IEC 61131-3 표준을 따른 PC용 소프트웨어 PLC의 개발

이철수*, 정 구**, 이제필***, 심주현****

Development of a Software PLC for PC Based on IEC 61131-3 Standard

Cheol-Soo Lee*, Gu Jeong**, Je-Phil Lee***, Ju-Hyun Sim****

Abstract

This paper describes a converting algorithm between programmable languages of a software PLC. It is based on IEC 61131-3 standard and PC. The proposed control logic is designed by the software model and common element with data type, variables, POUs(program organization unit) and execution control unit commonly used within programmable languages of IEC 61131-3 Standard. The generation method of object file is proposed on five programmable language based on IEC 61131-3. It is represented as follows; 1) the generation method using conversion algorithm from LD to IL with FBD(function block diagram), 2) the generation method using C code generation algorithm from SFC using the SFC execution sequence with FBD and ST(structured text).

The proposed control logic generator was implemented by Visual C++ 6.0 and MFC on MS-windows NT 4.0.

Key Words : Control Logic Generator(제어 로직 생성기), IEC 1131-3 Standard(IEC 1131-3 표준), LD(래더 다이어그램), IL(명령어 리스트), SFC(sequential function chart)

1. 서 론

PCL(Programmable Logic Controller)는 자동제어 시스템의 핵심을 이루는 기기이다. 기존의 대부분 PCL들은 제작회사마다 자체의 독특한 프로그래밍 언어와 명령어를 사용함으로써 산업 응용에 있어 많은 오류와 인적 자원의 낭비를 가져왔다. 근래 들어 자동제어 시스템에 표준화의 필요성이 부각되자 국제 전기 위원회(International Electro-technical Commission)는 PLC 언어를 모두 통합하여 IEC 61131-3 표준을 규정하였고, 현재 개방화와 표준화의 형태로 발전하는

* 전남대학교 산업공학과(이철수 cscam@chonnam.ac.kr)
** 포스데이타(주)
*** 삼성테크윈(주)
**** 전남대학교 산업공학과 대학원

추세에 있다.

PLC 프로그래밍 언어는 그 동안 많은 연구가 있었다. 기존의 관련된 연구로는 래더 다이어그램(ladder diagram, LD) 해석 방법에 관한 연구가 발표되었다[1]. LD에서 명령어리스트(instruction list, IL)로의 상호 변환 알고리즘에 관한 연구[2]에서의 LD에서 IL로의 변환은 LD의 검색위치를 구하고 그 위치의 마디 정보 테이블을 작성하여 마디를 연결하고 IL명령어로 변환 하는 알고리즘에 관한 연구이다. C언어를 이용하여 기계장치나 프로세서 공정의 순차적인 동작 상태를 플루차트로 표현한 연구[3]가 있으며, 윈도우 기반 PLC구동 프로그램 개발에 관한 연구[4]는 예전의 PLC가 DOS환경을 기반으로 한 프로그램들이 주류를 이루었지만 근래 들어 다른 운영체제의 많은 프로그램이 있으며, 위 연구에 있어서는 기존의 프로그램과 하드웨어와의 호환성을 고려하여 하드웨어적 요소는 그대로 사용하고 Windows 환경의 구동 프로그램을 개발한 내용에 관한 연구이다.

본 논문에서는 PC용 소프트웨어 PLC 개발을 위한 IEC 61131-3 기반의 제어 로직 생성기의 방법론과 프로그램 언어간의 변환 알고리즘을 제안한다. IEC 61131-3 표준의 소프트웨어 모델과 프로그램 언어에서 공통적으로 쓰이는 데이터 타입과 변수, 프로그램 구성단위, 실행 제어 단위로 구성된 공통요소를 기반으로 하여 제어 로직 생성기를 생성한다. 래더 다이어그램은 실제로 PLC에 수행되기 위해서는 IL(instruction list)로 암호화되어 바뀌게 되는데, 본 논문에서는 LD(ladder diagram)에서 IL과 C 코드로 변환하며, 제어의 순서를 설정하는 SFC(sequential function chart)는 순서도와 같은 제어의 흐름을 가지고 있어 C코드로 변환하게 하였다.

2. IEC 61131-3의 개요

기존의 다양하고 분산화 되어 있었던 PLC 프로그래밍 언어의 취약성을 보완하기 위하여 IEC 위원회는 IEC 61131-3 표준을 규정하여 종래의 PLC언어를 통합하고 세계적인 공통의 언어를 제정함으로써 개방화와 표준화를 이루었다. IEC 61131-3 표준은 크게 소프트웨어 모델과 프로그램 언어의 공통요소로 구성되어 있다.[8][9][10][11]

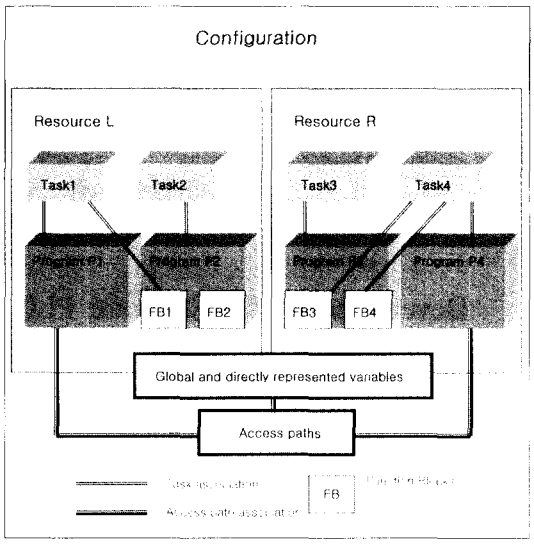

Fig. 1 IEC 61131-3 software model

2.1 IEC 61131-3 소프트웨어 모델

소프트웨어 모델은 범용적인 소프트웨어 개발을 목적으로 규정되었다. PLC 소프트웨어 모델의 바깥쪽 레이어의 구성(Configuration), 가상기계(virtual machine)와의 인터페이스를 담당하는 리소스(Resource), 리소스 내의 실행 단위인 테스크(Task)와 프로그램 구성 단위(program organization units, POUs)로 이루어져 있다. 프로그램 구성 단위(POUs)는 특별한 입력 값에서 새로운 결과값을 만드는 SIN, COS와 같은 수학 함수(Function)와 더 작고 다루기 쉬운 블록으로 프로그램을 작성할 수 있는 타이머와 카운터 같은 함수블록(Function Block), 실제로 수행하는 응용프로그램(Program)으로 구성되어 있다(Fig. 1참조).

2.2 IEC 61131-3 공통 요소

Fig.2의 IEC 61131-3 공통요소는 프로그램 언어에서 공통적으로 사용하는 요소로써 데이터(Data)와 프로그램 구성 단위(POUs), 실행제어 단위(Execution control unit)의 3가지로 분류된다..

공통요소의 데이터는 변수와 데이터 타입으로 구성되어 있다. 데이터 타입은 사용된 변수의 타입을 정의하는데 사용된다. 데이터의 타입을 정의함으로써 사전에 발생할 수 있는 에러를 방지할 수 있고, 일반적인 데이터 타입으로는 Boolean, Integer, Real, Byte, Word, String 뿐만 아니라 날짜(Date)와 시간(Time_of_Day)이 있다. 변수는 서로 다

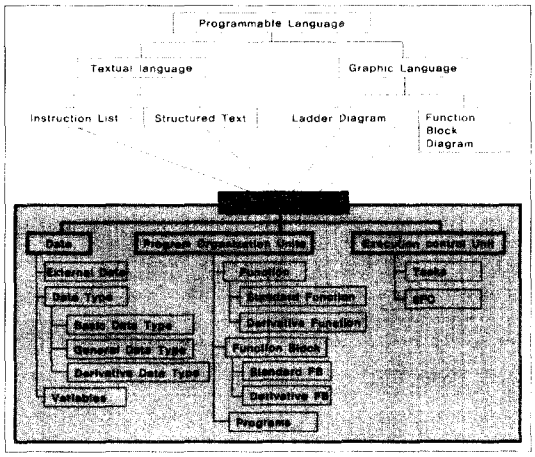

Fig. 2 IEC 61131-3 common element

른 프로그램 구성단위(POUs)에 있는 각각의 정의 부분에서 선언될 수 있으며, 각각의 변수는 POU 안에서 지역(Local) 변수로 쓰여짐으로 충돌 없이 재사용이 가능하다. 변수의 종류는 입력, 출력, 입출력, 내부, 외부변수로 나누어진다.

프로그램 구성 단위(POUs)는 소프트웨어 모델에서와 같이 함수(Function)와 함수블록(Function Block), 프로그램(Program)으로 분류된다.

실행 제어 단위(Execution control unit)는 소프트웨어 모델의 테스크와 제어의 순서를 설정하는 SFC로 분류된다.

3. 제어 로직 생성기의 전반적인 구성

본 논문에서 제안한 제어 로직 생성기는 IEC 61131-3 소프트웨어 모델과 공통요소를 바탕으로 구현되었다.

IEC 61131-3 표준의 소프트웨어 모델 규격에 따라 구성(Configuration)은 하나의 PLC시스템에서 사용될 수 있으며, 하나만 생성할 수 있게 구현하였다. 가상기계의 인터페이스를 담당하는 리소스(Resource)는 구성에서 하나 이상의 리소스를 포함 할 수 있도록 하였다. 하나의 리소스는 다시 여러 개의 프로그램(Program)을 실행할 수 있으며, 테스크(Task)는 프로그램(Program)과 함수(Function), 함수블록(Function Block)의 실행을 담당하도록 구현하였다. 프로그램으로 작성될 수 있는 언어는 IEC 61131-3 표준에서 규정하고 있는 4개의 언어(IL, LD, ST, FBD)와 제어의 순서를 설정하는 SFC언어로 작성할 수 있게 하였다. 제어 로직 생성기의 전체 프로세스는 하나의 프로젝트를 생성하면 구성(Configuration)과 리소스(Resource), 프로그램을 작성할 수 있는 프로그램(Program) 리스트가 생성된다. Fig. 3과 같은 절차에 따라 I/O 입출력 변수를 설정하고 테스크 및 함수를 설정한 후, 프로그램 내에서 사용할 변수를 설정한다. IL, LD, ST, FBD, SFC 언어 중 하나로 프로그램을 작성한다. FBD를 LD언어로 작성한 프로그램과 LD언어로 작성된 프로그램은 IL로의 변환 알고리즘을 통해 컴파일한 후 실행파일을 생성한다. FBD와 ST, SFC언어로 작성된 C

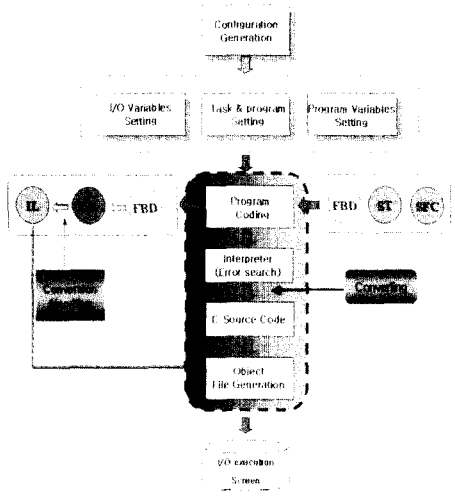

Fig. 3 Process of the control logic generator

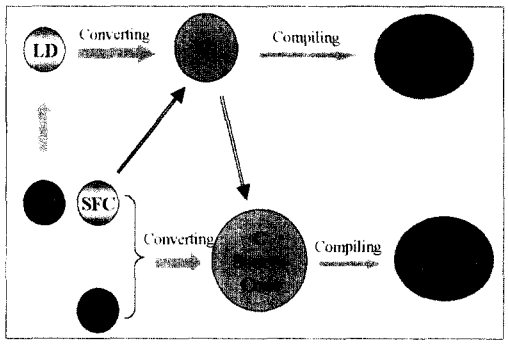

Fig. 4 Logic execution file generation

프로그램은 인터프리터에서 에러 유무를 검색하며 C소스 코드로의 변환 과정을 통해 변환한 후 컴파일 하게 되면 실행 파일을 생성한다. 최종적으로 이 로직 실행 파일을 실행하면 INPUT/OUPUT 실행화면에 입출력 접점을 보여준다.

로직 실행파일 생성의 세부적인 과정은 Fig.4와 같다. FBD언어는 LD언어로 작성될 수 있으며, LD언어로 작성된 프로그램은 LD에서 IL로의 변환 알고리즘을 적용하여 IL로 변환한 후 컴파일하여 오브젝트 데이터파일(object data file)을 생성한다. FBD와 ST, SFC언어는 C 코드로의 변환 과정를 거쳐 소스 코드를 생성하고 컴파일하여 실행 파일을 생성한다. SFC언어로 작성된 프로그램은 C코드 변환 알고리즘을 적용하여 C 소스 코드를 생성하고 컴파일 과정을 거쳐 오브젝트 코드파일(object code file)을 생성한다.

3.1 Ladder Diagram(LD)에서 Instruction List (IL)와 C 코드로의 변환 알고리즘

래더 다이어그램이 실제로 PLC에서 수행되기 위해서는 명령어 리스트(IL)로 암호화되어 바뀌게 되는데, LD에서 Il과 C 코드로의 변환 알고리즘은 Fig. 5와 같다.

각각의 마디를 생성한 후 병렬관계 데이터를 생성하고, 생성된 병렬관계 데이터에서 쌍을 찾는다. 이 찾아진 쌍들로부터 포함관계를 비교하고 이러한 과정이 끝나면 IL 과 C 코드로 전환하기 위한 제어로직을 생성하게 된다.

3.1.1 마디 리스트 생성

마디란 래더 다이어그램의 접점에서 병렬연결이 있는 부분으로 정의한다. 기존의 래더 다이어그램에서 접점의 연결부분을 검색하고 병렬연결이 있는 부분을 마디(Q)로 정의한다. 이렇게 정의되어진 마디들을 마디 리스트라 한다. 마디리스트는 한 개의 Rung 단위로 생성되어 변환되는데, 래더 다이어그램의 시작 행에서 병렬연결이 있는 끝 행 까지를 하나의 Rung으로 정의한다.

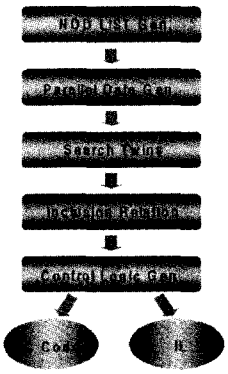

Fig. 5 IL generation process from LD

마디 검색은 Rung의 시작 행에서 우측으로 검색한 마디의 번호가 증가하고 우측의 마디가 더 이상 없는지를 검색한 후 마디가 없으면 다음 행의 마디를 검색한다. 다음 행에서 다시 또 우측으로 갈수록 번호가 증가하면서 모든 마디를 검색하고 번호를 부여한다. 이러한 과정을 반복하면서 마디에 순서대로 번호를 부여하고 모든 마디를 검색하여 마디 리스트를 생성한다.

3.1.2 병렬관계 데이터 생성

병렬관계 데이터란 래더 다이어그램에서 병렬로 연결된 부분에서의 시작 행과 시작 열, 끝행 그리고 병렬관계의 시작과 종료를 검색하여 모든 병렬관계를 데이터로 정리한다.

래더 다이어그램에서 Rung의 처음 시작위치와 끝 위치 즉 출력이 있는 위치를 확인한 후에 Rung의 시작위치에서 수평방향으로 병렬관계(수직선)의 유무를 체크 한다. 만약 수직선이 있으면 수직선의 시작 행과 시작열, 끝행, 병렬관계의 시작(Open)과 종료(Close)를 확인하여 이를 저장한다.

3.1.3 쌍 찾기

관계데이터에서 끝행이 같은 것을 찾아내서 단 두개이면 한 쌍으로 보고 이것을 쌍으로 한다.

여러 개의 끝행이 같은 행을 가진 관계 데이터가 있을 경우에는 열을 비교하여 쌍을 찾는다. Open의 가장 작은 열과 Close의 가장 작은 열이 쌍이 된다. 그 다음으로 Open의 작은 열과 Close의 작은 열이 또 한 쌍이 된다. 결국엔 여러 개의 끝행이 같은 관계 데이터들이 있을 때 이러한 과정을 반복하면서 쌍(twins)을 찾아낸다. 찾아진 쌍들은 Fig.6에

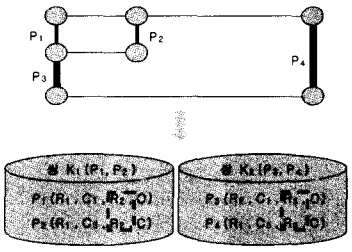

Fig. 6 Searching twins

서 와 같이 K_i로 표현하였다.

3.1.4 포함관계

관계 데이터의 찾아진 쌍들은 끝행이 같은 행을 가지고 있으므로 Fig.7과 같이 시작 행과 끝 행을 연결하여 하나의 사각형을 만들어 준다. 시작 행이 다른 경우의 찾아진 쌍들은 병렬 관계데이터의 시작 행을 비교하여 최 외각의 직사각형을 만든다.

이렇게 만들어진 직사각형이 가지고 있는 열을 비교하여 포함관계를 설정한다. 그림에서 볼 수 있듯이 쌍 $K_1(P_1, P_2)$와 쌍 $K_2(P_3,P_4)$의 두 쌍에서는 쌍 K_2가 가지고 있는 열이 쌍 K_1가 가지고 있는 열을 함께 가지고 있고 더 많은 열을 가지고 있기 때문에 더 큰 포함관계를 갖는다.

쌍 K_1는 열(C_1, C_2, C_3)을 가지고있고 쌍 K_2는 열(C_1, C_2, C_3, C_4, C_5)을 가지고 있으므로 쌍 $K_1(P_1,P_2)$ < 쌍 $K_2(P_3, P_4)$ 이다.

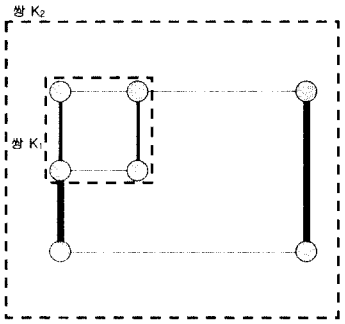

Fig. 7 Inclusion relation

3.1.5 쌍을 이용한 로직 생성

마디 리스트와 병렬관계 리스트 그리고 쌍을 찾은 다음 포함관계를 비교한 래더 다이어그램에서 로직 생성을 하기 위해서 먼저 래더 다이어그램에서 출력접점을 찾는다. Fig. 8과 같이 출력접점에서 래더 다이어그램의 시작점(R_1, C_1)을 찾기 위해 왼쪽 열 방향과 위쪽 행 방향으로 검색해간다. 출력접점에서 시작접점으로 검색된 마디(Q)를 AND로 묶는다(Q1∩Q2). AND로 묶인 마디와 쌍이 존재하는 것을 찾고 이 중 가장 큰 포함관계를 갖는 순서대로 OR로 연결한다.

AND로 묶인 마디와 포함관계가 큰 것의 쌍((Q1,Q2)∪Q4)중에서 같은 마디(Q1,Q2)가 있으면, 하나로 써준다((Q1∩Q2)∪Q4). 다음으로 작은 것의 쌍(Q1∪Q3)을 찾

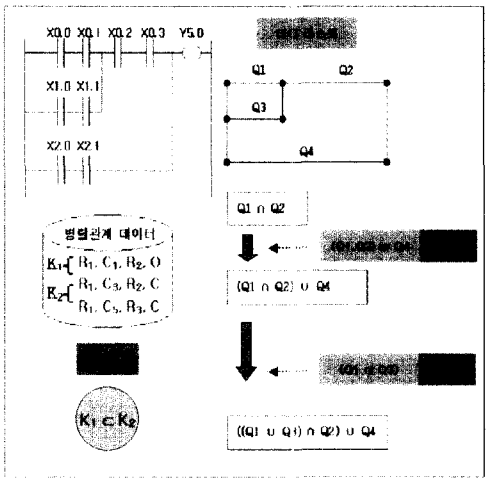

Fig. 8 Control logic generation

는다. 이전에 써준 쌍들과 비교해서 같은 것(Q1,Q3)이 있으면 하나로 다시 써주고 더 이상 쌍이 없는지를 검색한 후 쌍이 없으면 다음 출력 접점을 검색하여 출력 접점이 있으면 이전의 과정을 반복한다. 출력접점이 더 이상 없으면 로직을 찾는 과정은 끝이 난다. Fig.8의 예에서 생성된 최종로직은((Q1∪Q3)∩Q2)∪Q4)이다.

3.1.6 제어 로직에서 IL 과 C 코드로의 변환

생성된 제어로직은 Fig. 9에서와 같이 C 코드와 IL로 변환된다. 로직에서 C 코드로의 변환은 로직 생성시 찾아낸 출력 접점으로 로직을 C 코드로 변환한다. 마디(Q)로 표현된 각각의 접점들은 AND 연산으로 하여 C 언어에서의 AND 연산자(&&)를 사용하여 아래의 그림과 같이 변환된다. 다음으로, 생성된 로직에서 IL로의 변환은 접점의 입력

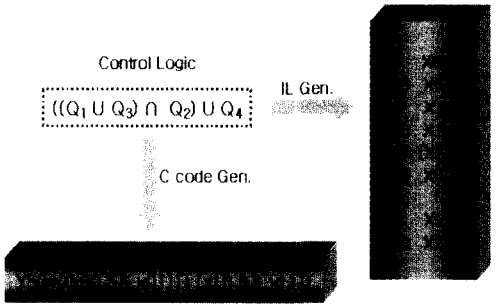

Fig. 9 Conversion from control logic to IL and C

을 나타내는 접점 X0.0은 LOAD(LD)로 교집합 기호(∩)는 AND, 합집합 기호(∪)는 OR 명령어로 변환한다. 마디(Q)사이의 각각 접점들은 AND 명령어로 변환한다.

3.2 SFC의 C코드 생성 알고리즘

제어의 순서를 설정해주는 SFC의 실행 방법은 프로그램이 시작되면 처음의 초기 스텝(시작 스텝)은 항상 활성화되고, 트랜지션의 조건을 만족하면 다음 스텝이 활성화된다. 다음으로 활성화된 스텝의 액션이 활성화 된다(Fig. 10).

SFC에서 C코드 생성 알고리즘은 SFC 실행 방법에 따라 C 코드로 작성해주는 생성 알고리즘이다. 처음의 초기 스텝(시작 스텝)의 상태를 나타내는 Step_status의 변수는 0으로 초기화 되어 있다. Switch문 안에서 초기화 스텝을 나타내는 step_s라는 case문에 이르게 되면 현재의 Step_status가 0이 되므로 초기 스텝인 S1이 활성화 된다. 다음으로 Action의 FILL이라는 함수를 실행한다.

분기의 여부를 If문으로 검색한 후 분기가 없으면 T1 조건의 만족여부를 검사하여 Step_status의 변수 값을 다음 스텝의 값(Step_status = 1)으로 설정해 준다. 만약 분기가 있는 경우라면 분기의 트랜지션의 조건이 만족하는지를 검사하여 분기 다음의 스텝 값을 설정해준다. 스텝의 상태 값이(Step_status = 1) 설정되면 S1이 활성화되고 다음으로 ACTION의 Empty라는 함수를 실행한다. 다시 분기의 여부를 검사하고 분기가 없는 경우에 T2 조건의 만족 여부를 검사하여 Step_status의 변수 값을 다음 스텝의 값(Step_status = 2)으로 설정해 주면 스텝 S2가 활성화 된다.

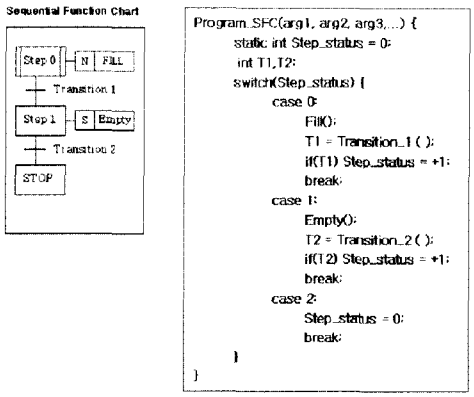

Fig. 10 C code generation from SFC

3.2.1 분기 SFC의 C코드 생성 알고리즘

SFC에서 분기가 있는 경우에 C 코드 생성 방법에 대한 알고리즘은 Fig.11의 B1_T와 B2_T, B3_T가 분기 트랜지션이며, 분기가 있는 경우에는 S1스텝부터 S2스텝 이전까지의 트랜지션(B2_T1)을 하나의 스텝으로 정의한다. 프로그램이 시작되면 초기스텝 S0가 활성화되고 트랜지션(T1)의 조건을 만족하면 S1스텝이 활성화된다. S1스텝이 활성화 되었을 때 트랜지션(B1_T)의 조건이 만족하면 B1_0가 활성화되고 트랜지션(B1_T0)의 조건이 만족하면 다음 단계의 S2스텝이 활성화 된다(S0→S1→B1_0 S2).

두 번째 분기의 경우 S1 스텝이 활성화 되었을 때 트랜지션(B2_T)의 조건이 만족하면 B2_0가 활성화되고 트랜지션(B2_T0)의 조건을 만족하면 B2_1이 활성화 되고 트랜지션(B2_T1)의 조건이 만족하면 다음 단계의 S2스텝이 활성화된다(S0→S1→B2_0→B2_1→S2).

세 번째 분기의 경우에 S1스텝이 활성화 되었을 때 트랜지션(B3_T)의 조건이 만족하면 B3_0가 활성화되고 트랜지션(B3_T0)의 조건이 만족하면 다음 단계의 S2스텝이 활성화된다(S0→S1→B3_0→S2). 트랜지션(T3)의 조건을 만족하게 되면 다음 단계의 스텝인 종료 스텝이 활성화되어 프로그램이 종료된다. 이러한 알고리즘으로 분기 트랜지션이 있는 SFC에서 생성한 C 코드는 아래와 같다(Fig. 12).

3.2.2 동시 시퀀스 SFC의 C코드 생성 알고리즘

동시 시퀀스가 있는 SFC의 경우에 C 코드 생성 알고리즘은 먼저 SFC에서의 동시 시퀀스는 Fig.9와 같이 두 줄의 가로줄로 표시하며, T1분기 밑에서부터 S2스텝 이전 까지를 하나의 스텝(S1)으로 규정한다. S1 스텝의 B1_TO과 B2_T1, B3_T0의 분기가 모두 만족 되어야만 다음 스텝(S2)이 활성화된다.

Fig. 13에서 보면 B1_T와 B2_T, B3_T가 동시 시퀀스의 분기 트랜지션이다. 프로그램이 시작되면 초기스텝 S0가 활성화되고 트랜지션(T1)의 조건을 만족하면 S1 스텝이 활성화된다. S1 스텝이 활성화 되었을 때 트랜지션(B1_T)의 조건이 만족하면 B1_0가 활성화되고 트랜지션(B1_T0)의 조건이 만족하면 트랜지션(B2_T)와 트랜지션(B3_T)가 만족하여 각각의 스텝이 활성화 되어 B2_T1과 B3_T0가 동시에 만족할 때까지 기다렸다가 B1_TO와 B2_T1, B3_T0의 분기가 모두 만족 되면 다음 스텝(S2)이 활성화된다. 생성된 C 코드는 Fig. 13과 같다.

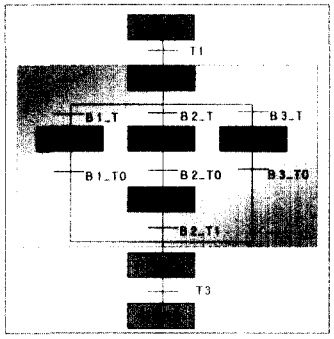

Fig. 11 Divergence SFC

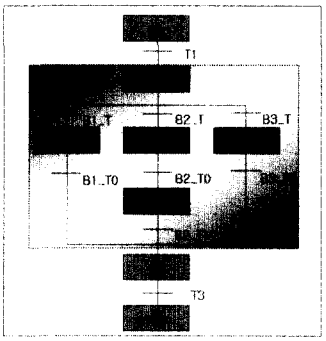

Fig. 13 Simultaneous sequence SFC

```
Case 1://s1
 Static int B1, B2, B3;
   int B1_T0, B2_T, B3_T, B1_T0,
     B2_T0, B2_T1, B3_T0;
     B1 = B2 = B3 = -1;
     B1_T=Transition B1_T();
  If(B1_T) {B1=0;
        Switch(B1){
           Case B1_0;
                Action_B1();
                B1_T0=Transition B1_T0();
                If(B1_T0)B1= + 1;
                Break;
           Default;
                Break;
        }
 }if(B1= =1) Step_status = + 1;
    break;
    B2_T=TransitionB2_T();
  Else if(B2_T) {B2=0;
        Switch(B2){
           Case B2_0;
                Action_B2();
                B2_T0 = TransitionB2_T0();
                If(B2_T0)B2= + 1;
                Break;
```

```
Case B2_1:
        Action_B2();
        B2_T1 = TransitionB2_T1();
        if(B2_T1)B2 = + 1;
        Break;
Default:
        Break;
        }
};if(B1 = = 2) Step_status = + 1;
    break;
    B3_T=TransitionB2_T();
  Else if(B3_T) {B3=0;
        Switch(B3){
           Case B3_0;
                Action_B3();
                B3_T0 = TransitionB3_T0();
                If(B3_T0)B3 = + 1;
                Break;
           Default;
                Break
        }
} if(B3 = = 1)step_status = + 1;
  break;
```

Fig. 12 C code generation from divergence SFC

```
Case 1://S1
static int B1, B2, B3;
  int B1_T, B2_T, B3_T, B1_T0,
    B2_T0, B2_T1, B3_T0;
    B1 = B2 = B3 = -1;
    B1_T=Transition B1_T();
 If(B1_T) {B1=0;
        Switch(B1){
           case B1_0;
                Action_B1();
                B1_T0=Transition B1_T0();
                if(B1_T0)B1= + 1;
                break;
           default;
                break;
        }
}
      B2_T=TransitionB2_T();
if(B2_T) {B2=0;
        Switch(B2){
           Case B2_0;
                Action_B2();
                B2_T0 = TransitionB2_T0();
                if(B2_T0)B2= + 1;
                break;
```

```
           Case B2_1;
                Action_B2();
                B2_T1 = TransitionB2_T1();
                if(B2_T1)B2 = + 1;
                break;
           default;
                break;
           }
}
B3_T=TransitionB2_T();
if(B3_T) {B3=0;
        Switch(B3){
           Case B3_0;
                Action_B3();

                B3_T0 = TransitionB3_T0();
                    if(B3_T0)B3 = + 1;
                break;
           default;
                break
           }
 } if(B1 = = 1 && B3 = = 1)
  Step_Status = + 1;
break;
```

Fig. 14 C code generation from simultaneous sequence

4. 적용 사례

로직 생성을 위한 제어 로직 생성기는 입력과 출력변수를 넣기 위한 I/O창과 테스크, File 창 그리고 프로그램을 작성하기위한 프로그램 창과 변수 창, 에러메시지 창으로 구성하였다.

I/O 창에는 외부기기로부터 입력과 출력을 설정하는 입력접점과 출력 접점 설정 다이얼로그 박스가 있다. 직접 외부기기에서 제어 로직 생성기로 들어오는 입력접점의 접점번호와 변수, 설명을 저장한다. 출력 접점 또한 제어 로직 생성기에서 나가는 신호를 외부기기의 출력 접점으로 보내도록 접점의 번호, 변수, 설명을 저장한다(Fig. 15).

테스크 창에는 수행할 프로그램(함수)에 대하여 우선 순위와 주기적인 테스크, 비주기적인 테스크를 선택하여 설정

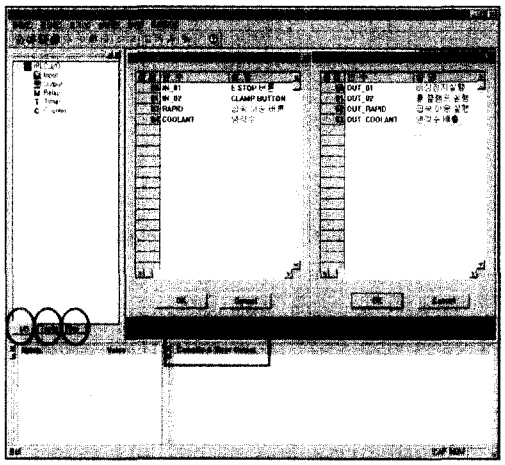

Fig. 15 Input/Output contact dialog box

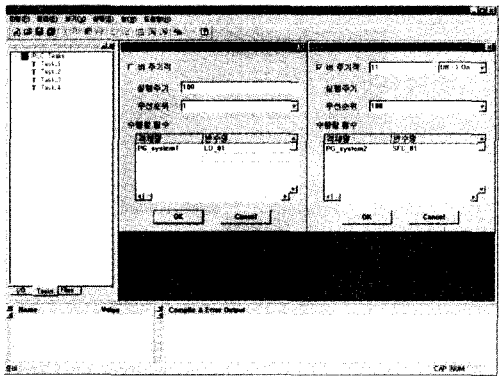

Fig. 16 Task Dialog Box

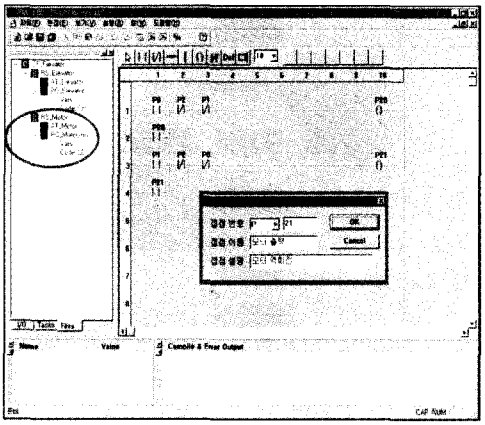

Fig. 17 LD Editor

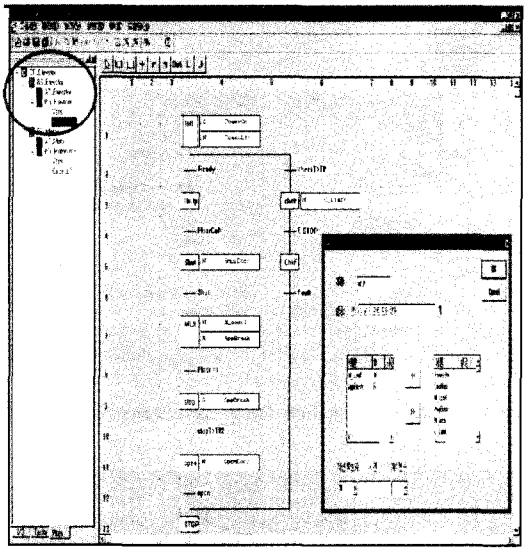

Fig. 18 SFC editor

한다.

주기적인 테스크인 경우에는 설정된 system1 함수의 실행주기와 우선순위에 따라 제어 로직을 반복 실행하며, 비주기적 테스크인 경우에는 우선순위와 입력접점 포트의 접점 변화 값에 따라 0(off)에서 1(on)로 변할 때와 1(on)에서 0(off)으로 변하는 조건 중 하나를 선택하도록 하여 접점 값의 변화에 따라 설정된 system2 함수의 제어 로직을 수행하게 구현 하였다(Fig.16).

Fig. 17은 래더 편집기를 이용하여 래더 프로그램을 작성한 예이다. 입력과 출력의 a접점 b접점을 선택하여 접점을 그릴 수 있게 하였고 접속 선으로는 수직선과 수평선을 사용 하도록 하고있다. 접점의 설정은 접점의 번호에서 출력을 의미하는 P와 접점번호, 이름과 설명을 입력하도록 하였다. 응용명령으로는 카운터와 타이머를 사용할 수 있도록 구현하였다.

SFC에디터(Fig.18)는 스텝의 이름과 설명 이미 작성된 액션 목록에서 액션 블록 목록으로 액션을 추가하고 액션 제한자를 부여할 수 있도록 하였다. 액션 특성자가 N일 때는 액션의 시간을 부여치 않는다. 액션의 실행완료를 나타내는 지시 변수는 0과 1로 두어 설정하게 하였다. 트랜지션은 분기하지 않는 트랜지션과 다음 스텝이 있는 방향에 따라 분기하는 트랜지션을 따로 두어 구현하였다.

5. 결 론

본 논문에서는 IEC 61131-3 표준을 따른 PC용 소프트웨어 PLC를 개발하기 위하여 제어 로직 생성기의 방법론과 프로그램 언어간의 변환 알고리즘을 제안하였다.

IEC 61131-3 표준의 소프트웨어 모델과 프로그램 언어에서 공통적으로 쓰이는 데이터 타입과 변수, 프로그램 구성단위와 실행 제어 단위를 포함하는 공통요소를 기반으로 하여 제어 로직 생성기를 생성하였고, IEC 61131-3 표준에서 규정하고 있는 5개의 언어에 대하여 실행파일을 생성하는 법을 제안하였다.

FBD언어는 LD언어로 작성될 수 있으며, LD언어로 작성된 프로그램은 LD에서 IL로의 변환 알고리즘을 적용하

여 IL로 변환한 후 컴파일하여 오브젝트 데이터파일(object data file)을 생성하였다.

FBD와 ST, SFC언어는 C 코드로의 변환 과정을 거쳐 소스 코드를 생성하고 컴파일하여 실행 파일을 생성하였고, 특히 SFC언어로 작성된 프로그램은 C코드 변환 알고리즘을 적용하여 C 소스 코드를 생성하고 컴파일 과정을 거쳐 오브젝트 코드파일(object code file)을 생성하였다.

제안된 제어 로직 생성기는 MS-windows NT 4.0에서 Visual C++과 MFC를 사용하여 구현하였다.

참 고 문 헌

(1) 김형석, 장래혁, 권욱현, 고속 프로그램형 논리 제어기 구현을 위한 래더 다이어그램 해석 방법, 제어 · 자동화 · 시스템공학 논문집, Vol. 5, No. 1, pp. 33～38, 1999.

(2) 안재봉, 유지훈, 신영민, 송인창, Ladder Diagram 과 Instruction List와의 상호 변환 알고리즘, 한국 자동제어 학술회의 논문집, pp. 629～633, 1990.

(3) 정호섭, 이정익, 김성동, C언어에 의한 플로우 차트형 PLC의 구현, 한국 정밀 공학회 춘계학술대회 논문집, pp. 215～218, 1999.

(4) 김태형, 엄태진, 정원지, 홍대선, 윈도우 기반 PLC구동 프로그램 개발, 한국 정밀 공학회 추계학술대회 논문집, pp. 1277～1280, 1999.

(5) 정 헌, 가상설비를 가진 객체지향 모델 기반의 PLC 프레임워크 구축에 대한 연구, 조선대 박사학위 논문, 1999.

(6) 이철수, 공장 자동화, 터보테크 출판부, pp. 13～57, 1999.

(7) 금국환, 그래픽 조직언어를 이용한 순차 제어용 프로그래밍 시스템 개발, 한국 정밀 공학회지,제13권 제4호, pp. 24～33, 1996.

(8) IEC(International Electro-technical Commission), INTERNATIONAL STANDARD IEC1131-3, IEC, 1993.

(9) IEC, Programming Industrial control systems using IEC 1131-3. IEC, 1995

(10) JohnHunt,SymposiumPaper,http://www.iica.org.au/letters/IEC1131, 1997.

(11) PLCopen Standardization in Industrial Control Programming, http://www.plcopen.org.

(12) 이철수, 정구, 이제필, 심주현, 제어시스템을위한 IEC 1131-3 기반의 제어로직 생성기의 개발, 한국공작기계학회 춘계 학술대회 논문집, pp. 171～176, 2001.