컴파일러 기술을 이용한 원전용 제어 프로그램의 시뮬레이터 설계

이 완 복, 노 창 현
중부대학교 게임학과
E-mail: wblee@joongbu.ac.kr

Design of A PLC Program Simulator for Nuclear Plant Using Compiler Technology

Wan Bok Lee, Chang Hyun Roh
Dept. of Computer Game, Joongbu Univ.

요 약

본 논문에서는 원전 계측제어시스템 구축을 위해 개발된 원전용 PLC 시뮬레이터의 설계 사항에 관해 소개한다. 원전용 계측제어시스템은 원전이라는 특수한 환경과 제약으로 말미암아, 일반적인 시뮬레이터 개발보다 엄격한 요건을 만족해야 한다. 이러한 요건에는 다양한 테스팅을 통하여 제어 프로그램의 안정성을 보장할 수 있어야 하며, 다수의 계측제어 프로그램들을 고속으로 동시에 실행할 수 있어야 한다. 본 논문에서는 이러한 문제점들을 극복하고자 PLC 제어 프로그램의 컴파일러를 제작하고, Compiled-Code 시뮬레이션 기법을 적용하여 고속으로 실행할 수 있는 시뮬레이터 생성 방안을 고안해 내었다.

I. 서 론

원전 계측제어시스템은 원자력 발전소에 산존하는 수많은 장치들을 운전, 제어, 감시하는 신경망과 두뇌에 해당한다. 특히, 원전 환경에서는 사소한 사건이나 사고가 큰 재앙을 초래할 수 있기 때문에, 이들 제어 시스템은 방대한 경우의 발생 가능한 사고에 대한 대응 시나리오를 가지고 발전소를 안전하게 제어하고 있다.

이러한 원전 계측제어시스템은 현재까지는 안

전과 기술 등의 문제점 때문에 대부분 외산 장비와 기술을 도입하여 구축해 왔다. 그러나 국내에서도 원자력 발전소와 관련된 설비들이 늘어나고, 원자력 핵심 기술의 자립 및 발전소의 안정화, 해외수출을 위해 지난 2001년 7월부터 과학기술부의 지원을 받아서 독자적인 프론티어 사업을 착수하게 되었다. 이 사업의 내용 중에는 원전 안전등급 제어기기(PLC)의 개발과 제어 프로그램을 저작할 수 있는 프로그래밍 툴인 pSET의 개발이 포함되어 있다.

현재까지 개발된 pSET은 화면에 Ladder Diagram(LD), Function Block Diagram(FBD) 등을 그래픽컬하게 편집/디버깅할 수 있는 기능을 제공한다. pSET에서 설계된 제어 프로그램은 PLC에 다운로드 되어 원전 제어에 이용될 수 있는데, 사용자의 활용성을 극대화하기 위해 시뮬레이션 기능을 추가적으로 개발하고 있다. 시뮬레이션 기능이 지원 되면, pSET 환경에서 하위 레벨의 PLC와의 연결성 문제 등을 배제하고 제어 프로그램을 작성할 수 있기 때문에 개발 툴 키트로서의 기능성이 더욱 높아지게 되며, 시스템의 안정성 제고와 개발기간의 단축을 가져올 수 있게 된다. 또한 pSET의 그래픽 편집기와 시뮬레이터가 연동되면 제어프로그램의 수행과정을 애니메이션으로 표현해 줄 수 있기 때문에, 작업 효율성이 그만큼 증대될 전망이다.

그러나 원전용 계측제어시스템은 원전이라는 특수한 환경과 제약으로 말미암아, 일반적인 시뮬레이터보다 엄격한 요건을 만족해야 한다. 이러한 요건중 하나는 다양한 테스팅을 통하여 제어 프로그램의 안정성을 보장할 수 있어야 한다. 또 다른 요건으로는 메모리 사용량이 최소화되어 다수의 계측제어 태스크가 동시에 탑재되어 고속으로 수행될 수 있어야 한다는 점이다. 이것은 원전 제어시스템이 기본적으로 다양한 시나리오에 대응하도록 설계되어야 하기 때문에 제어 로직 자체가 복잡하고 긴 경우가 많기 때문이다.

본 논문에서는 이러한 문제점들을 극복하고자 PLC 제어 프로그램의 컴파일러를 제작하고, Compiled-Code 시뮬레이션 기법[1]을 적용하여 고속으로 실행할 수 있는 시뮬레이터 생성 방안을 고안해 내었다.

본 논문은 다음과 같이 구성되었다. 2장에서는 원전용 PLC 시뮬레이터에서 특별히 요구되는 요건들에 대해 설명하고, 3장에서는 이러한 요건들을 충적시킬 수 있는 설계 아이디어에 대해 소개한다. 4장에서는 PLC 제어 프로그램들을 C 코드로 변환하기 위해 구현된 컴파일러와 이것을 이용한 시뮬레이터 구성에 대해 설명하고, 5장에서 결론을 맺는다.

II. 원전 제어 프로그램 시뮬레이터의 요건

2.1 PLC 소프트웨어 개발 환경

원전 안전등급 PLC의 소프트웨어 개발은 주로, pSET이라는 개발 환경을 통해서 이루어진다. pSET은 전기연구원과 포항공대에서 개발한 저작 툴로서 아래 그림 1과 같은 그래픽 환경을 제공한다. 현재 pSET 저작 툴은 IEC 표준 문서[2]에서 지정한 다섯 가지 PLC 언어 중에서 IL, LD, FBD의 세 가지 언어를 이용한 제어 프로그램의 설계를 지원하며, 추후 SFC 언어 또한 지원할 예정이다.

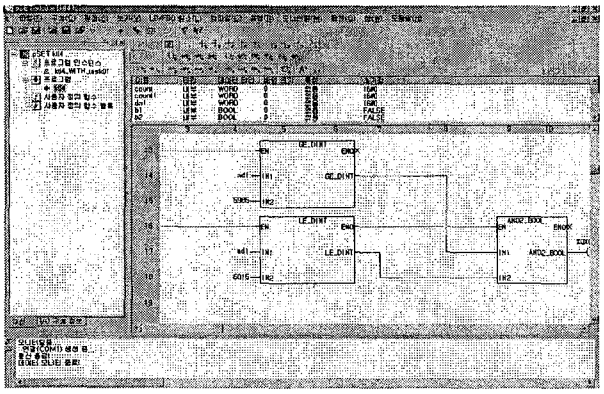

그림 1 pSET: 안전등급 PLC 소프트웨어 개발 환경

원전용 제어 프로그램은 원전이라는 특수 환경과 제약으로 인해 일반적인 공정 제어 프로그램과는 다른 다음과 같은 요건들을 요구한다.

- 제어 프로그램의 안전등급 보장
  원전용 제어 시스템은 대단히 위험할 수 있는 Plant를 제어하는 것이기 때문에, 제어 프로그램 자체가 오동작할 수 있는지 충분히 검증하여야 한다. 현재 C, C++ 등의 고급 언어로 구현된 프로그램에 대해서는 Mc Cabe[3], Cantata[4]를 비롯한 다양한 분석 도구들이 나와 있으나, PLC에 실장되는 LD, FBD 등의 저수준 언어에 대한 분석 도구는 매우 취약한 형편이다. 그러므로 PLC 언어로 설계된 원전 제어 프로그램에 대한 안전등급 보장을 지원할 방안을 마련하여야 한다.

- 대규모 제어 프로그램의 고속 처리 방안
  원전용 제어 프로그램은 많은 수의 장치와 접점, 그리고 다양한 오류 상황에 대한 대처 방안을 고려하여 안전하게 플랜트를 제어하여야 하므로, 제어 프로그램이 긴 경우가 많다. 또한, 하나의 PLC는 다수 개의 태스크를 동시에 실행하여야 하기 때문에, 각각의 제어 프로그램이 고속으로 처리될 수 있는 방안이 있어야 한다.

Ⅲ. 원전 제어 프로그램 시뮬레이터의 설계

시뮬레이션은 컴퓨터를 이용한 모델의 해석과정으로 볼 수 있으며, 성능평가를 위해 자주 사용되어 지고 있다. 일반적인 상용의 PLC 시뮬레이터는, PLC 로직을 Rung 단위로 또는 스캔 시간 단위로 실행하는 기능을 제공한다. 그러나, 원전용 PLC에 탑재될 제어 프로그램은 앞 2장에서 설명한 바와 같이, 원전이라는 특수한 환경에서 필요한 요건들을 충족하여야 한다.

우리는 이러한 문제점을 해결하기 위하여 PLC 제어 프로그램을 C-언어로 먼저 변환하였다. 이렇게 할 경우에는 생성된 C-언어 프로그램을 바탕으로 다양한 상용 분석 도구를 활용할 수 있게 된다. 그러나, 이 과정에서 주어진 제어 로직 프로그램을 올바르게 (Validity) C-언어로 변환하였는가 하는 의문점이 생길 수 있는데, 이러한 오류가 발생하지 않도록 하기 위해 우리는 컴파일러 제작 도구인 Lex, Yacc를 사용하여 변환기 (LD/FBD 컴파일러)를 구현하였다.

그러나, pSET 편집기 환경에서는 PLC 제어 프로그램을 그래픽 환경에서 제작하기 때문에, LD/FBD 컴파일러를 사용하려면, LD/FBD 컴파일러가 해석할 수 있는 표준 양식이 정의되어야 한다. 이러한 필요에 의해,

본 연구에서는 LD/FBD GL이라는 PLC 로직 명세 언어를 BNF 양식으로 정의하였다.
pSET 편집기에서 사용자가 설계한 제어 프로그램은 결과적으로 LD/FBD GL 언어로 저장될 수 있는데, 이것은 다시 LD/FBD 컴파일러에 의해 C-code 명세로 변환된다.

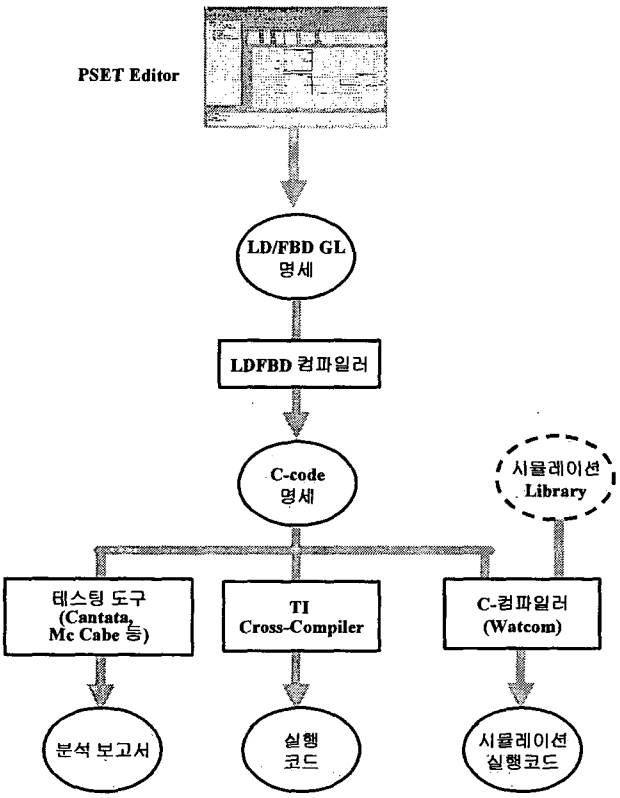

그림 2 PLC 제어 프로그램의 처리 과정

결국, 그림 2와 같은 과정을 거쳐서 시뮬레이션 실행 코드를 생성하게 된다. 그림 2의 과정에서 중간에 생성되는 C-code 명세는 세 가지 용도로 사용되어 질 수 있다. 첫째는 제어 프로그램의 안정성을 평가하기 위하여 상용 테스팅 및 분석 도구인 Mc Cabe[3]나 Cantata[4]를 이용하여 제어 프로그램의 안정성을 평가할 수 있다. 둘째로는 생성된 C-코드와 시뮬레이션 라이브러리 코드를 컴파일/링크하여 시뮬레이션 실행 코드를 생성할 수 있다. 시뮬레이션 라이브러리 코드에는 제어 프로그램을 Rung 단위 또는 Scan Time 단위로 Break Point를 지정하거나 실행할 수 있는 Run Control을 지원하며, 이외에도 특정 IO 포트나 메모리 내용을 조회하고 수정할 수 있는 기능을 제공한다. 세 번째 용도로는 크로스 컴파일 과정을 거쳐서 실제 타겟 PLC상에서 실행될 수 있는 다운로드 코드로 변환되는 것이다. 현재 개발되고 있는 원전용 PLC는 상업용 PLC의 구조와는 달리 빠른 수행을 위해 다운로드 코드가 PLC에 탑재된 마이크로 프로세서의 기계어로 구성될 필요가 있다. 이러한 구조는 다운로드 코드가 PLC 측에서 해석(Interpretation)되는 과정의 Overhead를 해소시키기 때문에 빠른 프로그램 수행이 가능하게 한다.
이러한, 접근 방안을 택하여 우리는 앞 2장에서 열거한 원전용 PLC 시뮬레이터의 요건 사항을 만족시킬 수 있다. 즉, pSET 편집기로 설계한 제어 로직에 대해, 안정성 평가, 성능 평가, 그리고 최종 다운로드 코드로까지 생성할 수 있는 Seamless한 개발 환경을 갖출 수 있게 되었다. 또한, PLC 제어 프로그램의 해석 과정은 LD/FBD 컴파일러에서 의해 실행 단계 전에서 이루어지기 때문에 제어 프로그램의 실행 속도 또한 매우 빠를 수 있게 되었다. 이러한 방식은 기존에 Cad 분야에서 적용되었을 때 수십배 이상의 속도 향상을 가능하게 하는 Compiled Code 시뮬레이션 기법의 일종으로 볼 수 있다[1].

3.1 시뮬레이션 과정

그림 3은 사용자가 시뮬레이션을 수행하는 과정에 대해 단계별로 설명한 것이다. 구체적으로 아래 단계를 따른다.

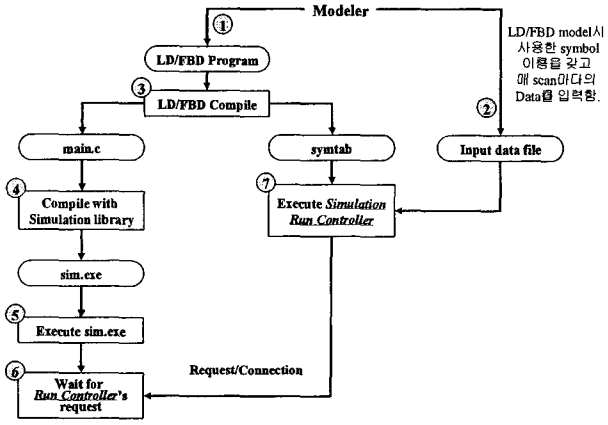

그림 3 시뮬레이션 과정

1. 모델러는 LD/FBD model을 pSET 편집기 환경에서 작성후, LD/FBD GL 형식으로 저장한다.
2. 시뮬레이션 과정에 사용할 입력 파일을 작성한다. 이때, 입력은 스캔 단위로 모든 입력 접점에 대한 값을 표시해야 한다.
3. 다음 장에서 소개하는 LD/FBD compiler를 이용하여 C코드(main.c)를 생성하고, 심볼 테이블 파일(symtab)을 얻는다.
4. 생성된 C 코드를 시뮬레이션 라이브러리와 함께 컴파일하여 시뮬레이션 실행 파일을 구한다.
5. 생성된 실행파일을 수행시켜서, 시뮬레이션이 가능한 상황으로 만든다.
6. 실행된 simulator는 TCP request를 기다린다. 시뮬레이션 제어는 이후에 Run Controller에 의해 진행된다.
7. 생성된 symtab 파일과 제공된 input data file을 사용하여, Run Controller는 wait상태에 있는 시뮬레이터와 TCP connection을 맺는다. 이 후에는 여러 가지 실행 및 디버깅 명령을 적용하여 시뮬레이션을 할 수 있다.

3.2 시뮬레이션 모드

시뮬레이션을 진행 및 제어하는 Run Controller는 옵션에 따라 다음과 같은 두 가지 모드로 동작할 수 있다.

- run-to-end mode : input data file을 읽어서 시뮬레이션을 하고 결과를 output file에 쓰는 모드. 시뮬레이션 중간에 모델러가 디버그 명령으로 일체 개입할 수 없다.
- debug mode : 디버깅을 위한 모드. 이 모드로 실행하면 프롬프트가 뜨게 되고 모델러는 Run Controller가 제공하는 다양한 명령어를 사용하여 시뮬레이션을 진행시킬 수 있다.

Ⅳ. PLC 제어 프로그램 컴파일러

4.1 LD/FBD GL 언어

PLC 제어 프로그램의 C 코드 변환을 위해서 먼저 그래픽에 바탕을 두고 있는 기존의 PLC 프로그래밍 언어를 텍스트 형태로 변환하였다. 우리는 이 형식을 LD/FBD GL(LD/FBD Graphic Language) 언어라고 부르는데, 이 언어는 LD/FBD 컴파일러가 제어 프로그램의 논리적 에러를 발견하고, PLC 제어 프로그램의 행위를 이해하기 위해서 필요하다.

LD/FBD 컴파일러의 입력으로써, pSET 편집기의 그래픽 편집 파일을 이용할 수도 있지만, 이 파일에는 그래픽 표현을 위해서 사용되는 정보가 복잡하게 얽혀 있어서, LD/FBD 컴파일러가 이용하기에는 효과적이지 못하다.

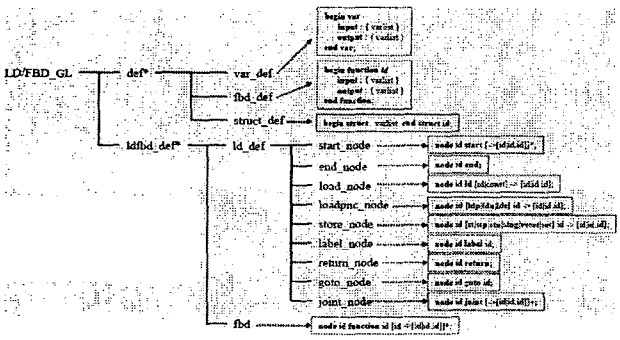

그림 4 LD/FBD GL 언어의 Syntax Tree

그림 4는 LD/FBD GL 언어의 문법 트리 구조를 보여준다. *def*는 제어프로그램에 사용되는 변수와 FBD로 선언된 객체와 사용자 정의 변수를 정의할 때 사용되는 문법이며, *ldfbd_def*는 제어 프로그램을 구성하는 각 원소들 간의 연결관계를 표현하는 문법이다. LD/FBD GL 언어로는 FBD 블록이 함께 사용된 프로그램도 표현할 수 있으나, 지면 관계상 FBD가 사용되는 경우는 배제하여 다음의 예제를 기준으로 설명하겠다.

4.2 예제: Drill Press PLC Control Circuit

개발된 언어를 간략히 소개하기 위하여 IEC 61131 표준 문서에 소개되고 있는 Drill Press PLC Control Circuit[2] 예제를 기준으로 설명하면 다음과 같다. 기본적으로 모든 input은 false로 설정되어 있다. Power input을 1로 올리게 되면 Led가 켜진다. Sw를 넣고, Left button과 Right button을 누르면 shield가 내려온다. M 입력을 이용하여 motor를 돌리게 되고 내리게 된다. 그림 5는 상기 예제에 부합하는 Ladder Diagram이다.

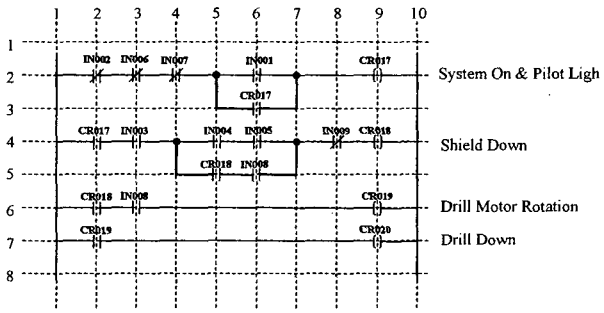

그림 5 Drill Press PLC Control Circuit

이 Ladder Diagram의 LD/FBD GL 언어 표현은 다음 그림 6과 같다.

```
begin var
    input :{ Power : bool; Off1 : bool; Sw: bool ;
      Left:bool; Right:bool; Off2 : bool;
      Off3:bool; M:bool; Off4:bool; }
    output :{ Led : bool; Shield : bool;
      Rotate : bool; Drill:bool; }
    end var;
    node _1_2 start -> _2_2;
    node _2_2 ldc Off1 -> _3_2;
    node _3_2 ldc Off2 -> _4_2;
    node _4_2 ldc Off3 -> _5_2;
    node _5_2 joint -> _6_2, -> _6_3;
    node _6_2 ld Power -> _7_2;
    node _6_3 ld Led -> _7_2;
    node _7_2 joint -> _9_2;
    node _9_2 st Led -> _10_2;
    node _10_2 end;
    node _1_4 start -> _2_4;
    node _2_4 ld Led -> _3_4;
    node _3_4 ld Sw -> _4_4;
    node _4_4 joint -> _5_4, -> _5_5;
    node _5_4 ld Left -> _6_4;
    node _6_4 ld Right -> _7_4;
    node _5_5 ld Shield -> _6_5;
    node _6_5 ld M -> _7_4;
    node _8_4 ldc Off4 -> _9_4;
    node _7_4 joint -> _8_4;
    node _9_4 st Shield -> _10_4;
    node _10_4 end;
    node _1_6 start -> _2_6;
    node _2_6 ld Shield -> _3_6;
    node _3_6 ld M -> _9_6;
    node _9_6 st Rotate -> _10_6;
    node _10_6 end;
    node _1_7 start -> _2_7;
    node _2_7 ld Rotate -> _9_7;
    node _9_7 st Drill -> _10_7;
    node _10_7 end;
```

그림 6. Drill Press 예제의 LD/FBD GL 언어 표현

위 LD/FBD GL 명세를 간단히 살펴보면, 다음과 같다. 먼저, 입력 변수의 이름과 타입이 중괄호 안에 명세되어 있으며, 출력 변수도 비슷한 방식으로 명세 된다. 변수 선언부가 끝나면, Power Rail과 각 IO 접점 및 코일의 연결 정보를 나타내는 node들의 연속이 있다. 처음 노드의 이름은 임의로 그래픽 에디터 상의 위치를 참고하여 "_1_2"라는 이름을 주었는데, 이 위치에는 Power Rail이 놓여 있기 때문에 start 타입의 노드가 명세된다.

"_1_2" 위치의 start 노드는 노드 "_2_2"와 연결관계가 있음을 나타낸다. 노드 "_2_2"는 아래 부분에서 load complement 원소의 이름임을 알 수 있다. 또한 이 노드에 연관된 접점의 이름은 Off1 이며, 이후 노드 "_3_2"와 연결되는 것을 나타내고 있다.

위 LD/FBD GL 코드의 후반부는 이와 유사하게 구성되어 있는데, 이러한 정보는 앞 그림 5에서 보이는 그래픽 로직 프로그램과 완전히 일대일로 대응함을 알 수 있다.

4.3 키워드

LD/FBD GL 언어에서 사용하는 키워드는 다음과 같다.

true, false, struct, var, node, ld , ldp, ldn, ldc, st, stp, stn, stng, reset, set, joint, start, return, goto, function, bool, word, dword, int, dint, real, string, time, date, tod, dt, begin, end, input, output

이중에서 ld는 load를, ldp (ldn)는 positive (negative) edge에서 load 하는 접점을, ldc는 load complement를 나타낸다. 마찬가지로 store에 대해서도 유사한 키워드 들이 지원되며, IEC 61131 표준에서 지원하는 데이터 타입들을 지원하는 키워드들이 있다.

4.4 LD/FBD 컴파일러

위 LD/FBD GL 명세로부터 C코드를 생성하는 컴파일러를 Windows XP환경에서 구현하였다. 이 과정에서 컴파일러의 높은 신뢰성을 위해 일반적인 컴파일러 설계 도구인 Lex과 Yacc을 사용하였다. Lex으로는 GNU flex version 2.5.4를 이용하였으며, Yacc로는 GNU bision 2.0을 사용하였다. 심볼처리는 AVLTree 자료구조를 이용하여 빠른 처리가 가능하도록 하였다.

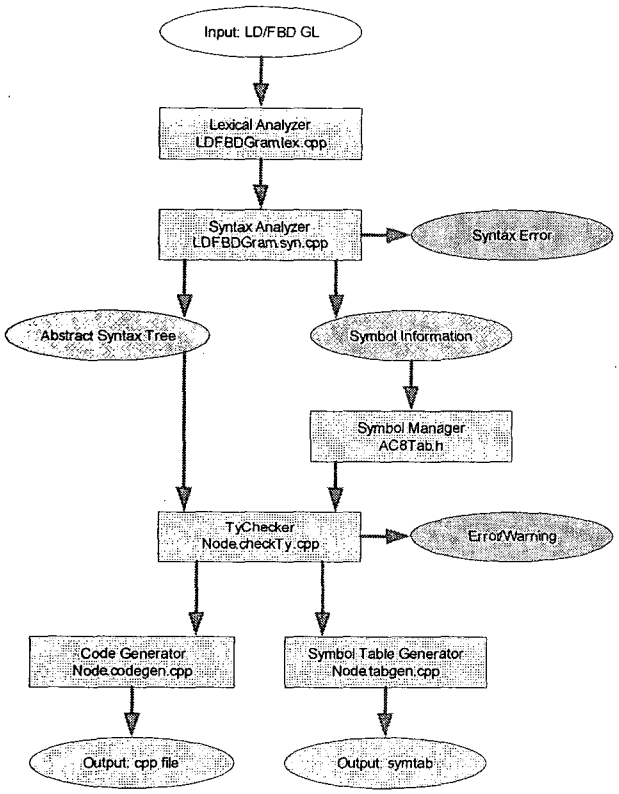

그림 7 LD/FBD GL 컴파일러 내부 구조

그림 7은 구현한 컴파일러의 내부 구조를 보여준다. 입력 파일은 Lex과 Yacc을 거치면서 구문 에러와 문법 에러를 검출하게 된다. 이후에, 내부적으로 문법 트리를 생성하고, 심볼들을 AVLTree 구조로 저장한 후, TyChecker 모듈에서 로직상의 오류를 검출하게 된다. 오류가 전혀 발견되지 않을시에는 코드생성기에서 C코드와 심볼 파일을 생성하여, 시뮬레이션 과정에서 이용될 수 있도록 한다.

앞의 그림 6에 나타난 Drill Press 예제 모델은 제작된 컴파일러를 통하여 오류 없이 C코드로 생성될 수 있었다.

이 C코드는 simulation algorithm을 구현한 라이브러리 파일과 컴파일/링크되어 하나로 합쳐져서 시뮬레이터가 될 수 있다. 이 시뮬레이터에 입력 신호를 인가하고 결과를 확인함으로써 설계하고자 하는 LD/FBD model의 동작을 검증할 수 있게 된다. 또한, 생성된 C코드는 앞 3장에서 말한바와 같이 시뮬레이션 이외의 다른 용도로도 사용되어 질 수 있다.

4.5 오류 검출

LD/FBD GL 컴파일러는 PLC 제어 프로그램에 존재할 수 있는 여러 오류들을 검출해 낸다. 이러한 오류는 크게 다음 세 종류로 크게 분류될 수 있다.

- Lexical Error
  - 구문 분석기(Lex)에 의해 검출되는 오류
  - 예: 잘못된 identifier, 잘못된 상수 등이 해당함.
- Syntax Error
  - 문법 분석기(Yacc)에 의해 검출되는 오류
  - 예: 괄호가 생략되거나, 세미콜론이 빠진 경우 등이 이에 해당함.

- 논리적 오류
  - 노드들간의 연결이 잘 못 되었을 경우
  - LD 로직에서 Cycle이 검출되었을 경우
  - Data type이 일치되지 않을 경우
  - Data 선언이 올바르게 되어 있지 않은 경우

V. 결 론

본 논문에서는 원전용 PLC 제어 프로그램의 시뮬레이터 제작 과정에 대해서 소개하였다. 원전용 제어 시스템은 원전이라는 특수한 환경에서 요구하는 요건들을 충족해야 하기 때문에, 일반전인 PLC 시뮬레이터와는 다른 접근 방법이 필요하다.

이런 문제 때문에 본 논문에서는 PLC 제어 프로그램을 C-언어로 변환하고, 이 후 상용 분석 도구를 통해 설계된 PLC 로직의 안정성을 테스팅할 수 있는 여건을 마련하였다. 또한, 안정성이 검증된 동일한 코드를 기반으로 다운로드 코드를 생성할 수 있게 하였다. 이러한 과정을 거쳐서 고품질의 제어 프로그램을 Seamless하게 설계할 수 있는 방안은 기존 PLC 시뮬레이터들로부터는 제공될 수 없었던 사항이다.

이 과정에서 우리는 그래픽 PLC 제어 프로그램을 텍스트 형식으로 표현할 수 있도록 LD/FBD GL 언어를 설계하고, 전용 컴파일러를 구현하였다. 이 컴파일러는 제어 프로그램의 각종 오류를 검출 할 수 있을 뿐만 아니라, 제어 프로그램의 행위를 완전히 이해한 후 매우 빠른 속도로 실행될 수 있는 C 코드를 생성한다. 생성된 C 코드는 이후 크로스 컴파일 과정을 거쳐 타겟에서 실행될 수 있다. 이러한 방식은 Compiled Code 시뮬레이션 기법에 해당하기 때문에 Interpretation 방식으로 시뮬레이션 되는 일반 시뮬레이션보다 최소 수십 배 이상 빠를 것으로 예측된다. 반면에, 처음 실행 코드를 생성할 때까지 초기 해석 시간이 많이 소요되는 단점이 있다.

현재까지는 IEC 표준에서 요구하는 5가지의 PLC 프로그래밍 언어 중 LD, FBD 만을 지원하고 있다. 향후, SFC 언어 또한 지원하는 것이 필요하며, 시뮬레이션 실행 결과를 더욱 친밀하게 접할 수 있도록 그래픽 환경을 개선할 예정이다.

참 고 문 헌

[1] D. Lewis, “A Hierarchical Compiled Code Event-Driven Logic Simulator”, *IEEE Trans. CADICS*, 10:726-737. 1991.

[2] ---, *IEC 61131-3 International Standard Part 3: Programming languages*, International Electro-technical Commission, 1993.

[3] T.J. Mc Cabe, "A complexity measure", *IEEE Trans. on Software Engineering*, 2(4):308-320, 1976.

[4] ---, *Testing C with Cantata++*, http://www.ipl.com.