Einrichten und steuern eines Motors im AS von B&R mit PLCopen Funktionsbausteinen

Unterrichtsprojekt

im Schwerpunkt
Energie- und Automatisierungstechnik

vorgelegt von

Thorben Oltmann

am 26.05 2017
an der Technikerschule Hannover

Einleitung

In dieser Ausarbeitung ist eine Schritt für Schritt Anleitung gegeben für die Inbetriebnahme eines Motors mit Hilfe des Automation Studios von B&R. Die Parametrierung und Steuerung ist mittels PLCopen Funktionsbausteinen im Strukturierten Text realisiert. Erklärungen und Grafiken zu den PLCopen Funktionsbausteinen wurden der B&R Hilfe entnommen.

Inhaltsverzeichnis

1 Ziele

Folgende Ziele sollen in dieser Dokumentation behandelt werden:

[1] Ein neues Projekt anlegen.

[2] Erstellen einer Hardwarekonfiguration.

[3] Erläuterung der notwendigen PLCopen Funktionsbausteinen für die Inbetriebnahme eines Motors.

[4] Erstellen von Strukturtypen und definieren von Variablen.

[5] Erstellen eines Programms im Strukturierten Text.

[6] Erstellen einer Projektsimulation.

[7] Steuern von Variablen im AS Watch/ Monitormodus

2 Software

Das Projekt wurde mit folgenden Versionsnummern erstellt:

Software		Versionsnummer
Automation Studio	V	4.2.8.054
Automation Runtime	V	J4.25
Visual Components	V	4.26.3
ACP10 Motion Software	V	3.16.2

3 Projektvorbereitung

Das Festlegen eines Programmier- und Einrückungsstils unterstützt die Verständlichkeit und Wartung für das Projekt und steigert die Softwarequalität. Folgende Regeln können, müssen aber nicht angewendet werden.

Programmiersprache:	Strukturierter Text (ST)	
Variablendeklaration:		
Indikator 1 (Datentyp)	BOOL	x
	BYTE	b
	WORD	w
	Doppelword	dw
	Integer (vorzeichenlos)	ui
	INT	i
	REAL	rl
	Short-Integer	si
	Short-Integer (vorzeichenlos)	usi
	Double Integer (vorzeichenlos)	udi
	String	str
	Typenstruktur	t
	Funktionsbaustein	fb
Indikator 2 (Begrenzer)	Begrenzer	_
Indikator 3 (Name)	Ein Wort	Abc
	Zwei Wörter	AbcDef
	Konstante Variable Ein Wort	ABC
	Konstante Variable Zwei Wörter	ABC_DEF

Beispiel:			
Variable:		x_NothaltPodest	
Indikator:	X[1]	_[2]	NothaltPodest[3]
Typ/ Name:	Datentyp	Begrenzer	Name

Einrückstil:	Orientiert am GNU-Stil (siehe Seite 11, Abb. 1)	
Kommentare:	Jedes Programm besitzt einen Programmkopf	(*========== INFORMATIONEN ==========*)
	Hauptteil eines ganzen Blocks bekommt Überschrift	(*== TEXT ==*) *Abstand eine Zeile* Quellcode
	Teilabschnitte bekommen eine Überschrift	(* TEXT *) *Abstand eine Zeile* Quellcode
	Eine Quellcodezeile wird oberhalb kommentiert	//TEXT Quellcode
Layout:	Seitenlayout	

Tabelle 1: Programmier- und Einrückstil

In dieser Vorlage finden die Regeln nur teilweise Anwendung.

4 Ein neues Projekt anlegen

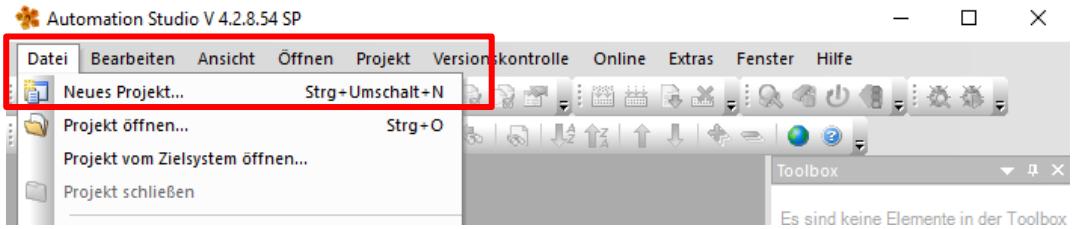


Abbildung 1: Ein neues Projekt anlegen - Ein neues Projekt erstellen

Schritt 1: Automation Studio starten.

Schritt 2: Reiter Datei ⇨ Neues Projekt.

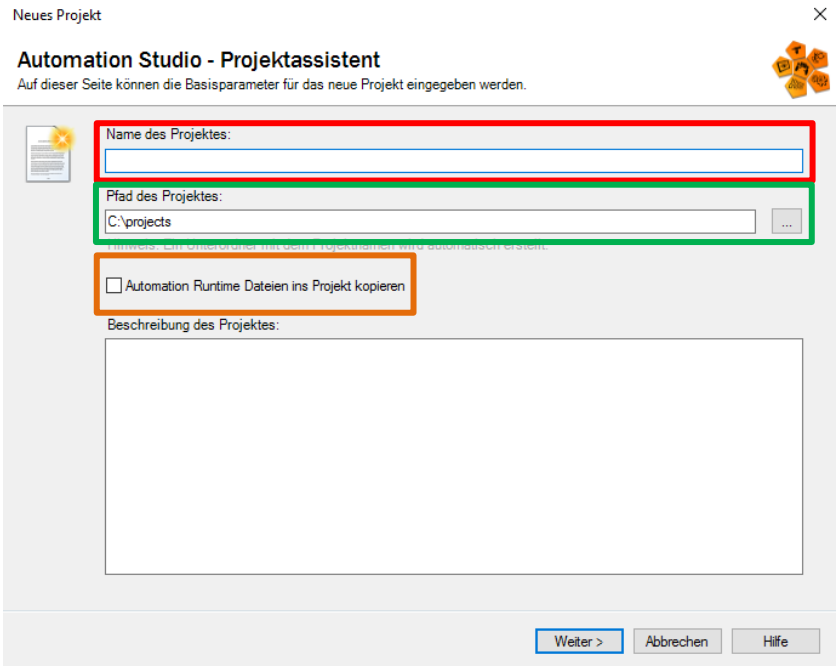


Abbildung 2: Ein neues Projekt anlegen - Projektassistent

Schritt 3: Einen Namen für das Projekt vergeben.

Info: Keine Umlaute oder Leerzeichen verwenden! Leerzeichen können dazu führen dass sich das Projekt nicht öffnen lässt.

Schritt 4: Projektpfad angeben.

Info: Ist ein Verzeichnis nicht vorhanden so wird es automatisch erstellt.

Schritt 5: Haken „Automation Runtime Dateien ins Projekt kopieren“ setzen.

Info: Dies hat den Vorteil, dass bei einem Projekt immer die passende AR Version mit dabei ist (z. B. bei der Weitergabe an einen Kollegen). Somit kann der Kollege auch dann dieses Projekt kompilieren, wenn er selbst die benötigte AR Version nicht installiert hat.

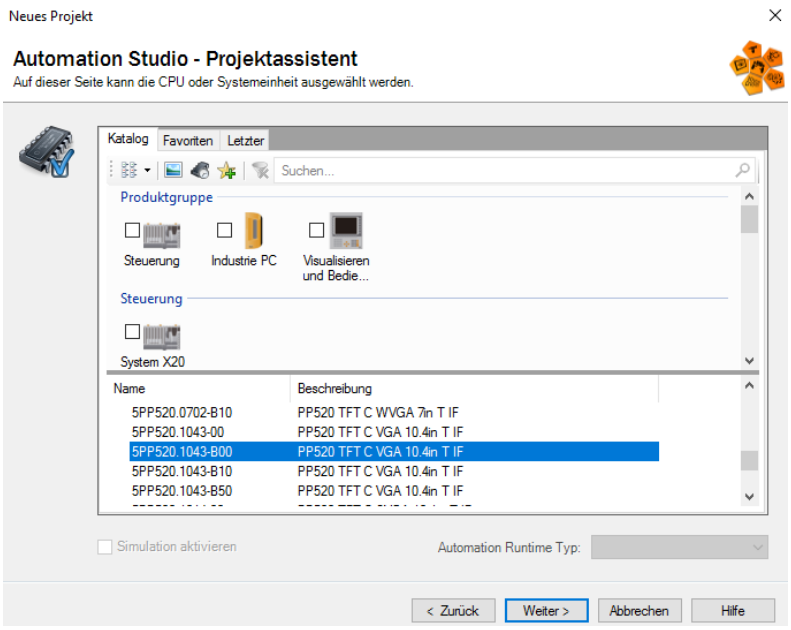


Abbildung 3: Ein neues Projekt anlegen - Wahl der Steuerung

Schritt 6: Im Auswahl Menü eine Steuerung auswählen.

Info: In dieser Dokumentation wird die gesamte Anlage simuliert. Es wurde sich für eine Steuerung mit Display entschieden.

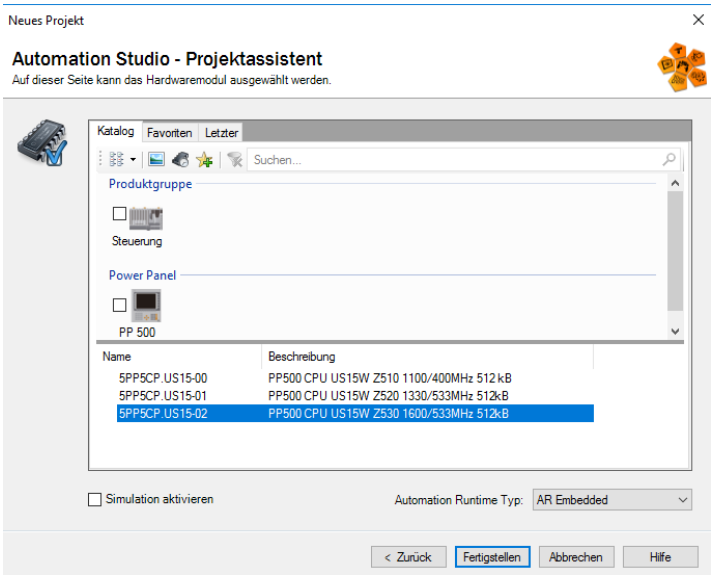


Abbildung 4: Ein neues Projekt anlegen - CPU Auswahl

Schritt 7: Im Auswahl Menü eine CPU auswählen.

Info: In dieser Dokumentation wird die gesamte Anlage simuliert. Es wurde sich für eine beliebige CPU entschieden.

Es kann die Option „Simulation“ ausgewählt werden. In diesem Fall wird die gesamte Konfiguration simuliert. In diesem Projekt wird die Simulation aber erst an späterer Stelle erstellt also bleibt die Option offen.

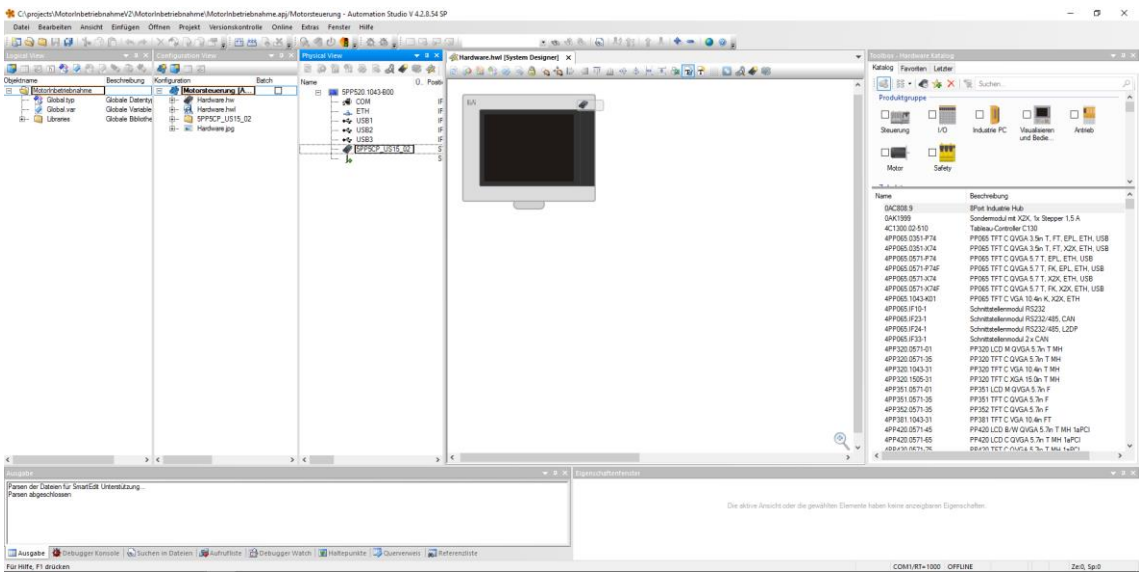


Abbildung 5: Ein neues Projekt anlegen - Benutzeroberfläche

Wenn alle Schritte durchgeführt sind erscheint die AS Benutzeroberfläche. Jetzt kann damit begonnen werden ein Programm zu schreiben und/ oder die Hardware erweitert werden.

Weiterführende Informationen bietet die B&R Hilfe. Nachfolgend sind die wichtigsten Hilfeseiten die sich auf dieses Kapitel beziehen aufgelistet:

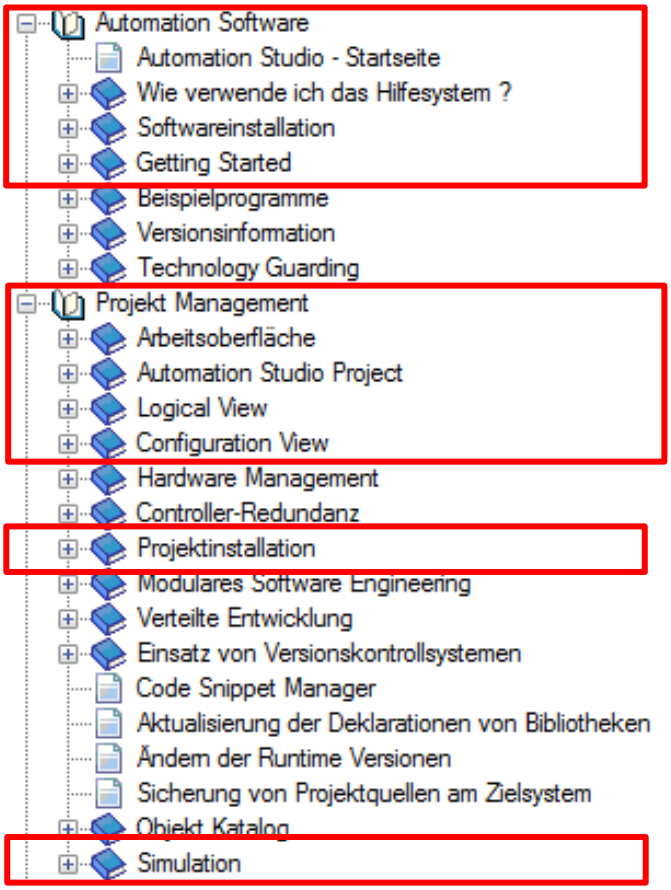


Abbildung 6: Ein neues Projekt anlegen - Hilfeseiten

5 Erstellen einer Hardwarekonfiguration

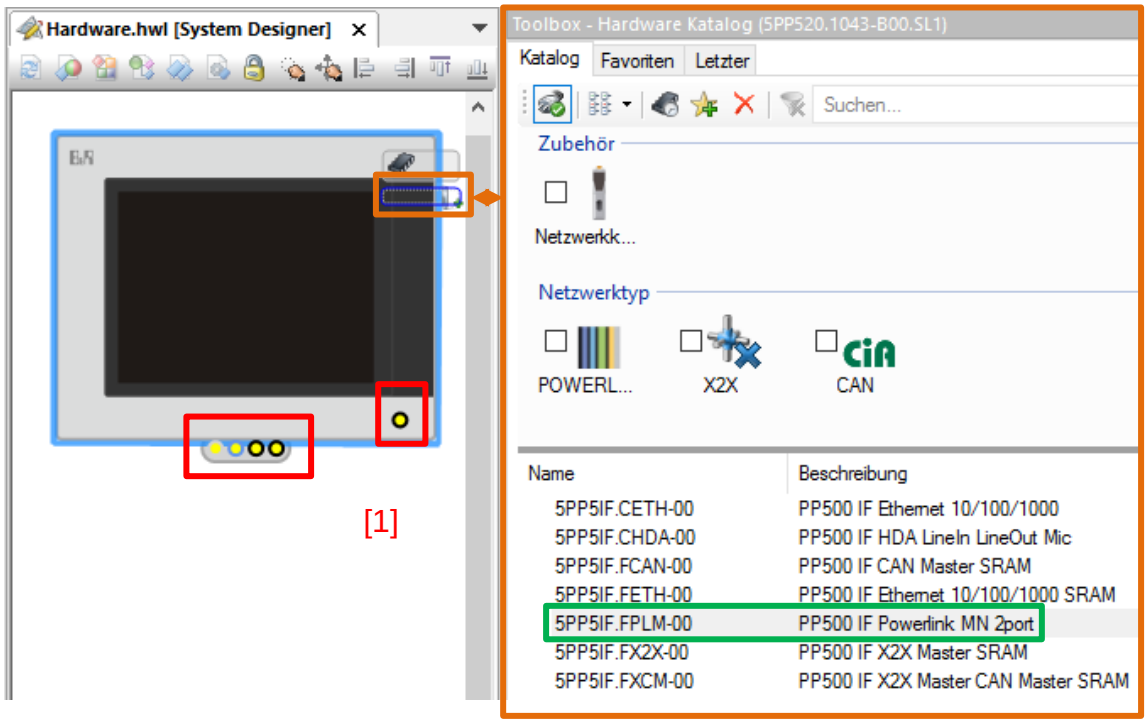


Abbildung 7: Erstellen einer Hardwarekonfiguration - Busschnittstelle

Wird der Mauszeiger auf die Knotenpunkte [1] gefahren wird einem eine Info angezeigt um was für einen Anschluss es sich handelt. Durch ein klicken auf die Knotenpunkte öffnet sich ein Kontextfenster in der Toolbox [2]. Dort werden einem die vorhandenen Möglichkeiten zur Konfiguration angezeigt.

Schritt 1: Busschnittstelle auswählen. Mit Doppelklick auf den Modulnamen im Kontextmenü wird jenes in die Hardwarekonfiguration eingefügt.

Info: In dieser Dokumentation wurde sich für eine Powerlink-schnittstelle entschieden.

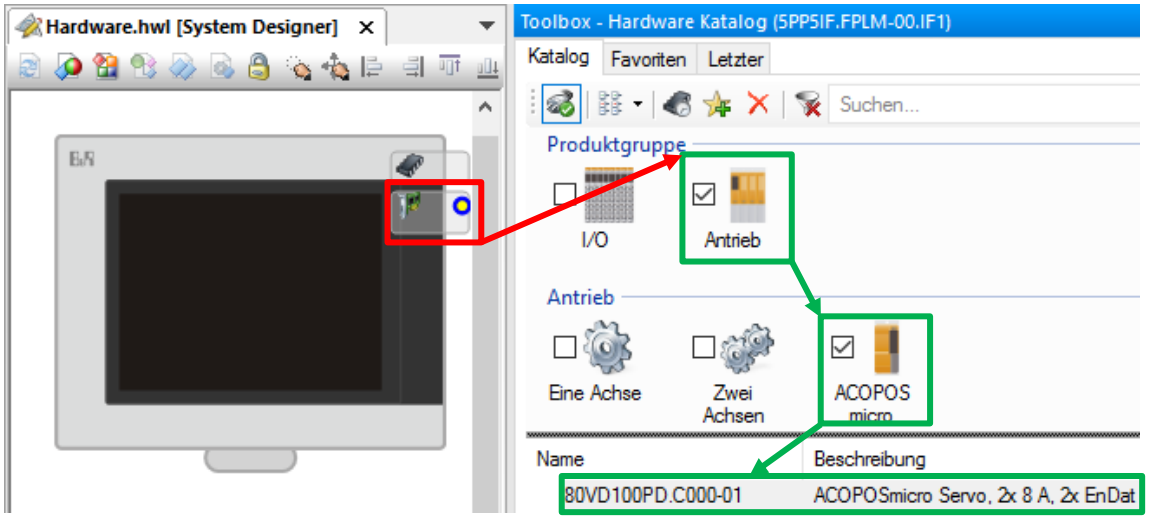


Abbildung 8: Erstellen einer Hardwarekonfiguration - Antrieb

Schritt 2: Schritt 1 wird nun über den Knotenpunkt Schnittstellenmodul wiederholt sodass im Kontextmenü die Hardware auftaucht die sich über Powerlink mit der Steuerung verbinden lässt. Die ausgewählte Hardware wird wieder über einen Doppelklick der Konfiguration hinzugefügt.

Info: *In dieser Dokumentation wurde sich für einen ACOPOSmicro als Antrieb entschieden.*

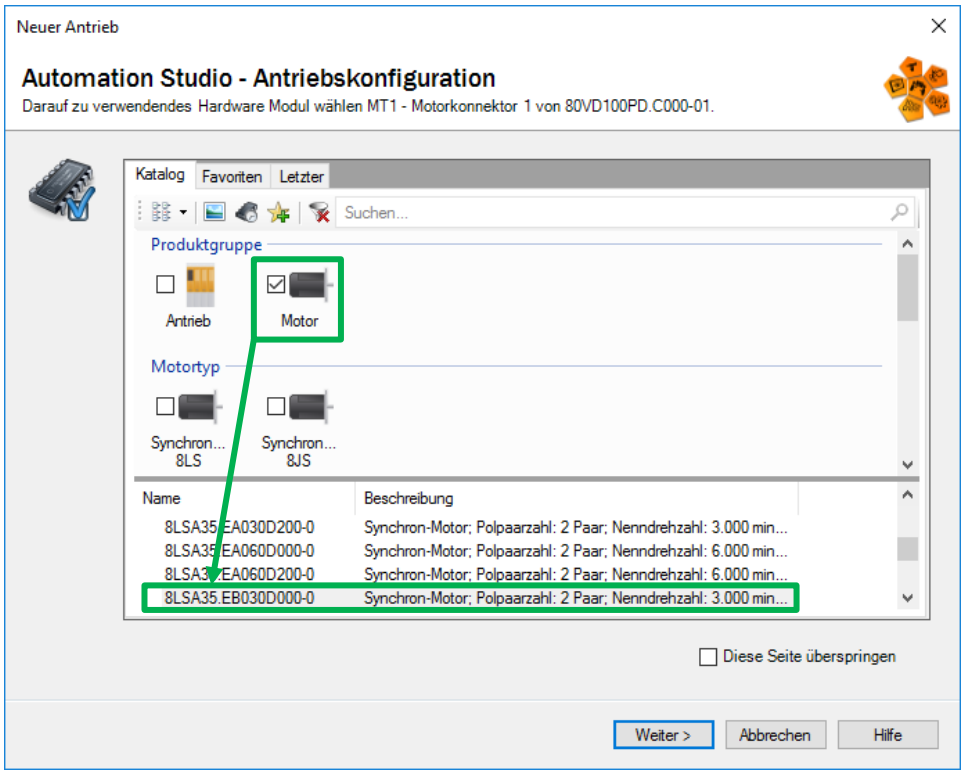


Abbildung 9: Erstellen einer Hardwarekonfiguration - Antrieb

Nachdem eine Antriebseinheit ausgewählt worden ist öffnet sich der Assistent für die Antriebskonfiguration.

Schritt 3: Motor auswählen.

Info: In dieser Dokumentation wurde sich für einen Standard Synchron-Motor entschieden.

Info: Auswahl der Option „Diese Seite überspringen“ überspringt die Motorkonfiguration. Bei elektronischen Typenschildern wie z.B. EnDat 2.x Unterstützung wird der Motortyp automatisch ausgelesen.

Nachfolgend kann, je nach Umrichtergröße, noch ein zweiter Motor im nachfolgenden Fenster auf dieselbe Weise ausgewählt werden.

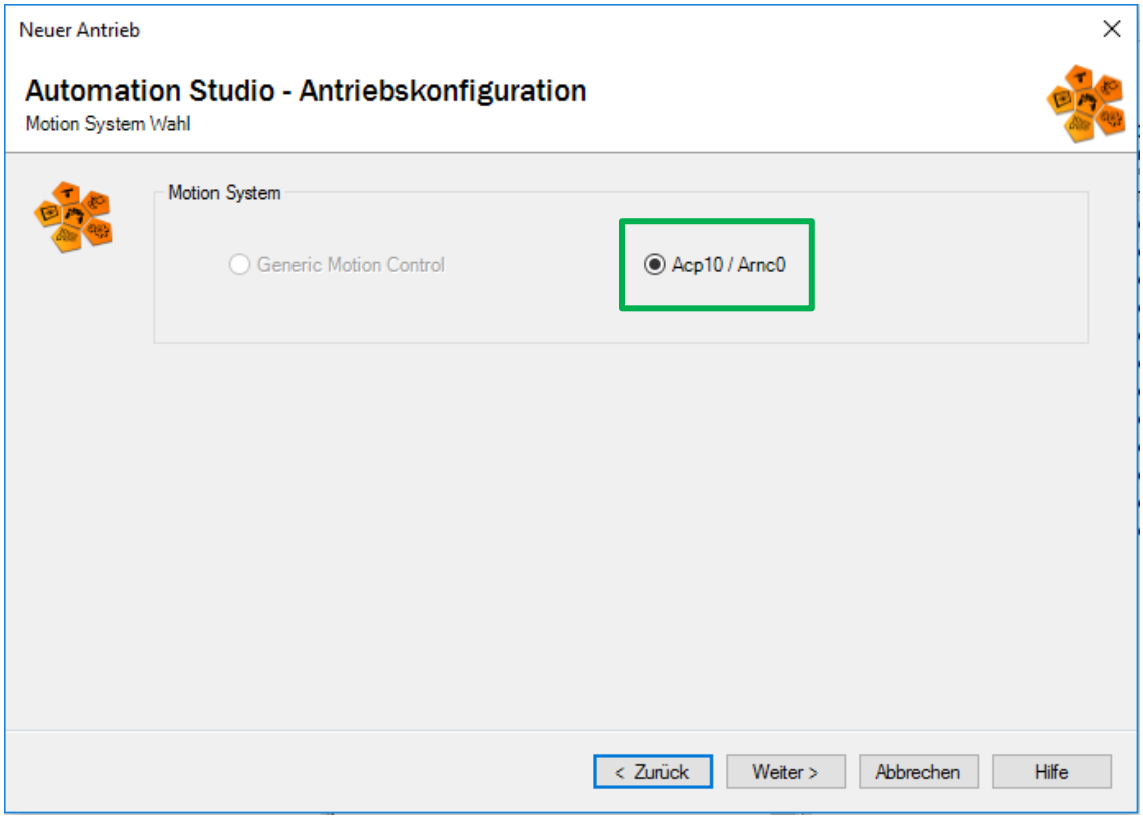


Abbildung 10: Erstellen einer Hardwarekonfiguration - Motion System

Schritt 4: Motion System Wahl.

Info: Acp10/ Arnc0 dient der Antriebssteuerung über ACOPOS Umrichter.

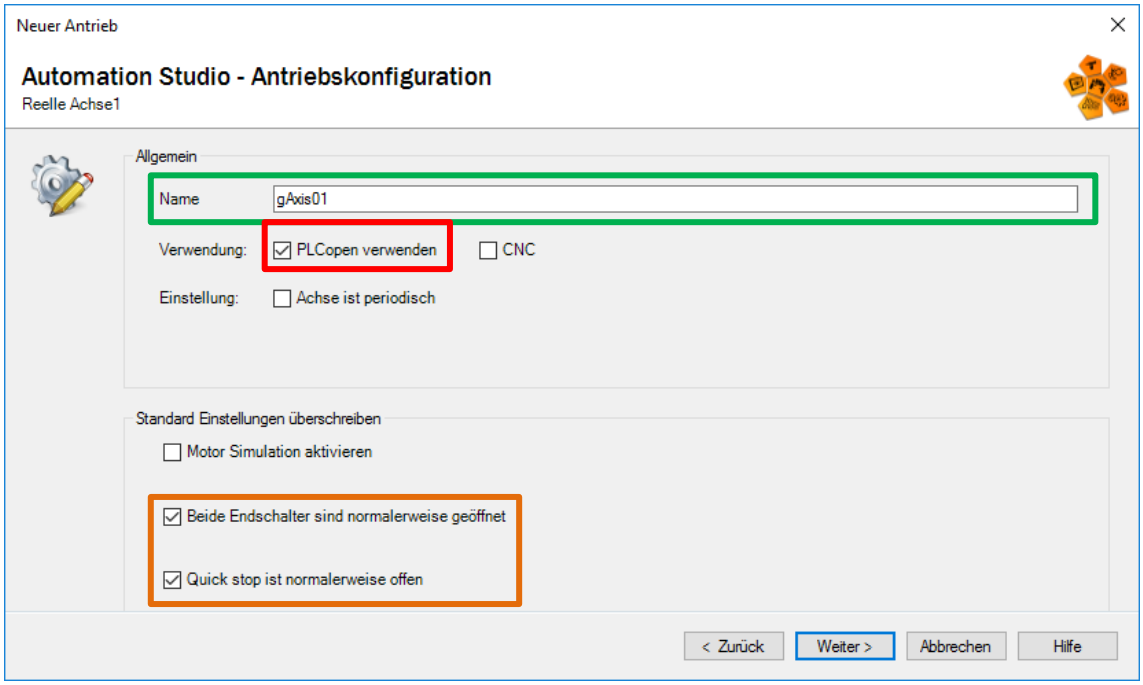


Abbildung 11: Erstellen einer Hardwarekonfiguration - Antriebskonfiguration

Schritt 5: Namen für die reelle Achse vergeben.

Info: Keine Umlaute oder Leerzeichen verwenden! Dieser Name wird als Achsobjekt in die Spalte "NC Object Name" in die NC Zuordnungstabelle eingefügt. Zusätzlich wird eine globale Prozessvariable mit diesem Namen und vom passenden Typ angelegt. Diese Variable wird während der Programmierung als Handle zur Achse verwendet.

Schritt 6: Option „PLCopen verwenden“ auswählen.

Info: Wenn diese Checkbox angehakt ist, dann wird, falls noch nicht im Projekt vorhanden, die Bibliothek "acp10_mc" ins Projekt eingefügt damit die Bedienung dieser Achse mit PLCopen-Funktionsblöcken möglich ist.

Schritt 7: „Beide Endschalter sind normalerweise geöffnet“ auswählen.

Info: Wenn ausgewählt, beide Endschalter werden auf "normal offen" konfiguriert. D.h. erst wenn an den Eingängen Spannung angelegt wird, wird der Endschalter ausgelöst. Diese Einstellung ist beispielsweise notwendig wenn keine physikalischen Schalter vorhanden sind. Durch diese Einstellung werden die Endschalter nicht mehr auf Drahtbruch überwacht und lösen erst mit Spannung aus!

„Quick stop ist normalerweise offen“ auswählen“ auswählen.

Info: Wenn angewählt, Notstop Eingang wird auf "normal offen" konfiguriert. D.h. erst wenn am Eingang Spannung angelegt wird, wird der Notstop ausgelöst. Diese Einstellung ist beispielsweise notwendig wenn kein physikalischer Schalter vorhanden ist. Durch diese Einstellung wird der Notstop nicht mehr auf Drahtbruch überwacht und löst erst mit Spannung aus!

Nachdem die Konfiguration für die reelle Achse 1 (gAxis01) vorgenommen wurde wird der Vorgang auf der nächsten Seite nochmal für die Achse 2 (gAxis02) vorgenommen. Das Erstellen der virtuellen Achsen kann übersprungen werden.

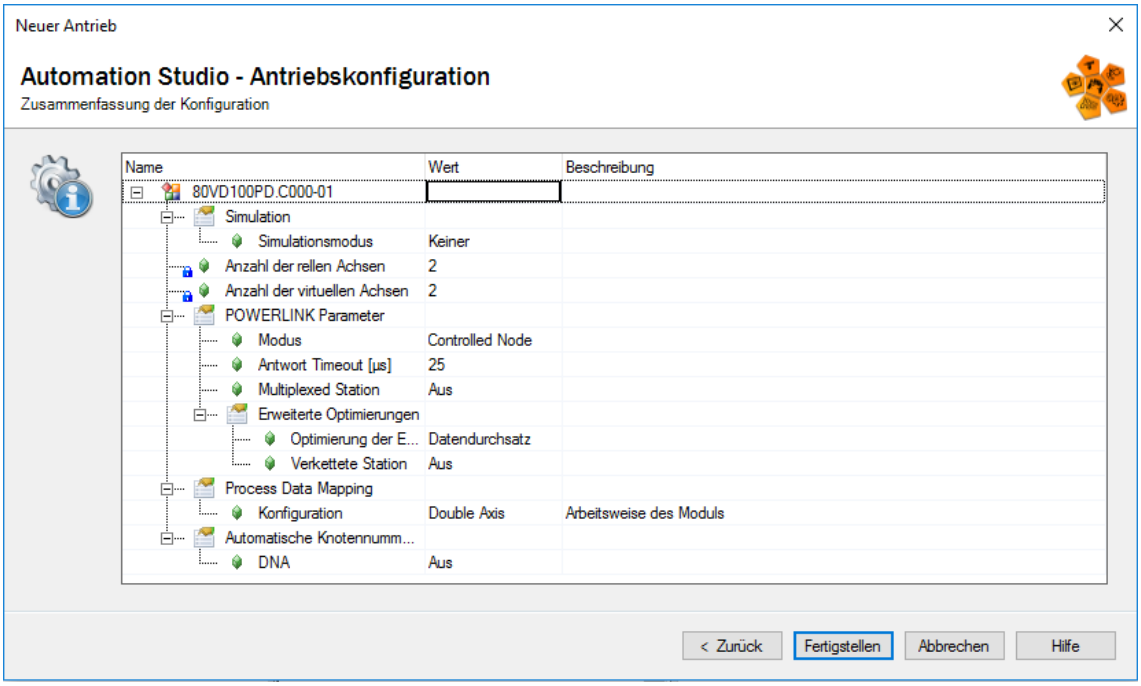


Abbildung 12: Erstellen einer Hardwarekonfiguration - Zusammenfassung

Nachdem die Auswahl und Konfiguration der Achsen stattgefunden hat gibt einem die letzte Seite nochmal eine kleine Übersicht der vorgenommenen Einstellungen.

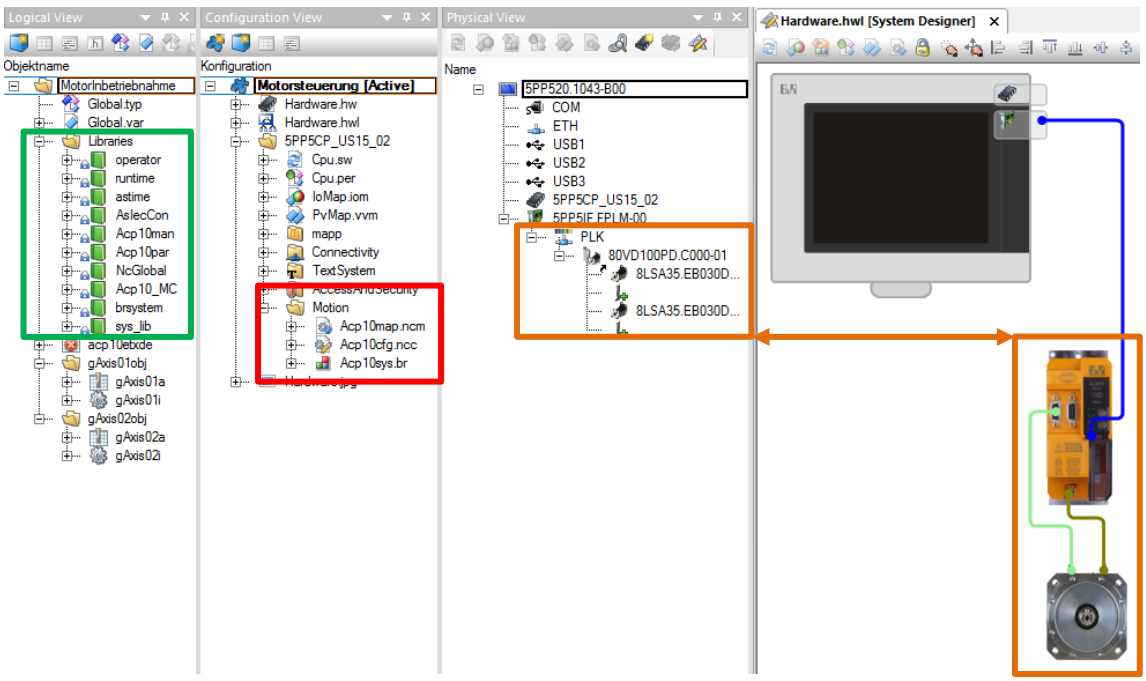


Abbildung 13: Erstellen einer Hardwarekonfiguration - Benutzeroberfläche Nachher

Wenn der Assistent für die Antriebskonfiguration beendet ist werden alle Einstellungen automatisch in das Projekt übernommen, Bibliotheken eingefügt, die Motion Konfiguration erstellt und die Hardwarekonfiguration aktualisiert.

Weiterführende Informationen bietet die B&R Hilfe. Nachfolgend sind die wichtigsten Hilfeseiten die sich auf dieses Kapitel beziehen aufgelistet:

Abbildung 14: Erstellen einer Hardwarekonfiguration - Hilfeseiten

6 Erstellen von Strukturtypen und definieren von Variablen

In den vorangegangenen Kapiteln sind die Grundsteine gelegt. Jetzt kann damit begonnen werden ein Programm zu schreiben um die Achse (gAxis01) anzusteuern. Dafür werden bestimmte Funktionsbausteine sowie Variablen benötigt. Um welche Funktionsblöcke und Variablen es sich dabei handelt und wie man diese anlegt und in ein Programm einbettet wird in diesem Kapitel näher erläutert.

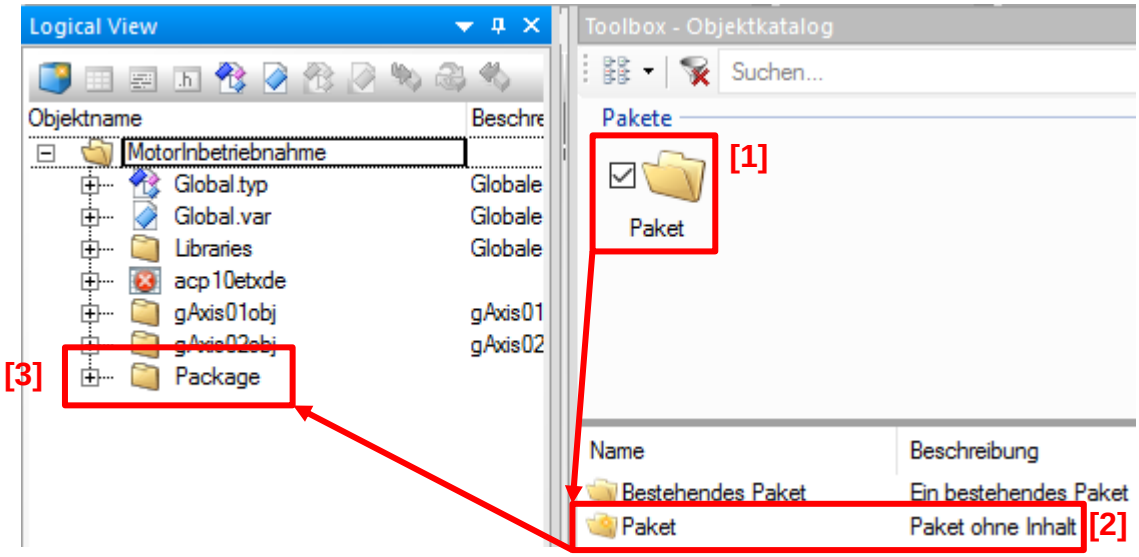


Abbildung 15: Erstellen von Strukturtypen und definieren von Variablen - Ordner

Schritt 1: Einen leeren Ordner einfügen.

Info: Im Logical View den Hauptordner anwählen. Im Kontextmenü der Toolbox erscheinen Optionen. „Paket“ auswählen und „Paket – Paket ohne Inhalt“ mit Doppelklick in das Projekt einfügen.

Info: Es ist immer sinnvoll eine Ordnerstruktur in seinem Projekt zu haben. In diesem Ordner wird später das Programm für die Ansteuerung der Achse (gAxis01) angelegt.

(optional) Namen vergeben.

Info: Rechtsklick auf den Ordner ⇨ Umbenennen.

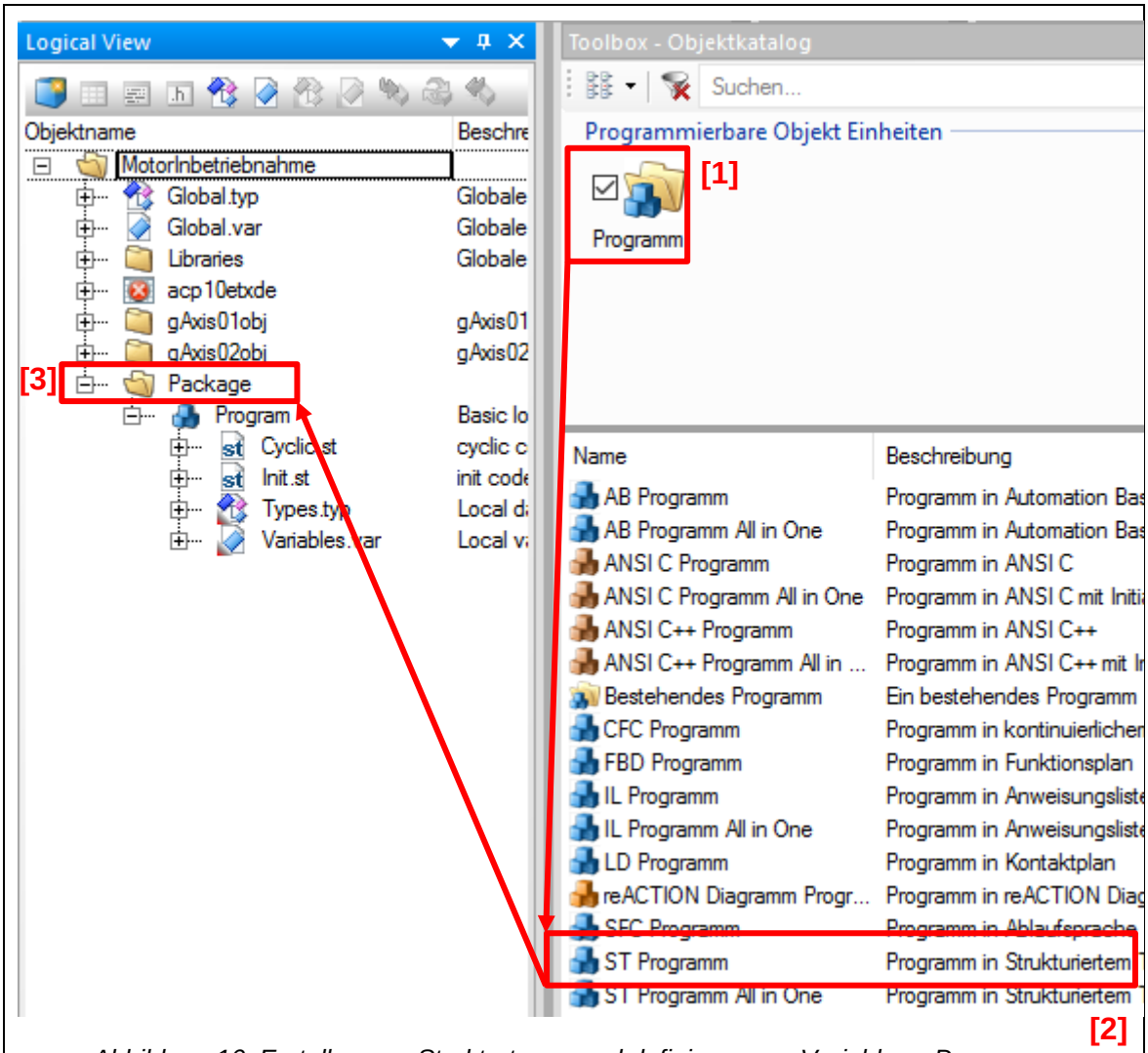


Abbildung 16: Erstellen von Strukturtypen und definieren von Variablen - Programm

Schritt 2: Ein ST Programm einfügen.

Info: Im Logical View den zuvor erstellten leeren Ordner „Package“ anwählen. Im Kontextmenü der Toolbox „Programm“ auswählen und „ST Programm“ mit Doppelklick in den Ordner einfügen.

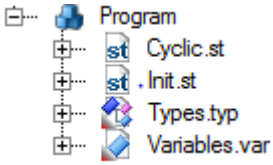


Abbildung 17

Cyclic:	Wird je nach verwendeter Zykluszeit wiederholt.
Init:	Wird nur einmalig bei Anlagenstart aufgerufen.
Types:	Variablen können hier in Grp. zusammengefasst werden.
Variables:	Hier werden Variablen ohne Grp. Bezug definiert.

Program::Variables.var [Variablendeklaration]*

Name	Typ	Referenz	Konstante	Retain	Duplizierbar	Wert	Beschreibung [1]
	==========						
	ACHSENSTEUERUNG						
	==========						
t_AchsenCtrl	AchsenCtrl_typ	☐	☐	☐	☑		Achsen Typenstruktur
udi_Achse1Obj	UDINT	☐	☐	☐	☑		Achsen Pointer
	==========						
	FUNKTIONSBAUSTEINE						
	==========						
fb_MC_Home	MC_Home	☐	☐	☐	☑		FB Home
fb_MC_MoveAbsolute	MC_MoveAbsolute	☐	☐	☐	☑		FB MoveAbsolute
fb_MC_Power	MC_Power	☐	☐	☐	☑		FB Power
fb_MC_ReadActualPosition	MC_ReadActualPosition	☐	☐	☐	☑		FB ReadActualPosition
fb_MC_ReadActualVelocity	MC_ReadActualVelocity	☐	☐	☐	☑		FB ReadActualVelocity
fb_MC_ReadAxisError	MC_ReadAxisError	☐	☐	☐	☑		FB ReadAxisError
fb_MC_ReadStatus	MC_ReadStatus	☐	☐	☐	☑		FB ReadStatus
fb_MC_Reset	MC_Reset	☐	☐	☐	☑		FB Reset
fb_MC_Stop	MC_Stop	☐	☐	☐	☑		FB Stop
fb_MC_BR_ReadDriveStatus	MC_BR_ReadDriveStatus	☐	☐	☐	☑		FB ReadDriveStatus
fb_MC_Halt	MC_Halt	☐	☐	☐	☑		FB Halt
	==========						
	SCHRITTKETTE						
	==========						
ui_AchsenSchritt	UINT	☐	☐	☐	☑		Aktueller Schritt
STATE_WAIT	USINT	☐	☑	☐	☐	0	Schritt: Warten auf Achs
STATE_POWER_ON	USINT	☐	☑	☐	☐	1	Schritt: Achsenregler
STATE_HOME	USINT	☐	☑	☐	☐	2	Schritt: Achse referenzie
STATE_READY	USINT	☐	☑	☐	☐	10	Schritt: Warten auf Bew
STATE_STOP	USINT	☐	☑	☐	☐	11	Schritt: Achse gestoppt
STATE_MOVE_ABSOLUTE	USINT	☐	☑	☐	☐	14	Schritt: Absolute Beweg
STATE_HALT	USINT	☐	☑	☐	☐	17	Schritt: Achse angehalte
STATE_ERROR_AXIS	USINT	☐	☑	☐	☐	100	Schritt: Achsenfehler
STATE_ERROR	USINT	☐	☑	☐	☐	101	Schritt: Fehler allgemein
STATE_ERROR_RESET	USINT	☐	☑	☐	☐	102	Schritt: Fehler zurückset

Abbildung 18: Erstellen von Strukturtypen und definieren von Variablen - Variables.var

Schritt 3: Variablen in „Variables.var“ anlegen. (siehe Schritt 3.1 – 3.4)

Info: Alle jetzt folgenden Variablen werden lokal angelegt. D.h. diese können nur in dem in Schritt 2 angelegten ST Programm verwendet werden!

Um eine Variable anzulegen muss wie folgt vorgegangen werden: (siehe kommende Seiten)

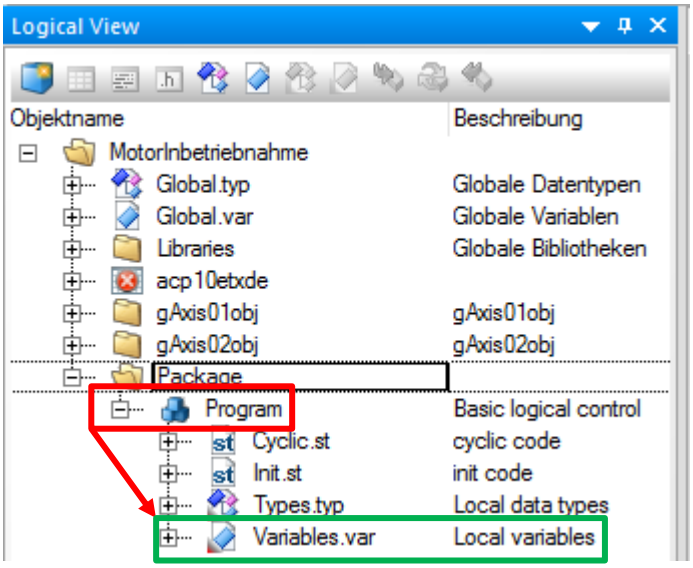


Abbildung 19: Erstellen von Strukturtypen und definieren von Variablen - Variables.var Tabelle

Schritt 3.1: Doppelklick auf „Variables.var“

Info: Es öffnet sich ein neues Fenster (siehe Abbildung 19 Seite 21). Dort können jetzt die Variablen angelegt werden.

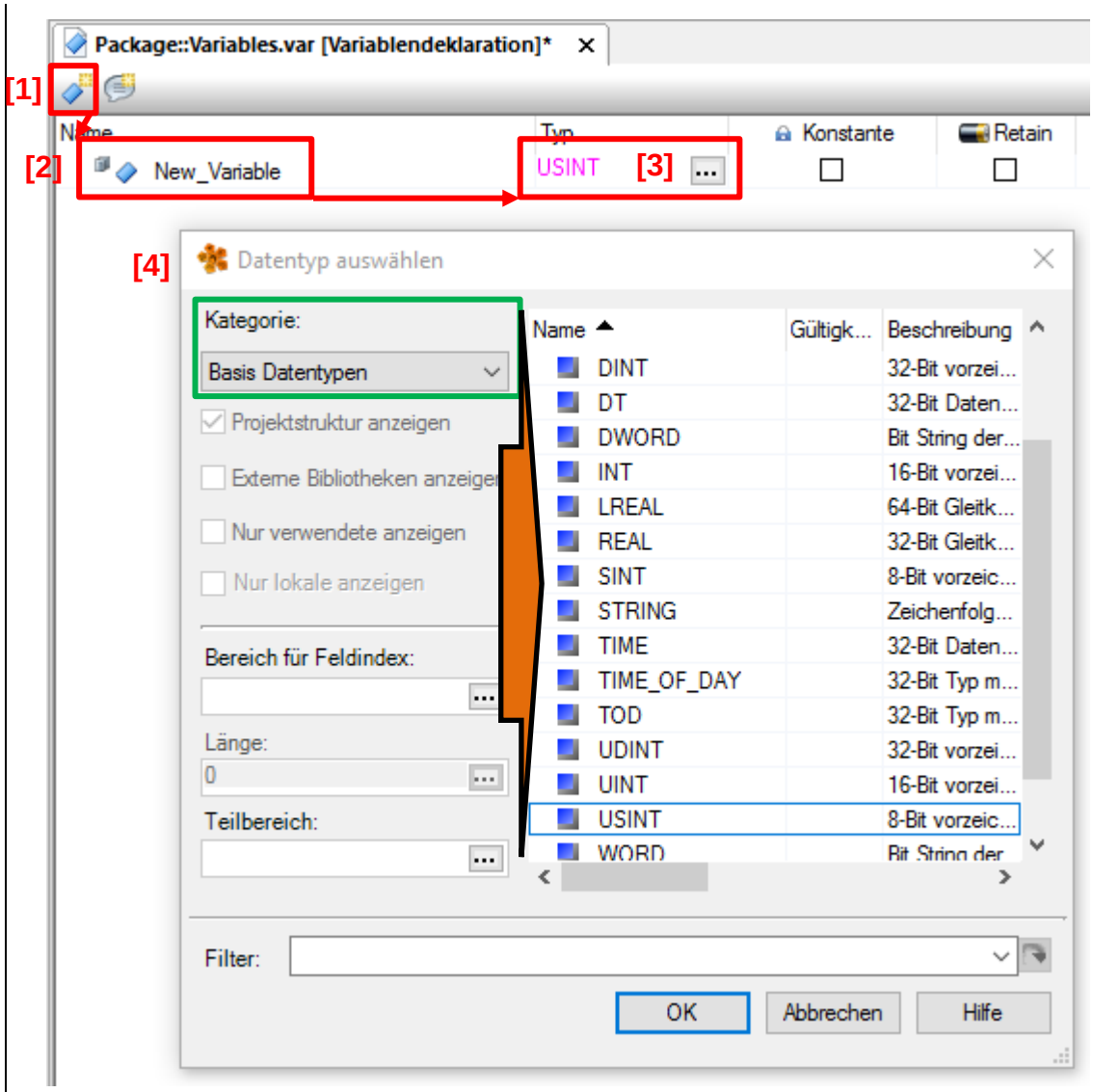


Abbildung 20: Erstellen von Strukturtypen und definieren von Variablen - Variable definieren

Schritt 3.2: Einfach Klick auf Variable hinzufügen [1].

Schritt 3.3: Der Variable einen Namen geben.

Info: Keine Umlaute oder Sonderzeichen verwenden!

Schritt 3.4: Doppelklick in das Feld „Typ“, danach auf die „Schaltfläche [3]“ einfach klicken.

Info: Es öffnet sich ein neues Fenster [4] wo man über die „Kategorie“ ⇨ „Basis Datentypen“ einen Datentyp für die Variable festlegen kann. Alternativ lässt sich der Datentyp auch in das Feld „Typ“ schreiben.

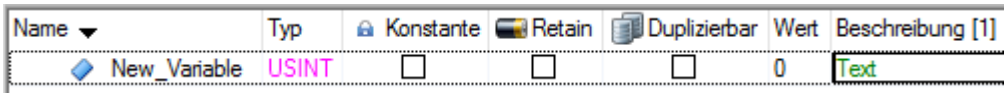


Abbildung 21: Erstellen von Strukturtypen und definieren von Variablen - Variablen Deklaration

Name: Variablenname.

Typ: Datentyp oder Pfadangabe auf einen FB oder Typenstrukturen.

Konstante: Variable als Konstante festlegen.

Retain: Batteriegepuffert (remanent).

Duplizierbar: Variable ist nachführbar (redundant).

Wert: Initialwert der Variable oder Konstante.

Beschreibung: Beschreibungstext der Variable.

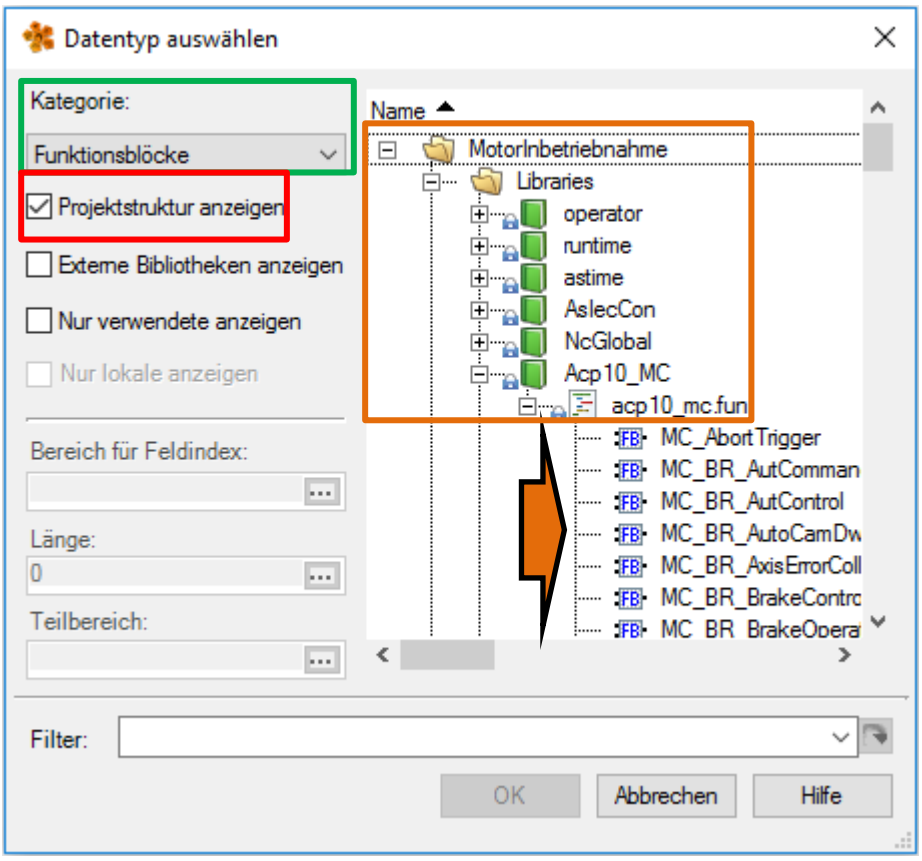


Abbildung 22: Erstellen von Strukturtypen und definieren von Variablen - Variable FB

Soll ein FB einer Variable zugewiesen werden so wird wie in **Schritt 3.3** vorgegangen. Über „Kategorie“ ⇨ „Funktionsblöcke“ wählen. Haken bei „Projektstruktur anzeigen“ setzen und dann über „Libraries“ ⇨ „Acp10_MC“ ⇨ „acp10_mc.fun“ zu den FB navigieren und den gewünschten Block auswählen.

Es gibt eine Ausnahme die erst in den nächsten Schritten behandelt wird, da diese Variable auf eine Typenstruktur mit mehreren Variablen zugreift.

Abbildung 23

Bis dahin sollten alle Variablen wie in Abbildung 19 auf Seite 21 nach den Schritten 3.1 – 3.4 angelegt worden sein.

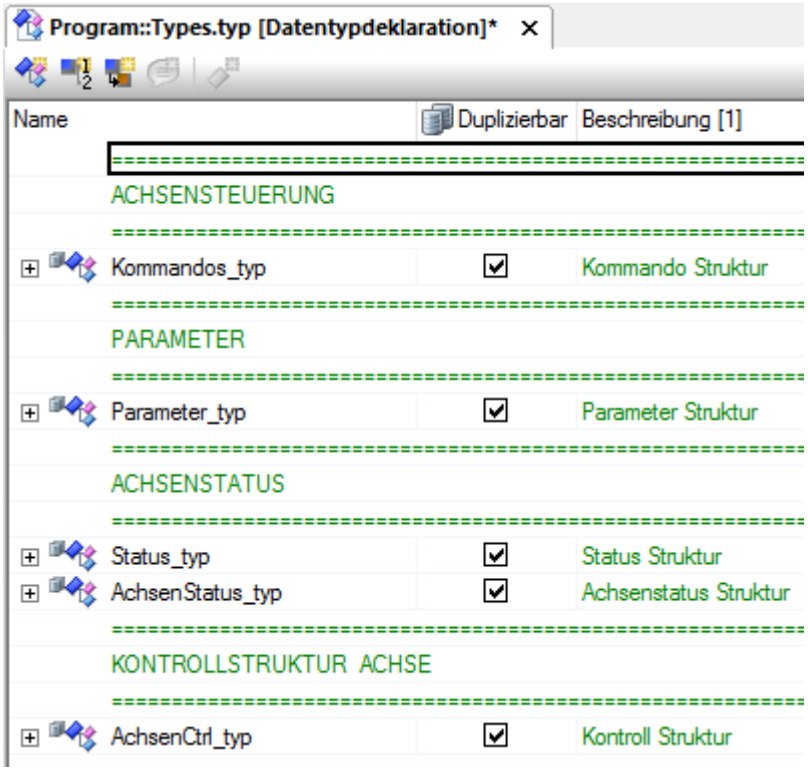


Abbildung 24: Erstellen von Strukturtypen und definieren von Variablen - Typenstruktur

Schritt 4: Typenstrukturen in „Types.typ“ anlegen. (siehe Schritt

Info: Alle jetzt folgenden Variablen und Typen werden lokal angelegt. D.h. diese können nur in dem in Schritt 2 angelegten ST Programm verwendet werden!

Um einen Typ anzulegen muss wie folgt vorgegangen werden: (siehe kommende Seiten)

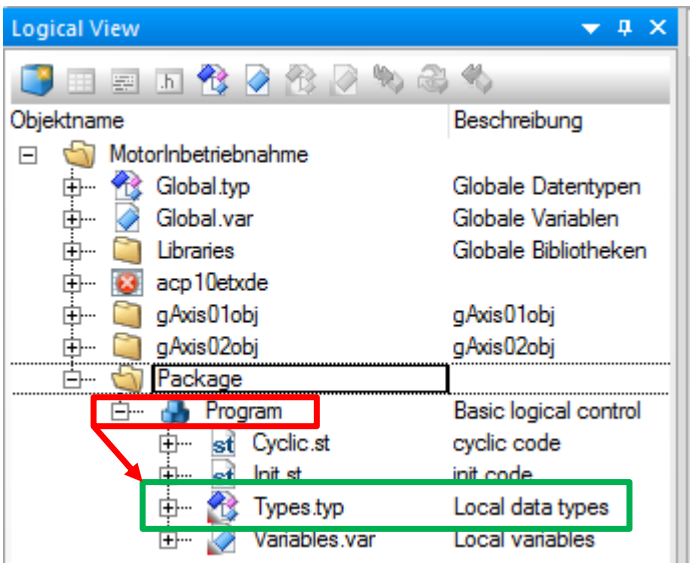


Abbildung 25: Erstellen von Strukturtypen und definieren von Variablen - Types.typ

Schritt 4.1: Doppelklick auf „Types.typ“.

Info: Es öffnet sich ein neues Fenster (siehe Abbildung 24 Seite 25). Dort können jetzt die Typen und Variablen angelegt werden.

Info: Sollen Strukturtypen Variablen zugewiesen werden, so muss das Projekt vorher immer gespeichert werden damit die Änderungen übernommen werden!

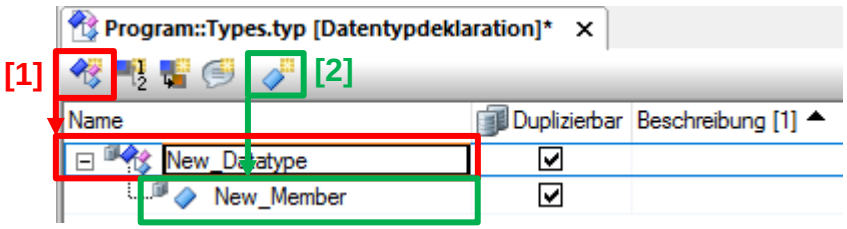


Abbildung 26: Erstellen von Strukturtypen und definieren von Variablen - Typ und Variable anl.

Schritt 4.2: Einfach Klick auf „Struktur hinzufügen“ [1].

Info: In der Struktur werden Variablen als „Gruppe“ zusammengefasst. Eine Struktur sorgt für mehr Übersichtlichkeit im Projekt. Zudem ist es einem möglich diese Struktur einer Variablen zuzuweisen mit Hilfe derer man im gesamten Projekt auf die Variablen innerhalb der Struktur zugreifen kann.

Schritt 4.3: Der Struktur einen Namen geben.

Info: Keine Umlaute oder Sonderzeichen verwenden!

Schritt 4.4: Einfach Klick auf die angelegte Struktur, danach auf die Schaltfläche „Typelement“ [2] einfach klicken und somit eine Variable hinzuzufügen.

Info: Diese Variable kann wieder wie in Schritt 3.4 auf Seite 23 und 24 deklariert werden. Jedoch mit dem Unterschied, dass sich diese Variable nicht remanent oder konstant deklarieren lässt.

Schritt 4.5: Dem Typelement einen Namen geben.

Info: Keine Umlaute oder Sonderzeichen verwenden!

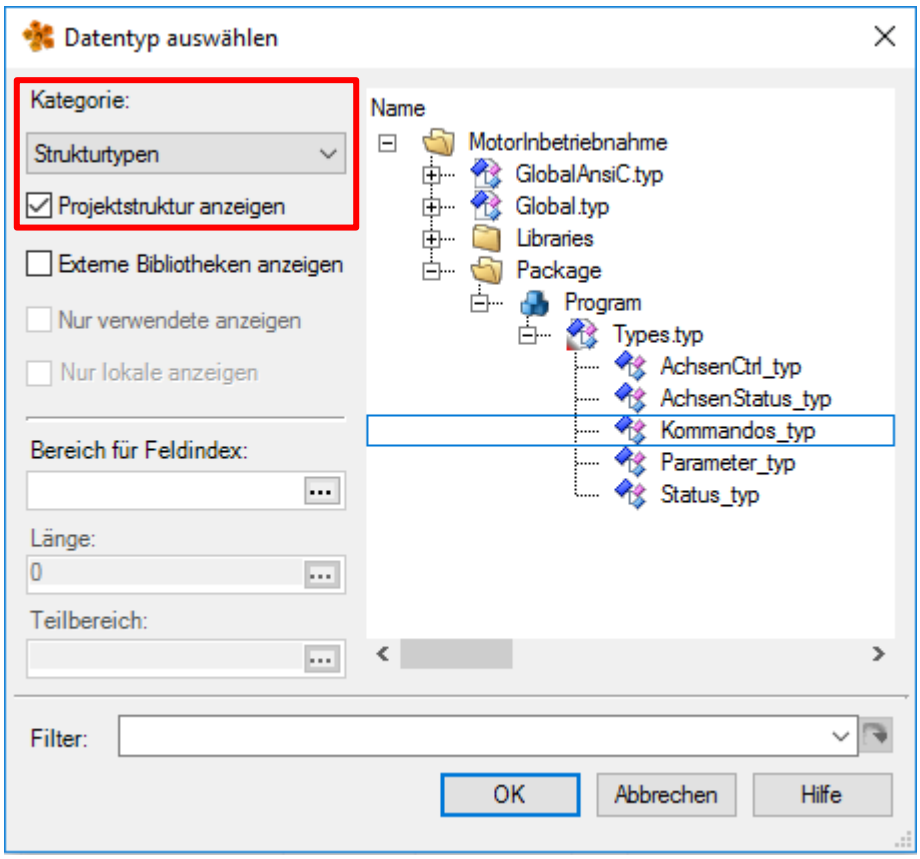


Abbildung 27: Erstellen von Strukturtypen und definieren von Variablen - Struktur Variable zuweisen

Schritt 4.6: Soll der Variable ein Strukturtyp zugewiesen werden wie auf Seite 25 Abbildung 24. So muss einfach im sich öffnenden Fenster die „Kategorie“ ⇨ „Strukturtypen“ geändert werden.

Die Schritte 4.2 – 4.6 werden jetzt wiederholt angewendet bis man folgende Strukturen fertig angelegt hat: (siehe kommende Seite 28, Abbildung 28)

Program::Types.typ [Datentypdeklaration]

Name	Typ	& Referenz	Duplizierbar	Wert	Beschreibung [1]
ACHSENSTEUERUNG					
Kommandos_typ			☑		Kommando Struktur
Power	BOOL	☐	☑		Regler einschalten
Home	BOOL	☐	☑		Achse referenzieren
MoveAbsolute	BOOL	☐	☑		Bewegung zu einer definierten Position
Halt	BOOL	☐	☑		Aktive Bewegung anhalten
Stop	BOOL	☐	☑		Aktive Bewegeung stoppen
ErrorAcknowle...	BOOL	☐	☑		Aktive Fehler zurücksetzen
PARAMETER					
Parameter_typ			☑		Parameter Struktur
Position	REAL	☐	☑		Positionsangabe für absolute Bewegung
Velocity	REAL	☐	☑		Geschwindigkeitsangabe
Direction	USINT	☐	☑		Drehrichtung
Acceleration	REAL	☐	☑		Beschleunigung
Deceleration	REAL	☐	☑		Entschleunigung
HomePosition	REAL	☐	☑		Referenzpositionsangabe
HomeMode	USINT	☐	☑		Referenzmodus
ACHSENSTATUS					
Status_typ			☑		Status Struktur
ErrorID	UINT	☐	☑		FehlerID
ErrorText	STRING[79][0..3]	☐	☑		Fehlertext
ActPosition	REAL	☐	☑		Aktuelle Position der Achse
ActVelocity	REAL	☐	☑		Aktuelle Geschwindigkeit der Achse
DriveStatus	MC_DRIVESTATUS_TYP	☐	☑		Aktueller Status der Achse
AchsenStatus_typ			☑		Achsenstatus Struktur
Disabled	BOOL	☐	☑		Achse freischalten
StandStill	BOOL	☐	☑		Achse stillsetzen
Homing	BOOL	☐	☑		Achse referenzieren
Stopping	BOOL	☐	☑		Achse stoppen
DiscreteMotion	BOOL	☐	☑		Achse diskrete Bewegung
ContinuousMot...	BOOL	☐	☑		Achse kontinuierliche Bewegung
Synchronized...	BOOL	☐	☑		Achse synchronisierte Bewegung
ErrorStop	BOOL	☐	☑		Achse Fehlerstop
KONTROLLSTRUKTUR ACHSE					
AchsenCtrl_typ			☑		Kontroll Struktur
Command	Kommandos_typ	☐	☑		Kommando Struktur
Parameter	Parameter_typ	☐	☑		Parameter Struktur
Status	Status_typ	☐	☑		Status Struktur
AxisState	AchsenStatus_typ	☐	☑		Achsenstatus Struktur

Abbildung 28: Erstellen von Strukturtypen und definieren von Variablen - Typenstrukturen

Sind alle Strukturen angelegt und Variablen definiert kann mit dem programmieren begonnen werden.

7 Erstellen eines Programms in ST

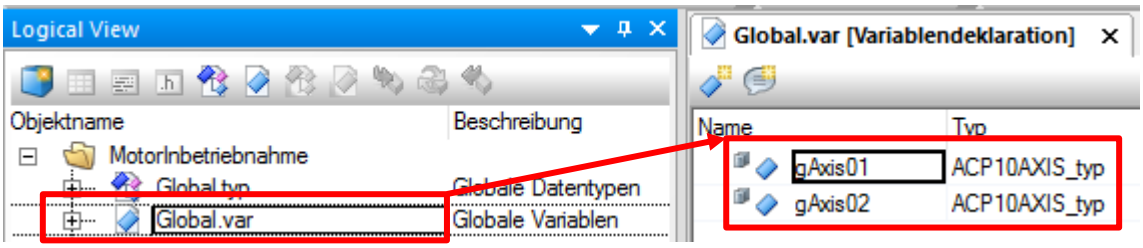
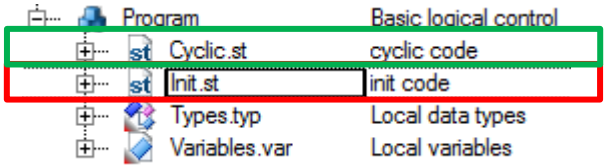


Abbildung 29: Erstellen eines Programms in ST

Nachdem die Antriebskonfiguration aus Kapitel 5 abgeschlossen wurde sind automatisch global zwei Variablen erstellt worden. Diesen Variablen sind die im Projekt hinterlegten Antriebsparametern zugewiesen, ACP10AXIS_typ. Der Datenaustausch wird über den NC-Manager geregelt.

Zuerst wird damit begonnen Startparameter beim Programmstart festzulegen.



Schritt 1: Doppelklick auf „Init.st“.

Info: Es öffnet sich ein neues Fenster wo wir jetzt den Programmcode schreiben werden.

```
(*=====================================================================*
  Programm:          Program
  Datei:             Init.st
  Auto:              B&R / T.Oltmann
  Erstellt:          26.05.2017
  Letzte Änderung:   26.05.2017
  geändert von:      T.Oltmann

  Notiz:
  1) Dieses Programm dient als Vorlage für die Dokumentation
     "Ansteuern eines Motors im AS mit ST".

=====================================================================*)
PROGRAM _INIT

  //Zuweisung der Parameteradresse an die UDINT Variable
    Achse1Obj
  udi_Achse1Obj := ADR(gAxis01);

  //Schrittzuweisung bei Programmstart
  ui_AchsenSchritt := STATE_WAIT;

  //Geschwindigkeitswert wird dem Parameter Velocity übergeben
  t_AchsenCtrl.Parameter.Velocity             := 1000;
  //Beschleunigungswert wird dem Parameter Acceleration übergeben
  t_AchsenCtrl.Parameter.Acceleration         := 5000;
  //Beschleunigungswert wird dem Parameter Deceleration übergeben
  t_AchsenCtrl.Parameter.Deceleration         := 5000;

END_PROGRAM
```

Abbildung 30: Erstellen eines Programms in ST - INIT Programm

Schritt 2: Zum Programmstart legen wird die Parameter fest mit denen sich unsere Achse bewegen soll. Dies muss allerdings nicht zwangsläufig im „Init.st“ geschehen. Wichtig ist jedoch, dass wir die Parameteradresse bzw. unsere Achsenadresse an unsere Variable „udi_Achse1Obj“ übergeben und unserer Schrittkette im „Cycle.st“ ihren Startschritt „STATE_WAIT“ zuweisen.

Info: Über die Tastenkombination „Strg“ + „Leertaste“ öffnet sich beim Schreiben der Variable ein Kontextmenü wo es einem erlaubt ist durch die gesamten angelegten Variablen im Projekt zu navigieren.

Ist das Init Programm in Abbildung 30 Seite 30 übernommen, geht es weiter mit dem zyklischen Programmtext. In den nachfolgenden Schritten wird die Verwendung der PLCopen Bausteine näher erläutert.

Schritt 3: Doppelklick auf „Cyclic.st“. (siehe **Schritt 1**)

MC_ReadStatus [1]

Dieser FB liefert den detaillierten Zustand der gerade aktiven Bewegung laut Zustandsdiagramm.

```
PROGRAM _CYCLIC

(*=====================================================================*
                          CONTROL SEQUENCE
=======================================================================*)

(* Status Informationen werden ausgelesen bevor die Schrittketten
    ausgeführt werden, dies garantiert kürzere Reaktionszeiten *)
[1]
(************************ fb_MC_ReadStatus ************************)
fb_MC_ReadStatus.Enable := NOT(fb_MC_ReadStatus.Error);
fb_MC_ReadStatus.Axis := udi_Achse1Obj;
fb_MC_ReadStatus();
t_AchsenCtrl.AxisState.Disabled             := fb_MC_ReadStatus.Disabled;
t_AchsenCtrl.AxisState.StandStill           := fb_MC_ReadStatus.StandStill;
t_AchsenCtrl.AxisState.Stopping             := fb_MC_ReadStatus.Stopping;
t_AchsenCtrl.AxisState.Homing               := fb_MC_ReadStatus.Homing;
t_AchsenCtrl.AxisState.DiscreteMotion       := fb_MC_ReadStatus.DiscreteMotion;
t_AchsenCtrl.AxisState.ContinuousMotion     := fb_MC_ReadStatus.ContinuousMotion;
t_AchsenCtrl.AxisState.SynchronizedMotion   := fb_MC_ReadStatus.SynchronizedMotion;
t_AchsenCtrl.AxisState.ErrorStop            := fb_MC_ReadStatus.Errorstop;
```

Abbildung 31: Erstellen eines Programms in ST - MC_ReadStatus

Zu Beginn des zyklischen Programms weisen wir unser Achsenobjekt aus Abbildung 30 Seite 30 dem Bausteineingang von MC_ReadStatus zu und rufen den FB gleich danach wieder auf ohne etwas zuzuweisen. In den nachfolgenden Programmzeilen werden die ausgelesenen Zustände der Achse unseren Variablen, die wir zuvor erstellt haben, zugewiesen.

Info: In der B&R Hilfe ist nachzulesen wie der jeweilige FB aufgebaut ist und was notwendig ist um diesen aufzurufen.

Info: In den nachfolgenden Abbildungen wird nicht näher auf die Programmierung, sondern nur auf die verwendeten Bausteine, eingegangen. Der Quellcode wird wie im aktuellen Projekt abgebildet und wurde von B&R übernommen.

MC_BR_ReadDriveStatus [2]

Mit diesem FB werden die Statusinformationen eines Antriebs ermittelt ob die Regler aktiv sind, QuickStop ausgelöst wurde usw..

MC_ReadActualPosition [3]

Dieser FB liefert die Position der Achse, solange der "Enable"-Eingang TRUE ist.

MC_ReadActualVelocity [4]

Dieser FB liefert die Geschwindigkeit der Achse, solange der "Enable"-Eingang TRUE ist.

MC_ReadAxisError [5]

Dieser Funktionsbaustein liefert die aktuelle Achsfehlernummer, solange "Enable" = TRUE ist. Zusätzlich dazu kann der Fehlertext zur aktuell angezeigten Fehlernummer ausgelesen werden.

```
[2](******************** fb_MC_BR_ReadDriveStatus ***********************)
fb_MC_BR_ReadDriveStatus.Enable          := NOT(fb_MC_BR_ReadDriveStatus.Error);
fb_MC_BR_ReadDriveStatus.Axis            := udi_Achse1Obj;
fb_MC_BR_ReadDriveStatus.AdrDriveStatus := ADR(t_AchsenCtrl.Status.DriveStatus);
fb_MC_BR_ReadDriveStatus();

[3](******************** fb_MC_ReadActualPosition *********************)
fb_MC_ReadActualPosition.Enable      := (NOT(fb_MC_ReadActualPosition.Error));
fb_MC_ReadActualPosition.Axis        := udi_Achse1Obj;
fb_MC_ReadActualPosition();
IF (fb_MC_ReadActualPosition.Valid = TRUE) THEN
    t_AchsenCtrl.Status.ActPosition := fb_MC_ReadActualPosition.Position;
END_IF

[4](******************** fb_MC_ReadActualVelocity *********************)
fb_MC_ReadActualVelocity.Enable      := (NOT(fb_MC_ReadActualVelocity.Error));
fb_MC_ReadActualVelocity.Axis        := udi_Achse1Obj;
fb_MC_ReadActualVelocity();
IF (fb_MC_ReadActualVelocity.Valid = TRUE) THEN
    t_AchsenCtrl.Status.ActVelocity := fb_MC_ReadActualVelocity.Velocity;
END_IF

[5](************************ fb_MC_ReadAxisError **********************)
fb_MC_ReadAxisError.Enable           := NOT(fb_MC_ReadAxisError.Error);
fb_MC_ReadAxisError.Axis             := udi_Achse1Obj;
fb_MC_ReadAxisError.DataAddress      := ADR(t_AchsenCtrl.Status.ErrorText);
fb_MC_ReadAxisError.DataLength       := SIZEOF(t_AchsenCtrl.Status.ErrorText);
fb_MC_ReadAxisError.DataObjectName   := 'acp10etxen';
fb_MC_ReadAxisError();
```

Abbildung 32: Erstellen eines Programms in ST - MC_BR_ReadDriveStatus und MC_ReadActualPosition

```
(*************** CHECK FOR GENERAL AXIS ERROR *****************)
    IF ((fb_MC_ReadAxisError.AxisErrorID <> 0) AND
        (fb_MC_ReadAxisError.Valid = TRUE)) THEN
        ui_AchsenSchritt := STATE_ERROR_AXIS;
(**************** CHECK IF POWER SHOULD BE OFF ****************)
    ELSIF  ((t_AchsenCtrl.Command.Power = FALSE) AND
            (fb_MC_ReadAxisError.Valid = TRUE)) THEN

        IF ((fb_MC_ReadStatus.Errorstop = TRUE) AND
            (fb_MC_ReadStatus.Valid = TRUE)) THEN

            ui_AchsenSchritt := STATE_ERROR_RESET;
        ELSE
            ui_AchsenSchritt := STATE_WAIT;
        END_IF
    END_IF

(* Wird das Stop Kommando außerhalb der CASE Struktur abgefragt
    verbessert dies die Reaktionszeiten *)

(******************CHECK FOR STOP COMMAND***********************)
    IF (t_AchsenCtrl.Command.Stop = TRUE) THEN
        IF ((ui_AchsenSchritt >= STATE_HOME) AND
            (ui_AchsenSchritt <= STATE_ERROR)) THEN
            (* Zurücksetzen aller positiven Flanken *)
            fb_MC_Home.Execute := FALSE;
            fb_MC_Stop.Execute := FALSE;
            fb_MC_MoveAbsolute.Execute := FALSE;
            fb_MC_ReadAxisError.Acknowledge := FALSE;
            fb_MC_Reset.Execute := FALSE;
            fb_MC_Halt.Execute := FALSE;

            (* Zurücksetzen der Kommandos *)
            t_AchsenCtrl.Command.Halt := FALSE;
            t_AchsenCtrl.Command.Home := FALSE;
            t_AchsenCtrl.Command.MoveAbsolute := FALSE;

            (* Springe in den nächsten Schritt *)
            ui_AchsenSchritt := STATE_STOP;
        END_IF
    END_IF
```

Abbildung 33: Erstellen eines Programms in ST

Das StopKommando sollte immer auch außerhalb der CASE Struktur abgefragt werden. Somit erreicht man kürzere Reaktionszeiten den Antrieb sicher zu stoppen im Notfall. Dasselbe gilt auch für die Abfrage ob eine Errormeldung ansteht.

MC_Power [6]

Dieser FB schaltet den Regler der Achse ein. Der Motor wird bestromt und hält die Position entsprechend der Regler-Parameter im Init-Parameter-Modul. Falls kein Fehler am Antrieb ansteht, wenn der FB aktiviert wird, ändert sich der Achszustand auf Standstill. Ansonsten wechselt die Achse in den Zustand Errorstop. Wird der "Enable"-Eingang auf FALSE gesetzt (auch bei aktiver Bewegung), dann wird eine Nothaltrampe gefahren und der Regler danach ausgeschaltet. Die Achse befindet sich danach wieder im Zustand Disabled.

```
CASE ui_AchsenSchritt OF

(******************** WAIT ************************)
    STATE_WAIT:  (* STATE: Wait *)
        IF (t_AchsenCtrl.Command.Power = TRUE) THEN
            ui_AchsenSchritt := STATE_POWER_ON;
        ELSE
            fb_MC_Power.Enable := FALSE;
        END_IF

        (* reset all FB execute inputs we use *)
        fb_MC_Home.Execute          := FALSE;
        fb_MC_Stop.Execute          := FALSE;
        fb_MC_MoveAbsolute.Execute  := FALSE;

        fb_MC_Halt.Execute              := FALSE;
        fb_MC_ReadAxisError.Acknowledge := FALSE;
        fb_MC_Reset.Execute             := FALSE;

        (* reset user commands *)
        t_AchsenCtrl.Command.Stop := FALSE;
        t_AchsenCtrl.Command.Halt := FALSE;
        t_AchsenCtrl.Command.Home := FALSE;

        t_AchsenCtrl.Command.MoveAbsolute := FALSE;

        t_AchsenCtrl.Status.ErrorID := 0;

(******************** POWER ON ***********************)
[6] STATE_POWER_ON:  (* STATE: Power on *)
        fb_MC_Power.Enable := TRUE;
        IF (fb_MC_Power.Status = TRUE) THEN
            ui_AchsenSchritt := STATE_READY;
        END_IF
        (* if a power error occured go to error state *)
        IF (fb_MC_Power.Error = TRUE) THEN
            t_AchsenCtrl.Status.ErrorID := fb_MC_Power.ErrorID;
            ui_AchsenSchritt := STATE_ERROR;
        END_IF
```

Abbildung 34: Erstellen eines Programms in ST – Schrittkette Teil 1

MC-Home [7]

Dieser FB führt die Referenzierung einer Achse durch. Je nachdem, ob der ausgewählte "HomingMode" eine Referenzfahrt voraussetzt, startet der FB diese auch.

```
(******************** READY **********************)
    STATE_READY:  (* STATE: Wartet auf Kommando *)
        IF (t_AchsenCtrl.Command.Home = TRUE) THEN
            t_AchsenCtrl.Command.Home := FALSE;
            ui_AchsenSchritt := STATE_HOME;

        ELSIF (t_AchsenCtrl.Command.Stop = TRUE) THEN
            ui_AchsenSchritt := STATE_STOP;

        ELSIF (t_AchsenCtrl.Command.MoveAbsolute = TRUE) THEN
            t_AchsenCtrl.Command.MoveAbsolute := FALSE;
            ui_AchsenSchritt := STATE_MOVE_ABSOLUTE;

        ELSIF (t_AchsenCtrl.Command.Halt = TRUE) THEN
            t_AchsenCtrl.Command.Halt := FALSE;
            ui_AchsenSchritt := STATE_HALT;
        END_IF

[7](******************** HOME **********************)
    STATE_HOME:  (* STATE: Startet die Referenzierung *)
        fb_MC_Home.Position     := t_AchsenCtrl.Parameter.HomePosition;
        fb_MC_Home.HomingMode   := t_AchsenCtrl.Parameter.HomeMode;
        fb_MC_Home.Execute      := TRUE;
        IF (fb_MC_Home.Done = TRUE) THEN
            fb_MC_Home.Execute  := FALSE;
            ui_AchsenSchritt    := STATE_READY;
        END_IF
        (* if a homing error occured go to error state *)
        IF (fb_MC_Home.Error = TRUE) THEN
            fb_MC_Home.Execute          := FALSE;
            t_AchsenCtrl.Status.ErrorID := fb_MC_Home.ErrorID;
            ui_AchsenSchritt            := STATE_ERROR;
        END_IF
```

Abbildung 35: Erstellen eines Programms in ST – Schrittkette Teil 2

Im STATE_READY wird auf die Kommandobefehle gewartet ob eine Bewegung ausgeführt werden soll, die Achse gestoppt oder angehalten werden soll usw.. Nach jedem Befehl wird in den jeweiligen Schritt gesprungen. Ist der Befehl ausgeführt worden in dem zugewiesenen Schritt, so wird wieder zurück in den Ready Schritt gesprungen.

MC_Halt [8]

Dieser Funktionsbaustein führt einen kontrollierten Bewegungshalt aus. Dieser kann jedoch durch einen Bewegungsstart abgebrochen werden.

MC_Stop [9]

Dieser Funktionsbaustein führt einen kontrollierten Bewegungsstopp durch und setzt die Achse in den Zustand Stopping.

[8]
```
(************************HALT_MOVEMENT*****************************)
    STATE_HALT:    (* STATE: Halt Bewegung *)
        fb_MC_Halt.Deceleration := t_AchsenCtrl.Parameter.Deceleration;
        fb_MC_Halt.Execute      := TRUE;
        IF (fb_MC_Halt.Done = TRUE) THEN
            fb_MC_Halt.Execute    := FALSE;
            ui_AchsenSchritt      := STATE_READY;
        END_IF
        (* Kontrolliere ob ein Fehler anliegt *)
        IF (fb_MC_Halt.Error = TRUE) THEN
            t_AchsenCtrl.Status.ErrorID := fb_MC_Halt.ErrorID;
            fb_MC_Halt.Execute          := FALSE;
            ui_AchsenSchritt            := STATE_ERROR;
        END_IF
```

[9]
```
(************************ STOP MOVEMENT ************************)
    STATE_STOP:  (* STATE: Stop Bewegung *)
        fb_MC_Stop.Deceleration := t_AchsenCtrl.Parameter.Deceleration;
        fb_MC_Stop.Execute      := TRUE;
        (* Wenn die Achse gestoppt wurde springe zurück in Ready *)
        IF ((fb_MC_Stop.Done = TRUE) AND
            (t_AchsenCtrl.Command.Stop = FALSE)) THEN
            fb_MC_Stop.Execute  := FALSE;
            ui_AchsenSchritt    := STATE_READY;
        END_IF
        (* Kontrolliere ob ein Fehler anliegt *)
        IF (fb_MC_Stop.Error = TRUE) THEN
            t_AchsenCtrl.Status.ErrorID := fb_MC_Stop.ErrorID;
            fb_MC_Stop.Execute          := FALSE;
            ui_AchsenSchritt            := STATE_ERROR;
        END_IF
```

Abbildung 36: Erstellen eines Programms in ST – Schrittkette Teil 3

MC_MoveAbsolute [10]

Dieser FB startet eine kontrollierte Bewegung auf eine vorgegebene absolute Position.

MC_ReadAxisError [11]

Dieser Funktionsbaustein liefert die aktuelle Achsfehlernummer, solange "Enable" = TRUE ist.

[10]

```
(******************** START ABSOLUTE MOVEMENT **********************)
    STATE_MOVE_ABSOLUTE:  (* STATE: Starte absolute Bewegung *)
        fb_MC_MoveAbsolute.Position      := t_AchsenCtrl.Parameter.Position;
        fb_MC_MoveAbsolute.Velocity      := t_AchsenCtrl.Parameter.Velocity;
        fb_MC_MoveAbsolute.Acceleration  := t_AchsenCtrl.Parameter.Acceleration;
        fb_MC_MoveAbsolute.Deceleration  := t_AchsenCtrl.Parameter.Deceleration;
        fb_MC_MoveAbsolute.Direction     := t_AchsenCtrl.Parameter.Direction;
        fb_MC_MoveAbsolute.Execute := TRUE;
        (* Überprüfe ob die Position erreicht ist *)
        IF (t_AchsenCtrl.Command.Halt) THEN
            t_AchsenCtrl.Command.Halt   := FALSE;
            fb_MC_MoveAbsolute.Execute  := FALSE;
            ui_AchsenSchritt := STATE_HALT;
        ELSIF (fb_MC_MoveAbsolute.Done = TRUE) THEN
            fb_MC_MoveAbsolute.Execute  := FALSE;
            ui_AchsenSchritt            := STATE_READY;
        END_IF
        (* Überprüfe ob ein Fehler anliegt *)
        IF (fb_MC_MoveAbsolute.Error = TRUE) THEN
            t_AchsenCtrl.Status.ErrorID  := fb_MC_MoveAbsolute.ErrorID;
            fb_MC_MoveAbsolute.Execute   := FALSE;
            ui_AchsenSchritt             := STATE_ERROR;
        END_IF
```

[11]

```
(******************** FB-ERROR OCCURED ************************)
    STATE_ERROR:  (* STATE: Error *)
        (* Überprüfe ob der FB einen Achsenfehler wiedergibt *)
        IF (fb_MC_ReadAxisError.AxisErrorCount<>0) THEN
            ui_AchsenSchritt := STATE_ERROR_AXIS;
        ELSE
            IF (t_AchsenCtrl.Command.ErrorAcknowledge = TRUE) THEN
                t_AchsenCtrl.Command.ErrorAcknowledge   := FALSE;
                t_AchsenCtrl.Status.ErrorID             := 0;
                (* Setze die Achse zurück wenn sie sich im Errorstop befindet *)
                IF ((fb_MC_ReadStatus.Errorstop = TRUE) AND
                    (fb_MC_ReadStatus.Valid = TRUE)) THEN
                    ui_AchsenSchritt := STATE_ERROR_RESET;
                ELSE
                    ui_AchsenSchritt := STATE_WAIT;
                END_IF
            END_IF
        END_IF
```

Abbildung 37: Erstellen eines Programms in ST – Schrittkette Teil 4

MC_Reset [12]

Der FB führt einen Übergang im Zustandsdiagramm von Errorstop auf Standstill oder Disabled aus, indem er alle internen achsbezogenen Fehler quittiert. Eine virtuelle Achse wird immer den Achszustand von Errorstop auf Standstill ändern. Der FB beeinflusst jedoch die Ausgänge der FB-Instanzen nicht.

```
(******************** AXIS-ERROR OCCURED *************************)
    STATE_ERROR_AXIS:   (* STATE: Axis Error *)
        IF (fb_MC_ReadAxisError.Valid = TRUE) THEN
            IF (fb_MC_ReadAxisError.AxisErrorID <> 0) THEN
                t_AchsenCtrl.Status.ErrorID := fb_MC_ReadAxisError.AxisErrorID;
            END_IF
            fb_MC_ReadAxisError.Acknowledge := FALSE;
            IF (t_AchsenCtrl.Command.ErrorAcknowledge = TRUE) THEN
                t_AchsenCtrl.Command.ErrorAcknowledge := FALSE;
                (* Quittieren von Achsenfehler *)
                IF (fb_MC_ReadAxisError.AxisErrorID <> 0) THEN
                    fb_MC_ReadAxisError.Acknowledge := TRUE;
                END_IF
            END_IF
            IF (fb_MC_ReadAxisError.AxisErrorCount = 0) THEN
                (* Setze die Achse zurück wenn sie sich im Errorstop befindet *)
                t_AchsenCtrl.Status.ErrorID := 0;
                IF ((fb_MC_ReadStatus.Errorstop = TRUE) AND
                    (fb_MC_ReadStatus.Valid = TRUE)) THEN
                    ui_AchsenSchritt := STATE_ERROR_RESET;
                ELSE
                    ui_AchsenSchritt := STATE_WAIT;
                END_IF
            END_IF
        END_IF
[12]
(******************** RESET DONE *************************)
    STATE_ERROR_RESET:   (* STATE: Warte bis der Fehler quittiert wurde *)
        fb_MC_Reset.Execute := TRUE;
        (* Zurücksetzen vom MC_Power falls zuvor im Fehler war *)
        IF (fb_MC_Power.Error = TRUE) THEN
            fb_MC_Power.Enable := FALSE;
        END_IF
        IF(fb_MC_Reset.Done = TRUE)THEN
            fb_MC_Reset.Execute := FALSE;
            ui_AchsenSchritt := STATE_WAIT;
        ELSIF(fb_MC_Reset.Error = TRUE) THEN
            fb_MC_Reset.Execute := FALSE;
            ui_AchsenSchritt := STATE_ERROR;
        END_IF

(******************** Schrittkette Ende *************************)
END_CASE
```

Abbildung 38: Erstellen eines Programms in ST – Schrittkette Teil 5

```
(*=========================================================================*
                          FUNKTIONSBLOCK AUFRUF
=========================================================================*)

(************************** fb_MC_Power ****************************)
fb_MC_Power.Axis := udi_Achse1Obj; (* Zeiger auf die Achse *)
fb_MC_Power();

(************************** fb_MC_Home *****************************)
fb_MC_Home.Axis := udi_Achse1Obj; (* Zeiger auf die Achse *)
fb_MC_Home();

(********************** fb_MC_MoveAbsolute ************************)
fb_MC_MoveAbsolute.Axis := udi_Achse1Obj; (* Zeiger auf die Achse *)
fb_MC_MoveAbsolute();

(************************** fb_MC_Stop *****************************)
fb_MC_Stop.Axis := udi_Achse1Obj; (* Zeiger auf die Achse *)
fb_MC_Stop();

(**************************fb_MC_Halt*******************************)
fb_MC_Halt.Axis := udi_Achse1Obj; (* Zeiger auf die Achse *)
fb_MC_Halt();

(************************** fb_MC_Reset ****************************)
fb_MC_Reset.Axis := udi_Achse1Obj; (* Zeiger auf die Achse *)
fb_MC_Reset();

END_PROGRAM
```

Abbildung 39: Erstellen eines Programms in ST – Funktionsblock Aufruf

Am Ende eines jeden Programms werden die Funktionsblöcke noch einmal aufgerufen. Das programm ist jetzt fertig geschrieben.

Abbildung 40

Das Projekt wird jetzt noch einmal gespeichert und danach „Neu kompiliert“. Im nächsten Kapitel wird eine Simulation des Projekts angelegt.

8 Projektsimulation

In diesem Kapitel wird beschrieben wie man aus seinem aktuellen Projekt eine Simulation erstellt.

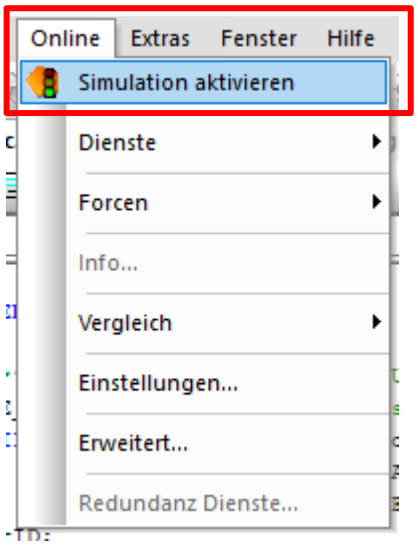


Abbildung 41: Projektsimulation - Simulation aktivieren

Schritt 1: Reiter „Online“ ⇨ „Simulation aktivieren“ anklicken.

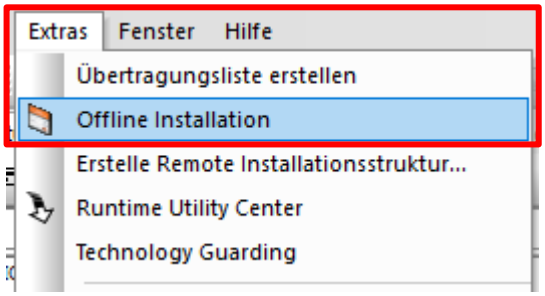


Abbildung 42: Projektsimulation - Offline Installation

Schritt 2: Reiter „Extras“ ⇨ „Offline Installation“ anklicken.

Info: Das Projekt wird nochmal automatisch kompiliert. Es öffnet sich ein neues Fenster. Dort wird die Installation vorgenommen.

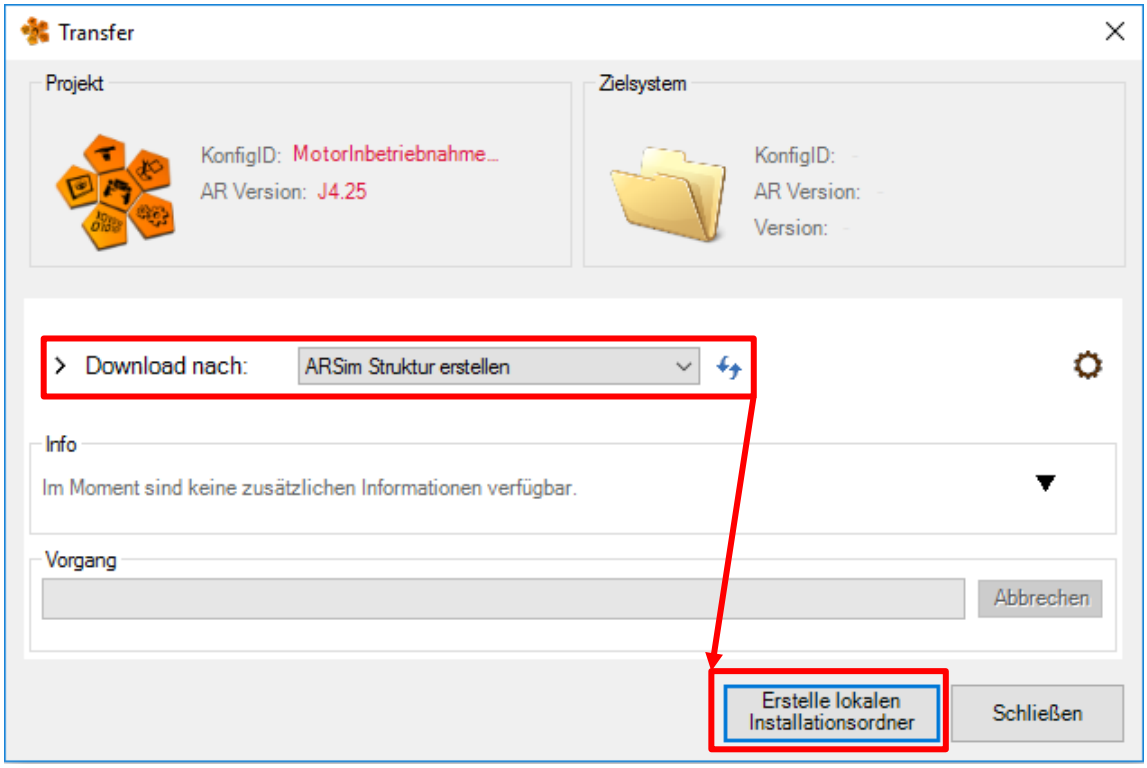


Abbildung 43: Projektsimulation - ARsim Struktur erstellen

Schritt 3: Download nach: „Arsim Struktur erstellen“ auswählen. Danach bestätigen.

Info: Es wird nun eine Arsim Struktur des Projektes angelegt. Danach ist es möglich z.B. Hardware und Programm zu steuern.

Abbildung 44: Projektsimulation - Firewall

Es ist notwendig der die Arsim Struktur in der Firewall einen Zugriff zu geben.

Nach erfolgreicher Installation kann das Transferfenster (siehe Abbildung 43, Seite 41) wieder geschlossen werden.

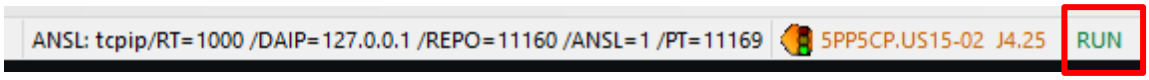


Abbildung 45: Projektsimulation - Run Modus

Die Steuerung sollte sich nun im Run-Modus befinden.

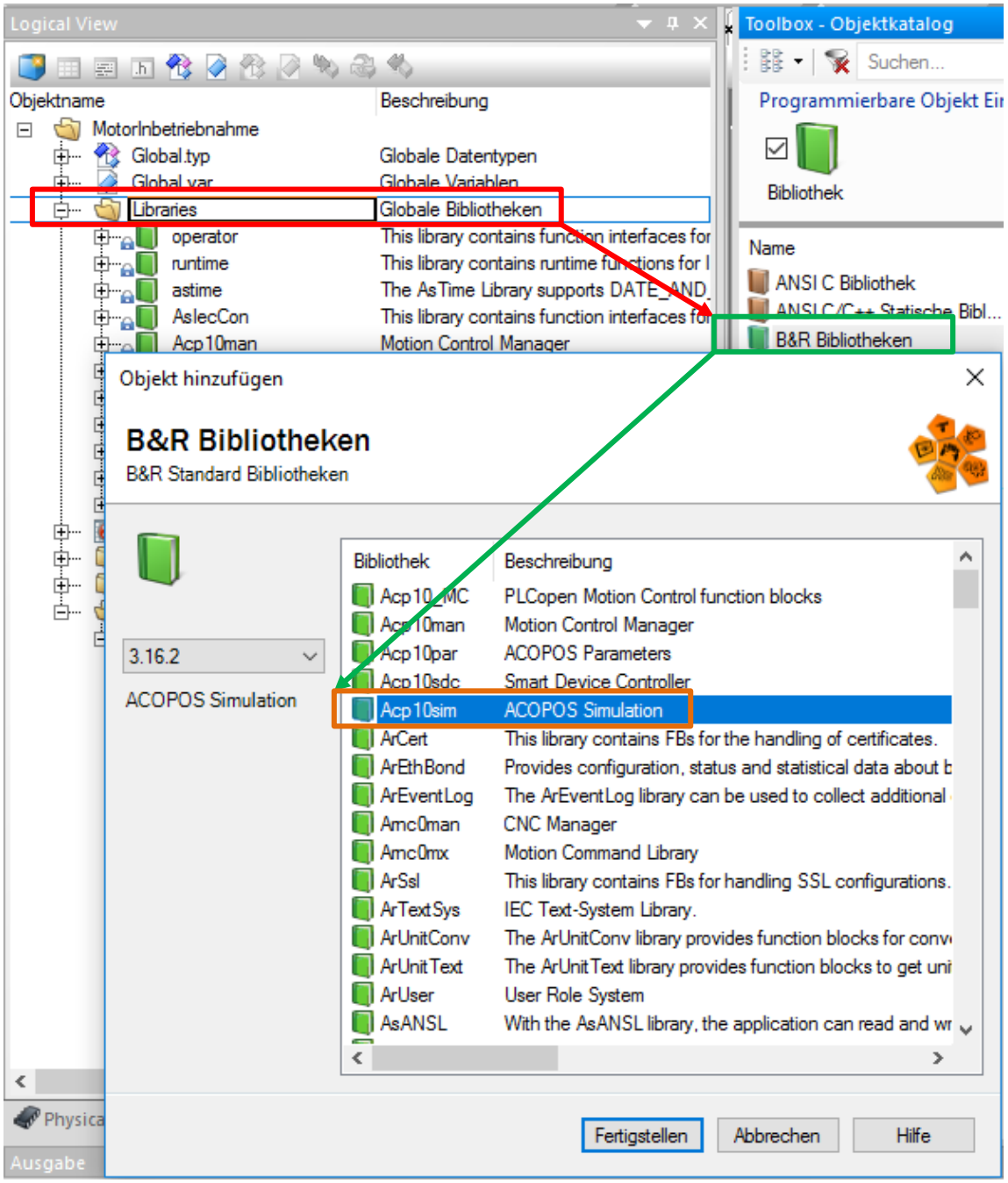


Abbildung 46: Projektsimulation - ACOPOSsim Bibliothek

Schritt 4: Acp10sim in die Projektbibliothek einfügen.

Info: „Libraries“ in der Projektstruktur anwählen ⇨ im Kontextmenü der Toolbox „B&R Bibliotheken“ doppelklicken ⇨ „Acp10sim“ auswählen und einfügen.

9 Monitormodus – Steuern von Variablen zur Laufzeit

In diesem Kapitel wird beschrieben wie man die deklarierten Variablen zur Laufzeit der Simulation steuern kann.

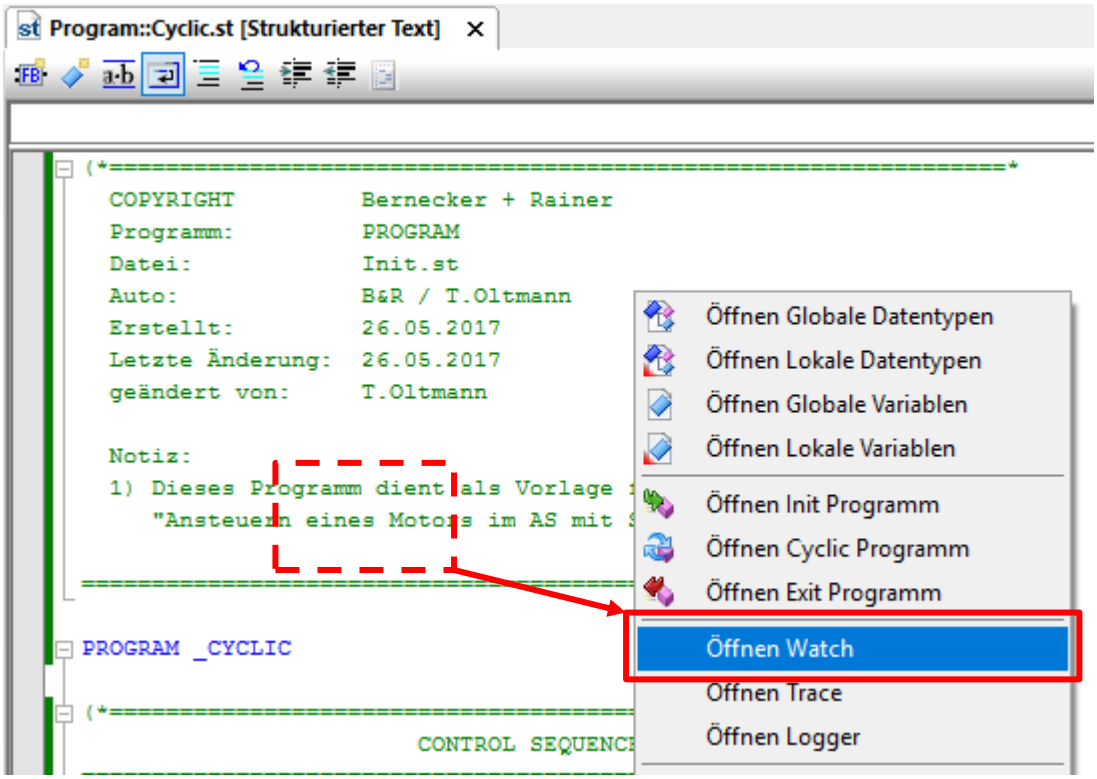


Abbildung 47: Monitormodus - Aktivieren

Schritt 1: Das „Cyclic.st“ Programm öffnen (siehe **Seite 31 Schritt 3**)

Schritt 2: Rechtsklick in das „Cyclic.st“ Fenster ⇨ „Öffnen Watch“ anklicken.

Info: Es öffnet sich ein neues Fenster. In diesem Fenster lassen sich deklarierte Variablen aus dem Projekt einfügen und steuern.

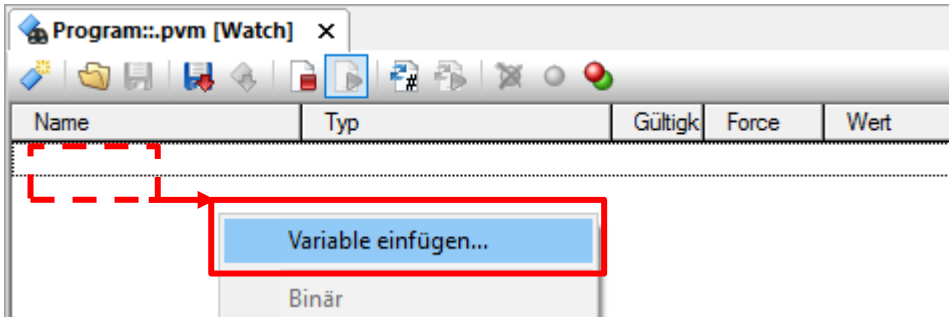


Abbildung 48: Monitormodus - Variablen einfügen

Schritt 3: Rechtsklick in das Fenster ⇨ „Variable einfügen…“.

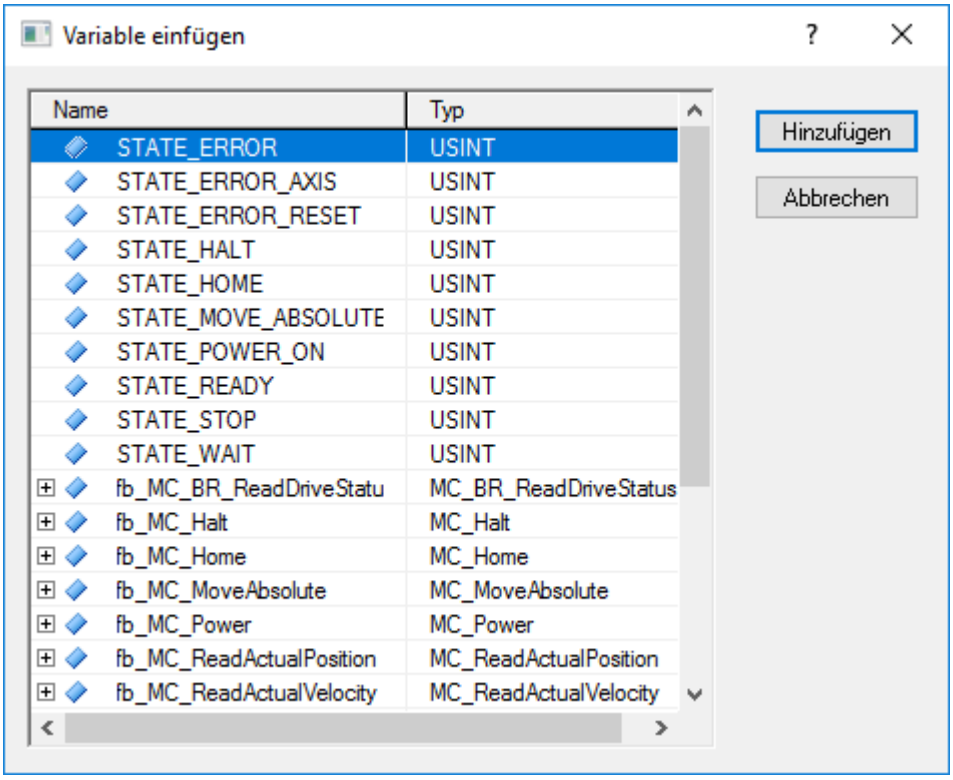


Abbildung 49: Monitormodus - Variablen auswählen

Schritt 4: Die gewünschten Variablen auswählen und „Hinzufügen“. (siehe Abbildung 50, Seite 46)

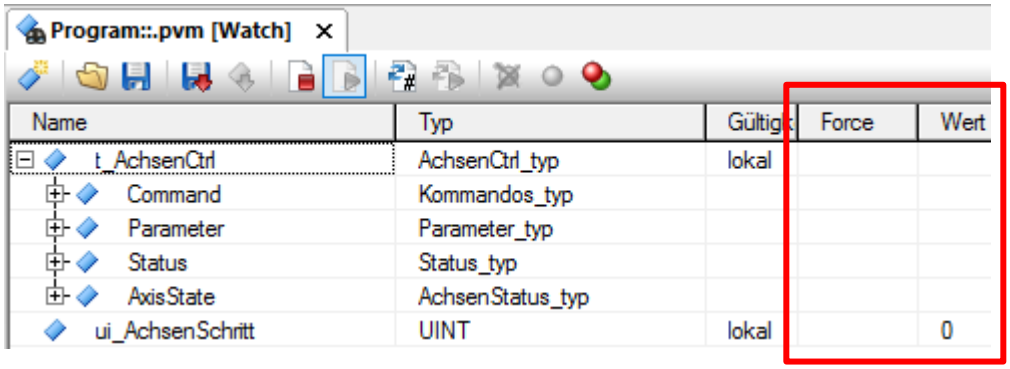


Abbildung 50: Monitormodus

t_AchsenCtrl Diese Variable umfasst alle Strukturvariablen die angelegt worden sind.

Command Über diesen Strukturtypen lässt sich der Motor steuern.

Parameter In der Parameterstruktur lassen sich Parameter zur Laufzeit ändern.

Status Zeigt den Status der Achse an.

AxisState Zeigt die Fehlerzustände der Achse an.

ui_AchsenSchritt Hier wird der aktuelle Schritt der Schrittkette wiedergegeben.

Über die Spalte „Wert/ Force“ können die Zustände der Variablen geändert werden.

Weiterführende Informationen bietet die B&R Hilfe. Nachfolgend sind die wichtigsten Hilfeseiten die sich auf dieses Kapitel beziehen aufgelistet:

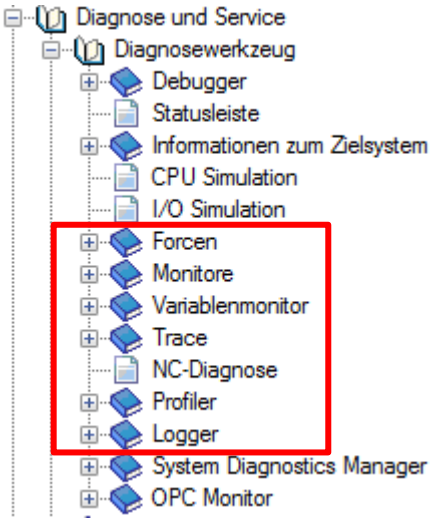


Abbildung 51: Monitormodus - B&R Hilfeseite

10 PLCopen Funktionsbausteine

In diesem Kapitel werden die notwendigen PLCopen Bausteine erläutert die für die Inbetriebnahme sowie zum ansteuern eines Motors notwendig sind. Die Erklärungen wurden der B&R Hilfe entnommen.

Zustandsdiagramm

„Das folgende Diagramm zeigt einen Standard, der das Verhalten der Achse auf oberster Ebene definiert, wenn mehrere Funktionsblöcke "gleichzeitig" aktiv sind. Diese Übersicht von Bewegungsarten ist nützlich, um kompliziertere Bewegungsabläufe zu erstellen und Fehlersituationen im Programm zu verarbeiten. Die Grundregel ist, dass Motion-Befehle immer sequenziell abgearbeitet werden, sogar wenn die Steuerung die Möglichkeit echter paralleler Prozessverarbeitung besitzt. Diese Befehle wirken auf das Zustandsdiagramm der Achse. Die Achse befindet sich immer in einem der definierten Zustände (siehe Zustandsdiagramm). Jeder Motion-Befehl ist ein Übergang, der den Zustand der Achse ändert und als Konsequenz die Art der Berechnung der aktuellen Bewegung beeinflusst. Das Diagramm bezieht sich auf eine einzelne Achse. Die Mehrachs-Funktionsblöcke MC_GearIn und MC_Phasing können aus Sicht des Zustandsdiagramms als mehrere Einzelachsen in bestimmten Zuständen betrachtet werden, z.B.: Der CAM-Master kann im Zustand Continuous Motion sein, der zugehörige Slave ist im Zustand Synchronized Motion. Die Kopplung einer Slave-Achse an eine Master-Achse hat keinen Einfluss auf die Masterachse. Die FB, die nicht im Zustandsdiagramm erscheinen, beeinflussen den Achszustand nicht. Der aktuelle Achszustand kann mit dem FB MC_ReadStatus ermittelt werden.“

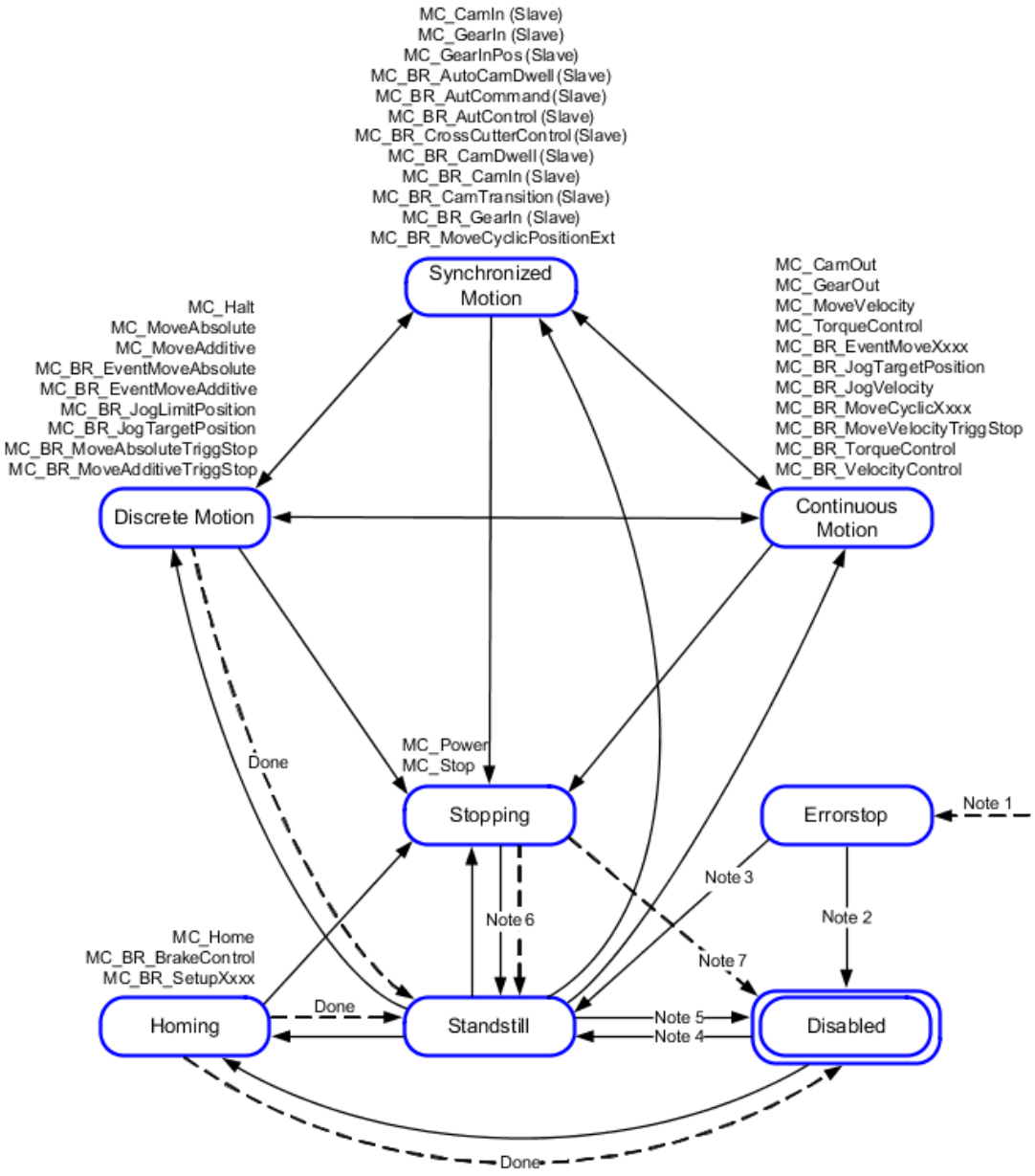


Abbildung 52: PLCopen - Zustandsdiagramm

Legende

Note 1 Aus jedem Zustand. Es ist ein Fehler auf der Achse aufgetreten.

Note 2 MC_Reset und MC_Power.Status = FALSE

Note 3 MC_Reset und MC_Power.Status = TRUE und MC_Power.Enable = TRUE

Note 4 MC_Power.Enable = TRUE und MC_Power.Status = TRUE

Note 5 MC_Power.Enable = FALSE

Note 6 MC_Stop.Done = TRUE und MC_Stop.Execute = FALSE

Note 7 MC_Power.Enable = FALSE während Bewegung, nach Stoprampe

PLCopen-Achszustände

Zustand Disabled

- Dieser Zustand zeigt an, dass die interne Initialisierung erfolgreich durchgeführt wurde und der Regler der Achse ausgeschaltet ist.
- Bei virtuellen Achsen existiert dieser Zustand nicht, diese befinden sich nach erfolgreicher Initialisierung im Zustand Standstill.

Zustand Standstill

- Dieser Zustand zeigt an, dass der Regler der Achse eingeschaltet ist und daher eine Bewegung gestartet werden kann.
- Bei virtuellen Achsen zeigt dieser Zustand an, dass eine Bewegung gestartet werden kann. Virtuelle Achsen besitzen keinen Regler.

Zustand Homing

- Dieser Zustand zeigt an, dass die Achse referenziert wird, ein Setup-Funktionsblock oder ein Haltebremsentest aktiv ist. Nach Beendigung des Vorgangs wird wieder automatisch in den Zustand Standstill oder Disabled gewechselt.
- In diesem Zustand kann die Verweildauer in Abhängigkeit des Referenziermodus stark variieren.

Zustand Errorstop

- Dieser Zustand zeigt an, dass ein schwerwiegender Achsfehler aufgetreten ist, durch den die Bewegung der Achse gestoppt wird.
- Mit den FBs MC_ReadAxisError oder MC_BR_ReadAxisError lässt sich die Fehlernummer sowie der Fehlertext und damit die Ursache des Zustandwechsels ermitteln.
- In diesem PLCopen-Achszustand ist das Starten einer Bewegung nicht möglich.
- Um diesen Zustand wieder zu verlassen, muss der FB MC_Reset aufgerufen werden. Abhängig vom Reglerzustand wird dann entweder in den Zustand Disabled (Regler ist aus) oder Standstill (Regler ist ein) gewechselt.
- Wird MC_Reset schon während einer aktiven Nothalt-Rampe aufgerufen, bleibt der Zustand Errorstop bis zum Ende der Bremsrampe aktiv. Danach erfolgt der Zustandswechsel. MC_Reset meldet während der Bremsrampe "Busy", erst an deren Ende "Done".

Folgende Ereignisse sind u. a. mögliche Gründe für einen Wechsel in den Zustand Errorstop:

Während einer aktiven Bewegung:

- Quickstop-Eingang der Achse wird aktiviert
- Schleppfehler
- Fehler in der Zwischenkreisspannung (Überspannung, Ausfall der Versorgung, usw.)
- Geberfehler
- HW-Endschalter in Bewegungsrichtung wird belegt
- SW-Endlage wird erreicht
- Ausfall der Kommunikation zum Antrieb
- Übertemperatur im Motor, Bremswiderstand, Leistungshalbleiter, etc.
- Fehlen der 24 V am Eingang Enable

Der Regler kann aufgrund eines Antriebsfehlers nicht eingeschaltet werden:

- Beide Endschalter sind belegt
- Motorparameter fehlen
- Geberfehler (keine Gebersignale, Leitungsstörung, usw.)
- Haltebremse: Unterspannung/Strom
- Haltebremse: Ansteuerungssignal ein und Ausgangstatus aus
- Haltebremse: Bremsausgang aktiv, aber keine Bremse in Motordaten
- Haltebremse: Ansteuerungssignal aus und Ausgangstatus ein
- PHASING_MODE ist ungültig
- Fehler am Motoranschluss X5 des Leistungsteils (kein Stromfluss, Erdschluss, Kurzschluss, usw.)
- Fehler beim Anschluss auf der Leistungsversorgung X3 (Zwischenkreisspannung außerhalb Toleranz)

Ein Event-Move-FB ist aktiviert und wartet auf das Startereignis:

- Startereignis tritt ein und HW-Endschalter in Bewegungsrichtung ist belegt

Zustand Stopping

- Dieser Zustand zeigt an, dass die Achse per Kommando gestoppt wird.
- In diesem PLCopen-Achszustand ist das Starten einer Bewegung nicht möglich.
- Wird der Zustandswechsel nach Stopping durch MC_Stop ausgelöst, bleibt die Achse solange in diesem Zustand bis der Eingang "Execute" auf FALSE gesetzt wird.

Zustand Discrete Motion

- Dieser Zustand zeigt an, dass die Achse eine Bewegung durchführt, die beim Erreichen einer bestimmten Position beendet wird.
- Bei einem Abbruch mit MC_Halt wird in diesem Zustand bis zum Erreichen des Stillstands verweilt.

Zustand Continuous Motion

- Dieser Zustand zeigt an, dass die Achse eine Bewegung durchführt, die solange fortgesetzt wird, bis sie von einem anderen Kommando gestoppt oder abgebrochen wird.
- Die Achse kann auch mit Geschwindigkeit "0" in diesem Zustand verharren.

Zustand Synchronized Motion

- Dieser Zustand zeigt an, dass die Achse (Slave) an eine andere Achse (Master) gekoppelt ist.
- Stillstand der Achse (keine Bewegung) in diesem Zustand zeigt an, dass die andere Achse (Master)
- keine Bewegung durchführt oder
- kein Startsignal (auf Slave bzw. FB) gegeben wird oder
- sich aus der aktuellen Kurvenscheibe bzw. den aktuellen Ausgleichsparametern keine Bewegung ergibt.

Nachfolgend werden die im Projekt verwendeten Bausteine nochmal aufgeführt:

MC_Power

Dieser FB schaltet den Regler der Achse ein. Der Motor wird bestromt und hält die Position entsprechend der Regler-Parameter im Init-Parameter-Modul. Falls kein Fehler am Antrieb ansteht, wenn der FB aktiviert wird, ändert sich der Achszustand auf Standstill. Ansonsten wechselt die Achse in den Zustand Errorstop. Wird der "Enable"-Eingang auf FALSE gesetzt (auch bei aktiver Bewegung), dann wird eine Nothaltrampe gefahren und der Regler danach ausgeschaltet. Die Achse befindet sich danach wieder im Zustand Disabled. Der "Status"-Ausgang zeigt an, ob der Motor bestromt ist; TRUE bedeutet Leistungsendstufe ist aktiv; FALSE bedeutet Leistungsendstufe ist inaktiv. Dieser FB kann auch für virtuelle Achsen aufgerufen werden. Er verhält sich wie für reelle Achsen und liefert TRUE am "Status"-Ausgang, jedoch ist der Aufruf dieses FBs für virtuelle Achsen nicht erforderlich.

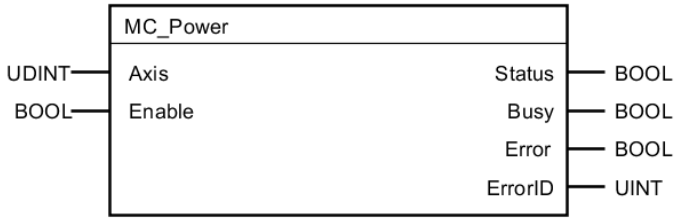


Abbildung 53: PLCopen Funktionsbausteine - MC_Power

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Enable	BOOL	FB ist aktiv solange der Eingang gesetzt ist.
OUT	Status	BOOL	Tatsächlicher Zustand der Leistungsendstufe
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer

MC_Power überprüft, ob die Initialisierung der Achse abgeschlossen und der Antrieb bereit ist. Antrieb nicht bereit. Diese Wartezeit von 7 Sekunden wurde eingeführt, um dem Zwischenkreis die Möglichkeit zu geben, sich vollständig aufzuladen. Das bedeutet, dass der FB erst nach 7 Sekunden einen Fehler meldet, falls der Regler noch nicht bereit ist. Wenn der Regler bereit ist und nachdem das Kommando zum Einschalten des Reglers übertragen worden ist, wartet der FB 2 Sekunden, ob der Antrieb den Reglerstatus "Eingeschaltet" zurückliefert. Falls innerhalb dieser Zeit der Status nicht gesetzt wird ist der Antrieb im Fehlerzustand. Diese Wartezeit von 2 Sekunden wurde eingeführt, damit der Antrieb gegebenenfalls Zeit für das Einphasen (bei Motoren mit Inkrementalgebern) oder zum Öffnen einer vorhandenen Haltebremse hat.

MC_Home

Dieser FB führt die Referenzierung einer Achse durch. Je nachdem, ob der ausgewählte "HomingMode" eine Referenzfahrt voraussetzt, startet der FB diese auch.

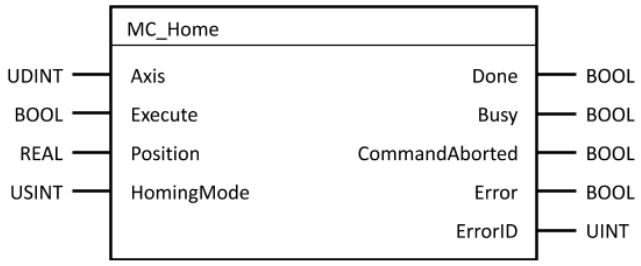


Abbildung 54: PLCopen Funktionsbausteine - MC_Home

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Execute	BOOL	Abarbeitung des FB wird bei steigender Flanke am Eingang gestartet.
IN	Position	REAL	Absolutposition nach dem das Referenzsignal erkannt wurde [PLCopen-Einh.] **Hinweis:** Bei den Modi mcHOME_RESTORE_POS und mcHOME_AXIS_REF wird die Position nicht von diesem Eingang übernommen.
IN	HomingMode	USINT	Referenzier Modi: mcHOME_DEFAULT (alle Parameter bis auf "Position" werden aus dem Init-Parameter-Modul übernommen) mcHOME_ABS_SWITCH mcHOME_SWITCH_GATE mcHOME_LIMIT_SWITCH mcHOME_REF_PULSE mcHOME_DIRECT mcHOME_ABSOLUTE (Offset für den Absolute Encoder) mcHOME_ABSOLUTE_CORR (Offset für den Absolute Encoder mit Zählbereichskorrektur) mcHOME_DCM (Abstandskodierte Referenzmarken) mcHOME_DCM_CORR (Abstandskodierte Referenzmarken mit Zählbereichskorrektur) mcHOME_RESTORE_POS (Position wiederherstellen aus Permanent-Memory, siehe MC_BR_InitEndlessPosition) mcHOME_AXIS_REF (Referenzieren mit den Parametern aus der Achsstruktur) (ab V 2.210) mcHOME_BLOCK_TORQUE (Referenzieren auf mechanische Begrenzung) (ab V 2.360) mcHOME_BLOCK_DS (Referenzieren auf mechanische Begrenzung) (ab V 2.360)
OUT	Done	BOOL	Abarbeitung erfolgreich. FB ist fertig.
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	CommandAborted	BOOL	FB wurde durch einen anderen FB abgebrochen.
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer

MC_Stop

Dieser Funktionsbaustein führt einen kontrollierten Bewegungsstopp durch und setzt die Achse in den Zustand Stopping.

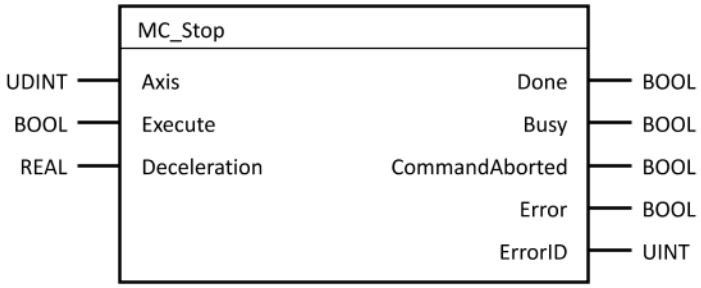


Abbildung 55: PLCopen Funktionsbausteine - MC_Stop

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Execute	BOOL	Abarbeitung des FB wird bei steigender Flanke am Eingang gestartet.
IN	Deceleration	REAL	Maximale Verzögerung [PLCopen-Einh./s²]
OUT	Done	BOOL	Kommando ausgeführt Geschwindigkeit 0 wurde erreicht
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	CommandAborted	BOOL	FB wurde durch einen anderen FB abgebrochen. Stopp wurde durch Power OFF abgebrochen
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer

Nach einer steigenden Flanke am Eingang "Execute" ruft der FB die ncaction (ncMOVE, ncSTOP) auf. Dabei wird ein spezieller Modus verwendet, der es erlaubt, die Verzögerung zum Antrieb zu übertragen. Dieser Vorgang dauert mindestens 3 POWERLINK-Zyklen. MC_Stop hat eine sehr hohe Priorität und bricht jede laufende Parameterübertragung ab.

MC_Halt

Dieser Funktionsbaustein führt einen kontrollierten Bewegungshalt aus. Dieser kann jedoch durch einen Bewegungsstart abgebrochen werden.

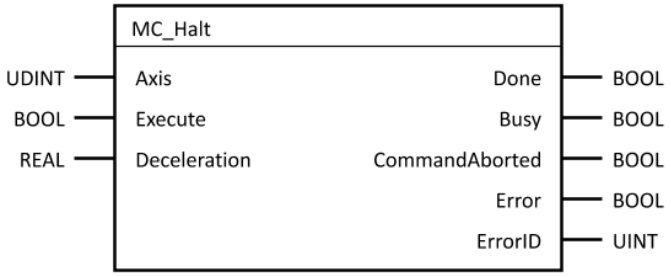


Abbildung 56: PLCopen Funktionsbausteine - MC_Halt

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Execute	BOOL	Abarbeitung des FB wird bei steigender Flanke am Eingang gestartet.
IN	Deceleration	REAL	Maximale Verzögerung [PLCopen-Einh./s²]
OUT	Done	BOOL	Abarbeitung erfolgreich. FB ist fertig. Geschwindigkeit 0 wurde erreicht
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	CommandAborted	BOOL	FB wurde durch einen anderen FB abgebrochen.
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer

Dieser Funktionsbaustein führt einen kontrollierten Bewegungshalt aus. Dieser kann jedoch durch einen Bewegungsstart abgebrochen werden.Nach einer steigenden Flanke am Eingang "Execute" werden alle Parameter, die nötig sind um eine Bewegung einer Achse abzubrechen, übertragen. Nach erfolgreicher Übertragung wechselt die Achse in den Zustand Discrete Motion. Wenn der Ausgang "Done" gesetzt wird, wechselt die Achse in den Zustand Standstill. Wird ein anderer Bewegungs-Funktionsbaustein gestartet, während MC_Halt aktiv ist, wird dieser abgebrochen und der Ausgang "CommandAborted" wird gesetzt. Aus diesem Grund kann mit dem Funktionsbaustein MC_Halt nicht gewährleistet werden, dass die Achse sicher zum Stillstand gebracht wird. Wird z. B. MC_MoveVelocity gestartet, während MC_Halt aktiv ist, wird dieser Funktionsbaustein abgebrochen und MC_MoveVelocity ist aktiv.

MC_ReadAxisError

Dieser Funktionsbaustein liefert die aktuelle Achsfehlernummer, solange "Enable" = TRUE ist.

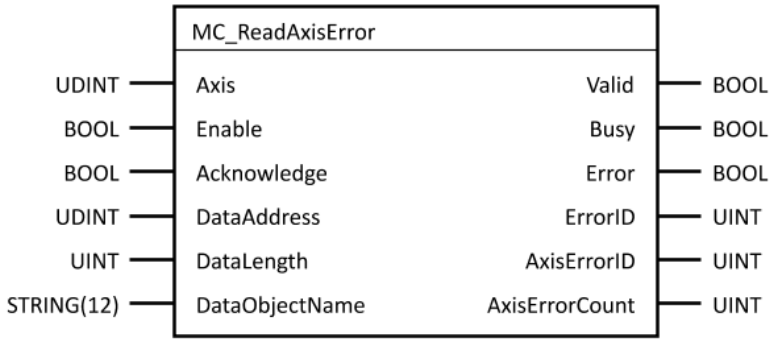


Abbildung 57: PLCopen Funktionsbausteine - MC_ReadAxisError

Klasse	I/O	Parameter	Datentyp	Beschreibung
B	IN	Axis	UDINT	Achsreferenz
B	IN	Enable	BOOL	Liest die interne Achsfehlernummer ständig, wenn eingeschaltet.
V	IN	Acknowledge	BOOL	Quittiert den aktuell angezeigten Achsfehler.
V	IN	DataAddress	UDINT	Adresse für Fehlertextstring (Anwendervariable) **Hinweis:** Pro Fehlernummer können bis zu vier Ausgabetexte ermittelt werden. Im NC-INIT-Parametermodul ist standardmäßig eine Zeilenlänge ("columns") von 80 Zeichen definiert. Der Fehlertextstring sollte deshalb standardmäßig aus vier Zeilen mit je 80 Zeichen bestehen.
V	IN	DataLength	UINT	Länge des Fehlertextstring (Anwendervariable)
V	IN	DataObjectName	STRING[12]	Name des Fehlertextmoduls
B	OUT	Valid	BOOL	Der FB hat den aktuellen Achsfehler erfolgreich gelesen.
E	OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
B	OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
B	OUT	ErrorID	UINT	Fehlernummer
B	OUT	AxisErrorID	UINT	Achsfehlernummer
V	OUT	AxisErrorCount	UINT	Anzahl der nicht quittierten Achsfehler

Dieser Funktionsbaustein liefert die aktuelle Achsfehlernummer, solange "Enable" = TRUE ist. Zusätzlich dazu kann der Fehlertext zur aktuell angezeigten Fehlernummer ausgelesen werden. Am Eingang "DataAddress" wird dazu die Adresse des Fehlertextspeichers angeschlossen, der Eingang "DataObjectName" dient zur Auswahl des Fehlertextmoduls (Sprachumschaltung). Ist der Eingang "DataObjectName" nicht belegt (=0), so wird die Einstellung aus dem NC-INIT-Parameter-Modul der Achse verwendet. Im NC-Init-Parameter-Modul kann auch die Formatierung des Fehlertextes vorgenommen werden.

MC_ReadStatus

Dieser FB liefert den detaillierten Zustand der gerade aktiven Bewegung laut Zustandsdiagramm.

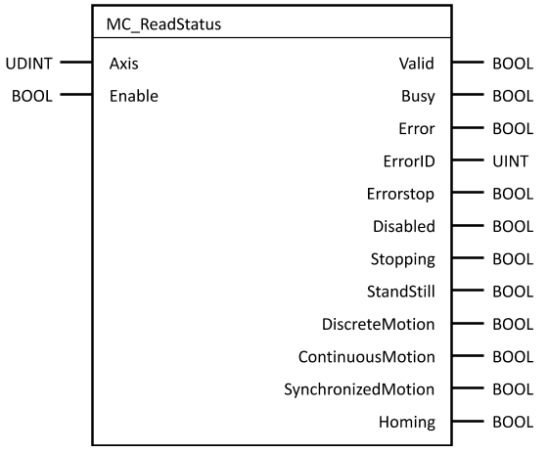


Abbildung 58: PLCopen Funktionsbausteine - MC_ReadStatus

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Enable	BOOL	Liest den Achs-Zustand ständig, wenn eingeschaltet
OUT	Valid	BOOL	Die Ausgangswerte des FB können verwendet werden. Die Ausgänge sind gültig.
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer
OUT	Errorstop	BOOL	Ein Fehler ist aufgetreten. Mit MC_Reset kann dieser quittiert werden. Wenn alle Fehler quittiert sind, wechselt der Achszustand auf **Disabled** oder **Standstill**.
OUT	Disabled	BOOL	MC_Power hat den Regler der Achse noch nicht aktiviert oder ein Fehler wurde durch MC_Reset quittiert und der Regler ist noch ausgeschaltet. Siehe Zustandsdiagramm.
OUT	Stopping	BOOL	MC_Stop ist aktiv. Siehe Zustandsdiagramm.
OUT	StandStill	BOOL	Es ist keine Bewegung aktiv. Die Achse wechselt in diesen Zustand, wenn MC_Power den Regler erfolgreich eingeschaltet hat. Siehe Zustandsdiagramm.
OUT	DiscreteMotion	BOOL	Aufgrund eines der folgenden FBs ist eine Bewegung aktiv: MC_MoveAbsolute MC_MoveAdditive MC_BR_MoveAbsoluteTriggStop MC_BR_MoveAdditiveTriggStop MC_BR_EventMoveAbsolute ("Mode" = mcONCE) MC_BR_EventMoveAdditive ("Mode" = mcONCE) MC_Halt Siehe Zustandsdiagramm.
OUT	ContinuousMotion	BOOL	Aufgrund eines der folgenden FBs ist eine Bewegung aktiv: MC_MoveVelocity MC_BR_MoveVelocityTriggStop MC_BR_EventMoveAbsolute ("Mode" = mcCYCLIC oder mcCYCLIC_ALL_EVENTS) MC_BR_EventMoveAdditive ("Mode" = mcCYCLIC oder mcCYCLIC_ALL_EVENTS) MC_BR_EventMoveVelocity MC_BR_MoveCyclicPosition MC_BR_MoveCyclicVelocity MC_BR_VelocityControl Die Achse wechselt ebenfalls in diesen Zustand, nachdem einer der nachfolgenden FBs aufgerufen wird: MC_CamOut MC_GearOut Siehe Zustandsdiagramm.
OUT	SynchronizedMotion	BOOL	Aufgrund eines der folgenden FBs ist eine Bewegung aktiv: MC_CamIn MC_GearIn MC_GearInPos MC_BR_AutControl MC_BR_CamDwell MC_BR_AutoCamDwell MC_BR_CamTransition
OUT	Homing	BOOL	MC_Home ist aktiv. Siehe Zustandsdiagramm.

MC_BR_ReadDriveStatus

Mit diesem FB werden die Statusinformationen eines Antriebs ermittelt. Die Informationen sind innerhalb einer Struktur vom Datentyp MC_DRIVESTATUS_TYP als boolsche Werte verfügbar und können somit ohne Probleme im Kontaktplan verknüpft werden. Die Adresse der Struktur muss dem FB über den Eingang "AdrDriveStatus" übergeben werden. Die Werte innerhalb der Struktur sind gültig, wenn der FB-Ausgang "Valid" auf TRUE gesetzt ist. Der Ausgang "Busy" bleibt so lange gesetzt, bis der Eingang "Enable" zurückgesetzt wird.

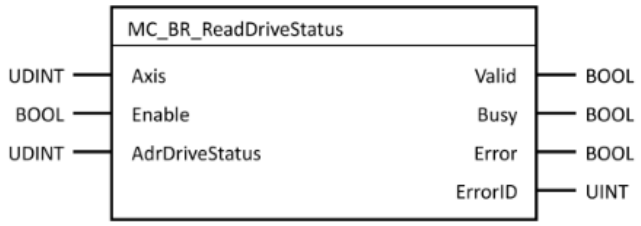


Abbildung 59: PLCopen Funktionsbausteine - MC_BR_ReadDriveStatus

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Enable	BOOL	Liest den Achs-Zustand ständig, wenn eingeschaltet
IN	AdrDriveStatus	UDINT	Adresse einer Struktur mit Datentyp MC_DRIVESTATUS_TYP
OUT	Valid	BOOL	Die Ausgänge sind gültig.
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer

Der Ausgang "Valid" wird gesetzt, sobald die Netzwerkinitialisierung (network.init) und die globale Initialisierung (global.init) abgeschlossen sind.

MC_ReadActualPosition

Dieser FB liefert die Position der Achse, solange der "Enable"-Eingang TRUE ist. Der "Valid"-Ausgang ist TRUE, wenn der Ausgang "Position" gültig ist.

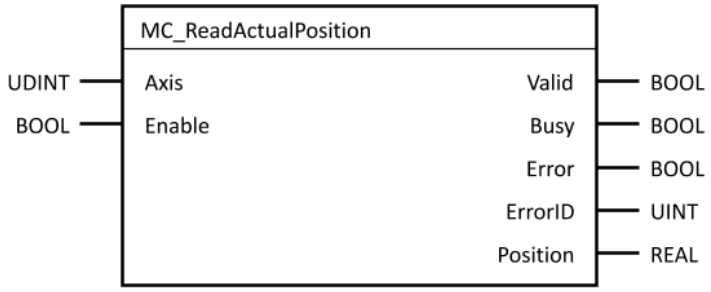


Abbildung 60: PLCopen Funktionsbausteine - MC_ReadAcutalPosition

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Enable	BOOL	Liest die Achsposition, wenn eingeschaltet
OUT	Valid	BOOL	Die Ausgangswerte des FB können verwendet werden.
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer
OUT	Position	REAL	Achsposition [Einh.]

In Verbindung mit aktivem MC_TorqueControl oder MC_BR_TorqueControl wird bei den FBs MC_ReadActualPosition und MC_ReadActualVelocity am Ausgang "Position" bzw. "Velocity" der Geberwert anstelle des Sollwerts angezeigt.

MC_ReadActualVelocity

Dieser FB liefert die Geschwindigkeit der Achse, solange der "Enable"-Eingang TRUE ist. Der "Valid"-Ausgang ist TRUE, wenn der Ausgang "Velocity" gültig ist. Der Ausgang "Error" zeigt an, dass das Lesen der Achsgeschwindigkeit nicht möglich ist.

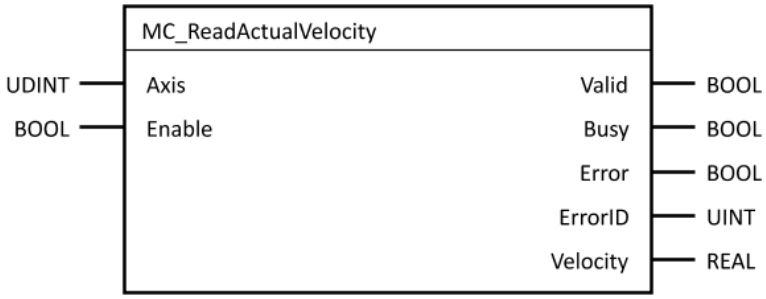


Abbildung 61: PLCopen Funktionsbausteine - MC_ReadActualVelocity

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Enable	BOOL	Liest die Geschwindigkeit, wenn eingeschaltet
OUT	Valid	BOOL	Die Ausgangswerte des FB können verwendet werden.
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer
OUT	Velocity	UINT	Wert der aktuellen Geschwindigkeit (in Achseinheiten [Einh./s])

In Verbindung mit aktivem MC_TorqueControl oder MC_BR_TorqueControl wird bei den FBs MC_ReadActualPosition und MC_ReadActualVelocity am Ausgang "Position" bzw. "Velocity" der Geberwert anstelle des Sollwerts angezeigt.

MC_MoveAbsolute

Dieser FB startet eine kontrollierte Bewegung auf eine vorgegebene absolute Position.

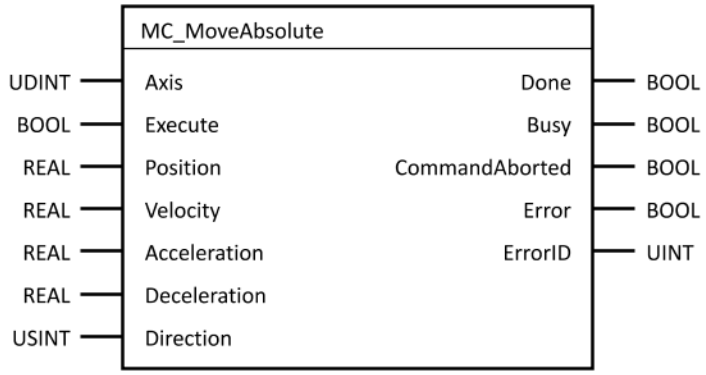


Abbildung 62: PLCopen Funktionsbausteine - MC_MoveAbsolute

I/O	Parameter	Datentyp	Beschreibung
IN	Axis	UDINT	Achsreferenz
IN	Execute	BOOL	Abarbeitung des FB wird bei steigender Flanke am Eingang gestartet.
IN	Position	REAL	Zielposition der Bewegung [PLCopen-Einh.]
IN	Velocity	REAL	Maximale Geschwindigkeit [PLCopen-Einh./s]
IN	Acceleration	REAL	Maximale Beschleunigung [PLCopen-Einh./s²]
IN	Deceleration	REAL	Maximale Verzögerung [PLCopen-Einh./s²]
IN	Direction	USINT	Bewegungsrichtung: mcPOSITIVE_DIR ... 0 mcNEGATIVE_DIR ... 1 mcCURRENT_DIR ... 2 mcSHORTEST_WAY ... 3 mcEXCEED_PERIOD ... 8 +mcAUTOMAT_POS ... 100
OUT	Done	BOOL	Abarbeitung erfolgreich. FB ist fertig. Zielposition erreicht
OUT	Busy	BOOL	FB ist aktiv und muss weiterhin aufgerufen werden.
OUT	CommandAborted	BOOL	FB wurde durch einen anderen FB abgebrochen.
OUT	Error	BOOL	Fehler bei Abarbeitung aufgetreten
OUT	ErrorID	UINT	Fehlernummer

Dieser FB startet eine kontrollierte Bewegung auf eine vorgegebene absolute Position. Nach einer steigenden Flanke am Eingang "Execute" werden alle Parameter übertragen, die für den Start der Bewegung notwendig sind. Die Achse wechselt in den Zustand Discrete Motion, nachdem alle Parameter erfolgreich übertragen wurden. Wenn der FB für eine reelle Achse verwendet wird, muss diese bereits referenziert worden sein.

Abbildungsverzeichnis

Glossar

Strukturierter Text:

Strukturierter Text (ST, auch SCL) ist eine der sechs in IEC 61131-3 festgeschriebenen Programmiersprachen für Automatisierungstechnik. Sie orientiert sich an PASCAL und enthält sowohl Sprachelemente dieser Sprache als auch SPS-typische Elemente. Typisch für Strukturierten Text sind Anweisungen, die wie in Hochsprachen bedingt oder in Schleifen ausgeführt werden können. SPS-typische Aufgaben wie Timer, Trigger, Counter und RS-FlipFlop kommen auch in ST die Funktionsbausteine der Standardbibliothek zum Einsatz. *(Quelle: Dr. Becker, Fachzentrum für Automatisierungstechnik, „Grundlagen der Automatisierungstechnik II“)*

PLCopen:

PLCopen ist fokussiert rund um die IEC 61131-3, den einzigen weltweiten Standard für industrielle Steuerungsprogrammierung. Dieser vereinheitlicht durch die Standardisierung der Programmierschnittstelle die Art und Weise wie industrielle Steuerungen gestaltet und bedient werden. Eine Standardprogrammierschnittstelle erlaubt es Personen verschiedene Programmelemente in verschiedenen Abschnitten des Software-Lebenszyklus herzustellen: Spezifikation, Design, Umsetzung, Test, Installation und Wartung. So hängen alle Teile an einer gemeinsamen Struktur und arbeiten harmonisch zusammen. Durch Zerlegung in logische Elemente, Modularisierung und moderne Softwaretechniken ist jedes Programm strukturiert, was seine Wiederverwendbarkeit erhöht, Fehler reduziert sowie die Programmier- und Nutzungsfähigkeit erhöht. *(Quelle: ISG Stuttgart, „PLCopen“)*

Stichwortverzeichnis

Literaturverzeichnis

Verfasser	Thema / Quelle	Letzter Zugriff
Uni Hamburg (2011): https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2011/cgk11-rothe-programmierstile-praesentation.pdf.	„Programmierstil.“	06.12.2016
Ulrich Kanngießer (2014): Programmierung mit Strukturierter Text.	„Strukturierter Text“	06.12.2016
Dr. Becker, Fachzentrum für Automatisierungstechnik: http://www.ipsta.de/download/automationstechnik/Kap13_ST%20.pdf	„Grundlagen der Automatisierungstechnik II“	06.12.2016
Uni Hannover (2015): https://www.ita.uni-hannover.de/automatisierungstechnik.html	„Automatisierung“	15.12.2016
B&R Hilfe Deutsch (2017): Gesamtes Kapitel 10 und Informationen über PLCopen Bausteine, sowie Programm	„B&R Messe“	26.05.2017
ISG Stuttgart: http://www.isg-stuttgart.de/de/isg/netzwerk/vereine-und-verbaende/plcopen.html	„PLCopen“	16.12.2016