and R.K. Iyer, Fault Injection Techniques and Tools, IEEE Computer, Apr, pp75-82.1997.
[6] A.A. Jerraya and W.Wolf, " Hardware/Software Interface Codesign for Embedded Systems, IEEE Computer, pp63~69, Feb., 2005.
[7] Ahyoung sung, Byoungju Choi, "Interaction Testing in an Embedded System using Hardware Fault Injection and Program Mutation, LNCS, Springer, Vol2931, pp192-204, Dec., 2003.
[8] R.A.DeMillo, R.J.Lipton and F.G.Syward, Hints on test data selection: Help for the practicing programmer: IEEE Computer, 11(4):pp34~41 April, 1978.
[9] SPRU296, Code Composer User' s Guide, Texas Instrument, 1999.