制御用 言語로서의 Ada

李澤熙 · 宋慶俊 · 金容準

〈要 約〉

Ada는 Embedded 시스팀을 위한 고급 프로그래밍 언어이다. 여기서는 Ada가 지닌 제어시스팀 개발용 언어로서의 특징을 살피고 Ada 개발환경인 APSE(Ada Programming Support Environment)를 분석하여 제어시스팀 개발에 응용할 수 있도록 하고자 한다.

Ⅰ. 서 론

컴퓨터 분야에서 하드웨어에 비해 소프트웨어의 개발 및 유지 보수 비용은 날로 그 비중이 높아지고 있다. 또한 소프트웨어의 규모가 커짐에 따라 그에 적합한 프로그래밍 언어와 보다 조직적이고 체계적인 개발지원환경이 필요하게 되었다. 일반적으로 소프트웨어 개발을 지원하는 환경이란 프로그래밍 언어와 개발을 위한 tool 뿐만 아니라 방법론까지도 포함한다.

Ada는 미국 국방성에서, 국방성 산하의 Embedded 컴퓨터를 대상으로 증대하는 소프트웨어 개발 및 유지 보수비를 줄이기 위해 Embedded 시스팀 개발에 사용되는 고급 프로그래밍 언어로서 개발한 것이다. 그렇기 때문에 Ada는 실시간 처리, 병렬처리, 예외상황 처리, 태스크 등 제어시스팀에 꼭 필요한 기능들을 고루 갖추고 있으며, 정보은폐(information hiding), 데이터 추상화(data abstraction) 등 소프트웨어 공학 개념을 충분히 반영한 고급언어이다. 또한 미국방성은 우수한 소프트웨어 시스팀을 용이하게 개발하기 위해서는 단순히 우수한 특성을 지닌 프로그래밍 언어만으로는 충분하지 못하며 프로그램 개발 사이클 전반에 걸쳐 이를 지원해 주는 좋은 프로그래밍 환경이 필요하다는 사실을 인식하고 Ada와 더불어 그의 지원환경인 APSE를 개발하였다. 본고에서는 Ada의 여러 특징

중 제어시스템 개발에 적합한 특징들과 APSE를 분석하여, 제어시스템 개발에 도움이 되도록 한다.

II. 본 론

1. Embedded 컴퓨터의 특성

Embedded 컴퓨터란 '장비속에 삽입(embedded)되어 장비의 일부로서 그 장비를 제어할 목적으로 쓰이는 컴퓨터'를 말한다. 예로서 비행기, 로보트, 수치제어 선반, 공정제어 등에 쓰이는 내장 컴퓨터들이 여기에 속한다. 그러므로 embedded 컴퓨터는 범용의 컴퓨터와는 달리 다음과 같은 요구조건을 만족해야 한다.

- 고도의 신뢰성 : embedded 컴퓨터의 down은 장비의 고장을 의미하며 이는 많은 인명과 막대한 경제적 손실을 초래하는 수도 있다. 그러므로 고도의 신뢰성을 보장하는 것은 embedded 컴퓨터의 가장 기본적인 요구조건이다.
- 실시간 동작 : 많은 감지기(sensor)나 작동기(actuator)와 연결되어 기계장비를 감시하고 제어하는데 실시간 응답으로 처리되어야 한다.
- 다양한 task-scheduling : 단일 컴퓨터에서 여러개의 태스크를 수행하는 경우가 일반적인데, 태스크들이 서로 밀접하게 관련되어 있고, 태스크마다 activate되는 방법도 다르며, 우선순위 처리문제 등 task-scheduling 방법이 복잡하다.
- 태스크간 통신 : 태스크들은 서로 긴밀하게 관련되어 정보를 주고받거나 한 태스크가 다른 태스크를 activate시키는 기능을 필요로 한다. 예로서 외부 데이터를 수집하는 태스크는 데이터 수집후 처리 태스크를 activate시키면서 정보를 넘겨주는 2개의 태스크간 통신을 생각할 수 있다.

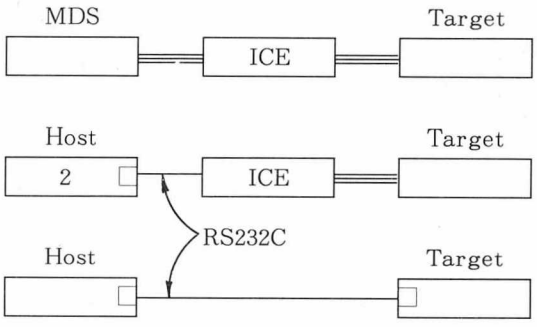
또한 embedded 컴퓨터에 내장되는 소프트웨어는 위와같은 조건을 만족시키기 위해 코드 크기가 작아야 하고 수행속도가 충분히 빠르지 않으면 안된다.

2. 제어 프로그램의 개발 환경

일반적으로 제어 프로그램의 개발환경은 host-target 개념에 기초를 두고있다. 가끔은 범용 컴퓨터가 내장 컴퓨터로 쓰이기도 하며, 이때는 그 컴퓨터 자체에 개발환경이 구축되어 자신을 target으로 소프트웨어 시스팀을 개발하기도 하나 대개의 경우 target 컴퓨터는 꼭 필요한 하드웨어만으로 이루어진 bare-machine이기 때문에 자체에 개발환경을 갖추기는 곤란하다. 그러므로 host 컴퓨터에 적절한 개발환경을 구축하고 거기에서 target 소프트웨어를 개발하게 된다.

개발된 프로그램은 주로 입출력 port를 사용하여 target에 직접 download시키거나 ICE(In-Circuit Emulator)를 사용하여 수행한다. ICE는 host의 시스팀 버스에 연결되는 형태와 입출력 port에 연결되는 형태의 두가지가 있는데 MDS(Micro-computer Development System)는 주로 전자의 형태를 취하고 후자의 경우는 범용 컴퓨터를 host로 사용할때 편리하다. 이와 같이 ICE를 사용하는 경우 target에 monitor가 없어도 되고 target 하드웨어 자체를 개발하는데도 쓰이는 강력한 기능을 구사하나, 사용이 어렵고 비싼 장비이므로 하드웨어의 개발에 주로 쓰이는 MDS에서 많이 사용한다. 직접 download시키는 방법을 쓰면 추가의 하드웨어 비용없이 개발자가 원하는 임의의 범용 컴퓨터에 개발환경을 구축할 수 있으나 target은 host로 부터 프로그램을 받을 수 있도록 입출력 port와 monitor 프로그램을 갖추고 있어야 한다. 그러므로 이 방법은 주로 target의 소프트웨어 개발에만 사용된다. 이러한 상황에 대표적인 예들을 도식적으로 나타내면 〈그림 1〉과 같다.

그리고 전통적인 개발 시스팀은 주로 MDS라 불리는 전용 컴퓨터를 host로 사용하고 target



〈그림 1〉 Host-target 환경

과 복잡하게 관련되어 있다. 그러나 개발자는 그들이 원하는 host 컴퓨터에서 target시스팀을 개발할 수 있기를 원하며, 그에 따라 host와는 독립적인 개발환경을 필요로 한다. 이는 개발하려는 target이 달라지면 그때마다 다른 MDS를 배워야 하는 불편함을 피하고 싶어하기 때문이다.

또한 개발 시스팀의 추세는 요구분석에서 target시스팀의 디버깅, 테스트까지의 전사이클을 지원함과 동시에 개발된 소프트웨어의 유지보수 문제, target 소프트웨어를 여러 사람이 공동으로 개발하는 것도 고려하여 지원할 수 있는 통합적인 개발 시스팀쪽으로 나가고 있다.

3. 제어용 언어로서의 Ada

Ada는 미국 국방성에서 embedded 컴퓨터 시스팀을 대상으로 증대하는 소프트웨어 개발 및 유지보수 비용을 줄이기 위해 'embedded 시스팀용 고수준 언어'로서 개발되었다. 그러므로 실시간 처리, 병렬 처리를 기본으로 하는 대형 embedded 소프트웨어의 개발에 적합하도록 package, 태스크, 예외상황 처리, 분리 컴파일 등의 기능을 가진 고급 언어이면서 또한 저수준 프로그래밍 특성도 지니고 있다.

다음에 열거한 사항은 Ada의 특징중 제어용으로 적합한 부분들이다.

가. 예외상황 처리

제어 시스팀에서 신뢰도는 매우 중요하다. 그러나 시스팀내의 하드웨어 고장이나 예상치 못한 데이터의 입력으로 설계시의 제어범위를 벗어나는 이상상황이 일어날 수 있다. 시스팀이 높은 신뢰도를 갖기 위해서는 설계자는 이러한 예외상황들의 발생을 미리 예측하고, 그 경우의 조치를 취할 수 있어야 하는데 이를 위해 Ada는 예외상황 처리기능을 제공한다.

예외상황의 선언은 변수를 선언하는것과 유사하며 사용자가 정의하는 예외상황 외에 Ada언어는 미리 정의된 5개의 예외상황을 가지고 있다. 이들 예외상황을 발생조건과 함께 나타내면 다음과 같다.

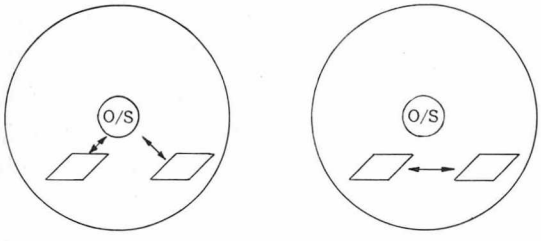
- CONSTRAINT ERROR : 변수나 array의 index 값이 제한범위를 벗어날때
- NUMERIC ERROR : 영(zero)으로 나누는 것과 같은 산술계산시 에러가 발생할때
- PROGRAM ERROR : 부프로그램을 부르거나 태스크를 activate시킬때 그 프로그램 선언부의 선언이 아직 유효하지 않는 (not elaborated)경우
- STORAGE ERROR : 한계를 넘어선 동적(dynamic)메모리 용량을 요구할 때
- TASKING ERROR : 태스크들의 랑데부시 잘못된 상황이 발생할때

대부분의 언어는 에러와 같은 예외상황이 발생한 경우 프로그램의 수행이 정지되면서 수행제어가 O. S로 넘어간다. 그러나 고신뢰도의 제어 시스팀에서는 이러한 상황이 발생 할지라도 프로그램 수행이 정지되어서는 안되며 그러한 상황에 대한 적절한 조치를 취할 수 있어야 한다. 따라서 Ada언어는 먼저 정의한 각 예외상황들에 대해 이를 다룰수 있는 예외상황 처리 루틴을 사용할 수 있게 한다.

나. 태스크(task)

대부분의 기존 고수준 프로그래밍언어는 병렬처리 문제를 해결하기 위해서 host 컴퓨터의 O.S에 의존하거나, 어셈블리언어를 사용해서 멀티태스킹 루틴을 작성한 뒤 혼합사용해야 했다.

Host 컴퓨터의 운영체제에 의존해서 소프트웨어를 작성할 경우 프로그램의 이식성이 극도로 저하됨은 물론, 멀티태스킹 프로그램을 작성하는 일 자체도 어려우며, 어셈블리 언어를 사용하여 서로 다른 태스크 끼리의 상호작용을 신뢰성 있게 제어한다는 것은 더욱 어렵다. 따라서 프로그램의 이식성과 프로그램 작성의 용이함을 위해서는 프로그래밍 언어 자체가 병렬처리 기능을 가지고 있는 경우가 가장 이상적이며, Ada 언어가 지니는 태스크 기능이 바로 그것이다. 〈그림 2〉은 일반적 병렬처리 모델과 Ada 태스크모델을 비교한 것이다.



The Traditional Model of Concurrency

The Ada Tasking Model

〈그림 2〉 병렬처리 방법

Ada에서 제공하는 태스크란 병렬처리될 수 있는 프로그램 단위를 말하며 이는 제어 상황에서 동시에 일어나는 사건들을 처리하는데 매우 유용하다.

다. 인터럽트

제어 시스팀에서 비주기적으로 수행되는 일(task)들은 주로 인터럽트에 의해서 수행이 요구된다. 인터럽트가 발생하면 미리 정의된 인터럽트 벡터에 의해 해당 루틴으로 프로그램의 수행이 넘어가게 된다. Ada는 저수준 그로그래밍 특성인 표현정의 기능을 제공하여 인터럽트 벡터의 위치선정을 임의로 가능케하며, 태스크의 entry문을 부름으로써 해당 인터럽트의 처리를 수행한다. 한 예를 〈예 1〉에 보였다.

〈예 1〉에서는 timer인터럽트에 대한 인터럽트처리 루틴을 하드웨어 메모리내의 16진수 1FF

```
task int_hdlr is
  entry time_int ;
  for timer_int use at 16#1FF# ;
end int_hdlr ;
task body int_hdlr is
begin
  accept timer_int do
  --interrupt service routine
  end accept ;
end int_hdlr ;
```

〈예 1〉 인터럽트 처리 루틴

번지에 위치시켰으며, 실제 timer_int가 발생하게 되면 태스크 int_hdlr내에 존재하는 entry timer_int를 불러 이를 수행함으로써 인터럽트를 처리한다.

라. 저수준 프로그래밍 특성

Embedded 컴퓨터의 소프트웨어는 하드웨어 시스팀에 종속된 부분을 다루어야 할 경우가 많이 발생한다. 즉, 입출력 port의 번지지정이나 메모리내의 데이터구조 표현 등이 그것이다. 과거에 고수준 언어를 사용해서 프로그램을 작성할때 이러한 문제의 해결을 위한 적합한 기능이 제공되지 않았기 때문에 고수준 언어와 어셈블리 언어를 복합 사용하였고 이는 결과적으로 문제 해결 및 프로그램의 이해와 유지보수를 어렵게 하였다.

Ada는 고수준 형태의 표현법을 사용하여 저수준 프로그래밍 특성을 다룰수 있는 표현정의 기능을 제공함으로써 이러한 문제들을 쉽게 해결할 수 있게 하였다. Embedded 시스팀 설계시 대부분의 경우가 하드웨어와 소프트웨어를 시기적으로 같이 설계해야되고 소프트웨어 설계시 하드웨어의 주변 장치와 인터페이스가 명확히 정의되지 않는다. 따라서 Ada 언어를 이용해서 소프트웨어를 설계할 때는 이러한 저수준 프로그래밍 특성을 고수준으로 추상화시켜 프로그램을 작성하고 하드웨어와 인터페이스가 명확하게 확정되면 표현을 정의하여 매핑시킬 수 있다.

Ada 언어는 length, enumerated type, record type, 주소 결정 등 4개의 표현기능을 제공한다. 그리고 시스팀에 종속적인 저수준 프로그래밍을 다루기위해 Ada는 시스팀에 종속적인 특성들을 모아놓은 package 'SYSTEM'을 가지고 있어 이들을 고수준에서 참조(reference)할 수 있도록 한다. 그에 따라 Ada는 부프로그램내에 기계어 코드를 포함시켜 어셈블리 프로그래밍을 가능케 한다. 다음의 예는 MC6800에 대한 기계어 삽입의 예를 보인것이다.

```
with MACHINE_CODE ;
procedure FSQRT(X        : in FLOAT ;
                RESULT : out FLOAT)is
--This procedure uses the MC68881 to
--calculate
--the square root of the parameter X.
begin
    CODE_2'(fmove_d, X'ref, fp0) ;
    CODE_1'(fsqret_d, fp0) ;
    CODE_2'(fmove_d, fp0, RESULT'ref) ;
end FSQRT ;
```

〈예 2〉 기계어 삽입

이와같이 'Built_In Assembler' 기능에 의해 Ada에서 하드웨어를 직접 access할 수 있고 time-critical한 부분에 대해 가장 이상적인 코드 생성이 가능하다. 결과적으로 어셈블리 언어의 효율을 고수준 언어인 Ada에서 얻을수 있게 된다.

4. Ada 프로그램 개발 환경

우수한 소프트웨어를 용이하게 개발하기 위해서는 우수한 특성을 지닌 프로그래밍 언어와 더불어 프로그램 전반에 걸쳐 이를 지원해주는 좋은 프로그래밍 환경이 필요하다.

미국방성은 이러한 사실을 고려하여 소프트웨어를 개발하는 전과정에 도움을 주는 Ada프로그래밍 지원 환경 APSE를 정의하였다. 이 Ada 프로그래밍 지원 환경은 프로그램개발 사이클 전반에 걸친 통합적인 개발 환경이며, 이는 프로그램 개발뿐만 아니라 소프트웨어의 유지 보수에도 매우 유용하다.

그러나 Ada 프로그램을 개발하는데 APSE가 꼭 필요한 것은 아니며, 컴파일러를 포함하여 기본적인 몇개의 tool만으로도 개발이 가능하다. 그렇지만 APSE하에서 작업할때 Ada 소프트웨어 시스팀의 개발 및 유지 보수는 더욱 효율적이다. 이는 APSE자체가 Ada라는 특정 언어를 지정하여 이를 지원하는 환경으로서 개발되었기 때문이다.

가. APSE의 구조

APSE는 host target 개념과 데이터베이스 개념, 확장성 등을 고려하여 일관성있게 설계되었다. 개발자는 이 APSE를 지닌 host 컴퓨터에서 그 자체나 다른 컴퓨터를 대상(target)으로 소프트웨어 시스팀을 개발하게 된다.

APSE는 Ada 언어로 작성되며 APSE tool간의 프로그램 이동은 DIANA(Descriptive Intermediate Attributed Notation for Ada)라 불리는 중간언어 코드를 사용한다. DIANA는 Ada 컴파일러의 front_end의 출력으로 source 프로그램에 대한 의미정보와 함께 위치정보를 포함하므로 원시 프로그램으로의 환원이 가능하고, 시스팀과 무관하므로 프로그램 이동시 유용하게 사용된다.

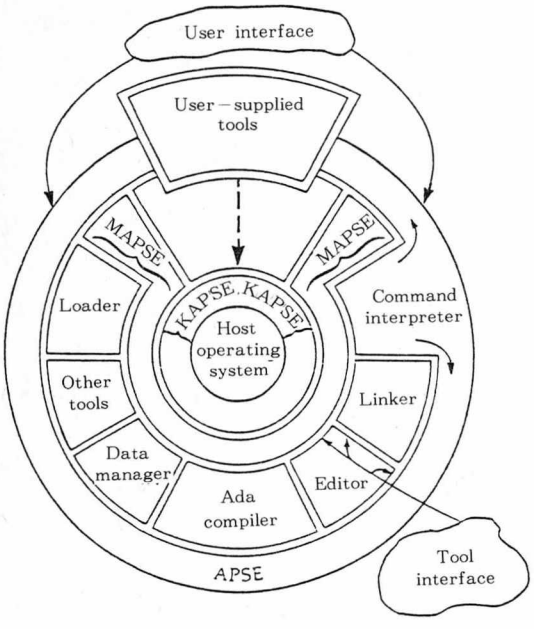
프로그래밍 환경의 가장 내부는 host 컴퓨터의 O.S가 차지하고 그 외부를 APSE의 핵심인 KAPSE(Kernel APSE)가 감싸고 있다. KAPSE는 APSE와 O.S간 인터페이스를 제공한다. 그리고 KAPSE만이 시스팀에 종속적이기 때문에 시스팀 이동시 KAPSE만을 다시 작성하면 나머지 APSE tool들을 변경없이 이식이 가능하다.

KAPSE의 외부는 개발과정에 사용되는 tool들로서 APSE라 불리며 다음의 기능을 제공한다.

- command processor : 사용자의 명령을 분석하는 기능

- text operation tools : text의 작성, 수정, formatting 등을 위한 기능
- Ada language processing tools : compiling, assembling, linking 등 프로그램 처리를 위한 기능
- program analysis tools : 프로그램 분석을 위한 기능
- file/database oriented tools : 파일/데이터의 관리 및 비교 검색 등의 기능

이들은 도식적으로 표현하면 〈그림 3〉과 같다.



〈그림 3〉 APSE의 구조

Ada 컴파일러는 Ada 언어가 제공하는 프로그램 단위들(subprogram, package, task)에 대해 개별적으로 컴파일한뒤 library에 저장하고 나중에 컴파일되는 프로그램 단위들이 이를 참조하고 사용할 수 있도록 하는 분리 컴파일 기능을 제공하는데, APSE는 이러한 개념을 확장시켜 소스 및 오브젝트 코드뿐만 아니라 개발에 관련된 모든 정보를 데이터베이스로 구성하여 개발 사이클 전반에 걸쳐 이를 참조할 수 있도록 한다. 이 데이터베이스를 이용함으로써 소프트웨어 개발과정 전반에 걸쳐 프로그램과 문서를 쉽게 관리할 수 있다.

나. APSE의 이용

소프트웨어 라이프 사이클은 대략 요구 분석(requirement analysis), 설계(design), 코딩(coding), 테스트, 설치(installation), 유지 보수의 6단계로 나눌 수 있다.

전통적인 많은 사례를 보면 개발의 각 단계마다 그에 적합한 tool들과 표현법을 사용하였다. 즉, 설계는 pseudo-code를 사용하고, 코딩은 그와 다른 프로그래밍 언어로 하는 경우가 많았으나 이러한 방법은 작업을 중복되게 하거나 모듈들을 연결하는데 문제점을 드러내기도 한다. 가장 이상적인 방법은 소프트웨어의 라이프 사이클 전반을 지원하는 일관된 도구들과 표현법을 쓰는 것이며 Ada와 Ada 프로그래밍 지원 환경이 이를 가능하게 한다. 이는 Ada가 프로그램 설계 언어(PDL : Program Design Language)로 사용할 수 있는 고수준 언어이기 때문이다.

III. 결 론

Ada 프로그래밍 언어는 처음부터 'Embedded 시스템용 고수준 언어' 라는 목표아래, 이를 위한 요구사항을 명확히 정의한 후 설계되었고, 검토 및 수정과정이 학계와 연구소 및 관련기관의 협조하에 이루어졌다.

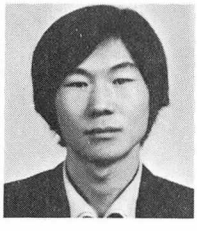
미국방성이 Ada를 이용해 개발하려는 분야는 주로 군용의 대규모 embedded 소프트웨어 시스템이며, 이러한 시스템은 실시간 처리와 병렬처리를 기본으로 하면서 유지보수 문제를 신중히 고려하여야 한다. 그러므로 Ada는 이러한 관점을 고려하여 설계되었으며 소프트웨어 공학 개념을 잘 반영한 고급 범용 언어로 성장하였다.

Ada는 아직 배우기 어렵고 코드 크기, 수행 속도 등 몇가지 문제점을 지니고 있어 그 성공 여부는 좀더 지켜봐야 하지만, Ada의 우수한 프로그래밍 언어 특성과 이를 지원하는 APSE외에도 미국방성이 정책적으로 지원하기 때문에

상당히 빠른 속도로 성장 보급되고 있다.

〈參 考 文 獻〉

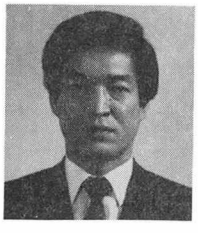
1. Reference Manual for the Ada Programming Language(ANSI/MIL-STD-1815 A), DOD, Jan. 1983.
2. STEELMAN : Requirement for High Order Computer Programming Languages, DOD, 1978.
3. "VADS Reference Manual," VERDIX Co, Jun. 1988.
4. John Voelcker, "Ada : From Promise to Practice ?," IEEE Spectrum, Apr. 1987.
5. Kjell W. Nielsen & Ken Shumate, "Designing Large Real-Time System With Ada," Communication of the ACM, Vol. 30, No. 8, Aug. 1987.
6. Howard Falk, "Developers Target Unix and Ada with Real-Time Kernels," Computer Design, Apr. 1988
7. Howard Falk, "Development Systems Evolve Toward Integrated, Host-independent Solutions," Computer Design, Apr. 1988.
8. Ron Wilson, "Embedded Systems Manipulate Distributed Tasks," Computer Design, Sep. 1987.
9. L. R. Collingbourne,"A Practical Approach to Developing Real-Time Ada Programs for Embedded System," ACM, Mar, '1987.
10. Greg Burns, "Cross-Debugging Real-Time Ada Programs," ACM, Mar., 1987.
11. 김용진외 2, Embedded System용 Ada언어 관련 S/W 연구, 과학기술처 최종연구보고서, 1987. 8.
12. 양광호외 1, Embedded System을 위한 Ada 프로그래밍환경 연구, 과학기술처 최종 연구보고서, 1988. 8



李 澤 熙 (Lee, Teag Heui)

1958년 4월 15일생

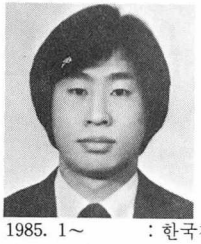
1981. 2 : 전남대학교 제어계측공학과 학사
1983. 2 : 서울대학교 대학원 제어계측공학과 석사
1984. 3~ : 한국전자통신연구소
1988. 12현재 : 응용 S/W개발실 연구원



金 容 準 (Kim, Yong June)

1947년 5월 19일생

1971. 2 : 서강대학교 물리학과 학사
1974. 1~1978. 3 : 한국과학기술연구소 전자계산실
1978. 4~ : 한국전자통신연구소
1988. 12현재 : 응용 S/W개발실 실장



宋 慶 俊 (Song, Kyung Joon)

1956년 1월 7일생

1982. 2 : 명지대학교 전자공학과 학사
1984. 2 : 명지대학교 대학원 전자공학과 석사
1982. 3~1985. 2 : 명지대학교 전자공학과 실험조교
1985. 1~ : 한국전자통신연구소
1988. 12현재 : 응용 S/W개발실 연구원