

Automatisierungstechnik nach internationaler Norm programmieren (19)

Autor: Dr. Ulrich Becker
Fachzentrum Automatisierungstechnik und vernetzte Systeme im BTZ Rohr-Kloster
Mail: Ulrich.Becker@BTZ-Rohr.de

Mächtiges Werkzeug Byte- und Wortbearbeitung

Folge 19 schloss das Thema „IT in der Automatisierungstechnik“ mit Untersuchungen zum Mailversandes in lokalen Automatisierungsnetzen ab. In dieser letzten Folge der Serie werden an einem Beispiel Methoden der Byte- und Wortverarbeitung dargelegt. Es wird weiter zum Ausgangspunkt der Folge zurückgekehrt: Dies war die Frage, wie weit man das Wissen Step 7 erweitern muss, wenn Komponenten eingesetzt werden sollen, welche nach IEC 61131-3 zu programmieren sind.

Der Klassiker: Verknüpfungssteuerungen und binäre Logik

Jahrzehntelang konnte Steuerungstechnik fast ausschließlich nur mit Kontakten von Relais und Schützen betrieben werden. Reihenschaltung von Kontakten führten zu UND-Verknüpfungen, Parallelschaltungen zu ODER und mit Öffnerkontakten wurden Negationen realisiert. Alle Signale waren parallel verarbeitete binäre Signale. Die Boolesche Algebra lieferte hilfreiche Regeln für die Verknüpfung derartiger Signale. Verständlich sind auch Bestrebungen, durch Vereinfachung von Schaltfunktionen möglichst viele Relaiskontakte einzusparen.

Die Zeiten haben sich deutlich gewandelt. Viele Aufgaben der Automatisierungstechnik, welche vormals mit binärer Logik und minimierten Schaltnetzen gelöst wurden, sind heute eleganter unter Verwendung von Byte- oder Wort-Operationen zu bearbeiten. Dies wird nachfolgend an einem Beispiel „Fertigungszelle“ **Bild 111** demonstriert. Dieses ist sehr einfach gewählt und deshalb durchaus auch mit binärer Logik zu lösen. Dem aber wird eine alternative Lösung gegenübergestellt.

In der Fertigungszelle werden drei Typen Teile A, B oder C im Durchlauf bearbeitet. Bei positiver Flanke des Signals des Endlagenschalters S0 wird der Typ des Bauteils durch Abtasten mit vier Lichtschranken erfasst. Je nach Teileform sind dann die Ventile Y1 ... Y4 sowie die Antriebe M1 und M2 dauerhaft wie in **Tabelle 15** angegeben zu schalten. Eine positive Signalfanke des Endlagenschalters S1 schaltet alle Aggregate aus. In **Bild 111** wurde die Belegung der Lichtschranken als Byte im Dualcode (2#) und auch im Hexacode (16#) aufgeführt. Jede abgedunkelte Lichtschranke liefert FALSE – Signal.

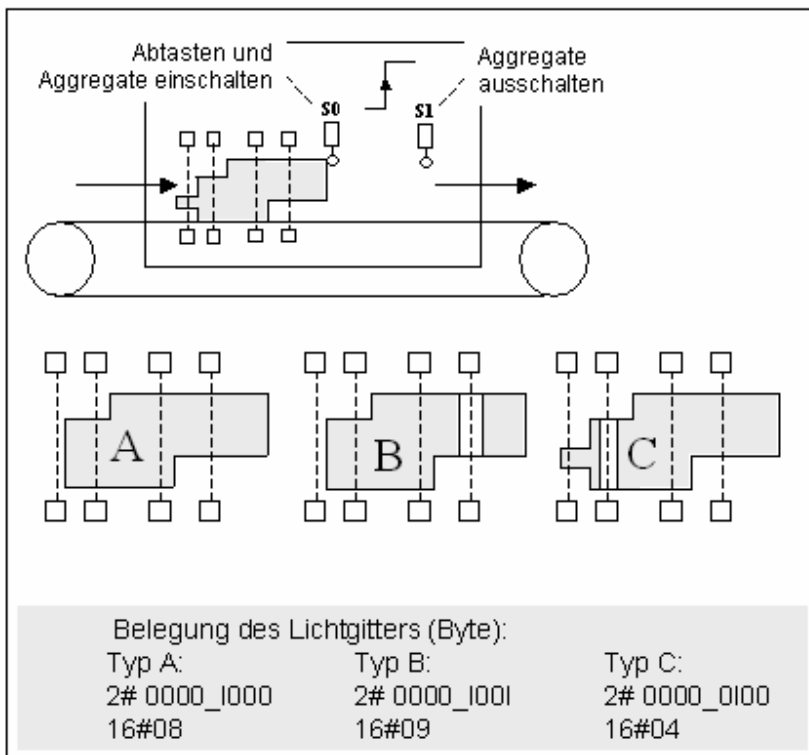


Bild 111: Aufgabenstellung Fertigungszelle

TYP A	Typ B	Typ C
Y0 = FALSE	Y0 = FALSE	Y0 = TRUE
Y1 = TRUE	Y1 = TRUE	Y1 = FALSE
Y2 = TRUE	Y2 = FALSE	Y2 = TRUE
Y3 = FALSE	Y3 = FALSE	Y3 = TRUE
M0 = TRUE	M0 = FALSE	M0 = TRUE
M1 = TRUE	M3 = TRUE	M1 = TRUE
Belegung des Ausgangsbyte:		
2# 0011_0110	2# 0010_0010	2# 0011_1101
16# 36	16# 24	16# 3D

Tabelle 15: Gewünschte Schaltung der Aggregate bei der Bearbeitung

Klassische Lösungen von Schaltnetzen führten zumeist über Schalttabelle und Schaltgleichung. In der Tabelle werden alle Kombinationen der Eingangssignale aufgeführt, zugehörige gültige Ausgangsbelegungen eingetragen und daraus die Schaltgleichung entwickelt. Bei vier Eingangssignalen müssten immerhin 16 Kombinationen berücksichtigt werden. Im übersichtlichen Beispiel aber können die Schaltgleichungen unmittelbar aus Tabelle 15 abgelesen werden. So muss Ventil Y0 nur eingeschaltet werden, wenn ein Werkstück Typ C erkannt wurde. Ventil Y1 muss dagegen für die Typen B und C geschaltet werden. Beispielhafte Ergebnisse der klassischen Lösung zeigt **Bild 112**. E0 bis E3 stehen für die vier Lichtschrankensignale.

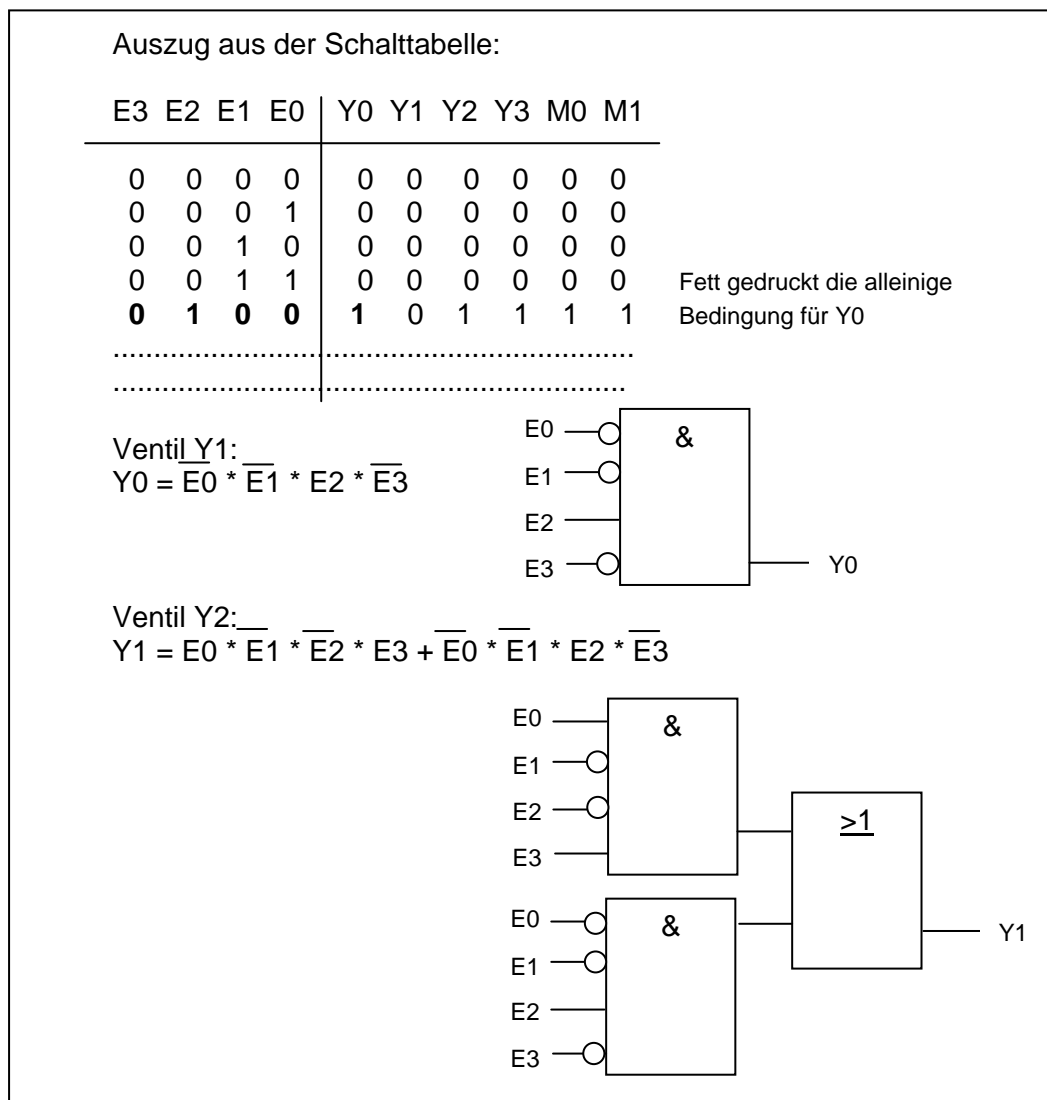


Bild 111: Klassische Lösungsansätze mit Schalttabelle und ODER-Normalform der Schaltgleichung

Eine alternative Lösung

Während bei der klassischen Lösung jedes Ausgangsbit einzeln bearbeitet wird, erlauben Byte- und Wortbefehle das gleichzeitige Bearbeiten von 8, 16 oder sogar 32 Bit im Byte-, Wort oder Doppelwortformat.

In den weiteren Ausführungen dieser Folge wird die Programmierung nach CoDeSys derjenigen nach Step7 gegenübergestellt. Damit soll auf die Frage zurückgekommen werden, wieweit das verbreitete Wissen über Step7 bei Einsatz anderer Komponenten erweitert werden muss (siehe Folge 1). In den Programmbeispielen ist die Step7-Syntax blau gekennzeichnet.

Zunächst zeigt **Bild 112** eine unvollständige Gegenüberstellung von Bit- und Byteverarbeitung. Wichtig ist: Bei CoDeSys unterscheiden die Operationen Laden (LD), Logik (AND, OR, XOR und NOT) sowie Zuweisen (ST) nicht zwischen Bit- und Byte-/Wort-/Doppelwortformat, während man dies bei Step7 vielfach sorgfältig unterscheiden muß.

Bitbearbeitung		Byte- und Wortbearbeitung	
Abfragen (Lesen)			
Laden LD	U bzw. O	Laden LD	Laden L
Bearbeiten			
Logische 1Bit-Verknüpfung AND, OR, XOR, NOT Flankenauswertung	U, O, X NOT FP, FN	Vergleichsoperationen: GT, GE, EQ, NE, LE Logische Verknüpfung: AND, OR, XOR Rechenoperationen: ADD, SUB, MUL, DIV Schieben und Rotieren: SHR, SHL, ROR, ROL Wandlung von Formaten Maskieren	==, <>, >, <, >=, <= für I, D, R z.B. ==I UW, OW, XOW bzw. UD, OD, XOD + -, *, / für I, D, R z.B. *R SR, SL, RR, RL für W, D z.B. SRW
Ausgeben (Schreiben)			
Zuweisen ST (Store) Setzen S Rücksetzen R	Zuweisen = SET, CLR neben S, R	Zuweisen (Store) ST	Transferieren T

Bild 112: Grundsätzliche Methodik der Bit- und Byte- bzw. Wortverarbeitung

Die Byte- bzw. Wortbearbeitung von Ein- und Ausgangssignalen gelingt solange problemlos, wie die Kartenbaugruppen oder Busklemmen problemorientiert genau für diese Formate zur Verfügung stehen. Das ist aber eher die Ausnahme.

Für das Beispiel Fertigungszelle seien die vier Lichtschranken auf Busklemmen mit Adresse %IX1.0 bis 1.3 gelegt. Die Aktoren seien an Ausgängen %QX 4.0 bis 4.5 angeschaltet. Die Byteoperation LD %IB1 würde dann mit den Bit 6 und 7 auch die Signale weiterer Busklemmen erfassen, welche für die Fertigungszelle nicht relevant sind. Genauso würde die Byteoperation ST %AB4 auch nicht relevante Ausgangsbusklemmen beschreiben. Deshalb müssen die Nutzsignale in Zwischenspeicher der gewünschten Formate eingetragen werden, um sie dann dort zu bearbeiten.

Der nachfolgende Kasten zeigt die Variablendeklaration des Programmbeispiels. Als Zwischenspeicher dienen die Variablen mit Namen „Lichtgitter“ und „Aktoren“ von Byteformat (rot hervorgehoben). Die Endlagenschalter S0 und S1 wurden an Busklemmen mit Adressen %IX0.6 und 0.7 geschaltet.

PROGRAM Fertigungszelle VAR S0 AT %IX0.6: BOOL; S1 AT %IX0.7: BOOL; Eingangsbyte AT %IB1:BYTE; Lichtgitter:BYTE; Aktoren:BYTE; Ausgangsbyte AT %QB4:BYTE posFI_0:R_TRIG; posFI_1:R_TRIG; END_VAR	Direkter Vergleich wenig sinnvoll; siehe dafür Symboltabelle 16 Für Lichtgitter und Aktoren wäre die Deklaration von <u>Statischen Variablen</u> mit Byteformat zweckmäßig. Im Beispiel aber wird mit dem <u>Merkerbereich</u> gearbeitet. Alternativ wäre auch Bytes eine globalen Datenbausteins möglich.
--	---

Wie erhalten die Zwischenspeicher genau die richtigen Biteinträge, mit denen dann die Byte- oder Wortverarbeitung gestartet werden kann? Selbstverständlich kann man mit den Befehlen Laden (LD) und Store (ST) einzelne Bits schreiben. CoDeSys ermöglicht in einfachster Weise das Selektieren der Bits von Variablen wie nachfolgend beispielhaft gezeigt. Bei Step7 löst man Bytes oder Worte über Merker oder globale Datenbits auf.

Bitweiser Eintrag von Ein- und Ausgangs-Signalen (Beispiele, nicht weiter verfolgt)		
LD Eingangsbyte.0 ST Lichtgitter.0 oder LD Aktoren.3 ST Ausgangsbyte.3	absolut U E1.0 = M100.0 oder U M102.3 = A4.3	symbolisch U "Lichtschranke_0" = "Lichtgitter_0" U "Aktoren_3" = "Y3"

Grundsätzlich ist aber das Arbeiten mit Masken und wortweiser UND-, ODER- bzw. auch ExklusivODER-Verknüpfung eleganter! Nach Maskierung liegen im Byte „Lichtgitter“ genau die vier Bits der Lichtschranken. Hierbei bezeichnen „n“ die Bits der Nutzsignale und „a“ die Bits anderer denkbarer Programmteile, welche auf Eingänge 4 bis 7 des Eingangsbytes zugreifen mögen.

Bei der Befehlsausgabe ist ähnlich vorzugehen. Zunächst werden mit der Maske Bit 6 und 7 selektiert, welche andere Programmteile in das Ausgangsbyte schreiben mögen. Danach werden die Nutzsignale Bit 0 bis 5 über ODER-Verknüpfung „beigefügt“. Auf den in Klammer gesetzten Befehl „ST Ausgangsbyte“ kann hier auch verzichtet werden. Mit ihm wird jedoch die Belegung des Bytes im Zwischenschritt sichtbar.

Awendung einer Maske für die Bearbeitung der Eingänge			
LD Eingangsbyte AND 16#0F ST Lichtgitter	Belegung des Byte a a a a _ n n n n 0 0 0 0 _ 1 1 1 1 (Maske) 0 0 0 0 _ n n n n	absolut L EB1 L B#16#0F UW T MB100	symbolisch L "Eingangsbyte" L B#16#0F UW T "Lichtgitter"
Awendung einer Maske für die Bearbeitung der Ausgänge			
LD Ausgangsbyte AND 16#C0 (ST Ausgangsbyte) OR Aktoren ST Ausgangsbyte	Belegung des Byte a a x x _ x x x x 1 1 0 0 _ 0 0 0 0 (Maske) a a 0 0 _ 0 0 0 0 0 0 n n _ n n n n a a n n _ n n n n	absolut L AB4 L B#16#C0 UW OR MB102 T AB4	symbolisch L "Ausgangsbyte" L B#16#C0 UW OR "Aktoren" T "Ausgangsbyte"

Im Kasten weiter unten ist das weitere Programm der Fertigungszelle aufgezeigt. Besondere Beachtung erfordern hier die Sprungbefehle. Warum sind Sprünge unverzichtbar? Binäre Logik erlaubt stets, Ausgangsbits abhängig von Bedingungen zu schreiben, denn die Befehle Zuweisen (=), Setzen (S) und Rücksetzen (R) sind „VKE-abhängig“. VKE bezeichnet das Verknüpfungsergebnis (1 Bit) der vorgelagerten Logik. Das Schreiben von Bytes oder Worten mit Store (ST) bzw. Step7: Transferiere (T) ist dagegen nicht abhängig vom VKE. Deshalb muß das Schreiben unterschiedlicher Werte in gleiche Adressaten mit Sprungbefehlen gezielt organisiert werden. Wird dies unterlassen, entstehen de facto die gefürchteten „Mehrfachzuweisungen“.

Wichtig ist weiter, daß trotz der Sprünge der Zyklus der Bearbeitung stets gesichert ist. Dies wird durch Beendigung des Programms an bestimmten Stellen und Rücksprung mit den Befehlen RET bzw. BEA bewirkt. Hier muß auch überlegt werden, wo zyklisch abzuarbeitende Programmteile und wo spezielle nicht zyklische Teile anzuordnen sind. Im Programm Fertigungszelle wird deshalb die Bearbeitung der Ausgänge wie die der Eingänge an den Anfang des Programms gesetzt! Weiter ist dafür zu sorgen, daß die Lichtschranken das Werkstück nicht zyklisch, sondern einmalig abtasten. Das wird durch Flankenauswertung der Endlagenschalter und gezielte Sprünge zu den Marken M1 und M2 bewirkt. Für die im Step7-Programm verwendeten Symbole gilt die Zuordnungstabelle **Tabelle 16**.

Element	Operand	Symbol
Schalter S0	E 0.6	“S0“
Schalter S1	E0.7	“S1“
Hilfsbit Flankenauswertung 0	M104.0	“Flankenmerker_0“
Hilfsbit Flankenauswertung 1	M104.1	“Flankenmerker_1“
Eingangsbyte1	EB1	“Eingangsbyte“
Zwischenspeicher Lichtgitter	MB100	“Lichtgitter“
Bit 0 von Lichtgitter	M100.0	“Lichtgitter_0“
Zwischenspeicher Aktoren	MB102	“Aktoren“
Bit 3 von Aktoren	M102.3	“Aktoren_3“
Ausgangsbyte4	AB4	“Ausgangsbyte“

Tabelle 16: Zuordnungstabelle mit Symbolen für Step7

Byteverarbeitung Fertigungszelle		
CAL posFI_0 (CLK:= S0) LD posFI_0.Q JMPC M1	(*Flankenauswertung S0*)	U "S0" FP "Flankenmerker_0" SPB M1
CAL posFI_1 (CLK:= S1) LD posFI_1.Q JMPC M2 RET	(*Flankenauswertung S1*)	U "S1" FP "Flankenmerker_1" SPB M2 BEA
M1: LD Lichtgitter EQ 16# 08 JMPC A	(*Abfrage Lichtgitter nach *Typ A*)	M1: L "Lichtgitter" L B#16#08 ==I SPB A
LD Lichtgitter EQ 16 # 09 JMPC B	(*Abfrage Lichtgitter nach *Typ B*)	L "Lichtgitter" L B#16#09 ==I SPB B
LD Lichtgitter EQ 16#04 JMPC C RET	(*Abfrage Lichtgitter nach *Typ C*)	L "Lichtgitter" B#16#04 ==I SPB C BEA
A: LD 16#36 ST Aktoren RET	(*Eintrag der Ausgangsbits *für Typ A*)	A: L B#16#36 T "Aktoren" BEA
B: LD 16#24 ST Aktoren RET	(*Eintrag der Ausgangsbits für *für Typ B*)	B: L B#16#24 T "Aktoren" BEA
C: LD 16#3D ST Aktoren RET	(*Eintrag der Ausgangsbits *für Typ C*)	C: L B#16#3D T "Aktoren" BEA
M2: LD 16#00 ST Aktoren RET	(*Ausschalten aller Ausgänge*) (*bei Betätigung von S1*)	M2: L B#16#0 T "Aktoren" BE

Umdenken Step7 – CoDeSys IEC 61131-3: ein Problem?

Die unterschiedliche Syntax bestimmter Operationen im IEC 61131-Programmiersystem CoDeSys und im System Step7 stellt bei einiger Routine und Verwendung der jeweiligen Referenzlisten sicher kein eigentliches Problem dar. Wohl aber sind Details der Datenverwaltung und der unterschiedliche Einsatz von Variablen und Symbolen stets zu beachten! Hierzu nochmals einige Ausführungen:

Sowohl CoDesys als auch Step7 verwenden globale und lokale Variablen. Bei lokalen Variablen besteht Übereinstimmung darin, daß deren Adressen in den Grenzen der verfügbaren Datenspeicher vorteilhaft vom System selbst verwaltet werden. Allerdings bestehen in Details der Nutzung erhebliche Unterschiede (**Bild 113**). Hierzu gehört auch die unterschiedliche Interpretation von Instanz eines Funktionsblockes (CoDeSys) und Instanzdatenbaustein eines Funktionsbausteins (Step7).

Variablen CoDeSys		Variablen Step7	
lokal	global	lokal	global
nur im aktuellen Baustein gültig	im gesamten Programm gültig	nur im aktuellen Baustein gültig	im gesamten Programm gültig
Adressen werden vom System automatisch verwaltet 1)		Adressen werden vom System automatisch verwaltet	Adressen müssen vom Programmierer verwaltet werden!
deklariert durch Bezeichner und Datentyp		deklariert durch Bezeichner und Datentyp	Absolute oder symbolische Syntax
Nutzung: nach den Regeln der Variablen-deklaration ohne Einschränkungen Beispiel global oder lokal:Teilezahl:INT;		Nutzung: - als Temporäre Variablen im L-Stack - als Statische Variablen im Instanzdatenbaustein eines FB	Nutzung: - im Merkerbereich - als Elemente von Datenbausteinen - als Timer und Zaehler
1) Nicht empfehlenswerter Sonderfall: Durch Bezeichnung als Merker können Variablen vom Programmierer adressiert werden. Beispiel: Teilezahl AT %MW10:WORD;		Beispiel: (in der Deklarationstabelle) #Teilezahl (INT)	Beispiele: "Teilezahl" oder MW10 "Status".LED1 oder DB2.DBX4.1 "Zeitglied1" oder T12

Bild 113: Datenhaltung in den Programmiersystemen CoDeSys und Step7

Im System Codesys werden auch globale Variablen von System automatisch verwaltet. Demgegenüber muß der Step7-Programmierer diese Daten selbst adressieren. Insbesondere die Verwendung von Merkern führt durch Mehrfachverwendung und unbeabsichtigten Byte- und gleichzeitig Bit-Zugriff immer wieder zu Fehlern. Sie können eingeschränkt werden, wenn man für alle globalen Daten konsequent Symbole verwendet. In Step5 wird dazu eine Symboltabelle editiert. Formal ergeben Symbole und Variablen ein vergleichbare Bild. Es bleibt aber der grundsätzliche Unterschied, daß hinter jedem Symbol eindeutig eine vom Programmierer festgelegte Adresse in den Speicherbereichen Merker, Datenbausteine, E/A-Peripherie oder Prozeßabbild liegt.

Die symbolische Programmierung wird im System Step7 in gleicher Weise erweitert auf die globalen Prozessabbilder der Ein- und Ausgänge. Demgegenüber werden bei IEC 61131-3 die Variablen mit dem Schlüsselwort „AT“ auf Ein- oder Ausgangsadressen gelegt. Neuere Versionen von CoDeSys im Verbund mit aktuellen Targets der Hersteller erlauben eine komfortable Hardware-Konfiguration ähnlich der von Simatic/Step7. Beim Anschalten von Busklemmen kann nun für jeden Ein- oder Ausgang ein Bezeichner vorgegeben werden, der als globale Variable wirkt. Dies ist der symbolischen Bezeichnung der Ein- und Ausgänge in Step7 sehr ähnlich.

Ausblick

Step 7 bleibt als "Industriestandard" mit hohem Marktanteil unverzichtbar für jeden, der sich dem Wachstumsmarkt Automatisierungstechnik zuwendet. Daneben wird es zunehmend wichtiger, auch andere Programmiersysteme zu beherrschen und sich mit der Norm IEC 61131-3 bekannt zu machen. Geräte der Atomatisierungstechnik dringen in immer „kleinere“ Applikationen ein und werden gleichzeitig immer leistungsfähiger. Ein Beispiel dafür ist das Kleinautomatisierungsgerät „CoDeSys easy“ des Unternehmens Moeller. **Bild 114** zeigt die Ergänzung des in allen Folgen verwendeten Trainingsracks mit diesem Baustein. Alle

Funktionen des Bandmodells einschließlich der Analogwertverarbeitung können von „CoDeSys easy“ gesteuert werden, denn in ihm ist der gesamte Operationsumfang von CoDeSys verfügbar. Die genormten Programmiersprachen, die Vernetzung mit Ethernet und der Preis dieser Technik sind Empfehlungen, manche noch bestehende konventionelle Lösung – z.B. mit Multifunktions- und Zeitrelais - abzulösen.

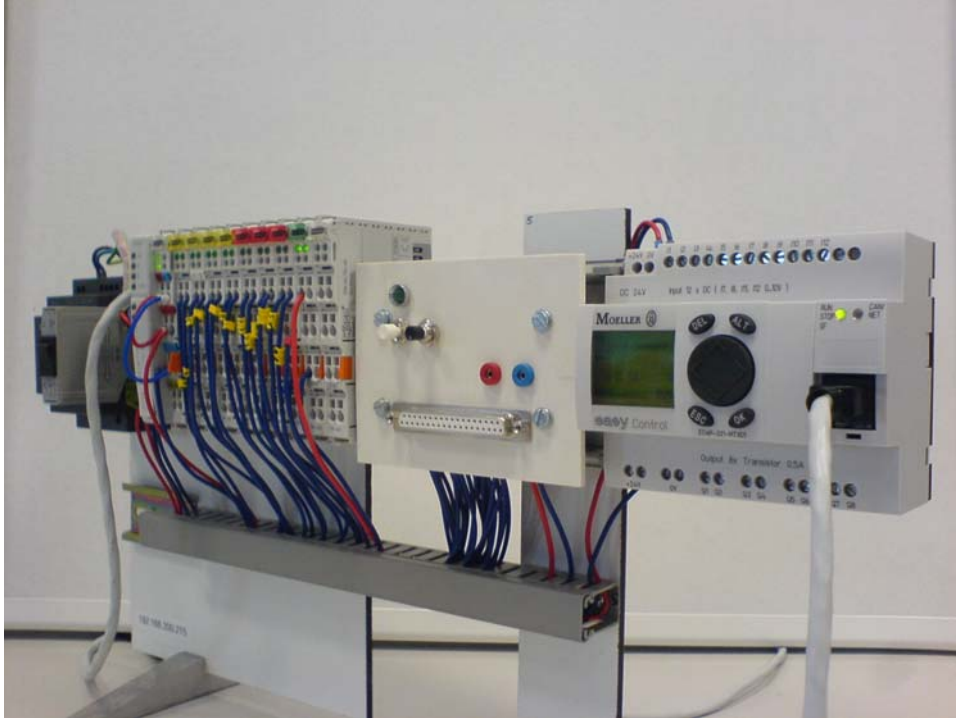


Bild 114: Kleinautomatisierungsgerät CoDeSys easy von Moeller (rechts)

Fazit:

Die abschließende Folge 20 der Serie „Automatisierungstechnik nach internationaler Norm programmieren“ zeigte Methoden der Byte- und Wortverarbeitung auf. Diese stellen zeitgemäße Alternativen zur binären Logik dar. An diesem Beispiel wird abschließend noch einmal eine vergleichende Betrachtung zur Datenhaltung in den Systemen CoDeSys und Step7 durchgeführt. Zukünftig wird es erforderlich sein, neben Step7 auch andere Programmiersysteme zu beherrschen. Die Deklaration lokaler und globaler Variablen im Kontext zur Vergabe von Symbolen ist der Schlüssel zur erfolgreichen Programmierung nach IEC61131-3.