韓國컴퓨터情報學會 論文誌
第14卷 第3號, 2009. 3.

2009-14-3-1-2

시스템 모델을 통한 PLC 기반 시스템의 RTOS 기반 시스템으로의 변환

김 제 웅*, 임 성 수**

System Model-driven Conversion from PLC-based Systems to RTOS-based Systems

Je Wung Kim*, Sung-Soo Lim**

요 약

본 논문에서는 현재 많은 산업 현장에서 자동 제어를 위한 시스템으로 사용하는 PLC를 대체하기 위한 시스템으로 RTOS 기반 시스템을 제안하였다. RTOS 기반 시스템은 PLC의 한계를 극복하고 시스템의 안정성과 신뢰성을 보장하기 위한 방법으로 PC와 RTOS를 사용하는 시스템이다. 그리고 PLC 기반 시스템을 RTOS 기반 시스템으로 변환하기 위한 방법으로 시스템 모델을 통한 변환 방법을 제안하였고 변환 절차와 변환 방법을 설명하였다. 시스템 모델은 PLC 기반 시스템을 RTOS 기반 시스템으로 변환하기에 앞서 시스템을 상위 레벨에서 하위레벨로 태스크 단위로 분석한 것으로 시스템을 모듈 별로 정의하고 정의된 모듈의 동작을 태스크로 세분화하여 정의한 것이다. 모듈 별로 시스템을 제어하는 것은 PLC를 통한 제어에 비해 성능뿐만 아니라 기능적으로도 향상을 가져오고 추후 시스템의 수정이나 변화 시에도 더 유연하게 대처할 수 있다.

Abstract

In this paper, We propose the alternative solution, RTOS-based system to replace the PLC that has used the automation system for industrial processes. RTOS-based system is constructed the PC and RTOS as hardware and software. It overcomes the limit of PLC and guarantees the stability and reliability. Also, PC has better performance and cheaper than PLC when operating and constructing the system. For many manufactures, these benefits alone are all the reason they need to switch from PLC-based system to RTOS-based system.To use the RTOS-based System, the PLC program needs the conversion to the RTOS task. And how to transform is the most important issue. So, we propose conversion method through the system model. The system model defines the operation of each module as the task after the system divided into module. Because the system divided into modules can control, the performance and the functionality of system improve, and the system can deal with a problem easily when repairing and changing.

▸ Keyword : PLC(Programmable Logic Controller), RTOS, PC-based Control, System Model

• 제1저자 : 김제웅
• 투고일 : 2008. 11. 26, 심사일 : 2008. 12. 3, 게재확정일 : 2009. 3. 2.
* 국민대학교 컴퓨터 공학부

Ⅰ. 서 론

PLC(Programmable Logic Controller)는 1960년대에 릴레이 소자를 대체하기 위해 개발된 시스템으로 지금까지 자동 제어를 위한 대표적인 시스템으로 산업 현장에서 사용되고 있다. PLC가 자동 제어를 위한 대표적인 시스템으로 산업 현장에 자리 잡은 이유는 산업 현장이라는 특수한 환경에서도 안정성과 신뢰성을 보장할 수 있도록 다양한 환경에 강하게 설계되었기 때문이다. 하지만 PLC는 지금까지의 계속된 노력에도 불구하고 아직까지 시스템의 표준화가 이루어지지 않아 제조업체마다 다른 규격의 I/O 장비를 사용하고 있고 PLC 프로그램의 프로그래밍 방법이 제조업체마다 달라 업체 및 종류에 따라 별도의 교육 및 지식을 필요로 한다. 그리고 타 업체의 I/O 장비와는 호환이 되지 않아 시스템 구축을 위한 장비 선정에 있어서도 한계를 가지고 있다. 결국 이러한 문제들은 비용적인 측면과 시간적인 측면에서 시스템의 구축과 유지 보수에 많은 손해를 초래하게 된다.[10]

본 논문에서는 PLC가 가지는 한계를 극복하고 기존의 PLC 기반 시스템의 안정성과 신뢰성을 보장할 수 있는 대안으로 RTOS 기반 시스템을 제안한다. RTOS 기반 시스템은 표준화된 시스템인 PC를 사용함으로써 PLC가 가지는 시스템의 한계를 극복하고 RTOS를 통해 안정성과 신뢰성을 보장하는 시스템이다.[2]

사용 중인 PLC를 대체하고 RTOS 기반 시스템을 산업 현장에서 사용하기 위해서는 기존의 PLC 프로그램을 RTOS 태스크로 변환해야 하고, 변환을 위한 변환 절차와 방법이 필요하게 된다. 그래서 본 논문에서는 PLC 기반 시스템을 RTOS 기반 시스템으로 변환하기 위한 프레임워크와 변화 절차를 제안하였고 제안된 절차를 통해 변환된 결과가 어떻게 PLC 기반 시스템의 성능을 보장 할 수 있는지를 증명하였다.

본 논문에서 제안하는 변환 방법은 시스템 모델을 통한 변환 방법이다. 시스템 모델은 시스템을 상위레벨에서 하위레벨로 태스크 단위로 분석한 것으로 시스템을 모듈 별로 정의하고 정의된 모듈의 동작을 태스크로 세분화하여 정의 한 것이다. PLC 프로그램을 시스템 모델을 통해 RTOS 태스크로 변환하는 것은 PLC 프로그램을 RTOS 태스크로 바로 변환하는 것에 비해 복잡한 일이지만 시스템의 동작들을 시스템 모델을 통해 모듈 별로 정의할 수 있으므로 동작의 효율성을 높일 수 있다. 그리고 시스템이 동작 중 오류가 발생할 경우에도 시스템의 동작을 모듈 별로 정의할 경우 오류가 발생하지 않은 시스템의 다른 모듈은 동작을 멈추지 않고 동작할 수 있으므로 효율적인 시스템 운영이 가능하다. 시스템 모델을 통한 변환의 또 다른 장점은 시스템에 새로운 기능을 추가하거나 기능의 수정이 필요할 시에도 추가되는 모듈에 대한 태스크만 추가하면 되므로 다른 태스크의 수정으로 인한 문제가 발생하지 않고 수정이 필요한 경우 해당하는 모듈의 태스크만 수정하면 되므로 시스템의 유지 보수나 업그레이드 시 발생하는 시간적인 비용을 줄일 수 있다는 점이다.

본 논문에서는 시스템 모델을 정의하기 위한 절차를 설명하고 예로써 e-printing 시스템을 사용한다. e-printing 시스템은 PCB의 패턴을 종이에 인쇄하듯 프린트하는 기법을 통해 PCB를 생산하는 장비이다. 그리고 저수지 수위 제어 시스템과 자동문 개폐 제어 시스템을 예로 들어 논문에서 정의한 변환 절차를 통해 PLC 프로그램을 RTOS 태스크로 변환하는 과정을 설명한다. 본 논문에서 사용하는 RTOS는 상용 소프트웨어인 WINDRIVER사에서 제공하는 VxWorks-6.3을 사용한다.

본 논문의 구성은 2장에서 PLC의 동작 구조와 PLC 프로그램 구조가 가지는 문제점을 설명하고 3장에서는 변환 방법으로 제안한 시스템 모델을 통한 변환 프레임워크를 설명한다. 4장에서는 시스템 모델을 통한 변환 프레임워크의 세부적인 변환 절차를 설명하고 5장에서는 예를 통해 PLC 프로그램을 RTOS 태스크로 변환하는 방법을 설명한다. 6장에서는 변환 결과를 통해 RTOS 태스크로의 변환 결과가 PLC 프로그램의 성능을 만족하는 것을 보여주고 7장에서는 본 논문의 내용을 정리한 다음 결론을 맺는다.

Ⅱ. PLC

PLC는 "논리 연산, 순서 조작, 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 통해서 행하는 디지털 조작형의 공업용 전자장치"로 정의된다.

2.1 PLC의 구조

PLC의 구성은 입력 부, 출력 부, 연산 부, 메모리 부 및 전원장치와 주변기기 등으로 구성되어 있다.

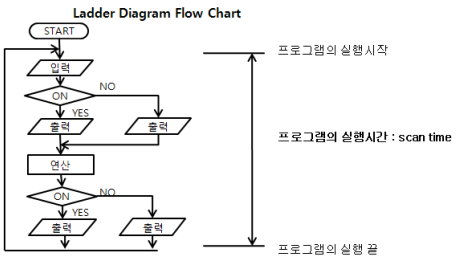


그림 1 PLC 프로그램의 실행 흐름과 스캔 타임
Fig. 1. Flow Chart and Scan Time of PLC Program

(그림 1)은 PLC 프로그램의 실행 흐름도와 스캔 타임을 설명한 그림으로 PLC 프로그램은 일정한 주기를 가지고 입력 부를 통해 데이터를 얻는 입력 과정과 입력 값에 대한 처리 과정, 처리 결과에 대한 결과 값을 출력부를 통해 출력하는 것을 출력과정을 반복 실행한다. 즉, PLC의 기본 동작 구조는 입력, 처리, 출력의 동작들이 주기적으로 반복 실행되는 구조이다.[10]

2.2 PLC의 스캔타임

PLC 프로그램은 전원이 켜지는 동시에 입력된 제어 프로그램을 처음부터 끝까지 일정시간을 주기로 반복 실행되는 순환 제어 프로그램으로 이러한 주기를 PLC에서는 스캔타임(Scan Time)을 통해 정의한다. (그림 1)에서 보듯 PLC 프로그램의 스캔타임은 프로그램의 시작에서부터 처리, 종료까지의 시간을 나타낸다.

스캔타임은 PLC 플랫폼의 성능에 큰 영향을 받으며 명령의 처리속도와 프로그램의 길이에 따라 정해지게 된다.

2.3 Ladder 다이어그램

대부분의 PLC 프로그램은 Ladder-다이어그램 구조를 통해 프로그래밍 되고 동작되고 있다.

그림 2 PLC 프로그램의 Ladder 다이어그램
Fig. 2. Ladder diagram of PLC program

(그림 2)는 입력과 출력간의 논리 구조를 Ladder 다이어그램 방식으로 프로그래밍 한 PLC 프로그램의 예이다. (그림 2)에서처럼 PLC 프로그램은 왼쪽의 입력을 나타내는 선과 오른쪽의 출력을 나타내는 선 사이에 필요한 작업들이 나열된 구조를 가지고 있으며 이를 Ladder-다이어그램 구조라 말한다.

Ladder-다이어그램에서 입력과 출력의 관계는 연결된 선(Line)을 통해 정의되며 출력 신호는 여러 개의 입력 신호의 조합으로 이루어진다. 조합은 입력과 출력간의 논리 관계를 말하며, 논리(logic)는 입력과 출력이 연결된 형태를 통해 정의된다. 만약 두 개의 입력 노드가 직렬로 연결되어 있다면 이들은 AND 논리 연산으로 연결되어 있고, 병렬로 연결되어 있다면 이들은 OR 논리연산으로 연결되어 있다.

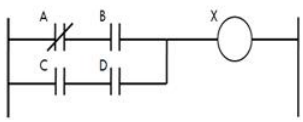


그림 3 PLC 프로그램의 논리 구조 예
Fig. 3. Logic example of PLC program

(그림 3)은 PLC 프로그램의 입력과 출력간의 논리 구조를 나타내며 (그림 3)의 PLC 프로그램은 다음과 같은 논리식으로 표현 될 수 있다.

(NOT A AND B) OR (C AND D) = X

2.4 PLC 프로그램의 문제점

PLC 프로그램이 가지고 있는 문제는 프로그램의 실행이 정적으로 실행된다는 것이다. PLC 프로그램의 정적인 실행은 모든 입력에 대한 처리가 입력 별로 독립적으로 처리되어지지 않고 시스템의 실행이 전체 입력에 대해 입력 처리 출력 순으로 단계별로 고정되어 진행되는 것을 말한다. 즉, PLC 프로그램은 시스템의 상태와 입출력 값과는 상관없이 언제나 일정하게 실행되며 이러한 점 때문에 PLC 프로그램에서는 시스템의 상태와 입출력 값과 같은 각각의 입력에 대한 출력의 시간적 요구사항을 만족 시키는 것이 불가능하다. 또한, 시스템 동작 중 시스템의 동작과는 상관없는 단순한 오류에도 시스템 전체에 큰 영향을 미쳐 심하게는 시스템의 동작을 멈추게 할 수도 있다. PLC의 또 다른 문제점은 스캔타임을 통해 나타난다. 스캔타임은 명령어의 처리속도를 결정 짓는 하드웨어인 PLC 플랫폼의 성능에 영향을 받으며 프로그램의 크기를 결정짓는 시스템의 요구사항에 영향을 받게 된다. 결

국, 시스템이 커질수록 PLC에서 수행되어야 할 작업이 많아지게 되고 스캔타임이 가질 수 있는 값에도 한계를 가지게 된다. 즉, 스캔타임은 시스템의 구성이나 프로그램의 변경에 따라 변하게 되고 이러한 점들은 시스템의 안정성과 신뢰성에 문제를 발생시킬 수 있게 된다.

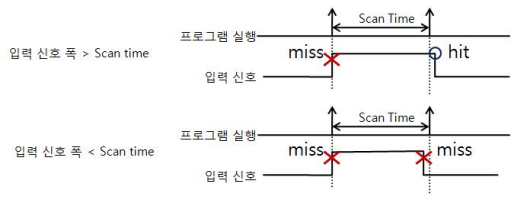


그림 4 스캔타임에 따른 데이터 손실
Fig. 4. Data loss through scan time

(그림 4)에서처럼 PLC 프로그램의 연산량이 많아져 입력 신호의 폭이 스캔 타임보다 작을 경우 최악의 경우에는 입력 값이 스캔 타임 이상 지속되지 않아 입력 값을 놓칠 수도 있는 경우가 발생하게 된다.[10]

본 논문에서는 이와 같은 PLC 프로그램이 가지는 문제를 해결하기 위한 대안으로 RTOS 기반 시스템을 제안하고 기존의 PLC 기반 시스템에서 RTOS 기반 시스템을 사용하기 위한 RTOS 기반 시스템으로의 변환과정을 설명한다.

III.시스템 모델을 통한 변환 프레임워크

시스템 모델을 통한 변환은 PLC 프로그램을 RTOS 태스크로 변환하기에 앞서 시스템을 분석하여 시스템 모델을 구하고 구한 시스템 모델을 바탕으로 PLC 프로그램을 재구성하여 RTOS 태스크로 변환하는 방법을 말한다.

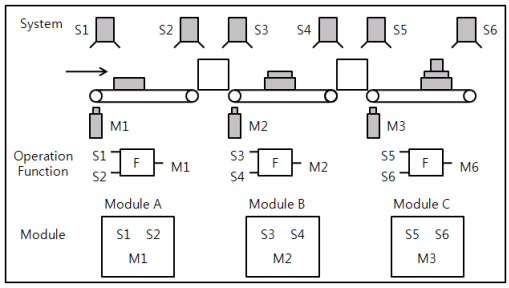


그림 5 시스템 모듈 예
Fig. 5. System module example

시스템 모델은 시스템을 분석하고 모듈 별로 정의한 후 정의된 모듈 별로 모듈의 동작을 태스크로 정의한 것을 말한다. 시스템을 모듈 별로 구성하는 것은 PLC 프로그램을 모듈 별로 효율적으로 재구성하여 시스템의 동작 효율을 높이기 위해서다. 모듈은 시스템에서 사용되는 하드웨어들을 동작의 독립성을 기준으로 시스템의 동작 순서에 따라 시스템을 분류한 것을 말한다.

(그림 5)는 시스템을 모듈로 분류한 예로 M은 모터를, S는 센서를 나타낸다. 시스템은 센서에서 얻은 데이터를 모터에 적용하여, 컨베이어 벨트를 동작시켜, 제품을 이동시킨다. 출력을 나타내는 모터 값은 두 개의 입력을 나타내는 센서 값을 통해 결정된다. 즉, (그림 5)의 시스템에서 모듈은 시스템에서 사용된 입력과 출력의 연관성을 통해 출력인 모터와 입력인 센서를 모듈 별로 분류한다.

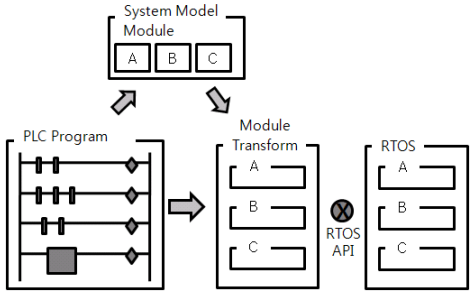


그림 6 시스템 모델을 통한 변환 프레임워크
Fig. 6. Conversion framework through system model

(그림 6)은 PLC 프로그램을 시스템 모델을 통해 RTOS 태스크로 변환하는 과정을 설명하는 프레임워크이다. (그림 6)의 프레임워크에서 시스템은 A, B, C 세 개의 시스템 모델로 모델링 되고 모듈 별로 필요한 태스크들이 정의 되어 있다. 이후 PLC 프로그램은 시스템 모델을 통해 A, B, C 세 개의 모듈로 재구성되고 태스크 별로 재구성된 PLC 프로그램은 RTOS API를 통해 RTOS 태스크로 변환한다. 자세한 변환 절차에 대한 내용은 다음 장에서 다룬다.

IV. PLC 프로그램을 RTOS 태스크로의 변환 절차

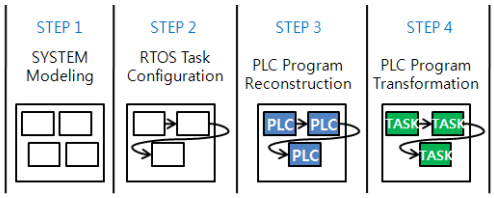


그림 7 PLC 프로그램의 변환 절차
Fig. 7. Conversion process of PLC

(그림 7)은 PLC 프로그램의 변환 절차를 나타낸다. 첫 번째 단계는 시스템 모델링 단계로 시스템을 모듈화 하고 모듈의 동작에 필요한 태스크들을 정의한다. 두 번째 단계는 정의된 태스크들 간의 순차적인 실행관계와 주기적인 동작 구조를 설정하기 위한 태스크들의 구조와 우선순위, 주기 등을 설정하는 단계이다. 세 번째 단계는 PLC 프로그램을 정의된 태스크를 통해 재구성하는 단계로 재구성된 PLC 프로그램들은 네 번째 단계에서 RTOS API를 통해 RTOS 태스크로 변환된다.

4.1 시스템 모델링

시스템 모델링은 시스템을 모듈화 하고 응용 프로그램에 필요한 태스크를 구성하는 단계이며, PLC 프로그램과 시스템 관련 문서를 통해 정의 한다. 시스템 모델링은 프로그램에서 사용된 입출력 디바이스와 시스템의 역할과 요구 사항 등을 분석하여 정의한다.

4.1.1 PLC 프로그램을 통한 시스템 모델링

PLC 프로그램을 통한 시스템 모델링은 PLC 프로그램을 분석하여 시스템을 모듈 별로 정의 하는 것으로 시스템에 사용된 I/O 디바이스와 시스템의 역할과 요구 사항 등을 분석하여 이를 시스템 모델링에 필요한 정보로 사용하는 것이다.

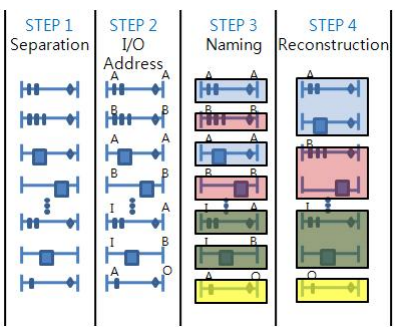


그림 8 PLC 프로그램의 모델링
Fig. 8. Modeling of PLC program

(그림 8)은 PLC 프로그램의 모델링 절차를 설명한다. 첫 번째 절차는 Separation단계로 PLC 프로그램인 Ladder-다이어그램을 라인 별로 나눠 하나의 작업 단위로 분류하는 단계이다. 두 번째 단계는 I/O Address단계로 작업들의 입•출력부를 사용된 포트 정보를 통해 모듈 별로 정의하는 단계이다. 세 번째 단계는 Naming 단계로 정의된 입•출력부의 모듈 이름을 통해 작업들의 모듈을 정의하는 단계이다. 그리고 마지막 단계는 각각의 작업들을 해당하는 모듈 별로 재구성하는 단계이다.

작업들의 모듈을 정의하는 방법은 입력과 출력에 할당된 모듈을 기준으로 입력과 출력에 할당된 모듈이 같은 모듈일 경우에는 사용된 모듈로 분류하고 다른 모듈일 경우에는 해당하는 출력이 시스템의 동작과는 상관없는 LCD와 같은 외부 출력 장치 일 경우에는 출력을 기준으로 작업을 분류하고 입력이 버튼과 같은 시스템의 특정한 동작을 강제로 시키는 외부 입력 장치일 경우에는 입력을 기준으로 작업을 분류한다.

4.1.1.1 PLC 프로그램을 통한 시스템 모델링 예

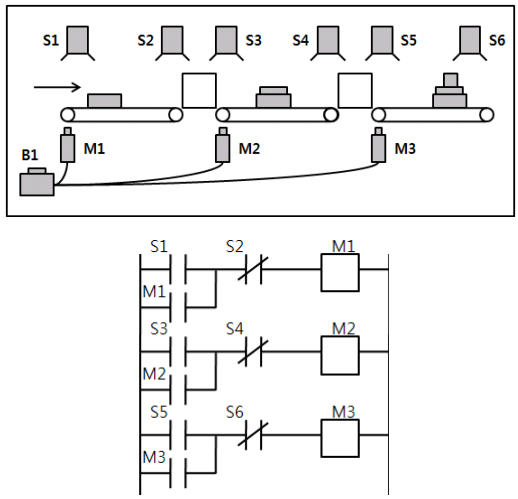


그림 9 입력장치를 추가한 시스템과 PLC 프로그램
Fig. 9. System added input and PLC program

(그림 9)는 (그림 5)의 시스템에 입력장치를 추가한 시스템으로 B1은 시스템의 입력장치로 On/Off 스위치를 나타낸다. PLC 프로그램에는 외부 입력 버튼의 처리, 시스템의 각 구간의 처리를 나타내고 있다.

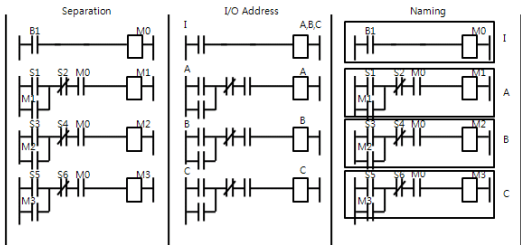


그림 10 PLC 프로그램을 통한 모델링 예
Fig. 10. Modeling example of PLC program

(그림 10)은 (그림 9)의 PLC 프로그램을 정의된 절차를 통해 프로그램을 모듈 별로 분류한 모습이다. 처음 Separation 단계에서는 PLC 프로그램을 각각의 작업 별로 분류 하고 I/O Address 단계에서는 각각의 작업들의 입력과 출력의 모듈을 정의한다. M0의 경우 외부 입력 장치에 의해 제어되는 값으로 이 값은 모듈 A, B, C의 동작에 영향을 미친다. 나머지 작업들은 각각의 작업들 간에 입력과 출력을 공유하지 않고 독립적으로 동작하므로 각각의 작업들을 A, B, C의 다른 모듈들로 할당한다. 그리고 Naming 과정에서 각각의 작업들의 모듈 명을 작업들의 입력과 출력을 통해 정의한다. 외부 입력을 사용한 작업은 입력 값을 기준으로 모듈 명을 I로 할당한다.

4.1.2 시스템의 구조 정보를 통한 시스템 모델링

시스템의 구조 정보를 통한 시스템 모델링은 시스템의 구조와 I/O 디바이스 정보를 정의 한 문서를 바탕으로 상위 레벨 시스템 분석법(High Level System Analysis)인 Outside-In 방법론을 사용한다.

4.1.2.1 Outside-In 방법론

Outside-In 방법론은 시스템을 외부에서 내부로 응용 프로그램을 태스크 단위로 분화해 가는 방법으로 다음과 같은 절차로 진행된다.[5]

Outside-In 방법론의 절차

1. 시스템을 처리과정에 따라 섹션 별로 정의
2. 섹션 별로 I/O 디바이스 정의
3. 입력 부와 출력부의 인터페이스 정의
4. I/O 디바이스에 따른 태스크 세분화

섹션은 시스템을 처리과정에 따라 분류한 단위로 섹션은 섹션의 입력과 출력 사이의 독립성을 기준으로 분류한다. 독립성은 하나의 섹션에서 사용된 입력과 출력이 다른 섹션의 입력으로 사용되지 않고 출력의 값에 영향을 안 미치는 것을 의미한다.

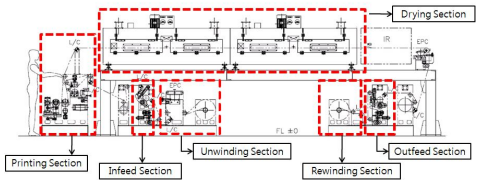


그림 11 e-Printing 시스템의 섹션 별 정의
Fig. 11. Section of e-Printing system

(그림 11)은 e-Printing 시스템을 예를 들어 시스템을 섹션 별로 정의 한 모습이다. e-Printing 시스템은 unwinding, infeed, printing, drying, outfeed, rewinding으로 섹션을 정의할 수 있다. 섹션 별로 정의된 시스템은 컨텍스트 다이어그램을 통해 모듈에서 각각의 모듈의 동작을 정의하는 태스크까지 분화해가며 정의한다.

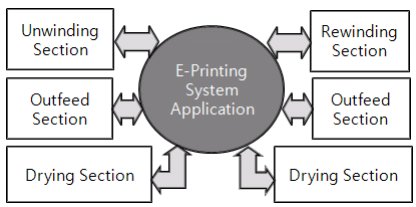


그림 12 High Level Context Diagram
Fig. 12. High Level Context Diagram

(그림 12)는 상위 레벨 컨텍스트 다이어그램(High Level Context Diagram)으로 시스템의 입력과 출력을 파악하고 상위 레벨에서 하위 레벨로 시스템을 정의하기 위한 다이어그램이다.

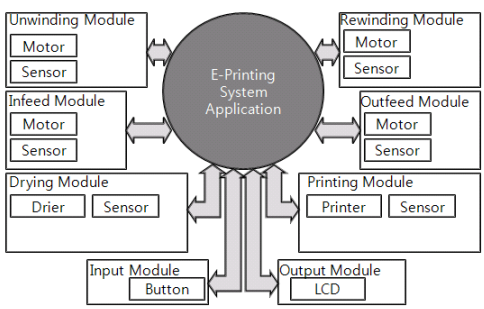


그림 13 섹션 별로 I/O 디바이스 정의
Fig. 13. I/O Device definition of Sections

(그림 13)은 (그림 12)의 섹션들을 섹션 별로 I/O 디바

이스를 정의한 모습이다. I/O 디바이스를 정의하는 것은 시스템을 동작 순서에 따른 논리적인 분류인 섹션에서 디바이스를 기준으로 한 물리적인 분류인 모듈로 분류하기 위해서다. 이 단계에서는 I/O 디바이스 정보를 통해 입력 모듈과 출력 모듈이 있을 경우 이를 추가적으로 정의한다.

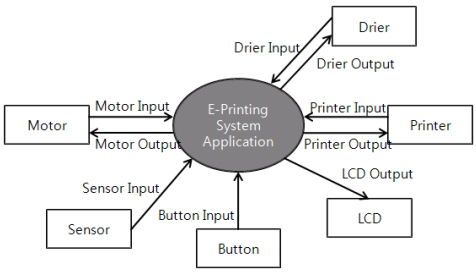


그림 14 I/O 디바이스를의 인터페이스 정의
Fig. 14. Interface definition of I/O devices

(그림 14)는 시스템을 I/O 디바이스를 기준으로 시스템의 인터페이스를 간략히 표현한 모습이다. 시스템을 I/O 디바이스를 기준으로 정의한 것은 각각의 모듈 별로 사용된 입•출력 장치들을 위한 태스크를 정의하기 위함이다.

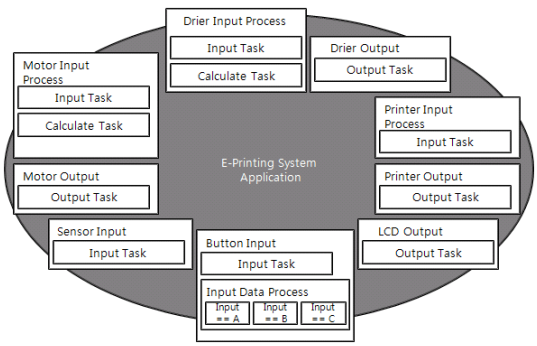


그림 15 I/O 디바이스에 따른 태스크 세분화
Fig. 15. Task definition of I/O devices

(그림 15)는 I/O 디바이스에 따른 태스크를 세분화한 모습이다. I/O 디바이스에 따른 태스크 세분화는 모듈 별로 사용된 I/O 디바이스를 통해 각각의 모듈들의 동작을 정의하는 태스크를 정의하는 단계이다. 태스크를 정의하는 기준과 태스크의 동작 특성에 따른 내부 설정 등은 다음 장에서 다룬다.

4.2 RTOS 태스크 설정

RTOS 태스크를 설정하는 단계는 시스템 모델링을 통해 모듈 별로 정의된 태스크들의 구조와 태스크들을 주기적으로 실행하기 위한 동작구조, 우선순위, 주기 등을 정의하는 단계이다.

4.2.1 UML

RTOS 태스크 설정을 위해 UML을 통해 태스크의 상태도(State Diagram)와 흐름도(flow Chart)를 정의하고 태스크들 간의 동작 구조를 분석하여 RTOS 태스크 설정에 필요한 태스크 구조와 우선순위, 주기를 설정한다.

4.2.1.1 입력-처리-출력 태스크 상태도

RTOS에서 정의하는 태스크들은 PLC 프로그램의 동작 순서인 입력, 처리, 출력 순으로 태스크의 상태도를 따라 태스크의 상태를 변화시켜가며 주기적으로 동작하게 된다. 태스크의 상태는 실행을 기다리는 Ready 상태와 동작 상태인 Run 상태, 그리고 실행을 중단하고 다른 태스크에게 실행 권을 넘기기 위한 대기 상태인 Suspend와 Delay 상태가 있다.

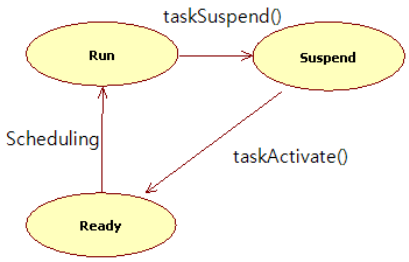


그림 16 Suspend 상태를 가지는 태스크 상태도
Fig. 16. Task state diagram with suspend state

(그림 16)은 Suspend 상태를 가지는 태스크의 상태로로 Suspend 상태는 taskSuspend() 함수를 통해 진입하며 taskActivate() 함수를 통해 Suspend 상태를 벗어나 Ready 상태로 진입한다.

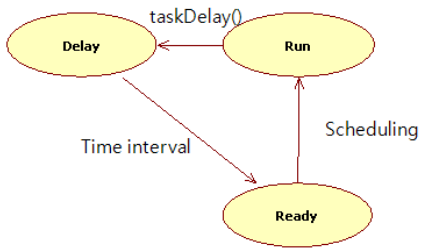


그림 17 Delay 상태를 가지는 태스크 상태도
Fig. 17. Task state diagram with delay state

(그림 17)은 Delay상태를 가지는 태스크의 상태도로 Delay 상태는 taskDelay() 함수를 통해 진입하며 일정 시간 이후 Ready 상태로 진입한다.

4.2.1.2 입력-처리-출력 태스크 순서도

PLC 프로그램을 통해 변환된 RTOS 태스크들은 입력-처리-출력 순으로 순차적으로 주기적으로 반복 실행되는 구조를 가진다.

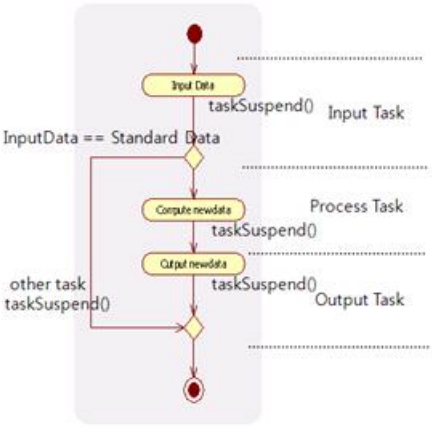


그림 18 입력-처리-출력 태스크 순서도
Fig. 18. Input-Process-Output Task flow chart

(그림 18)은 입력-처리-출력 태스크 순서도로 태스크들은 자신의 실행 뒤 다음 태스크에게 실행 권을 넘기기 위해 자신은 Suspend 상태로 이동한다. 입력 태스크의 경우 입력 값이 자신이 원하는 값과 일치하는 경우, 처리나 출력 태스크로 실행 권을 넘기지 않고 다른 태스크들도 Suspend 상태로 이동시켜 다음 주기의 실행을 기다리게 된다. 태스크를 입력, 처리, 출력의 태스크 순으로 구성하는 것은 다른 태스크들과의 자원 공유 시 발생하는 자원 공유에 따른 태스크들의 지연 현상을 막기 위함이다.

4.2.2 태스크의 종류

태스크는 작업의 실행 주기를 기준으로 주기적인(Periodic) 태스크와 비주기적인(Aperiodic) 태스크로 나뉘고 태스크의 구조에 따라 작업이 무한 반복되는지 혹은 한번 실행되고 종료되는지에 따라 끝이 있는 구조(Run-to-Completion Structure)와 무한반복 구조(Infinite loop Structure)로 나뉜다.

4.2.2.1 태스크의 주기성에 따른 분류

주기적으로 실행되는 태스크들은 시스템이 외부로부터의 특별한 처리가 없는 한 계속해서 반복적으로 수행되어야 하는 태스크들을 말한다. 비주기적으로 실행되는 태스크는 시스템의 특별한 제어가 필요한 경우 처리하는 태스크들이다. 이런 경우는 외부 입력 버튼으로 들어오는 신호의 처리와 내부적인 시스템에 오류가 발생한 경우의 처리를 나타내는 태스크이다.

4.2.2.2 태스크의 구조에 따른 분류

```
Int tMytask_startup()
{
        system_init();
        application_init();
        create_task();
        delete_StartupTask();
}
```

그림 19 끝이 있는 구조를 갖는 태스크 코드
Fig. 19. Task code of Run-to-Completion structure

(그림 19)는 끝이 있는 구조를 가지는 태스크의 코드이다. 끝이 있는 구조를 갖는 태스크는 한 번 실행 후 바로 종료가 되는 구조로 태스크의 재실행시에는 처음부터 실행을 다시 시작하는 구조를 가진다. 이러한 태스크들은 시스템의 초기화와 같은 작업을 한다.

```
Int tMytask_input()
{
        Set_Input_Memory();
        while(1){
                Get_Input_Data();
                Process_task_Activate();
                Suspned or Dleay_task();
        }
}
```

그림 20 무한 반복 구조를 갖는 태스크 코드
Fig. 20. Task code of infinite loop structure

(그림 20)은 무한 반복 실행하는 구조를 갖는 태스크의 코드로 자신이 원하는 작업을 끝낸 후 자신의 상태를 Delay 나 Suspend 상태로 전이시켜 스케줄러에게 실행 권을 넘기고 자신은 다음 실행을 기다리게 된다. Delay 상태를 가지는 태스크는 시스템의 On/Off 신호의 처리나 시스템의 긴급 상황을 처리와 같은 특정한 입력에 대한 처리를 하는 태스크들을 지정하고 Suspend 상태를 가지는 태스크는 시스템의 입력, 처리, 출력과 같은 순차적인 동작을 실행하는 태스크들로 지정한다.

4.2.3 태스크 우선순위

태스크의 우선순위는 스케줄러에 의해 태스크가 실행될 시 가장 먼저 실행되는 태스크를 선정하기 위한 기준이 된다. PLC 프로그램은 우선순위에 대한 정보가 없으므로 RTOS 기반의 태스크로 변환 시 각각의 태스크에 대한 우선순위를 추가적으로 지정해 주어야 한다.

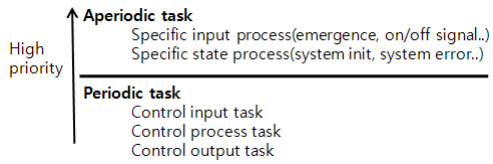


그림 21 태스크의 우선순위
Fig. 21. Task Priority

(그림 21)은 태스크의 주기성과 동작특성, 순차적인 실행 관계를 통해 태스크의 우선순위를 할당한 모습이다. 우선순위는 우선 태스크의 주기성을 기준으로 비주기적으로 동작하는 태스크들을 주기적으로 동작하는 태스크보다 더 높은 우선순위를 가지도록 한다. 이는 비주기적으로 동작하는 태스크들이 시스템의 특별한(Specific) 상황들을 처리하기 위한 태스크들이기 때문이다.

이후 태스크들의 우선순위 할당은 태스크들의 동작 특성과 태스크들의 순차적인 실행관계를 기준으로 정한다. 비주기적인 태스크 중에서는 외부 입력 신호에 대한 처리 태스크가 시스템 상태 처리 태스크보다 상대적으로 높은 우선순위를 가진다. 이것은 외부에서 들어온 입력신호의 경우 사용자에 의해서 시스템의 상태를 강제적으로 지정하는 태스크이므로 시스템의 상태와는 상관없이 더 빠르게 실행되어야 하기 때문이다. 주기적으로 실행되는 태스크들의 우선순위는 시스템 모델을 통해 정의 된 입력, 처리, 출력 태스크들의 순으로 할당한다. 이것은 태스크들의 우선순위를 통해 태스크들을 순차적으로 실행되도록 하기 위해서다

4.2.4 태스크의 주기

태스크의 실행 주기는 태스크를 Delay 상태로 전이시키는 taskDelay() 함수나 RTOS에서 제공하는 Timer를 통해 할당할 수 있다. Delay는 Delay 상태를 포함하는 태스크일 경우 사용되며 Delay 함수의 시간 지연 값을 통해 주기를 설정할 수 있다. 타이머는 Suspend 상태를 포함하는 태스크를 주기적으로 동작시키기 위해 사용되며 타이머 함수에 지정한 시간을 통해 태스크는 해당하는 주기가 되면 Ready 상태로 들어가게 되고 스케줄러에 의해 스케줄링 된다.

PLC 프로그램에서의 주기는 스캔타임을 통해 전체프로그램의 동작 주기를 정의한다. RTOS의 경우 주기는 타이머를 통해 결정되며 타이머는 모듈 별로 존재한다. 각각의 모듈의 주기는 PLC 프로그램의 스캔타임 값을 똑같이 정의하여 PLC 프로그램과 같은 동작 주기를 갖도록 한다.

S = PLC Scan Time
P = RTOS Task Period 일 때,
Pi(i=0,1,2 ….n) = S, n = Task의 수

4.2.5 태스크의 실행 구조와 모습

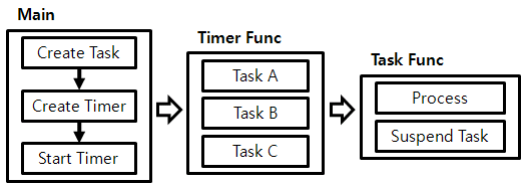


그림 22 태스크들의 동작 블록 다이어그램
Fig. 22. Operation block diagram of tasks

(그림 22)는 태스크들의 동작 블록 다이어그램으로 시스템은 초기화 과정에서는 태스크와 타이머를 생성하고 타이머를 실행한다. 타이머는 타이머함수(TimerFunc)를 실행하여 입력(Input), 처리(Process), 출력(Output) 태스크의 상태를 Ready 상태로 옮겨놓는다. 이후 태스크들은 스케줄러에 의해 우선순위에 따라 순차적으로 실행되며 실행 시 자신에게 지정된 동작을 하고 Suspend 상태로 태스크의 상태를 전이시켜 다음 주기에 다시 실행되길 기다리게 된다.

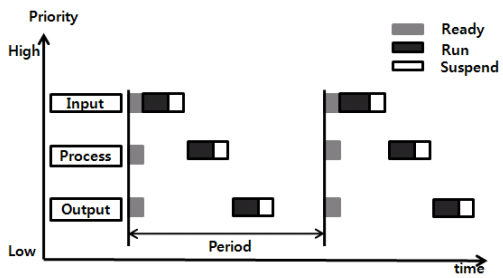


그림 23 입력-처리-출력 태스크의 실행과정
Fig. 23. Execution of Input-Process-Output tasks

(그림 23)은 입력, 처리, 출력 순으로 우선순위가 할당된 태스크들의 실행 과정이다. 태스크들은 우선순위에 따라 순차

적으로 실행된다.

4.3 PLC 프로그램의 재구성

PLC 프로그램의 재구성은 PLC 프로그램을 RTOS 태스크로 변환하기 위해 모듈 별로 재구성하는 것으로 시스템 모델을 통해 구성된 태스크 안에 들어갈 기능을 채우기 위한 작업이다. 이 작업을 통해 PLC 프로그램은 모듈 별로 그리고 태스크 별로 재구성된다.

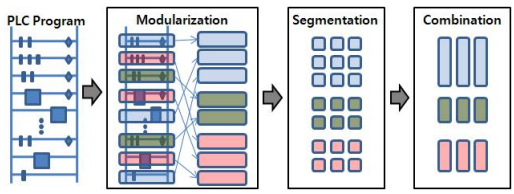


그림 24 PLC 프로그램의 재구성 절차
Fig. 24. Reconstruction Process of PLC program

(그림 24)는 PLC 프로그램의 재구성 절차로 PLC 프로그램은 모듈화(Modularization), 세분화(Segmentation), 조합(Combination) 과정을 거쳐 재구성된다.

4.3.1 모듈화

모듈화는 PLC프로그램을 통한 시스템 모델링에서 설명한 방법대로 모듈화를 진행한다.

4.3.2 세분화

모듈 별로 재구성된 PLC 프로그램은 세분화 과정을 통해 입력, 처리, 출력으로 나뉘게 되고 이러한 세분화 과정은 입력들의 논리 조합을 나타내는 코드와 PID 제어와 같은 제어 블록 코드로 구분하여 진행한다.

4.3.2.1 논리 코드의 세분화

논리 코드는 입력 값의 상태에 따라 출력이 변하는 PLC 코드를 말한다.

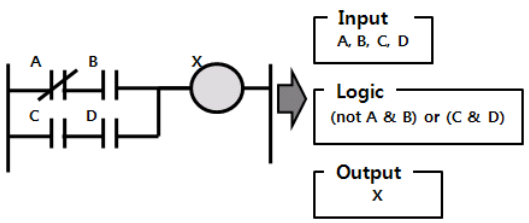


그림 25 논리 코드의 세분화
Fig. 25. Segmentation of logic code

(그림 25)는 하나의 작업을 나타내는 PLC 논리 코드를 세분화 과정을 통해 세분화한 모습이다.

4.3.2.1 제어 블록 코드의 세분화

제어 코드는 PLC 프로그램에서 블록 구조를 가지는 함수를 말하며, 입력 값의 변화에 따라 출력이 변하는 PLC 코드를 말한다. PID 제어는 입력 값과 피드백 된 값을 통해 출력이 결정되는 구조로 (그림 26)과 같이 세분화 한다.

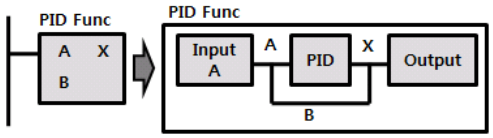


그림 26 제어 블록 코드의 세분화
Fig. 26. Segmentation of control block code

(그림 26)은 제어 블록 코드의 세분화로 A는 현재의 입력 값을 B값은 이전의 피드백 된 출력 값을 X는 현재의 출력 값을 나타낸다. 하나의 블록으로 표현되는 PLC 코드들은 세분화 과정을 통해 입력, 처리, 출력의 순으로 세분화 한다.

4.3.3 조합

조합 과정은 세분화 된 PLC 코드들을 모듈 별로 입력은 입력끼리, 처리는 처리끼리, 출력은 출력끼리 묶는 과정이다. 이러한 조합 과정을 통해 하나의 모듈에서 많은 태스크가 동작하여 생길 수 있는 시스템의 과부하를 막고 작업의 효율성을 가져올 수 있다.

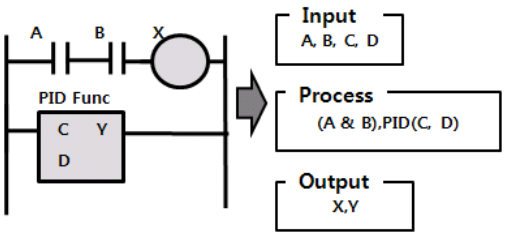


그림 27 PLC 코드의 조합 예
Fig. 27. Combination of PLC code

(그림 27)은 논리 코드와 제어 블록 코드를 세분화하여 입력, 처리, 출력별로 조합한 모습이다.

4.4 PLC 프로그램의 RTOS 태스크 변환

PLC 프로그램의 RTOS 태스크 변환 과정은 PLC 프로그램에서 사용된 문법을 RTOS API를 사용한 문법으로 바꾸는

과정으로 PLC 프로그램의 문법들은 RTOS의 API로 변환된다.

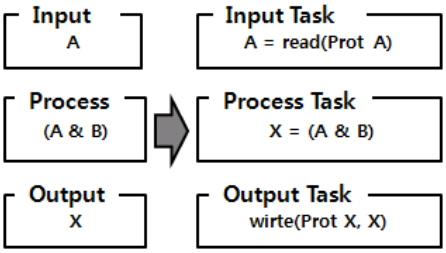


그림 28 변환 예
Fig. 28. Conversion example

(그림 28)은 변환 예로 세분화와 조합과정을 통해 구성된 입력, 처리, 출력 코드를 태스크에서 동작하는 코드로 변환시킨 모습이다.

V. PLC 프로그램의 RTOS 태스크 변환 예

5.1 저수지 수위 제어

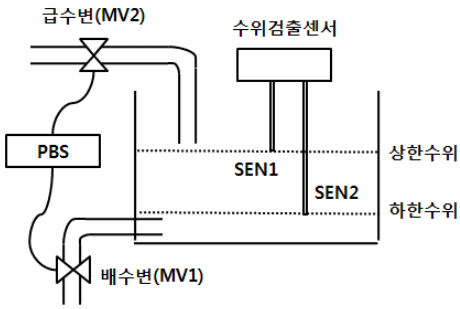
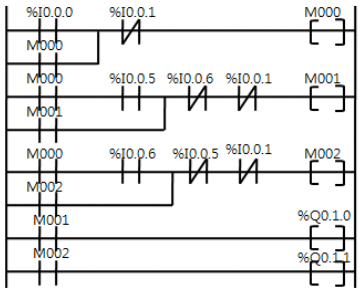


그림 29 저수지의 수위 제어 시스템
Fig. 29. Water level control system

(그림 29)는 저수지의 수위 제어 시스템으로 급수 변과 배수 변을 통해 저수지의 물의 양을 조절하고 수위 검출 센서를 통해 저수지의 수위를 제어하는 시스템이다.[10]

5.1.1 시스템 변환



번호	명칭	번호	명칭
%I0.0.0	가동 PBS	%Q0.1.0	배수변(MV1)
%I0.0.1	정지 PBS	%Q0.1.1	급수변(MV2)
%I0.0.5	상한 SEN1		
%I0.0.6	하한 SEN1		

그림 30 저수지 수위 제어 PLC 프로그램과 데이터 표
Fig. 30. PLC program and data sheet of water level control system

(그림 30)은 저수지 제어 시스템의 PLC 프로그램과 데이터 표로 저수지 제어 시스템은 구동 부와 조정 부로 시스템을 모듈 별로 정의 할 수 있다.

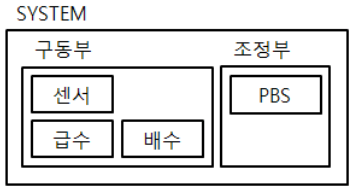


그림 31 저수지 제어 시스템 모델
Fig. 31. System model of water level control system

(그림 31)은 저수지 제어 시스템의 시스템 모델로 구동 부는 수위를 검출하는 센서와 수위를 조절하는 급수와 배수로 구성되어 있고 조정 부는 가동과 정지 버튼으로 동작하는 PBS(Push Button Switch)로 구성되어 있다. 구동 부에서 센서는 급수와 배수를 위한 입력이 되고 급수와 배수는 구동 부의 출력이 된다. 구동 부는 입력, 처리, 출력으로 조정 부는 입력과 처리로 태스크를 정의한다. 태스크들의 우선순위는 다음과 같이 정의한다.

구동부

Sensor Input Task Priority 〉 Process Task Priority 〉 Output Task Priority

조정부

PBS Input Task Priority 〉 Process Task Priority

```
Void tControlInput()
{
        A = read(%I0.0.0);
        B = read(%I0.0.1);
}
```

```
Void tControlProcess()
{
        if((m_bSystemOn || A) and not B)
                m_bSystemOn = TRUE;
        else
                m_bSystemOn = FALSE;
}
```

그림 32 조정 부의 태스크
Fig. 32. Task of Control module

```
void tOperateInput()
{
        UpSensor = read(%I0.0.5);
        DownSensor = read(%I0.0.6);
}
```

```
void tOperateProcess
{
        If((UpSensor or m_bMV1 ) && !DownSensor && m_bSystemOn)
                m_bMV1 = TRUE;
        else
                m_bMV1 = FALSE;

        if ((DownSensor or m_bMV2) && !UpSensor && m_bSystemOn)
                m_bMV2 = TRUE;
        else
                m_bMV2 = FALSE;
}
```

```
Void tOperateOutput()
{
        Write(%Q0.1.0, m_bMV1);
        Write(%Q0.1.1, m_bMV2);
}
```

그림 33 구동 부의 태스크 변환
Fig. 33. Task Conversion of operation module

(그림 32)와 (그림 33)은 저수지 제어 시스템의 태스크로 PLC 프로그램을 세분화, 조합 과정을 통해 입력, 처리, 출력 순으로 재구성된 작업들을 RTOS API를 통해 RTOS 태스크로 변환한 모습이다. 조정 부는 모듈화에서 정의한 입력, 처리 태스크로 구성된다.

5.2 자동문 개폐 제어

(그림 34)는 자동문 개폐 제어 시스템으로 문 안팎에 설치한 센서(SEN1, SEN2)를 통해 사람을 감지하면 문을 열고 일정 시간 이후 센서로부터 사람이 인지되지 않았을 때 문을 닫는 시스템이다. LS1과 LS2는 문의 열림과 닫힘을 검출하는 센서이다.[10]

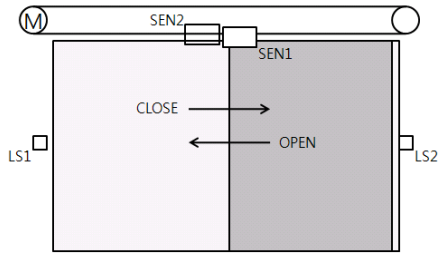
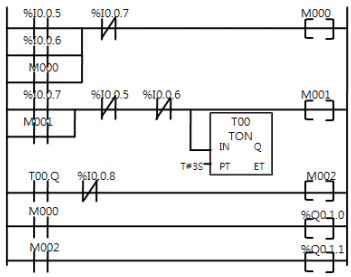


그림 34 자동문 개폐 제어 시스템
Fig. 34. Automatic door control system

5.2.1 시스템 변환



번호	명칭	번호	명칭
%I0.0.5	SEN1(문밖)	%Q0.1.0	문 열림
%I0.0.6	SEN2(문안)	%Q0.1.1	문 닫힘
%I0.0.7	LS1(열림 검출)		
%I0.0.8	LS2(닫힘 검출)		

그림 35 자동문 개폐 제어 PLC 프로그램과 데이터 표
Fig. 35. PLC program and data sheet of automatic door control system

(그림 35)는 자동문 개폐 제어 시스템의 PLC 프로그램과 데이터 표로 자동문 개폐 제어 시스템은 구동 부로 시스템을 모듈 별로 정의 할 수 있다.

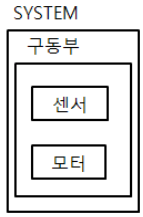


그림 36 자동문 개폐 제어 시스템 모델
Fig. 36. System model of automatic door control system

(그림 36)은 자동문 개폐 제어 시스템의 시스템 모델로 시스템은 자동문을 제어하는 구동 부로 구성된 모듈로 정의 할

수 있다. 구동 부는 문을 개폐하는 모터와 사람을 인식하는 센서, 문의 개폐를 확인하는 센서로 구성되어 있다. 구동 부에서 센서는 입력이 되고 모터는 출력이 된다.

```
void tOperateInput()
{
        OutSensor = read(%I0.0.5);
        InSensor = read(%I0.0.6);
        OpenSensor = read(%I0.0.7);
        CloseSensor = read(%I0.0.8);
}
```

```
void tOperateProcess
{
        if((m_bOpen || OutSensor || InSensor ) & ! OpenSensor)
                m_bOpen = TRUE;
        else
                m_bOpen  = FALSE;

        if((m_bOpen || OpenSensor) & InSensor & OutSensor)
                m_bTON = TON(TRUE);
        else
                m_bTON = TON(FALSE);

        if((m_bTON & !CloseSensor)
                m_bClose = TRUE;
}
```

```
Void tOperateOutput()
{
         Write(%Q0.1.0, m_bOpen);
         Write(%Q0.1.1, m_bClose);
}
```

그림 37 구동 부의 태스크 변환
Fig. 37. Task Conversion of operation module

(그림 37)은 자동문 개폐 제어 시스템의 변환된 태스크로 PLC 프로그램을 세분화와 조합 과정을 통해 입력, 처리, 출력의 태스크로 재구성하여 RTOS 태스크로 변환하였다.

VI. 변환 결과

PLC 프로그램의 RTOS 태스크 변환 시 생성되는 태스크의 수는 RTOS기반의 시스템에서 성능을 좌우 하는 중요한 요소가 된다. 그러므로 PLC 프로그램을 통한 RTOS 태스크 변환 시 생성되는 태스크의 수는 PLC 프로그램의 작업량과는 상관없는 정량화 된 수치일 필요가 있다..

본 논문에서 제안한 시스템 모델을 통한 RTOS 태스크 변환은 변환된 태스크의 수가 PLC 프로그램의 작업량과는 상관없이 시스템 모델링 과정에서 정의한 시스템의 모듈의 개수에 따라 정해진다.

태스크들의 수는 입출력 모듈들은 입력과 처리, 처리와 출력을 나타내는 두 개의 태스크로 변환되고, 입출력 모듈을 제외한 모듈들은 입력, 처리, 출력의 3개의 태스크로 변환된다.

변환된 태스크의 수는 다음과 같은 식으로 구할 수 있다.

$$N = M * 3 + I * 2 + O * 2 \quad \cdots\cdots\cdots\cdots \text{(수식 1)}$$

(수식 1)에서 M은 입력과 출력 모듈을 제외한 모듈의 수를 말하며, I는 입력 모듈의 수를, O는 출력 모듈의 수를 나타낸다.

표 1 시스템 모듈의 수와 태스크 수
Table 1. Number of system module and task

System	모듈의 수 (M, I, O)	PLC 작업	RTOS task
저수지 제어	1, 1, 0	5	5
자동문 개폐 제어	1, 0, 0	5	3

(표 1)에서 처럼 저수지 제어 시스템의 경우 모듈 하나에 입력 모듈 하나로 구성되어 있으므로, (수식 1) 을 통해

$$1 * 3 + 1 * 2 + 0 * 2 = 5$$

총 5개의 태스크로 구성된다.

자동문 개폐 제어 시스템의 경우 하나의 모듈로 구성되어 있으므로

$$1 * 3 + 0 * 2 + 0 * 2 = 3$$

총 3개의 태스크로 구성된다.

변환 결과를 통해 알 수 있듯이 본 논문을 통해 제시된 방법으로 변환된 태스크의 수는 모델링 과정에서 정의한 시스템 모델의 수에 따라 정량화 된다. 또한, 시스템의 추가적인 작업이나 작업의 수정에도 증가하거나 감소하지 않고 시스템 모듈의 추가나 제거 시 그 수가 변하게 된다. 이러한 정량 태스크의 수는 태스크의 수에 따른 시스템의 과부하를 예상하여 모듈의 수를 제한하거나 시스템의 성능을 조절할 수 있다.

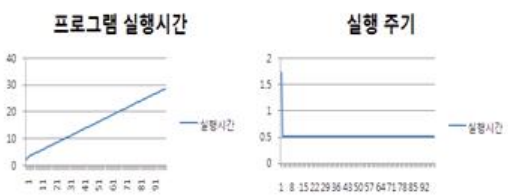


그림 38 저수지 시스템의 태스크 수행 결과
Fig. 38. execution result of water level control system

(그림 38)의 그래프는 저수지 제어 시스템의 RTOS 태스크를 수행 했을 때의 결과이다. 전체 태스크를 실행했을 때 프로그램의 실행 시간과 실행 주기를 나타낸 그래프로 나타낸 결과이다. 당연한 결과이지만 PLC 프로그램을 변환한 RTOS 태스크의 실행은 일정한 주기로 동작하는 것을 알 수 있다.

VII. 결론

본 논문에서는 PLC 기반의 시스템을 RTOS 기반의 시스템으로 변환하기 위한 방법으로 시스템 모델을 통한 변환 방법을 제안하였고 e-Printing 시스템과 저수지 제어 시스템, 자동문 개폐제어 시스템을 예로 들어 변환과정을 설명하였다.

시스템 모델을 통해 변환된 태스크는 모듈간의 독립성을 보장하는 모듈화 된 태스크로 시스템의 동작에 따른 특성을 모듈 별로 혹은 태스크 별로 반영 할 수 있어 시스템의 성능 향상을 가져올 수 있고 시스템의 유지 보수와 수정을 위한 추가적인 작업 시에도 해당하는 모듈에 대한 작업만 하면 되므로 비용적인 부분과 시간적인 부분에서도 향상을 가져온다.

변환 결과를 통해서는 시스템 모델을 통한 변환 시에도 PLC 시스템과 RTOS 시스템이 같은 성능을 보장함을 확인 할 수 있었고 변환된 태스크의 수를 통해 태스크의 수가 PLC 프로그램의 작업량과는 상관없이 모듈의 수에 따라 일정한 수가 생성됨을 확인 할 수 있었다.

Acknowledgement

본 논문은 2007년 국민대학교 교내연구비를 지원받아 수행한 연구이며 2008년도 교육과학기술부의 재원으로 과학재단의 지원(R01-2006-000-10864-0)을 받아 수행하였음.

참고문헌

〔1〕 Masao Ogawa, Yutaka Henmi, "Recent Developments on PC+PLC based Control Systems for Beer Brewery Process Automation Applications," SICE-ICASE, International Joint Conference, pp.1053-1056, Oct. 2006

〔2〕 Dave Gee, "The How's and Why's of PC Based Control," Pulp and Pager Industry Technical Conference, pp.67-74, June 2001

〔3〕 Lei Lin, Houjun Wang, "Research on Technique of Real-time Communication between Programmable Logic Controller and Microcomputer," ICCCAS, Vol. 2, pp. 1410-1413, June 2004

〔4〕 Jane W.S.Liu, "*Real-Time Systems*," Prentice Hall, pp. 130-134, 2000

〔5〕 Quing Li, "*Real-Time Concepts for Embedded Systems*," CMP Books, pp. 214-216, 2005

〔6〕 Windriver Instruments, *http://www.windriver.com*

〔7〕 Windriver Instruments, "*User's Guide 2.5*"

〔8〕 Windriver Instruments, "*VxWorks 6.3 Kernel Programmer's Guide*"

〔9〕 Windriver Instruments, "*VxWorks 6.3 Application Programmer's Guide*"

〔10〕 이광만, "*PLC 제어 이론과 프로그래밍*," 일진사, 2004년

〔11〕 임성수, "원격 실시간 제어 시스템을 위한 정적 프로그램 분석에 의한 보안 기법," 한국컴퓨터정보학회논문지, 제10권, 제3호,.75-88쪽, 2005년 7월

〔12〕 황월, "실시간 멀티프로세서 시스템에서의 태스크 스케줄을 위한 L-RE 좌표 알고리즘," 한국컴퓨터정보학회 논문지, 제12권, 제3호, 147-153쪽, 2007년 7월

저 자 소 개

김 제 웅

2006: 국민대학교 전자정보 공학사
2008: 국민대학교 컴퓨터 과학 이학석사.
현 재: 티맥스 코어
관심분야: 임베디드 시스템

임 성 수

1993: 서울대학교 컴퓨터 공학과 공학사
1995: 서울대학교 컴퓨터공학과 공학석사.
2002: 서울대학교 전기컴퓨터공학과 공학박사
현 재: 국민대학교 컴퓨터공학과 교수
관심분야: 임베디드 시스템