論文2003-40CI-5-8

리눅스 상에서 실시간 태스크에 의한 소프트웨어 PLC의 구현

(An Implementation of Software PLC Based on Real-Time Tasks of Linux)

金 龍 碩 *

(Yong-Seok Kim)

요 약

본 논문에서는 소프트웨어 PLC를 구현하는 방안을 제시하였다. 하드웨어는 표준적인 산업용 PC와 입출력 보드들로 구성된다. 운영체제로는 수십 마이크로초의 정밀한 단위로 태스크들의 스케줄링이 가능한 실시간 리눅스를 사용하였다. 소프트웨어 PLC를 위해 개발한 실시간 태스크들은 실시간 리눅스의 커널 모드에서 실행되며, 사용 목적에 따라 작성된 PLC 프로그램을 해석하고 실행하는 작업을 일정한 주기로 반복한다. 또한 PLC 프로그램을 편리하게 작성할 수 있도록 자체 프로그램 문법을 제안하고 이를 위한 컴파일러도 구현하였다. 이렇게 구현된 PLC는 가격 대비 성능이 우수하며 소규모의 응용분야들에 유용하게 활용할 수 있다.

Abstract

This paper presents an implementation of software PLC. The hardware is based on off the shelf industrial PC's and input/output boards. The operating system is real-time Linux which can schedule tasks in tens of microseconds. The tasks for the software PLC are running in kernel mode of real-time Linux, and interpret and execute application PLC programs periodically. In addition, a simple PLC programming language is presented for the software PLC, and its compiler is implemented. This approach is cost-effective and useful for small PLC's.

Keywords : PLC, Linux, Real-Time Task.

Ⅰ. 서 론

PLC(Programmable Logic Controller)는 1960년대 말에 GM사의 요구에 의해 출현한 이래 각종 자동화 시스템을 구현하는데 있어서 없어서는 안될 핵심 기기로 자리잡고 있다. PLC의 기본 동작은 주기적으로 입력 장치들로부터 데이터를 읽어 들이고, 프로그램 메모리에 저장된 프로그램을 순차적으로 실행하면서 출력 데이터를 결정하고, 주기의 마지막에 그 결과를 출력장치들에 내보내는 과정을 반복한다[1, 3]. PLC의 프로그래밍 방법에 대한 국제적인 표준화 노력이 이루어져 왔지만, 현재 대부분의 PLC 제조업체는 독자적인 하드웨어 규격 및 프로그램 개발 환경을 제공한다.

최근 여러 업체에서 산업용 PC를 기반으로 하여 각종 입출력 유닛을 장착하여 PLC를 구현함으로써 개발기간 단축 및 저가격을 실현하고 있다. Schleicher 사의 ProSycon은 PC를 기반으로 하여 PLC 기능을 구현한 것으로서 Windows NT 하부에 Wind River Systems 사의 실시간 운영체제인 VxWorks를 사용하고 있다[4].

* 正會員, 江原大學校 電氣電子情報通信工學部
(Division of Electrical and Computer Engineering, Kangwon National University)

※ 이 논문은 강원대학교 두뇌한국21사업에 의하여 지원되었음.

接受日字:2002年8月21日, 수정완료일:2003年8月14日

PLC 기능을 수행하는 부분은 VxWorks 상에서 실행하고 Windows NT는 사용자 접속과 통신 등을 담당한다. 입출력 장치는 직접 PC에 장착하지 않고 CANopen 필드버스를 지원하는 장치들과 통신망을 통하여 연결한다.

ProSycon은 상용 운영체제를 사용하는데 반해서 공개된 운영체제인 Linux를 활용하여 PLC 기능을 구현하기위해 추진되고 있는 것으로서 PuffinPLC(일명 MatPLC)가 있다[5]. 이것은 PLC 입출력을 위한 프로그램 모듈들을 별도의 리눅스 프로세스들로 구현하고 이들 간에 공유메모리를 통하여 데이터를 교환한다. 리눅스 프로세스는 스케줄링 지연시간이 수십 밀리초 이상으로 길어질 수 있으므로 이 시간 이상의 느린 속도로 동작해도 무방한 응용 분야에 한해서 PuffinPLC를 사용할 수 있다.

본 논문에서는 산업용 PC에 각종 입출력 모듈들을 장착한 형태의 하드웨어를 사용하고, 리눅스 운영체제의 커널 모드에서 실행되는 실시간 태스크들에 의해 PLC 기능을 실행하도록 구현한 소프트웨어 PLC인 RTSP(Realtime Task Software PLC)에 대해 기술한다. PuffinPLC는 수십 밀리초 이하 단위의 동작속도가 필요한 부분에는 적용할 수가 없었던데 반해서, RTSP는 커널 모드에서 실행되는 실시간 태스크로 구현함으로써 수십 마이크로초 단위의 정밀한 단위로 스케줄링이 가능하다.

PLC 프로그램을 리눅스 상에 구현하기 위해서는 일차적으로 항상 일정한 주기로 반복적으로 실행되는 실시간 태스크를 지원해야 한다. 이를 위해 본 논문에서는 리눅스의 커널 모드에서 실시간 태스크들을 실행할 수 있도록 지원하는 실시간 리눅스를 기반으로 하여 PLC 기능을 수행하는 실시간 태스크들을 구현한다. 실시간 태스크 실행이 지원되는 리눅스의 예로는 New Mexico Tech. 의 RTLinux를 비롯하여, 이탈리아 DIAPM 대학의 RTAI, Kansas 대학의 KURT, UC Irvine의 RED Linux등이 있다. 상용 버전으로는 FSMLabs 사의 Open RTLinux, Lineo사의 Embedix Realtime, MontaVisa사의 HardHat Linux, TimeSys사의 Linux/Real-Time, LinuxWorks 사의 BlueCat RT, REDSonic 사의 REDICE-Linux 등이 있다[6].

일반적인 리눅스는 기본적으로 실행할 프로세스들을 전체적으로 빨리 완료할 수 있도록 처리하면서 프로세스들에게 공평하게 시간을 할당하는 방식을 사용하므로 실시간 처리 능력과는 거리가 멀다. 리눅스에 실시간 기능을 제공하는 방법으로는 대체로 다음의 두 가지 방식중 하나를 사용한다. 하나는 리눅스 커널에 별도의 실시간 커널을 부가하여 커널 모드의 실시간 태스크들을 최우선으로 실행하고, 남는 시간을 활용해서 일반적인 리눅스 프로세스들을 실행하는 방법이다[7, 8]. RTLinux나 RTAI가 여기에 해당하는 것으로서 실시간 기능을 기존의 리눅스에 비교적 쉽게 구현할 수 있고 수십 마이크로초 단위의 정밀한 스케줄링이 가능한 장점이 있다. 반면에 이들은 실시간 태스크 프로그램이 커널 모드에서 작동하므로 응용 프로그램에 오류가 있으면 시스템 전체가 정지되는 위험을 내포하고 있다.

다른 하나의 방법은 기존의 리눅스 소스코드를 전반적으로 수정하여 실시간 태스크들이 대기하는 시간을 줄이는 방법으로서 HardHat Linux나 Linux/Real-Time, 그리고 최근의 리눅스 커널 버전 2.5에서 시도하는 방법이다. 이 방법은 일반적인 리눅스 프로세스들처럼 실시간 프로세스들의 오류에 대해서도 시스템 전체에 문제가 발생하지 않도록 보호할 수 있고 실시간 응용 프로그램에서도 다양한 리눅스의 기능들을 그대로 활용할 수 있는 장점이 있다. 반면에 새로운 리눅스 커널 버전에 대해 실시간 기능을 추가하는 일이 아주 복잡한 작업이 되고 스케줄링 시간 단위도 수 밀리초 이상으로서 정밀한 스케줄링을 제공할 수 없다.

본 논문에서는 커널 모드 실시간 태스크를 지원하는 RTLinux를 기반으로 하여 소프트웨어 PLC인 RTSP를 구현함으로써 수십 마이크로초 단위의 정밀한 스케줄링이 가능하도록 하였다. PLC를 특정 목적에 사용하고자 하는 일반 사용자들은 운영체제 내부에 대한 지식이 충분하지 않고, 따라서 RTLinux의 커널 모드에서 실행되는 태스크들의 프로그램을 직접 작성하게 되면 프로그램 오류에 의해 전체 시스템에 심각한 문제를 야기할 수도 있다. RTSP를 사용하면 일반 사용자들은 커널 모드 태스크의 프로그램을 직접 작성하지 않고 래더 다이어그램과 유사한 형식으로 PLC 프로그램을 작성한다. 이 PLC 프로그램은 RTSP에서 제공하는 커널 모드 태스크가 해석하여 실행한다.

PLC 프로그램 작성을 위해서는 전통적으로 래더 다이어그램을 주로 사용해 왔는데 IEC(International Electrotechnical Commission) 에서는 PLC 프로그램들의 호환성을 높이기 위해서 프로그래밍 언어에 대한 표준을 제시하고 있다[2]. 표준에 포함된 프로그래밍 언

어로는 명령어 목록 (instruction list), 래더 다이어그램 (ladder diagram), 기능 블록 다이어그램(function block diagram), 및 구조화 텍스트(structured text) 등이 있다. 본 논문에서는 래더 다이어그램 개념과 유사한 형태의 프로그래밍언어를 고안하여 활용한다.

Ⅱ. PLC의 기본 구성

PLC는 "논리연산, 순서조작, 타이머, 카운터 및 산술연산 등의 제어동작을 실행시키기 위해 제어 순서를 일련의 명령어 형식으로 기억하는 메모리를 가지고, 이 메모리의 내용에 따라 기계와 프로세스의 제어를 디지털 또는 아날로그 입출력을 통하여 행하는 디지털 조작형의 공업용 전자장치"로 정의된다[3]. 기본적인 구성은 응용 목적에 따라 사용자가 작성한 PLC 프로그램을 순차적으로 읽어 들여 실행하는 CPU 유닛, ON/OFF 신호를 입력하거나 출력하는 디지털 입출력 유닛, 아날로그 데이터를 처리하는 아날로그 입출력 유닛, 및 기타 특수 목적에 사용되는 특수 유닛들로 구성된다.

내장 프로그램 방식의 PLC는 일반적으로 순차제어 프로그램을 메모리에 저장하여 두고 프로세서가 이를 차례대로 읽어서 해석하고 실행하는 방식을 따른다. 프로그램 메모리에 저장된 프로그램은 일정한 주기로 첫 프로그램 스텝부터 마지막 스텝까지 순차적으로 실행하고 맨 끝 스텝을 실행하고는 다시 처음으로 돌아가는 방식을 취한다. 매 주기마다의 실행 내용은 먼저 입력 유닛들로부터 접점상태를 읽어 들여 일정한 장소에 저장한 다음 이를 바탕으로 첫 스텝부터 끝까지 순차적으로 명령을 수행한다. 이 과정에서 출력 유닛을 위한 내용들을 일정한 메모리에 저장한 다음 주기의 마지막에 이 내용들을 출력 유닛에 출력한다. 이러한 한 주기 동안의 일련의 작업 과정을 스캔이라고 한다.

PLC 프로그램의 매 스텝마다 실행되는 명령어는 사용자가 시퀀스 프로그램을 작성할 때 사용하도록 PLC 메이커에서 정의한 명령어로서 PLC 메이커 마다 명령어 집합이 다를 수 있다. 그러나 명령어의 형태만 상이할 뿐이지 기능은 대동소이 하다. 명령어의 종류는 통상 두 가지로 나눈다. 기본 시퀀스 명령어는 릴레이 회로의 시퀀스 전개도와 같이 시퀀스를 PLC 내에서 실현하기 위한 명령어로서 제어신호의 입출력 명령어와 AND나 OR와 같은 기본 논리 연산 명령어로 구성되어 있다. 응용 명령어는 전송, 사칙연산, 비교, 데이터 형식의 변환과 같은 고도의 시퀀스제어 프로그램을 작성할 수 있도록 다양한 기능을 제공한다.

데이터 메모리는 비트 데이터와 워드 데이터로 구별된다. 신호의 입출력을 위한 메모리는 프로그램 내부에서 특정한 부호로 정해지고, 메모리 영역이 할당된다. 이들은 스위치, 센서, 액추에이터, 타이머 및 카운터 등의 ON/OFF에 대응하는 비트 데이터이다. 워드 데이터로는 PLC 프로그램에서 사용하는 내부 데이터 변수나 타이머/카운터의 설정치 및 현재치 등을 저장한다. 비트 데이터도 내부의 가상적인 릴레이처럼 사용되는 비트 변수로 사용할 수 있다.

타이머는 입력조건이 성립되면 작동을 시작하여 설정값 또는 0에 도달하면 접점 출력이 ON 상태로 된다. 타이머의 모드로는 한번 조건이 성립되면 계속 ON 상태를 유지하는 모드와, 조건이 만족되지 않으면 OFF 상태로 전환되는 모드가 존재한다. 디지털 입출력 유닛은 신호 방식에 따라 다양한 형태가 있지만 본 논문에서는 적절한 변환기를 거쳐서 TTL 전압으로 입출력되는 것을 전제로 한다. 아날로그 입출력 유닛은 CPU 유닛에 의해 직접 제어되고 A/D 변환기나 D/A 변환기 역할을 한다. 고급 기능의 아날로그 유닛의 경우에는 공유메모리를 통하여 CPU 유닛과 데이터를 교환하기도 한다. 본 논문에서는 이 두 가지 경우를 모두 지원하도록 한다.

PLC에 사용되는 특수 유닛들로는 PID 제어에 사용되는 PID 유닛, PLC의 스캔 시간보다 고속의 계수 작업에 사용되는 고속 카운터 유닛, 서보 모터나 스테핑 모터에 의한 정밀한 모터의 제어를 위한 위치 결정 유닛, 다른 기기와의 통신을 위한 통신 유닛 등이 있다. 본 논문에서는 이러한 유닛들도 아날로그 입출력 유닛과 마찬가지로 워드 데이터로 처리되는 유닛으로 간주하고, 통신 유닛은 필드버스와 같은 별도의 모듈이 있거나 대부분의 PC에서 기본으로 지원되는 이더넷 카드가 장착되어 있다고 가정한다.

Ⅲ. RTSP의 태스크 구조

1. 실시간 리눅스와 실시간 태스크

실시간 리눅스 운영체제인 RTLinux는 대부분의 실시간 운영체제들과 같이 고정 우선순위에 기반을 둔 스케줄링을 채택하고 있으며 기존의 리눅스에 최소한의 수정만 가하면서도 실시간 태스크의 실행을 효과적

으로 구현하고 있다[7, 8]. 모든 실시간 태스크들은 커널 모드에서만 실행되며, 따라서 이들 태스크에 오류가 있으면 전체 시스템이 중단되는 위험을 내포하고 있다.

운영체제 커널은 그 실행이 인터럽트로부터 유발되며 따라서 RTLinux는 모든 인터럽트를 가로채서 받아들이고 실시간 태스크들을 자체 실시간 스케줄러에 의해 실행시키고 남는 시간에 기존의 리눅스 커널이 실행되도록 한다. 이 과정에서 인터럽트들 중에서 실시간 태스크들에 관련되는 것들은 직접 처리하고 그렇지 않은 것들은 내부에 보관했다가 기존 리눅스 커널로 넘겨준다. 따라서 RTLinux 입장에서는 기존의 리눅스 커널 및 모든 리눅스 프로세스들은 한 개의 낮은 우선순위의 태스크로 간주되어 처리되는 형태를 취한다. <그림 1>은 리눅스 커널에서 작동되는 RTLinux의 구성과 실시간 태스크 및 기존 리눅스와의 동작관계를 보여준다.

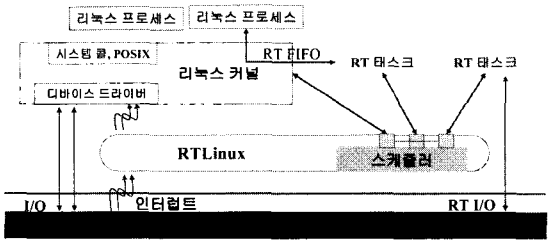

그림 1. RTLinux의 구성
Fig. 1. Structure of RTLinux.

2. RTSP의 구조

RTSP는 일반적인 산업용 컴퓨터에 디지털 입출력 보드와 필요에 따라 아날로그 입출력 보드 및 기타 특수 보드들을 장착하여 PLC 하드웨어를 구성하고, 실시간 리눅스의 커널 모드에서 실행되는 실시간 태스크인 PLC 태스크에 의해 이러한 장치들의 입출력 작업을 일반적인 PLC와 같은 방식으로 처리하도록 한다. 기본적인 입출력 장치로는 DI (Digital Input), DO (Digital Output), AD (Analog to Digital Conversion Input), DA (Digital to Analog Conversion Output) 등이 있다.

PLC 태스크는 RTLinux의 스케줄러에 의해 실행되는 주기적인 태스크이며, 전체적인 RTSP의 기본 구성은 <그림 2>와 같다. 사용자 모드의 리눅스 프로세스들은 RTSP를 위한 디바이스 특수파일을 통하여 커널 모드의 PLC 태스크와 정보를 교환한다. PLC 프로그램은 plcprog 디바이스 특수파일을 통하여 PLC 태스크로 전달하고, PLC 태스크는 정지 상태에서 실행 재개 상태로 변할 때 이 내용을 읽고 프로그램의 오류 여부를 검사한 후 내부의 프로그램 버퍼에 보관한다.

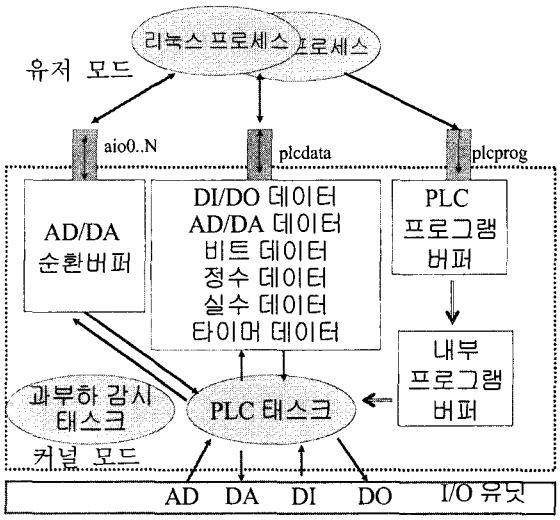

그림 2. RTSP의 기본 구성
Fig. 2. Basic Struture of RTSP.

디바이스 특수파일 plcdata의 버퍼에는 입출력 유닛과 내부 변수 데이터들에 대한 현재의 값을 유지하도록 한다. 디바이스 특수파일 aioN 의 AD/DA 순환 버퍼에는 AD 입력 데이터는 읽어 들인 순서대로 보관하였다가 리눅스 프로세스들이 읽어낼 수 있도록 하고, DA 로 출력할 데이터는 리눅스 프로세스로부터 받은 데이터를 보관했다가 순차적으로 출력한다. 입출력 데이터 외에 Bit, Integer, 및 Float 데이터는 PLC 프로그램에서 사용할 수 있는 변수들의 현재 값이며 각각 비트형, 정수형 및 부동소수점 데이터 형에 해당한다. Timer 데이터는 타이머들의 남은 시간을 나타낸다.

PLC 태스크는 사용자가 적재하는 정해진 형태의 PLC 프로그램을 주기적으로 수행한다. PLC 태스크의 기본 작업은 일반적인 PLC와 마찬가지로 DI 채널에서 데이터를 읽어 들이고, 프로그램에 따라 DO 출력을 결정한 다음 DO 채널로 데이터를 내보낸다. RTSP의 PLC 프로그램은 개념적으로 래더 로직과 유사한 내용을 텍스트 파일로 작성하여 컴파일한 다음 PLC 태스크에 적재된다. 여기에 타이머와 카운터를 사용할 수 있으며, DI 채널, DO 채널, AD 채널, DA 채널, 타이머, 및 카운터 등의 현재 값을 입력 조건으로 하여 PLC 프로그램을 작성할 수 있다.

PLC 태스크는 리눅스 프로세스에 의해 적절히 조정될 수 있다. 조정 내용은 실행 주기, 실행 모드, 실행 정지, 실행 재개 등이다. <그림 3>은 PLC 태스크의 실행 상태, 주기 등을 조정하고 현재 상태를 읽어내는 과정을 보여준다. 태스크들의 조정은 리눅스 디바이스 특수파일인 config를 통해 이루어진다. 태스크의 현재 상태 및 RTSP 전반적인 정보는 info 디바이스 특수파일을 통해서 리눅스 프로세스들이 읽어낼 수 있다.

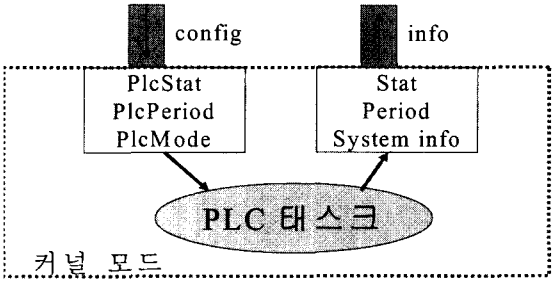

그림 3. PLC 태스크의 조정
Fig. 3. Configuration of PLC Task.

3. PLC 태스크

PLC 태스크는 RTLinux 상에서 실행되는 커널 모드 실시간 태스크로서 PLC 기능을 실행하는 핵심적인 역할을 한다. PLC 태스크의 실행 과정은 <그림 4>와 같다.

RTLinux는 태스크들의 실행 주기를 설정하여 주기적으로 실행하도록 할 수 있다. PLC 태스크는 프로그램 정지 모드에서는 기본 주기로 실행되면서 사용자로부터의 명령을 검사한다. 이 과정에서 프로그램 실행 모드로의 전환이 요구되면 사용자로부터 제공된 PLC 프로그램을 내부 메모리로 읽어 들여서 적재한다. 성공적으로 프로그램 적재가 끝나면 프로그램 실행주기를 읽어 와서 자신의 태스크 주기로 설정하고 상태를 실행모드로 전환한다. 반대로 실행 모드에서 정지 모드로의 전환이 요구되면 자신의 주기를 기본 주기로 전환하고 상태를 정지 모드로 전환한다.

PLC 프로그램은 읽어 들이는 과정에서 기본적인 프로그램 오류들을 검사하고 문제가 발견되면 프로그램 적재를 취소한다. 주된 오류로는 입출력 채널이나 타이머, 카운터 등의 번호가 허용된 범위를 벗어나는 경우와 타이머가 아닌 항목에 타이머 동작이 지정된 경우 등이다.

실행 모드가 계속 유지되는 상태에서는 PLC 타이머

```
plc_task_period = TASK_PERIOD_DEFAULT;
set task period to plc_task_period;
plc_task_stat = TASK_STAT_STOP;

while (1) {                    /* loop here forever */
 wait for the next period;
 read new_period and new_stat;

 if (plc_task_stat == TASK_STAT_STOP && new_stat == TASK_STAT_RUN) {
 read and update the PLC program;
 plc_task_period = new_period;
 set task period to plc_task_period;
 plc_task_stat = TASK_STAT_RUN;
 continue;
 }
 if (plc_task_stat == TASK_STAT_RUN && new_stat == TASK_STAT_STOP) {
 plc_task_period = TASK_PERIOD_DEFAULT;
 set task period to plc_task_period;
 plc_task_stat = TASK_STAT_STOP;
 continue;
 }
 if (plc_task_stat == TASK_STAT_RUN) {
 update the timers for the time interval of plc_task_period;
 read DI data from the digital input units;
 run PLC program step by step and store DO data;
 write DO data to the digital output units;
 }
}
```

그림 4. PLC 태스크의 실행 알고리즘
Fig. 4. Algorithm of PLC Task.

들의 값을 자신의 주기에 해당하는 값만큼 증가시키고 설정치에 도달하면 더 이상 증가하지 않는다. 그 다음에 디지털 입력 장치들로부터 데이터를 읽어 들여서 내부의 DI 데이터 버퍼에 기록한 후 내부에 적재된 PLC 프로그램을 한 스텝씩 읽어서 실행한다. 이 과정에서 DO 데이터들이 결정되고 변수들도 PLC 프로그램에 따라 적절히 수정된다. 마지막으로 DO 데이터들을 디지털 출력 장치들로 내보낸 후에 다음 주기가 시작될 때까지 기다린다.

아날로그 입출력은 PLC 태스크의 매 주기마다 실행하지 않고 PLC 프로그램에 표현된다. 즉 A/D 변환이나 D/A 변환을 위한 조건이 만족되면 이때에 실행된다. A/D 변환의 경우에는 읽어 들인 데이터는 내부의 순환 버퍼에 기록되며 동시에 AD 데이터 버퍼에 현재의 값으로 갱신된다. 순환 버퍼에 기록된 데이터는 리눅스 프로세스에 의해 순차적으로 읽어낼 수 있다. 반대로 D/A 변환의 경우에는 내부 순환 버퍼에 있는 데이터들을 차례대로 출력하고 동시에 DA 데이터 버퍼

에 현재의 값으로 갱신된다. 리눅스 프로세스는 출력될 데이터를 순환 버퍼가 비기 전에 계속 공급한다. 입력 데이터의 경우 순환 버퍼가 가득 차면 버리고, 출력 데이터의 경우 순환 버퍼가 비면 이전 값을 유지한다.

Ⅳ. PLC 프로그램 형식

1. PLC 프로그램 표현 문법

RTLinux의 커널 모드에서 실행되는 PLC 태스크는 주기적으로 PLC 프로그램을 한 스텝씩 읽어서 실행한다. PLC 프로그램은 내부에서 처리하는데 효율적인 방식으로 표현되지만 PLC 프로그램 작성자는 보다 친숙한 텍스트 파일로 작성하고 이를 위한 전용 RTSP 컴파일러로 컴파일하여 내부에서 처리하는 PLC 프로그램 코드를 생성한다. PLC 프로그램 문법은 기본적으로 C 언어와 유사한 형식을 따른다. 컴파일된 프로그램은 디바이스 특수 파일인 plcprog를 통하여 PLC 태스크에 적재한다. PLC 태스크는 정지 상태에서 실행 상태로 전환되는 시점에 이 프로그램을 내부 메모리로 읽어 들여서 실행한다.

PLC 프로그램에서 입력이나 출력 대상이 되는 데이터 항목으로는 표 1과 같은 6가지 그룹이 있다.

표 1. PLC 프로그램에서 사용하는 데이터 항목

Table 1. Data Types of PLC Program.

그룹이름	데이터	크기	내부처리	표시형식
DIO	DI, DO의 현재 데이터	32bit	비부호정수	D*n*
AIO	AD, DA의 현재 데이터	32bit	비부호정수	A*n*
BIT	비트형 변수	32bit	비부호정수	B*n*
INT	정수형 변수	32bit	정수	I*n*
FLOAT	실수형 변수	32bit	부동소수점	F*n*
TIMER	타이머의 경과시간 (msec)	32bit	정수	T*n*

DIO (Digital Input/Output): 디지털 입출력 데이터로서 DI의 경우 가장 최근에 읽어 들인 값이고 DO의 경우 가장 최근에 출력한 값이다. 이 그룹은 항목번호 0부터 DIO 채널 개수만큼 존재한다. PLC 프로그램 상에 그룹 항목 (GroupItem)으로 표시할 때에는 Dn (n은 채널 번호)으로 표시한다. 예를 들어 D0의 경우 DIO 그룹 채널 0의 데이터를 의미한다. 하나의 데이터 항목에는 최대 32개의 비트까지 포함될 수 있으며 PLC 태스크는 전체 32bit를 하나의 비부호 정수로 처리한다.

AIO (Analog Input/Output): 아날로그 입출력 데이터로서 AD의 경우 가장 최근에 읽어 들인 값이고, DA의 경우 가장 최근에 출력한 값이다. 이 그룹은 항목번호 0부터 AIO 채널 개수만큼 존재한다. PLC 프로그램 상에 그룹 항목으로 표시할 때에는 An (n은 채널 번호)으로 표시한다. 예를 들어 A2의 경우 AIO 그룹의 채널 2의 데이터를 의미한다. 하나의 데이터 항목에는 최대 32개의 비트까지 포함될 수 있으며 PLC 태스크는 전체 32bit를 하나의 비부호 정수로 처리한다. AD 채널은 읽어 들인 데이터를 AIO 항목에 갱신하고, 순환 버퍼에도 순차적으로 기록된다. DA 채널은 순환 버퍼에서 데이터를 가져와서 출력하고 출력한 결과를 AIO 항목에 갱신한다. 만약 순환 버퍼가 비어 있으면 현재의 값을 유지한다.

BIT: bit 단위의 변수로 사용할 수 있는 데이터로서 항목번호 0부터 BIT 그룹 항목 개수만큼 존재한다. PLC 프로그램 상에 그룹 항목으로 표시할 때에는 Bn (n은 항목 번호)으로 표시한다. 예를 들어 B1의 경우 BIT 그룹의 항목 1의 데이터를 의미한다. DIO 데이터와 마찬가지로 하나의 데이터 항목에는 32개의 비트를 포함하며 PLC 태스크는 전체 32bit를 하나의 비부호 정수로 처리한다.

INT (integer): 32bit 정수형 변수로 사용할 수 있는 데이터로서 항목번호 0부터 INT 그룹 항목 개수만큼 존재한다. PLC 프로그램 상에 그룹 항목으로 표시할 때에는 In (n은 항목 번호)으로 표시한다. 예를 들어 I0의 경우 INT 그룹의 항목 0의 데이터를 의미한다.

FLOAT: 32bit 실수 (부동 소수점)형 변수로 사용할 수 있는 데이터로서 항목번호 0부터 FLOAT 그룹 항목 개수만큼 존재한다. PLC 프로그램 상에 그룹 항목으로 표시할 때에는 Fn (n은 항목 번호)으로 표시한다. 예를 들어 F0의 경우 FLOAT 그룹의 항목 0의 데이터를 의미한다.

TIMER: 타이머 기능을 가진 데이터 그룹으로서 타이머가 작동을 시작한 후 현재까지의 경과시간을 밀리초 (msec) 단위로 표시한다. 중지된 타이머 항목의 값은 항상 -1이다. 항목번호 0부터 TIMER 그룹 항목 개수만큼 존재한다. PLC 프로그램 상에 그룹 항목으로 표시할 때에는 Tn (n은 항목 번호)으로 표시한다. 예를

들어 T0의 경우 TIMER 그룹의 항목 0의 데이터를 의미한다. PLC 태스크는 전체 32bit를 하나의 정수로 처리한다.

PLC 프로그램의 한 문장은 기본적으로 **if** 문을 사용하며 "**if** condition **then** action1 **else** action2 **fi**" 형식을 따른다. 이의 변형으로서 **ifbecome** 문이 있는데 "**ifbecome** condition **then** action1 **else** action2 **fi**"형식을 따른다. 주석 처리와 매크로 프로세서의 문법은 C 언어 형식을 그대로 따른다. **if** 문은 조건 condition이 참이면 **then** 다음의 동작 action1을 실행하고 거짓이면 else 다음의 동작 action2를 실행한다. **ifbecome** 문은 조건 condition이 거짓인 상태에서 참으로 바뀌는 시점에만 **then** 다음의 동작 action1을 실행하고 그렇지 않은 경우에는 **else** 다음의 동작 action2를 실행한다. **if** 대신 **init**를 사용하면 프로그램 실행 시작 시에 한번만 실행되고, **if** 없이 그냥 action만 표시하면 매 스캔마다 항상 실행된다. **if** 문의 **then**이나 **else** 속에 새로운 **if** 문이 계속하여 중첩될 수 있다.

PLC 프로그램의 전체 문법은 다음과 같다. 여기서 [ ] 속은 생략 가능한 부분을 의미한다. 이탤릭체는 함수나 변수의 이름을 의미하고 굵은체는 PLC 프로그램의 예약어를 의미한다.

program = init-statement, if-statement, 또는 action의 반복
statement-list = if-statement 또는 action의 반복
init-statement = **"init"** action
if-statement = **"if"** 또는 **"ifbecome"** condition **"then"** [statement-list] [**"else"** statement-list] **"fi"**
condition = condition-func "(" group-item "," data, ... ")"
action = action-func "(" group-item "," data, ... ")" ";"
condition-func = "IsBitOn", "IsBitAnyOn", "IsBitOff", "IsBitAnyOff", "IsBitEq", "IsBitNe", "IsLt", "IsLe", "IsEq", "IsNe", "IsGe", 또는 "IsGt"
action-func = "SetBitOn", "SetBitOff", "SetBitEq", "Set", "Add", "Sub", "Mul", "Div", "BitAnd", "BitOr", "BitXor", "BitNeg", "BitShl", "BitShr", "TimerOn", "TimerReset", "AioAdc", "AioDac" 또는 "FuncN"
group-item = D0, D1, ..., B0, B1, ..., A0, A1, ..., I0, I1, ..., F0, F1, ..., 또는 T0, T1, ...
data = group-item, bit-expression, integer-expression, 또는 float-expression
bit-expression = 비부호 정수나 bit-definition과 이들간의 AND **"&"**, OR **"|"**, 또는 XOR **"^"**와 NEG **"~"**, 및 **"(...)"**에 의한 우선 연산
integer-expression = 정수와 이들간의 덧셈 **"+"**, 뺄셈 **"-"**, 곱셈 **"*"**, 또는 나눗셈 **"/"**과 음수 **"-"**, 및 **"(...)"**에 의한 우선 연산
float-expression = 실수와 이들간의 덧셈 **"+"**, 뺄셈 **"-"**, 곱셈 **"*"**, 또는 나눗셈 **"/"**과 음수 **"-"**, 및 **"(...)"**에 의한 우선 연산
bit-definition = bit0, bit1, ..., bit31

condition-func와 action-func는 각각 condition 함수와 action 함수의 이름으로서 기본적으로 FuncName (GroupItem, Arg2, Arg3, ...)의 형식을 따르며, 각 함수의 표현 형식은 <표 2>와 같다. condition 함수는 실행 결과를 **if**나 **ifbecome** 문의 판단 조건으로만 사용하고, action 함수는 실행 결과를 GroupItem에 저장한다. **init** 문의 action 함수는 실행 시작 시에 한번만 실행된다. 함수의 첫 인수 GroupItem에는 항상 그룹 항목을 사용하고, 이후의 인수는 직접 수치 또는 이들의 수식으로 지정하거나 다른 그룹 항목을 지정할 수 있다. Add, Sub, Mul, Div 등의 실행 함수와 isLt, isLe, isEq, isNe 등의 조건 함수는 데이터 형을 고정하지 않은 함수이므로 첫 인수인 GroupItem과 이후 인수의 데이터 형이 일치하지 않을 경우에 자동으로 GroupItem의 데이터 형으로 변환해준다.

프로그램의 편의를 위하여 C 언어에서 사용하는 매크로인 **"#define"**, **"#ifdef"**, **"#ifndef"**, **"#if"**, **"#else"**, **"#elif"**, 및 **"#endif"**과 주석 처리인 "/* ... */"나 "// ..."를 C 언어와 동일한 방법으로 PLC 프로그램에서 사용할 수 있다.

TIMER 그룹은 PLC 프로그램에 타이머 기능을 제공하는 것으로서 조건부에 지정된 작동 조건에 의해 작동을 시작하여 지정된 타임아웃 시간 (msec 단위) 후에 타임아웃 되는 기능을 가지고 있다. TIMER 그룹 항목의 값은 작동 중에 있지 않을 경우 -1이고, 작동 중에 있을 경우 작동을 시작한 시점부터 현재까지 경과된 시간을 msec 단위로 표시한다. 지정된 타임아웃 시점에 도달하면 항목의 값이 더 이상 증가하지 않는다. RTSP에는 두 가지 종류의 타이머를 사용할 수 있다. 자동 리셋 (auto-reset) 타이머는 입력 조건부가 만족되면 0에서 시작하여 지정된 타임아웃 시간이 될 때까지 시간이 증가하고, 중간에 입력 조건부가 만족되지 않으면 다시 -1로 초기화된다. 계속유지 (continuous) 타이머는 자동 리셋 타이머와는 달리 한번 조건이 만족되면 이후에는 조건이 만족되지 않더라도 Timer-

Reset 함수가 실행될 때까지 작동을 계속한다. TIMER 그룹에 대한 작업으로는 TimerOn 및 TimerReset 중의 하나를 사용할 수 있다. TimerOn은 타이머를 설정하며 타임아웃 시간은 Data 부분의 32bit 정수로 msec 단위로 표시한다. 따라서 1 msec부터 24일까지의 타임아웃 시간을 설정할 수 있다. 그러나 타임아웃 시간의 최소 시간 단위는 PLC 태스크의 주기보다 짧은 정밀도로 설정하는 것은 의미가 없다. TimerReset는 작동 중인 타이머를 중지시킨다. 현재의 타이머 값은 PLC 프로그램의 조건에 사용될 수 있다.

표 2. PLC 프로그램에서 사용하는 함수 목록
Table 2. Function List of PLC Program.

함수 표시 형식	실행내용 (C 언어 형식)	용도
IsBitOn(Item, onbits)	(ItemData & onbits) == onbits	condition
IsBitAnyOn(Item, onbits)	(ItemData & onbits) != 0	condition
IsBitOff(Item, offbits)	(ItemData & offbits) == 0	condition
IsBitAnyOff(Item, offbits)	(ItemData & offbits) != offbits	condition
IsBitEq(Item,mask,data)	(ItemData & mask) == (data & mask)	condition
IsBitNe(Item,mask,data)	(ItemData & mask) != (data & mask)	condition
IsLt(Item, data)	ItemData < data	condition
IsLe(Item, data)	ItemData <= data	condition
IsEq(Item, data)	ItemData == data	condition
IsNe(Item, data)	ItemData != data	condition
IsGe(Item, data)	ItemData >= data	condition
IsGt(Item, data)	ItemData > data	condition
SetBitOn(Item, onbits)	ItemData | onbits	action
SetBitOff(Item, offbits)	ItemData & ~offbits	action
SetBitEq(Item,mask,data)	(ItemData & ~mask) | (data & mask)	action
Set(Item, data)	data	action
Add(Item, data)	ItemData + data	action
Sub(Item, data)	ItemData - data	action
Mul(Item, data)	ItemData * data	action
Div(Item, data)	ItemData / data	action
BitAnd(Item, mask, data)	(ItemData & ~mask) | (ItemData & data & mask)	action
BitOr(Item, mask, data)	(ItemData & ~mask) | (ItemData | data & mask)	action
BitXor(Item, mask, data)	(ItemData & ~mask) | (ItemData ^ data & mask)	action
BitNeq(Item, mask)	(ItemData & ~mask) | (~ItemData & mask)	action
BitShl(Item, data)	ItemData << data	action
BitShr(Item, data)	ItemData >> data	action
TimerOn(Item, data)	타이머 Item 작동, timeout = data *msec*	action
TimerReset(Item)	타이머 Item 리셋	action
AioAdc(Item)	AD 채널 Item ADC 실행	action
AioDac(Item)	DA 채널 Item DAC 실행	action

타이머를 자동 리셋 모드로 설정하기 위해서는 **ifbecome** 문에 원하는 조건을 지정하고 **then** 다음의 action 함수로 TimerOn 함수를 사용하며 또 다른 **if** 문에 반대의 조건을 지정하고 **then** 다음의 action 함수로 TimerReset를 사용한다.

ifbecome condition **then** TimerOn(AutoTimerItem, timeout)
fi
if condition **then else** TimerReset(AutoTimerItem) **fi**

타이머를 계속유지 모드로 설정하기 위해서는 자동 리셋 모드를 위한 경우에서 TimerReset 부분을 사용하지 않으면 된다.

다음의 프로그램은 업다운 카운터를 위한 RTSP PLC 프로그램 예이다. 이 외에도 PID 제어 프로그램이나 타이머를 활용한 프로그램들을 쉽게 작성할 수 있다.

```
#define PRESET_VALUE   8      // counter preset value
#define DI_ITEM        DI     // DI group item input channel
#define DI_ENABLE      bit0   // ENABLE input signal
#define DI_LOAD        bit1   // LOAD input signal
#define DI_RESET       bit2   // RESET input signal
#define DI_UP          bit3   // UP COUNT input signal
#define DI_DOWN        bit4   // DOWN COUNT input signal
#define DO_ITEM        DI     // DO group item output channel
#define DO_ENABLE      bit0   // ENABLE output signal
#define DO_UP_DONE     bit1   // reach to preset output signal
#define DO_DOWN_DONE   bit2   // reach to zero output signal
#define CNT_ITEM       I0     // COUNT group item
#define CNT_PV_ITEM    I1     // COUNT preset value item

init Set(CNT_PV_ITEM, PRESET_VALUE);// init CNT_PV_ITEM
if IsBitOn(DI_ITEM, DI_ENABLE) then     // if ENABLE is ON
  SetBitOn(DO_ITEM, DO_ENABLE);         // set ENABLE OUT
  if IsBitOn(DI_ITEM, DI_RESET) then    // if RESET is ON
    Set(CNT_ITEM, 0);                   // set COUNT to 0
  else                                  // if RESET is OFF
    if IsBitOn(DI_ITEM, DI_LOAD) then   // if LOAD is ON
      Set(CNT_ITEM, CNT_PV_ITEM);       // set to PRESET VALUE
    else                                // if LOAD is off
      ifbecome IsBitOn(DI_ITEM, DI_UP)  // if UP becomes ON
        then Add(CNT_ITEM, 1); fi       // add 1 to counter
      ifbecome IsBitOn(DI_ITEM, DI_DOWN)// DOWN becomes ON
        then Sub(CNT_ITEM, 1); fi       // subtract 1
    fi
  fi
else
  SetBitOff(DO_ITEM, DO_ENABLE);        // off ENABLE OUT
fi
```

```
if IsGe(CNT_ITEM, CNT_PV_ITEM) then  // if reach to preset
  SetBitOn(DO_ITEM, DO_UP_DONE);     // ON UP_DONE
else                                 // if not reach to
  SetBitOff(DO_ITEM, DO_UP_DONE);    // OFF UP_DONE
if IsLe(CNT_ITEM, 0) then            // if reach to 0
  SetBitOn(DO_ITEM, DO_DOWN_DONE);   // ON DOWN_DONE
else                                 // if not reach to 0
  SetBitOff(DO_ITEM, DO_DOWN_DONE);  // OFF DOWN_DONE
fi
```

2. PLC 프로그램의 표현 능력

RTSP의 PLC 프로그램 문법은 실행 조건에 대하여 임의의 논리를 사용할 수가 있고, 동작 실행에 대하여도 임의의 연산을 표현할 수 있다. 동작 실행 부분에 있어서 정수나 부동 소수점 항목의 경우에는 가감승제를 이용하는 임의의 연산을 실행할 수 있다. 이는 마치 CPU의 가감승제 기본 명령어들을 조합하여 임의의 연산을 구현하는 것과 같다. 일반적인 수식 형태로 표현하고 자동으로 기본 연산들의 조합으로 컴파일할 수 있도록 컴파일러를 보완하는 작업은 추후 과제로 남겨둔다.

실행 조건에 대해서는 비부호 정수, 정수, 및 부동 소수점 연산에 대하여 필요한 비교연산을 제공하고 있다. 단순 비교가 아니고 일정한 연산을 한 결과를 비교해야 할 경우에는 사전에 실행 함수들을 이용하여 필요한 연산을 한 후에 그 결과를 실행 조건부에서 비교연산을 실행한다. 비교 연산들 간에 AND나 OR 연산이 필요한 경우에는 BIT 그룹의 항목을 임시 변수로 활용하여 표현할 수 있다. 다음은 AND 연산이 필요한 경우의 예로서 "if cond1 AND cond2 AND cond3 then statement-list fi" 와 같은 연산이 필요한 경우는 IsBitOn 함수와 B0 항목의 bit0, bit1, bit2를 임시변수로 활용하여 다음과 같이 표현할 수 있다.

```
SetBitOff(B0, bit2 | bit1 | bit0);
if cond1 then SetBitOn(B0, bit0); fi
if cond2 then SetBitOn(B0, bit1); fi
if cond3 then SetBitOn(B0, bit2); fi
if IsBitOn(B0, bit2 | bit1 | bit0) then statement-list fi
```

OR 연산이 필요한 경우의 예로서 "**if** cond1 OR cond2 **then** statement- list **fi**" 와 같은 연산이 필요한 경우에는 IsBitAnyOn 함수와 B0 항목의 bit4 및 bit5를 사용하여 다음과 같이 표현할 수 있다.

```
SetBitOff(B0, bit5 | bit4);
if cond1 then SetBitOn(B0, bit4); fi
if cond2 then SetBitOn(B0, bit5); fi
if IsBitAnyOn(B0, bit5 | bit4) then statement-list fi
```

다른 방법으로는 B0 항목의 bit4 만을 임시변수로 활용하여 다음과 같이 표현할 수도 있다. 이것은 하드웨어에서 wired-or 논리와 유사한 형식이다.

```
SetBitOff(B0, bit4);
if cond1 then SetBitOn(B0, bit4); fi
if cond2 then SetBitOn(B0, bit4); fi
if IsBitOn(B0, bit4) then statement-list fi
```

NOT 연산은 if 문의 else 부분을 이용하거나 사전에 BitNeg 실행함수를 이용하여 표현할 수 있다. 한 예로서 "**if** NOT cond **then** statement-list **fi**" 와 같은 연산이 필요한 경우는 다음과 같이 표현할 수 있다.

```
if cond then else statement-list fi
```

따라서 AND, OR, 및 NOT 연산이 모두 표현 가능하므로 임의의 조건을 PLC 프로그램으로 표현할 수 있게 된다. 다음은 이들의 조합으로 이루어진 조건의 한 예로서 "**if** (cond1 AND cond2) OR (cond3 AND (NOT cond4)) **then** statement-list **fi**" 와 같은 연산을 위한 RTSP PLC 프로그램을 보여준다.

```
SetBitOff(B0, bit4 | bit3 | bit2 | bit1 | bit0);
if cond1 then SetBitOn(B0, bit0); fi
if cond2 then SetBitOn(B0, bit1); fi
if IsBitOn(B0, bit1 | bit0) then SetBit(B0, bit4); fi
if cond3 then SetBitOn(B0, bit2); fi
if cond4 then else SetBitOn(B0, bit3); fi
if IsBitOn(B0, bit3 | bit2) then SetBit(B0, bit4); fi
if IsBitOn(B0, bit4) then statement-list fi
```

V. RTSP의 성능 평가

RTSP의 성능을 평가하기 위해서 메인 보드는 어드벤텍 사의 내장형 시스템용 PCM-5820 싱글보드 컴퓨터를 사용 했으며 입출력 보드로는 ADLINK 사의 ACL-8112DG 보드를 사용하였다. PCM-5820은 상당히 낮은 성능의 보드로서 펜티엄 급의 NS GX1 233MHz

를 사용하고 있으며 ACL-8112DG는 16비트 DI 채널, 16비트 DO 채널 및 2개의 12비트 DA 채널과 멀티플렉스 모드로 작동하는 16개의 12비트 AD 채널들이 있다.

PLC 프로그램의 실행 속도인 스캔 시간은 프로그램의 스텝 수, DI 및 DO 채널 개수, AD 및 DA 실행 회수 등에 따라 증가한다. 입출력 실행 시간은 사용하는 입출력 보드의 성능에 따라 달라진다. RTSP에서 프로그램 스텝 수는 프로그램에 사용된 조건 함수 및 액션 함수의 개수와 일치한다. 실제 실행 시간을 측정해본 결과 실제 실행되는 함수 당 0.92μsec가 소요되며 실행되지 않는 함수 당 0.27μsec가 소요되었다. ADLINK 사의 입출력 보드로 측정해본 결과 DI 채널당 입력 시간은 5.2μsec가 소요되었고, DO 채널당 출력 시간은 4.6μsec가 소요되었다. D/A 변환 채널은 한번 실행에 5.0μsec가 소요되었고 A/D 변환 채널은 한번 실행에 12.5μsec가 소요되었다.

일반적인 소규모 PLC와 비슷하게 16비트 DI 채널을 8개 (총 128비트 DI), 16비트 DO 채널을 8개 (총 128비트 DO), AD 채널 8개, DA 채널 8개를 사용한다고 가정하면 입출력에 소요되는 시간은 다음과 같다. 우선 아날로그 입출력을 실행하지 않으면 PLC 태스크가 한번 프로그램을 스캔하면서 DI 입력과 DO 출력을 위해 소요되는 시간은 $T_{io} = 5.2 \times 8 + 4.6 \times 8 = 78.4$ μsec이다. 따라서 1msec 주기로 PLC 태스크가 실행되기 위해서는 실행 가능한 프로그램 스텝 수는 $N_{step} < (1000 - 78.4) / 0.92 = 1001$ 이다. 즉, 프로그램 스텝 수가 1000개 이내이면 PLC 태스크를 1msec 주기로 실행 할 수 있다는 것을 의미한다. 여기서 모든 프로그램 항목이 실행되는 최악의 경우를 가정하였고 이 정도 규모의 실제 시스템에서도 프로그램 스텝수가 1000개정도면 어느 정도 충분할 것이므로 1msec의 정밀도로 실행이 가능해진다. 여기서 AD 및 DA 채널들의 입출력도 함께 사용하는 경우에는 매 스캔마다 실행되는 최악의 상황을 가정하면 $T_{io} = 5.2 \times 8 + 4.6 \times 8 + 5.0 \times 8 + 12.5 \times 8 = 218.4$μsec이고 이 경우 1msec 실행 주기를 위해서는 $N_{step} < (1000 - 218.4) / 0.92 = 850$개이다.

규모가 큰 PLC처럼 DI 채널 512비트 (16비트 채널 32개), DO 채널 512비트 (16비트 채널 32개), AD 채널 32개, DA 채널 32개를 가정하면 $T_{io} = 5.2 \times 32 + 4.6 \times 32 + 5.0 \times 32 + 12.5 \times 32 = 873.6$μsec이고 실행주기를 1msec로는 어렵지만 2msec로 늘리면 가능한 스텝 수는 $N_{step} < (2000 - 873.6) / 0.92 = 1224$개이고 실행주기를 3msec로 늘리면 스텝 수는 2311개로 늘어난다. 약간의 여유시간을 가정하더라도 5msec 정도의 주기라면 충분히 처리가 가능할 것이다. 그러나 실제 시스템의 경우에는 하드웨어의 제약으로 인해 이 정도 규모의 입출력 보드를 장착하기는 상당히 무리가 있을 것이다.

VI. 결 론

본 논문에서는 산업용 PC를 기반으로 하여 시장에서 구입가능한 입출력 보드들을 장착하여 PLC를 구성하는 방법을 제시하였다. 이를 위해서 산업용으로도 그 용도를 넓혀가고 있는 리눅스를 활용하여 실시간 스케줄링을 지원하는 RTLinux를 설치하고 여기에 실시간 태스크들로 PLC 프로그램을 실행하도록 구현하였다. 또한 PLC 프로그램을 편리하게 작성할 수 있도록 자체 프로그램 문법을 제안하고 이를 위한 컴파일러도 구현하였다. 구현된 시스템으로 측정해 본 결과 중대규모 시스템 정도의 경우에도 5msec 정도의 스캔 시간을 확보할 수 있음을 확인하였다. 그러나 실제 시스템에서는 중대규모 정도의 입출력 보드를 장착하기에는 일반 PC 호환 하드웨어 사양으로는 무리가 있으므로 본 논문에서 구현한 RTSP는 소규모 PLC의 대용 시스템으로 활용하기에 적당할 것이다.

본 논문에서 구현한 소프트웨어 PLC는 실제 자동화 환경에서 활용하는데 있어서 여러 가지 장점이 있다. 우선 리눅스를 기반으로 하고 있으므로 실시간 실행이 필요한 PLC 기능 실행과 동시에 TCP/IP를 기반의 통신 기능과 그래픽 사용자 환경을 그대로 활용할 수 있으며 웹 서버를 포함한 다양한 소프트웨어들을 그대로 활용할 수 있다. 둘째로 RTLinux는 실시간 태스크들이 커널 모드에서 실행되므로 응용 분야에 따라 개발된 프로그램의 오류에 의해 전체 시스템이 마비될 위험이 많지만, RTSP 환경에서는 일반 사용자들은 커널 모드 태스크 프로그램은 직접 작성하지 않고 래더 다이어그램과 유사한 형식으로 PLC 프로그램을 작성하고 이것을 RTSP에서 제공하는 태스크가 해석하여 실행하므로 이런 위험이 없다. 또한 커널 모드의 실시간 태스크를 위한 프로그램을 작성하려면 이를 위한 별도의 지식이 필요하지만, RTSP에서는 단순한 문법에 따라 프로그

램을 작성법을 쉽게 익힐 수 있다. 셋째로 범용 PC 규격을 활용하므로 필요한 성능 및 입출력 보드들을 저렴한 가격으로 구입하여 활용할 수 있다.

추후 과제로는 PLC 프로그램의 문법을 확장하고 이를 위한 컴파일러를 개발하여 보다 편리한 프로그램 작성 환경을 개발할 필요가 있다. 더 나아가서는 그래픽 환경에서 래더 다이어그램이나 여러 가지 국제 표준 방식으로 프로그램을 작성할 수 있도록 하고, 작성된 프로그램을 사전에 검증해볼 수 있도록 하는 프로그램 개발 툴들을 제공해야 한다.

참 고 문 헌

[1] IEC 61131-1, Programmable Controllers - Part 1: General Information, International Electrotechnical Commission, 1993.

[2] IEC 61131-3, Programmable Controllers - Part 3: Programming Languages, International Electrotechnical Commission, 1993.

[3] 김원희, 안항목, PLC 이론과 실제, 도서출판 기술, 1995

[4] Schleicher Gmbh & Co., ProNumeric/ProSycon Operating Manual - Controller on IPC Hardware for CNC/PLC and PLC, http://www.schleicher-de.com, 2003.

[5] M. de. Sousa, "Linux-Based PLC for Industrial Control,", Embedded Linux Journal (http://embedded.linuxjournal.com), Issue 3, May 2001.

[6] LinuxDevices.com, "The Real-time Linux Software Quick Reference Guide," http://www.linuxdevices.com/articles/AT8073314981.html, April 2002.

[7] M. Barabanov, A Linux-based Real-Time Operating System, New Mexico Institute of Mining and Technology, 1997.

[8] V. Yodaiken, "The RTLinux Manifesto", Department of Computer Science, New Mexico Institute of Technology, http://www.rtlinux.org, Nov. 1999.

[9] P. Bhowal, R. Mall, and A. Basu, "Estimating micro-PLC execution time for time critical system design," Journal of Systems Architecture 45(14) pp. 1245-1248, Elsevier Science B.V., 1999.

[10] M. Ohman, S. Johansson, and K. E. Arzen, "Implementation aspects of the PLC standard IEC 1131-3,", Control Engineering Practice 6(4) pp. 547-555, Elsevier Science B.V., 1998.

[11] T. F. Al-Khudairy, B. A. Al-Hashemy, and M. A. J. Al-Baker, "Design of a VMEbus-based programmable logic controller (PLC)," Microprocessors and Microsystems 21(5) pp. 329-336, Elsevier Science B.V., 1998.

저 자 소 개

金 龍 碩(正會員)

1984년 : 서울대학교 해양학과(전자공학 부전공) 학사. 1986년 : KAIST 전기및전자공학과 석사. 1989년 : KAIST 전기및전자공학과 박사. 1989년~1994년 9월 : 한국생산기술연구원에서 "ISDN 기반의 G4 팩스" 개발. 1994년 10월~1995년 2월 : 전자부품연구원에서 "대화형 CATV 시스템을 위한 가입자 단말기" 개발. 1995년 3월~현재 : 강원대학교 전기전자정보통신공학부 재직. <주관심분야 : 운영체제, 실시간 시스템, 내장형 시스템, 병렬처리 등>