Nano-Q+에서 스마트 센서 디바이스 관리 시스템

(A Smart Sensor Device Management System in Nano-Q+)

김 범 석 † 소 선 섭 †† 김 병 호 ††† 은 성 배 ††††
(Bumsuk Kim) (Sunsup So) (Byeongho Kim) (Seongbae Eun)

요 약 센서노드 운영체제는 다종다양한 센서를 효율적으로 관리하기 위해서 통일된 API와 효율적인 디바이스 드라이버 매니저를 지원하여야 한다. 하지만 Tiny-OS, Nano-Q+ 등의 기존 운영체제들은 이와 같은 디바이스 드라이버 매니저를 지원하지 않는다.

본 논문에서는 센서 I/O 서브시스템을 제안하여 응용프로그래머에게 통일된 API를 제공하며 디바이스 드라이버의 장탈착이 용이한 디바이스 관리 매니저를 제시한다. 탈부착이 가능한 스마트 센서를 위하여 원격 디바이스 드라이버 업데이트 방식을 제안한다. 이 방식은 일부 센서가 변경되었을 때 전체 응용이 아닌 디바이스 드라이버만의 다운로딩이 가능하다.

ETRI가 개발한 Nano-Q+에 상기한 기능을 추가하여 설계하고 구현하였다. 기존 운영체제와 성능을 비교 평가하였고 디바이스 드라이버 부분 다운로딩이 다운로딩 속도를 획기적으로 개선시켰다.

키워드 : 센서노드 OS, 스마트 센서, 디바이스 드라이버, 디바이스 관리 시스템, Nano-Q+, 표준화

Abstract Sensor Node OS should support unified API and efficient sensor device management system to overcome the diversity of sensors and actuators. However, conventional OSs like Tiny-OS and Nano-Q+ do not.

In this paper, we propose a sensor device driver management system that present application programmers with unified API and easy deployment of sensors. When a sensor is deployed in our device management system, the device driver is downloaded. This scheme differs from traditional OS like SOS in that only sensor device driver is downloaded, not the whole application image.

We designed and implemented the system into Nano-Q+. We described the comparison with other OSs and showed that our system obtains the considerable speedup of downloading.

Key words : Sensor Node OS, Smart Sensor, Device Driver, Device Management System, Nano-Q+, Standardization

1. 서 론

유비쿼터스 센서 네트워크(USN)는 물리 공간에 빛, 소리, 온도, 움직임 같은 물리적 데이타를 센서 노드로 감지하고 측정하여 무선 네트워크를 통해 사용자에게 전달하는 시스템이다. 센서노드의 핵심 요소로는 센서, MCU, RF 통신 모듈, 전력부를 들 수 있다.

USN에서는 여러 가지 센서들이 활용되는데 크게 2가지의 문제점을 내포하고 있다. 그 중 첫 번째는 센서의 종류가 매우 많고 특성이 서로 다르다는 것이고, 두 번째는 센서가 센싱환경에 노출되어야만 한다는 점이다.

먼저 센서는 센싱값이 변환되어야 할 성질에 따라, 변환될 값의 범위에 따라, 사용되는 환경에 따라 매우 다양하다. 예를 들어, 온도 센서는, 측정값의 범위에 따라, 상온에서 동작하는 반도체 온도 센서, 수온을 재는 막대

† 학생회원 : 한남대학교 정보통신공학과
eecorea@naver.com
†† 정 회 원 : 공주대학교 컴퓨터공학부 교수
triples@kongju.ac.kr
††† 정 회 원 : 경성대학교 컴퓨터공학부 교수
bkim@ksu.ac.kr
†††† 종신회원 : 한남대학교 정보통신공학과 교수
sbeun@hnu.kr
논문접수 : 2007년 9월 6일
심사완료 : 2007년 12월 26일

정보과학회논문지 : 컴퓨팅의 실제 및 레터 제14권 제1호(2008.2)

형 온도 센서, 1,000도 이상의 매우 뜨거운 온도를 재는 온도 센서 등으로 다양하다. 센서의 출력도 전압, 전류, 주파수 등으로 다양하며 증폭이 필요한 센서, ADC에 바로 붙일 수 있는 센서 등, 다양하다.

또한 센서는 센싱 대상 속에 장치되어야 하는데 예를 들어 수온을 측정하는 센서 노드는 센서 자체가 물속에 설치되어야 한다. 하지만 MCU, RF 통신모듈, 전력부는 방수케이스에 넣어 보호해야 한다. 따라서 센서를 센서 노드에서 가능한 분리하여 설계, 구현하여야 한다.

센서를 분리시키면 외부의 극한 환경에 노출된 센서가 먼지, 습기 등에 고장날 때 센서 부분만 교체해 주어도 된다. 이는 센서 노드 전체를 교체해 주어야 하는 것보다 편리할 뿐만 아니라 원가 절감과 기존의 센서 네트워크와의 적응도 빨라지게 된다.

센서를 MCU와 RF 통신 모듈과 분리시키는 방식은 새로운 센서가 발견되었을 때 이를 인지하고 알맞은 디바이스 드라이버를 찾아 사용할 준비가 되도록 해야 한다. 우리는 이를 탈·부착이 가능한 스마트 센서라 부른다.

센서노드 운영체제는 센서의 2가지 문제점을 해결할 수 있어야 한다. 첫째로 다양한 센서들 때문에 하드웨어 개발 시엔 센서 인터페이스를 설계하기가 어려우며 센서노드 운영체제에선 응용프로그래머에게 통일된 API를 제공하지 못한다. 기존의 센서노드 운영체제로 잘 알려진 Tiny-OS[1], Nano-Q+[2] 등에서도 센서 디바이스를 위한 통일된 API 체계를 갖지 못하고 있다. 센서마다 별도의 API가 만들어지며 응용프로그래머는 센서마다 서로 다른 API를 사용해야 한다.

둘째로 탈부착이 가능한 스마트 센서를 위해서 센서노드 운영체제는 센서 디바이스 드라이버를 별도로 관리할 수 있어야 한다. 즉, 센서 디바이스가 설치되었을 때 중앙의 서버에서 적절한 디바이스 드라이버만을 다운로딩할 수 있어야 한다. 센서 노드는 디바이스 드라이버 부분만을 다운로딩 받아 기존 응용프로그램과 동적으로 연결하여 효율성을 높여야 한다. 기존의 센서노드 운영체제들은 이러한 기능을 지원하지 못하고 있는데 SOS[3], Nano-Q+[2], SENSOS[4] 등이 무선으로 응용프로그램을 다운로딩하는 기능을 지원하지만 디바이스 드라이버 만의 무선 다운로딩은 지원하지 않는다.

본 논문에서는 센서의 2가지 문제점을 해결하는 기법을 제안한다. 센서 I/O 서브시스템을 제안하여 응용프로그래머에게 통일된 API를 제공하며 디바이스 드라이버의 장탈착이 용이한 디바이스 관리 매니저를 제시한다. 탈부착이 가능한 스마트 센서를 위하여 원격 디바이스 드라이버 업데이트 방식을 제안한다. 이 방식은 일부 센서가 변경되었을 때 전체 응용이 아닌 디바이스 드라이버만의 다운로딩이 가능하여 다운로딩 속도를 획기적으로 개선시켰다. Nano-Q+에 상기한 기능을 추가하여 설계하고 구현한다.

2. 배 경

2.1 리눅스 디바이스 드라이버 체계

리눅스는 다양한 하드웨어를 응용 프로그램에서 사용할 수 있도록 디바이스 드라이버 관리체계[5]를 갖고 있다. 리눅스의 디바이스 드라이버 관리체계는 4가지 관리 기법으로 사용자에게 투명한 관리와 접근 기법을 제공한다.

첫째, 데이타 추상화이다. 리눅스에서 사용되는 디바이스 드라이버는 크게 Character device, Block device, Network device로 나뉜다. 이 기본적인 구조에서 각각의 하드웨어에 맞게 디바이스 드라이버의 구조가 다양하게 변형된다.

둘째, 각 추상화에 대한 사용자 API[6]이다. 응용 프로그램은 open(), close(), read(), write()와 같은 파일 처리용 함수를 이용하여 디바이스 파일을 일반 파일처럼 다뤄서 하드웨어를 처리할 수 있다.

셋째, 각 디바이스에 대한 관리기법이다. 디바이스 드라이버를 디바이스 제작자가 운영체제 내에 동적으로 삽입할 수 있는 체계를 지원하다.

넷째, 디바이스 이름의 정의기법이다. 사용자는 디바이스들을 "dev/tty0"등의 이름으로 접근하고 이것이 파일 시스템의 디바이스 메이저 번호로 연결된다. 디바이스 드라이버는 메이저 번호로 등록하는데 이를 파일 시스템이 연결하는 기능을 갖는다.

리눅스의 디바이스 드라이버 관리 기법은 센서노드 운영체제의 디바이스 드라이버에 대한 통일된 관리 기법의 모태가 된다. 센서노드 운영체제는 리눅스보다 훨씬 간단하므로 리눅스의 구조와 같지 않지만 그 원리는 매우 비슷하다.

2.2 기존 센서노드 운영체제

센서노드는 저전력 소모가 가장 중요한 이슈 중의 하나이므로 대부분이 8비트 MCU기반의 1칩 솔루션을 사용한다. ATmega128[7]의 경우, 데이타 저장을 위한 4KB의 SRAM과 코드 저장을 위한 128KB의 플래시 메모리를 가지고 있다. 데이타 저장 메모리가 4KB뿐이므로 센서노드 운영체제도 이에 맞도록 설계 및 구현한다.

2.2.1 Tiny-OS

기존 센서노드 운영체제로서 가장 먼저 개발된 Tiny-OS[1,8,9]는 이벤트 발생 중심의 상태 변화 방식을 채택한 센서 네트워크용 운영체제로써, 동시적인 프로세싱 및 제한된 하드웨어 메모리 공간에서의 효율적인 성능을 지원해주는 운영체제이다.

Tiny-OS의 센서 디바이스는 컴포넌트 기반 구조이

고 이벤트 기반 멀티태스킹을 지원하며 NesC라는 프로그래밍 언어를 사용한 점이 특징이다. 그러나 센서의 다양성과 복잡성을 고려한 특징은 없다.

2.2.2 SOS

SOS[3]는 메시지 패싱, 동적 메모리 할당, 모듈의 자율적인 적재와 제거를 지원하는 공동 커널(common kernel)을 지원하며, 동적 재구성이 특징이다. 이것은 새로운 모듈 업데이트가 필요할 때 무선 네트워크를 통하여 동적으로 변경할 수 있는 구조를 제공한다. 그러나 센서의 통합된 관리 기법을 제공하지 않는다.

2.2.3 MANTIS

MANTIS[10]는 레이어 기반의 운영체제이며, 하드웨어를 추상화시키는 디바이스 드라이버의 특징을 가진다. 그러나 이미 디바이스의 순서가 커널에 고정되어 있어 새로운 디바이스의 추가가 쉽지 않다는 단점이 있다.

2.2.4 Nano-Q+

Nano-Q+[2] 운영체제는 한국전자통신연구원(ETRI)에서 개발된 센서 노드나 센서 네트워크를 위한 초소형 운영제제로써 다음과 같은 특징을 가진다.

에너지 소모를 최소화하고자 저전력 슬립모드를 제공하고, 제한된 메모리 사용을 최소화하도록 멀티스레드 간의 스택을 공유한다. C 기반의 프로그램을 지원하며 멀티스레드 스케줄러 방식 등의 특징을 가진다. Nano- HAL이라는 디바이스 드라이버 영역이 있어 하드웨어를 추상화하지만 사용자는 API를 다 알아야만 하는 단점이 있다.

본 논문에서는 잘 정의된 API들과 통일된 디바이스 관리 기법을 제공함으로써 기존 센서노드 운영체제의 문제점을 해결하려 한다. 센서 디바이스 관리 기법은 많은 디바이스에 상관없이 프로그래머에게 통일된 API를 제공한다. 이러한 API를 이용하여 센서 디바이스 드라이버 제작자는 손쉽게 드라이버를 제작할 수 있다. 게다가 동적 디바이스 드라이버 추가방식을 채택하여 센서 디바이스 드라이버의 장탈착이 용이하다.

2.3 IEEE1451

IEEE1451[11]은 미국의 NIST[12]에서 주도하고 있는 표준으로서 1451.0~1451.6, 총 7개의 표준으로 나뉘어 진행 중이다. 이 표준은 트랜스듀서의 다양성을 극복하는데 초점을 맞추고 있다.

센서나 트랜스듀서 제조사들은 TIM(Transducer Interface Module)이라는 부분까지 표준에 맞춰 개발하고, 네트워크 연결 모듈 개발자들은 NCAP(Network Capable Application Processor)부분을 담당하면 표준에 정의한 인터페이스 및 트랜스듀서의 특성을 기술하는 TEDS(Transducer Electronic Data Sheet)에 의해 트랜스듀서의 투명성을 지원받는다. NCAP은 1451.0와 1451.1과 관련 있는데 1451.0[13]는 TIM에 연결된 물리

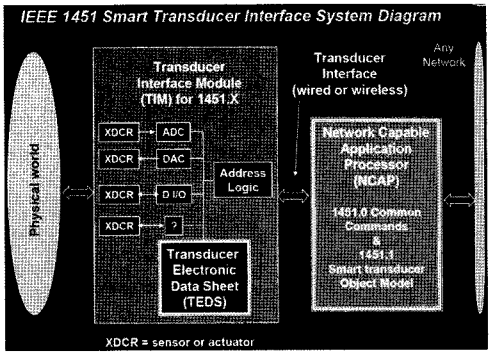


그림 1 IEEE 1451 스마트 트랜스듀서 구조

적인 트랜스듀서들의 특성을 파악할 수 있는 방법을 제공하며 1451.1[14]은 네트워크를 통하여 트랜스듀서들을 발견하고 접근할 수 있는 객체지향적인 접근 표준을 제공한다.

IEEE1451에서 정의된 채널 TEDS는 크게 4부분으로 나누어지는데 데이타 구조와 관련된 정보, 물리적 특성에 관련된 정보, 데이타 해석 시에 요구되는 정보, 타이밍 관련된 정보를 제공한다.

IEEE1451에서 정의하는 트랜스듀서와 NCAP과의 인터페이스는 센서노드 플랫폼을 개발할 때 많이 참고할 수 있으나 1451표준에서는 전원이 별도로 공급되고 저전력을 위한 제어도 어려우므로 1451 표준이 센서노드 플랫폼을 대체한다고 볼 수 없다.

3. 설 계

3.1 센서 I/O 서브시스템

센서 I/O 서브시스템은 센서 디바이스의 특징이나 세부 정보를 커널이나 응용프로그래머에게 숨기고 응용프로그램 개발자에게 센서 I/O장치에 대한 통일된 접근 방식을 제공해주는 것이다. 센서 I/O 서브시스템은 다음과 같은 구조를 가진다.

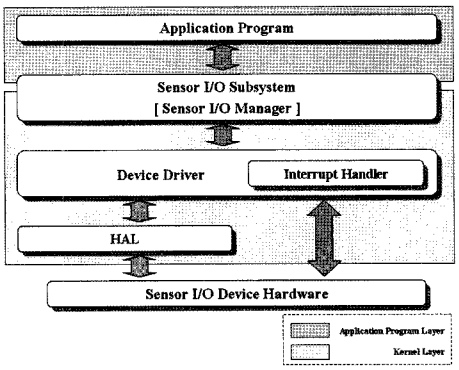


그림 2 센서 I/O 서브시스템

센서 I/O 디바이스는 온도, 습도, 조도, 움직임 센서등의 각종 센서 및 구동기를 포함한다. 센서 I/O 서브시스템은 응용프로그램과 디바이스 드라이버를 연결하는 역할을 담당하여 센서 I/O 디바이스를 제어하기 위한 통일된 I/O API를 제공한다. 디바이스 드라이버에서 하드웨어의 제어가 가능하고 HAL(Hardware Abstraction Layer)에 의해 센서 I/O 디바이스 하드웨어가 추상화된다. 센서 I/O 서브시스템은 다음의 기능을 갖는다.

① 응용프로그램을 위한 API를 정의한다.
② API의 각 함수를 처리한다.
③ 디바이스 드라이버를 센서 I/O 서브시스템에 등록하는 기능을 제공한다.
④ 응용프로그램이 특정 센서 디바이스 I/O를 요구할 때 그 디바이스의 드라이버를 호출한다.

센서 I/O 서브시스템은 디바이스 드라이버의 선택적 등록이 가능하다. 이는 응용에 필요한 함수들만 등록하여 모든 디바이스 드라이버가 메모리에 로드될 필요가 없으며, 등록할 때 드라이버의 순서가 정해지므로 드라이버 등록의 유연성을 보장할 수 있다. 그러므로 새로운 센서 디바이스 드라이버가 I/O 서브시스템에 변경 없이 추가되어 센서 I/O 하드웨어에 쉽게 접근 가능하다.

3.2 센서 I/O 함수 API

센서 I/O 서브시스템에서는 추상화에 대한 API를 정의한다. 리눅스 드라이버의 추상화 방법을 응용하지만 센서 I/O 서브시스템에서는 5개만 사용한다.

표 1 I/O 서브시스템 API 셋

함수	설 명
Open	센서 디바이스 전원을 인가하여 사용을 준비한다
Close	센서 디바이스 전원을 차단하여 사용을 중지한다
Read	센서 디바이스로부터 데이타를 읽어들인다
Write	구동기에 데이타를 보낸다
Iotcl	디바이스를 제어하는 명령어를 전달한다

3.3 정의된 API의 함수 처리

API로 정의된 함수는 디바이스 드라이버에서 함수로 구현해야 한다. 각 센서 디바이스 제작자는 센서 I/O 함수 API 형식에 맞추어 각 함수를 구현하면 된다.

3.4 디바이스 드라이버 등록

응용프로그램은 초기화 때 사용할 센서의 디바이스 이름과 디바이스 타입을 등록한다. 디바이스 이름은 드라이버에서 제공하는 함수의 이름을 등록하고 디바이스 타입은 표 2와 같이 구분하여 사용한다. 이는 응용프로그램에서 디바이스 타입을 구분하기 위함이다.

그림 3과 같이 insert_TD()에 의해 디바이스를 등록하면 TDT(Transducer Device Table)에 등록되며 등록한 순서에 맞춰 인덱스가 부여된다. 응용프로그램이

표 2 디바이스 타입

등록	설명
0	센서
1	이벤트 센서
2	구동기

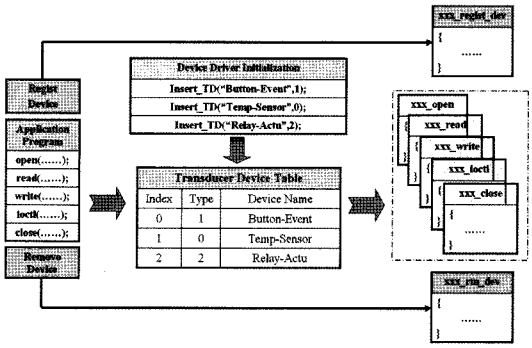


그림 3 센서 디바이스 매니저 구조

디바이스를 open()하면 TDT의 인덱스를 리턴하고 함수 호출시 이 인덱스를 활용한다.

3.5 I/O함수 연동

등록된 디바이스 정보는 실제 구동함수와 연동 되어야 한다. 이는 각 디바이스에서 제공하는 등록함수로 가능하며 driver_regist_drv()함수로 정의 된다. insert_TD()로 디바이스의 동록을 한 뒤 driver_regist_drv()함수로 연동해 주어야만 사용이 가능하다. 결국 I/O 서브시스템 API를 호출하여 실제 디바이스를 제어한다.

3.6 디바이스 드라이버 부분 다운로딩 기능

부트로더는 플래시 메모리에 상주하며, 전원이 켜지거나, 리셋이 됐을 때, 응용의 요구에 의해 수행된다. 부트로더가 탈·부착이 가능한 스마트 센서의 기능을 지원하면 새로운 디바이스의 발견이나 디바이스 드라이버의 업데이트가 있을 때 적절한 디바이스 드라이버를 서버로부터 다운받을 수 있다.

원격 무선 업데이트 기능은 응용 프로그램의 요구에 따라 부트로더가 실행되고 서버로부터 응용 프로그램을 다운로딩하여 애플리케이션 섹션을 업데이트 하는 것이다. 그러나 응용 프로그램 전체를 다운받는 것보다는 디바이스 드라이버 부분만을 다운받는 것이 효율적이다. 그러기 위해서는 디바이스 부분은 응용프로그램과 분리되어 플래시 메모리에 저장되어야 한다.

그림 4는 응용프로그램과 디바이스 드라이버 부분이 분리된 예를 나타내고 있다. 각 디바이스 드라이버는 정해진 주소에 저장되며 응용프로그램은 정해진 주소를 참조하여 디바이스 드라이버를 사용하게 된다.

앞서 언급하였듯이 디바이스 드라이버가 등록되면 실

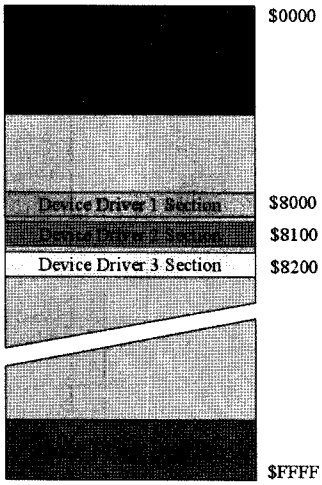


그림 4 디바이스 드라이버 부분 다운로드

제 구동함수의 시작 포인터가 TDT에 저장되는데 구동함수의 시작포인터가 드라이버의 다운로딩된 실제 플래시 메모리 주소를 지정하면 된다. 각 함수의 주소는 디바이스 드라이버별로 컴파일 된 심볼테이블을 참조하여 계산한다. 센서 I/O 함수 API의 실제 주소 값은 다음과 같다.

$$A_n = D_s + S_r \quad (1)$$

A_n : I/O 함수 API의 실제 주소값

D_s : 디바이스 드라이버의 저장 번지

S_r : 심볼테이블 참조값

예를 들어, 디바이스 드라이버 1은 8,000h번지부터 시작되고 심볼테이블에서 xxx_open()은 10h번지부터 참조된다면 실제 디바이스 드라이버 1의 open()은 8,010h번지가 되는 것이다.

그림 4에서 드라이버의 저장 위치가 100h의 일정한 간격을 갖는 것은 아니며 드라이버의 크기에 따라 달라질 것이다.

3.7 다운로드 파일 프로토콜

디바이스 드라이버 부분 다운로딩을 위해서 부트로더는 다운로드 파일에 대한 정보를 알아야 한다. 그림 5는 다운로드 파일 프로토콜을 보여준다.

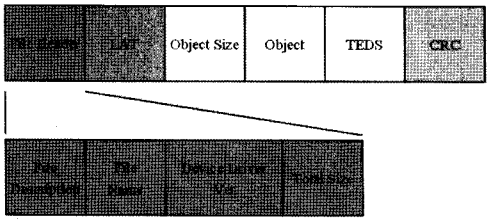


그림 5 다운로드 파일 프로토콜

File Header : 파일의 구분, 파일의 이름, 디바이스 드라이버의 버전, 총 파일 크기를 포함한다.

LAT(Link Address Table) : 드라이버의 각 함수별로 심볼테이블을 참조한 값의 테이블이다.

Object Size : RWW 섹션에 쓰여질 디바이스 드라이버 부분만의 크기를 나타낸다.

Object : 실제 디바이스 드라이버 파일이다. 이진파일 형태로 제공된다.

TEDS : 디바이스의 상세 관련 정보등을 포함한다. IEEE1451 표준을 따르며 센서 값의 데이타 구조나 물리적 특징등을 기록한다. 응용프로그램은ioctl() 함수를 호출하여 TEDS 정보를 가져간다.

CRC : 정확한 다운로딩이 매우 중요하므로 CRC체크를 한다.

4. 구현 및 성능평가

4.1 Nano-Q+에서의 구현

본 연구는 한국전자통신연구원(ETRI)에서 개발한 센서노드나 센서네트워크 운영체제인 Nano-Q+를 기반으로 진행하였다. Nano-Q+는 ATmega128L을 사용하는 MCU부분과 무선통신에서의 CC2420을 사용한 RF모듈, 센서와 구동기를 지원한다. 쓰레드 기반 스케줄러를 지원하며 RF 메시지 핸들링을 지원하는 네트워크 프로토콜 스택을 지원한다. 응용부분은 시스템 API를 통해 운영체제부분과 상호작용하는 방법을 제공한다.

3장에서 기술한 디바이스 드라이버 매니저와 디바이스 드라이버 부분 다운로딩을 위한 부트로더를 Nano-Q+에 적용하였다.

그림 6은 ZigBee를 지원하는 하드웨어 모듈이다. ATmega128L MCU와 CC2420통신 모듈을 사용하며 트랜스듀서들은 커넥터에 부착되는 구조이다. C 언어를 사용하여 구현하였으며 컴파일러는 avr-gcc를 사용하였다.

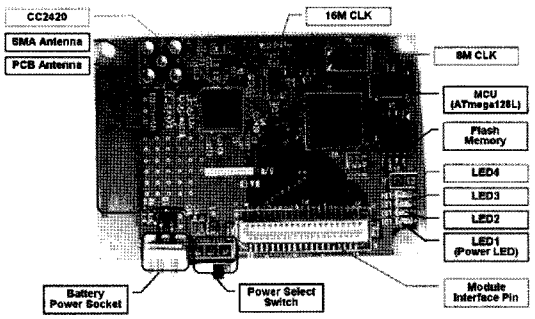


그림 6 ZigBee 지원 하드웨어 모듈

- CPU Clock : 8MHz

- Flash : 128Kbyte
- RAM : 4Kbyte
- Radio : 250kbps
- Radio 형태 : ZigBee
- OS : Nano-Q+
- Compiler : avr-gcc

4.2 API 구현

센서 I/O 함수 셋을 총 5가지로 정의하였다. 정의된 함수의 표준 C 구조체는 다음과 같다. 응용프로그램에서는 이 함수를 이용하여 디바이스 드라이버에 접근한다.

```
typedef struct {
  INT8U (*Open)(void);
  void (*Close)(void);
  INT8U (*Read)(void* pdata, INT8U size);
  INT8U (*Write)(void* pdata, INT8U size);
  INT8U (*Ioctl)(void* pdata, INT8U size);
} UNIFORM_IO_DRV;
```

API의 각 함수 처리는 디바이스 제작자가 API 형식에 맞춰 디바이스 드라이버를 구현하는 것이다. 각 함수의 구현은 driver_Open(), driver_Close(), driver_Read() 형태로 구현한다. 이는 디바이스 드라이버의 이름과 같게 하여 디바이스의 등록과 함수의 연결을 편리하게 하기 위함이다. 다음은 Light_Sensor의 Open, Close, Read가 구현된 예를 보여준다.

```
INT8U Light_Sensor_Open(void)
{
  ADC_LIGHT_POWER_PORT_INIT();
  /* ADC사용을 위해 전원 이용을 초기화한다 */
  LIGHT_SENSOR_POWER_ON();
  /* Light_Sensor에 전원을 인가한다 */
  PutString("Light Sensor Open\r\n");
  return 0;
}

void Light_Sensor_Close(void)
{
  LIGHT_SENSOR_POWER_OFF();
  /* Light_Sensor에 전원을 차단한다 */
  PutString("Light Sensor Close\r\n");
  return 0;
}

INT8U Light_Sensor_Read(Read_Data *pData,
INT8U size)
{
  INT16U light_raw_data;
  SensorInit (LIGHT_SENSOR);
  light_raw_data =
      ADC_RAW_DATA(LIGHT_SENSOR);
  /* 센서 값을 읽는다 */
  memcpy(&pData->data,&light_raw_data,
      sizeof(INT16U));
  /* 센서값을 넘겨준다 */
  return sizeof(light_raw_data);
}
```

동적 디바이스 드라이버의 등록은 응용프로그램이 초기화 때 이루어진다. insert_TD()에 의해 디바이스의 이름과 디바이스 타입을 TDT에 저장하고 등록되는 순서에 따라 인덱스를 부여받는다. 디바이스 이름은 드라이버에서 제공하는 함수의 이름을 등록하는데 조도센서의 경우 Light_Sensor임을 알 수 있다.

그러므로 조도센서의 등록은 다음과 같다.

```
insert_TD("Light_Sensor",0);
```

insert_TD()의 원형은 다음과 같고 등록이 되면 디바이스마다 3개의 정보를 갖는다. 인덱스와 디바이스 종류, 디바이스 이름이 TDT에 저장된다.

```
void insert_TD(char * device_name, INT8U type)
{
  pTDT[index].index = index;
  pTDT[index].type = type;
  pTDT[index].device_name = device_name;
  index++;
}
```

응용프로그램이 디바이스 I/O를 요구할 때 디바이스 관련 센서 I/O와 연동시킨다. 각 디바이스는 부여된 인덱스에 심볼테이블을 참조하여 올바른 함수의 포인터를 가리키도록 정의한다.

4.3 디바이스 드라이버 부분 다운로딩 구현

디바이스 드라이버 부분 다운로딩을 지원하기 위해 부트로더에 무선 업데이트 기능을 포함한다. 다운로딩은 부트로더에서 실행되며 동작중인 응용프로그램에서 부트로더를 실행하도록 섹션을 옮긴다.

응용프로그램에서 부트로더로, 부트로더에서 응용프로그램으로 이동하기 위한 매크로는 다음과 같다.

```
#define ENABLE_BOOT_INT() do { \
```

```
    MCUCR = 1 << IVCE; \
    MCUCR = 1 << IVSEL; \
    } while (0)
#define ENABLE_APP_INT() do { \
    MCUCR = 1 << IVCE; \
    MCUCR = 0 << IVSEL; \
    } while (0)

#define GOTO_APP_SECTION()
    do { asm("jmp 0"); } while (0)
#define GOTO_BOOT_SECTION()
    do { asm("jmp 0x1e000"); } while (0)
```

응용프로그램의 요구에 의해 부트로더가 실행되면 부트로더는 응용프로그램에서 사용한 정보들이 있어야 한다. 예를 들어, 무선 네트워크 채널, 노드번호, 라우팅 경로 등은 부트로더 섹션에서도 무선 네트워크를 위해 필요한 정보들이다. 응용프로그램이 사용한 정보를 EEPROM을 사용하여 부트로더로 전달해 준다.

센서 네트워크에 많이 사용되는 마이크로컨트롤러인 ATmega128은 부트로더 섹션으로 사용할 수 있는 메모리의 최대 크기가 4KB이다. 그러므로 다양한 기능을 지원하는 복잡한 무선 네트워크 프로토콜을 사용할 수 없다. 본 논문에서는 Chipcon사에서 제공하는 Simple MAC을 사용하는데 최소한의 무선 네트워크를 지원하고 있다.

디바이스 드라이버 부분 다운로딩을 위한 프로토콜 정보와 플래시 메모리 제어, EEPROM제어, CRC 처리 등이 부트로더에 포함하고 있다.

그림 7에서 디바이스 드라이버와 다운로딩 대상 노드를 선택하고 start 버튼을 클릭하여 다운로드를 시작한다. 상태 바가 증가하면서 다운로딩 되고 있음을 확인할 수 있다.

그림 8은 디바이스 드라이버 부분 다운로딩을 마친 후 응용 프로그램이 정상적으로 동작함을 보여주고 있다. 온도 센서와 조도 센서가 디바이스 드라이버 체계에 의해 정상 동작함을 확인할 수 있다.

4.4 성능평가

본 논문에서 디바이스 드라이버 부분 다운로딩을 구현하였다. 전체 응용프로그램을 모두 다운로딩 하는 방식보다 디바이스 드라이버 부분만 다운로딩 받는 효율은 다음과 같다.

$$\frac{D_{size}+A_{size}}{D_{Size}} \quad (2)$$

D_{size} : 디바이스 드라이버 부분 크기

A_{size} : 응용프로그램 부분 크기

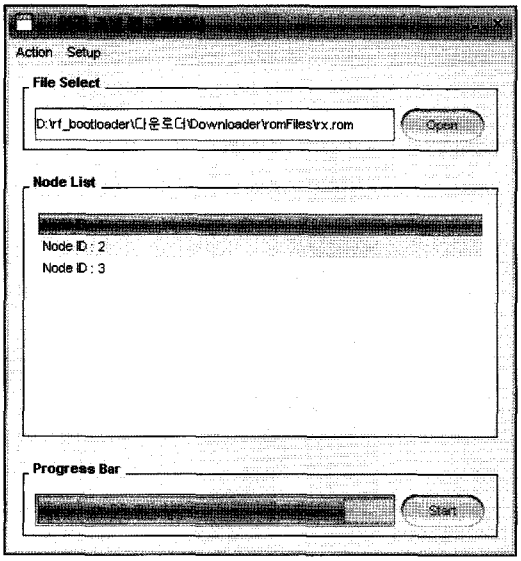


그림 7 디바이스 드라이버 부분 다운로딩

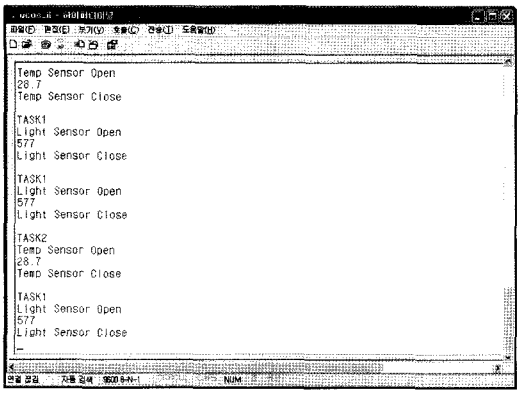


그림 8 다운로드 완료 후 실행화면

디바이스 드라이버 부분의 크기가 응용프로그램에 비해 작을수록 효율은 증가하고 다운로딩 시간이 짧아짐을 확인할 수 있었다.

디바이스 드라이버 부분 다운로딩을 세 가지 응용 프로그램에 적용한 결과, 표 3의 결과를 보였다. App1은 온도, 조도, 습도의 3가지 디바이스를 가지는 응용프로그램이다. App2는 움직임 센서와 구동기를 가지는 응용프로그램이다. App3은 자기센서와 구동기를 가지는 응용프로그램이다. 디바이스의 종류에 따라 디바이스 드라이버의 크기가 달라지게 되고, 파일의 크기에 비례하게 다운로딩 시간이 나타남을 보였다.

본 논문에서 구현한 운영체제(Nano-Q+2라고 칭함)와 기존 운영체제의 특징을 비교, 분석하였다. 표 4에서 볼 수 있듯이 기본 운영체제와 유사한 특징을 가지면서 트

표 3 응용 프로그램 다운로딩 결과

Application	-	Code Size	Downloading Time
App 1 (온,조,습)	App+DD	88KB	1.5 sec
	DD	1KB	0.1 sec
App 2 (움,구)	App+DD	56KB	0.8 sec
	DD	1KB	0.1 sec
App 3 (자,구)	App+DD	65KB	1.0 sec
	DD	1KB	0.1 sec

App+DD : 응용프로그램 + 디바이스 드라이버
DD : 디바이스 드라이버
App1 : 온도센서, 조도센서, 습도센서
App2 : 움직임센서, 구동기
App3 : 자기센서, 구동기

랜스듀서의 다양성을 극복할 수 있는 디바이스 드라이버 체계를 보유하고 디바이스 드라이버 부분 다운로딩을 지원한다.

5. 결론 및 향후 연구 방향

본 논문에서는 다양한 센서들을 통일된 방식으로 다룰 수 있는 API 및 디바이스 드라이버 체계를 기술하였다. 디바이스 드라이버 부분 다운로딩은 디바이스 드라이버를 선택적 등록 하였고 다운로딩 시간이 짧아짐을 확인하였다.

센서 API 및 디바이스 드라이버 체계는 리눅스의 디바이스 드라이버 체계를 간략화하여 설계하였다. 디바이스 드라이버 부분 다운로딩은 부트로더에 원격 다운로딩 부분을 포함하고 디바이스 드라이버 체계에 심볼테이블을 이용한 포인터 참조를 추가하여 설계하였다.

구현결과 많은 응용에서 다양한 센서 장착 시에 통일된 API와 디바이스 드라이버 체계가 사용자의 편리성을 강화함을 보였고 디바이스 드라이버 부분 다운로딩이 응용프로그램 전체 다운로딩보다 짧은 다운로딩 시간을 가지게 되었다.

향후 연구방향은 디바이스 드라이버 서버에서 적절한 드라이버를 검색하는 방식을 연구하는 것이다.

참 고 문 헌

[1] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in tinyos," Proc. of the First USENIX/ACM Symposium on Networked Systems Design and Implementation(NSDI 2004), 2004.

[2] S. Park, J. Kim, K. Lee, K. Shin, and D. Kim, "Embedded Sensor Networked Operating System," Proc. of 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2006.

[3] C. C. Han, R. Kumar, R. Shea, E. Kohler, and M.B. Srivastava, "A dynamic operating system for sensor nodes," Proc. of MobiSys, 2005, pp.163-176.

[4] Manseok Yang, Sun Sup So, Steve Eun, Brian Kim, Jinchun Kim, "Sensos: A Sensor Node Operating System with a Device Management Scheme for Sensor Nodes," International Conference on Information Technology (ITNG'07), 2007, pp. 134-139.

[5] A. Rubini, Linux Device Drivers, O'Reilly & Associates, Inc., 1998.

[6] Robert Love, Linux System Programming, OReilly, 2007.

[7] 윤덕용, AVR ATmega128 마스터, Ohm사, 2005.

[8] T. Schmid, H. Dubois-Ferriere, and M. Vetterli, "Sensorscope: experiences with a wireless building monitoring sensor network," Proc. of Workshop on Real-World Wireless Sensor Networks, 2005.

[9] P. Volgyesi amd A. Ledeczi, "Component-based development of networked embedded applications," Proc. of 28th Euromicro Conference, 2002.

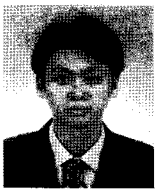
[10] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han, "MANTIS: System Support For MultimodAl NeTworks of In-situ Sensors," Proc. of 2nd ACM International Workshop on Wireless Sensor Networks and Applications, 2003, pp.50-59.

[11] Institute of Electrical and Electronics Engineers, Inc., "IEEE Standard for Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model," Mixed-Mobile Communication Working

표 4 센서노드 운영체제의 비교

Features	TinyOS	SOS	MANTIS	Nano-Q+	Nano-Q+2
Low Power Mode	○	○		○	○
Dynamic Reprogramming	○	○			○
Priority-based Scheduling			○	○	○
Real-time Guarantee			○	○	○
Transducer Device Driver					○
Device Driver Reprogramming					○
Embedded Model	Component Based	Module Based	Thread Based	Thread Based	Thread Based

Group of the Technical Committee on Sensor Technology TC-9 of the IEEE Instrumentation and Measurement Society, June 1999.
[12] National Institute of Standerds and Techology, IEEE 1451 Website http://ieee1451.nist.gov
[13] Draft Standard for a Smart Transducer interface for Sensors and Actuators: Common Functions, Communication Protocols, and Transducer Electronic Data Sheet(TEDS) Format, TC-9 of the IEEE Instrumentation and Measurement Society, Dec. 2005.
[14] T. Brooks and K. Lee, "IEEE1451 smart wireless machinery monitoring and naval vessels," Proc. of 13th International Ship Control Systems Symposium, Orlando Florida, Apr. 2003.



김 범 석
2006년 한남대학교 정보통신공학과 졸업(학사). 2008년 한남대학교 정보통신공학과 졸업(석사). 2008년~현재 삼성전자 연구원. 관심분야는 센서네트워크, 임베디드 운영체제

소 선 섭
1986년 이화여자대학교 전산학과(학사) 1998년 KAIST 전산학과(석사). 2001년 KAIST 전산학과(박사). 1988년~1995년 국방과학연구소 연구원. 1995년~현재 공주대학교 컴퓨터공학부 부교수. 관심분야는 소프트웨어 테스팅, 임베디드 소프트웨어, 센서네트워크

김 병 호
1990년 연세대학교 전산과학과(학사). 1992년 KAIST 전산학과(석사). 1997년 KAIST 전산학과(박사). 1997년~1998년 포스데이타 과장. 1998년~2005년 브레인21 대표이사. 2007년~현재 경성대학교 컴퓨터공학과 전임강사, 관심분야는 센서네트워크

은 성 배
1985년 서울대학교 전산학과(학사). 1987년 KAIST 전산학과(석사). 1995년 KAIST 전산학과(박사). 1995년~현재 한남대학교 정보통신공학과 교수. 관심분야는 실시간 시스템, 유비쿼터스 센서네트워크