PERFECTION IN AUTOMATION
A MEMBER OF THE ABB GROUP

TM440

# 모션제어 – 기본 기능

# 버전 정보

| 버전 | 날짜 | 수정내역 | 번역 | 검수 |
|---|---|---|---|---|
| 1.0 | 2017.11.10 | 첫번째 버전<br>TM440TRE.00_V4200 (V1.0.0.2) with Mapp | - | 임은 |
| | | | | |
| | | | | |
| | | | | |

Table 1: Versions

**선행 및 필요 조건**

| | |
|---|---|
| 교육 자료 | TM240 – Ladder Diagram(LD) 또는 TM246 – Structured Text(ST)<br>TM410 – Working with Omtegrated Motion Control |
| 소프트웨어 | Automation Studio 4.2<br>Automation Runtime 4.08<br>Mapp Technology 1.00.0 |
| 하드웨어 | X20 controller + ACOPOS / ACOPOSmulti<br>Or<br>Simulation |

# II 목차

# 1 소개

어플리케이션 프로그램은 각 위치 결정 임무에 기본이다.

이 곳에서 커맨드가 지정되고 드라이브로 전달되며 공정 신호가 평가되는 곳이다. 어플리케이션 프로그램은 자동 시퀀스 공정에서 사용되는 드라이브를 제어한다. 드라이브 진단은 어플리케이션 소프트웨어와 화면 시스템을 포함한다.

위지 지정 응용 프로그램을 설계하기 위한 기초는 사용 가능한 도구의 개요를 만드는 것이다.

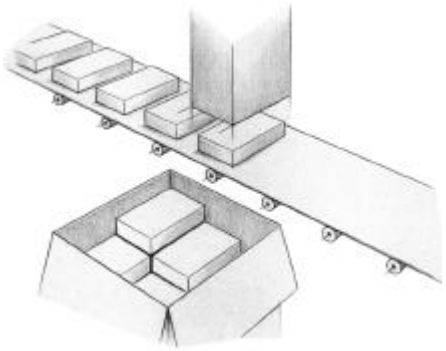


**그림 1 대략적인 Palletizing 이미지**

이 교육 자료는 어플리케이션 프로그램에 MpAxis 라이브러리의 mapp technology 컴포넌트 사용법 및 통합을 다룬다.

다른 연습 문제를 통해 펑션 블럭의 행동을 이해하고 기능을 적용하는 중요한 기본 지식을 습득할 것이다. 레더 다이어그램은 펑션 블럭을 호출하기위한 이미지 소스 파일과 예제로 사용된다. 어플리케이션 프로그램을 만드는데 어떤 프로그래밍 언어도 사용할 수 있다.

## 1.1 학습목표

선택된 실습을 통해서, 당신은 어플리케이션과 PLCopen 호환 표준 기능 통합 개요를 배울것이다. 이는 드라이브 준비와 기본 구동을 위해 사용된다.

- Mapp technology 컨셉 개요에 대해 알게 될 것이다.
- MpAxis 라이브러리와 라이브러리에 포함된 펑션 블럭 구조에 대해 알 수 있다.
- PLCopen 호환 표준 펑션 블럭을 어떻게 프로젝트에 통합하는지 배울 것이다.
- 드라이브 준비와 기본적인 구동 시행을 위한 필수 단계를 배울 것이다.
- 어플리케이션 프로그램의 정형화된 구조와 PLCopen 상태 개통도 내 관계를 이해할 수 있을 것이다.
- "axis period"과 "axis factor"의 의미와 계산 및 구성 방법을 배울 것이다.
- MpAxis 컴포넌트 관리와 드라이브 구성 로드와 저장을 알 수 있을 것이다.
- 어플리케이션 프로그램 내 통합 상태 평가를 위한 접근에 대해 알 수 있을 것이다.
- ACP10_MC 라이브러리 구조와 개요에 대해 알 수 있다.

## 1.2 PLCopen standard

소프트웨어의 역할은 자동화 시스템에서 지속적으로 증가한다. 개발, 지원 및 유지 보수 같은 어플리케이션에 복잡성이 증가되고 있다.

PLCopen 협회는 자동화 분야에서 다양한 활동 영역을 통합하고 표준화 한다는 목표로 창안되었습니다. 활동 영역은 드라이브 기술, 안전 기술, IEC 61131-3 표준 뿐만 아니라 OPC UA 와 XML 까지 포함한다.


**그림 2 PLCopen 협회 활동과 다른 영역**

PLCopen 협회와 활동에 관한 세부 정보는 인터넷 http://www.plcopen.org/ 에서 찾을 수 있다.

다른 시스템 솔루션들이 모두 동일한 방식으로 동작하는 것을 보증하기 위해 가이드라인이 개발되었다.

협회에 가입되어 있는 모든 자동화 솔루션 공급자들은 PLCopen 을 사용하는 지정된 시스템을 위해 모두 동일한 소프트웨어 인터페이스를 제공한다. B&R 은 주요 멤버이다.

### B&R 의 완벽한 지원

PLCopen 을 준수하는 펑션 블럭은 B&R 드라이브 솔루션에서 이용가능하다. 표준화된 펑션 블럭의 사용 덕분에 프로젝트 설정과 구성을 빠르고 쉽게 수행할 수 있다.

# 2 Mapp technology

Mapp 기술과 함께, 우리는 사용자에게 광범위한 기능 구현을 위한 사용하기 쉬운 인터페이스를 제공한다. 많은 복합적 구동, 그러한 로딩과 레시피 데이터(recipe data), 드라이브 축 제어, 그리고 프로세스 값 레코딩은 사용하기 쉬운 mapp 기술 컴포넌트에 의해 수행된다.

**그림 3 mapp technology**

Mapp 기술은 구성과 프로그래밍을 통합한다. 기능은 표준 라이브러리를 사용하여 어플리케이션 프로그램 내에 추가된다. 게다가, mapp 은 mapp 구성요소가 어플리케이션 소프트웨어 내에 그들의 추가 구성이 필요없는 구성 인터페이스를 제공한다.

Application layer – mapp technology

- Concept
- Getting started
- Components

## 2.1 mapp 기술 컴포넌트 사용 지침

아래의 단계는 mapp 컴포넌트를 사용할 때 처음에 수행하는 단계이다.

- 컨피규레이션 뷰(Configuration view)로 이동
- 툴박스에서 mapp[1]기술 패키지 추가
- Mapp 컴포넌트를 위한 표준 구성을 툴박스에서 추가
- 컨피규레이션 뷰 / MpLink 이름 재정의
  컨피규레이션 뷰은 다음과 같다:

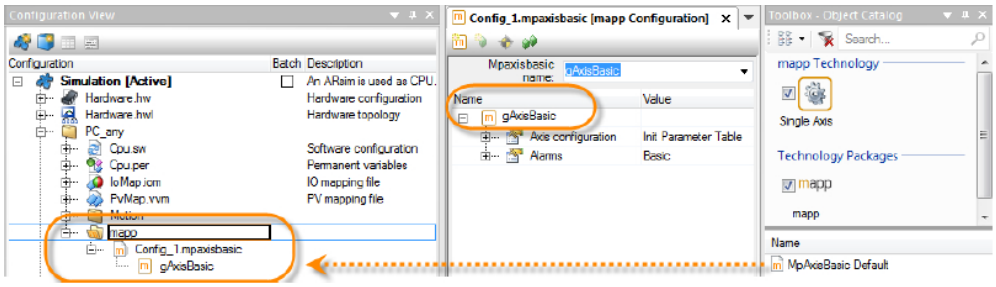



**그림 4 mapp 기술 패키지와 함께 MpAxisBasic 기능을 위한 표준 컨피규레이션 뷰**

컨피규레이션 뷰에서 MpLink 는 ADR() 펑션 사용으로 프로그램 내부 펑션 블럭으로 전달 된다. 맵 컴포넌트와 프로그래밍 코드 연결을 설정한다.

Application layer – mapp technology ₩ Concept ₩ Component design ₩ Adding mapp components

[1] Mapp 기술은 "Modular APPlication technology"을 상징한다.

### Mapp 컴포넌트 호출(Calling mapp components )

Mapp 컴포넌트 펑션 블럭은 모든 제어 싸이클에서 호출되어야 한다. 상위 레벨 프로그램을 사용할 때(Ansi C, ST, etc.), 프로그램 마지막에 mapp 컴포넌트를 불러오는 것이 좋다.

모든 mapp 펑션 블럭은 “Enable”입력이 있다. 이 입력은 mapp 컴포넌트에서 사용되고, 각 mapp 컴포넌트를 자동으로 로드하기 위한 구성이다. Mapp 컴포넌트에 성공적인 초기화는 출력에 “Active = TRUE”로 나타난다.

Application layer – mapp technology ₩ Concept ₩ Component design ₩ Using mapp components

### 다운로드 행동

제어기는 프로그램을 전송할 때마다, Automation studio 에서 설정한 표준에 따라서 재시작된다. 드라이브 어플리케이션 구현 단계 동안 “Copy mode” 사용을 권장한다. 이렇게 사용하는 동안, 제어기를 여러번 재 시작할 필요가 없다. 전송 타입 구성은 활성화된 컨피규레이션 뷰 / 바로 가기 메뉴를 사용하여 열 수 있다. 전송 방법은 Transfer 탭 내부에 Advanced 버튼을 사용하여 설정한다.

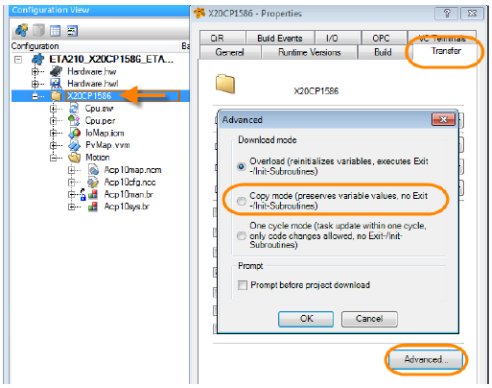



**그림 5 컨피규레이션 뷰 / 속성 창 /전송 방법 설정**

Application layer – mapp technology ₩ Concept ₩ Component design ₩ Using mapp components

Real-time operating system ₩ Target system ₩ SG4 ₩ Download

### 구성 파일

모든 mapp 컴포넌트를 위한 구성을 이용할 수 있다. 그 구성은 Automation Studio 내 컨피규레이션 뷰에서 생성, 변경되며, WebXs 는 웹기반 인터페이스 또는 어플리케이션 프로그램이다. Mapp 구성에 대한 추가 정보는 Automation Studio 도움말에서 찾을 수 있다.

Application layer – mapp technology ₩ mapp ₩Concept

- Component design ₩ Adding mapp components
- Configuring components

## 2.2 mapp 기술 컴포넌트를 위한 진단 옵션

mapp 기술 컴포넌트는 몇가지 다른 방법을 통해 모니터링 및 진단한다. Automation Studio 내 진단 옵션 리스트, 웹기반 진단과 화면작화 어플리케이션이 있다.

### 모니터링 모드에서 프로그래밍 언어

많은 경우, 어플리케이션 프로그램에서 접근하는 첫번째 방법은 모니터링 모드이다. 프로그램 코드와 프로세스 변수 값은 문맥에서 보인다. 모든 mapp 기술 컴포넌트는 “Error”와 “StatusID” 출력을 가지고 있으며, 초기 진단을 실행하는데 사용된다.

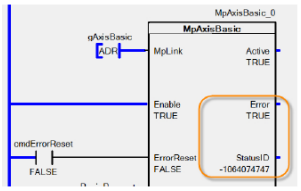
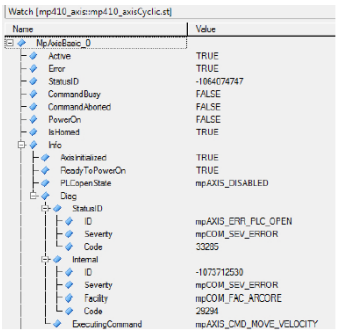



**그림 6 모니터링 모드 중 레더 다이어그램**




### 와치 윈도우(Watch window)

와치 윈도우는 프로그램 단축 메뉴나 소프트웨어 구성에서 <Watch>메뉴를 선택함으로서 로직컬 뷰에서 열린다. 펑션 블럭 인스턴스 변수를 툴바를 사용하거나 단축 메뉴로 추가한다. 예를 들어, Error, StatusID, CommandBusy 출력과 구조 정보는 현재 상태 진단에 도움을 줄 수 있다. 이 에러 번호에 대한 설명은 각 펑션블럭 도우말에서 찾을 수 있다.

**그림 7 와치 윈도우에 MpAxisBasic 펑션 블럭 변수 사용**

### 로거

에러 내부에서, mapp 기술 컴포넌트으로부터에 추가 정보는 "$mapp" 이름 로거 파일에 기록된다. 에러 번호는 Automation Studio 도움말에서 직접 검색하거나 <F1>키를 누름으로서 호출될 수 있다. 추가 정보는 강조된 로거 항목 섹션에서 볼 수 있다. 예를 들어서 PLCopen 에러이면, 이것은 펑션에 영향을 주고 logger 항목 섹션 세부 정보에 에러 원인이 설명되어 있다.

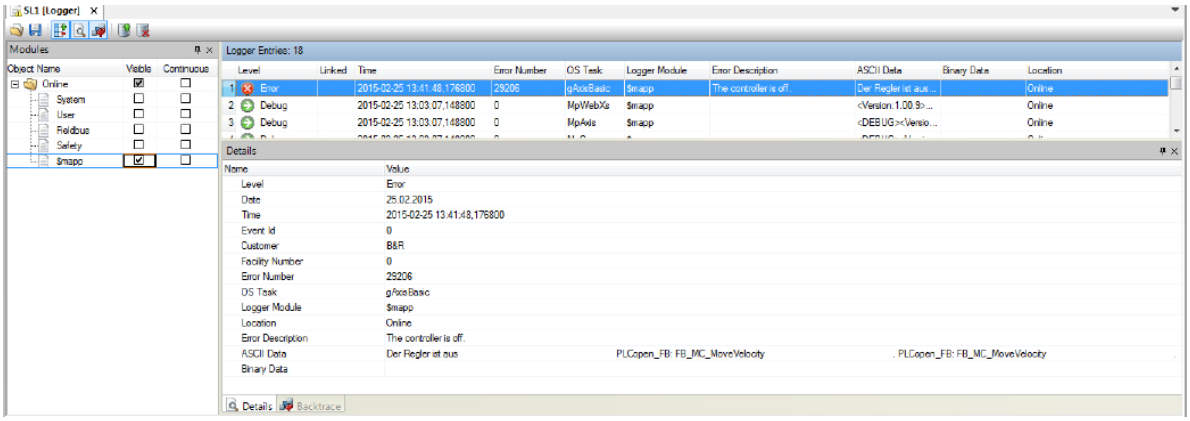



**그림 8 로거 윈도우에 PLCopen 에러**

### 트레이스(Trace)

Automation Studio 트레이스 기능과 함께, 프로세스 변수 값은 실시간으로 녹화되고 저장된다. 이것은 파라미터 입력 타이밍과 변수 상태를 효과적으로 시작화 할 수 있습니다.

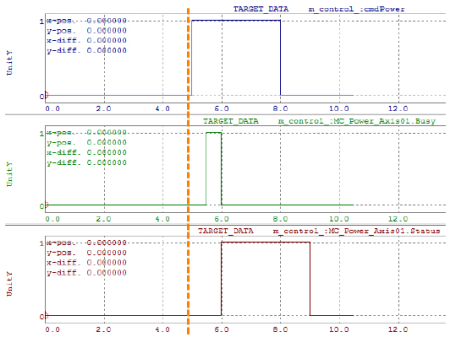


**그림 9 cmdPower 명령어로 제어기 전원 켜기: 명령과 상태 정보 사이에 시간 관계**

### 시스템 진단 매니저(System Diagnostics Manager)

"Application Status"버튼은 시스템 진단 매니저를 호출할 수 있다. mapp 기술을 위한 WebXs 에 직접 열린다. 더 나아가, SDM 내 mapp 컴포넌트를 위한 로거 데이터가 저장이 가능한다. 또한 드라이브 축을 위한 기본 진단도 가능하다. SDM 은 HTML 컨트롤을 사용하여 화면작화 객체에 직접 포함될 수 있다.

### Mapp WebXs

Mapp 기술 WebXs 를 사용함으로서, 사용된 모든 mapp 컴포넌트들은 웹기반 인터페이스 위에 보여진다. Mapp 컴포넌트 구성과 알람은 진단 이외에 컴포넌트의 인스턴스 변수를 통해 제공된다[2].

### MpAlarm 컴포넌트 사용한 화면 컴포넌트 통합

Mapp 기술 컴포넌트는 알람을 미리 지정한다. 특정 유저가 알람을 구성할 수 있다. 화면 구성 알람 시스템에 Mapp 알람 출력은 MpAlarm 구성을 사용하여 사용 될 수 있다.

### MpComLoggerUI 컴포넌트를 사용한 화면 컴포넌트 통합

Mapp 기술에 모든 관리 기능은 이벤트 로거에 저장된다. 이 로거 항목들은 pComLoggerUI 컴포넌트를 사용한 화면 컴포넌등에 쉽게 통합 될 수 있다. 필터 기능은 각 mapp 컴포넌트, 특정 에러 번호, 이벤트 타입 검색이 가능하다. 그러므로 로거 항목을 필터링하기 위한 추가 프로그램은 필요 없다.

Diagnostics and service ₩ Diagnostic tool ₩

- Logger
- Watch windows
- Monitors ₩ programming languages in monitor mode
- Trace
- System Diagnostics Manager

Application layer – mapp technology ₩

- WebXs
- Components ₩ Infrastructure
  - MpAlarm – Support for alarm management
  - MpCom – mapp management ₩ function blocks ₩ MpComLoggerUI
- Diagnostics ₩ Logger window

[2]컴포넌트 사용에 따라 웹기반 구성이 수행된다.

# 3 제어 프로젝트에 축 통합하기

Automation Studio 내 드라이브 구성 생성 절차는 트레이닝 모듈 “TM410 – Working with Integrated Motion control”에 설명되어 있다. 드라이브 구동은 NC 테스트를 사용하여 수행될 수 있다.

수행된 예는 축 구동 제어를 위하여 어떻게 어플리케이션 프로그램이 단계별로 생성되는지 보여준다.

축 객체 접근을 위한 축 레퍼런스가 우선적으로 요구된다. 축 레퍼런스는 드라이브 구성 마법사를 통해 생성되고 전역변수 선언과 NC mapping 테이블에 추가된다.

제어기가 시작될 때 드라이브는 NC Init 모듈로부터 데이터를 받아서 자동으로 초기화 된다. (이것은 NC mapping 테이블에 할당된)

Motion ₩ Projection Configuration ₩ Motion control

- Settipng up an axis
- Configuration module ₩ NC mapping table
- Configuration module ₩ NC Init module

**시작하기**
각 시작은 어떻게 MpAxisBasic mapp 기술 컴포넌트가 드라이브 준비와 기본 구동을 위해 수행되는지보여준다.

Application layer – mapp technology ₩ Getting Started ₩ Quickly starting an axis

## 3.1 MpAxis 컴포넌트

MpAxis mapp 기술 컴포넌트는 드라이브 제어와 구성을 위하여 표준 기능을 제공한다. MpAxisBasic 펑션 블럭은 드라이브 제어에 사용된다. MpAxisBasicConfig 펑션 블럭은 드라이브 구성을 관리하기 위해 사용된다. 다음 펑션 그룹은 MpAxisBasic 펑션 블럭으로 단일 축 제어를 다룬다.

- 드라이브 준비
- 기본 구동
- 자동 튜닝
- 에러 다루기
- 주기적인 드라이브 데이터

MpAxisBasic 라이브러리는 ACP10_MC 라이브러리 펑션 블럭을 기반이다. 그러므로 두 라이브러리는 하나에서 또 다른 하나로 호환 가능하고 어플리케이션에서 보충 될 수 있다 (“ACP_10_MC 와 MpAxis 호환” 참조).

모든 mapp 컴포넌트는 오픈 스탠다드를 기본으로 하고, 기술 펑션과 라이브러리는 어플리케이션에서 유저가 직접 사용할 수 있다.

사용된 펑션 블럭과 펑션은 Automation Studio 도움말에서 상세한 정보를 확인할 수 있다.


**그림 10 mapp 컴포넌트는 오픈 스탠다드, 기술 펑션과 라이브러리를 기반이다.**

Application layer – mapp technology ₩ Components ₩ Mechatronics ₩ MpAxis – individual and muti-axis controllers ₩ Technical information ₩ The use of internal PLCopen

Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC

## 3.2 프로그램 생성 및 MpAxisBasic 추가

MpAxisBasic 펑션 블럭을 호출하여 Automation Studio 를 확장하는 것이 필수적이다. 준비 단계에서는, 프로젝트 / 컨피규레이션 뷰(Configuration View)에 mapp 기술 패키지를 추가하여 확장된다. 그 다음, MpAxisBasic 컴포넌트 표준 구성을 컨피규레이션 뷰에 추가한다. MpAxis 라이브러리는 로지컬 뷰(Logical view)에 추가된다.

마지막으로, MpAxisBasic 펑션 블럭은 컨피규레이션 뷰에 설치된 것으로 프로그램에 추가되고 MpLink 와 연결된다.

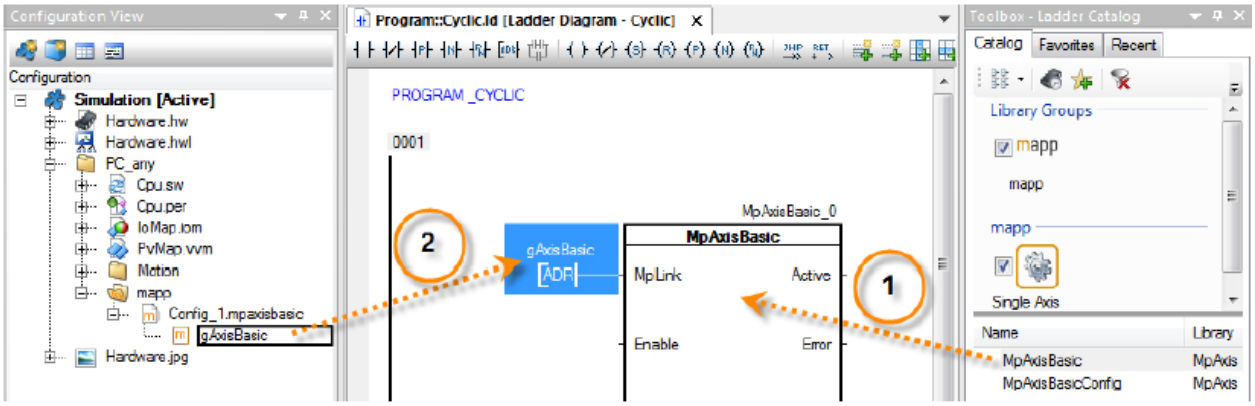



**그림 11 (1)툴박스로부터 MpAxisBasic 추가와 컨피규레이션 뷰에서 Mplink 주소 전송**

Application layer – mapp technology ₩ Concept ₩ Component design ₩ Adding mapp components

### 예제: MpAxisBasic 을 위한 mapp 기술 구성 추가

이제 mapp 기술 패키지를 추가하여 Automation Studio 를 확장하고, MpAxisBasic 컴포넌트 표준 구성을 추가하자. MpAxis 라이브러리를 전송하고, 그 다음 프로그램은 새로 생성하고, MpAxisBasic 펑션 블럭을 추가한다. ADR() 펑션을 사용하여 컨피규레이션 뷰에 MpLink 를 연결한다.

1) 컨피규레이션 뷰(Configuration view) 열기
2) 툴박스에서 mapp 기술 패키지 추가
3) 컨피규레이션 뷰에 MpAxisBasic 표준 구성 추가
4) 로지컬 뷰(Logical view)에 MpAxis 라이브러리 추가
5) “m_control“ 레더 다이어그램 프로그램 추가
6) 툴박스로부터 MpAxisBasic 펑션 블럭 추가
7) ADR() 펑션을 사용하여 MpAxisBasic 펑션 블럭에 컨피규레이션 뷰에 MpLink 할당

이제 mapp 컴포넌트를 이용하여 원하는 드라이브에 접근할 수 있는 프로젝트 준비가 끝났다.

## 3.3 축 레퍼런스와 구동 파라미터 연결

축에 접근할 수 있고 기본 구동 수행을 위하여 축 레퍼런스와 움직임 파라미터 전송은 필수이다.

**축 레퍼런스(axis reference) 활용**
드라이브 구성 위자드와 함께, ACP10AXIS_typ 타입에 전역 프로세스 변수(global PV)가 생성된다. 프로세스 변수 이름은 NC mapping 테이블에 자동으로 입력된다. 그리고 연결은 파워링크(POWERLINK)을 통해 실제 하드웨어와 소프트웨어 객체가 연결된다.

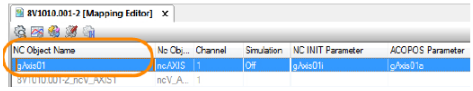



그림 12 NC mapping table 에 축 레퍼런스

축 레퍼런스 주소(address)는 “Axis“ 파라미터를 사용하여 모든 펑션 블럭에 할당된다.

레더 다이어그램에서, 주소 연결은 어드레스 펑션(ADR) 사용으로 대체될 수 있다.

다른 펑션 블럭에서 축에 접근할 때 동일한 방식을 사용한다.

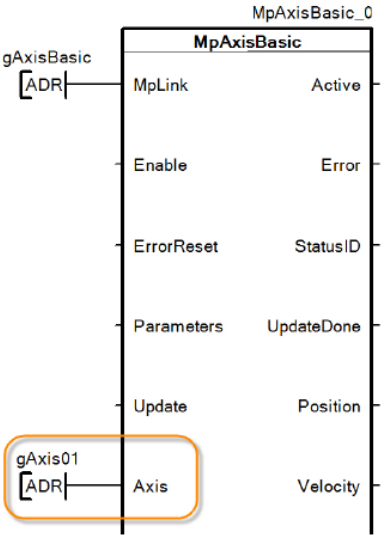



**그림 13 축 레퍼런스는 주소 접속을 통하여 MpAxisBasic 을 위한 펑션 블럭이 할당된다.**

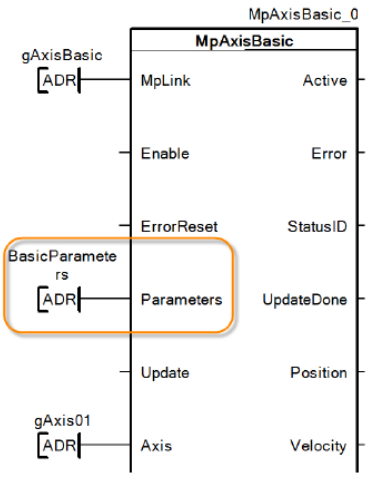



**그림 14 구동 파라미터와 데이터 전송(MpAxisBasicParType)**

## 구동 파라미터 전송

MpAxisBasic 펑션 블럭을 위하여, 구동 파라미터와 데이터 구조 전송이 필수이다. 데이터 구조는 표준 값이 미리 설치되어 있다. 아래의 구동 파라미터는 전송 될 수 있다.

- 속도와 가속도
- 거리, 위치 그리고 회전 방향
- 홈잉(Homing) 파라미터
- 조그(Jog) 파라미터
- 토크 제어
- 주기적으로 축 정보 읽기 위한 설정
- 오토 튜닝

변수 선언에 데이터 구조를 설정할 때 초기화 값은 자동으로 “0“으로 입력된다. 데이터 구조에 디폴트 값을 사용하기 위해서, 변수 선언 초기값은 반드시 지워야한다.

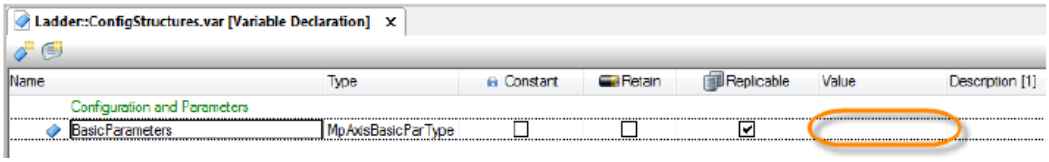



**그림 15 변수 선언에서 디폴트값 초기화와 초기값 삭제**

Application layer – mapp technology ₩ Components ₩ Mechatronics ₩ MpAxis – individual and muti-axis controllers ₩ Function blocks ₩ MpAxisBasic Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC ₩ Concept ₩ Implementation

**예제: 축 레퍼런스 할당과 제어기 켜기**

기존 프로그램은 축 레퍼런스, 데이터 구조, 기본 파라미터를 MpAxisBasic 펑션 블럭으로 전송할 수 있도록 확장된다.

"Enable" 입력은 MpAxisBasic 컴포넌트를 사용 가능하도록 반드시 TRUE 로 설정한다.

드라이브 준비를 위하여[3] 드라이브를 켜는 것이 필수다. 만일 "Active" 출력=TRUE 이고, 제어기는 MpAxisBasic 펑션 블럭 입력 "POWER"을 통하여 켜진다.

1) 어드레스 펑션을 사용하여 축 레퍼런스 입력 할당
2) "BasicParameter"구조와 "Parameter"입력 연결
3) 프로그램을 제어기에 전송
4) "Enable"입력을 TRUE 로 설정
5) "Active" 출력과 "Info.ReadyToPowerOn" 출력이 나올때 대기
6) "POWER"를 통하여 드라이브 스위치 켜기
7) MpAxisBasic 상태 출력 관찰

"BasisParameters"구조는 MpAxisBasicParType 데이터 타입을 기본으로 한다. 구조는 디폴트값으로 미리 초기화되어 있다. 예를 들어서 이를 고려해 가져오면. 구성된 속도와 가속도는 반드시 유닛 구성시스템과 일치하여야 한다.

Application layer – mapp technology ₩ Components ₩ Mechatronics ₩ MpAxis – individual and muti-axis controllers ₩ Function blocks ₩ MpAxisBasic Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC ₩ Function blocks ₩ Drive preparation

[3] ACOPOS 멀티 시스템은, 전원 모듈를 반드시 소프트웨어를 통해 켜야 한다.

## 3.4 펑션 블럭 구동과 상태 평가

이 장은 PLCopen-호환 펑션 블럭의 구동을 요약적으로 언급한다. 우리는 어떻게 펑션 블럭을 구동하는지 살펴보는 것 뿐만 아니라 구동을 모니터링하기 위한 옵션을 이용할 수 있다. 모든 펑션 블럭은 단일화된 오퍼레이팅 파라미터와 단일화된 리턴 상태 정보를 사용하여 접근할 수 있다. 이는 어플리케이션을 단순화 하고 프로그래밍 중 명료성을 추가한다.

### 타이밍 다이어그램

Automation Studio 도움말에서, 펑션 블럭 운영을 위한 입력 상태가 어떻게 작동하는지 타이밍 다이어그램의 도움으로 설명된다.

이 예시는 “Power“ 펑션은 컴포넌트들이 활성화 되어있을 때만 이용가능함을 보여준다. 컴포넌트를 “Enable“ 상태로 설정하는 것이 필수이다. 제어기가 켜지고 구동이 활성화 되어있을 때 “Enable“이 비활성화이면, 구동이 중단되고 제어기 스위치가 꺼질 것이다. 제어기는 컴포넌트가 재활성화되고 “POWER“ 입력에 상승엣지가 발생했을 때 켜진다.

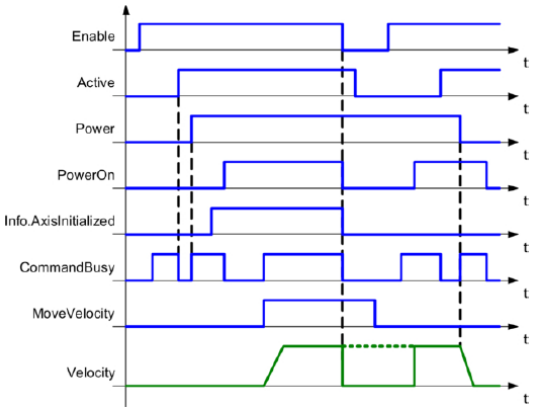



**그림 16 타이밍 다이어그램: “Enable“ 입력 에 따른 다른 명령과 상태 출력에 주는 효과**

Application layer – mapp technology ₩ Components ₩ Mechatronics ₩ MpAxis – individual and muti-axis controllers ₩ Function blocks ₩ Timing diagrams

### 상태 정보

에러 이벤트에서, “Error“ 출력값은 TRUE 가 된다. “StatusID“ 출력은 Automation Studio 도움말에서 검색할 수 있는 수치화된 정보를 포함한다. 현재 상태에 대한 추가 정보는 출력 구조 “Info“에서 이용가능하도록 만들어져 있다. 이것은 브라우저 내 WebXs 를 사용시 보여질 수 있다.

아래의 이미지는 WebXs 에서 MpAxisBasic 컴포넌트와 함께 확장된 info 구조를 보여준다. “Error“ 출력= TRUE 이고 "StatusID" 출력에 변수가 주어진다. 그 info 구조는 "MOVE_VELOCITY" 명령을 실행할 때 발생한 PLCopen 에러를 가리킨다. 그 에러번호는 29206 이다. Automation studio 도움말에서 에러번호를 검색하면, 설명을 찾을 수 있다. 에러 설명을 참조하면, 제어기가 켜지지 않은 상태에서 구동을 시작을 시도하였기 때문에 발생된 에러이다.

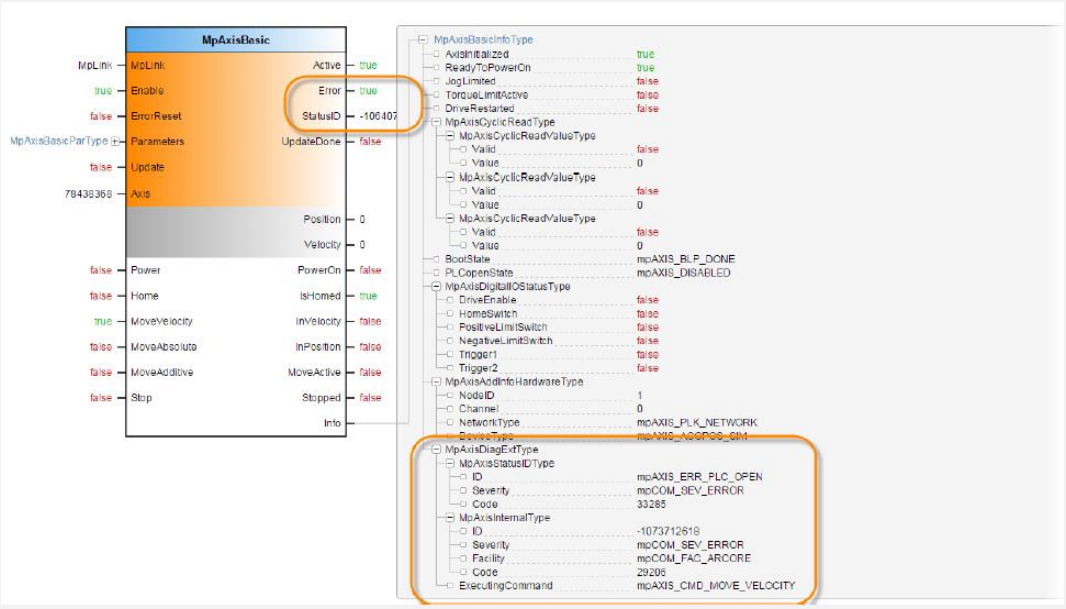



그림 17 포괄적인 info 구조 – WebXs 내 대표: PLCopen 에러 29206 – “제어기 꺼짐“

또한 Automation Studio 로거 “$mapp”에 입력이 발생된다. 추가 정보는 에러 원인을 감지를 위한 에러 번호을 읽을 수 있다.

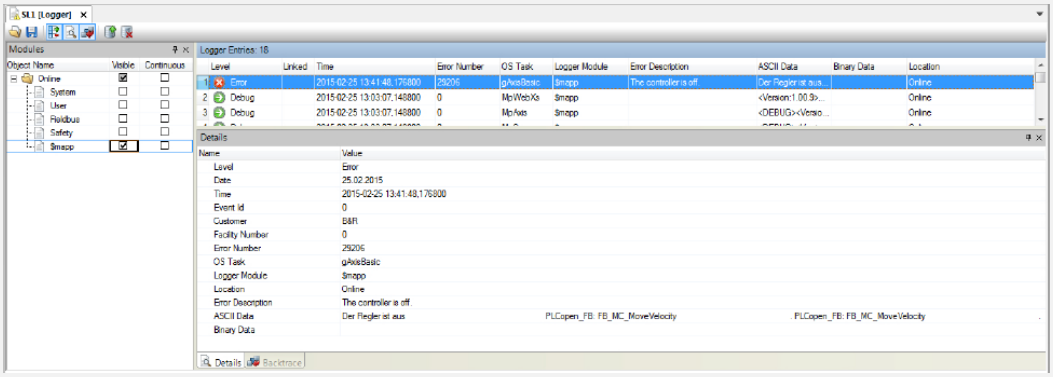



그림 18 “$mapp” 로거 파일: PLCopen 에러 29206 – “The controller is off“

에러 원인은 수정되고 에러는 "ErrorReset" 입력으로 인식된다. 제어기 스위치를 “POWER” 입력에 상승 엣지가 있을 때 다시 켜진다.

**예제: 와치 윈도우(Watch window)를 사용한 기본 구동 실행**

기본 구동은 MpAxisBasic 컴포넌트를 사용하여 수행되야 한다. 첫번째 단계로, 필수 컴포넌트는 이미 구성하고, 호출되었다. 이제 BOOL 타입 명령 변수를 선언하고 MpAxisBasic 입력과 연결하라. 각 명령은 와치 윈도우(Watch window)를 통하여 첫번째로 실행된다. 아래 명령들을 반드시 사용하여야 한다:

- "cmdPower" 제어기 스위치를 켬
- "cmdHome" 축 홈잉[4]을 위한
- "cmdErrorReset" 에러 리셋
- "cmdMoveVelocity" 스피드 설정 값을 사용하여 축 이동
- "cmdMoveAdditive" 추가 구동 수행
- "cmdStop" 활성화된 구동 정지

1) 명령 변수 선언
2) MpAxisBasic 입력과 명령 변수 연결
3) 기본 파라미터를 사용하여 스피드(1000 units/second)와 거리(1000 units) 구성
4) 제어기 전원을 켜고("cmdPower"), “PowerOn” 출력 대기
5) 축을 홈잉하고("cmdHome"), "isHomed" 출력 대기
6) 구동을 수행하고("cmdMoveVelocity"), "InVelocity" 출력 관찰
7) MpAxisBasic 출력과 상태 구조 확인

## 3.5 드라이브 상태 개요

드라이브 구동을 위한 PLCopen 표준에서 정의 된 상태가 결정된다. 그들은 위치 제어 어플리케이션의 상태 개요를 제공할 뿐만 아니라 에러 상황 진행 상태를 전달한다.

| 상태 | 설명 |
|---|---|
| Disabled | 드라이브 제어기가 꺼진다. |
| Standstill | 드라이브는 현재 동작이 실행되지 않고 위치결정을 위한 명령 대기 중이다. |
| Homing | 드라이브는 홈잉 절차를 수행 중이다. |
| Errorstop | 드라이브는 에러 발생 후 정지 상태이다. (= error stop) |
| Stopping | 드라이브는 활성화된 움직임을 멈추고 있다. |
| Discrete motion | 드라이브는 지정 위치로 동작 중이다. 그 동작은 지정된 목표를 가진다. |
| Continuous motion | 드라이브는 타켓 위치 없이 움직이고 있다. 그 동작은 지정된 목표가 없다. |
| Synchronized motion | 드라이브는 다른 드라이브와 동기되었다. |

**표 1 PLCopen 모션 제어 상태 다이어그램 개요**

[4] 예제에서 "mpAXIS_HOME_MODE_DIRECT"를 사용한다.

위치 결정 어플리케이션에서 가장 중요한 상태는 PLCopen 상태 다이어그램에 포함된다. 이들은 위치 결정 시퀀스를 코디네이팅하는데 사용된다. 현재 축에 PLCopen 상태는 ACP10_MC 라이브러리로부터 MpAxisBasic 컴포넌트 info 구조나 MC_ReadStatus 펑션 블럭을 이용하여 읽는다.


**그림 19** PLCopen 모션 제어 다이어그램의 상태

가상 축은 드라이브 제어기를 가지고 있지 않다. 실제 축과 달리, 가상 축은 Disabled 상태가 없다. NC 오프젝트 "가상 축"은 대기 상태(Standstill)에서 시작하고 활성화를 위한 MC_Power 펑션 블럭이 필요하지 않다.

Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC ₩ Concept ₩ State diagram

이 상태 사이에 전환은 특정 명령을 호출하면서 초기화된다.

아래는 시퀀스 예시이다.

축이 대기 상태(Standstill)에 있다고 가정해보자. 홈잉(Homing) 절차를 성공적으로 수행하자마자 "MoveAbsolute" 명령은 동작을 시작하는데 사용될 수 있다.

타겟 위치에 도달한 후, 드라이브는 "Initial state"(초기 상태)를 반환 한다. 만약 드라이브 에러가 위치 결정 액션 중에 발생한다면 축은 에러 상태로 들어간다("Error" 입력). 드라이브 에러가 고쳐진 후 "ErrorReset" 기능을 사용하여 인지될 수 있다.

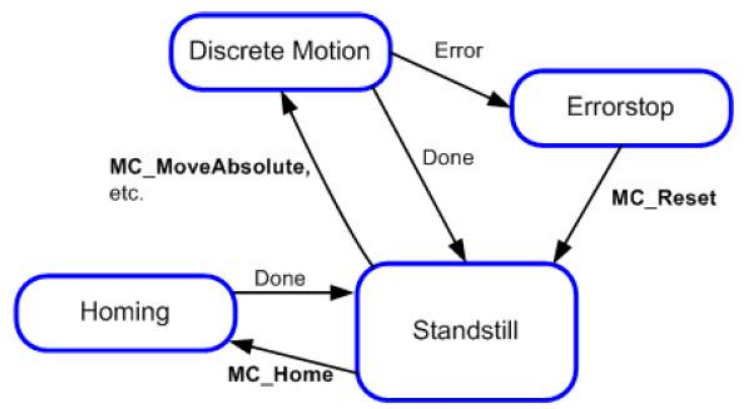



**그림 20 절대위치 구동 수행을 위한 시퀀스 상태**

**예제: PLCopen 상태 읽어오기**
현재 PLCopen 상태는 MpAxisBasic 의 info 구조를 사용하여 알 수 있다. Info 구조는 와치 윈도우(watch window)나 WebXs 에 표시될 수 있다.

제어기 전원 켜고 홈잉(homing) 과정을 수행하라. 매 행동 후, 현재 PLCopen 상태를 체크하라.

1) MpWebXs 라이브라리 삽입
2) 프로젝트 전송
3) 와치 윈도우에 info 구조와 WebXs 도 확인

**예제: 드라이브 에러 강제, 드라이브 에러 삭제**
기본 구동 수행 후 제어기 전원을 끄거나 레그 에러를 초과하게 시도해 보아라. 부가적으로 StatusID 을 관찰하고 Automation Studio 도움말에서 그 상태를 진단하라. 입력"cmdErrorReset"을 통하여 얻어진 에러를 인지하라.

1) 드라이브 에러 유발
2) 출력 “Error“와 "StatusID" 확인
3) Info 구조 확인
4) "ErrorReset" 입력을 통한 에러 인지
5) "Error", "StatusID"와 info 구조 확인

**예제: 자동으로 제어기 스위치 켜기와 직접 홈잉 절차 수행**
현존하는 프로그램은 반드시 어느 정도 자동화되어 있다. MpAxisBasic 컴포넌트는 제어기를 켠 즉시 활성화가 가능하다. 또한, 제어기는 컴포넌트가 성공적으로 활성화 된 이후 자동으로 켜진다. 그리고 나서 홈잉이 즉시 수행될 것이다. 추가적인 명령은 기본 축 구성 상태에서 수행 될 수 있음을 인지하라. 에러 이벤트에서, 인지가 끝나면 모든 명령은 컨트롤러가 켜지기 위해 재시작되고  구동 시작을 위한 모든 명령이 비활성화 된다.

1) 기계 상태와 프로젝트 설정이 필요한 단계[5]
2) "Info.ReadyToPowerOn" output is = TRUE 일때 "cmdPower" 설정
3) ""PowerOn" output is = TRUE 일때 "cmdHome" 설정
4) "IsHomed" output is = TRUE 일때"cmdHome" 리셋
5) "Error" output is = TRUE 일때 모든 명령 리셋

드라이브는 현재 기본 구동 수행을 위한 준비가 되어 있다. 프로그램은 "cmdErrorReset"이 성공적으로 수행된 이후 폐루프 컨트롤러가 자동적으로 켜지고 홈으로 다시 가도록 광범위하게 확장되어 있다.

[5] 기계 상태를 설정할 때 상위 레벨 프로그래밍 언어 사용을 추천한다. 레더 다이어그램 같은 그래픽 프로그래밍 언어는 MpAxisBasic 에 명령 승인을 위해 사용할 수 있다.

# 4 구성관리

Mapp 기술을 구현할 때 다른 구성 접근 유형이 있다. 드라이브 파라미터는 NC Init 모듈(NC init module)에 저장되고 NC 매핑 테이블(NC mapping table)을 통하여 할당됩니다. NC Init 모듈 대안으로, mapp 기술은 호환성 있는 드라이브 축 관리 구성을 제공한다. 이는 mapp 컴포넌트 초기화시 컨피규레이션 뷰(Confiugration view)에서 관리되고 드라이브에 쓰여진다.

## 4.1 축 초기화와 구성

### Mapp 컴포넌트를 사용한 초기화

모든 mapp 컴포넌트를 추가할시 표준 구성도 추가된다. NC Init 모듈은 MpAxisBasic 의 디폴트 구성으로 대체된다. 구성은 컨피규레이션 뷰에서 조정된다. "Axis configuration" 값을 "Enabled"로 변경함으로서, 모든 드라이브 파라미터는 mapp 컴포넌트 구성이 설정된다. 이 설정은 제어기가 시작된 후 로딩되고, MpAxis 컴포넌트 활성화되고 첫번째 명령이 수행된다. (예: “Power” 입력에 상승 엣지)

또한, mapp 구성 파일은 제어기 메모리에서 설치된다. 런타임 동안 MpAxisBasicConfig 와 함께 변경된다. (”드라이브 구성의 저장과 로딩” 참조)

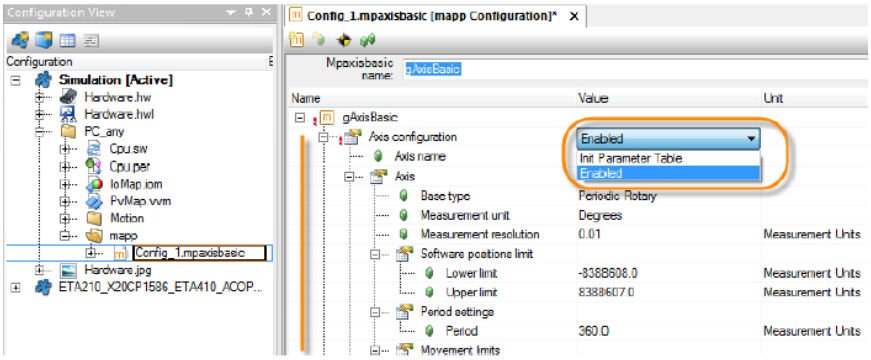



그림 21 컨피규레이션 뷰 내 mapp 컴포넌트의 구성 활성화

회전 테이블 어플리케이션 사례가 설정을 설명하기 위해 사용될 것이다.

**업무 정의:**

회전 운반체는 프로세싱을 위하여 물체를 다른 스테이션으로 운반해야 한다(특정한 각도 포지션은 최대 360° 회전 내에서).

위치 정확도는 반드시 0.1° 이내여야 한다. MpAxisBasic 과 함께"MoveAbsolute" 펑션 커멘드는 위치 접근에 사용된다.

이 절차를 조금 쉽게 하기 위하여, 위치에는 특정 각도가 할당된다.(소수점 한자리):

*BasicParameters.Distance := 135.0; (* Goal: 135° *)*

운반체는 서보모터를 이용한 기어(기어 비= 5:1)로 운반된다.

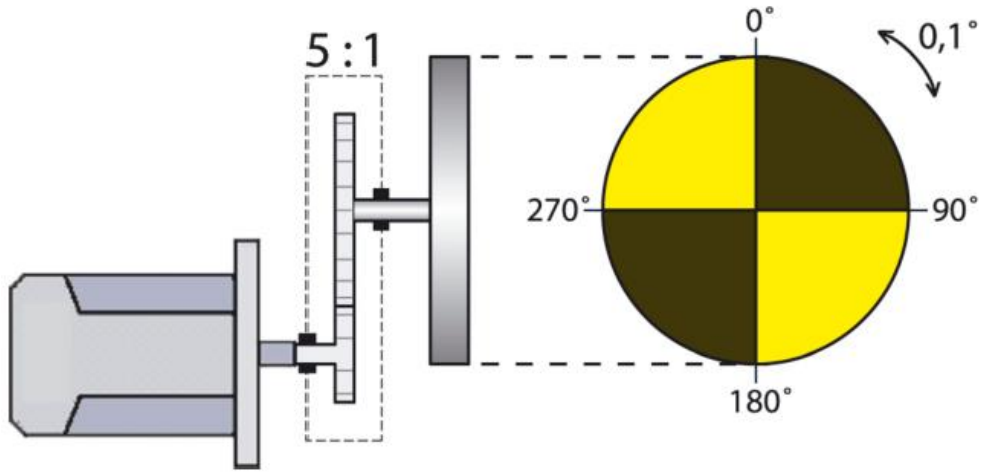



그림 22 기구 구조 스케치

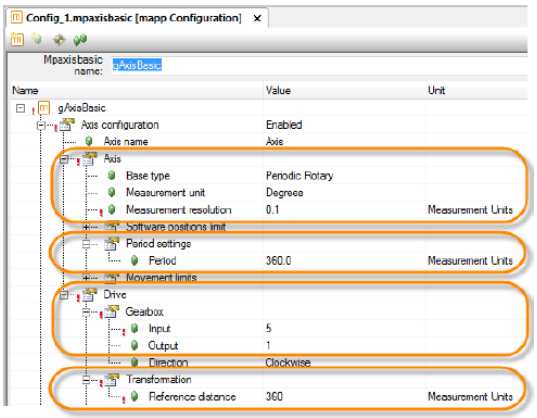



그림 23 회전 테이블 사례를 위한 mapp 구성

축 구성에서 측정 유닛, 측정 분해능, 기어비, 축 구간 뿐만 아니라 레퍼런스 거리는 기어 출력 대비 회전 수가 명시되어야 한다. 다음 구성은 예제를 위한 설명이다.

이 구성을 시작할 때 결과:
각도는 측정 유닛으로 사용된다. 정밀도는 0.1°이다. 축 구간(axis period)은 360°로 동일하다. 기어비는 5:1 로 정해진다. 기어 출력 내 회전은 회전 테이블 위의 회전과 동일하다. 레퍼런스 거리가 360°인 이유이다.

Application layer – mapp technology ₩ Components ₩ Mechatronics ₩ MpAxis – indi-vidual and muti-axis controllers ₩ Technical information ₩

- ACOPOs axis initialization and configuration
- Units of measurement and axis scalling

**NC Init 모듈과 NC mapping 테이블을 통한 초기화**
제어기가 시작될 때 NC Init 모듈은 NC 매니저가 축 레퍼런싱 초기화를 위하여 설정된 기본 축 파라미터를 사용한다.
이 구성은 MpAxisBasic 컴포넌트로부터 수행되며 자동으로 mapp 구성 파일에 저장된다. 변경은 MpAxisBasicConfig 컴포넌트를 통하여 런타임중 활성화된다.

추가 정보: "포지션 행동 설정과 스케일링" 참조

Application layer – mapp technology
- Components ₩ Mechatronics ₩ MpAxis – individual and muti-axis controllers ₩ Technical information ₩ ACOPOs axis initialization and configuration
- Concept ₩ Component configuration

Motion ₩ Project development ₩ Motion control ₩ Configuration modules ₩ NC Init modules
- NC Init module
- NC mapping tables

## 4.2 드라이브 구성 저장과 로딩

MpAxisBasicConfig 컴포넌트 사용시, 구성 파일 형태로 플레쉬 메모리에 저잔된 드라이브 구성은 어플리케이션에서 저장하고 로딩할 수 있다. 데이터 구조는 로지컬 뷰에 MpAxis 드라이버 구성의 구조에 식별자 컴포넌트와 연계되어 있다. MpAxisBaisc 을 위한 MpAxisBasicConfig 는 동일한 MpLink 를 사용한다.
MpAxisBaisc 컴포넌트를 활성화 할 때, 첫 명령어 (예: "Power"입력) 입력 후 구성 파일이 드라이브로 전송된다.

NC Init 모듈이 NC Test 윈도우에서 저장되고 제어기로 전송된다면, 구성 파라미터는 자동적으로 제어기에 mapp 구성으로 전송된다. MpAxisBasicConfig.Load 명령으로 구성이 로드되고, 드라이브 어플리케이션에 추가 프로세싱을 위한 변경된 파라미터를 이용이 가능하다.

Application layer – mapp technology / Components / Mechatronics / MpAxis – individual and multi-axis controllers / Technical information / ACOPOS axis initialization and configuration

**예제: 드라이브 구성 읽기**

예제 목적은 전체 드라이브 구성을 로드하는 것이다. "cmdLoadConfiguration" 명령을 사용하여 실행된다. 위치 제어기의 비례 게인(the proportional gain of the position controller)은 검증을 위해 결정되어야 한다. 드라이브 구성은 MpAxisBasicConfig 펑션 블럭을 사용하여 수행 될 수 있다. 축 연결은 MpLink 기본 축 구성을 사용한다.

1) MpAxisBasicConfig 추가
2) 구성 구조 선언
3) "cmdLoadConfiguration" 사용함으로서 드라이브 구성 로드

### 예제: 스핀들 드라이브를 위한 엔코더 인터페이스와 축 팩터 구성

그리퍼의 포지셔닝은 스핀들 드라이브를 통해 발생한다. 각 모터 회전을 위하여, 0.5mm 의 피드 – 포워드가 있다. 포지셔닝 정밀도는 정확히 1mm 이어야 한다. 설정 값은 cm [6]로 입력된다. 전체 이송 경로 길이는 1m 이다.

맵 구성 내 측정 단위, 측정 정밀도, 기어 비, 그리고 레퍼런스 거리를 위하여 설정하라.

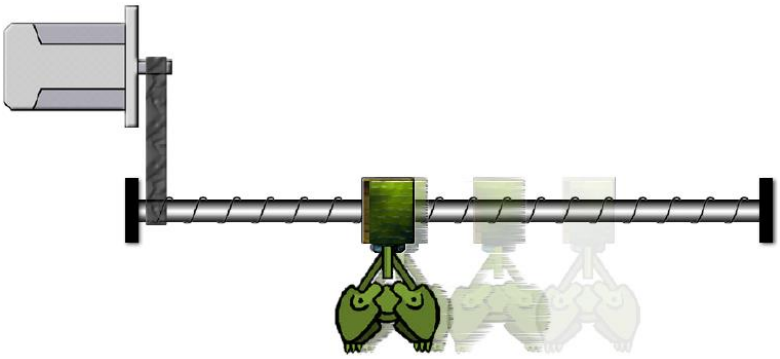


**그림 24 스핀들 드라이브의 스케메틱**

1) 측정 단위를 위한 적절한 구성 결정
2) 설정 값이 cm 로 입력 될 수 있도록  정밀도 결정
3) 구성 구조를 이용한 필요한 파라미터 전송
4) MpAxisBasic 컴포넌트 삽입
5) 구성이 드라이브로 전송될 수 있도록 MpAxisBasic 컴포넌트 활성화
6) 결과 확인

최대 속도, 레그 에러, 가속도, 소프트웨어 리밋은 반드시 고려되어야 한다. 또한 축 파라미터 유닛을 변경할 때 적절한 비율로 변경되어야 한다 (부하의 단위).

[6] 0.1cm 또는 2.4cm 움직임 거리와 같은 항목들이 허용된다

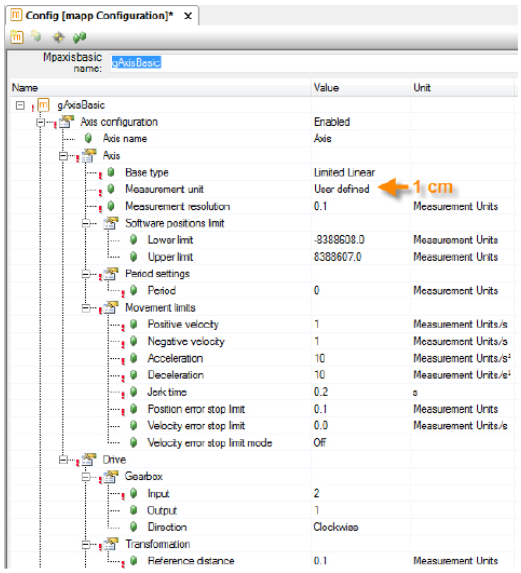



**그림 25 스핀들 드라이브를 위한 mapp 구성**

다음 구성은 태스크 정의의 결과이다:

1cm 는 사용자가 정의한 유닛(user defined)으로서 사용된다. 측정 분해능(measurement resolution)은 0.1cm 이다.

가/감속 움직임 파라미터는 또한 1cm 메인 유닛을 참조한다. 예를 들어 1cm/s 속도 구동 명령이 할당되었으면, 초당 20 모터 회전은 반드시 수행되어야 한다.

기어 비율은 1:2. 0.1 cm 이동속도는 기어 출력 회전과 함께 발생한다(레퍼런스 거리).

구성이 MpAxisBasicConfig 펑션 블럭을 통하여 수행되었다면, 파라미터는 반드시 다음 구성 구조에 할당되어야 한다.

```
// axis configuration (logical, PLCopen units)
AxisConfiguration.Axis.BaseType := mpAXIS_LIMITED_LINEAR;
AxisConfiguration.Axis.MeasurementUnit := mpAXIS_UNIT_GENERIC; (*unit 1 cm*)
AxisConfiguration.Axis.MeasurementResolution := 0.1; (*min. Value = 0.1 cm*)
AxisConfiguration.Axis.PeriodSettings.Period := 0;
// drive configuration (mechanical)
AxisConfiguration.Drive.Gearbox.Input := 2;
AxisConfiguration.Drive.Gearbox.Output := 1;
(*Reference distance: 0.1 cm per load revolution*)
AxisConfiguration.Drive.Transformation.ReferenceDistance := 0.1;
// movement limits
AxisConfiguration.Axis.MovementLimits.VelocityPositive := 1; (*1 cm/sec*)
AxisConfiguration.Axis.MovementLimits.VelocityNegative := 1; (*1 cm/sec*)
AxisConfiguration.Axis.MovementLimits.Acceleration := 10; (*10 cm/sec² *)
AxisConfiguration.Axis.MovementLimits.Deceleration := 10; (*10 cm/sec² *)
```

다음 변수는 설정 결과이다.

| Entry – PLCopen unit movement distance | Number of motor revolutions |
|---|---|
| 0.2 cm | 4 revolutions |
| 2.6 cm | 52 revolutions |
| 90 cm | 1800 revolutions |

**표 2 움직임 거리의 입력값과 모터 회전수의 상관관계**

# 5 프로그래밍 팁

어플리케이션 프로그램은 드라이브 제어를 위한 자동 시퀀스를 만드는 방법이다. 이 과정에서 모든 펑션 블럭을 프로그램에서 불러내는 것이 중요하다. 에러 이벤트는 반드시 고려해야 하며, 필요하다면 적절하게 응답해야 한다.

어플리케이션 프로그램에서 구조화된 시퀀스를 위해, 기계 상태는 하이 레벨 언어를 사용 할 수 있다. 이미지기반 프로그래밍 언어에서, 스텝이 사용될 수 있다. 예를 들어서 제어 프로그램 플로우에 사용될 수 있다.

**그림 26 어플리케이션 프로그램 상세하기 살펴보기**

## 5.1 제어 구조 사용하기

펑션 블럭은 드라이브 기능 제어를 위한 입력을 제공한다. 프로세스는 명령 변수를 이용하여 프로그램에서 지정된 위치에서 시작될 수 있다. 상태 출력과 출력 파라미터는 관련있는 펑션의 현 상태 정보를 제공한다. 다음 정보가 제공된다:

- 실행이 성공적인가?
- 그렇지 않다면 에러가 발생하였는가?
- 프로세스 상태:
  - 축이 동작 중인가?
  - 타겟 위치에 축이 도달하였는가?
  - 홈잉 절차가 성공적인가?

정보는 드리아브 어플리케이션에서 프로그램 시퀀스를 제어하기위해 사용될 수 있다. 프로그램은 에러 발생 유무에 따라 다르게 반응해야 한다.

이 제어 구조는 특히 기계 스텝 변환에 관한 펑션 프로세스 타입을 관리하는데 매우 적합한 제어구조이다 (state machine).[7]

이 구조는 스텝 인덱스 사용으로 결정될 수 있는 각 스텝 시퀀스 실행을 허용한다.

다이어그램은 이러한 제어 구조를 적용한 가능성 있는 디자인을 보여준다.

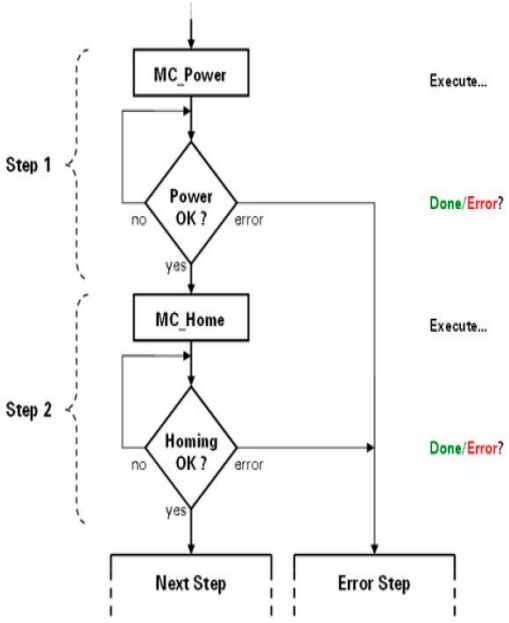



**그림 27 제어 구조 및 구조화 프로그램 예시**

[7]모든 하이 레벨의 프로그래밍 언어는 제어 구조를 갖는다. 이것은 state machine 구현을 가능하게 하거나 스텝 전환 메커니즘을 가능하게 한다. 그래픽 프로그래밍 언어에서, 펑션은 몇몇 디지털 시그널의 로지컬 링크를 통하여 가능하다.

필요한 명령어( 제어기 켜기, 홈잉 등)는 각 제어 구조 스텝에서 실행될 수 있다. Error, Done, Error-ID 같은 상태 파라미터를 통해 어플리케이션이 지속되는 스텝을 결정할 수 있다.

이는 어플리케이션에 깔끔한 디자인을 제공하고 향후 확장성을 개방시켜준다.

## 5.2 어플리케이션 프로그램에서 에러 표시기

Mapp 컴포넌트에 문제 진단을 위한 몇 가지 기본 인터페이스가 있다 ( “mapp 기술 컴포넌트를 위한 진단 옵션” 참조).

에러 핸들링은 에러, StatusID, 정보 출력을 사용하여 드라이브 어플리케이션용 프로그램을 구현 할 수 있다. 에러 번호에 대한 상세한 설명은 Automation Studio 도움말에서 찾을 수 있다.

**그림 28 어플리케이션 프로그램에서 에러 평가**

Application layer - mapp technology ₩ Concept ₩ Component design ₩ Inputs and outputs

### 로거(Logger)

에러를 발생시키는 구체적 데이터는 "$mapp" 로거 파일에 자동으로 입력된다. 로거 파일은 Automation Studio 에서 열 수 있고 System Diagnostics Manager 를 통해 PC 에 저장된다. 대안으로, 로거 파일은 "AsEventLog" 라이브러리를 사용하여 읽을 수 있다.

Mapp 컴포넌트 MpComLoggerUI 는 화면작화 어플리케이션에서 이벤트로 표시 되도록 돕는다. mapp 컴포넌트, 에러 번호 , 메세지 타입에 대한 필터를 제공한다.

비주얼 컴포넌트를 위한 HTML 제어를 사용함으로서, SDM 페이지를 포함 할 수 있다. 로거, 기계 화면작화 어플리케이션에 직접 연결된 것을 표시한다.

Application layer - mapp technology ₩ Diagnostics

Application layer - mapp technology ₩ Components ₩ Infrastructure ₩ MpCom ₩ Function blocks ₩ MpComLoggerUI

Diagnostics and service ₩ Diagnostic tool

- Logger
- System Diagnostics Manager

Visualization ₩ Visual Components VC4 ₩ Control reference ₩ HTML view

Programming ₩ Libraries ₩ Configuration, system information, runtime control

- AsArLog
- ArEventLog

### MpAlarm 컴포넌트과 화면작화 컴포넌트 알람 시스템

MpAlarm 컴포넌트를 사용하여, 미리 정의된 MpAxis 컴포넌트 알람은 MpAlarmUI 펑션 블럭을 사용하여 비주얼 컴포넌트 알람 시스템에 전달될 수 있다. 그 항목은 그룹, 시간, 우선순위로 필터되어 표시된다. 표시된 문자를 위하여 언어 변경이 사용될 수 있다 ( “TM640 – Alarm System, Trends and Diagnostics” 참조)

Application layer – mapp technology ₩ Components ₩ Infrastructure ₩ MpAlarm – Support for alarm management

Visualization ₩ Visual Components VC4 ₩ Shared Resources ₩ Alarm

### 예제: 이벤트 에러에서 명령 변수 재설정

이벤트 에러에서, 어플리케이션 프로그램의 모든 명령 변수는 자동으로 리셋되어야 한다. 에러가 발생할 때, “Error”와 “StatusID”를 통하여 표시된다.

1) “Error” 출력에 상승 엣지 평가
2) 이벤트 에러 발생 후, "cmdPower", "cmdHome", "cmdMoveAdditive" 명령 자동 리셋

### 예제: 로거와 함께 축 에러 읽고 축 에러 인지

예를 들어, 제어기를 끈 상태에서 구동 시작을 시도 한다면, 에러가 발생할 것이다. 에러 상태는 MpAxisBasic 펑션 블럭 출력에 표시된다. 에러 원인의 상세 설명은 "$mapp" 로그파일에 저장된다. 진단이 끝난 뒤, 에러는 “ErrorReset” 입력을 통하여 인지되어야 한다.

1) 제어기 전원 내림과 움직임 시작
2) 로거 오픈
3) Automation Studio 도움말 내 에러 번호 설명 확인[8]
4) 에러 원인과 드라이브 에러 인지 분석

“ErrorReset“ 펑션 블럭 입력은 모든 에러와 Disabled 상태인 축 입력을 리셋한다. 인지(acknowledge)가 끝난 후, “Power“ 입력에 상승 엣지를 다시 공급하여 제어기 전원 켜는 것이 가능하다.

[8]에러 설명과 도움말 페이지는 로거에서 에러 메시지를 선택하고 F1 키를 누름으로서 열린다.

### 예제: 성공적으로 에러 리셋후 제어기 전원 자동으로 켜기

모든 에러를 성공적으로 리셋한 후, 드라이브 어플리케이션을 통하여 제어기가 자동으로 켜져야 한다.

1) “Error” 출력의 하강 엣지 평가
2) "cmdPower" 명령과 함께 제어기 전원 다시 켜기
3) 필요시, 홈잉 절차 수행

### 예제: MpComLoggerUI 컴포넌트와 축 에러 읽기

로거에 입력되는 mapp 이벤트를 평가하기 위하여 MpComLoggerUI 컴포넌트를 사용하라.

1)MpComLoggerUI 컴포넌트 추가
2)MpAxisBasic 컴포넌트에 MpLink 연결
3)mpCOM_CONFIG_SCOPE_COMPONENT 로 스코프 파라미터(scope parameter) 초기화
4)UIConnect 구조 선언및 연결
5)변경사항 전송
6)에러를 발생시키고 UIConnect 구조 내 항목 분석
예를 들어, PLCopen 에러를 발생시키려면, 제어기가 꺼진 상태에서 동작 시작 명령어를 발생시켜라.

Application layer - mapp technology ₩ Components ₩ Infrastructure ₩ MpCom ₩ Function blocks₩ MpComLoggerUI

# 6 MpAxisBasic 추가 기능

MpAxisBasic 컴포넌트는 더 많은 기능을 제공하고, 드라이브 준비와 기본 구동 수행을 위하여 사용된다. 게다가 이미 사용된 펑션, 토크 리밋, 조그 모드, 오토 튜닝 뿐만 아니라 주기적 드라이브 데이터 읽기에 활용된다.

Programming ₩ Examples ₩ Motion ₩ Motion control ₩ Muti-axis operation

**예제: 주기적 드라이브 데이터 읽기**
주기적 드라이브 데이터 읽기는 MpAxis 컴포넌트의 구성 구조를 사용하여 가능하다. 드라이브 렉 에러(lag error)를 정기적으로 읽도록 컴포넌트를 구성하라.

1) 주기적 렉 에러(lag error) 읽기를 위한 항목 활성화
2) 변경 사항을 제어기로 전송
3) 드라이브 켜기와 움직임 실행
4) 렉 에러 확인과 필요시 Automation Studio trace 를 이용하여 기록하기

**예제: 토크 리밋 구현**
드라이브 제어에서 토크리밋 특성화는 파라미터 구조 내 MpAxisBasic 입력을 통하여 직접 구현 가능한다. 예를 들어서 토크 리밋을 0.5Nm 로 설정하라. 최대 토크가 20%를 초과시 에러가 반드시 출력 될 것이다.

1) 토크리밋 전송
2) 최대 값 모니터링을 위하여 임계 값 전송
3) 드라이브를 켜고, 홈잉 절차 수행
4) 구동 시작
5) "TorqueLimit" 입력을 활성화시켜서 토크 활성화
6) 토크가 한계치이고 이벤트 에러 발생시 드라이브 제어기 행동을 구성하라

펑션 블럭의 선택적 파라미터는 Automation Studio 도움말을 통하여 표시된다. 체크박스 위에 펑션 블럭 예시의 이미지를 활성화 함으로서 나타난다.

**예제: 오토튜닝 수행과 구성 저장**
MpAxis 컴포넌트는 드라이브 제어기에서 자동으로 파라미터 식별 기능을 제공한다. 속도 제어기와 위치 제어기를 위한 제어기 파라미터를 식별하기 위하여, 오토 튜닝 펑션을 사용하라. 축이 초기 파라미터 식별 동안 구동을 허락하지 않음을 고려하여라. 피드-포워드 제어를 위한 파라미터 결정과 같은 추가 식별 절차를 위한 구동 수행시 필수이다.

1) 오토튜닝 구성을 위한 모드
2) 튜닝 절차를 위한 최대 전류, 최대 스피드와 거리 선정
3) 변경 사항을 제어기로 전송
4) "MpAxisBasic.Autotune" 입력을 사용한 튜닝 절차 시작
5) 결과 확인

당신이 드라이브 구성에서 계산된 파라미터를 저장하기 위하여 드라이브 어플리케이션을 확장하라.

# 7 PLCopen 모션 라이브러리(ACP10_MC)

PLCopen 모션 라이브러리는 Automation Studio 도움말에 목록이 있고 이 라이브러리 이름은 ACP10_MC 이다. B&R 드라이브 솔루션을 제어하기 위한 표준화된 PLCopen 펑션 블럭을 포함한다.

ACP10_MC 라이브러리는 어떤 시스템이든 사용될 수 있는 "위치 이동" 이나 "홈잉(Homing) 절차" 같은 기본 기능을 포함한다. 사용자에게 특정 분야의 표준 어플리케이션을 위한 완전히 획일적인 소프트웨어 인터페이스를 제공한다.

실제 B&R 드라이브 솔루션 기능 영역은 표준을 포함하여 이를 뛰어 넘는다.

예를 들면, 커플 링 축에 표준 펑션[9] 을 보완 가능하도록 기본 위치 결정 기능과 강력한 펑션 블럭으로 확장된다.


**그림 29 PLCopen Motion Control logo**

ACP10_MC 라이브러리는 기능의 장점을 동일한 방식으로 유지하면서 사용자에게 제공할 수 있도록 B&R 특유의 기능을 포함하며 확장되었다. 이 심화 기능은 PLCopen 표준을 준수하여 동작된다.

## 7.1 펑션 그룹

ACP10_MC 라이브러리는 상대적으로 많은 펑션 블럭을 포함하며, 그들의 사용과 기능에 따라서 대략 몇 개의 그룹으로 나뉜다,

**기본 기능:**

- 드라이브 준비
- 추가적이고 절대적인 기본 동작
- 조그 모드
- 드라이브 상태 결정
- 설정 및 실제 값 읽기
- 드라이브 에러 인식과 결정
- 위치 측정
- PLCopen 축 파라미터 관리

**구동, 드라이브 구성 및 데이터 변경:**

- 토크 제어
- 트레이스 펑션
- 주기적인 설정 값 생성
- 축 파라미터 관리

[9]다중 축과 커필링 펑션의 상세 오버뷰는 트레이닝 모듈 "TM441 - Motion Control: Multi-axis Functions"에서 제공된다

**다 축 제어:**

- 전기(electronic) 기어박스 생성
- 위상 및 오프셋 이동(Phase and offset shifting)
- 캠 프로파일을 이용한 드라이브 연결
- 캠 프로파일오토맷 설정 및 제어

**기술 기능:**

- 마크 제어 등록(Registration mark control)
- 유닛 컷팅(Cutting units)
- 캠 프로파일 오토맷(Cam Profile Automat)

Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC ₩ Function blocks

Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC ₩ Function blocks ₩ Overview of the supported function blocks

## 7.2 ACP10_MC 와 MpAxis 호환

MpAxis mapp 구성 펑션 블럭은 ACP10_MC 라이브러리의 펑션 블록을 기본으로 한다. 두 라이브러리는 서로 완전하게 호환 가능하다. 두 개 라이브러리에서, 축은 축 레퍼런스를 통하여 접근된다.

드라이브를 위한 파라미터와 명령은 명령이 활성화 되었을 때 전송된다. PLCopen 상태 다이어그램의 상태는 변경된다. ACP10_MC 라이브러리의 펑션 블럭을 사용하여 MpAxisBaisc 으로 프로그래밍 된 어플리케이션에 기능을 추가 할 수 있다.

예를 들어서, 기능 호환성에 따라서 MpAxisBasic 을 통해 제어기를 켜거나 홈잉을 수행하고 ACP10_MC 라이브러리 내 MC_MoveAdditv 펑션 블럭을 사용하여 기본 구동을 할 수 있다. MpAxisBaisc 컴포넌트 구조 info 를 통해 “Standstil”에서 “Discrete Motion”으로 넘어가는 변화를 볼 수 있다.

**예제: mapp 기술과 APC10_MC_library 결합**

드라이브 제어를 위한 mapp 컴포넌트는 PLCopen 표준을 기반으로 한다. 이러한 이유 때문에, 모든 mapp 컴포넌트는 ACP10_MC_library 내 펑션 블럭과 호환된다. MpAxisBasic 컴포넌트는 모든 관련된 펑션을 제공한다; ACP10_MC 라이브러리 내에 펑션 블럭을 사용하여 모션 어플리케이션을 확장이 가능하다. 드라이브 데이터는 축 레퍼런스를 통하여 두 가지 방법으로 접근할 수 있다.

이 예제에서, 드라이브는 MpAxisBasic 컴포넌트를 사용하여 준비하고,
구동은 ACP10_MC 라이브러리의 MC_MoveVelocity 펑션 블럭을 사용한다.

1) MC_MoveVelicity 펑션 블럭을 어플리케이션에 추가
2) MpAxisBasic 와 함께 제어기 전원을 켜고 홈잉 절차 수행
3) MC_MoveVelicity 를 사용하여 정방향으로 구동
4) MpAxisBasic 출력을 사용하여 상태 변화 확인

Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC ₩ Function blocks ₩ Basic movements

## 7.3 포지션 행동 설정과 스케일링

Mapp 구성 대신에 축을 초기화를 위하여 NC init 모듈(NC init module)이 사용되었을 때, 위치 값을 조절하기 위해 NC mapping 테이블에 추가적인 설정이 가능하다. 이 설정과 함께, mapp 구성 도움과 함께, 주기적 위치 행동(Periodic position behavior)과 포지션 스케일링을 얻을 수 있다.

주기적 위치 행동(Periodic position behavior)은 턴테이블과 같은 어플리케이션에 요구된다.

### 엔코더 인터페이스의 구성

모터 회전 유닛 스케일은 축을 위한 엔코더 인터페이스 부하 단위로 스케일 된다. 부하 단위는 축 파라미터 유닛으로서 언급된다. 값은 양의 정수를 사용한다.

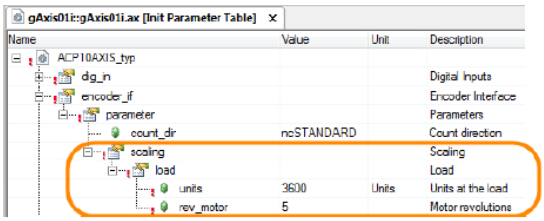



**그림 30 NC Init 모듈: 모터 회전 단위 비 설정**

### NC mapping 테이블 내 기간과 팩터 설정

모든 PLCopen-호환 펑션 블럭은 위치 사항을 위하여 REAL 데이터 타입으로 사용된다.

NC 맵잉 테이블(NC mapping table)에서 PLCopen_ModPos="<Period>,<Factor>" 입력을 통하여, 포지션 기간(position period)과 축 팩터(axis factor)를 위한 값은 포지션 값 조절하기 위해 미리 정의된다.

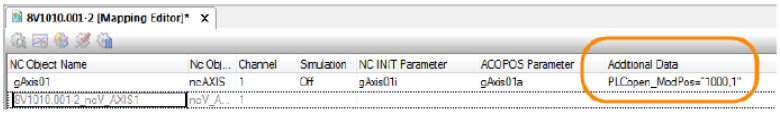



**그림 31 NC mapping 테이블의 추가 세팅**

예를 들어, 이미지에서 1000 축 파라미터 단위가 모터 회전에 할당되어 있다. 각 회전당 1000 축 파라미터 단위에 달한다. 어플리케이션 요구에 따라 회전이 나눠지는 것이 가능하다(엔코더 최대 해상도를 마음에 둔다.)
DINT 는 포지션 데이터 타입이다.

PLCopen 어플리케이션에서 속도와 가속도 값은 스케일링을 참조하시오.

NC Init 모듈 사용 외에도, mapp 기술은 구성 관리 확장을 제공한다. 어플리케이션에서 설정을 위한 펑션 블럭을 추가하는 방법 대신에 컨디규레이션 뷰에서 구성 설정을 추가하는 방법으로 대체될 수 있다.

추가적인 정보는 “구성관리”를 참조하라.

### 7.3.1 Axis factor

PLCopen 펑션 블럭은 축 위치결정을 위해 REAL 데이터 타입을 사용한다. 이 데이터 타입은 소수점 자리를 사용할 수 있으며, 간단하고 매우 흥미롭게 스케일링을 변환할 수 있다.

스케일링(scaling)은 변환이라고도 한다. 이미지로 표현된 아래 예제가 설명해준다.

특정 어플리케이션은 1 ㎛이하의 정밀 위치결정을 요구한다. 우리가 설정한 파라미터가 위치결정 유닛 당 1 ㎛거리로 설정 할 수 있다고 가정하자.

지금 우리가 PLCopen 펑션 블럭으로 위치결정을 밀리미터(㎜) 단위로 구체화 할 수 있다면 이득일 것이다. 그리고 이 팩터로 얻은 것들은 정확하다. 방정식은 아래와 같다.

$$PLCopenUnits = \frac{AxisparameterUnits}{Factor}$$

우리가 factor 를 1000 으로 설정할 수 있다면, 우리는 밀리미터(㎜)까지 스케일링 할 수 있을 것이다. PLCopen 펑션 블럭에 상응하는 1 값으로 거리를 움직인다면, 우리 축은 실제로는 1000 축 파라미터 유닛으로 움직일 것이다.

| <Factor> | PLCopen units [REAL] | Axis parameter units [DINT] | Movement distance |
|---|---|---|---|
| 1 | 1.0 | 1 | 1 ㎛ |
| 1000 | 1.0 | 1000 | 1 ㎜ |
| 1000 | 0.001 | 1 | 1 ㎛ |

**표 3 설정 factor 에 따른 축 유닛과 PLCopen 유닛 비교**

Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC ₩ Concept ₩ Implementation ₩ Axis factor

### 7.3.2 Axis period

연속적인 축 구동을 위해, 위치 값을 주기적으로 결정하는 것은 종종 중요하다. 주기에서 0 보다 큰 값이 사용되었다면, 위치 값은 이 입력에 따라 적용된다. 그것은 엔코더 파라미터에 있는 축 스케일링을 직접 나타낸다. 모든 PLCopen 펑션 블럭은 주기적인 위치와 함께 "작업"한다.

예를 들어, <Period> = 1000 이라면, 0 으로 리셋되고 999 까지 다시 증가하기 전에 positive 움직임을 하는 동안은 위치 값은 0 부터 999 까지 계속 증가할 것이다.

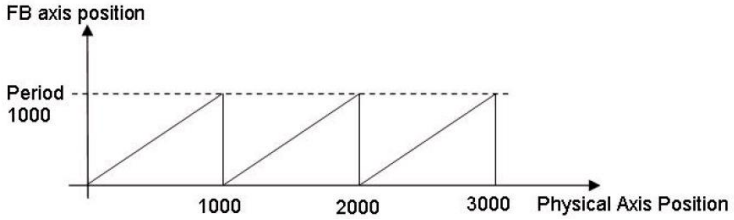



**그림 32 주기적인 축 위치: 축 기간 = 1000 PLCopen 유닛**

주기적 위치 변환은 <Period> = 0 이 입력되면 비활성화 될 수 있다. 주기적 운동이 요구되지 않더라도 특정 팩터는 여전히 사용될 것이다. 이 경우, 축 동작 범위는 ±8,3888,608 유닛(units, $\pm 2^{23}$)까지 제한된다.[10] 왜냐하면 이것은 REAL 데이터 타입에서 정확성을 잃지 않고 디스플레이 될 수 있는 가장 큰 값이기 때문이다.

Programming ₩ Libraries ₩ Motion libraries ₩ ACP10_MC ₩ Concept ₩ Implementation ₩ Axis period

구동이 기록될때, 축 파라미터 유니는 항상 NC Trace 에 기록된다. 속도와 가속도는 축 파라미터 유닛을 따른다.

**예제: 회전 축 구성**

간단한 예는 설정을 실행하고 가능성을 증명 할 것이다.

**테스트 정의:**
피벗팅 케리어(A pivoting carries)는 반드시 제품 프로세싱을 위해 다른 스테이션으로 구동해야한다. (specific angular positions 360° 회전 내)

위치 결정은 0.1° 이내의 정밀도이다. MC_MoveAbsolute 펑션 블럭은 위치 접근을 위해 사용될 것이다.

이 절차를 조금 더 쉽게 하기 위해 위치는 각도로 설정된다.(소수점 첫째 자리)

```
MoveAbsolute_0.Position := 135.0; (* goal: 135° *)
```

캐리어는 기어로 연결된 서보 모터로 움직인다. (기어비 = 5:1)

적합한 엔코더 값과 예정된 위치 설정 값을 찾아라.

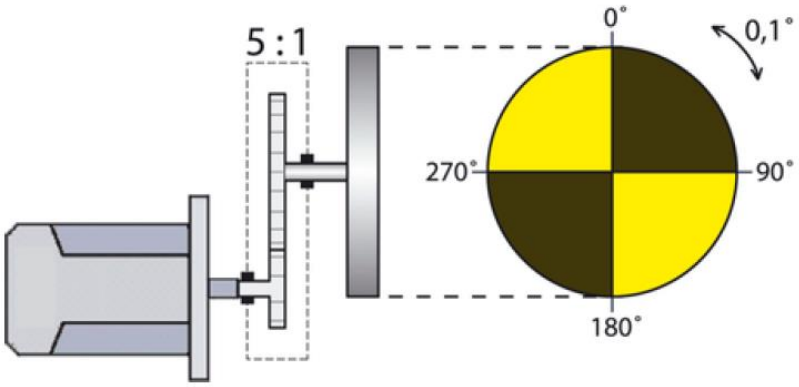



그림 33 기구 구조 스케치

[10]제한은 REAL 데이터 타입에 손실 없이 정확하게 디스플레이 할 수 있는 최대 숫자이기 때문에 제한이 있다.

1) NC Init 모듈을 위한 적절한 엔코더 설정 결정
2) NC mapping 테이블에 축 피리어드와 축 팩터를 위한 설정
3) 포지셔닝 어플리케이션에 "cmdMoveAbsolute" 추가
4) 회전내에서 다른 각도 구동 테스트

**가능한 해결책:**
먼저, 엔코더 인터페이스를 위한 기본 설정은 NC Init 모듈에서 이루어진다. 0.1 ° 단계에서 위치 확인을 위하여, 3600 유닛(units)은 회전 운반체의 회전을 위해 필요하다. 이 유닛들은 기어비 감소 때문에 모터 5 회전에 분배된다. 3600 유닛 구동이 실행되었다면, 모터는 정확히 5 바퀴를 수행하고 회전체는 정확히 360 도 회전한다. 기어는 그러므로 이미 포함되어 있다.

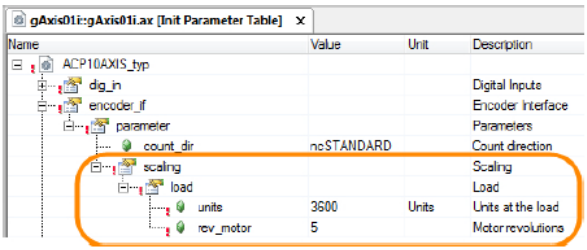



**그림 34 Init 파라미터 모듈에서 엔코더 인터페이스 설정**

포지셔닝 펑션을 위하여 원하는 스케일링을 얻기 위해, 축 피리어드와 축 팩터가 구성되어야 한다. 이를 위해. 축 팩터 10 과 축 피리어드 3600 이 설정되어야 한다(엔코더 인터페이스에 직접 설정)

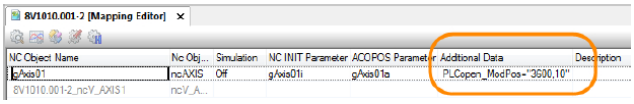



**그림 35 NC mapping 테이블 내의 기간 축 심화 세팅**

| Axis factor | PLCopen units [REAL] | Axis parameter units[DINT] | Movement distance Carrier | Angle of rotation Motor |
|---|---|---|---|---|
| 1 | 1.0 | 1 | 0.1° | 0.5° |
| 10 | 1.0 | 10 | 1° | 5° |
| 10 | 72.0 | 720 | 72° | 360° |
| 10 | 135.0 | 1350 | 135° | 675° |
| 10 | 360.0 | 3600 | 360° | 1800° |

**표 4 축 팩터 설정에 따른 PLCopen 유닛과 포지셔닝 유닛 비교**

예제에서 보이는 것처럼, 72 PLCopen units 구동은 NC Trace 에서 720 axis parameter units 위치 증가를 의미한다.

720 PLCopen units 이송 경로는 예제 구성에서 2 축 피리어드 이송 경로를 의미할 뿐만 아니라(회전 운반체의 2 회전에 상응한다) ,NC Test 에서 7200 axis parameter units 만큼 위치가 증가한다.

(0 의 위치를 이전 원점 복귀 절차를 기반으로 한) 현재 PLCopen 포지션은 0 이다. NC Trace 는 실제 포지션을 7200 값으로 보여준다. 이는 모터 10 회전에 상응한다.

# 8 요약

강력한 MpAxis 컴포넌트는 드라이브 펑션을 제어하기 위하여 이용가능하다. 이 펑션 블럭은 PLC 표준을 준수하고 효율적인 설계와 유용성으로 구분된다.

제어 어플리케이션의 통합은 사용자가 mapp 기술에 자신감을 갖자 마자 시작된다 (이것의 꽤 단순하다). 정교한 구성관리, Automation Studio 로거 윈도우의 에러 정보에 직접 통합될 뿐만 아니라 System Diagnostics Manager 구성요소 구조를 통하여 이용가능하다. 화면작화 어플리케이션에 드라이브 알람을 통합하는 것은 MpAlarm 컴포넌트로 활성화된다.


그림 36 mapp 기술은 펑션의 포괄적인 포트폴리오를 제공

드라이브 어플리케이션 강화는 펑션 블럭이 다른 모션 제어 라이브러리들과 호환성이 있기 때문에 가능하다. 예를 들어 ACP10_MC 라이브러리 내 펑션 블럭과 MT_LoadSim 라이브러리가 있다.

# 9 해결책

## 9.1 솔루션: 자동으로 제어기를 켜고 직접 홈잉 절차 수행

```
(*****************************************
* COPYRIGHT – Bernecker + Rainer
*****************************************
* Program: mp410_axis
* Created: Pctober 2014
*****************************************
* Implementation of program mp410_axis
*****************************************)

PROGRAM _INIT

        (* set basic parameters *)
        BasicParameters.Velocity  := 1000; (* 1000 units / second*)
        BasicParameters.Distance := 1000; (* 1000 units distance*)

END_PROGRAM


PROGRAM _CYCLIC

 CASE sStep OF
   enINIT:
       (* do nothing; sequence is started manually *)

   enSTART:

      MpAxisBasic_0.MpLink := ADR(gMpLink_AxisBasis_gAxis1);
      MpAxisBasic_0.Axis     := ADR(gAxis01);
      MpAxisBasic_0.Enable := 1;
      MpAxisBasic_0.MpLink := ADR(BasicParameters);

       (* is axis ready to be powered on? *)
       IF MpAxisBasic_0.Info.ReadyToPowerOn = TRUE THEN
         sStep:= enPOWER_ON;
       END_IF;


   enPOWER_ON:
      cmdPower := TRUE;

       (* axis is powered on? *)
       IF MpAxisBasic_0.PowerOn = TRUE THEN
         sStep := enHOME;
       END_IF;
```

```
enHOME:
    cmdHome:= TRUE;

    (* axis is homed? *)
    IF MpAxisBasic_0.IsHomed = TRUE THEN
      sStep      := enOPERATION;
      cmdHome:= FALSE;
    END_IF;


enOPERATION:

    (* commands basic movements *)
    cmdMoveVelocity;
    cmdMoveAdditivie;
    cmdStop;


    (*reset command cmdMoveAdditive when position is reached*)
    IF EDGEPOS(MpAxisBasic_0.InPosition) = TRUE THEN
      cmdMoveAdditivie:= FALSE;
    END_IF;

    (* update parameters*)
    cmdUpdate;

    (* axis in error step? *)
    IF MpAxisBasic_0. Error = TRUE THEN
      sStep      := enERROR;
    END_IF;


enERROR:
    (* power off *)
    cmdPower := FALSE;

    (* reset all commands *)
    cmdMoveVelocity  := FALSE;
    cmdMoveAdditivie := FALSE;
    cmdStop          := FALSE;
    cmdUpdate        := FALSE;

    (* implement error handling herereset all commands *)
    cmdErrorReset;
    IF MpAxisBasic_0.Error = FALSE THEN
      sStep:= enSTART;
      cmdErrorReset:= FALSE;
    END_IF;
```

```
  END_CASE

  (* execute commands *)
  MpAxisBasic_0.Power          := cmdPower;
  MpAxisBasic_0.Home           := cmdHome;
  MpAxisBasic_0.ErrorReset     := cmdErrorReset;
  MpAxisBasic_0.MoveVelocity := cmdMoveVelocity;
  MpAxisBasic_0.MoveAdditive := cmdMoveAdditivie;
  MpAxisBasic_0.Stop           := cmdStop;
  MpAxisBasic_0.Update         := cmdUpdate;

  (* call all mapp components *)
  MpAxisBasic_0();

  (* movement with MC_MoveVelocity so watch status changes of component MpAxisBasic*)
  MC_MoveVelocity_0.Axis             := ADR(gAxis01);
  MC_MoveVelocity_0.Execute          := cmdMoveVelocity_ACP10_MC;
  MC_MoveVelocity_0.Velocity         := 1000;
  MC_MoveVelocity_0.Acceleration := 2000;
  MC_MoveVelocity_0.Deceleration := 2000;
  MC_MoveVelocity_0.Direction        := mcPOSITIVE_DIR;

  (* call all ACP10_MC function blocks *)
  MC_MoveVelocity_0();

END_PROGRAM

PROGRAM _EXIT

  (* disable all mapp components*)
   MpAxisBasic_0.Enable := FALSE;
   MpAxisBasic_0();

END_PROGRAM
```