'91 韓國自動制御學術會議論文集 1991. 10.22~24

연속공정 자동화를 위한 Function Block Diagram형 제어언어의 설계 및 구현

°조영조, 윤태웅, 이준수, 오상록, 최 익, 김광배
한국과학기술연구원 제어시스템연구실

Design and Implementation of a Control Language for Continuous Process Automation : Function Block Diagram Approach

°Y. J. Cho, T. W. Yoon, J. S. Lee, S. R. Oh, I. Choy, K. B. Kim
Control System Lab., Korea Institute of Science and Technology

ABSTRACT

A graphic control language using function block diagram approach is designed and implemented, applicable to real-time control for continuous process automation system. The procedure implementing the control language is composed of three parts, editor, compiler, and executer. The editor generates the control algorithm file, which contains function block information in the text form, by menu-driven method on the color graphic screen. The compiler translates the contents of the control algorithm file to machine codes and their related data. Then, the executer generates a task that makes the machine codes executed at every sampling period in the target processor. The validity of the concept in its design and implementaion is assured by on-line simulation in the multi-function controller designed for continuous process automation.

1. 서 론

1974년 마이크로프로세서의 출현 이후 거의 모든 산업 공정에서 디지탈 컴퓨터는 공정 자동화 시스템의 제어기로 사용되어 왔다. 초반기 산업용 컴퓨터에서는 어셈블리어, BASIC, FORTRAN 등 일반 컴퓨터 언어가 공정제어에 그대로 사용되었으나, 산업용 컴퓨터가 시퀀스 제어용 PLC (Programmable Logic Controller)와 루프제어용 DDC (Direct Digital Controller) 등으로 전용화 됨에 따라 응용 목적에 따른 공정제어 전용의 제어 언어들이 출현하게 되었다. 현재 공정제어용 디지탈 컴퓨터에 사용되는 제어 언어에는 여러 종류가 있으나 주요한 4가지를 들면, 텍스트(text) 언어로 어셈블리어 형태의 IL(Instruction List)과 고급언어 형태의 ST(Structured Text) 언어가 있고, 그래픽 언어로 LD (Ladder Diagram)와 FBD(Function Block Diagram) 언어가 있다.[1]

공정제어용 디지탈 컴퓨터에 사용되는 제어 언어는 제어 알고리즘의 종류와 이를 수행하는 하드웨어의 구성 상태에 따라 구현 방법이 달라진다. 최근 공정 자동화에서 사용자 인터페이스가 강화되고 단일 하드웨어에 시퀀스 및 루프 제어 기능을 모두 포함시키는 추세에 따라, 공정제어 컴퓨터는 대부분 컴퓨터 지원 설계(CAD) 기법을 사용한 시퀀스 및 루프 복합 제어용 FBD 언어를 지원한다.[2][3] 그러나, 이러한 공정제어용 FBD 언어는 그 설계 및 구현에 대한 특별한 기준이 없어 제어기 하드웨어의 제작사마다 고유한 소프트웨어 패키지로 존재하고, 제작사의 기술 보호로 공정제어의 온라인 제어기에 응용되는 FBD 언어의 설계 및 구현 기법에 관한 구체적인 연구 보고는 거의 접하기 어려운 실정이다. 다만, 최근 디지탈 제어기의 구현에 관한 한 조사 보고에서 FBD 언어에 사용 가능한 코드 발생기(code generator)의 개념만이 소개된 바 있다.[4]

따라서, 본 논문에서는 공정제어 분야에서 제어언어의 중요성을 부각시키고 복잡한 제어이론을 컴퓨터 소프트웨어로 손쉽게 구현하려는 구체적인 시도로서, 연속공정 자동화 시스템에 온라인으로 응용되는 복합 다기능 제어기에 대해 FBD 언어를 설계, 구현하는 한 방법을 제안하고자 한다.

2. 연속공정 자동화를 위한 다기능제어기의 구성

제철공장의 냉간, 열간 압연공정이나 강관공정, 금속공장의 도금공정 등 연속공정의 자동화 시스템은, 생산관리 및 최적화를 위한 관리제어컴퓨터(SCC: Supervisory Control Computer)를 상위계층으로 하여, 다수 전동기들의 속도 및 장력제어 등 복합적인 기계 자동화를 담당하는 라인제어유닛(LCU: Line Control Unit)과, 세척, 도금, 탈수, 회석 등 계장제어 기능을 담당하는 공정제어유닛(PCU: Process Control Unit)이 실시간 네트워크를 통해 하위계층으로 연결되는 계층, 분산 구조를 취한다.[5] 본 논문에서 대상으로 하는 연속공정 자동화를 위한 다기능제어기(MFC: Multi-Function Controller)는 복합 기계 자동화 기능을 수행하는 라인제어유닛의 핵심부로, 수백점의 아날로그 / 디지탈 / 펄스 신호를 직접 입출력하고, 각종 시퀀스 처리를 위한 PLC(Programmable Logic Controller)와 전동기의 속도 / 전류 루프제어를 위한 EPC(Electrical Power Controller) 들을 필드버스(field bus)를 통해 연결하여, 시퀀스 및 루프 복합 제어 알고리즘을 실시간 온라인으로 처리한다.

다기능제어기는 그림1과 같이 VMEbus와 공유메모리를 통해 시스템 제어 및 조작자 인터페이스 (SC&OI: System Control and Operator Interface), 라인제어(LC: Line Control), 네트워크 인터페이스(NI: Network Interface) 등 3종류의 32비트 프로세서 모듈들이 연결되는 다중 프로세서 구조를 갖는다.

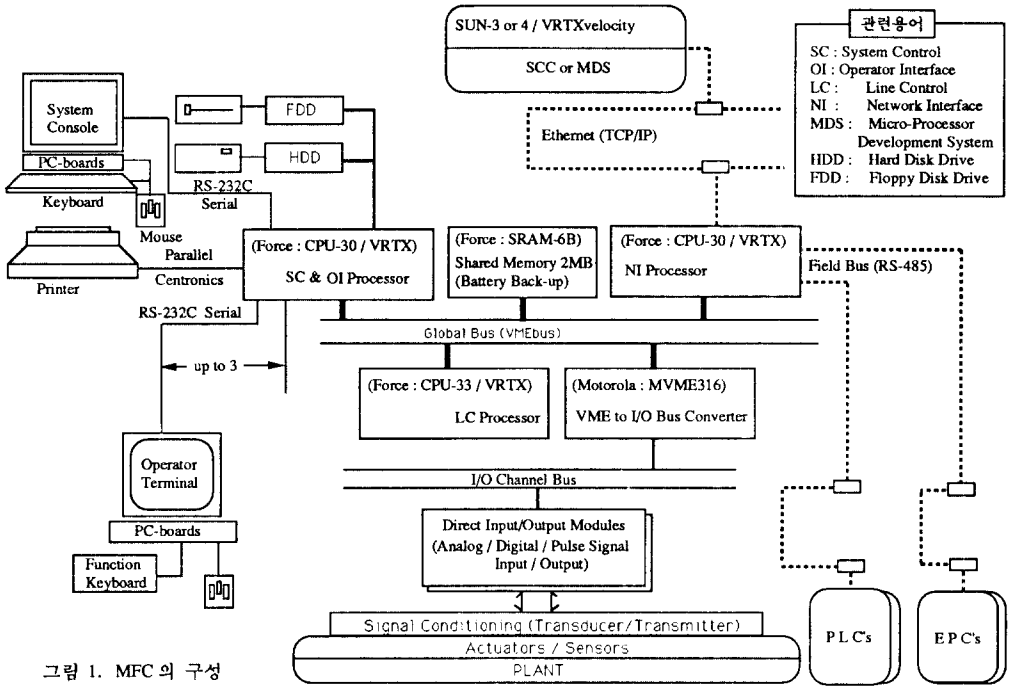

그림 1. MFC 의 구성

다기능제어기의 사용자 인터페이스로는 FBD 언어를 통해 라인제어 알고리즘을 구성하는 등의 시스템 조작 및 관리용 시스템 콘솔(System Console)과 단순조작 및 공정 모니터링용 조작자 터미날(Operator Terminal)이 있는데, 모두 VGA 칼라 모니터를 갖는 디스크없는 IBM-PC 호환기종으로 구현된다. 한편, MFC가 입출력하는 데이타는 각종 신호 입출력 모듈들을 통한 직접 입출력(DIO: Direct Input/Output) 데이타와, 관리제어 컴퓨터와의 Ethernet(ETH) 통신 데이타, 사용자 인터페이스 장치로 부터의 설정치(SETP: Set Point) 데이타, PLC/EPC와의 필드버스 통신 데이타 등 5종류가 된다.

3. 제어 알고리즘의 구성 및 수행 절차

다기능제어기에서 FBD 언어를 통해 제어 알고리즘을 구성하고 이를 수행하는 절차를 개념적으로 도시하면 그림2와 같다. 시스템 콘솔은 사용자에게 FBD 언어 편집기와 텍스트 편집기를 제공한다. 즉, 사용자는 FBD 언어 편집기의 그래픽 메뉴를 이용하여 다양한 기능 블럭들을 배치, 연결함으로써 제어 알고리즘을 콘솔 화면상에 블럭 다이아그램(block diagram)으로 그려내고 그 결과를 제어 알고리즘 화일로 저장할수 있으며, 필요한 경우 텍스트 편집기를 이용하여 제어 알고리즘 화일을 직접 편집할 수 있다.

SC&OI 프로세서가 구동하는 하드디스크 내에 구조화 텍스트(Structured Text) 언어의 형식으로 저장되는 제어 알고리즘 화일은 제어언어 컴파일러에 의해 기계어 코드와 그 연관 데이타로 바뀌어 공유 메모리의 정해진 작업번호 영역에 저장되어 제어작업 데이타베이스를 형성한다. 그러면, LC 프로세서는 입출력을 담당하는 각 프로세서에서 처리된 입출력 데이타베이스를 참조하여 다중 작업 스케줄

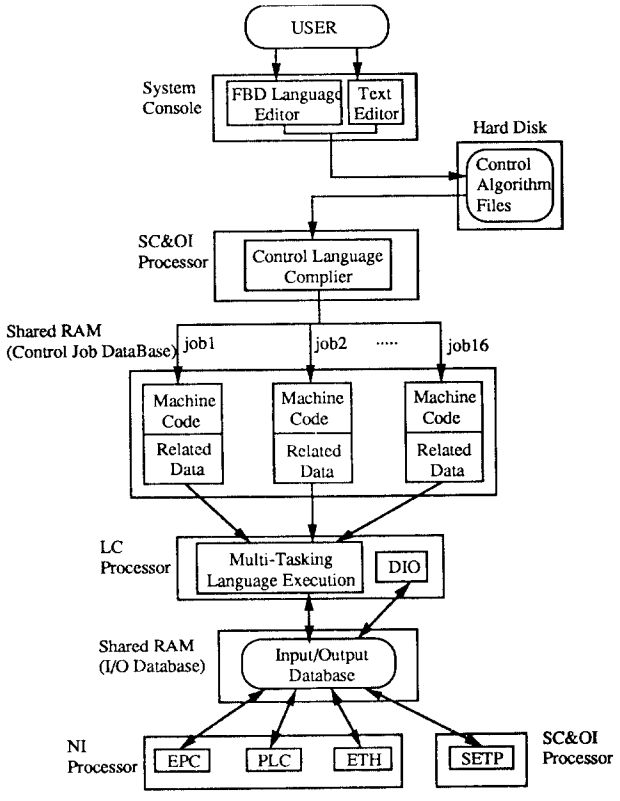

그림 2. 제어 알고리즘의 구성 및 수행 절차

(multi-tasking schedule)에 의해 정해진 작업번호의 컴파일된 기계어 코드를 수행하게 된다.

4. 기능 블럭의 정의

4.1. 원시 기능 블럭

FBD 언어를 설계하는데 가장 선행되어야 할 일은 제어 알고리즘 블럭 다이아그램을 그릴 때 사용하는 최소 단위로 원시 기능 블럭(Primitive Function Block)을 정의하는 일이다. 본 논문에서는 다기능제어기의 라인 제어에 사용되는 원시 기능 블럭으로 크게 5그룹의 60여개 블럭을 정의한다. 이 5가지 그룹별 동작 특성을 기술하면 다음과 같다.

(1) 신호 입력 그룹

외부로 부터 다기능제어기로 신호를 입력하는 기능 블럭 그룹으로, DIO, EPC, PLC, ETH, SETP 등 5종류 데이타 의입력장치에 따라 구분되는 dai (direct analog input), ddi (direct digital input), dpi (direct pulse input), tra (EPC read analog), trd (EPC read digital), prx (PLC read X device), pry (PLC read Y device), prm (PLC read M device), prr (PLC read register), era (Ethernet read analog), erd (Ethernet read digital), asp (analog set-point), dsp (digital set-point) 등 13가지 종류의 블럭이 이 그룹에 속한다. 기능 블럭의 종류에 따라 프로세서 번호와 장치 번호 및 채널번호등의 파라미터값이 요구된다.

(2) 신호 출력 그룹

MFC 로 부터 외부로 신호를 출력하는 기능 블럭 그룹으로, DIO, EPC, PLC, ETH, SETP 등 5종류 데이타의 출력 장치에 따라 구분되는 dao (direct analog output), ddo (direct digital output), twa (EPC write analog), twd (EPC write digital), pwx (PLC write X device), pwy (PLC write Y device), pwm (PLC write M device), pwr (PLC write register), ewa (Ethernet write analog), ewd (Ethernet write digital) 등 10 가지 종류의 블럭이 이 그룹에 속한다. 신호 입력 그룹과 마찬가지로 프로세서 번호, 장치 번호, 채널 번호등을 파라미터로 갖는다.

(3) 산술 연산(Arithmetic Computation) 그룹

일반 사칙 연산, 삼각함수 연산 등 제어에 필요한 기본적인 산술 연산을 담당하는 기능 블럭 그룹으로, add (addition), sub (subtraction), mult (multiplication), div (division), abs (absolute value), sqrt (square root), sin (sine), cos (cosine), tan (tangent), asin (arc sine), acos (arc cosion), atan (arc tangent), max (maximum), min (minimum), inv (inverse), scale 등 16가지 블럭이 여기에 속한다.

(4) 논리 연산 그룹

다기능제어기의 시퀀스 제어 기능을 위해 마련된 기능 블럭 그룹으로 and, or, not, exor (exclusive or), time (timer), cnt (counter) 등 6가지 기능 블럭이 현재 정의되어 있다.

(5) 제어 연산 그룹

다기능제어기의 루프제어에 사용되는 기능 블럭 그룹으로 lim (limit), dzon (dead zone), dely (delay), intg (integration), derv (derivative), filt (1st order ow pass filter), ldlg (lead-lag), pi (proportional+integral control), epi (PI control with error input), pid (proportional+integral+derivative control), epid (PID control with error input), ramp (ramp function generation), amux (analog multiplexer) 등 13가지 기능 블럭이 정의되어 있다.

위와 같이 정해진 원시 기능 블럭들은 제어 언어에서 구문을 정의할 때 사용되고, 아울러 다기능제어기의 LC프로세서 내에 기능 호출 라이브러리(function call library)를 형성하는데, 제어 언어 컴파일러를 거쳐 만들어진 제어 언어의 기계어 코드를 수행하는 과정에서 코드의 종류에 따라 기능 호출된다.

4.2. 매크로(MACRO) 기능 블럭

MFC가 라인 제어 기능을 수행하는데 있어 자주 사용되는 복잡한 연산에 대하여, 그 연산 기능을 하나의 매크로 기능 블럭으로 대치시킬 수 있다. 즉, 사용자는 임의의 명칭으로 원시 기능 블럭을 조합한 매크로 기능 블럭을 정의할 수 있는 것이다. 단, 현재 매크로 기능 블럭에 사용되는 원시 기능 블럭의 총 갯수는 50개로 제한되어있다.

5. 데이타베이스 구조

5.1. 제어 알고리즘 화일

제어 알고리즘 화일은 블럭 다이아그램을 구성하는 각종 기능블럭들이 필요로 하는 정보를 구조화 텍스트의 형태로 배치한 것으로 화일 구조는 다음과 같다.

```
(매크로 선언부)
%%
(원시기능블럭 및 매크로 호출부)
END
%%
(매크로 정의부)
```

(1) 매크로 선언부

제어 알고리즘에 사용되는 매크로 기능블럭에 대해 그 이름과 입력 갯수를 정의하는 부분으로, 하나의 매크로 기능블럭에 대해 다음과 같은 형식의 한문자열이 선언된다.

```
MACRO    NAME (l:m:n)
```

여기서, NAME은 사용자가 임의의 알파벳 문자 10개까지로 지정되는 매크로 이름, l은 매크로가 취하는 아날로그 입력의 갯수, m은 디지탈 입력의 갯수, n은 파라미터의 갯수를 나타낸다.

(2) 원시 기능블럭 및 매크로 호출부

제어 알고리즘을 구성하는 원시 기능블럭들과 그 연결 상태를 표시하거나, 매크로 사용시 매크로 기능블럭을 호출하는 부분이다. 블럭 다이아그램을 구성하는 각 기능

블럭은 고유의 블럭인식번호(block identification number)를 통해 인식되는바, 한 기능블럭의 블럭인식번호는 출력번호에 대응되고, 블럭의 연결상태에 따라 이 번호가 다른 블럭의 입력번호로도 작용하게 된다. 단, 신호출력 그룹에 속하는 기능블럭은 그 출력이 다른 블럭의 입력에 연결되지 않으므로 블럭인식번호가 출력으로 지정되지 않고, 신호입력 그룹에 속하는 기능블럭은 입력이 바로 입력장치가 되므로 입력부에 장치의 인식을 위한 장치번호가 지정된다. 한편, 기능블럭의 출력 데이타 종류를 구별하기 위하여, 블럭인식번호 앞에 아날로그인 경우 F를, 디지탈인 경우 D를 붙인다.

이 부분에 속하는 원시 기능블럭과 매크로 기능호출 블럭에 대한 각각의 명령어 형식을 예시하면 다음과 같다.

```
D2 = ddi(p0, d1, c2, "Label_1")
......
F10 = pid(F2, F4, 1, 3, 1)
......
F20 = MACRO SUBR(F10, 10)
......
```

위의 원시기능블럭 dai는 프로세서 번호 0, 모듈번호 1, 채널번호 2에 대응되는 디지탈 신호 입력장치를 구동하여 그 출력 데이타를 블럭인식번호 2로 대응시키고, pid는 블럭인식번호 2와 4로 지정된 두 블럭의 출력을 입력으로 하여 3개 파라미터(1, 3, 1)를 통해 비례/적분/미분 기능을 수행한 후 그 출력을 블럭인식번호 10에 대응시킨다. 또한, SUBR이라는 이름의 매크로 기능블럭은 블럭인식번호 10에 대응되는 pid 블럭의 출력을 입력하고 10을 파라미터로 하여 정의된 기능을 수행한 후 그 복귀값(return value)을 블럭인식번호 20에 대응시킨다.

(3) 매크로 정의부

매크로 선언부에서 선언되어 기능 호출되는 매크로 기능블럭들에 대해 그 알고리즘을 원시 기능블럭의 조합으로 나타내는 부분으로 전체 화일 구조상 일종의 서브루틴으로 작용된다. 매크로 정의부의 명령어 구조는 다음과 같다.

```
MACRO NAME (Mi1, ..., Mil, Nj1, ..., Njm, Pk1,...,Pkn)
{
    (Primitive Function Blocks)
    RETURN(Mi) 또는 RETRUN(Nj)
}
```

위에서 표시한 바와 같이, 매크로를 구성하는 원시 기능블럭들에 대해 그 출력이 아날로그인 경우 M, 디지탈인 경우 N문자를 블럭인식번호 앞에 대응시고, 기능호출시 넘겨받는 파라미터 값은 문자 P 이후에 숫자를 매겨 내부에서 사용하게 된다. 한 매크로 기능블럭의 끝에는 항상 RETURN 명령이 존재하는데, 복귀값은 내부 원시 기능블럭에서 블럭인식신호로 지정된 바 있는 Mi 또는 Nj 값이 된다.

5.2. 기능블럭 기계어 코드

다기능제어기의 공유 메모리 내에 저장되는 제어 언어의 기계어 코드와 그 연관 데이타의 구조는 다음과 같다.

```
struct    f_code    {
    unsigned char   code; /* code # /
    unsigned char   type;
    unsigned short int   blk; /* bid */
    union { /* address of input value */
        double    *ain;
        unsigned char   *din;
    } unin[IMAX];
    double    *prein[IMAX];
    union { /* addr. of output value */
        double    *aout;
        unsigned char   *dout;
    } uout;
    double    *preout;
    double    *para; /* param. addr. */
    unsigned short int   next;
};
```

한 원시 기능블럭을 구성하는 기계어코드는 위에서 정의된 바와 같이, 기능블럭의 code번호(code), 출력의 delay step 수(type), block id number(blk), 3개 까지의 입력변수에 대한 address (uin[3]), 입력변수의 과거값에 대한 address(prein[3]), 출력변수에 대한 address(uout), 출력변수의 과거 값에 대한 address(preout), 파라미터의 시작 address(para), 다음 수행하여야할 기능블럭 기계어코드의 offset address (next) 등으로 구성되어 있다. SC&OI프로세서의 제어언어 컴파일러에서는 이러한 데이타 구조의 값들을 컴파일 시에 만들어 주고, 구성된 많은 기능블럭들의 기계어 코드들을 서로 연결해주는 역할을 한다.

6. Function Block Diagram형 제어 언어의 구현

6.1. 제어 언어 편집기

FBD언어 편집기는 시스템 콘솔의 화면상에 제어 알고리즘을 블럭 다이아그램으로 구성하는 데 있어서 다루기 쉽고 사용자에게 친근하도록 메뉴드라이브 방식의 칼라그래픽 인터페이스를 제공한다. 편집기의 화면은 그림3의 예에서와 같이 상단에 7개의 메뉴명(Move, Save, Load, Print, Simulation, Macro, Quit)을 디스플레이하고, 대부분의 영역을 블럭 다이아그램 편집영역으로 한다. 블럭 다이아그램 편집영역에는 초기에 블럭 형태의 커서가 빨간색으로 표시되고, 화살표 key에 의하여 이 블럭커서가 4방으로 움직인다. 따라서, 사용자는 원하는 위치에서 return key를 누름으로서 각종 기능블럭을 선택하고, 선택된 기능블럭에 대해 Ins key에 의해 블럭들을 서로 연결할 수 있게 된다. 한 화면에 들어갈 수 있는 block 의 수는 수평방향(y-방향)으로 7개, 수직방향(x-방향)으로 5개이므로 총 35개인데, 편집을 시작할 때 x및 y방향으로 들어갈 수 있는 블럭 수를 사용자가 지정할 수 있게 되어 있으며, 한 화면보다 큰 영역을 사용자가 지정할 경우에는 커서블럭이 수평 또는 수직 방향의 양단에 있을 때 화면이 스크롤(scroll)된다.

한편, 화면의 편집영역에서 상단부와 하단부에는 초기에

각각 입력블럭과 출력블럭들만 지정할 수 있게 되어있고, 입출력 블럭을 지정하면 8문자이내의 Label 이름이 쓰여지므로 그 크기가 편집 영역 중앙의 연산블럭들보다 더 긴 블럭 형태를 취한다. 또한, 출력 블럭이 들어가는 화면 하단부의 열(row)은 move 메뉴를 사용하면 하단부로 화면이 스크롤되면서 움직이게 된다.

편집된 블럭 다이아그램은 save 메뉴에 의해 SC&OI 프로세서에 그림화일 및 제어 알고리즘 화일로 저장되며, 그림3의 블럭 다이아그램에 대한 제어 알고리즘 화일을 예를 들면 그림4와 같다.

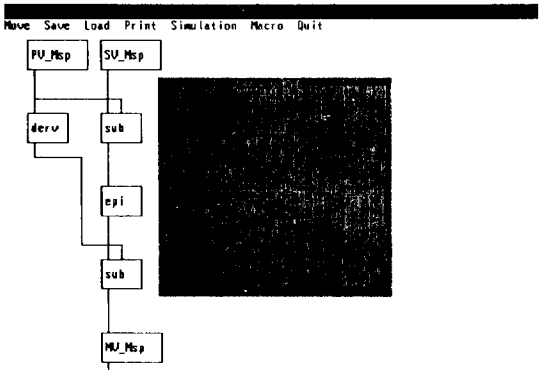

그림 3. Block Diagram 편집 화면

```
%%
F0 = dai(p0, d0, c0, "PV_Msp")
F1 = asp(d0, c0, 2, "SV_Msp")
F20 = derv(F0, 1)
F21 = sub(F41, F20, 1)
F41 = epi(F21, 3, 3)
F61 = sub(F41, F20, 1)
dao(F61, p0, d0, c0, "MV_Msp")
END
%%
```

그림 4. 제어 알고리즘 화일 (예)

6.2. 제어언어의 컴파일러

SC&OI프로세서의 디스크 내에 편집, 저장된 제어 알고리즘 화일은 SC&OI프로세서의 컴파일러 프로그램에 의해 기계어 코드와 그 연관 데이타로 바뀌어 공유메모리에 저장된다.

제어언어의 컴파일러는 Unix내에 시스템 서비스로 제공되는 lex (lexical analyzer)와 yacc (yet another compiler compiler) 등 두가지 유틸리티 및 C-언어에 의해 구현된다. 여기서, lex의 규칙에 따라 작성된 프로그램에서는 알고리즘 화일 내용이 되는 제어언어의 구문 규칙을 해석하여 여러가지 형태의 토큰(token)을 제공하고, yacc의 프로그램 규칙에 따라 작성된 파서(parser)에서는 이 토큰들의 배치 방식에 따라 명령어의 기능을 구분하여 기계어 코드 및 그 연관 데이타를 만들어 낸다.

컴파일러의 lex 프로그램은 제어 알고리즘 화일의 구문 구조에 따라 13개의 토큰을 정의하고 이를 파서로 전달하며, 기호와 특수문자는 토큰의 형태가 아닌 기호 그 자체로 파서에 전달한다. 한편, C-언어의 comment와 같이 /* comment */ 의 형태로 된 구문은 lex프로그램에서 그냥 skip된다. 그러면, 컴파일러의 파서 기능을 하는 yacc 프로그램에서는 lex 프로그램으로 부터 토큰 및 특수기호를 전달받고, 토큰의 배치 양식에 따라 명령어들을 해석한다. 즉, 앞서 기술한 제어 알고리즘 화일의 기능 블럭에 해당되는 명령어 하나 하나에 대하여, 파서는 기계어 코드의 데이타 구조에 대한 각 요소 데이타들을 만들어내고, 기계어 코드와 함께 해당 입출력 변수 및 파라미터 영역을 공유 메모리에 지정하게 된다. 특히, 한번 정의되어 여러번 기능호출되는 매크로 정의부에는, 호출될 때마다 다른 내용의 기계어 코드를 갖도록 컴파일러의 PC(Program Counter)개념이 적용되어 데이타가 구성된다. 또한, 한 알고리즘 화일에 대해 컴파일된 데이타는 공유 메모리내의 제어작업 데이타 영역중 100KB의 한 작업영역만을 지정하게 되어, LC프로세서가 총 16개의 제어 작업을 수행할 수 있도록 컴파일러가 설계되었다.

6.3. 제어 언어의 수행

컴파일되어 공유메모리의 제어 작업 영역에 저장되는 제어언어의 기계어 코드들은 LC 프로세서의 다중처리(multi-tasking) 방식에 의해 온라인으로 수행된다. 한 제어 알고리즘 화일에 대한 제어언어의 기계어 코드내에는 모두, 앞서 기술한 데이타 구조에서 나타난 바와 같이, 다음 수행되어야 할 기계어 코드의 포인터(pointer) 정보가 포함되어 있어, 수행기(executer)는 한 기계어 코드의 수행을 끝내고 이 포인터가 가리키는 다음 기계어 코드를 그 이후에 수행하는 일련의 과정을 한 샘플 주기 내에 수행하게 된다. 여기서, 기계어 코드내의 다음 포인터 값(next)이 -1이면, 한 샘플 주기내에 끝내야 할 마지막 기계어 코드임을 나타내는 것이다.

6.4. 실시간 시뮬레이션

제어 언어의 구현 방식에 대한 타당성 검토 및 다기능 제어기의 동작 실험을 위하여 임의의 2차 모의 플랜트에 대해 다음과 같은 제어 알고리즘을 제어언어를 통해 구현하였다.

(모의 플랜트)

$$\frac{1}{s^2+s+1}$$

(제어 알고리즘)

```
MACRO PID(2|0|3)
%%
stime(0.1)
F1 = dai(p0, d0, c0, "ProcVal")
F2 = asp(d0, c0, 50, "Setpoint")
F10 = MACRO PID(F2, F1, 4, 3, 1)
dao(F10, p0, d0, c0, "ManiVal")
```

```
END
%%
MACRO PID(M1, M2, P1, P2, P3)
{
      M3 = sub(M1, M2, 1)
      M4 = epi(M3, P1, P2)
      M5 = derv(M2, P3)
      M7 = sub(M4, M5, 1)
      RETURN(M7)
}
```

모의 플랜트는 4차의 Runge-Kutta 방법으로 시뮬레이션하되 제어 알고리즘의 주기가 0.1초임을 고려하여 0.01초의 주기를 갖도록 하였고, 온라인 실시간 동작 실험을 위하여 그 입출력을 아날로그 신호 입출력 모듈의 각 채널로 지정하였다. 또한, 0.01초의 샘플주기를 갖는 플랜트 시뮬레이터를 한 프로세스로 취급하여, 제어 알고리즘 수행기 프로세스와 함께 LC 프로세서내에서 다중 처리되도록 하였다. 그림5는 실시간 시뮬레이션을 통해 D/A변환된 모의 플랜트 출력을 오실로스코프 파형으로 본 것으로, 예상된 제어 출력 특성을 보인다.

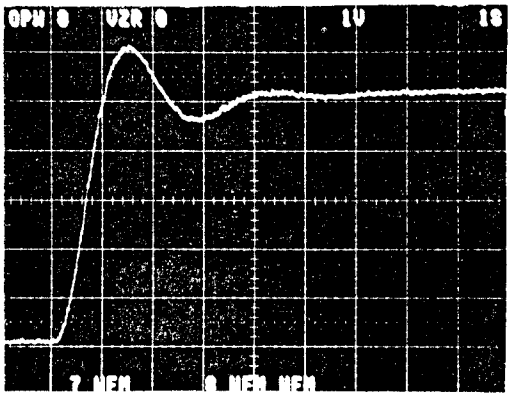

그림 5. 온라인 시뮬레이션 결과

7. 결 론

본 논문에서는 연속공정 자동화시스템에 실시간 온라인으로 적용되는 라인제어용 다기능제어기에 대해 컴퓨터 지원 설계 방식의 Function Block Diagram 제어 언어를 구현하는 한가지 방법을 제안하고, 실시간 시뮬레이션을 통해 그 타당성을 확인해 보았다. 특히, 본 논문에서 제시한 제어 언어를 위한 데이타베이스 구조는 복잡한 제어 알고리즘의 구성시 생기는 컴파일, 링크, 다운로드 등에 소요되는 시간을 극소화하고, 제어 알고리즘의 변경도 매우 용이하게 할 수 있도록 설계되었다. 이 제어 언어의 구현을 통해 제시한 기능 블럭의 묘사와 데이타베이스 구성 방식 및 편집기, 컴파일러 등의 개념은 연속공정 이외의 모든 자동화 공정 제어에서도 공통적으로 사용 가능할 것으로 보인다.

참고문헌

[1] 권욱현, 변대규, "프로그래머블 콘트롤러", 전기학회지 제37권 4호, 1988년 4월

[2] Thomas Pauly, "The ABB Master System Philosophy", Control Engineering, pp 5-8, August 1989 Vol.II

[3] George J. Blickley, "Multiloop Control Integrated with Logic and Setpoint Programming", pp 124-125, Sept. 1988

[4] H. Hanselmann, "Implementaion of Digital Controller", Automatica Vol.23, No.1, pp 7-32, 1987

[5] 김광배 외, "연속공정 자동화용 다기능 제어시스템의 개발", '91 로보틱스및자동화연구회워크샵, pp 107-113, Mar. 1991.