임베디드 리눅스를 이용한 웹 기반 빌딩자동화시스템

Web-based Building Automation System using Embedded Linux

최 병 욱*, 신 은 철, 이 수 영
(Byoung-Wook Choi, Eun-Cheol Shin, and Soo Yeong Yi)

Abstract : In this paper, a Web-based management system for the building network is described. A multi-protocol converter based on SoC and embedded Linux is designed. The open source licensing, reliability, and broad hardware support are key reasons for use of embedded Linux in embedded industry. The multi-protocol converter integrates control network of RS-485 and LonWorks devices through TCP/IP protocol for a client with Java applet. The system consists of three-tier architecture, such as a client, a server that is performed on a multi-protocol converter, and control devices. The developed system includes the inverter motor control system with modbus protocol for the RS-485 network. The experiment results show that the multi-protocol converter using embedded Linux is a flexible and effective way to build a Web-based monitoring and control system.

Keywords : remote monitoring and control, web-based control, embedded linux, multi-protocol converter, BAS, lontalk

I. 서론

최근 SOC(System On a Chip) 기술이 발전함에 따라 UART 또는 이더넷(Ethernet) 드라이버와 같은 주변 장치들이 하나의 칩 안에 내장되어 출시되고 있다. 이러한 주변 장치들에 대한 드라이버 프로그램과 응용 프로그램을 개발자가 모두 작성하기 위해서는 많은 시간과 노력이 필요할 뿐만 아니라 신뢰성 또한 보장할 수 없다. 이러한 이유로 SOC 기반의 시스템에 운영체제가 많이 사용되고 있다. 그러나 현재 사용되고 있는 거의 모든 임베디드 운영체제가 상용이기 때문에 기술 의존적이 되고, 지원되는 하드웨어 또한 한정되어 있는 단점이 있다. 이러한 실시간 운영체제 또는 상용 임베디드 운영체제의 대안으로 임베디드 리눅스가 부각되고 있다. 임베디드 리눅스는 오픈 소스이며, 일반 리눅스와 마찬가지로 강력한 네트워킹과 다양한 하드웨어 아키텍처를 지원한다. 또한 다양한 응용 프로그램 소스와 개발 툴, 라이브러리를 무료로 제공하기 때문에 개발 비용을 절감하는 효과를 얻을 수 있다. 이처럼 임베디드 리눅스는 타 임베디드 운영체제에 비해 많을 장점을 가지고 있어서, 임베디드 시스템 분야에 많이 적용되고 있다[1,2].

빌딩자동화시스템(BAS, Building Automation System)은 빌딩의 공조설비, 방범/방재 등의 관리 기능을 관제실에 설치된 컴퓨터를 이용하여 관리하는 시스템을 말한다[3]. 대부분의 대형 빌딩에 빌딩자동화시스템이 설치되어 운영되고 있다. 최근에는 생활 수준이 높아짐에 따라 중소형 건물, 또는 일반 가정에도 홈오토메이션(Home Automation)이라는 이름으로 빌딩자동화시스템과 같은 시스템이 설치되는 추세이다. 그러나 기존의 시스템은 고가의 중앙 제어 컴퓨터를 기반으로 운영되며, 이기종의 프로토콜을 이용하는 제어기를 사용하기 위해 다수의 프로토콜 컨버터가 필요하다. 이런 구조의 시스템은 시스템의 구조가 복잡할 뿐만 아니라 비용이 많이 드는 단점이 있다.

본 논문에서는 기존의 BAS 시스템의 단점을 보완한 웹기반 BAS를 제시한다. 웹기반 BAS의 핵심 부분은 멀티-프로토콜 변환기(Multi-protocol converter)이다. 이 장치는 BAS에 많이 사용되는 RS-485 통신과 론웍스(LonWorks)[4], TCP/IP를 지원한다. 또한 운영체제로 RTOS(Real-Time Operating System) 및 상용 임베디드 운영체제 대신 서두에서 언급한 임베디드 리눅스를 사용한다. 그림 1은 본 논문에서 구현된 웹기반 BAS의 블록 다이어그램을 나타낸다. 실험을 위해 로컬 제어기로는 RS-485 통신 기반의 모터 제어기와 론웍스 기반의 제어기(에어컨, 펌프, 온도센서, 조도센서, 7-세그먼트, 발광다이오드)를 개발하여 연결하였다. 이러한 제어 장치는 빌딩자동화시스템에서 많이 사용되는 제어 단말기를 구현한 것으로 버스 구조로 확장이 가능하다. 또한 멀티-프로토콜 변환기는 TCP/IP를 이용하여 상위 프로그램과 연결되어 로컬 제어기를 제어하고 상태를 감시한다. 상위 프로그램으로 자바 애플릿을 이용하여 클라이언트에서 별도의 프로그램을 필요로 하지 않는다.

II. 멀티-프로토콜 변환기

1. 시스템 구성

멀티-프로토콜 변환기는 2개의 프로세서로 구성되어 있다.

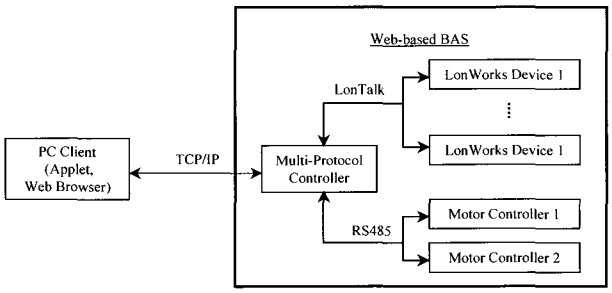

그림 1. 웹기반 BAS의 시스템 블록 다이어그램.
Fig. 1. System Block Diagram of the Web-based BAS.

* 책임저자(Corresponding Author)
논문접수 : 2004. 1. 19., 채택확정 : 2004. 3. 17.
최병욱, 신은철 : 선문대학교 제어계측공학과
(bwchoi@sunmoon.ac.kr, unchol@empal.com)
이수영 : 전북대학교 전자정보공학부(suylee@moak.chonbuk.ac.kr)
※ 본 논문은 과학기술부/학국과학재단지정 선문대학교 공조기술 연구센터에서 지원하였음.

표 1. 멀티-프로토콜 변환기의 하드웨어 사양.
Table 1. Hardware Specificaiton of the Multi-Protocol Converter.

CPUs		S3C4530A, CY7C53150
Memory	Flash	8MB + 32KB
	EEPROM	128KB
	SDRAM	16MB + 32KB
	DPRAM	2KB
Peripherals		2 Serial Ports
		2 RS-485 Chnnel
		10/100Mbps Ethernet
		LonWorks Transceiver

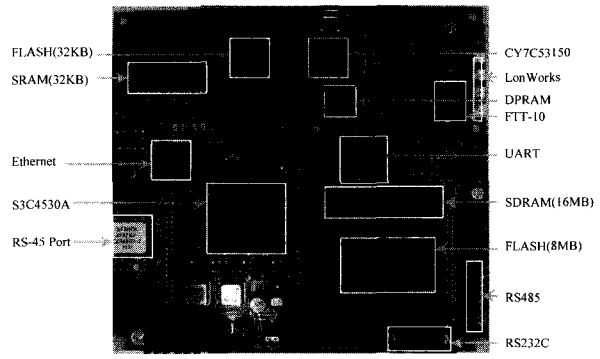

그림 2. 멀티-프로토콜 변환기의 시스템 구성.
Fig. 2. System Layout of the Multi-Protocol Converter.

첫 번째 프로세서는 TCP/IP와 RS-485통신을 위한 부분으로 ARM7TDMI기반의 삼성 S3C4530A SoC(System on a Chip)이다. 두 번째 프로세서는 론웍스 통신을 위한 부분으로 Cypress의 CY7C52150 프로세서이다. 이 두 프로세서간의 인터페이스를 위해 DPRAM(Cypress, CY7C136, Dual-Port RAM)을 이용하였다. 주변 장치로는 RS-232, RS-485, LonWorks 포트, Ehternet 포트가 있다. RS-485 통신과 네트워크를 위해 UART 컨트롤러와 이더넷 컨트롤러를 사용하였으며, 론웍스 통신을 위해 FTT-10 트랜스시버를 사용하였다. 하드웨어 스펙은 표 1과 같으며, 시스템 실사진을 그림 2에 나타내었다[5, 6].

2. 리눅스 포팅

리눅스 포팅은 기본적으로 32비트 마이크로프로세서인 i386에 처음으로 개발되어 현재는 MMU(Memory Management Unit)이 있는 여러 마이크로프로세서 아키텍처에 포팅되어 있다. 그러나 본 연구에서 사용하는 S3C4530A는 MMU가 없기 때문에 기존의 리눅스 커널을 사용할 수 없다. 때문에 MMU가 없는 마이크로프로세서를 위한 임베디드 리눅스인 유시리눅스(uClinux)를 멀티-프로토콜 변환기에 포팅하였다. 유시리눅스 커널은 일반 리눅스 커널과 달리 가상 메모리 지원을 제거 하였으며, 멀티-타스킹(multi-tasking)을 위한 방법에도 차이가 있다. 가상 메모리를 제거했다는 의미는 하드웨어적으로 이를 지원하지 못하기 때문에, 물리 메모리와 가상 메모리를 같은 주소로 사용한다는 것이다. 따라서 일반 리눅스 커널의 메모리 맵과는 다르게 유시리눅스 커널은 그림 3과 같이 힙(heap)과 스택(stack)의 위치가 바뀌어 있어 고정된

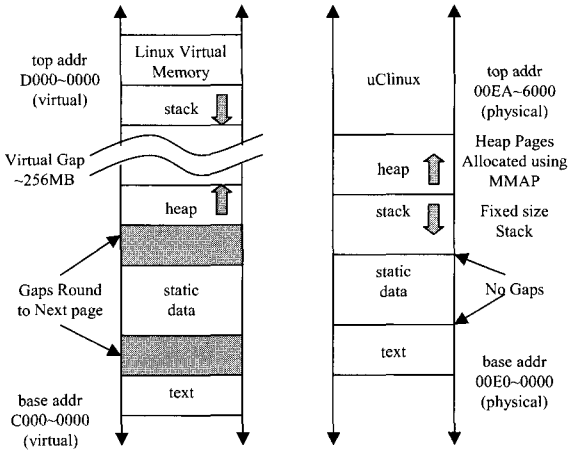

그림 3. 일반 리눅스와 유시리눅스의 메모리 맵.
Fig. 3. Memroy maps of Linux and uClinux.

스택을 사용한다. 멀티-타스킹을 위한 전형적인 리눅스의 fork system call은 copy-on-write라는 기법으로 이루어졌지만 유시리눅스에서는 메모리 사용이 제한적이기 때문에 vfork system call과 함께 쓰레드(thread)를 사용한다[7].

유시리눅스를 포팅하는 과정은 로우레벨(low-level) 포팅과 하이레벨(high-level) 포팅으로 이루어진다[8,9]. 로우레벨 포팅은 하드웨어와 관련된 부분으로 다음과 같이 진행된다.

- 엔트리 포인트(entry point) 정의
- 시스템 레지스터와 PLL 설정
- 시리얼 포트 초기화
- 스택 초기화
- 프로세서 타입과 머신 타입 설정
- 리눅스 커널(start_kernel)로 점프

하이레벨 포팅은 리눅스 커널과 관련된 부분으로 로우레벨에서 호출되어지는 C언어 코드이다. 진행과정은 다음과 같다.

- 리눅스 커널 배너 출력
- 프로세서와 머신 정보 설정
- 트랩(exception vector) 초기화
- 인터럽트 초기화
- 스케쥴러에서 요구되는 데이터 초기화
- 스프트웨어 인터럽트 초기화
- 콘솔 초기화
- 인터럽트 인에이블
- 디바이스 드라이버 초기화

멀티-프로토콜 변환기도 하이레벨 포팅과 로우레벨 포팅 과정을 통해 유시리눅스를 포팅하였으며, 이더넷과 RS-485를 위한 디바이스 드라이버를 작성하였다.

3. 프로세서간의 인터페이스

멀티-프로토콜 변환기는 2개의 프로세서로 구성되어 있기 때문에 이들 사이에 데이터를 인터페이스할 수 잇는 방법이 필요하다. 프로세서간의 데이터를 인터페이스할 수 잇는 방법으로는 시리얼 통신이나, 패러럴 통신, DPRAM을 이용하는 방법이 있는데, 이중 DPRAM을 이용하는 방법을 선택하였다.

표 2. 론웍스 디바이스의 하드웨어 스펙
Table 2. Hardware Specification of LonWorks Devices

CPUs	CY7C53150
Memory	Flash 32KB
	SRAM 32KB
Transceiver	FTT-10A
Peripheral	11 GPIOs, Memory-Mapped I/O

DPRAM은 2개의 데이터 버스와 어드레스 버스를 가지고 있어서 양방향(Left Side, Right Side)으로 데이터를 읽기/쓰기가 가능하며, 데이터의 변경과 데이터의 동시 쓰기를 방지하기 위해 인터럽트 신호와 busy 신호가 제공된다. DPRAM은 Left Side 또는 Right Side에서 데이터를 저장할 때 인터럽트 신호가 자동으로 생성되지 않는다. 때문에 사용자는 데이터를 쓰고 난 후에 인터럽트 신호를 만들어주어야 한다. 예를 들어, Left Side에서 데이터를 저장한 후 Right Side에 인터럽트 신호를 발생시키기 위해서는 0x7FF 번지에 write 신호를 보내야 한다. 이 인터럽트 신호를 클리어하기 위해서는 Right Side에서 0x7FF번지에 Read 신호를 보내주면 된다. 반대의 경우 0x7FE번지를 쓰거나 읽으면 된다.

III. 로컬 디바이스

1. 론웍스 디바이스

론웍스(LonWorks)는 론토크(LonTalk) 프로토콜을 이용한 제어 네트워크를 말하며, 이 프로토콜 스펙을 하드웨어적으로 구현한 프로세서를 뉴런칩(Neuron Chip)이라 한다. 본 논문에서는 이 프로세서를 가지고 론웍스 디바이스를 구현하였다. 론웍스 네트워크를 위한 트랜스시버(Transceiver)에는 트위스트 페어(Twist Pair)와 전력선(Power Line), RF(무선통신), 광케이블 등이 있는데, 그 중 트위스트 페어 트랜스시버(FTT-10A)를 사용하였다. 이 트랜스시버는 76.8Kbps의 속도로 통신할 수 있으며, Free Topology 방식의 배선이 가능하고, 리피터(Repeater)와 같은 외부 장치없이 500m까지 통신이 가능하다. 구현된 론웍스 디바이스의 하드웨어 사양은 표 2와 같다.

론웍스 디바이스는 다른 제어 장치와 범용 I/O(General Purpose I/O) 또는 메모리 맵 I/O(Memory-Mapped I/O)를 이용하여 연결하기 위해 그림 4와 같이 범용으로 설계하였다. 메모리 맵 I/O는 본 논문에서 사용한 CY7C53150 프로세서가 외부 인터페이스를 제공하기 때문에 구현 가능한 부분이다.(CY7C53120 프로세서는 외부 메모리 인터페이스를 제공하지 않는다.) 외부 CS(Chip Select)는 어드레스 버스 중 0 ~ 2 비트의 어드레스를 이용하여 8개를 만들었다. 외부 메모리 I/O를 위한 CS를 만들기 위해서는 뉴런 C 개발 툴에서 메모리 맵을 설정해 주어야 한다[10].

이상과 같이 구현된 론웍스 디바이스에 에어컨 제어 회로, 펌프 제어 회로, 온도 센서 회로, 조도센서 회로 7-세그먼트 회로, 발광 다이오드 회로를 추가하여 그림 5과 같이 구현하였다.

2. 론웍스 디바이스의 소프트웨어 구조

각각의 제어기는 뉴런 C(Neuron C)를 이용하여 프로그램하였다. 뉴런 C는 뉴런 프로세서를 개발하기 위한 언어로써

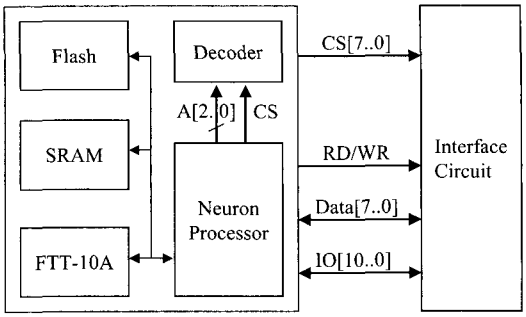

그림 4. 론웍스 디바이스의 블록 다이어그램.
Fig. 4. Block diagram of LonWorks devices.

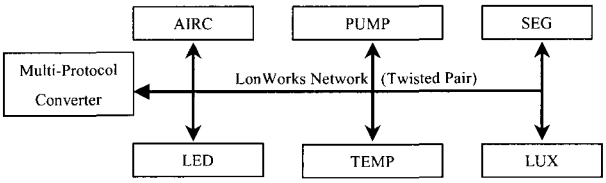

그림 5. 론웍스 기반의 제어 네트워크.
Fig. 5. Control Network based on LonWorks.

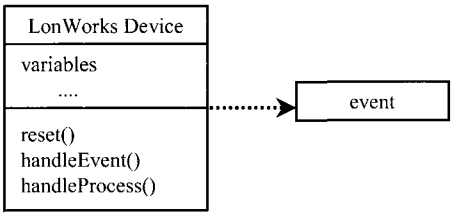

그림 6. 론웍스 디바이스의 S/W 구조.
Fig. 6. Software structure of LonWorks devices.

ANSI C를 기반으로 만들어졌으며, 네트워크와 I/O 포트를 위한 데이터 타입이 추가되었다. 또한 이벤트-핸드링을 지원하기 때문에 이벤트 방식의 프로그램이 가능하다[11].

각각의 론웍스 디바이스는 그림 6과 같이 reset, handleEvent, handleProcess 함수로 구성된다. Reset()는 디바이스에 전원이 인가되면 가장 먼저 실행되는 함수이다. 이 함수에서는 디바이스를 초기화하는 로직(Logic)이 들어간다. HandleEvent() 함수는 이벤트가 발생했을 때 실행되는 함수로 네트워크 변수의 값이 변경되거나 타이머 오버플로우등의 이벤트가 발생하면 실행된다. HandleProcess() 함수는 handleEvent() 함수에서 이벤트의 종류 및 값에 따라 실행되는 함수이다. 이 함수에서는 I/O 포트와 메모리 맵 I/O를 이용하여 인터페이스 회로를 제어하는 신호를 만든다.

각각의 론웍스 디바이스를 개발하였지만 아직 론웍스 디바이스가 론웍스 네트워크로 연결된 것은 아니다. 이것들을 서로 연결하기 위해서는 바인딩(Binding)이라는 작업이 필요하다. 바인딩은 론웍스 네트워크에 연결된 론웍스 노드들을 연결하는 작업이다(론웍스 네트워크에 연결된 디바이스를 론웍스 노드라 한다.) 바인딩을 하기 위해서는 우선 네트워크 변수나 네트워크 태그를 선언해주어야 한다. 본 논문에서는 네트워크 변수만을 사용하였기 때문에 이 부분에 대하여만

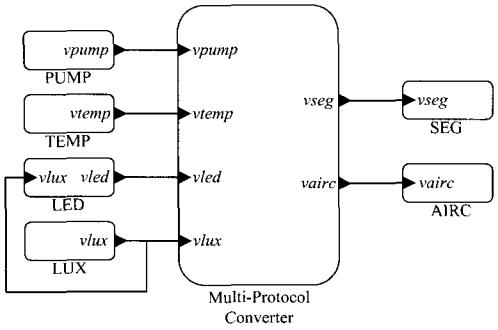

그림 7. 론웍스 노드의 바인딩 구조.

Fig. 7. Network Binding for LonWorks devices.

논하겠다. 네트워크 변수는 표준 네트워크 변수와 사용자 네트워크 변수가 있는데, 그 중 사용자 네트워크 변수를 사용하였다. 네트워크 변수는 반드시 입력과 출력의 데이터 타입이 같아야 하며, 입출력이 1:1 또는 1:n으로 연결되어야 한다. 그림 7인 론웍스 노드들의 바인딩 구조를 나타낸다.

각각의 네트워크 변수에 대하여 알아보겠다. 멀티-프로토콜 변환기는 각각의 론웍스 노드와 1:1로 연결되고, LUX와 LED만 서로 연결되어 있다. 멀티-프로토콜 변환기의 vpump와 vtemp, vled, vlux는 입력 네트워크 변수이고, vseg와 vairc는 출력 네트워크 변수이다. vseg는 SEG 노드의 세그먼트 번호와 값을 저장하며, vairc는 AIRC 노드와 연결된 에어컨의 On/Off 또는 온도 UP/DOWN을 제어하기 위한 정보를 저장한다. vpump는 PUMP 노드에 연결된 펌프의 On/Off 상태값을 저장하며, vtemp와 vlux는 TEMP와 LUX 노드에 연결된 온도센서와 조도센서의 값을 저장한다. vled는 LED 노드에 LED 상태값을 나타낸다.

3. RS-485 통신 기반의 디바이스 구성

본 논문에서는 RS-485 통신을 위한 로컬 디바이스로 모터 제어기를 사용하였다. 이 모터 디바이스는 모드버스 프로토콜을 이용하여 데이터를 송수신한다. 모드버스 프로토콜은 산업용 기기 공급자들이 일반적으로 사용하는 프로토콜이다. 모드버스 프로토콜은 제어기 자신이 연결되어 있는 네트워크 종류와는 무관하게 인식하여 사용할 수 있는 메시지 구조 형태로 되어 있다. 이 메시지는 제어기가 다른 장치에 요청 및 응답을 하는데 사용된다. 이들 메시지 필드의 구성은 공통 형태를 유지한다. 모드버스 통신은 마스터-슬레이브(Master-Slave) 기법이 사용되는데, 이는 단지 하나의 장치(마스터)만이 트랜잭션(Transaction)을 초기화시킬 수 있다는 의미이다. 그러면 다른 장치(슬레이브)들은 요청된 데이터를 마스터에게 응답해주거나 요청된 동작을 수행한다. 일반적으로 마스터는 호스트 컴퓨터나 프로그래밍 패널이 되며, 슬레이브는 프로그램 가능한 디바이스가 된다. 본 논문에서는 멀티-프로토콜 변환기가 마스터이며, 모터 제어기는 슬레이브이다.

멀티-프로토콜 변환기와 모터 제어기간의 사용된 모드버스 패킷(Packet)은 그림 8과 같다.

General Packet Format은 마스터에서 슬레이브로 명령을 내릴 때 사용되는 형태이고 General Acknowledge Packet Format은 마스터로부터 요청을 받은 슬레이브가 마스터로 응답을 보낼 때 사용되는 형태이다.

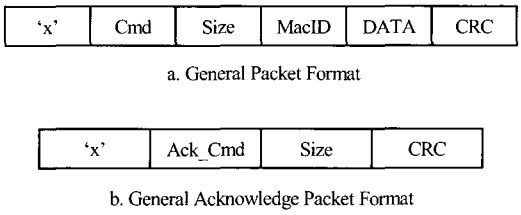

그림 8. 모드버스 패킷 구성.

Fig. 8. Packet structure of Modbus protocol.

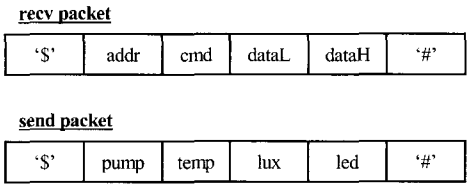

그림 9. TCP/IP 패킷 구조

Fig. 9. Structure of TCP/IP Packet

IV. 데이터 변환 메커니즘

멀티-프로토콜 변환기는 TCP/IP 소켓을 통해 상위 프로그램과 통신을 하기 때문에 TCP/IP 데이터를 로컬 디바이스에 전달하거나 로컬 디바이스의 데이터를 TCP/IP 데이터로 변경할 수 있는 메커니즘이 필요하다.

멀티-프로토콜 변환기의 서버 프로그램은 상위 프로그램으로부터 TCP/IP 소켓을 통해서 데이터를 받으면 RS-485 디바이스에 보내는 데이터인지 론웍스 디바이스에 보내는 데이터인지를 비교하여 데이터를 로컬 디바이스에 보낸다. 예를 들어, RS-485 디바이스에 보내는 데이터인 경우 멀티-프로토콜 변환기에 포팅된 유시리눅스의 RS-485 디바이스 드라이버를 통해서 데이터를 전달하며, 론웍스 디바이스의 데이터인 경우 DPRAM에 받은 데이터를 저장함으로써 데이터를 론웍스 디바이스에 보낸다. 론웍스 디바이스의 상태값을 읽어올 때는 DPRAM에 저장된 것을 읽어온다. 서버 프로그램의 동작 순서는 다음과 같다.

1) RS-485 디바이스 드라이버를 연다. 만약 디바이스 드라이버가 열리지 않을 경우, 에러 메시지를 출력하고 프로그램을 종료한다.

2) 소켓 서버를 생성하고 클라이언트(상위 프로그램)의 접속을 기다린다.

3) 클라이언트에서 접속하면 접속에 대한 쓰레드를 생성한 후 클라이언트와 통신을 한다.

4) 클라이언트로부터 요청이 들어오면 데이터를 비교하여 해당되는 로컬 제어기로 데이터를 보내거나 로컬 제어기로부터 상태값을 받아 클라이언트로 보낸다.

5) 클라이언트로부터 종료 메시지를 받거나, 접속이 끊기면 생성된 쓰레드를 소멸시키고 다시 클라이언트의 접속을 기다린다.

클라이언트와 멀티-프로토콜 변환기 사이의 데이터 패킷 포맷은 그림 9와 같다.

패킷 구조에서 '$'와 '#'은 패킷의 처음과 마지막을 나타낸다.

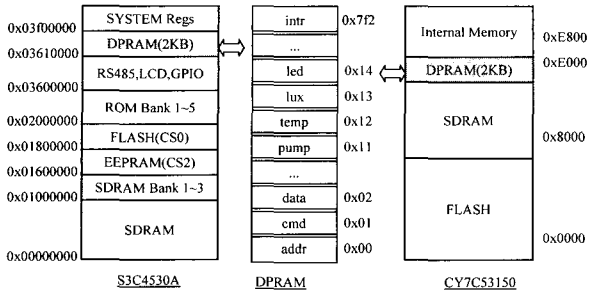

그림 10. 프로세서간의 데이터 전송 구조.

Fig. 10. Data communication mechanism between processors.

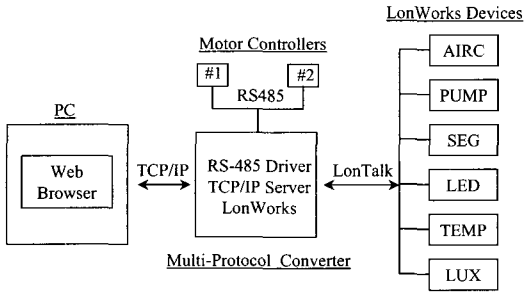

그림 11. 웹기반 BAS의 시스템 구성.

Fig. 11. System structure of a Web-based BAS.

나타낸다. Recv_packet에서 addr 필드는 프로토콜 타입을 나타낸다. Cmd 필드는 명령을 나타내며, dataL과 dataH는 cmd에 대한 부가적인 정보가 들어간다. Send_packet에서 pump와 temp, lux, led는 각 디바이스의 상태값이 들어간다.

두 프로세서간의 데이터 전송은 그림 10과 같은 메모리 구조로 인터럽트 신호를 이용하여 이루어진다. 멀티-프로토콜 변환기에서 론웍스 디바이스로 데이터를 전송할 때는 DPRAM에 addr과 cmd, data를 저장한 후 인터럽트 신호를 발생시키면 그림 6의 론웍스 디바이스에 프로그램된 eventHandle() 함수가 호출된다. 호출된 함수에서는 DPRAM의 addr와 cmd, data값을 읽어서 어느 론웍스 디바이스의 데이터 값인지 비교하여 해당 네트워크 변수에 저장함으로써 데이터를 전송한다. 론웍스 디바이스의 상태를 읽어올 때는 방식이 다르다. 각각의 론웍스 디바이스는 설정된 주기마다 멀티-프로토콜 변환기로 상태값을 보낸다. 상태값을 받은 멀티-프로토콜 변환기는 이 값들을 DPRAM에 저장한다. 상위 프로그램에서 상태값을 요구하면 멀티-프로토콜 변환기의 서버 프로그램은 DPRAM에 저장된 값을 읽어서 보낸다.

V. 웹기반 BAS 구현

1. 시스템 구성

이상과 같이 구현한 멀티-프로토콜 변환기와 로컬 디바이스를 이용하여 그림 11과 같이 웹기반 간이형 빌딩자동화시스템을 구성하였다. 웹기반의 시스템을 구현하기 위해 임베디드 리눅스에 임베디드 웹서버인 보아(Boa) 웹서버를 포팅하였다. 사용자 GUI로는 자바 애플릿을 사용하였다. 자바 애플릿은 멀티-프로토콜 변환기의 서버 프로그램과 TCP/IP 소켓을 이용하여 통신을 한다.

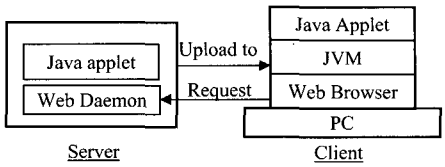

그림 12. 자바 애플릿의 실행 메커니즘.

Fig. 12. Running mechanism for Java Applet.

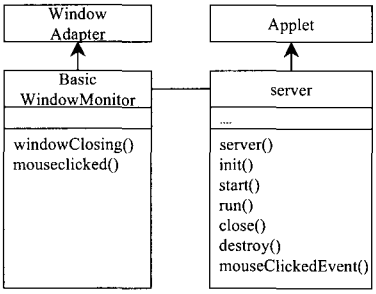

그림 13. 자바 애플릿 프로그램의 구조.

Fig. 13. Structure of Java Applet program.

2. 자바 애플릿

웹기반 시스템을 제어하기 위해서는 ASP, PHP, JSP와 같은 서버 스크립트(Script) 언어를 사용해야 한다. 그러나 이러한 서버 스크립트 언어는 임베디드 웹 서버에서 구동하기에는 무겁기 때문에 사용할 수 없다. 그 다음으로 사용할 수 있는 방법이 CGI를 이용하는 방법이다. 하지만 CGI는 서버에서 실행되기 때문에 임베디드 시스템에는 적합하지 않다. 이러한 문제점에 대안으로 자바 애플릿을 제안한다. 자바 애플릿은 서버 스크립트를 필요로 하지 않기 때문에 임베디드 웹 서버만으로 구동가능하며, 클라이언트에서 실행되므로 서버에 부하를 주지 않는다.

선마이크로시스템(Sun MicroSystems)에서 만든 자바 언어는 객체지향 언어로써 임베디드 시스템을 위해 플랫폼에 구애받지 않는 언어로써 처음 개발되었다. 하지만 자바언어가 플랫폼에 구애받지 않기 위해서는 플랫폼에 자바 가상머신이 내장되어 있어야 한다. 이 자바 가상머신을 임베디드 시스템 보드에 올린다는 것은 많은 무리가 따른다. 하지만 자바 애플릿은 서버에서 동작하는 것이 아니라 클라이언트에서 다운로드된 후 실행된다. 이 때 클라이언트에 자바 가상머신이 설치되어 있어야 한다. 하지만 약간의 단점이 있다. 이는 접속시 서버로부터 자바 애플릿 프로그램을 다운받아야 하기 때문에 시간이 다른 웹 문서에 비해 많이 걸린다. 이러한 시간이 필요한 이유는 다운로드 받는 애플릿 프로그램이 자바 컴파일러로 컴파일된 자바 바이트 코드로 HTML 문서에 비해 크기가 크기 때문이다. 그러나 최근 초고속 인터넷 통신의 발달로 인해 다운로드에 걸리는 시간은 문제가 되지 않는다.

그림 12는 자바 애플릿의 동작 메커니즘을 간략히 도식화한 것이다. 클라이언트는 웹서버에 자료를 요청하며, 웹 데몬은 자료를 처리할 수 잇는 자바 애플릿을 클라이언트에게 전달한다. 그러면 클라이언트는 자신이 갖고 있는 자바 가상 머신을 통해 전달 받은 자바 애플릿을 수행하여 결과를 얻는다.

그림 13은 본 논문에서 구현한 자바 애플릿의 구조이다.

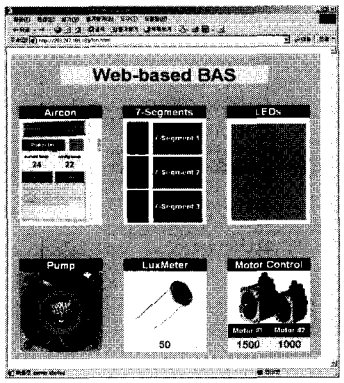

그림 14. 자바 애플릿으로 구현한 GUI.
Fig. 14. GUI implemented by Java Applet.

자바 애플릿은 기본적으로 init(), start(), close(), destroy()의 함수로 구성된 생명 주기(life cycle)를 갖기 때문에 이들 함수에 프로그램을 초기화하고 종료될 때 수행해야 하는 동작을 포함시켰다. Init()함수는 애플릿이 실행될 때 가장 먼저 호출되는 함수로 변수들을 초기화하고, GUI 컨테이너와 컴포넌트를 생성한다. 그 다음으로 start()함수가 호출되며, 이 함수는 멀티-프로토콜 변환기의 서버 프로그램과 접속을 하고 타이머 타스크(TimerTask)를 실행한다. 타이머타스크는 1초에 한번식 로컬 디바이스의 상태를 멀티-프로토콜 변환기에 요구할 때 사용된다. Close()화 destroy()함수는 프로그램이 종료될 때 호출된는 함수로서 서버에 프로그램이 종료됨을 알리고 클라이언트 소켓을 닫는다. Run()함수는 타이머타스크에서 실제로 동작하는 함수로 멀티-프로토콜 변환기의 서버 프로그램에 상태값을 요구한다. 로컬 디바이스를 제어하기 위해 제어 버튼을 클릭하면 WindowMonitor로부터 mouseClicked 이벤트가 발생하여 각 버튼에 다른 mouseClickedHandle 함수가 실행된다. 이 함수는 앞서 설명한 멀티-프로토콜 변환기의 서버 프로그램에 제어 명령에 따른 데이터 패킷을 만들어서 보낸다. 그림 14는 자바 애플릿으로 구현한 GUI 화면이며, 그림 15은 실제로 구현한 웹기반 간이형 빌딩자동화시스템의 실사진이다.

VI. 결론

본 논문에서는 임베디드 리눅스 기반의 멀티-프로토콜 변환기를 개발하고 웹기반 간이형 빌딩자동화시스템을 개발하였다. 멀티-프로토콜 변환기는 산업계와 빌동자동화시스템 분야에서 많이 사용되는 RS-485와 론웍스, TCP/IP를 지원한다.

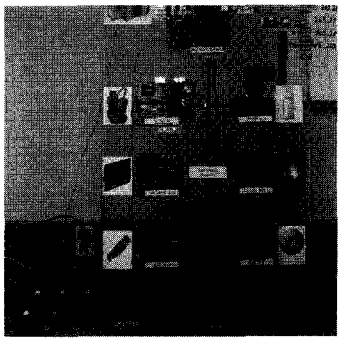

그림 15. 구현된 웹기반 간이형 BAS.
Fig. 15. Implemented small Web-based BAS.

또한 사용자가 별도의 프로그램 없이 인터넷이 연결된 곳이면 어디서든 모니터링과 제어가 가능하도록 임베디드 리눅스에 임베디드 웹서버를 포팅하였으며, 자바 애플릿을 이용하여 사용자 인터페이스를 구현하였다. 또한 운영체제를 상용 운영체제가 아닌 임베디드 리눅스를 적용함으로써 자동화 분야에 임베디드 리눅스가 사용 가능함을 보였다. 이러한 개념은 이 기종의 프로토콜이 혼재된 제어시스템을 통합적으로 웹에서 관리 및 제어가 가능함을 보이고 있다.

참고문헌

[1] A. Lennon, "Embedded linux", IEE Review. pp. 33-37, May 2001.
[2] 김주민, 김홍렬, 양광웅, 김대원, "Linux와 TCP/IP를 이용한 분산 실시간 이동로봇 시스템 구현에 관한 연구", 제어 · 자동화 · 시스템공학논문지, 제 9 권 제 10 호, pp. 789-797, 2003.
[3] http://www.advancedbuildings.org/main_t_load_build_automation_sys.htm
[4] http://www.echelon.com/products/lonworks/default.html, "Introduction to the LonWorks system"
[5] http://www.arm.com/armtech/ARM7TDMI?OpenDocument
[6] S3C4530A User's guide, Samsung, 2001.
[7] http://unclinux.org/description
[8] 조덕연, 최병욱, "임베디드 리눅스를 이용한 산업용 인버터의 웹 기반 원격관리", 제어 · 자동화 · 시스템공학논문지, 제 9 권 제 4 호, pp. 340-346, 2003.
[9] 최병욱, 고경철, 문전일, 임계영, 임베디드 리눅스 실습 및 활용, 홍능과학출판사, 2003.
[10] NodeBuilder User's Guide, Echelon, 1996.
[11] Neuron C Programmer's Guide, Echelon, 2003.

최 병 욱

1962년 2월 13일생. 1986년 한국항공대학교 항공전자공학과 졸업(공학사), 1988년 한국과학기술원 전기 및 전자공학과 졸업(공학석사). 1992년 동대학 대학원 박사과정 졸업(공학박사). 1992-2000년 LG산전 연구소 엘리베이터 연구실장, 임베디드 시스템 연구팀장, 2001~2003년 ㈜임베디드웹 대표이사. 2000년~현재 선문대학교 제어계측공학과 교수. 관심분야는 소트프웨어 엔지니어링, 내장형 시스템, 실시간 운영체제, 임베디드 리눅스, 필드버스 및 분산 제어 시스템, 지능제어로봇.

이 수 영

1964년 10월 2일생, 1988년 2월 연세대학교 전자공학과 졸업(공학사), 1990년 2월 한국과학기술원 전기 전자공학과 졸업(공학석사), 1994년 8월 한국과학기술원 전기 및 전자공학과 졸업(공학박사), 1995년 3월-1999년 8월 한국과학기술원 휴먼로봇연 구센터 선임연구원, 1997년 2월 –1998년 2월 Univ. of Southern California, Dept. of CS, Post Doc. 1999년 9월-현재 국립전북대학교 전자정보공학부 조교수, 관심분야 인간형 로봇, 이동로봇, 지능제어.

신 은 철

1977년 6월 15일생. 2003년 선문대학교 전자공학과, 제어계측공학과 졸업(공학사). 2003년~현 동대학교 제어계측과 석사 과정. 과심분야 임베디드 시스템, 임베디드 리눅스, 실시간 운영체제.