Ladder diagram (LD)

TM240

Perfection in Automation

www.br-automation.com

Requirements

Training modules:	TM210 – The Basics of Automation Studio
	TM213 – Automation Runtime
	TM223 – Automation Studio Diagnostics
Software:	None
Hardware:	None

Table of contents

1. INTRODUCTION

Ladder Diagram 프로그래밍 언어는 릴레이 래더 로직(relay ladder logic)이라고 종종 불립니다. Ladder Diagram 은 컨트롤러 시스템의 프로그래밍을 위해 매우 인기가 있는 그래픽적인 언어입니다. 많은 제조업체들은 Ladder Diagram 프로그래밍 언어를 사용하여 시스템을 구현합니다.

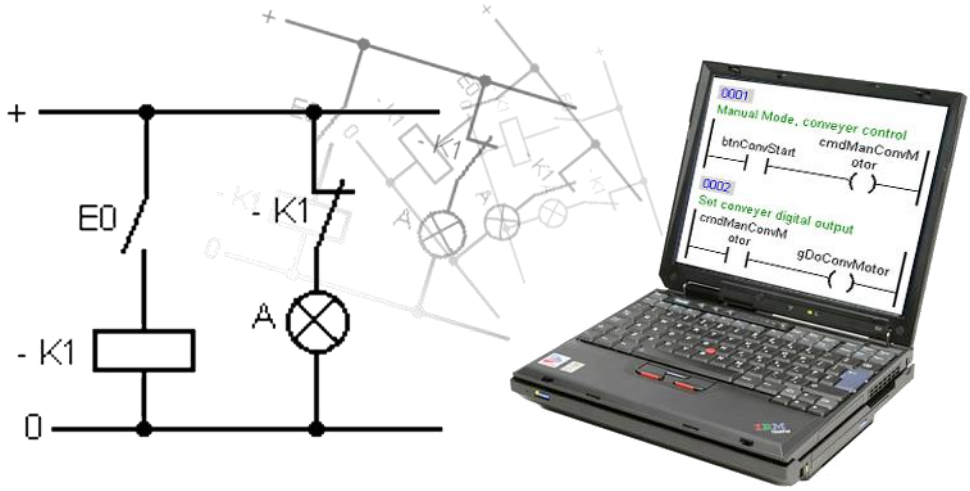


Fig. 1 Introduction

뒤이어 계속되는 장에서, 당신은 Ladder diagram 의 역사, 이점과 개관(overview)를 얻을 것입니다.

예제들은 이용할 수 있는 기능들의 더 좋은 설명을 위하여 주어질 것입니다.

1.1 Objectives

Ladder Diagram 으로 프로그래밍을 할 때 이용할 수 있는 가능성의 개관(overview)를 얻을 것입니다.

논리(logic) 프로그래밍을 위해 Ladder Diagram 과 기호(symbol)의 기본적인 요소를 배우게 될 것입니다.

프로그램 플로우 제어 요소(program flow control element)를 이용하여 유연한 프로그램 개발을 할 수 있습니다.

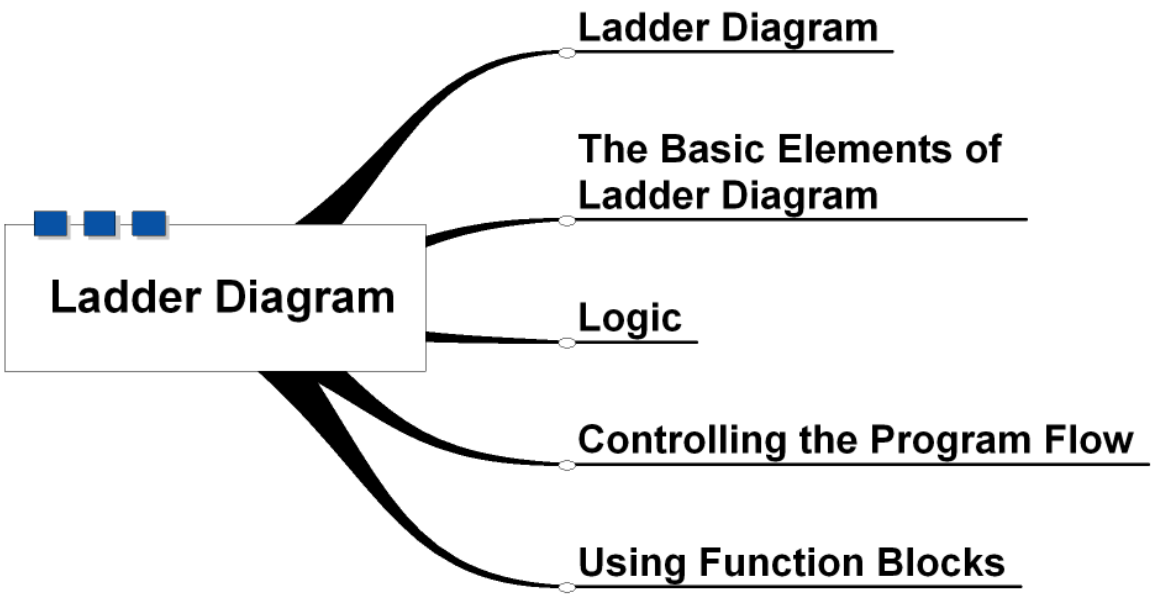


Fig. 2 Overview

2. LADDER DIAGRAM

2.1 The history of ladder diagrams

PLC(Programmable Logic Controller)의 최초의 개념은 약 1968 년에 미국(USA)에서 개발되었습니다. PLC 개념은 배선에 의한 시스템을 대신하여 프로그램을 할 수 있도록 마이크로프로세서에 기반을 두고 개발되었습니다.

PLC 는 Ladder diagram 에 전반적인 기초를 형성하게 되었습니다. 그것은 릴레이 회로(relay circuitry)에 기반을 둔 논리적인 제어 시스템의 개요에 관한 표현입니다. 그 때, 개념은 비교적 짧은 훈련만으로도 빠른 셋업과 단순한 논리적인 제어 시스템 프로그래밍의 매우 빠른 방법이 되었습니다. 단순한 논리 제어 프로그램을 위해, Ladder diagram 은 이상적이고, 사용하기 쉽고, 마스터하기 쉽습니다. 아마도 이와 같은 이유가 산업에서 PLCs 가 경이적으로 성공한 주된 이유일 것입니다.

많은 제조업자들은 그들의 프로그래밍 시스템 기초를 Ladder diagram 에 둡니다. 불행하게도, 어떠한 공개된 표준의 부족은 각 판매인의 시스템이 약간 다른 것을 의미했습니다. 많은 제조업자들은 기능성(functionality)을 늘리기 위해 자주 “특별한 작업(special operation)”을 더했습니다.

1990 년대에 들어서는, 문자 그대로 수천 명의 PLC 제조업자들 각자 그들만의 프로그래밍 시스템과 명령(instruction) 세트가 있었습니다. 비록 다른 시스템을 위해 써졌음에도 불구하고 유사(프로그램이 구조화 되었던 방법)했을 뿐만 아니라 명령 세트는 공급원에서 다른 사람들에게까지 사용되었고, 다양화 되었습니다.

하드웨어 특유의 특색은 항상 생기는 것처럼 보였습니다. 이 상황은 사용자를 특별한 제조업자에게 구속하는 경향이 있었습니다.

1979 년에, 작업 그룹은 PLCs 를 위한 공통의 기준을 만들기 위해 International Electrotechnical Commission (IEC)에 의해 셋업 되었습니다. 이 작업 그룹은 5 개의 분리된 파트로 이루어져 있었고 새로운 표준(IEC 61131 으로 알려지기 위해)을 개발하기로 결정했습니다.

파트 III(“PLCs 를 위한 프로그래밍 언어(*Programming Languages for PLCs*)”)는 1993 년에 출판되었고, PLC 소프트웨어를 위한 규정을 포함했습니다. 파트 III 는 PLC 구성, 프로그래밍, 데이터 저장(storage)를 포함합니다.

IEC61131-3 는 전통적인 PLC 프로그래밍을 겨냥한 비평에 대해 연설하고 있습니다. 그것은 제조사 특유의 트레이닝이 요구되지 않는 일반적인 PLC 프로그램 개발을 위한 프레임워크를 제공합니다. 대부분의 PLC 와 산업용 제어 시스템 제조사들은 현재 이 표준을 채택했습니다.

2.2 General information

Ladder Diagram(LD)는 그래픽을 바탕으로 한 프로그래밍 방법입니다. 그것은 전자 회로의 상징적인 표현입니다. 기호(symbol)는 실제로 전기 장치를 위해 사용된 도식적인 기호와 유사하도록 선택되었습니다. 이러한 이유로, 한번도 PLC 를 보지 않았던 전기 기술자도 Ladder Diagram 프로그래밍 언어를 이해할 수 있었습니다. LD 의 도식적인 기호(contact 와 coil)와 커넥션 라인들은 필요한 논리(logic)를 생성하는데 사용될 수 있습니다.

B&R 은 Ladder Diagram 프로그래밍을 위해서 61131-3 표준을 채택하였습니다.

2.3 Properties

Ladder Diagram 은 다음의 특징들을 갖고 있습니다:

- 그래픽 프로그래밍 언어(Graphical programming languages)
- Wiring diagram 과 유사함(Similar to wiring diagram)
- 간단, 명료한 프로그래밍(Simple and clear programming)
- 직관적인 사용(Intuitive to use)
- 에러 검출에 용이함(Easy to find errors)
- IEC 61131-3 표준 컴파일(Complies with the IEC 61131-3 standard)

2.4 Possibilities

Automation Studio 와 함께 제공되는 Ladder Diagram 프로그래밍 언어는 다음의 가능성을 제공합니다:

- 디지털 입/출력과 내부 불린(Boolean) 변수를 사용.
- 아날로그 입/출력 사용
- 펑션 블록(Function block) 사용
- 프로그램 플로우 제어(flow control) (jumps, cancelling)
- 진단 도구(Diagnostics tools)

3. THE BASIC ELEMENTS OF LADDER DIAGRAM

연속적으로 전원을 제공하는 “버스 바(bus bar)”라고 불리는 수직의 공급 선이 왼쪽에 있다고 상상하십시오. 오른쪽으로 가지를 치고 나오는 선은 “지시선(instruction line)” 으로 알려져 있습니다.

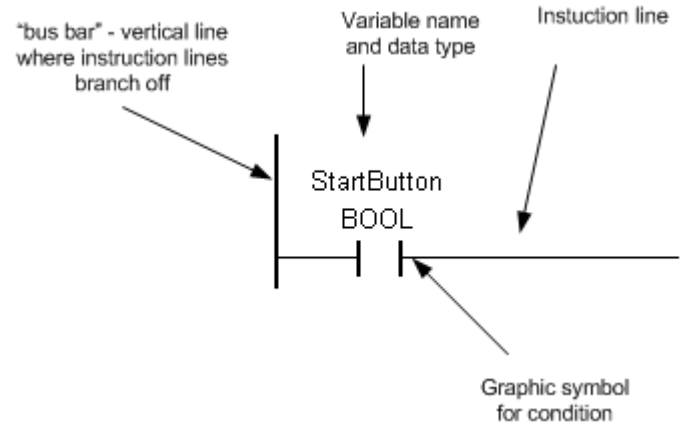


Fig. 3 Basic elements of Ladder Diagram

Ladder Diagram 은 그 자체가 2 개의 기초적인 부품으로 이루어져 있습니다. 우측이 **지시(instruction)**를 포함하는 동안 좌측이 **상태 논리(condition logic)**를 유지합니다. 이러한 조건의 논리적인 조합은 우측의 지시가 언제, 어떻게 실행될 지를 결정합니다. 먼 우측의 요소는 코일(예를 들면 램프, 모터, 릴레이 등)이라고 명명됩니다.

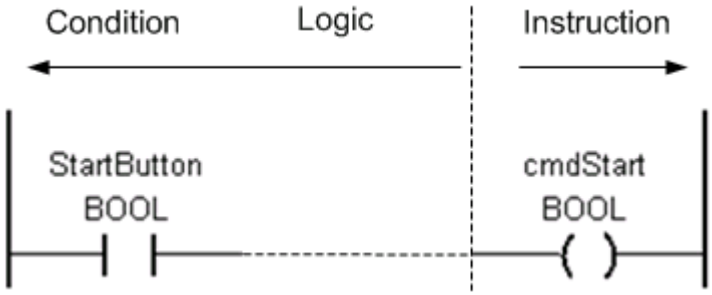


Fig. 4 Logical conditions and instructions

3.1 Networks

네트워크는 특정의 기능을 표현하는 회로입니다. 그것은 **요소(element), 가지(branch),** 그리고 **블록(block)**으로 이루어져 있습니다. 네트워크는 완전한 기능을 반영하며 Ladder Diagram 의 기초적인 단위입니다.

완전한 Ladder Diagram 프로그램은 몇 개의 네트워크로 이루어져 있습니다.

네트워크의 시작은 왼쪽의 **수직 선(bus bar)**입니다. 만일 2 개 또는 그 이상의 회로가 수직선에 접속된다면, 그들은 같은 네트워크에 속하고 있는 것입니다.

최고 50 개의 선(line)과 50 개의 칼럼(column)이 네트워크에 존재할 수도 있습니다. 완전한 Ladder Diagram 의 사이즈는 오직 PC 와 컨트롤러의 메모리 용량에 따라 제한됩니다.

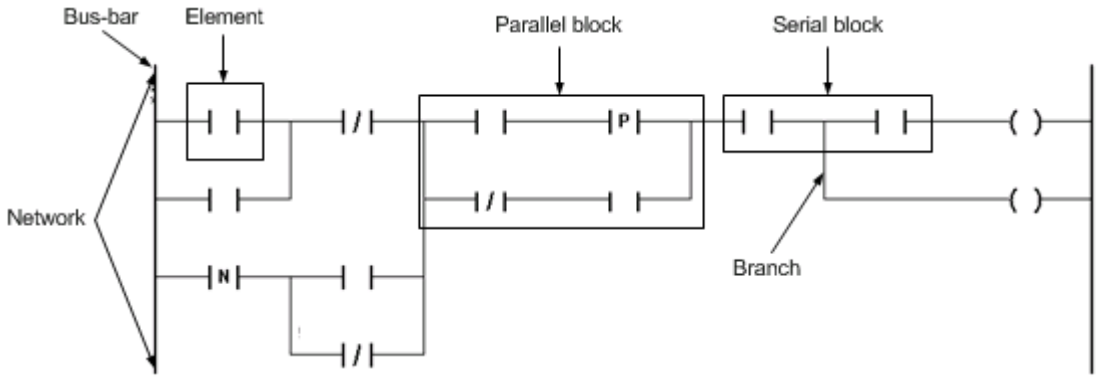


Fig. 5 Ladder diagram network

4. LADDER DIAGRAM SYMBOLS

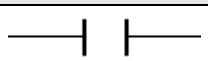
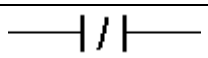
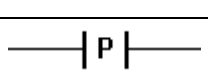
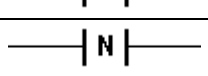
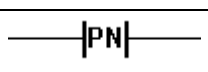
4.1 Contacts

Contact 는 많은 Ladder Diagram 기호들 중의 하나입니다. 그들은 첫째 칼럼(column)과 다른 기호들과의 링크되는 형태 일부에 놓일 수 있습니다. 그들은 우측 위에는 놓일 수 없습니다; 이 자리는 코일(coil)을 위해 예약되어 있습니다. Contact 는 펑션 블록(function block)의 디지털 입/출력에 링크 될 수 있습니다.

네트워크 내에서 contact 를 링크하는 것은 하나 이상의 코일에 할당할 수 있습니다. 모든 contact 는 변수 선언 창(variable declaration window)에서 정의되었거나, 정의 될 변수 이름에 의해 참조됩니다. 그러한 특별한 조건이 평가될 필요가 있을 때마다 각 contact(입력, 출력 또는 내부 변수에 관계없이)는 프로그램에서 내내 사용될 수 있습니다.

Contact 간의 커넥션은 바라는 제어 논리(control logic)에 의존합니다. 이들은 직렬(series), 병렬(parallel), 또는 주어진 출력(coil)을 제어하기 위해 요구되는 직렬(series), 병렬(parallel) 구성에는 무엇이나 놓일 수 있습니다.

오직 BOOL 타입의 변수만 contact 에 할당 할 수 있습니다.

Type of contact	Symbol
Normally open	
Normally closed contact	
Positive edge	
Negative edge	
Both edges	

4.1.1 What are normally open and normally closed contacts?

산업 환경에서, **normally open** 과 **normally closed** 용어를 듣는 것은 흔한 일입니다. 두 용어는 접촉(contact), 입력(input), 출력(output) 등의 단어에 적용합니다. (입력(input), 출력(output), 접촉(contact) 등에 대해 말하는 모든 조합은 같은 의미를 갖고 있습니다.)

Normally close contact 는 누르게 될 때까지 전기를 전도합니다. **Normally open contact** 는 아래로 누를 때까지 전기를 전도하지 않을 것입니다.

만일 **normally closed contact** 이 선택되면, 누군가 종 스위치를 밀 때까지 종은 연속적으로 소리가 날 것입니다. 스위치를 누르면 contact 는 열리고 전기는 끊깁니다. 만일 **normally open contact** 를 사용하면 동작은 반대로 하게 됩니다.

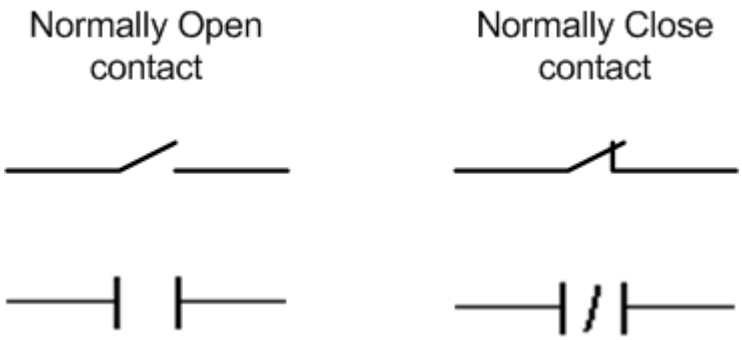
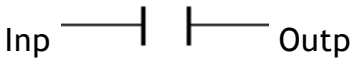


Fig. 6 Normally open and normally closed contacts

Normally closed contact 가 사용되는 한 가지 예로는 기계에 붙어 있는 안전 문(safety door)을 들 수 있습니다. 만일 이 문이 열리면, 접촉은 중단되고 공급 회로도 중단 됩니다. 이것은 사고를 예방하게 할 수 있습니다.

Normally open 과 Normally closed contact 는 출력과 감지 장치에도 적용할 수 있습니다.

4.1.2 Normally open contact



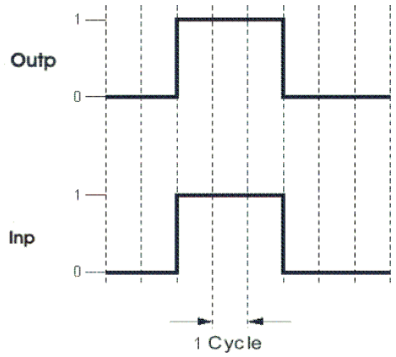
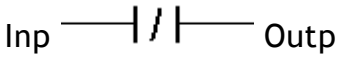


Fig. 7 Relation between an input and output

만일 contact 가 눌리지 않으면, 전기는 전도되지 않고 논리적인 상태는 **FALSE(0)**가 됨.

누르게 될 때, 물리적인 상태는 **"ON"**으로 바뀌고 명령은 **TRUE(1)**가 됨.

4.1.3 Normally closed contact



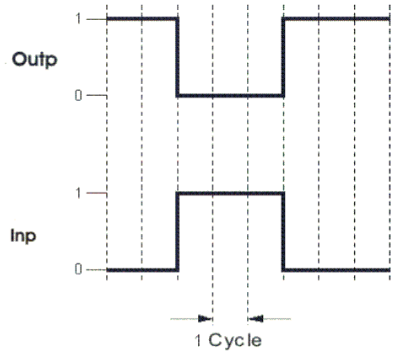


Fig. 8 Relation between an input and output

이 기호는 변수**(BOOL)**의 상태를 역으로 바꿈.

입력 신호가 설정되는 출력을 위해 더 이상 필요가 없을 때 사용됨.

만일 입력이 **TRUE** 로 설정되면 출력의 상태는 **FALSE** 로 설정됨.

4.1.4 Positive edge

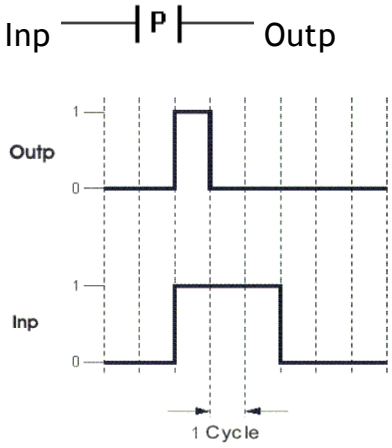


Fig. 9 Relation between an input and output

이 기호는 디지털 신호의 포지티브 에지(positive edge)를 형성하기 위해 사용됨.

변수의 값이 **FALSE** 에서 **TRUE** 로 바뀔 때, 즉 포지티브 에지(positive edge)가 발생할 때, 이 contact 는 cycle 에 **TRUE** 를 리턴함. 이것은 셋(set)/리셋(reset)을 하거나 포지티브 에지(positive edge)의 카운트를 위해 사용될 수 있음.

4.1.5 Negative edge

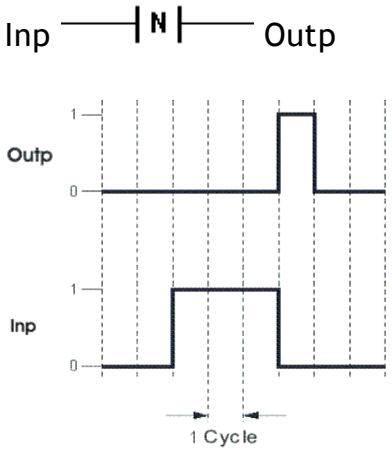


Fig. 10 Relation between an input and output

이 기호는 네거티브 에지(negative edge)를 형성하기 위해 사용됨.

만일 변수의 값이 TRUE 에서 FALSE 로 바뀌면, 상태는 cycle 에 TRUE 를 리턴함. 이것은 출력을 셋(set) 또는 리셋(reset) 하거나 네거티브 에지(negative edge)의 카운트를 위해 사용될 수 있음.

4.1.6 Positive and negative edge

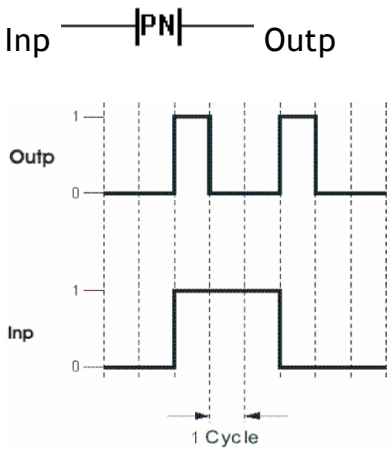


Fig. 11 Relation between an input and output

이 기호는 포지티브(positive)와 네거티브(negative) 에지(edge)의 형성을 위해 사용됨.

이 동작은 포지티브와 네거티브 에지가 전환하는 평행선에 대응함.

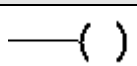
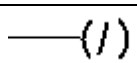
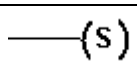
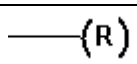
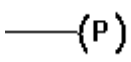
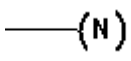
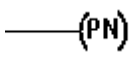
4.2 Coil

Coil 은 기본적인 Ladder Diagram 요소 중의 하나입니다. 그것은 항상 출력으로서 Ladder Diagram 의 오른쪽 위에 놓입니다. Coil 은 contact 나 펑션 블록(function block)의 오른쪽에 접속할 수 있습니다. 적어도 1 개의 Coil 은 Ladder Diagram 에 나타날 것입니다. 몇 개의 병렬 coil 또한 사용될 수 있습니다.

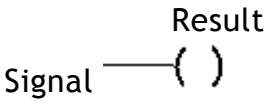
각 coil 은 디지털 출력이나 프로그램 내에서 추가하는 네트워크에서 입력으로서 나중에 공급하게 될 내부 변수를 위해 사용될 수 있습니다.

프로그램이 동작하는 동안 Contact 는 항상 조회됩니다; 만일 논리 연결(logic continuity)이 발견되면, contact 는 TRUE 로 설정됩니다.

오직 Boolean 변수들만 coil 에 할당할 수 있습니다.

Type of contact	Symbol
Coil	
Negated coil	
Set coil	
Reset coil	
Positive transition coil	
Negative transition coil	
Both edges	

4.2.1 Coil



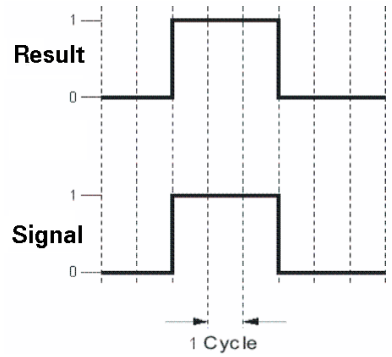
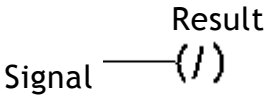


Fig. 12 Relation between an input and output

만일 연결(continuity)이 발견되면 Coil 은 스위치를 켜게 됨(switched on).

4.2.2 Negated coil



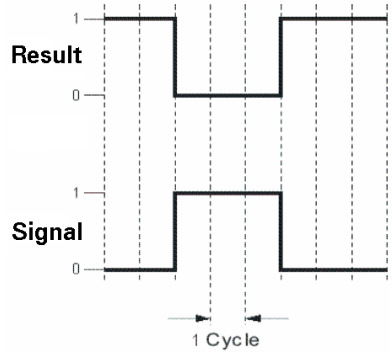


Fig. 13 Relation between an input and output

만일 연결(continuity)이 발견되면, coil 은 스위치를 끄게 됨(switched off); 그렇지 않으면 스위치를 켜게 됨(switched on).

4.2.3 Set

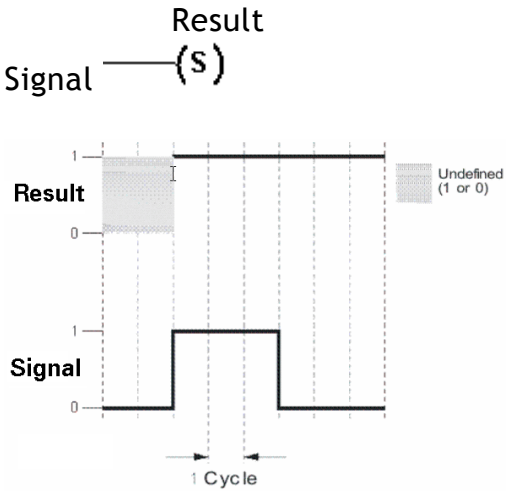


Fig. 14 Relation between an input and output

만일 논리적인 연결(continuity)이 발견되면, 이 coil 은 변수를 **TRUE** 로 설정함.

이 상태는 변수가 리셋(reset)될 때까지 유지됨. 이러한 이유 때문에, 이 coil 은 또한 조건적인 설정으로서 참조됨.

4.2.4 Reset

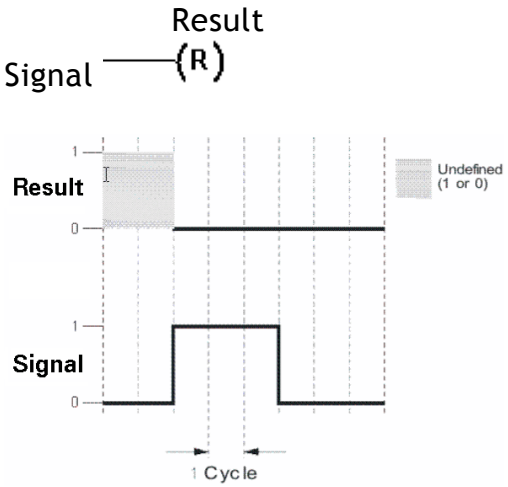


Fig. 15 Relation between an input and output

만일 연결(continuity)이 발견되면 이 coil 은 변수를 **FALSE** 로 설정함.

4.2.5 Positive transition coil

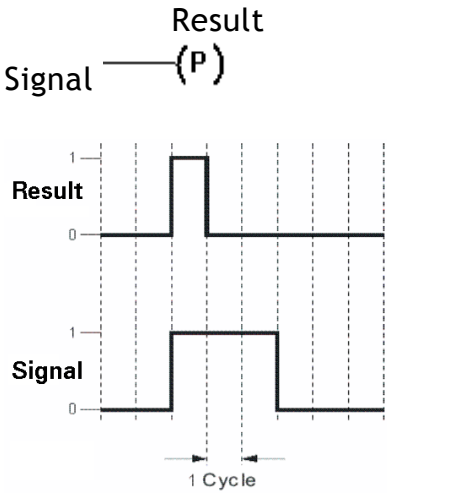


Fig. 16 Relation between an input and output

만일 연결(continuity)이 발견되면 이 coil 은 변수를 **TRUE** 로 설정함.

연결(continuity)이 발견될 다른 모든 cycle 을 위해, 출력은 다시 **FALSE** 로 돌아감.

4.2.6 Negative transition coil

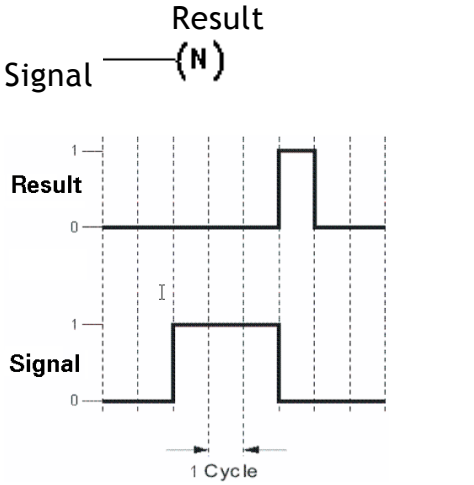
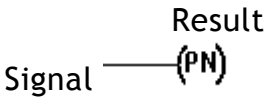


Fig. 17 Relation between an input and output

만일 어떤 연결(continuity)도 발견되지 않으면, 이 coil 은 1 cycle 동안 변수를 **TRUE** 로 설정함.

어떤 연결(continuity)도 발견되지 않는 모든 다른 cycle 을 위해, 변수의 값은 **FALSE** 로 돌아감.

4.2.7 Positive and negative transition coil



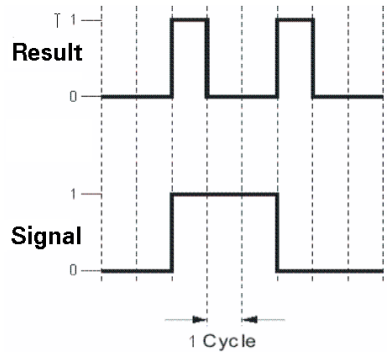


Fig. 18 Relation between an input and output

이 coil 은 포지티브(positive)와 네거티브(negative) 에지(edge) 출력의 기능을 결합시킴.

Task: Part 1: Conveyor belt

이 교육 자료에서, 우리는 컨베이어 벨트를 제어하기 위해 4 단계의 어플리케이션을 만들 것입니다.

디지털 출력"**gDoConvMotor**"와 함께 "**btnConvStart**" 버튼을 누르는 것에 의해 컨베이어 모터를 제어하는 프로그램을 생성하십시오. 입력을 출력과 직접 접속하지 마십시오. 아래 이미지에서 나타낸 것과 같이 중간의 변수로서 명령 변수(command variable) "**cmdManConvMotor**"을 사용하십시오.

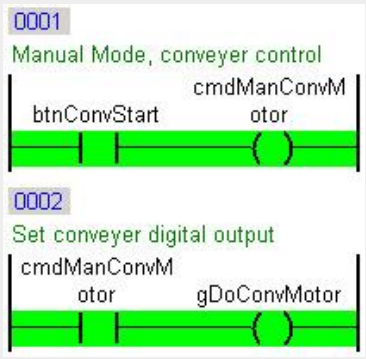


Fig. 19 Task 1 part1, operators

프로그램의 모든 새로운 부품은 초록색(green)으로 표시되었습니다.

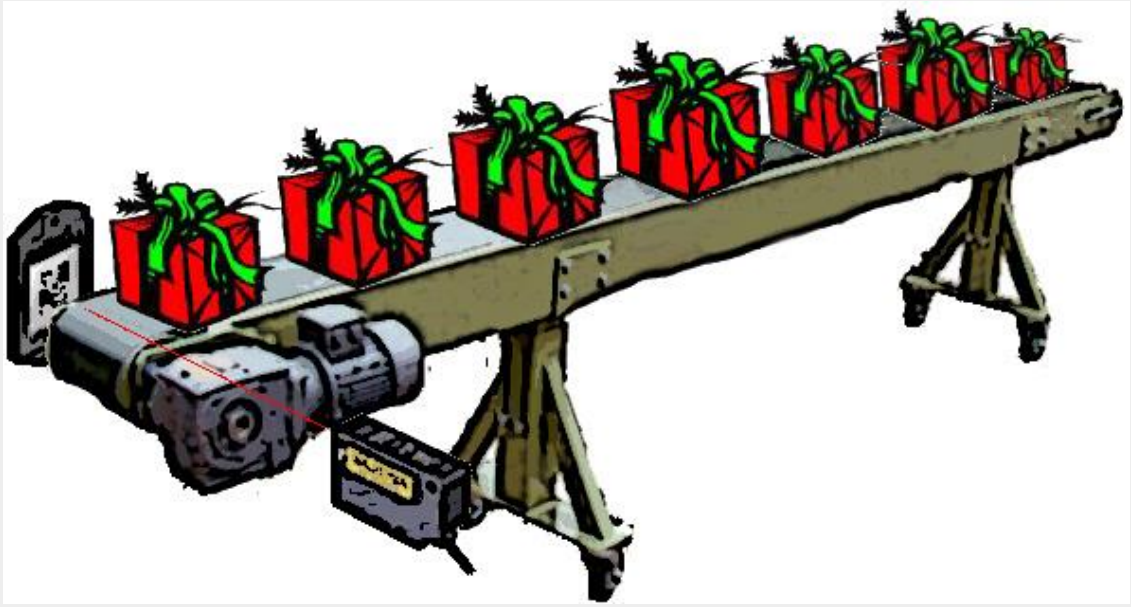


Fig. 20 Conveyor belt

5. LOGIC

5.1.1 AND operation

Fig. 21 Series blocks

만일 2 개 또는 그 이상의 contact 가 직렬로 접속되면, 그 결과는 논리적인 **AND** 연산임.

모든 조건이 만족됐을 때, 출력은 **TRUE** 로 설정됨.

Truth table:

Contact 1	Contact 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

5.1.2 OR operation

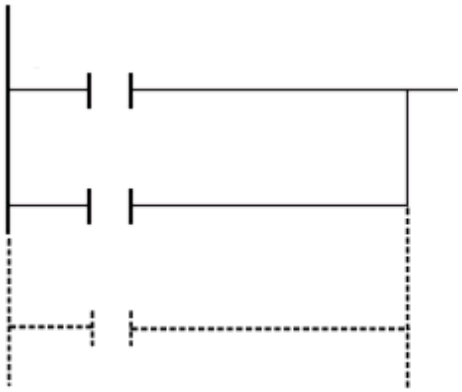


Fig. 22 Parallel blocks

병렬 블록은 **OR** 연산과 같음.

만일 이 병렬 가지 중의 하나라도 **TRUE** 라면, 출력은 **TRUE** 로 설정됨.

Truth table:

Contact 1	Contact 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

5.1.3 XOR operation

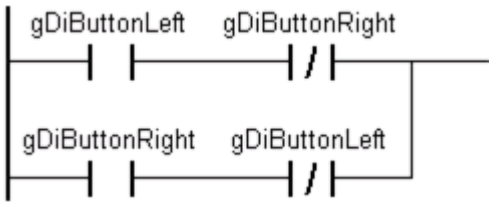


Fig. 23 Exclusive OR

Exclusive OR 연산은 논리적인 **AND** 와 **OR** 연산의 조합임.

만일 2 개의 입력중의 하나가 **TRUE** 라면 출력은 **TRUE** 임.

만일 양쪽 입력 **TRUE** 라면 출력은 **FALSE** 임.

Truth table:

gDiButtonLeft	gDiButtonRight	Output
0	0	0
0	1	1
1	0	1
1	1	0

5.1.4 Branch

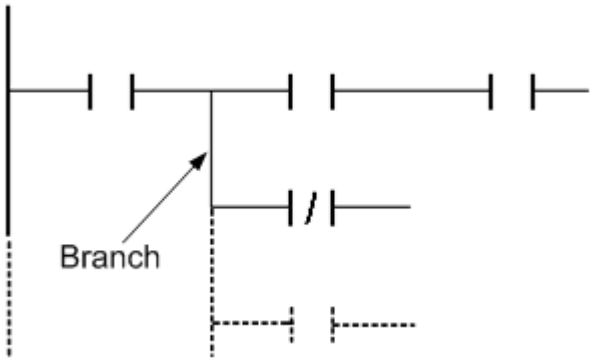


Fig. 24 Branch

네트워크에서 가지(branch)는 적어도 2 개의 열(row)과 서로를 접속하는 수직의 라인을 참조함.

5.1.5 Merge

Merge 선(line)은 가지 선(branch line)과 평행하게 동작하고, 가지 회로(branch circuit)을 폐 회로에 합병하는 또 다른 선으로서 정의됩니다.

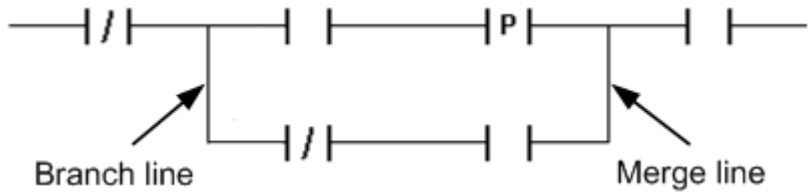


Fig. 25 Merge line

하나의 수직 선은 아래의 이미지에서와 같이, 양쪽 모두 가지 선(branch line)일 수 있고 Merge 선(line)일 수 있습니다.

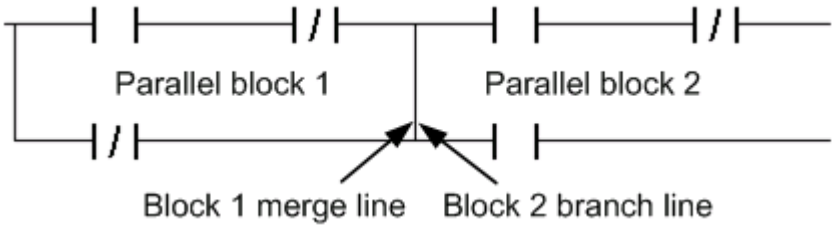


Fig. 26 Branch line and merge line

Task: Part 2: Conveyor belt

현재 컨베이어 벨트를 버튼을 누르는 것에 의해 제어하는 것은 가능합니다. 지금 우리는 자동 모드를 위해 더 약간의 기능을 추가하고 싶습니다.

다음의 경우 컨베이어 벨트를 시작하십시오:

- 만약 컨베이어 벨트의 마지막 센서인 **“gDiLoadConvEnd”**에 어떤 물질도 검출되지 않는다면.
- 만약 컨베이어 벨트의 마지막 센서에 물질이 검출되고 기계가 디지털 입력 **“gDiMachAskMat”**에서 더 많은 물질을 요구한다면.

다음의 경우 컨베이어 벨트를 정지하십시오:

- 만약 컨베이어 벨트의 마지막 센서에 물질이 검출되고 기계가 더 이상의 어떤 물질도 요청하지 않으면.

당신의 프로그램은 이렇게 나타날 것입니다:

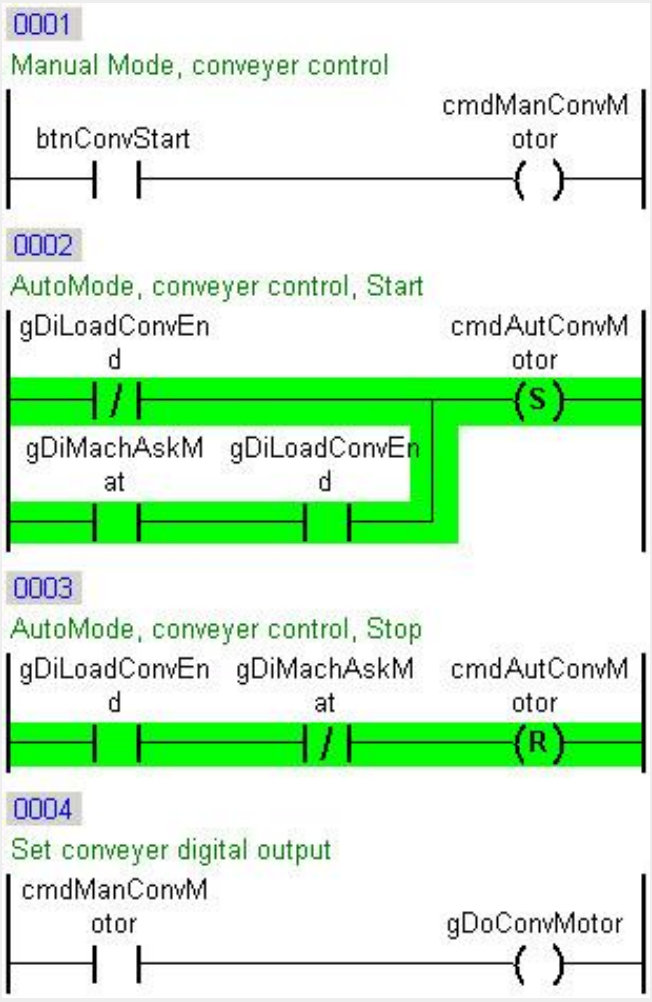


Fig. 27 Part 2 - Source code

6. CONTROLLING THE PROGRAM FLOW

6.1.1 Conditional jump

조건부 점프(conditional jump)는 조건을 사용하는 상징적인 이름과 함께 네트워크에 점프를 참조합니다.

만일 조건이 TRUE 라면, 점프(jump)는 일어납니다. 유일한 이름과 함께 점프 라벨(jump label)은 각 점프에 대해 반드시 존재해야 합니다.

그것은 프로그램에서 네트워크를 뛰어넘기 위해 사용됩니다. 이것은 프로그램 플로우(flow)에게 효율적으로 제어되는 것을 허용합니다. 네트워크가 점프된 후로 만일 그들이 필요하지 않으면 프로그램 런타임은 또한 줄어들게 된다.

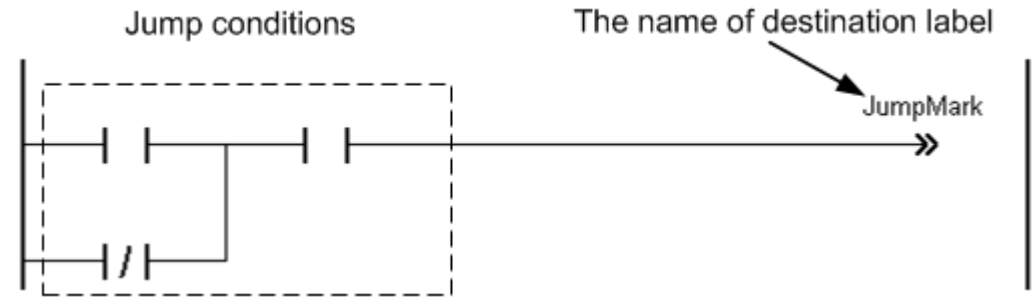


Fig. 28 Jump

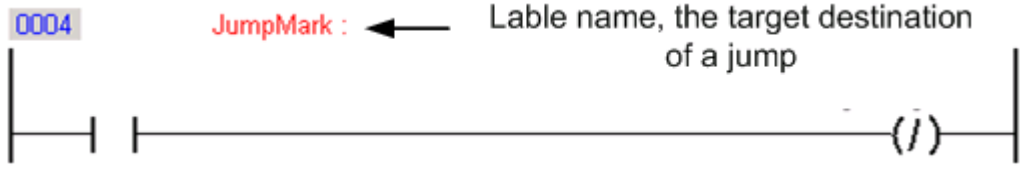


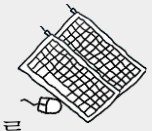
Fig. 29 Conditional jump

6.1.2 Return

Return 명령은 어떤 점에서 Ladder Diagram 을 종료하기 위해서 사용됩니다. 그 후에 어떤 네트워크도 더 이상 실행되지 않습니다.

Fig. 30 Return

Task: Part 3: Conveyor belt



우리는 수동과 자동모드를 변환하여 사용할 수 있는 입력**"gDiAutoMode"**를 추가할 것입니다.

- 만일 "**gDiAutoMode**"가 TRUE 라면, 자동모드에 속해 있는 네트워크는 실행됩니다.
- 만일 "**gDiAutoMode**"가 FALSE 라면, 수동모드에 속해 있는 오직 네트워크만 실행됩니다.
- 조건부 점프(condition jump)를 사용하십시오.
- 자동모드에 새로운 변수 **"cmdAutConvMotor"**를 사용하십시오.

당신의 프로그램은 그 후 이렇게 나타날 것입니다:

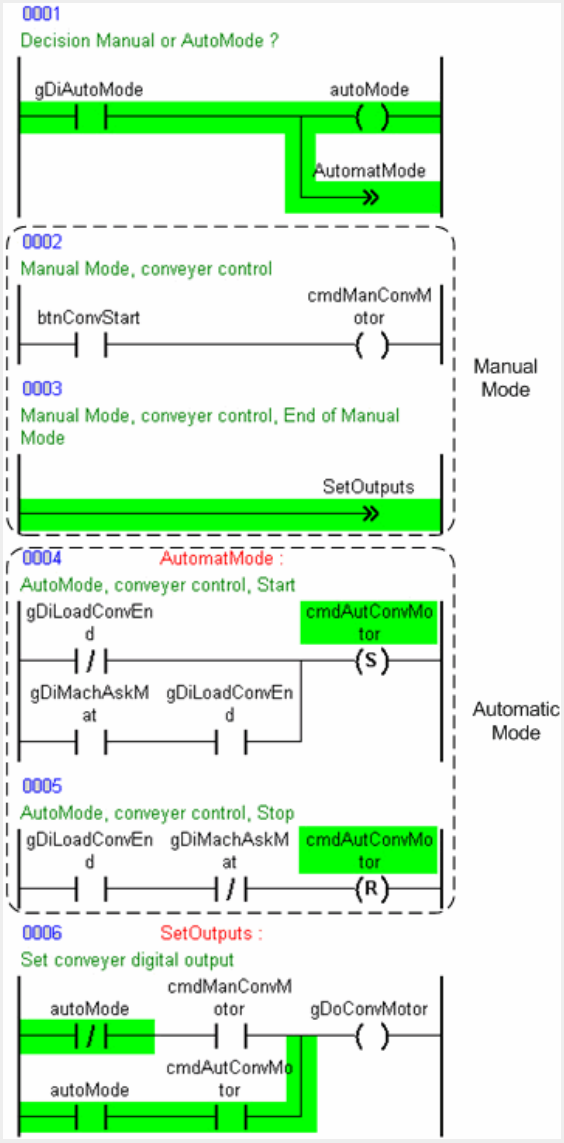


Fig. 31 Part 3 - Source code

7. USING FUNCTION BLOCKS

Automation Studio 의 Ladder Diagram editor 는 펑션 블록(function block)을 사용하는 것을 허용합니다.

만일 펑션 블록(function block)이 삽입되면, 입력 논리 조건들(input logic condition)은 펑션 블록(function block)을 위해 로직(logic)을 가동하는 contact 지시(instruction)에 의해 표현됩니다. 펑션 블록(function block)은 상태를 저장하거나 함수의 결과값 출력으로서 하나 이상의 coil 을 갖고 있을 수 있습니다. 만일 펑션 블록(function block)이 항상 활성화 될 것이라면, 커넥션은 수직선(vertical line)을 사용하는 펑션 블록(function block)에 만들어 질 수 있습니다.

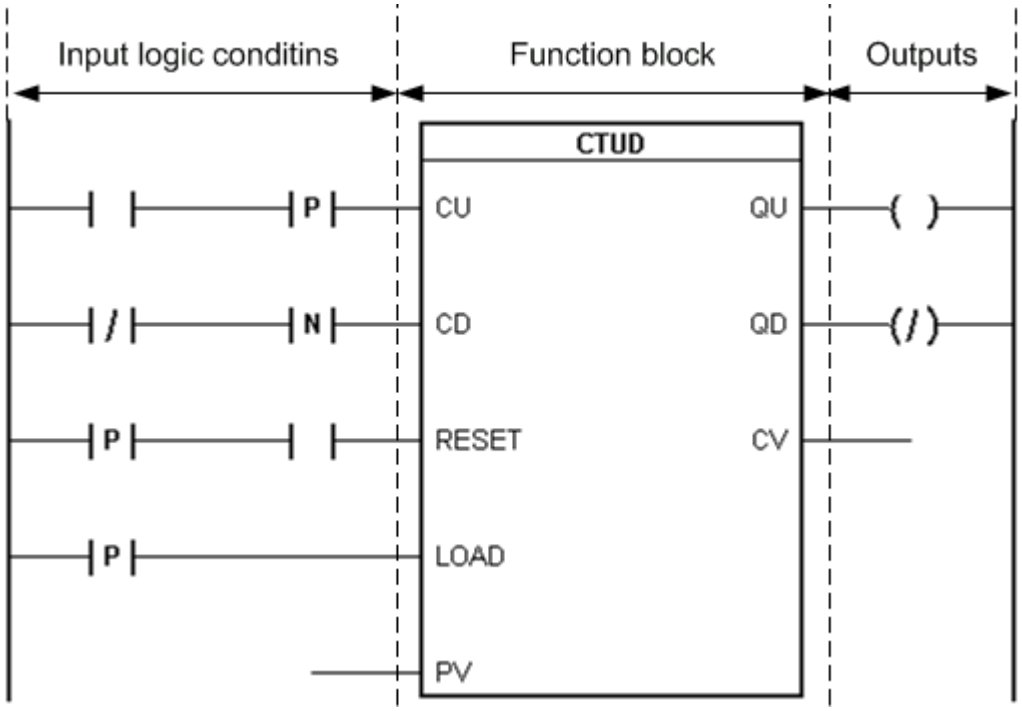


Fig. 32 Function block in Ladder Diagram

펑션 블록(function block)은 또한 “아날로그(analog)” 입력과 출력을 가질 수 있습니다.  아이콘이나 스페이스바(spacebar)로 부터 변수에 접속할 수 있습니다.

Task: Part 4: Conveyor belt

컨베이어 벨트가 자동모드에서 수송하고 있는 물질의 양을 카운트 하십시오.
STANDARD 라이브러리에 있는 **CTU** 평션 블록(function block)을 사용하십시오.

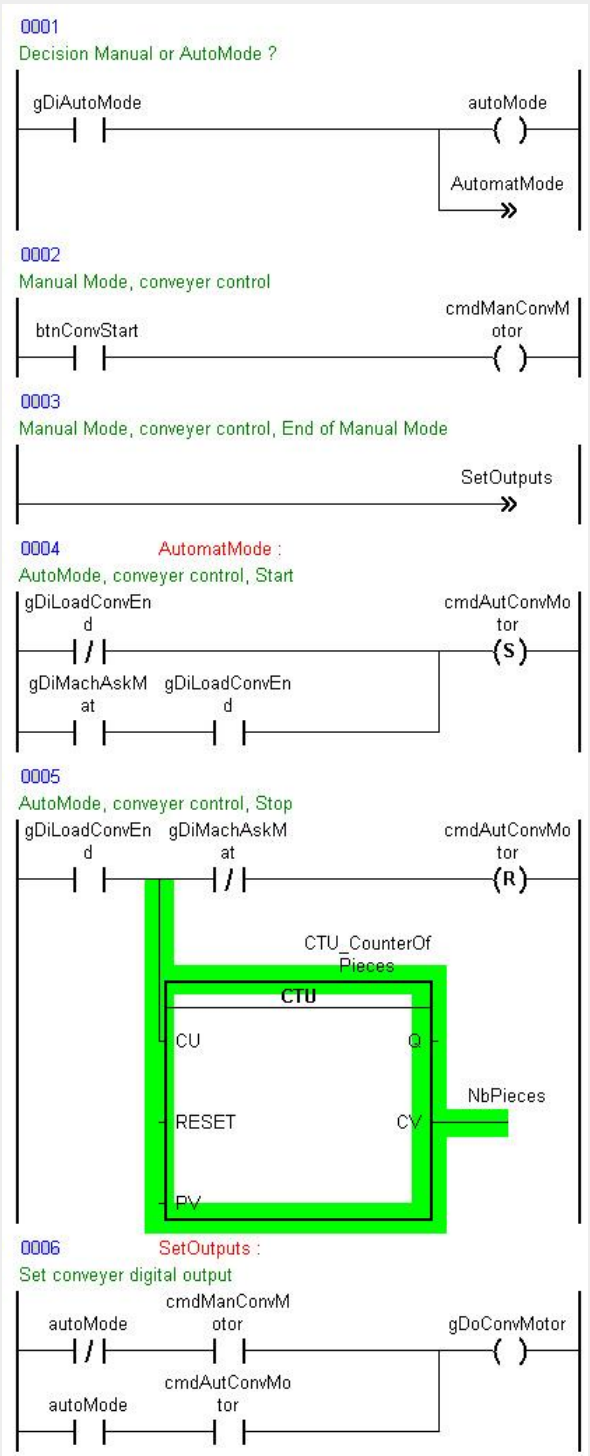


Fig. 33 Part 4 - Source doe

8. POWER FLOW

만일 논리 연결(logic continuity)이 네트워크에 나타나면, 그 출력은 **TRUE** 입니다. 동력(power)은 네트워크에서 왼쪽부터 오른쪽으로 흐릅니다. 진로(course)가 **return** 이나 **jump** 에 의해서 변경될 때, 네트워크는 차례로 실행됩니다.

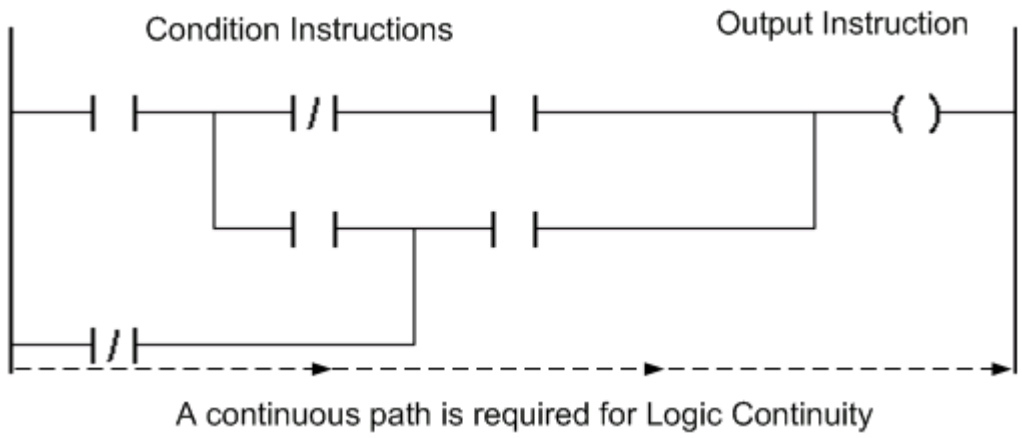


Fig. 34 Logic continuity

논리 연결(logic continuity)의 다른 몇 개의 가능성이 이 네트워크에 있습니다.

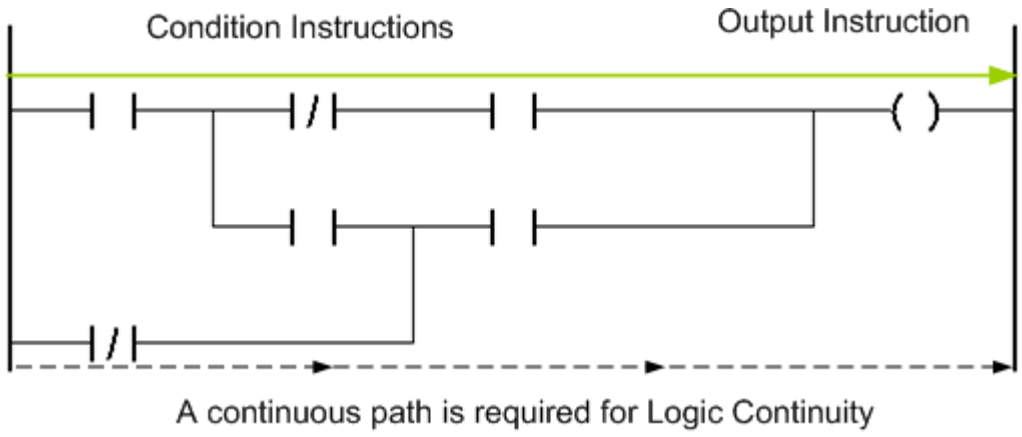


Fig. 35 Logical passage to the first command line

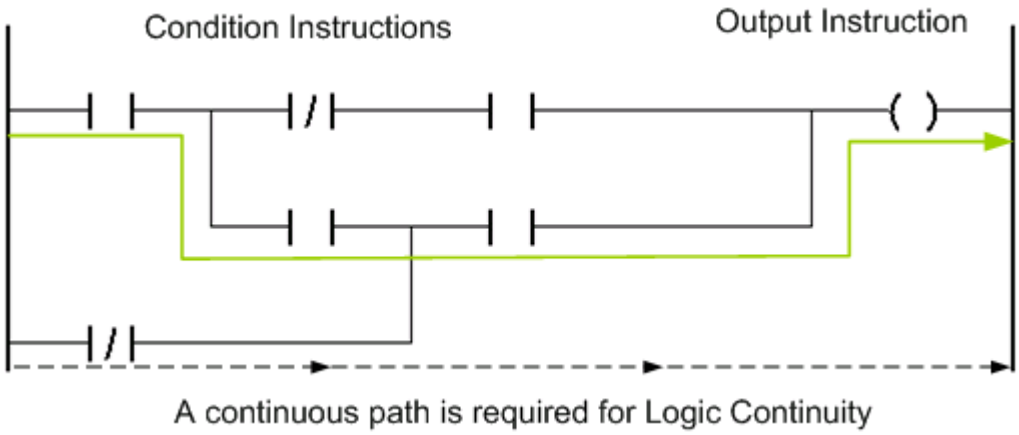


Fig. 36 The second continuous path

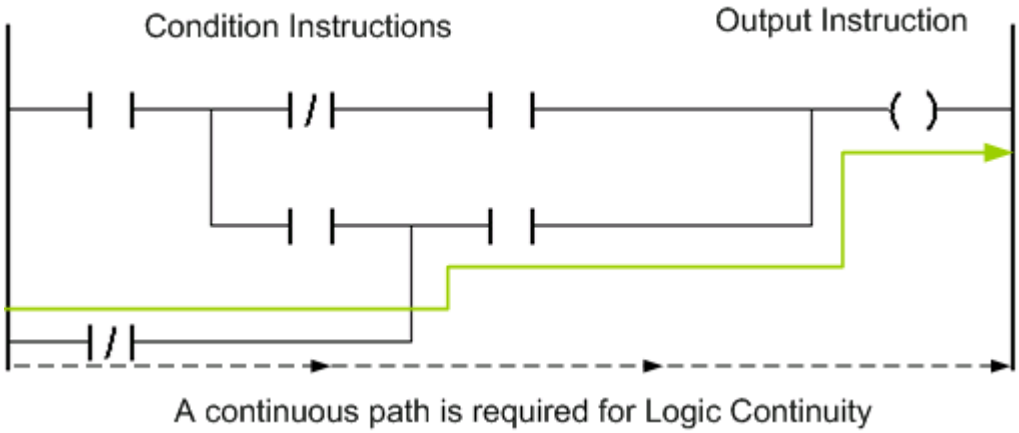


Fig. 37 The third continuous path

하드웨어-배선 릴레이 로직(hard-wired relay logic)과는 다르게, 역으로 작용하는 동력 흐름(power flow)은 나타낸 바와 같이 아래의 이미지에서처럼 PLC 로직에서는 가능하지 않습니다.

만일 로직(logic) 역으로 흐르는 것의 실현을 요구하면, 사용자는 모든 contact 요소에 전방의 동력 흐름(forward power flow)과 더불어 그것을 프로그램 해야만 합니다.

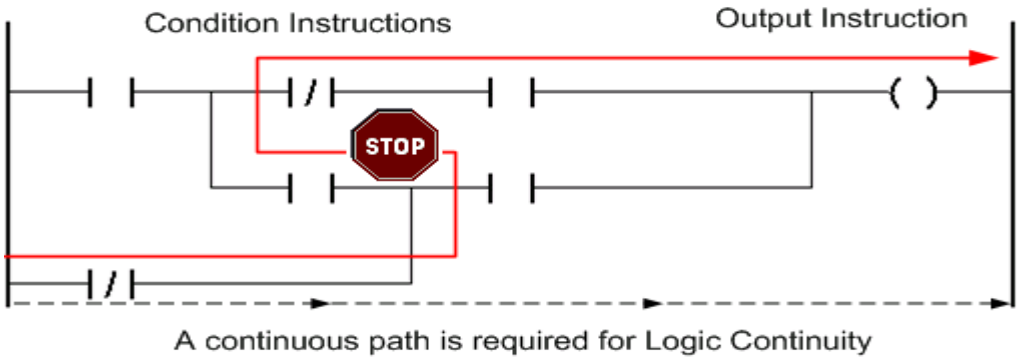


Fig. 38 A reverse continuity path is not possible

9. QUESTIONS AND EXERCISES

9.1 Questions

왜 Ladder Diagram 시스템은 그렇게 광범위하게 받아들여 지는 것입니까? 그것은 왜 개발되었습니까?

- 하드웨어-배선 릴레이 로직(hard-wired relay logic)의 대용으로서 공급될 프로그램을 할 수 있는 로직(logic)으로서 개발되었습니다.

Ladder Diagram 은 언제 사용되어야 합니까?

- 그것은 논리적인 제어 작업(logical control task)과 조작(operation)에 있어서 이상적입니다.

네트워크란 무엇입니까?

- 네트워크는 적어도 1 개의 contact 와 1 개의 coil (입력과 출력)로 이루어져 있습니다. 만일 어떤 커넥션도 2 개의 열(row) 사이에 없으면, 그들은 2 개의 다른 네트워크에 속해 있는 것입니다. 그들은 프로그램 되었던 순으로 네트워크 안에서 실행됩니다.

LD 에서 contact 란 무엇입니까?

- Contact 의 주된 목적은 출력(coil)을 제어하기 위해 논리적인 상태를 형성하는 것입니다.

LD 에서 coil 이란 무엇입니까?

- Coil 은 논리적인 상태에서 일어나는 결과를 받습니다.

9.2 Exercises

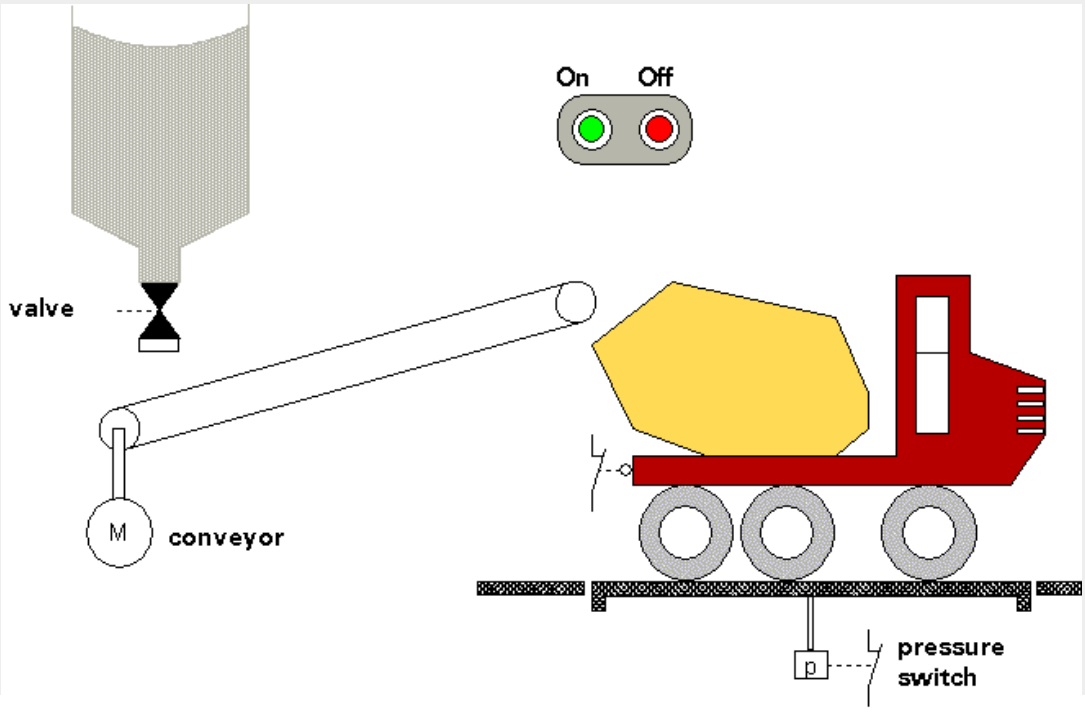
Task: Concrete filling system

시멘트를 섞고 있는 시설에서, 시멘트는 컨베이어 벨트를 사용하고 있는 차량 위에 로드(load)됩니다.

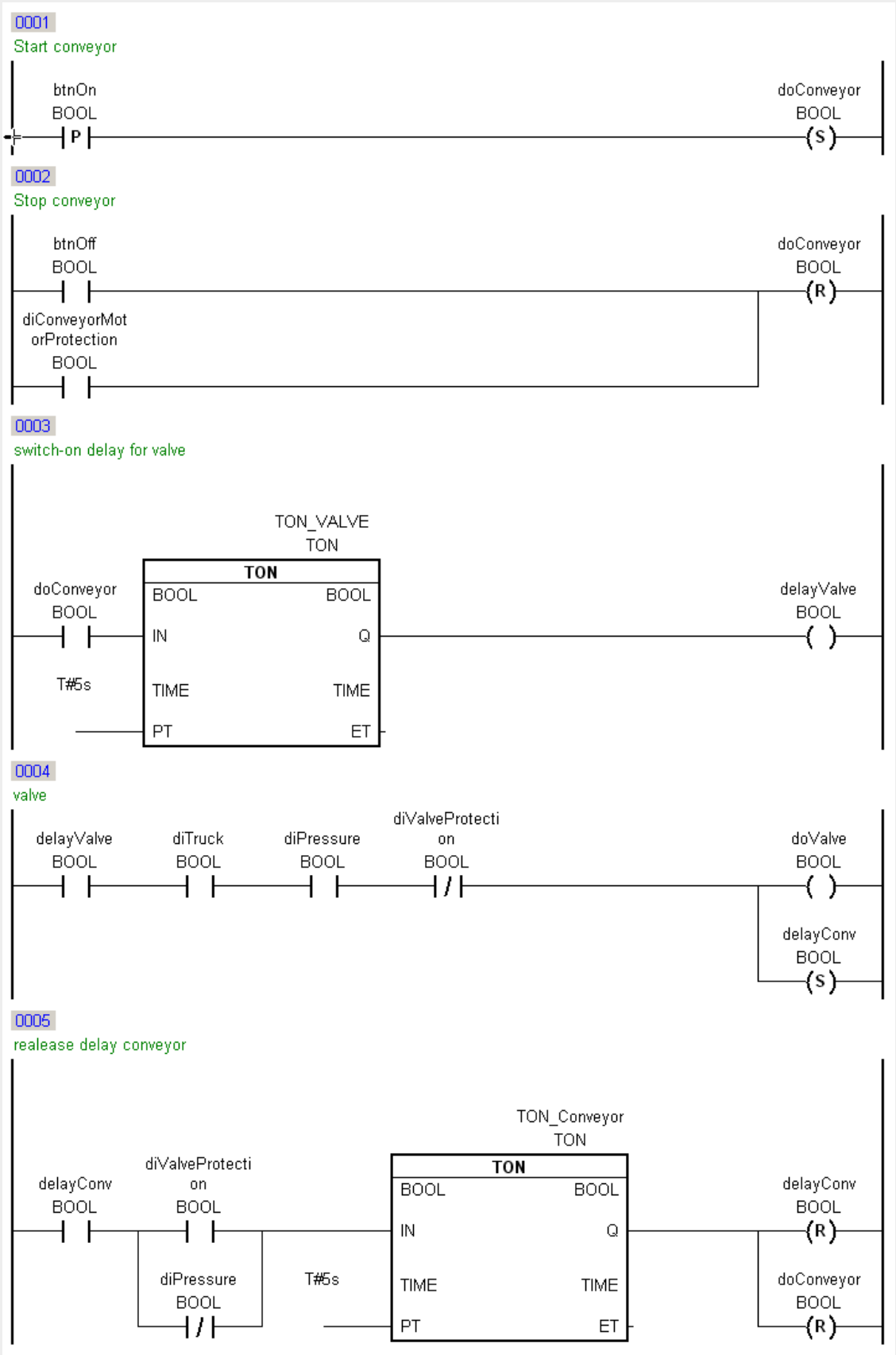
채우는 것(filling)은 On 버튼(btnOn)을 활성화 하면서 시작됩니다. 마그네트 밸브(magnet valve)(**doValve**)를 사용하여 활성화 되는 유압은 오직 컨베이어 벨트가 5 초 동안 가동하면 열립니다. 그리고 차량은 벨트(belt) 아래에 있어야 합니다.(**diTruck**).

차량에게 허락되었던 총 무게에 도달 하자마자(**diPressure**) 마그네트 밸브는 스위치를 끄게 됩니다. 그러나 컨베이어 벨트는 또 다른 5 초 동안 작동을 계속 해야합니다.

전체 시스템은 Off 버튼이 눌리면 즉시 스위치를 끄게 됩니다(**btnOff**). 만일 컨베이어 벨트가 중단되면(**diConveyorMotorProtection**) 마그네트 밸브(**doValve**)와 컨베이어 벨트(**doConveyor**)는 즉시 스위치를 꺼야 합니다. 만일 마그네트 밸브가 중단되면(**diValveProtection**), 그것은 즉시 닫혀야 합니다. 그러나 벨트는 또 다른 5 초 동안 비어 있는 채로 작동해야만 합니다.



Ladder Diagram 은 아래와 같을 것입니다:



10. SUMMARY

Ladder Diagram 을 사용한 프로그래밍은 아직도 인기가 있습니다. 그것은 논리적인 스위치 프로그램을 위해 개발되었기 때문에 하드웨어-배선 릴레이 로직이 대체될 수 있었을 것입니다.

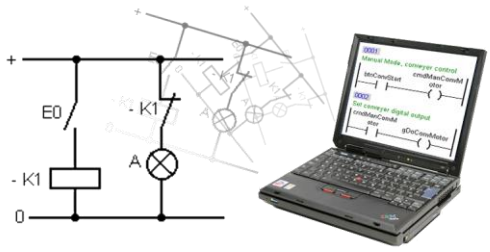


Fig. 39 Summary

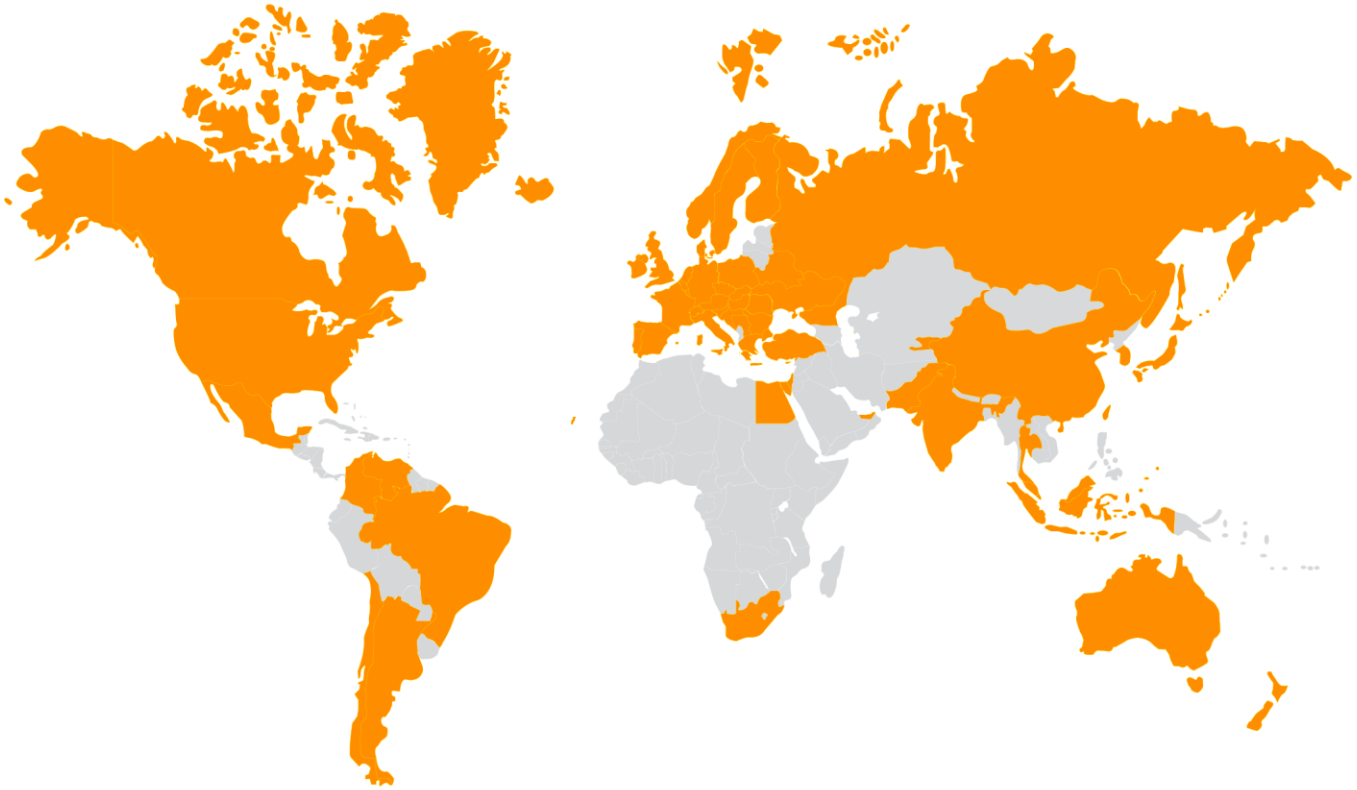
아날로그 신호와 평션 블록을 사용하는 것은 Ladder Diagram 을 사용하고 있는 강력한 어플리케이션을 만드는 것을 가능하게 합니다. 추가적인 프로그램 플로우 제어 요소들은 기능 범위를 확장합니다.

Automation Studio 는 Power Flow 를 사용하여 프로그램 시퀀스를 트레이스(trace) 할 수 있습니다. 색(color)은 현재 전도되고 있는 전기 라인 상태를 표시하기 위해 사용됩니다.

CORPORATE HEADQUARTERS

Bernecker + Rainer Industrie-Elektronik Ges.m.b.H.
B&R Strasse 1
5142 Eggelsberg
Austria
Tel.: +43 (0) 77 48 / 65 86 - 0
Fax: +43 (0) 77 48 / 65 86 - 26
info@br-automation.com
www.br-automation.com

140 offices in more than 55 countries - **www.br-automation.com/contact**



Australia • Argentina • Austria • Belarus • Belgium • Brazil • Bulgaria • Canada • Chile • China • Colombia • Croatia • Cyprus
Czech Republic • Denmark • Egypt • Emirates • Finland • France • Germany • Greece • Hungary • India • Indonesia
Ireland • Israel • Italy • Japan • Korea • Luxemburg • Kyrgyzstan • Malaysia • Mexico • The Netherlands • New Zealand
Norway • Pakistan • Poland • Portugal • Romania • Russia • Serbia • Singapore • Slovakia • Slovenia • South Africa