論文94-31B-11-5

공정 제어기 구성을 위한 제어 언어에 관한 연구

(A Study on a Control Language for the Configuration of Process Controllers)

徐暢浚*, 金炳國*

(Chang-Jun Seo and Byung Kook Kim)

要約

본 논문에서는 대규모 공정 제어 시스템에 대한 다중루프 제어기를 구성하기 위해 고급 공정 제어 언어를 개발하였다. 다중루프 제어기를 구성하기 위해 필요한 기본 기능들을 기능블럭으로 정의하고, 이를 간단하고 이해하기 쉬운 아스키 코드들의 열인 기능코드로 표현하였다. 공정에 대한 제어 알고리듬은 기능코드들의 순서적 나열에 의해 구성된다. 다양한 적용 대상 하드웨어 환경에서 사용하기 위해 기능코드로 구성한 제어 프로그램을 C-언어 프로그램으로 변환하는 변환 프로그램을 구현하였다. 제안된 제어언어는 사용하기 간편하고 쉬우며 확장이 가능하고 다양한 제어시스템들에 적용될 수 있다. 시뮬레이션 결과는 제안된 제어언어가 실제 환경에서 사용이 유효함을 보여준다.

Abstract

In this paper, a high-level process control language is developed to construct multi-loop controllers for large scale process control systems. Function blocks are defined which are basic functions necessary to configure a multi-loop controller. Each block is presented to the function code which is a line of ASCII codes and has the characteristics to be simple and to be easily understood. A control algorithm for a process is attained by means of the arrangement of function codes with order. In order to be used to various environments of target hardwares, a transformation program is prepared that transfers a control program configured by function codes to a C-language program. The proposed control language is easy and simple to use, possible to expend, and able to apply to various control systems. Simulation results are included to show the availability for the usage of the proposed control language in real world.

I. 서 론

마이크로프로세스의 발달은 많은 특수한 용도의 장치들을 모듈화되고 프로그램이 가능한 장치들로 대신

*正會員, 韓國科學技術院 電氣 및 電子工學科
(Dept. of Elec. Eng., KAIST)

接受日字 : 1994年 3月 23日

할 수 있게 하였다. 이러한 장치들은 기존의 것들에 비해 저렴화, 규격화등의 장점을 가지고 있고 많은 경우에 그 장치의 효율성, 신뢰성등이 소프트웨어(software: S/W)에 의해서 좌우되게 되었다. 그래서 보다 효과적인 S/W의 개발을 위해서 마이크로프로세서(micro-processor)고유의 어셈블 언어를 사용한 프로그래밍이 아닌 고급언어가 필요하게 되었다. 일반적으로 많이 알려진 고급언어(C, Pascal등)들은 많은 장점을 갖고 여러 분야에 사용되고 있으나 특정 분야에 대해서는 비효율적일 수 있다.

제어분야에서 실제로 제어 알고리듬을 작성하는 사람에게 고급언어의 복잡한 구조와 프로그램기법을 익혀서 프로그래밍하게 하는 것은 현실적으로 타당하지 않다. 이런 이유 때문에 제어를 위한 고급언어로써 concurrent Pascal, modular[1], realtime BASIC for CAMAC[2]등이 개발되었고 이들 실시간 제어용 고급언어들은 기존의 일반 고급언어를 제어 목적에 맞게 변형한 형태를 취하고 있다. 그리고 Microcont[2], Block Diagram Language[3]등은 언어의 구조적인 면이나 융통성 등에서 앞의 언어들보다는 저급하나 다른 방향에서 제어목적에 맞게 개발된 제어언어들이다.

본 논문에서는 공정제어에 직접적인 응용을 목적으로 공정제어에 적합한 새로운 제어언어를 제안하고자 한다. 이는 제어 알고리듬 구성에 필요한 기본적인 기능들을 찾아 기능블럭으로 정의하고, 원하는 제어 알고리듬들을 기능블럭을 적절히 이용하여 구성하는 방식으로 이룩한다. 이러한 방식은 사용자에게 제어 알고리듬 구성에 필요한 기본적인 기능인 기능불럭의 역할만을 익히도록 함으로 손쉬운 사용을 허락한다. 제안될 제어언어의 적용 범위를 넓히기 위해 제어언어가 실제 수행될 target H/W에 독립적인 구현 방법을 또한 제시한다. 그리고 제안한 제어언어를 실제 공정 제어 시스템을 모사하는 시뮬레이터에 적용하여 유용성을 보인다.

본 눈문의 구성은 I장 서론에 이어 공정 제어언어의 구성 형식을 II장에서 보이고 그 구현 방법을 III장에서 보인다. IV장에서 서울 화력 4호기 보일러 시뮬레이터에 제안된 공정 제어언어를 적용한 예를 보이고 V장에서 결론을 맺는다.

Ⅱ. 공정 제어 언어

일반적으로 공정을 제어하기 위해 사용되는 제어 알고리듬은 다입력 다출력의 구조이며 내부적으로 여러개의 로컬 루프(local loop)가 상호 연관성을 갖고 이루어진 다중 루프(multi-loop) 제어 구조를 갖고 있다. 이것은 제어 알고리듬의 설계를 한꺼번에 모두를 취급할 수 없게 한다. 그래서 대규모 공정의 제어를 위하여 제어기 구성에 있어서 모듈(module)화에 의해 확장성을 갖고 복잡한 제어 시스템의 구성을 용이하게 하는 것이 필요하다. 이러한 것을 가능하게 하는 것으로 configurable 제어기를 생각할 수 있다.

제어기를 구성하는 방법에 따라 여러 다른 형태의 configurable 제어기를 생각할 수 있으나 본 논문에서는 다음과 같은 형태의 configurable 제어기를 제안한다. 우선 제어 알고리듬을 구성하는데 있어 자주 사용되거나 유용한 그리고 기본이 되는 기능들을 추출하여 이것을 기능블럭으로 정의한다. 그리고 나서 기능블럭을 이용하여 하나의 원하는 제어 알고리듬이 되게끔 기능블럭들을 효과적으로 연결하는 것이다. 이러한 방법으로 제어 알고리듬을 구성할 경우 구성자는 각 기능블럭에 대한 상세한 내부구조는 알 필요가 없으며 단지 그 기능블럭의 기능과 입·출력의 내용만을 알면 된다.

기능블럭에 의한 제어기 구성방법을 실제로 사용하기 위한 표현 방법에는 여러 가지가 있을 수 있겠으나 기능블럭에 의한 제어기 구성방법을 잘 표현할 수 있는 프로그래밍 언어를 적절히 정의하고 이러한 언어에 의한 프로그래밍으로써 제어기의 제어 알고리듬을 구성하는 방법이 유용할 것이다. 왜냐하면 일반적으로 디지탈 컴퓨터에 의한 알고리듬의 구성은 프로그래밍언어에 의한 프로그래밍을 의미하고, 컴퓨터 사용자는 프로그래밍 언어에 의한 프로그램 작성에 익숙해 있기 때문이다. 그러나 프로그래밍 언어가 복잡한 구조를 가지고 쉽게 이해할 수 없는 코드(code)들을 사용하여 정의되었다면 이러한 언어는 사용가치가 없을 것이다. 그래서 프로그래밍 언어가 유용하게 사용되기 위해서는 하고자 하는 내용을 잘 표현할 수 있는 것은 물론이고 구조가 간단하고 사용되는 코드들이 쉽게 이해할 수 있는 것이어야 한다.

본 논문에서는 기능블럭에 의한 제어기 구성방법을 잘 표현하는 제어언어를 제안한다. 개괄적인 언어의 구조와 제어기 구성을 설명하면 다음과 같다. 우선 기능블럭의 입출력을 쉽게 나타낼 수 있고 다른 기능블럭과의 구별이 쉬우며 그 블럭의 기능을 쉽게 알 수 있는 형태의 각 기능블럭에 대한 하나의 코드 즉 기능코드를 정의한다. 이것은 하나의 기능블럭에 하나의 기능코드가 대응되는 관계를 갖는다. 그래서 제어 알고리듬의 구성은 기능블럭의 유기적인 연결관계로서 표현될 수 있는 제어 알고리듬을 기능코드에 의해서 전후 입출력 관계에 맞게 순서적으로 나열하는 방식으로 프로그램 파일을 작성함으로써 이루어진다. 여기서 기능코드의 나열로써 작성한 프로그램 파일을

configuration file이라 명명한다.

기능코드와 configuration file에 대한 표현 형식은 다음과 같다.

1. 기능코드(function code)

기능블럭을 ASCII문자열로 표현한 기능코드의 표현 형식은 다음과 같다.

블럭 번호 = 기능코드 이름 (입력 포트 번호 리스트 : 파라메타 값 리스트)

여기서 리스트는 해당 항목이 있는 경우와 없는 경우 모두를 포함하며 전자의 경우는 하나 혹은 여러개를 가질 수 있음을 의미한다. 여러개일 경우는 항목들의 구분을 컴마(,)로 하며 항목이 존재하지 않을 경우는 비워둔다. 각 기능코드는 configuration file에서 한 줄(one line)에 해당되며 각 부분에 대한 구체적인 규칙 및 의미는 다음과 같다.

i) 블럭 번호: 기능블럭(코드)의 나열로써 제어기를 구성할 때, 각 블럭(코드)에 중복없이 할당되는 음이 아닌 정수인데 그 블럭의 출력포트들에 할당된 번호(출력포트 번호)중 가장 작은 것이다. 그리고 출력포트 번호는 블럭 번호에서 하나씩 증가하여 할당되며, configuration file에서 연속된 기능 코드 사이의 블럭 번호의 할당은 뒤지는 기능 코드의 블럭 번호가 선행 기능 코드의 블럭 번호 보다 선행기능 코드의 출력 포트 갯수 만큼 크거나 같도록 하여진다.

ii) 기능 코드 이름: 기억하기 쉽고 기능을 적절히 표현할 수 있는 알파벳과 숫자로 이루어진 문자열로써 시작은 항상 알파벳 이어야 하지만 길이에는 특별한 제한이 없다. 서로 다른 기능 코드는 다른 기능 코드 이름으로 표현 되어야 한다.

iii) 입력 포트 번호: 기능 블럭의 각 입력 포트에 할당된 번호인 입력 포트 번호는 입력을 받게 될 자기 자신 혹은 다른 기능 코드의 출력 포트 번호이다. 이것은 기능 코드들 사이의 연결 상태를 나타내는데, 여러개의 입력 포트가 있을 경우에는 기능코드의 해당란에 모든 번호들이 반드시 기재 되어야 하며 입력 포트의 순서에 맞게 또한 나열 되어야 한다.

iv) 파라메타 값: 기능 코드가 자기 고유의 기능을 수행하기 위해 사용하는 숫자를 말하는데, 모든 기능 코드에 존재하는 것은 아니며 기능 코드에 따라 갯수나 각 파라메타의 형(정수 혹은 소수)이 다를 수 있다. 그러나 파라메타의 형에 대해서는 사용자가 구분하여 사용할 필요가 없다. 기능 코드가 파라메타를 갖고 있는데도 불구하고 파라메타 해당란에 값을 기재하지 않았다면 미리 지정된 default 값이 파라메타 값으로 사용된다. 이때 여러개의 파라메타가 있는 경우에 값을 기재하지 않더라도 각각을 구분하는 컴마(,)는 반드시 있어야 한다.

다음은 첫번째 입력에 첫번째 파라메타 값을 곱한 값과 두번째 입력에 두번째 파라메타 값을 곱한 값을 더하여 출력하는 기능을 갖는 기능 코드의 예를 나타낸다.

3 = sumII (1, 2 : 10.0, 15.0)

여기서 3은 블럭 번호이고 1과 2는 입력 포트 번호이고 10.0과 15.0은 파라메타 값이다.

구현된 기능 코드는 부록 A에 나타내었다. 기능코드 리스트에서 알 수 있듯이 일반 연산을 하는 기능 코드들(예를 들면 SUMII, MULT, DIVID 등), 논리 연산을 하는 기능 코드들(예를 들면 ORII, ANDII, QOR 등),그리고 아날로그 제어의 핵심인 동적 특성을 갖는 기능 코드들(예를 들면 PIDPVSP, SMITHP 등)로 이루어져 있다.

2. Configuration File

Configuration file은 기능블럭으로 구성될 수 있는 제어 알고리듬을 기능코드의 나열로서 표현한 프로그램 파일이다. 이러한 file 의 구성 형식과 요소는 다음과 같다.

i) Configuration file: ASCII 문자로 이루어진 text file인데 최대 블럭 번호의 정의문, 최대 출력 포트 번호의 정의문, 기능 코드, comment문, blank line등으로 구성된다.

ii) 최대블럭번호의 정의문: configuration file에 존재하는 기능 코드에 부여된 가장 큰 블럭 번호 혹은 그보다 큰 정수를 C-언어의 define문 형식에 맞게 *MAX_BLK_NUM*에 정의한다. 즉 만약 10이 최대 블럭 번호와 같거나 큰 수라면 다음과 같다.

```
#define MAX_BLK_NUM 10
```

iii) 최대 출력 포트 번호의 정의문: configuation file의 기능코드에 최대로 크게 할당된 출력포트번호 혹은 그보다 큰 정수를 C-언어의 define문 형식으로 *MAX_OUT_NUM*에 정의한 것이다. 만약 그 수가 15라면 다음과 같다.

```
#define MAX_OUT_NUM 15
```

iv) 기능 코드: 앞절의 설명과 같다.

v) Comment문: '!' 다음부터 line의 끝을 알리는 return 문자까지의 문자열로써 configuration file의 이해를 돕거나 설명을 위해 사용 될 수 있다.

vi) Blank line: ASCII 문자가 없는 line이 존재

하는 것을 허용한다.

최대 블럭 번호 및 최대 출력 포트 번호 정의문은 configuration file을 C-언어 프로그램으로 바꾸어 target processor에서 실행할 때 사용되는 데이타베이스(database)에 대한 메모리 요구량을 가장 작게 미리 지정해 주어 불필요한 메모리의 할당을 억제하기 위함이다. 기준값으로부터 벗어난 정도를 계산하는 의미를 갖는 기능 블럭들로 표현된 그림 1에 대응하는 configuration file의 예를 아래에 나타내었다.

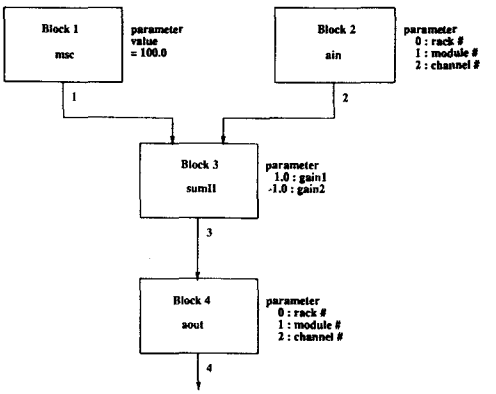


그림 1. 기능블럭에 의한 알고리듬의 구성예
Fig. 1. Example of an algorithm construction by function bolcks.

```
! This is an example of configuration file
#define MAX BLK NUM 4
#define MAX OUT NUM 4
1 = msc(:100.0)          !manual set constant
2 = ain(:0,1,1)              !analog input
3 = sumII(1,2:1.0,-1.0) !weighed summation
4 = aout(3:0,0,0)            !analog output
!End of example file
```

이것은 1번블럭의 출력인 100.0과 외부로 부터의 입력인 2번 블럭의 출력의 차를 외부로 출력하는 기능을 수행한다.

Ⅲ. 공정 제어 언어의 구현

일반적인 프로그래밍 언어가 그러하듯 기능코드에 의해 작성된 configuration file 또한 target hardware(H/W)에서 직접 수행될 수 없다. 즉 target이 그 코드들을 직접 알지 못한다. 그래서 target H/W에서 직접 수행될 수 있는 코드로 변환해 주어야 한다. 만약 target으로써 어떤 H/W가 고정적으로 선택되었고 그 H/W에 맞는 변환이 이루어졌다면 configuration file이 갖고 있는 일을 수행할 수 있을 것이다. 그러나 일반적으로 target으로 사용될 수 있는 H/W의 종류는 여러가지가 될 수 있다. 이럴 경우 target H/W가 달라짐에 따라 그 H/W에 맞는 변환 프로그램을 준비해야 한다. 이것은 큰 제약조건이 아닐 수 없다.

본 논문에서는 configuration file로부터 target에 맞는 코드를 직접 생성하지 않고 그 중간단계로서 C-언어 프로그램으로의 변환과정을 둔다. 즉 configuration file을 우선 C-언어 프로그램으로 변환한다. 그리고나서 C-언어 프로그램을 target에 맞게 컴파일하여 실행 코드를 생성한다. C-언어 프로그램으로부터 target에 맞는 실행코드를 생성하는 컴파일러는 일반적으로 시중에서 쉽게 구할 수 있으므로 여러종류의 target H/W에 대해서 직접 작성할 필요가 있다. 그래서 configuration file로부터 C-언어 프로그램으로의 변환만을 준비하면 된다. 이러한 방식은 target H/W의 선택에 상관없이 사용될 수 있는 장점을 갖는다. Configuration file을 C-언어 프로그램으로 변환하는 방법에는 여러가지가 있겠으나, 각 기능 코드가 수행해야 할 고유 기능을 이행하는 서브루틴(subroutine) 함수를 우선 C-언어로 작성하고, 이러한 함수들을 configuration file의 기능 코드의 열거 순서에 따라 호출함으로서 파일의 전체 기능을 수행케 하는 총괄 함수를 작성하는 방법이 합리적 이고 용이하다. 따라서 C-언어 프로그램으로의 변환은 정의된 기능코드 모두에 대해 기능 코드가 갖는 기능을 수행하는 C 함수를 직접 작성하고 제어 알고리듬을 나타내는 임의의 configuration file은 작성된 C 함수의 호출로 이루어진 C 프로그램으로 바꾸어 줌으로써(바꾸는 변환기를 준비함으로써) 이루어진다.

즉 각 기능 코드에 대한 서브루틴 함수의 작성 부분과 configuration file에 대해 각 기능코드에 해당하는 서브루틴 함수를 기능 코드 나열 순서에 따라 열거함으로써 이루어지는 함수 부분으로 나뉘어 진다. 전자는 기능 코드의 종류가 추가 될 때마다 개발자가 직접 프로그램하는 것이 합당하고, 후자는 configuration file이 바뀔 때마다(즉 제어 알고리듬이 바뀔 때 마다) 달라지므로 자동적으로 생성되게 하는 것이 바람직하다. 이러한 변환 방식은 새로운 기능코드의 추가를 그 기능 코드에 대한 C 함수의 추가로서 이룰 수 있으므로 용이한 확장성을 가짐을 말해준다.

1. 기능코드에 대한 C 서브루틴 함수

각 기능 코드에 일대일로 대응되는 C 서브루틴 함

수는 대응되는 기능코드가 수행해야 할 일들을 C-언어로 프로그래밍한 함수를 말한다. 이러한 함수의 이름은 기능코드 이름과 동일하며, 함수의 argument는 출력 포트 변수의 pointer, 입력 포트들로부터의 입력 값들 그리고 파라메타 pointer순으로 구성되어 있다. 입력과 출력 argument의 형(type)은 정수(integer)와 소수(float)의 두 형태를 동시에 허용하는 다음과 같은 형태이며

```
typedef union union_var{
        int ival;
        float fval;
} UNION_VAR;
```

파라메타 argument의 형은 각 기능코드에 따라 독립적으로 지정된 구조형(struct type)인데, 파라메타에 대한 변수와 dynamic 특성을 갖는, 즉 메모리를 요구하는 코드에 대하여 계산된 값을 기억하기 위해 사용되는 변수인 내부 변수(internal variable)의 선언으로 구성된다. 그 이름은 기능코드 이름에서 숫자와 대문자는 그대로이고 소문자는 대문자로 바뀌어서 만들어 졌다. 예를 들어 PID 제어 기능코드의 그 구조는 다음과 같다.

```
typedef struct _pidpvsp{
UNION_VAR par[7]; /* parameter variable
*/
UNION_VAR invar[3]; /* internal variable
*/
} PIDPVSP;
```

C 서브루틴 함수의 구조는 다음과 같다.

```
functioncodename(pout, in1,in2, ···, ,ppar)
UNION_VAR *pout; /*pointer of output*
UNION_VAR in1,in2, ···; /*input */
FUNCTIONCODENAME    ppar; /*pointer
of parameter */
{
C-program for function code of
functioncodename
}
```

2. Configuration file에 대한 C 프로그램 변환

기본적인 구조는 그림 2에 나타낸 것과 같은데 우선 에디터(editor)로 만든 configuration file로부터 기능코드 이름, 블럭 번호, 입력 포트 번호, 그리고 파라미터 값 등(token이라 부름)을 찾아내고(어휘적 분석), 이런 token을 받아 configuration file의 각 기능 코드에 대한 문법을 점검함과 동시에 C-언어 프로그램을 생성하게 된다. 어휘적 분석을 통한 token의 추출은 yylex( ) 함수에서 하게 되는데 이 함수는 configuration file에서 어휘 즉 token을 찾아내는 규칙을 나타낸 lex 원시 파일을 lex 컴파일 함으로써 만들어 진다. 문법 점검은 configuration file이 갖는 문법을 BNF형으로 표현한 yacc 원시 파일을 만들므로써 가능하며, C-언어 프로그램은 configuration file에서 모든 기능코드가 인식 되었을 때 즉 문법적 오류가 없다는 것이 확인 되었을 때 grammer rule의 action 부분의 프로그램에 의해 생성된다. 이 때 생성된 프로그램은 여러 파일에 나뉘어져 있으며 각각은 고유의 기능을 수행 한다. 이와 같은 일은 yyparse( )함수에 의해 행해지는데 이는 grammer rule과 action으로 이루어진 yacc 원시 파일을 yacc 컴파일 함으로써 만들어 진다. 사용 되

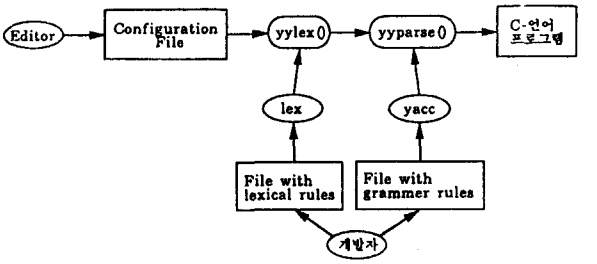


그림 2. C-언어 프로그램 생성에 관한 개념도
Fig. 2. Conceptual diagram for the generation of C-lanuage program.

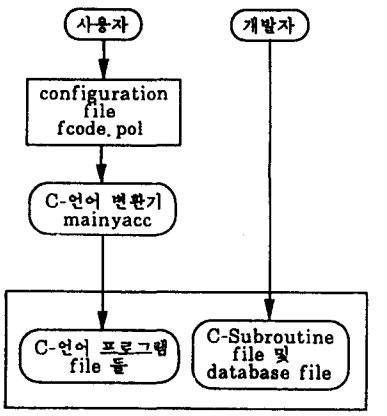


그림 3. C-언어 프로그램 생성의 전체 흐름도
Fig. 3. Overall flow-diagram for the generation of C-language program.

어진 lex와 yacc은 UNIX에서 제공하는 유틸리티(utility)이다. 이와 같은 생성기의 중요한 특징은 configuration file에 존재하는 문법적 오류를 찾아주면서 코드 변환을 이행한다는 것이다.

그림 3은 C-언어 프로그램 생성에 관한 전체적인 흐름도를 나타내었다. 이것은 사용자가 작성한 configuration file에 대한 최종적인 C 프로그램이 C 언어 변환기가 생성한 파일들과 개발자가 프로그램한 C 서브루틴 함수 파일, 변수와 데이타를 정의 및 선언한 데이타 베이스 파일들로 이루어 짐을 나타낸다.

Ⅳ. 응용 예

제안된 공정 제어 언어의 유용성을 알아보기 위해 대상 플랜트로서 서울 화력 발전소의 4호기 보일러의 정상상태(steady state)시 동작을 모사하는 시뮬레이터(simulator)[8]에 대해, 7개의 제어 루프에 대한 제어 알고리듬을 제안된 공정 제어언어를 사용하여 구성하고 적용한다.

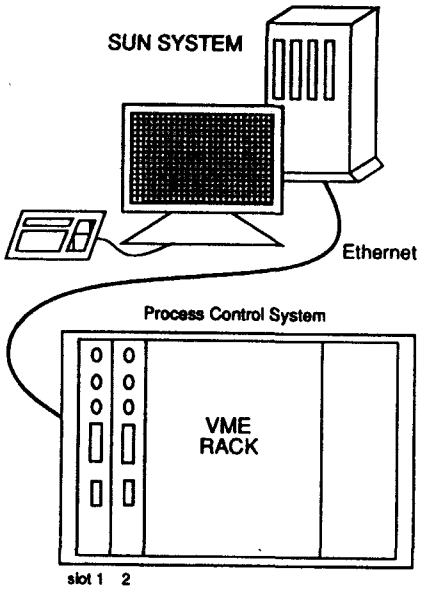


그림 4. 시뮬레이션 시스템의 전체 하드웨어 구조
Fig. 4. Overall hardware structure of simulation system.

시뮬레이션 시스템은 DCS(Distributed Control System)의 기본적이고 필수적인 기능만을 갖는 축소된 형태로 구성되었는데 전체 하드웨어 구조는 그림 4와 같다.

1. 관리제어 서브시스템(Supervisory Control Subsystem:SCS)

UNIX 운영체계를 갖는 SUN-workstation으로 제어 알고리듬을 제어언어에 의해 작성하는 기능 및 컴파일하는 기능을 가지며 실시간 제어시 공정치(process value: PV)와 조작치(manipulated value: MV)를 그래픽 디스플레이하는 기능을 갖는다. 즉 DCS에서 EWS와 OIS 기능을 수행한다.

2. 공정 제어 서브시스템 (Process Control Subsystem: PCS)

VME 시스템에 속해 있는 CPU30 마이크로프로세서 보드로써 실시간 운영체계인 VxWorks를 탑재하고 있다. SCS에서 구성된 제어 알고리듬을 실시간으로 수행하는 역할을 하며 매 샘플링 시각마다 조작치 즉 제어값을 계산하기 위해 PSS로부터 PV를 입력받고, 이를 이용해 제어값을 계산하고, 계산된 제어값을 PSS로 보내는 역할을 한다.

3. 공정 시뮬레이션 서브시스템(Process Simulation Subsystem: PSS)

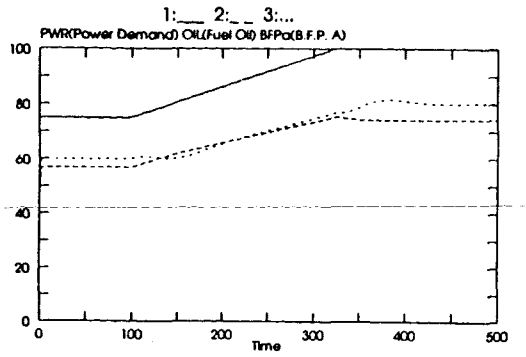
VME 시스템에 속해 있는 한장의 CPU30 마이크로프로세서 보드로써 실시간 운영체계인 VxWorks를 탑재하고 있다. 서울화력 4호기 보일러에 대해 정상상태의 동작을 모사하는 프로그램을 내장하고 있어 매 샘플링 시각마다 PCS로부터 MV를 받아 PV를 계산하여 PCS로 보내는 역할을 한다.

4. 서브시스템들 간의 통신

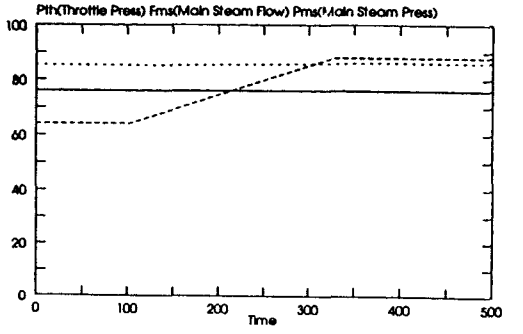
각 서브시스템들 간의 데이타 전송은 다음과 같다. 우선 SCS과 PCS 간에는 Ethenet을 통해 internet domain에서 socket을 사용하여 이루어지고, PCS와 PSS 간에는 VxWorks에서 지원하는 backplane network을 이용하는데 이는 우선 PCS에 공유 메모리를 정의하고 전송해야 할 데이타를 메모리상에 미리 정해진 번지에 저장하고 읽음으로서 이루어진다. SCS와 PSS 사이의 직접적인 통신은 시스템의 구성상 PCS를 통해 이루어지고 공정치가 SCS로 전달되는 상향의 단방향 통신만이 이루어진다.

시뮬레이션은 다음과 같은 절차에 의해 이루어졌다.

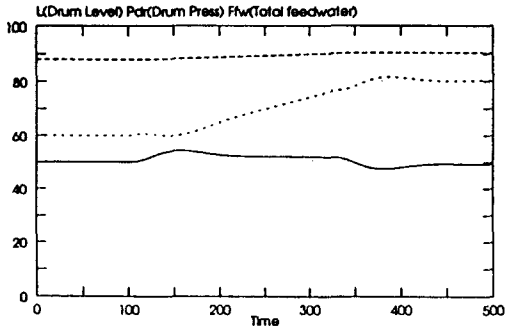
1) 서울화력 4호기 보일러를 제어하기 위한 제어 알고리듬을 구성하였다. 기존의 아날로그 공기식 제어기에 적용했던 제어 알고리듬을 분석하여 동일한



(a)



(b)



(c)

그림 5. 보일러 제어 시스템의 제어 입출력 신호
(a) 부하 변동 곡선 및 제어 입력 신호
(b) 압력 루프의 공정치 신호 (c) 수위 루프의 공정치 신호

Fig. 5. Control input/output signals of the boiler control system. (a) power demand curve and control input signal (b) process value signal of pressure loop (c) process value signal of level loop

기능을 하는 제어 알고리듬을 공정 제어언어에 의해 구성하였다. 전체 7개 루프 중에서 압력 루프와 수위 루프에 대한 제어 알고리듬의 configuration file을 부록 B에 나타내었다. 적용된 모든 루프의 기본 제어기는 모두 PID 제어기이다.

2) 구성된 제어 알고리듬의 configuration file을 C 원시 프로그램으로 변환 한다.

3) 변환된 C 원시 프로그램을 PCS에서 실행될 수 있게 컴파일한다. 여기서 사용된 컴파일러는 GNU의 cross-compiler이다.

4) 서울화력 4호기 보일러를 모사하는 프로그램을 PSS에서 실행될 수 있게 컴파일 한다.

5) PCS와 PSS에 대한 실행 파일을 네트웍을 통해 각각에 다운로드(download)한다.

6) 우선 시뮬레이터인 PSS를 수행시키고 다음으로 제어기인 PCS를 수행시킨다. 그리고 SCS에서 제어 상황을 모니터링 한다.

시뮬레이션 상황은 샘플링 시간이 250[msec]이고 초기에 서울화력 4호기의 최대 출력에 75%의 출력으로 운전하고 있다가 분당 4%의 증가율로 100%의 최대 출력까지 올렸을 경우의 제어 양상을 보고자 했는데 보일러 제어에서 가장 중요시 되는 압력 루프와 수위 루프의 MV와 PV를 그림 5에 나타냈는데 잘 제어되고 있음을 볼 수 있다. 이러한 결과는 실제 공정과 같은 규모의 대상 시스템의 적용을 보인 것이기에 제안된 제어언어의 실질적인 사용 가능성을 입증한 것이다.

V. 결 론

마이크로프로세서 기술의 진보로 인해 그에 따른 S/W 기술의 발전이 가능하게 되었다. 본 논문에서는 공정을 제어하는 알고리듬을 보다 쉽게 작성할 수 있는 공정 제어언어를 제안했고 이러한 언어를 실제로 사용할 수 있는 방법을 또한 제시하였다. 제안된 공정 제어 언어는 그 구조가 간단하여 제어 알고리듬을 프로그램하기 용이한 특징을 갖고 있다.

제안된 공정 제어 언어의 구현방법은 새로운 기능코드의 추가가 용이하여 보다 많은 기능의 확장이 가능하며 C-언어 변환의 중간과정을 거침으로 해서target H/W의 선택에 상관없이 적용할 수 있는 장점을 갖는다. 그리고 제시된 공정 제어 언어를 실제 공정에 준하는 규모의 시스템에 적용하여 그 효용성을 보였다. 제시된 공정 제어 언어의 유연성으로 인해 요즘 많은 좋은 결과를 내는 진보된 제어 알고리듬들도 기능코드화해서 쉽게 사용할 수 있으리라 기대한다.

부록 A: 구현된 기능코드 리스트

일련번호	기능코드이름	출력수	입력수	파라미터수	내부변수수	내용
1	FG	1	1	12	0	Function generator
2	LDLG	1	2	2	2	Lead/Lag
3	HILO	1	1	2	0	High/Low limiter
4	ROOT	1	1	1	0	Square root
5	RATELIM	1	2	2	1	Rate limiter
6	ATRANS	1	3	2	3	Analog transfer
7	HS	1	4	0	0	High selector
8	LS	1	4	0	0	Low selector
9	HLCOM	2	1	2	0	High/Low compare
10	ITRANS	1	3	0	0	Integer transfer
11	SUMIV	1	4	0	0	4-input summer
12	SUMII	1	2	2	0	2-input weighted summer
13	MULT	1	2	1	0	Multiply
14	DIVID	1	2	1	0	Divide
15	PIDERR	1	3	6	3	PID error input
16	PIDPVSP	1	4	7	3	PID · PV and SP
17	PIDVERR	1	3	6	3	PID velocity error input
18	PIDVPVSP	1	4	7	3	PID velocity PV and SP
19	ADAPT	1	1	3	0	Adapt
20	AIN	1	0	3	0	Analog input
21	AOUT	1	1	3	0	Analog output
22	NOT	1	1	0	0	NOT logic
23	MEMORY	1	3	1	1	Memory
24	QOR	1	8	2	0	Qualified OR logic
25	ANDII	1	2	0	0	AND logic 2-input
26	ANDIV	1	4	0	0	AND logic 4-input
27	ORII	1	2	0	0	OR logic 2-input
28	ORIV	1	4	0	0	OR logic 4-input
29	TCSDI	1	0	3	0	TCS digital input
30	TCSDO	1	1	3	0	TCS digital output
31	MSS	1	0	1	0	Manual set switch
32	MSC	1	0	1	0	Manual set constant
33	MSI	1	0	1	0	Manual set integer
34	DTRANS	1	3	0	0	Digital transfer
35	BLINK	1	2	0	0	Blink
36	DSUMIV	1	4	4	0	Digital sum with gain
37	ATREND	1	1	2	99	Analog trend
38	XFCT	1	2	3	0	Analog transfer with input-dependent gain
39	PULRAT	1	1	1	1	Pulse rate
40	TIMER	1	1	2	3	Timer
41	UPDOWN	3	4	3	2	Up/Down counter
42	RAI	1	3	0	0	Redundant analog input
43	RDI	1	3	0	0	Redundant digital input
44	XOR	1	2	0	0	Exclusive OR logic
45	GDC	1	3	14	200	General digital controller
46	SMITHP	1	5	5	200	Smith predictor
47	MOVAVG	1	2	2	300	Moving average
48	POLY	1	1	16	0	Polynomial
49	IPOL	2	2	29	0	Interpolator
50	MADD	9	18	0	0	3 x 3 matrix addition
51	MMUL	9	18	0	0	3 x 3 matrix multiplication
52	TRIG	1	1	3	0	Trigonometric
53	EXPO	1	1	1	0	Exponential
54	POWER	1	2	1	0	Power
55	LOGA	1	1	2	0	Logarithm
56	BCDI	1	0	7	0	BCD input
57	BCDO	1	0	7	0	BCD output
58	MPEST	5	3	4	23	Model parameter estimator
59	ADAPPS	3	6	3	0	Adaptive parameter schedular
60	REGRESS	10	10	14	20	Regression

부록 B: 응용 예의 Configuration File의 일부분

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'    Filename : hanjeon pol
'        Hanjeon DCS Control Configuration File
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

#define MAX_BLK_NUM 225
#define MAX_OUT_NUM 225

! < 1 > Pressure Loop
 1 = msc( ; 76.0)
 2 = ain( ; 0, 0, 1) !-------------- Pth
 3 = msc( ; 0.0)
 4 = mss( ; 1)
 5 = pidpvsp(2, 1, 3, 4; 1.0, 4.0, 0.0033333, 0.0, 50.0, -50.0, 0)
 6 = msc( ; 50.0)
 7 = sumII(5, 6; 1.0, 1.0)
 8 = hilo(7; 100.0, 0.0)
 9 = ain( ; 0, 0, 2) !--------------------Fms
10 = ain( ; 0, 0, 3) !--------------------Pms
11 = msc( ; 0.0)
12 = sumII(10, 11; 0.12, 0.0)
13 = xfct(9, 12; 1.094586, 0.6026, 10.24)
14 = ain( ; 0, 0, 4) !--------------------Fas
15 = msc( ; 0.0)
16 = sumII(14, 15; 0.08, 0.0)
17 = sumII(13, 16; 1.0, 1.0) ! goto other loop
18 = sumII(8, 17; 1.0, 0.8)
19 = msc( ; 60.0)
20 = sumII(18, 19; 1.0, -1.0)
21 = msc( ; 100.0)
22 = msc( ; 200.0)
23 = msc( ; 200.0)
24 = ls(20, 21, 22, 23; )
25 = ain( ; 0, 0, 5) !------------------------Fo
26 = msc( ; 60.0)
27 = sumII(25, 26; 1.0, -1.0)
28 = msc( ; 0.0)
29 = sumII(27, 28; 0.95, 0.0)
30 = sumII(29, 26; 1.0, 1.0)
31 = hilo(30; 100.0, 0.0)
32 = msc( ; -100.0)
33 = msc( ; -100.0)
34 = hs(24, 31, 32, 33; ) ! goto other loop
41 = msc( ; 100.0)
42 = msc( ; 200.0)
43 = msc( ; 200.0)
44 = ls(24, 41, 42, 43; )
45 = msc( ; 0.0)
46 = mss( ; 1)
47 = pidpvsp(25,44,45,46; 1.0,1.0,0.033333,0.0,50.0, -50.0, 0)
48 = msc( ; 50.0)
49 = sumII(47, 48; 1.0, 1.0)
50 = hilo(49; 100.0, 0.0)
51 = aout(50; 0, 0, 1) !---------------------OIL
```

```
! < 2 > Level Loop
56 = ain( ; 0, 2, 4) !------------------------L
57 = nec( ; 22.0)
58 = sumII(56, 57; 1.0, -1.0)
59 = ain( ; 0, 2, 5) !------------------------Pdr
60 = nec( ; 0.0)
61 = sumII(59, 60; 0.12, 0.0)
62 = xfct(58, 61; 1.390625, -0.357143, 10.56)
63 = sumII(62, 57; 1.0, 1.0)
64 = nec( ; 50.0)
65 = sumII(64, 63; 1.0, -1.0)
66 = sumII(17, 65; 1.0, 1.0) ! cometo other loop
67 = ain( ; 0, 2, 6) !------------------------Ffu
68 = nec( ; 0.0)
69 = nec( ; 1)
70 = pidpvsp(67, 68, 68, 69; 1.0, 2.0, 0.016667, 0.0, 50.0, -50.0, 0)
71 = nec( ; 50.0)
72 = sumII(70, 71; 1.0, 1.0)
73 = hilo(72; 100.0, 0.0)
74 = nec( ; 100.0)
75 = sumII(73, 74; 1.0, -1.0)
76 = hilo(75; 100.0, 0.0)
77 = aout(73; 0, 1, 4) !----------------------BFPa
78 = aout(73; 0, 1, 5) !----------------------BFPb
79 = aout(76; 0, 2, 0) !----------------------BFPc
```

參 考 文 獻

[1] Edward Nelson, A Programming Language for Real Time Automative Control Applications," *ACC*, 1980.

[2] A. Lewis, Implimentation of a Standard Language for Real Time-Distributed Process Control," *Proc. 2nd IFAC/IFIP Symposium on Software for Computer Control*, 1979.

[3] Eeldeink, A Block-Diagram Language to Implement Controllers in a Distributed Computer Control Network," *Proc. 2nd IFAC/IFIP Symposium on Software for Computer Control*, 1979.

[4] 정현규, 제어언어를 이용한 다중루프 프로그램형 제어기에 관한 연구, "한국과학기술원 전기 및 전자공학과, 석사논문, 1988

[5] H. Takahashi, An Automatic-Controller Description Language, "*IEEE Trans. of Soft. Engin.*, vol.SE-6, no.1, Jan., 1980.

[6] 박종헌, 다중로보트 시스템을 위한 로보트 언어 개발에 관한 연구", 한국과학기술원 전기 및 전자공학과, 석사논문, 1988

[7] Bailey Controls, Bailey Network 90 - Function Code Reference Manual"

[8] 김재선, 신호 흐름도 모델을 이용한 화력 발전소 드럼형 보일러 시뮬레이터의 개발에 관한 연구", 한국과학기술원 전기 및 전자공학과, 석사논문, 1990

[9] T. Mason and D. Brown, lex & yacc, O'Reilly & Associates Inc., 1990.

著 者 紹 介

徐暢浚(正會員) 第 29卷 第 12號 B編 參照
현재 한국과학기술원 전기 및 전자공학과 박사과정 재학중

金炳國(正會員) 第 29卷 第 8號 B編 參照
현재 한국과학기술원 전기 및 전자공학과 부교수