임베디드 리눅스 기반의 다중 프로토콜 제어기 개발 및 빌딩자동화시스템과의 연동 적용

Development of Multi-protocol Controller based on Embedded Linux and Its Application to BAS

최 병 욱*, 김 현 기, 신 은 철
(Byoung-Wook Choi, Hyun-Gy Kim, and Eun-Cheol Shin)

Abstract : In this paper, we developed a multi-protocol controller based on SoC and embedded Linux and applied it to integrate with BAS in a unified TCP/IP socket communication. The multi-protocol controller integrates control networks of RS-485 and LonWorks devices to BAS. The system consists of three-tier architecture, such as a BAS, a multi-protocol converter, and control devices. By using UML, we modeled the system architecture. In order to show the feasibility of system architecture, it was applied to a small BAS system. The experimental results show that the multi-protocol controller using embedded Linux is flexible and effective way to develop a building control system.

Keywords : microprocessor, embedded linux, multi protocol controller, BAS, MMI

I. 서론

실시간 운영체제의 단점을 극복하고 보안할 수 있는 운영체제로써 임베디드 리눅스가 활용되고 있다. 리눅스는 1991년 리누즈 토발즈(Linus B. Torvalds)가 처음 커널(Kernel)를 작성하고 공개한 이래 토발즈의 관리하에서 공개 프로젝트 형식으로 개발되고 있는 운영체제이다[1]. 이러한 리눅스를 이용하여 임베디드 시스템 환경에 맞게 커널의 하드웨어 의존적인 부분을 수정한 것이 임베디드 리눅스이다. 임베디드 리눅스의 장점은 공개 소스이며, 커널 코드가 무료라는 것과 폭넓은 하드웨어 아키텍처의 지원과 개발에 필요한 툴, 라이브러리가 공개되어 있다는 점이다. 따라서 초기 개발비용이 저렴하며 개발과정에서부터 모든 소스 코드가 완전히 공개되어 있기 때문에 많은 개발자들이 문제가 발생될 소지를 빨리 발견하고 제거함으로써 배포 버전에 오류가 포함될 가능성이 다른 어떤 운영체제에 비해 적어 안정적이다. 또한 다양한 마이크로프로세서에 포팅(porting)되어 있고 디바이스 드라이버 소스도 공개되어 있다. 이처럼 임베디드 리눅스가 다른 운영체제보다 갖는 많은 장점들로 인하여 임베디드 시스템 분야에 많이 적용되고 있다[2].

본 연구에서는 공장자동화(FA) 및 빌딩자동화시스템(BAS) 환경에 사용되고 있는 여러 프로토콜 중에 RS-485와 론웍스(LonWorks)를 지원대상으로 설정하고, 이를 지원하는 다중 프로토콜 제어기를 설계하였다[3]. 이 제어기는 RS-485와 TCP/IP 통신을 위해 ARM7TDMI 코어 기반의 시스템 온 칩(SoC)인 삼성의 32bit S3C4530A와 론웍스를 위해 TMPN3150 프로세서를 사용하였다. 또한 두 프로세서간의 데이터 통신을 위한 인터페이스로 DPRAM(Dual-Port RAM)을 사용하였다[4,5].

메인 제어장치인 S3C4530A 프로세서는 RS-485와 TCP/IP를 구현하기 위해 임베디드 리눅스를 포팅하였다. 단 사용되는 칩에는 메모리 관리 유닛(MMU, Memory Management Unit)이 없기 때문에 일반 리눅스가 아닌, 메모리 관리 유닛의 기능을 소프트웨어적으로 대치한 리눅스 커널인 유시리눅스(uClinux)를 이용하였다. 그리고 론토크(Lontalk) 프로토콜을 위한 TMPN3150 칩은 뉴런 칩(Neuron Chip)이라 하며, 칩 안에 프로토콜 스펙이 내장되어 있다[6]. 이렇게 하드웨어 구축과 포팅이 이루어지면 현재 구현된 RS-485 통신을 이용하는 제어기와 론토크 프로토콜을 사용하는 제어 장치에 대한 제어 및 관리를 위한 유시리눅스 기반의 서버 프로그램 작성을 수행하게 된다. 또한 사용자가 로컬 제어기 제어를 위하여 중앙 제어 컴퓨터의 MMI(Man-Machine Interface)를 구현한다. 그리고 구현된 MMI와 응용 프로그램 사이에 인터페이스를 위해 윈도우 기반의 TCP/IP 클라이언트 프로그램을 작성한다. 또한 로컬 제어기를 모니터링 및 제어하기 위한 데이터 패킷(packet)이 구현되어야 하는데 RS-485에 대해서는 모드버스(modbus) 프로토콜을 구현하였다.

II. 다중 프로토콜 제어기 개발

1. 다중 프로토콜 제어기

다중 프로토콜 제어기는 2개의 마이크로프로세서로 구현되어 있다. 즉 주 제어장치로 사용되는 S3C4530A와 론웍스 기반의 네트워크 인터페이스를 위한 뉴런칩인 TMPN3150이다. 또한 2개의 마이크로프로세서간의 데이터 통신은 DPRAM을 사용하였다. 주 제어장치에서는 여러 가지 통신 인터페이스를 위한 주변장치가 구현되어 있는데 TCP/IP를 지원하기 위해 S3C4530A에 내장된 이더넷 컨트롤러가 이용되었으며, 산업용 제어장치와 연동을 위하여 RS-485 디바

* 책임저자(Corresponding Author)
논문접수 : 2004. 1. 10., 채택확정 : 2004. 3. 25.
최병욱, 신은철 : 선문대학교 제어계측공학과
(bwchoi@sunmoon.ac.kr/unchol@empal.com)
김현기 : 세연테크 연구소(pathk@seyeon.co.kr)
※ 본 논문은 과학기술부/한국과학재단지정 선문대학교 공조기술 연구센터에서 지원하였음.

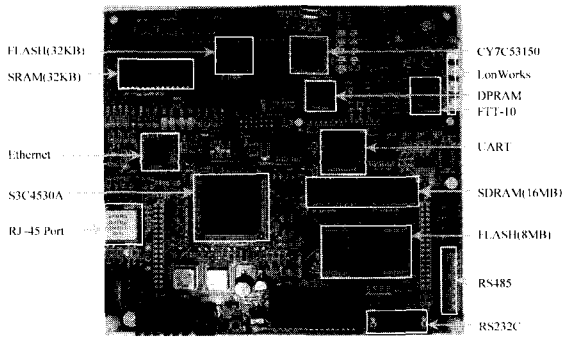


그림 1. SoC를 이용한 다중 프로토콜 제어기.
Fig. 1. Multi-Protocol Controller based on a SoC.

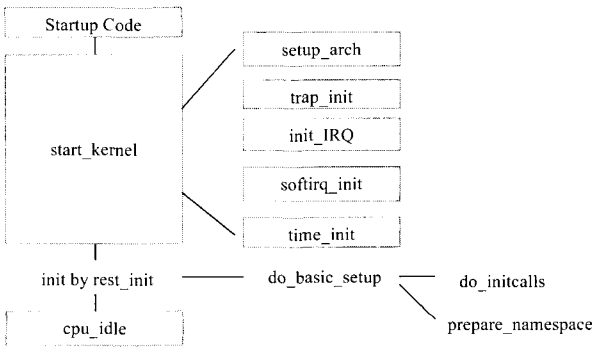


그림 2. 유시리눅스의 부팅 순서.
Fig. 2. Booting Sequence of uClinux.

이스가 내장되어 있다. 또한 임베디드 리눅스의 콘솔 제어 장치를 위한 RS-232가 구현되어 있다. 시스템 메모리는 주 제어장치와 뉴런칩 부분으로 구분되어지는데, 리눅스 동작을 위한 SDRAM(16MB), 커널과 루트 파일시스템을 저장할 Flash(8MB), 시스템 부트로더를 위한 EEPROM (128KB)이 주 제어장치 부분이다. 그리고 뉴런칩에서는 어플리케이션을 저장하는 Flash(32KB)와 어플리케이션이 동작되는 SRAM (32KB)로 구성된다.

론웍스 네트워크를 위해 뉴런칩인 TMPN3150을 사용하였고, 론웍스 네트워크를 위한 트랜스시버(Transceiver)로는 FTT-10A를 사용하였다. 이 트랜스시버는 트위스트 페어(Twist Pair)로 다른 론웍스 디바이스와 버스 구조에 의하여 연결된다. 또한 뉴런칩은 메모리 맵 I/O(Memory-mapped I/O)를 위한 외부 버스가 존재하고 있기 때문에 독자적인 프로그램 메모리(Flash)와 동작 메모리(SRAM)가 독자적으로 구현되어 있다. 또한 이 프로세서들간의 인터페이스를 위해 DPRAM을 사용하였으며, 메모리 구성은 다음에 설명하도록 하겠다.

그림 1은 전체 시스템 구성을 나타내고 있으며 배치 도면에서 보듯이 2개의 마이크로프로세서와 빌딩자동화와 산업용 제어장치를 통합할 수 있는 주변회로로 구성되어 있다.

2. 임베디드 리눅스 포팅

메모리 관리 유닛이 없는 프로세서를 지원하기 위하여 개발된 임베디드 리눅스로써 유시리눅스가 리눅스 콘소시엄에서 현재 개발되고 있다. 유시리눅스는 가상 메모리를 사용하지 않고 물리적 메모리를 직접 사용하는 플랫(Flat) 메모리 모델로 구현되었으며, MMU 기능을 배제시켰다. 일반 리눅스와 유시리눅스의 메모리 할당 방법에는 차이가 있다. 일반 리눅스의 경우 영역간의 메모리 충돌을 방지하기 위한 공간이 존재한다. 그러나 유시리눅스는 MMU를 사용하지 않기 이유로 메모리 공간이 물리적 공간과 일치한다. 따라서 프로그램 작성시 각각의 메모리 영역이 넘치지 않도록 프로그램머는 스스로 주의해야 한다[7].

리눅스 포팅은 커널 중에서 하드웨어에 의존하는 부분을 수정하여 원하는 임베디드 시스템에서 동작하도록 구현하는 것이다. 따라서 하드웨어 의존적인 부분과 하드웨어 독립적인 부분에 대한 구분에 의하여 포팅이 시작되며, 로우 레벨(Low Level) 포팅과 하이 레벨(High Level) 포팅으로 이루어진다[7,8]. 로우 레벨 포팅은 하드웨어 의존적인 부분으로 어셈블리어로 작성되며, 하이 레벨 포팅은 리눅스 커널과 관련된 부분으로 C언어로 작성된다. 멀티 프로토콜 제어기 또한 이러한 과정을 거쳐 임베디드 리눅스로 포팅하고, 이더넷과 RS-485를 위한 디바이스 드라이버를 작성하였다. 그림 2는 유시 리눅스의 부팅 순서를 나타낸다. Startup Code는 로우 레벨 포팅에 의해서 작성된 것이고, start_kernel과 이것과 연결된 부분은 하이 레벨 포팅에서 작성된다.

3. 프로세서간의 인터페이스

멀티 프로토콜 제어기는 2개의 프로세서로 구성되어 있기 때문에 이들 간의 데이터를 인터페이스 할 수 있는 방법이 필요하다. 프로세서간의 인퍼페이스하는 방법으로는 시리얼 통신이나 패러럴 통신 또는 DPRAM을 이용하는 방법이 있는데, 이 중 DPRAM을 이용하는 방법을 선택하였다. DPRAM은 세마포어와 인터럽트를 지원하기 때문에 동시 쓰기 등의 문제로 인해 데이터가 손상되는 것을 방지할 수 있으며, 포로세서에서 불필요하게 DPRAM에 저장된 데이터의 변경 여부를 확인할 필요가 없다. 본 논문에서는 사이프레스(Cypress)의 CY7C136 DPRAM을 이용하였다. 이 램은 2K*8 크기를 가지며, Left Side와 Right Side로 구성되어 있다. 각각의 Side는 Busy와 인터럽트(Interrupt) 신호가 있다. Left Side나 Right Side에서 데이터를 저장하면 반대편에 인터럽트 신호를 만들어 주어야 한다. 인터럽트 신호는 Left Side의 경우 0x7FF 번지에 Write 신호를 내보내면 된다. 그러면 Right에 인터럽트 신호가 발생한다. 인터럽트 신호를 클리어하기 위해서는 Right Side에서 0x7FF번지에 Read 신호를 보내주면 된다. 반대의 경우 0x7FE 번지를 쓰거나 읽으므로써 인터럽트를 발생시키고 클리어시킬 수 있다.

그림 3은 전원이 인가되었을 때 다중 프로토콜 제어기의 메모리 맵을 나타내고 있다. S3C4530A 프로세서에는 DPRAM이 0x03640000 ~ 0x036407FF에 위치하며 TMPN3150 프로세서는 0x8000 ~ 0x87FF에 위치한다. 즉 두 시스템은 상대방의 존재를 모르며 단지 DPRAM에 쓰고 그리고 상대방에게 인터럽트를 발생시켜 정보를 전달할 뿐이다. 본 논문에서 각각의 정보에 따른 DPRAM의 메모리 맵을 그림 3에 표시하였다. 이것을 통하여 론웍스 디바이스에게 명령을 내리고 론웍스 디바이스의 상태값을 읽어 온다.

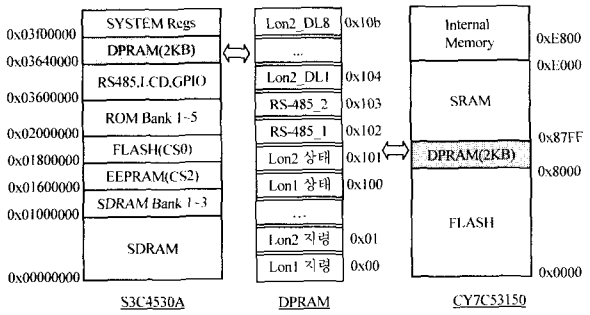


그림 3. 멀티 프로토콜 제어기의 메모리 맵.
Fig. 3. Memory map of multi-protocol controller.

Command (read/write) 1Byte	Protocol 1Byte	Node (Controller) 1Byte	Function 1Byte	Data 8Byte	'₩n' 1Byte

그림 4. 서버와 클라이언트 통신을 위한 프로토콜.
Fig. 4. Protocol for communication between client and server.

III. 시스템간의 통신

1. MMI와 다중 프로토콜 제어기간의 통신

다중 프로토콜 제어기는 상위 제어기와 TCP/IP를 통해서 통신한다. 따라서 다중 프로토콜 제어기는 서버가 되며, 상위 제어기는 클라이언트로서 데이터 통신이 이루어진다. 서버와 클라이언트가 서로 통신하기 위해서는 이들 간의 데이터 패킷 규약이 필요하기 때문에 본 연구에서는 그림 4와 같이 프로토콜을 만들었다.

프로토콜에서 *Command*는 로컬 디바이스에 명령('1')을 내리거나 상태값을 요구('0')하기 위해 사용된다. *Protocol*은 다중 프로토콜 제어기에서 어떤 통신 방식의 로컬 제어기를 제어할 것인가를 나타낸다. 본 연구에서는 '0'을 론웍스 디바이스로, '1'을 RS-485 디바이스로 설정하였다. *Node*는 각 통신 방식에 따른 로컬 디바이스의 ID를 나타낸다. Lon#1을 '0', Lon#2를 '1', RS-485 디바이스는 '0'으로 설정하였다. *Function*은 각 로컬 디바이스가 갖는 기능을 정의한다. *Data*는 로컬 디바이스에 전달하는 데이터 값이다. 마지막으로 '\n'은 데이터 패킷의 끝을 나타낸다.

2. 로컬 제어기와의 통신

론웍스 디바이스간의 통신 방법으로는 네트워크 변수(Network Variable)과 네트워크 태그(Network Tag)가 있다. 본 논문에서는 이 중 네트워크 변수를 사용하였다. 네트워크 변수를 론웍스 디바이스에 연결하기 위해서는 바인딩(binding)이라는 작업이 필요하다. 바인딩은 론웍스 네트워크로 론웍스 디바이스를 연결하는 작업니다.

멀티 프로토콜 제어기는 그림 5와 같이 네트워크 변수를 바인딩하였다. 바인딩 구조를 *ARMLON*노드와 *Lon1*노드 간의 연결을 예를 들어 설명하겠다. *ARMLON*의 *lons1*과 *Lon1*의 *lonr1*은 출력 네트워크 변수이고 *ARMLON*의 *lonr1*과 *Lon1*의 *lons1*는 입력 네트워크 변수이다. 이러한 바인딩 통하여 출력 네트워크 변수와 입력 네트워크 변수 간의 통신이 이루어지며, 결과적으로 분산 제어를 위한 데이터 통신이 이루어지게 된다. 통신 프로토콜은 론웍스 개발 장비인

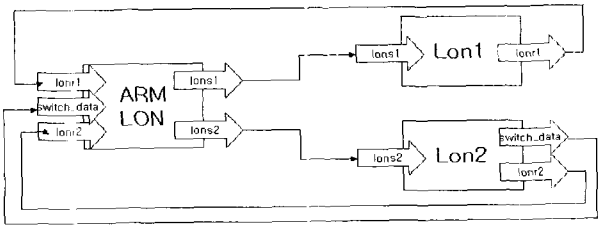


그림 5. 론웍스 디바이스간의 네트워크 바인딩.
Fig. 5. Network binding for LonWorks devices.

시작프레임 '~'	소스 어드레스	목적 어드레스	데이터 사이즈	데이터..	끝 프레임 '₩0'

그림 6. RS-485 제어기를 위한 데이터 패킷 포맷.
Fig. 6. Data packet format of RS-485 controller.

론빌더(LonBuilder)에 의하여 내장되게 된다. 따라서 개발자는 바인딩에 의하여 프로토콜의 구현 없이 제어 노드간의 데이터 통신을 쉽게 구현할 수 있다.

RS-485 통신 방식의 로컬 제어기는 그림 6과 같은 데이터 패킷 포맷으로 데이터를 주고받는다. 이와 같은 방식을 통하여 소스와 목적 노드에 대한 데이터 통신이 이루어지며, 다중으로 연결된 노드에 대하여 필요한 노드와의 통신을 수행하게 된다.

3. 서버 프로그램

본 연구에서 임베디드 리눅스 기반의 다중 프로토콜 제어기 안에 내장된 서버 프로그램은 일반 리눅스에서 사용되는 메아리 서버를 응용한 것이다. 이것을 그대로 크로스 컴파일(Cross compile)을 하면 에러가 발생한다. 유시리눅스와 일반 리눅스의 차이로 인한 것이다. 이 서버 프로그램은 단지 fork()를 vfork() 함수로 대치함으로써 프로세스 생성 대신에 쓰레드 생성으로 프로그램을 사용할 수 있다. 이처럼 리눅스는 서두에서 기술했듯이 많은 소스가 오픈되어 있으며 임베디드 리눅스와 일반 리눅스의 차이를 잘 알고 있으면 공개된 소스를 약간 수정함으로써 임베디드 리눅스 시스템에 적용할 수 있다.

그림 7은 이 서버가 규약한 메시지를 받았을 때 다중 프로토콜 제어기가 수행하는 부분을 나타낸다. 서버 프로그램이 클라이언트로부터 메시지를 받으면, 가장 먼저 *Command*를 체크한다. 체크한 값이 '0'이면 DPRAM에 저장된 상태값을 읽어서 클라이언트로 보내고, '1'이면 *Node*를 비교하여 '0'이면 론웍스 디바이스에 *Function*과 *Data*를 보내고, '1'이면 RS-485 디바이스에 *Function*과 *Data*를 보낸다.

IV. 실험

1. 실험 장치

본 연구에서 모니터링을 위해 기본의 빌딩자동화시스템에 존재하는 그래픽 에디터 빌더를 이용하여 구현된 자동화 시스템을 위한 GUI(Graphic User Interface)를 모델링 하였다[10]. 그림 8은 일반적인 MMI 프로그램의 구조를 나타내며 본 연구에서 사용되는 MMI 프로그램도 이러한 구조로 이

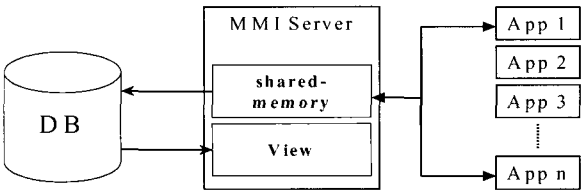
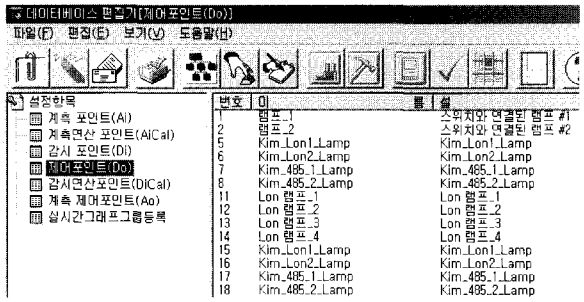
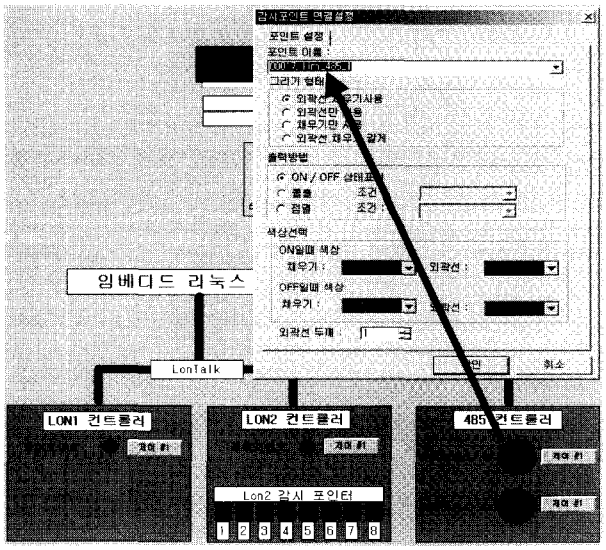


그림 8. MMI Server 구조.
Fig. 8. Stucture of MMI Server.



(a) 제어 노드의 BAS에서 데이터 베이스 연결



(b) GUI와 DB의 제어 포인트 연결

그림 9. 제어 노드 설정을 위한 MMI 설계.
Fig. 9. Desinging of MMI to handle control nodes.

루어져 있다. MMI Server는 전체적인 시스템을 관리하는 프로그램이다. 즉, 실제 기능은 등록된 어플리케이션에서 하며, MMI Server는 어플리케이션과 데이터를 공유 메모리를 통해서 데이터를 인터페이스한다. 그리고 공유 메모리에 저장된 데이터를 데이터 베이스에 저장하고 화면에 나타내는 기능을 한다.

어플리케이션 등록 방법을 간단히 살펴보면 첫 번째로 그림 9의 (a)와 같이 시스템의 입출력 포인터에 대한 정보를 데이터 베이스에 저장한다. 두 번째로 MMI 뷰(view)의 부분을 정의하는 그래픽 에디터를 이용하여 입출력에 사용할 포인터의 인터페이스를 구성한다. 구성된 입출력 포인터를 등록함으로써 이것들의 값이 변경되었을 때 데이터 베이스에 변경된 값이 갱신되고 MMI Server의 뷰에 결과 값이 나타난다. 세 번째로 공유 메모리를 생성해주는 프로그램을 작성하여 공유 메모리의 순서와 데이터 베이스의 인덱스 번호를 서로 연결해 준다. 이 같은 작업을 해주면 공유 메모리에 저장된 값이 데이터 베이스의 해당 포인터에 저장된다.

본 연구에서도 MMI 기반의 제어 모니터링을 하기위해 각각의 제어기에 따른 포인터를 데이터 베이스 저장하고 MMI 화면 구성하였다. 또한 다중 프로토콜 제어기의 서버 프로그램과 통신하기 위해 MMI 방식의 인터페이스가 추가된 TCP/IP 클라이언트 프로그램을 작성하였다.

2. 빌딩자동화시스템과의 연동 실험

그림 10은 본 논문에서 구현한 전체 시스템을 나타낸다. 그림에서 BAS는 상용화된 장비를 이용하였으며, 다중 프로토콜 제어장치는 임베디드 리눅스와 SoC인 S3C4530A로 구현하였다. 그리고 로컬 제어장치로 산업용 제어장치인 모터를 위한 RS-485 다중 통신 장치와 빌딩자동화시스템을 위한 론웍스 제어 노드를 구현하였다. 실험에서는 간단히 2개씩을 적용하였지만 버스 구조이기 때문에 물리적인 제한 범위 내에서는 계속 확장이 가능하다.

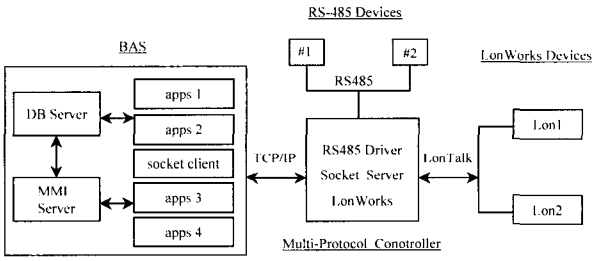


그림 10. BAS의 구성도.
Fig. 10. Simple BAS for experiment.

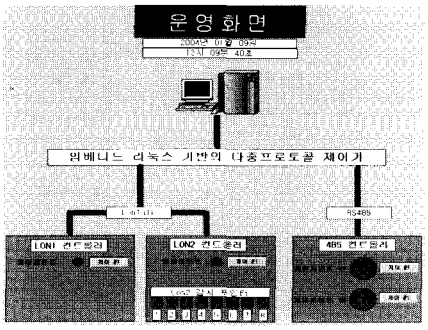


그림 11. 설계된 MMI 화면.
Fig. 11. Designed MMI for experiment.

이상과 같이 구성한 시스템을 중앙 제어 컴퓨터의 데이터 베이스에 제어 포인트를 등록해야 한다. 본 실험에서는 4개의 디지털 출력 포인터와 12개의 디지털 입력 포인터(론 포인트 2개, Lon#2 DI 채널 8개, RS-485 포인트 2개)구성하였다. 그 다음으로 MMI의 화면을 디자인하고 해당 감시 포인터를 데이터 베이스의 등록된 입력 포인터와 연결한다.

표 1. Lon2의 DI 채널에 의해서 동작하는 다른 로컬 제어기.
Table 1. Other local controller handled by DI Channel of Lon2.

Channel	Object of controlment
DI_1	DO control of Lon1
DI_2	DO1 control of RS485 Controller
DI_3	On/Off of Motor 1 (Low Speed)
DI_4	On/Off of Motor 2 (Low Speed)
DI_5	Speed Switch of Motor 1 (Middle)
DI_6	Speed Switch of Motor 1 (High)
DI_7	Speed Switch of Motor 2 (Middle)
DI_8	Speed Switch of Motor 2 (High)

그리고 지령을 내리기 위한 MMI 화면에 지령 버튼을 넣고 데이터 베이스의 등록된 출력 포인트와 연결한다. 디자인된 MMI 화면은 그림 11과 같다.

위와 같이 MMI 디자인 및 설정이 끝나면 다음으로 해야 할 것이 TCP/IP 방식의 클라이언트 프로그램을 작성이다. 이 프로그램을 작성할 때 MMI 뷰와 연동하기 위해서 다음과 같은 기능을 넣어야 한다. 받은 데이터를 해당 공유 메모리에 쓰는 인터페이스와 MMI의 제어 버튼을 눌렸을 때 발생하는 해당 레지스터 윈도우 메시지 핸들러 함수를 작성한다. 또한 주기적으로 상태를 요청하는 부분이 필요하기 때문에 타이머 함수를 이용한다.

동작의 예를 기술해 보겠다. 모든 시스템에 전원이 인가되면 DPRAM에 로컬 제어기의 초기 상태값이 저장된다. 그럼 클라이언트는 다중 프로토콜 제어기의 서버 프로그램에 접속하여 설정한 주기(1sec)마다 로컬 제어기의 상태값을 가져온다. 이 값을 클라이언트 프로그램은 데이터 베이스에 등록된 제어 포인트의 인덱스 번호와 매핑되어 있는 공유 메모리에 상태값을 저장함으로써 해당 제어기의 상태를 MMI 뷰에 나타낸다.

이번에는 지령 버튼을 눌렸을 때의 동작을 살펴보겠다. Lon#2를 예를 들어 설명하겠다. Lon#2의 제어 버튼을 누르게 되면 레지스터 윈도우 메시지가 발생하여 등록된 통신 프로세서(본 실험에서는 TCP/IP 클라이언트)에게 데이터 베이스의 인덱스 번호와 지금 상태의 반대 값(디지털 포인터의 경우)을 전달한다. 그럼 본 실험에서 설계된 해당 패킷의 위치에 전달받은 값을 저장하고 다중 프로토콜 제어기의 서버 프로그램에게 패킷을 전달한다. 서버 프로그램은 자신이 가지고 있는 로컬 제어기의 상태값과 비교하여 어떤 로컬 제어기에게 지령을 내릴지를 판단한다. 본 예에서는 Lon#2 이므로 서버 프로그램은 Lon#2의 지령 상태를 저장하는 DPRAM의 특정 어드레스에 지령 값을 저장하고 다중 프로토콜 제어기는 뉴런칩에 인터럽트를 걸어준다. 그러면 프로세서는 자신과 바인딩된 론웍스 디바이스의 상태 버퍼와 DPRAM에 저장된 값을 비교하여 어떤 론 디바이스에게 지령을 내릴지를 결정한 후, 해당 네트워크 변수에 값을 입력한다. 네트워크 변수에 입력된 값을 전달 받은 Lon#2 디바이스는 그 네트워크 변수의 값에 대한 자신의 일(미리 프로그램된)을 하게 된다.

본 실험에서 설정한 동작은 LED를 On/Off하고 실제 On/Off 값을 입력받아 다시 다중 프로토콜 제어기의 뉴런

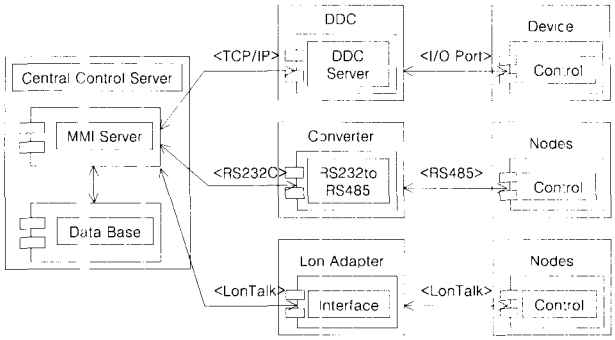


그림 12. 중앙 제어 컴퓨터 기반의 BAS 구조도.
Fig. 12. Component diagram of a BAS based on a Central Control Computer.

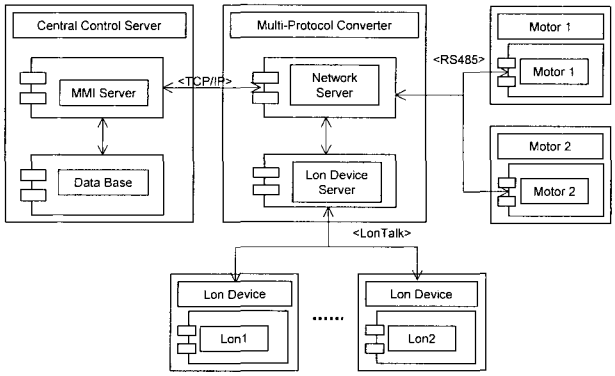


그림 13. 멀티 프로토콜 제어기를 적용한 BAS의 구조도.
Fig. 13. Component diagram of the proposed BAS using multi protocol controller.

칩에게 자신의 상태를 너트워크 변수를 통해 알리는 것이다. 전달받은 뉴런 칩은 자신과 바인딩된 로컬 제어기의 상태 버퍼와 DPRAM의 값을 업데이트하고 그 값을 다시 클라이언트에게 전달한다. 그 이후의 동작은 앞에서 기술한 것과 같다. RS-485 통신 방식의 로컬 제어기도 이와 비슷하게 동작하여 제어 및 자신의 상태값을 할당된 DPRAM에 저장하게 된다.

본 실험에서 Lon#2의 8개의 DI 채널 상태에 따라 다른 로컬 제어기를 제어하고 모니터링할 수 있는 운영 프로그램이 TCP/IP 클라이언트 프로그램 내에 구현되었다. 그 동작은 테이블 1에 정리하였다.

V. 결론

본 연구에서 임베디드 리눅스 기반의 산업용 다중 프로토콜 제어기 개발과 빌딩자동화시스템에 연동적용하였다. 지원하는 프로토콜로는 RS-485와 론웍스를 설정하고, 그것을 지원할 수 있는 시스템 보드를 설계하여 구현하였다. 그리고 기존에 많이 사용되던 RTOS(Real-Time Operating System)가 아닌 임베디드 리눅스를 채택함으로써 자동화 분야에 임베디드 리눅스가 사용 가능함을 보였고, 더 나아가 개발시 경제적 부담을 감소할 수 있다는 것을 보였다.

그림 12와 같은 구조를 가지고 있는 기존의 빌딩자동화

시스템은 중앙 제어 컴퓨터에 각각의 통신 방식에 따른 제어기가 연결되어 구조가 복잡해지고 고성능의 컴퓨터가 필요함으로 비용이 증가되는 단점이 있다. 이에 반해 그림 13에서와 같이 다중 프로토콜 제어기를 사용할 경우는 여러 통신 방식의 제어기를 분산 처리할 수 있을 뿐만 아니라 TCP/IP를 지원함으로써 원격의 중앙 제어 컴퓨터로부터 제어를 받을 수 있기 때문에 군관리 시스템에 사용될 수 있는 장점이 있다.

참고문헌

[1] I. Bowman, S. Siddiqi, and M. C. Tanuan, Concrete Architecture of the Linux, 1998.

[2] A. Lennon, "Embeddeding linux", *IEE Review*. pp. 33-37, May 2001.

[3] http://www.echelon.com/products/lonworks/default.html, "Introduction to the lonworks system"

[4] http://www.arm.com/armtech/ARM7TDMI OpenDo cument

[5] http://www.samsung.com/Porducts/Semiconductor/SystemLSI/Networks/PersonaalNTASSP/CommunicationProcessor/S3C4530A.html, S3C4530A Use r's guide

[6] TOSHIBA Co., TAEC Neuron Chip Databoot, pp. 18-20, 2001.

[7] http://www.uclinux.org/description

[8] B. W. Choi, K. C. Koh, J. L. Mun, and G .Y. Im, Embedded Linux, Hong-Reung, Seoul, pp. 119-120, 2002.

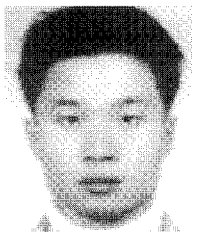
[9] Cowell SysNet Co. Ltd., Integrated Management System Manual, pp 14-50, 2002.

최 병 욱

제어 · 자동화 · 시스템공학 논문지 제 10 권 제 4 호 참조.

신 은 철

제어 · 자동화 · 시스템공학 논문지 제 10 권 제 4 호 참조.



김 현 기

2001년 2월 선문 대학교 공과대학 제어계측학과 졸업. 2003년 7월 동교 공과대학원 시스템 및 제어 전공 졸업. 현 세연테크 연구원 근무, 관심분야 마이크로프로세서 응용, 임베디드 시스템, 임베디드 리눅스, 실시간 운영체제.