한국산학기술학회논문지
Vol. 9, No. 1, pp. 41-46, 2008

임베디드 소프트웨어를 위한 디버깅 어댑터 설계

김용수[1*], 한판암[2]

The Design of Debugging Adapter for Embedded Software

Yong-Soo Kim[1] and Pan-Am Han[2]

요 약 임베디드 소프트웨어는 대상 시스템의 내부 자원과 호스트 시스템의 환경에 매우 민감하므로 수행시 대상 시스템과 동일한 환경에서 디버깅해야 한다. 그러나 대상 시스템의 자원에 직접적으로 접근하여 시스템 상태를 조사하거나 제어하는 기법들은 내부 신호나 자원에 대한 접근이 제한되어 있는 SoC (System-On-a-Chip) 소프트웨어를 디버깅하기는 부적합하다. 본 논문에서는 JTAG을 기반으로 원격 시스템에 접근하여 임베디드 소프트웨어를 디버깅할 수 있는 어댑터를 제안한다. 이는 원격 디버깅을 위한 환경 구축에 많은 어려움 있는 기존의 환경 구축의 경제적 제한점을 해소할 수 있다. 따라서 본 논문은 원격 시스템내의 임베디드 소프트웨어를 디버깅할 수 있는 경제적인 인터페이싱 환경을 제공한다.

Abstract Since embedded software is sensitive to the resources and environment of target system, it should be debugged in the same environment as actual target system. However, existing tools to debug embedded software, in which access to internal signal or resources is limited, are uneconomical. In the thesis, economical and practical JTAG Adapter that can use open GDB is suggested. It can remove existing limitations of environment implementation that have many difficulties in implementing an environment for remote debugging. Hence, the thesis provides economical interfacing environment that can debug ubiquitous embedded software inside remote system.

Key Words : Embedded Software, JTAG, USB, Remote Debugging

1. 서론

유비쿼터스 컴퓨팅 시대가 도래하면서 임베디드 운영체제를 중심으로 임베디드 소프트웨어[1, 2, 3, 4]의 수요가 가히 폭발적으로 증가되고 있다. 그로 인해서 임베디드 소프트웨어 또한 점차 복잡하고 다기능적인 동작을 요구하게 된다. 임베디드 소프트웨어는 해당 시스템의 자원과 타이밍에 민감하여 실제 시스템과 동일한 환경에서 디버깅해야 하지만, 메모리나 입출력 장치 등과 같은 모듈들을 하나의 프로세서에 집적한 SoC(System-On-a-Chip)[5, 6]는 내부 신호나 자원에 대한 접근이 제한되어 디버깅을 더욱 힘들게 한다. 아울러, 시스템의 자원도 충분하지 않기 때문에 요구조건에 적합한 SoC 프로그램을 개발하기 위해서는 자원이 풍부한 원격 시스템에서 디버깅할 수 있는 원격 디버깅 도구[7, 8, 9, 10, 11, 12, 13, 14]의 필요성이 증대되고 있다. 임베디드 소프트웨어를 디버깅하는 전통적인 기법으로 실시간 개발에 편리하고 입출력의 오류 정정을 위한 하드웨어 및 소프트웨어 기능을 가진 ICE(In Circuit Emulator)와 디지털 시스템의 논리 신호를 포착하여 디버깅하는 logic analyzer 가 있다.

본 논문에서는 산업 표준화된 JTAG(Joint Test Action Group)[10]을 기반으로 원격지의 타켓 시스템에 접근하여 SoC 소프트웨어를 디버깅할 수 있는 어댑터를 제안한다. 제안된 어댑터는 호스트 시스템의 USB(Universal Serial Bus)포트를 사용하여 대상 시스템의 JTAG에 연결되는 구조를 가진다. 이는 대상 시스템에 영향을 주지 않고 고속의 USB포트를 사용하므로서 보다 경제적이고 효과적인 원격 디버깅을 가능하게 한다. 또한, 어댑터의 내부메모리에 JTAG 핸드쉐이킹을 위한 펌웨어를 다운로드 방식으로 구현하여 사용자 변경이 용이하도록 디버깅 환경의 유연성과 기존 디버깅 도구의 확장성을 가지도록 하였다. 2절에서는 본 연구의 배경으로 임베디드 소프트

[1]거창전문대학 컴퓨터정보시스템과
[2]경남대학교 공과대학 컴퓨터공학부
*교신저자: 김용수(yskim@kc.ac.kr)

웨어 디버깅을 위한 전통적인 디버깅기법과 JTAG과 USB에 대해 살펴보고, 3절에서는 제안된 환경의 핵심적인 부분인 JTAG 어뎁터에 대하여 설명하고 핸드쉐이킹를 위한 어뎁터내의 펌웨어에 대해 설명한다. 마지막으로 결론 및 향후 과제를 제시한다.

2. 연구배경

2.1 디버깅 기법

기존의 기법 중 ICE는 하드웨어 모방이 가능하여 실시간 개발에 편리한 수단으로서 실시간 입출력의 오류 정정을 위한 하드웨어 및 소프트웨어 기능을 가진다. logic analyzer는 디지털 시스템의 논리 신호를 포착, 기록하고 나타내는 다수의 채널을 가지는 도구로 2진 형태와 사각 파형 모양의 일련의 펄스 형태로 출력한다. 이러한 기법들은 대상 시스템의 자원에 직접적으로 접근하여 시스템 상태를 조사하거나 제어하므로 내부 신호나 자원에 대한 접근이 제한되어 있는 SoC 프로그램을 위한 디버깅으로는 부적합하다. 또한, 매우 고가의 장비로 비실용적인 제약사항을 가지고 있다. 그러므로 SoC 프로세서 자체에서 제공되는 디버그 모듈을 이용하여 SoC 소프트웨어를 디버깅하는 도구를 제안하기 위해서 JTAG을 기반으로 한다.

2.2 JTAG (Joint Test Action Group)

Boundary-Scan으로 더 많이 알려져 있는 JTAG은 칩 내부에 외부 핀과 일대 일로 연결되어 있는 Boundary Cell을 두어 프로세서가 할 수 있는 모든 동작을 Cell을 통하여 모든 동작을 인위적으로 수행하므로써 하드웨어 테스트나 연결 상태 등을 체크할 수 있는 기능을 가진다. 전체적인 인터페이스는 TAP(Test Access Port)라고 하는 5개의 핀(TDI, TMS, TCK, nTRST, TDO)에 의해서 제어되며, 이를 이용하여 프로세서의 상태와는 상관없이 디바이스의 모든 외부 핀을 구동시키거나 값을 읽어 들일 수 있다. Boundary-Scan 테스트 회로는 TAP (Test Access Port) 핀, TAP 콘트롤러, 명령 레지스터과 일련의 테스트 데이터 레지스터로 구성된다. 이들은 Boundary-Scan 레지스터, 바이패스 레지스터, 디바이스 식별 레지스터, 그리고 데이터 종속 레지스터를 포함한다. [그림 1]은 이들을 나타내고 있다.

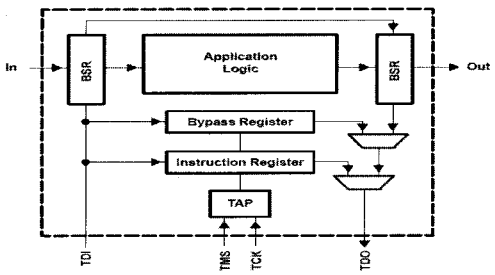

그림 1. TAP의 블록도

2.3 USB (Universal Serial Bus)

USB는 사용의 편리와, 넓은 확장성 그리고, PC와 전화와의 연결을 위해서 1994년 Compaq, Intel, Microsoft, Nec등이 주축이 되어 개발되었다. 크게 두 가지 종류의 chip이 Host쪽에서 USB와의 연결을 담당하며 Intel과 Compaq에서 제공된다. 현재 USB Spec. 2.0이 나와 있는 상태이며, Spec 1.1에서는 최대 12Mbps를 그리고, Spec. 2.0에서는 최대 120Mbps를 지원한다. USB는 계층적 구조로 구성되며, master/slave protocol을 사용해서 USB device와 통신한다. 즉, Device 측에선 Host측으로의 통신선로를 만들지 못한다. 따라서 Host에서만 통신의 시작을 할 수 있다. USB Host Controller Driver는 대부분 이미 구현된 상태이고 안정 되었기에, 본 논문에서는 해당되는 어뎁터의 USB Client Driver만을 구현한다.

3. 어뎁터의 구현

3.1 기존 도구의 문제점

리눅스 환경의 GDB 기반에서 임베디드 시스템의 JTAG를 이용한 기존의 원격 디버깅 도구는 크게 두 가지 방법으로 분류된다. 첫 번째는 임베디드 시스템과 호스트 시스템 간의 정보교환을 위해서 임베디드 시스템의 특성에 맞는 에뮬레이터를 이용하는 방법과 두 시스템간의 신호 변환만을 담당하는 어댑터를 이용하는 방법이 있다. 에뮬레이터를 이용한 도구는 임베디드 시스템에서 수행되는 프로그램을 OPENice32-Axxx[13] 나 BDI2000[10]등과 같은 에뮬레이터을 이용하여 원격 디버깅을 지원한다. 그리고 GDB기반의 Jelie[8]와 같이 어댑터를 이용한 도구는 임베디드 시스템에서 JTAG 포트를 이용하여 GDB 기반의 디버깅 명령어를 사용할 수 있도록 지원하며, 호스트 시스템의 Parallel/USB 인터페이스와 임베디드 시스템의 JTAG 포트를 제어하기 위한 기능 등을 가지고 있다.

이러한 에뮬레이터를 사용하는 기존의 도구들은 고가의

전용 에뮬레이터에서만 수행되므로 수행 환경이 제한적이라는 단점을 가진다. 그리고 어댑터를 사용하는 Jelie 도구는 호환성있는 저가의 어댑터가 없다는 문제점을 가진다. 대표적인 디버깅 기능으로 중지점(breakpoint) 기능과 추적점(tracepoint) 기능을 들 수 있는데, 기존의 도구들은 중지점 기능만을 갖는 문제점을 가지고 있다. 중지점 기능은 프로그램에 오류 가능성이 있는 지점에 중지점를 설정하여, 프로그램 수행 시에 설정된 중지점에 도달하면 프로그램의 수행을 중지시키고 사용자의 디버깅 관련 명령어를 입력받아서 변수들의 값을 확인하거나 변경하는 기능이다. 그러나, 이러한 중지점 기능은 프로그램의 논리적인 실행 흐름과 변수들에 대한 메모리 값의 변화 등을 파악할 수 없기 때문에 프로그램의 수행 양상을 실시간으로 감시하기는 어렵다. 이에 반해 프로그램에 설정된 특정한 위치에서 프로그램에 대한 제어나 간섭이 없이 수행정보를 실시간으로 기록할 수 있는 기능을 추적점이라 하는데, 기존의 오픈소스 기반의 도구에서는 실시간 운영체제와 임베디드 소프트웨어에 대한 고려가 미흡하였으므로 추적점 기능을 지원하지 않는다.

3.2 어뎁터의 설계

본 논문은 하드웨어 기반 임베디드 소프트웨어 디버깅 도구인 GDB와 Intel/PXA 프로세서를 위한 디버깅 모듈을 설계하였다. 본 도구는 임베디드 시스템의 자원 제약성 문제를 해결하기 위해서 호스트 부분과 대상 부분으로 구성된다. 호스트 시스템은 GDB 명령을 PXA210, PXA250, PXA260등의 intel 프로세서를 위한 디버깅 명령으로 전환하는 기능과 데이터 전송을 위한 USB와 대상 시스템의 JTAG을 제어하기 위한 기능을 가진다. 대상 시스템에는 수행되는 프로그램을 제어하고, 그 결과를 호스트 시스템에 반환하는 소프트웨어 모듈인 디버그 핸들러를 둔다. 이에 관한 전체적인 구조는 [그림 2]와 같다.

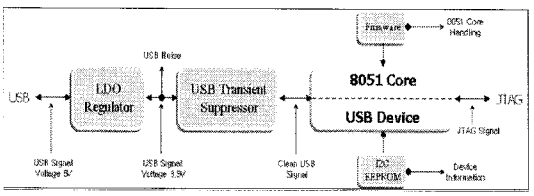

그림 2. 제안 어뎁터의 구조

또한, 호스트와 타켓 시스템간의 원활한 연결을 위하여 Adapter를 설계하고 구현하였다. Cypress사의 AN시리즈를 사용하여 제작되었고, 저가형으로 실용성이 매우 높다고 하겠다. Adapter의 중심적인 역할을 담당하는 USB 칩셋은 EZ-USB로 널리 알려진 Cypress 사의 AN칩[15]을 사용하였다. [그림 3]은 AN칩의 구조를 보인 그림이다. 그림에서도 알 수 있듯이 USB 신호를 JTAG 신호로 즉, SIE(Serial Interface Engine)에서 입출력되는 USB 신호를 적절히 제어하는 것이 필요하다.

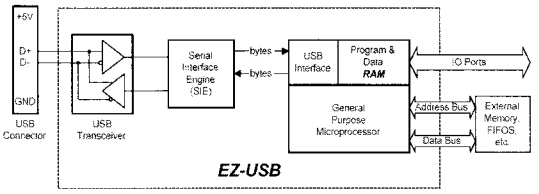

그림 3. EZ-USB 칩셋의 구조도

이를 위하여 내부메모리에 펌웨어가 요구된다. 어뎁터에는 8KB의 내부메모리가 존재하는데 USB포트로부터 전달된 신호를 대상 시스템의 JTAG 단으로 내보내기 위해서는 이 내부 메모리에 적절한 펌웨어가 다운로드 되어야 한다. 펌웨어는 일반적으로 C 코드로 작성하여 Hex code로 변환 후 호스트의 디바이스 드라이브를 통해 내부 메모리에 다운로드 되게 된다. 펌웨어가 정상적으로 연결간의 핸드쉐이킹을 원활하게 하기 위해서는 TAP(Test Access Port) 컨트롤러의 16가지 테스트 로직 오퍼레이션의 순서를 제어하기 위한 동기적인 유한 상태 머신을 이해해야 한다. TAP은 버스 마스터를 통해 제어된다. 버스 마스터는 자동적인 테스트 장비이거나 프로그램이 가능한 로직 디바이스로 테스트 액세스 포트를 인터페이스한다. TAP 컨트롤러는 TCK의 상승 엣지나 파워업에 대해서만 상태를 변화시킨다. TCK의 상승 엣지에서 테스트 모드 상태 (TMS) 입력 시그널의 값은 상태 변화의 순서를 제어한다. TAP 컨트롤러는 자동적으로 파워업시에 초기화된다. 또한 TAP 컨트롤러는 5개의 TCK주기동안 TMS 입력으로 1의 시그널을 적용함으로써 초기화될 수 있다. [그림 4]는 TAP 컨트롤러에서 발생하는 상태 전이를 나타낸다. 각 전이를 위한 값은(0 or 1) TMS 값을 의미한다. 모든 응용 프로세서의 디지털 신호들은 PWR_EN 핀을 제외하고 바운드리 스캔에 참여한다는 것에 주의해야 한다. 이것은 스캔 오퍼레이션으로 하여금 응용 프로세서의 파워를 오프하는 것을 금지시킨다.

본 연구에 사용된 AN칩은 8051 코어를 사용하므로 프로그램을 작성할 때 8051용 개발툴을 사용할 수 있다. 또 Re-Numeration 을 통해 펌웨어를 소프트웨어적으로 다운로드 할 수 있으므로 롬라이터가 별도로 필요없는 장점이 있다. 칩의 하드웨어가 기본적인 USB와 권련된 기본적인 기능은 내장하고 있으므로 펌웨어는 JTAG과 관련된 변환 모듈로 구성된다. 또한, Full Speed 를 지원하고, 충분한 수의 입출력 핀을 갖고 있다. 입출력은 A,B,C 이렇게 3 개의 8비

트 포트가 있으며 한 포트 내의 핀이라도 각각 입출력을 선택하여 설정할 수 있다. 물론 입출력 핀 중 일부는 특수한 목적의 핀으로 사용될 수도 있고 이는 펌웨어로 조작이 가능하다.

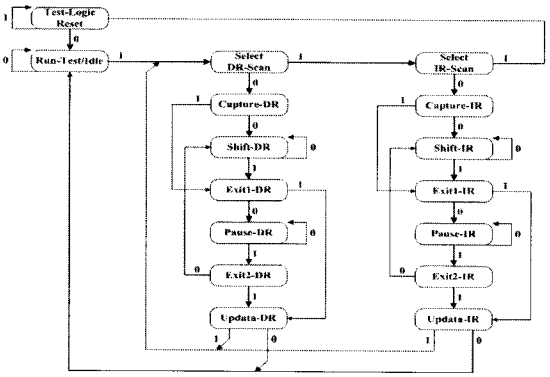

그림 4. TAP Controller의 상태 전이도

[그림 5]는 펌웨어 세부 모듈중에서 JTAG를 초기화 시키는 모듈을 C 코드로 보이고 있다.

```
void jtagReset(void)
{
    unsigned char loop_count;
    OUTB=JTAG_PWR;
    OUTB=JTAG_PWR | JTAG_CLK;
    for(loop_count = 0; loop_count < 5; loop_count++)
    {
        OUTB=JTAG_PWR | JTAG_TMS;
        OUTB=JTAG_PWR | JTAG_TMS | JTAG_CLK;
    }
    OUTB=JTAG_PWR;
    OUTB=JTAG_PWR | JTAG_CLK;
    OUTB=JTAG_PWR;
    OUTB=JTAG_PWR | JTAG_CLK;
    OUTB=JTAG_PWR;
    jtagRestarted = 1;
}
```

그림 5. 펌웨어를 위한 C코드의 일부분

3.3 어뎁터의 구현 및 실험

[표 1]은 구현된 어뎁터에서 사용되는 칩 셋의 입출력 핀(PB)과 대상 시스템의 JTAG 포트에 전달될 신호의 매핑을 보인 것이다. 입출력 핀을 제어하기 위하여 4개의 레지스터(PORTBCFG, OUTB, OEB, PINSB)를 두고 있는데, 먼저 전원 인가 시에 포트의 기능을 선택할 수 있는 PORTBCFG 레지스터는 디폴트 값으로 0으로 초기화 되어야 한다. 0은 일반적인 입출력용도를 의미하며, 1은 정해진 특수한 용도로 사용하겠다는 의미이다. OEB 레지스터는 입력의 방향을 선택하는 용도인데, 출력 용도로 사용하기 위해 펌웨어에서 각 레지스터 값을 1로 초기화해야 한다. 전원이 칩에 인가되게 되면, EZ-USB 코어는 I2C 포트에 접속하고 있는 EEPROM를 찾게 된다. 만일 EEPROM이 탐지되게 되고 0번지의 내용이 '0xB0' 이면, EZ-USB 코어는 내부 기억 장치로 Vendor ID, Product ID, Device ID을 EEPROM에서 복사한다. EZ-USB 코어는 이 때 Get_Descriptor -Device 요구의 일부로서 호스트에게 이 바이트들를 공급한다. 만일 0 주소의 내용이 '0xB2'일 때는 7 주소부터 펌웨어가 있는 것으로 간주하고 내부 메모리에 다운로드하게 된다. 본 연구에서는 24LC00 칩을 사용하여 해당 칩의 ID만을 제공한다. 호스트 시스템은 이 정보에 일치되는 드라이버가 OS에 의해 로드되게 되고, 드라이버가 EZ-USB의 RAM에 8051를 위한 펌웨어 다운로드하고, 8051 코어가 동작을 개시하게 된다.

표 1. USB & JTAG 포트의 핀 연결

B Port	JTAG
PB0	TCK
PB1	TDO
PB2	TMS
PB3	TRST
PB4	RESET
PB5	TDI
PB6	Not used
PB7	Not used

대상 시스템에 전달된 JTAG신호는 해당 프로세서에 인식되기 위하여 Debug Handler등의 내부 모듈등이 부가적으로 필요할 것이다. 디버그 핸들러는 임베디드 시스템의 미니 명령 캐쉬에 상주하여 호스트 시스템의 추적점 서버와 디버깅 메시지를 통신하는 모듈인데, 이는 인텔 XScale 코어 디버거의 구조에서 Hot-Debug 솔루션[20]을 이용하였다. 인텔 XScale 코어의 디버그 구조인 Hot-Debug는 사용자의 응용 프로그램과 관련된다. 상위 수준의 응용 프로그램은 디버거의 실행환경에서 수행된다. 사용자가 Hot-Debug의 디버그 세션을 사용하고자 할 경우, 디버거는 Hot-Debug가 임베디드 시스템에서 지원되는 지를 검사해야 한다. Hot-Debug가 지원되면, 디버거는 JTAG을 통해 외부 디버그 브레이크를 사용해서 코어에 디버그 예외를 발생시킨다. 코어는 응용 프로그램의 리셋 핸들러와 연결된 디버그 핸들러 스텁에게 실행의 제어권을 넘긴다. 스텁은 JTAG을 통해서 디버거에게 mini-IC에 디버그 핸들러 코드를 다운로드 가능하다는 메시지를 보낸다. 다운로드를 마친 디버거는 디버그 핸들러 스텁에게 다운로드된 디버그 핸들의 시작 주소를 보낸다. 그러면, 스텁은 디버그 핸들러의 시작부분으로 분기하여 디버그 세션을 시작한다. [그림 6]은 구현된 어뎁

터의 실제 모습이고, [그림 7]은 동작 실험을 위하여 어뎁터를 사용하여 실제 대상 시스템의 IDCODE를 읽어 인식되었음을 확인할 수 있는 화면이다.

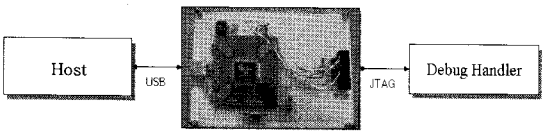

그림 6. 실제 어뎁터의 모습

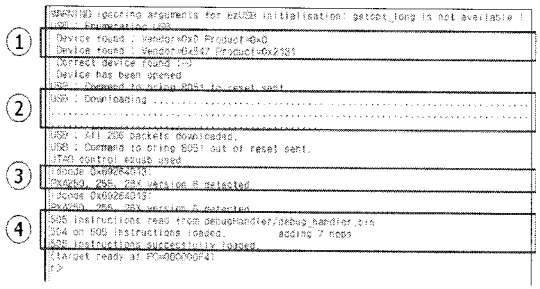

그림 7. 정상적인 동작확인을 위한 화면

4. 결론 및 향후연구

본 논문에서는 원격 디버깅을 위한 경제적인 USB-JTAG Adapter를 제안하였다. 제안된 어뎁터는 고가의 ICE 장비를 대신할 수 있는 저가의 경제성있는 장비로 이용될 수 있다. 또한, 기존의 공개 디버깅 도구을 이용하여 그 동작도 원활함을 확인하였다. 향후, PXA 시리즈 프로세서에서 제한적으로 수행되는 환경을 다양한 프로세서를 대상으로 사용할 수 있는 추가적인 연구가 필요하다.

참고문헌

[1] Heung-Nam Kim, and Chea-deok Lim, *Technology Trends and Development Strategies on Embedded Software for Ubiquitous Computing Era*, Feb. 2003.

[2] Straunstrup, J., Andersen, H.R., Hulgaard, H., Lind-Nielsen, J., Behrmann G., Kristoffersen K., Skou A., Leerberg HH., Theilgaard N.B., "Practical verification of embedded software," *Computer*, Vol. 33(5), pp. 68-75, May 2000.

[3] Alessandro Rubini, and Jonathan Corbert, *Linux Device Drivers Second Edition*, O'Reilly & Associate, Inc., June 2001.

[4] Ziruanm, Y., Dcym, S., Rodgers, M., "Test of future system-on-chips," International Conference on Computer Aided Design, IEEE/ACM, pp. 392-398, 2000.

[5] Kenneth H.Peters, Software Development and Debug for System On-A-Chip," *Embedded Systems Conference*, 1999.

[6] MacNamee, C., Heffernan, D., "Emerging on-chip debugging techniques for real-time embedded systems," *Computing and Control Engineering Journal*, pp. 295-303, Dec. 2000.

[7] Pilet, J. and S. Magnenat, *Jelie:Manuel de L'utilisteur*, Ecole Poly technique Federale De Lausanne Lap., 2003.

[8] Ashling, *PATHFINDER Source-Level Debugger for Embedded Micro-controller Development*, 2003.

[9] Kiyokuni Kawachiya, and Takao Moriyama, *A Symbolic Debugger for PowerPC-Based Hardware Using the Engineering Support Processor (ESP)*, IBM Research, Aug. 1997.

[10] Hai T. S., "Multiple Ports Remote Embedded Debugger Server," *International Conference on Electro/information Technology*, May 2006.

[11] R. Stallman, R. Pesch, S. Shebs, et al., *Debugging With GDB*, 2003.

[12] Intel Co., *Intel XScale Micro architecuture for the PXA255 Processor*, 2003.

[13] ARM Ltd., *Real-Time Debug*, 2002.

[14] Asset InterTech, Inc., and R. G. *Bennefits, Boundary-Scan Tutorial*, 2000.

[15] Cypress Semiconductor, EZ-USB Technical Reference Manual, 2002.

김 용 수(Yong-Soo Kim) **[정회원]**

- 1993년 2월 : 한국방송통신대학교 전자계산학과 (이학사)
- 1995년 8월 : 경남대학교 컴퓨터공학과 (공학석사)
- 2000년 2월 : 경남대학교 컴퓨터공학과 (박사수료)
- 2007년 6월 ~ 현재 : 거창전문대학 컴퓨터정보시스템과 부교수

<관심분야>
소프트웨어공학, 데이터베이스

한 판 암(Pan-Am Han) **[정회원]**

- 1969년 2월 : 동국대학교 (경영학사)
- 1975년 2월 : 동국대학교 (경영학석사)
- 1992년 2월 : 인천대학교 (경영학박사)
- 1980년 3월 ~ 현재 : 경남대학교 컴퓨터공학부 교수

<관심분야>
소프트웨어공학, 소프트웨어 품질관리