Grafischer ProgrammEditor
(*SPS Editor*)

(c)2003-2010 SAM Electronics

November 2010



Inhaltsverzeichnis

1 Einleitung **6**

2 Bildschirm-Layout **7**

3 Programmaufruf-Parameter **8**

4 Tastenbelegung **9**

5 Menüs **10**

5.1 Menü Datei 10

5.2 Menü Bausteine 11

5.3 Menü Bearbeiten 11

5.4 Menü Anzeigen 11

5.5 Menü Übersetzen 12

5.6 Menü Seiten 12

5.7 Menü Zoom 13

5.8 Menü Betriebsarten 14

5.9 Menü Hilfe 14

5.10 Menü Exit 14

6 Sonderfunktionen **15**

6.1 SPS-Code passivieren (auskommentieren) 15

7 Datentypen **16**

7.1 Datentyp Boolean 16

7.2 Datentyp Festpunkt 16

8 Register **18**

9 Variable **19**

9.1 Temporäre Variable 20

10 Konstanten **21**

11 Bausteine **22**

11.1 Komplexe Bausteine 24

11.1.1 Formelbausteine 24

11.1.2 Strukturierter Text 25

11.1.2.1 Zugriff auf Register 27

11.1.2.2 Integer Funktionen 28

11.1.2.3 Festpunkt Funktionen 28

11.1.2.4 Boolesche Funktionen 29

11.1.2.5 Strukturierter-Text Klone-Bausteine 29

11.1.3 Makros 29

11.1.4 Digitale Filter 31

11.1.5 Funktionsblöcke 32

11.1.6 Sub-Prozesse 33

11.1.7 Zustandsautomaten 33

11.2 Analoge Standardbausteine 34

11.2.1 Konstanten-Bausteine (CONS, xCONS, tCONS) 34

11.2.2 Analog-In Bausteine (A-In2, A-In4, AIn2-Indx, AIn4-Indx) 35

11.2.3 Analog-Out Bausteine (AOut2, AOut4, AOut2-Indx, AOut4-Indx) 36

11.2.4 Move Bausteine (Move) 37

11.2.5 Addierer (ADGR, ADDW) 37

11.2.6 Subtrahierer (SUGR, SUBW) 37

11.2.7 Multiplizierer (MUL) 38

11.2.8 Multiplizierer Unsigned-Integer (MUL_UI) 38

11.2.9 Dividierer (DIVI) 39

11.2.10 Modulo (MOD) 39

11.2.11 Quadratwurzel (SQRT) 39

11.2.12 Betrag/Absolutwert (ABS) 39

11.2.13 Negierer (NEG) 40

11.2.14 Verdoppler (DOPP) 40

11.2.15 Halbierer (HALB) . . . 40
11.2.16 Proportional-Glied (PROP) . . . 40
11.2.17 Integrator (INTF) mit konstantem Integrationsfaktor . . . 41
11.2.18 Integrator (INTF) mit variablem Integrationsfaktor . . . 41
11.2.19 Integrator (INTL) mit Begrenzung und Wertvorgabe . . . 41
11.2.20 Tiefpass (PT1) . . . 41
11.2.21 Mittelwertbildung (AVER) über n Messwerte . . . 42
11.2.22 Verzögerung (T) um eine Zykluszeit . . . 42
11.2.23 Grenzwert-Bausteine (Limit) . . . 42
11.2.24 Kennlinie mit 33 Stützpunkten (KnL33) . . . 43
11.2.25 Kennlinie mit 65 Stützpunkten (KnL65) . . . 43
11.2.26 Kennlinie aus (X,Y) Wertepaaren . . . 43
11.2.27 Kennlinie mit 65 Stützpunkten (aKL65) ...nur positive X-Werte . . . 44
11.2.28 Minimum und Maximum Bausteine (MIN, MAX) . . . 44
11.2.29 Analoger Umschalter (SEL) . . . 44
11.2.30 Analoger Einschalter (SEL 0) . . . 44
11.2.31 Komparator(Vergleicher) Bausteine (A=B, A>B, A>=B) . . . 45
11.2.32 Sinus (SIN) . . . 45
11.2.33 Cosinus (COS) . . . 45
11.2.34 Tangens (TAN) . . . 46
11.2.35 Cotangens (COT) . . . 46
11.2.36 ArcusSinus (ASIN) . . . 46
11.2.37 ArcusCosinus (ACOS) . . . 46
11.3 Digitale Standardbausteine . . . 47
11.3.1 Konstanten-Bausteine (TRUE, FALSE) . . . 47
11.3.2 Digital-In Bausteine (D-In, DIn-Indx) . . . 47
11.3.3 Digital-Out Bausteine (D-Out, DOut-Indx) . . . 48
11.3.4 Register_To_Bits (WBIT) . . . 48
11.3.5 Bits_To_Register (BITW) . . . 49
11.3.6 Move digital (Move) . . . 49
11.3.7 Digitaler Umschalter (SELB) . . . 50
11.3.8 UND-Baustein (AND) . . . 50
11.3.9 ODER-Baustein (OR) . . . 50

11.3.10 Exklusiv-ODER Baustein (XOR) . . . 50
11.3.11 Nicht-Baustein (NOT) . . . 51
11.3.12 Wortweise logische Verknüpfungen (AND_W, OR_W, XOR_W) . . . 51
11.3.13 Wortweise NOT Verknüpfung (NOT_W) . . . 51
11.3.14 Logischer Generator (LGEN) . . . 51
11.3.15 Frequenzteiler (FRQD) . . . 51
11.3.16 Konstante(k) Anzugverzögerung (ANZG) . . . 52
11.3.17 Variable Anzugverzögerung (ANZG) . . . 52
11.3.18 Konstante(k) Abfallverzögerung (ABFL) . . . 52
11.3.19 Variable Abfallverzögerung (ABFL) . . . 53
11.3.20 Monoflop(Mono) mit konstanter(k) Impulsdauer . . . 53
11.3.21 Monoflop(Mono) mit variabler Impulsdauer . . . 53
11.3.22 Raising edge Trigger (r-trigg) . . . 54
11.3.23 Falling edge Trigger (f-trigg) . . . 54
11.3.24 Set/Reset Speicher (SR-FF) . . . 54
11.3.25 Reset/Set Speicher (RS-FF) . . . 54
11.3.26 Reset/Set Flip-Flop (FFlop) . . . 54
11.3.27 Zähler vor/rück (CTUD) . . . 55
11.3.28 Alarmspeicher (ALSP) . . . 55
11.4 Telegramm-Bausteine . . . 56
11.4.1 STN-Telegramm auf RS-485/422 senden (STN-Tx) . . . 56
11.4.2 STN-Telegramme empfangen (STN-Rx) . . . 57
11.4.3 TP2-Slave (TP2) . . . 57
11.4.4 RS-485 MODBUS Bausteine . . . 58
11.4.4.1 RS-485 MODBUS-Slave (MOD slv) . . . 58
11.4.4.2 RS-485 MODBUS-Master: Read-Multiple-Register (MBM F3) . . . 58
11.4.4.3 RS-485 MODBUS-Master: Write-Register (MBM F6) . . . 59
11.4.4.4 RS-485 MODBUS-Master: Write-Multiple-Register (MBM F16) . . . 59
11.4.5 UDP MODBUS Bausteine . . . 60
11.4.5.1 UDP MODBUS-Slave (MOD slv) . . . 60
11.4.5.2 UDP MODBUS-Master: Read-Multiple-Register (MBM F3) . . . 60
11.4.5.3 UDP MODBUS-Master: Write-Register (MBM F6) . . . 61
11.4.5.4 UDP MODBUS-Master: Write-Multiple-Register (MBM F16) . . . 61

11.4.6 CAN Bausteine . . . 62
11.4.6.1 CAN Sende (CAN-Tx) . . . 62
11.4.6.2 CAN Sende mit Identifier-Offset (CAN-Tx-Id) . . . 63
11.4.6.3 CAN Empfang (CAN-Rx) . . . 64
11.4.6.4 CAN Empfange mit Identifier-Offset (CAN-Rx-Id) . . . 65
11.4.6.5 CAN Empfange multiple (CAN-RxM) . . . 66
11.4.6.6 CAN Bus-Monitor (CBusMon) . . . 67
11.4.7 CAN-OPEN Bausteine . . . 67
11.4.7.1 Send-NMT(CO NMT) . . . 67
11.4.7.2 Send-SYNC (CO sync) . . . 68
11.4.7.3 Schreib Konstante ins Objektverzeichnis (SDO SKO) . . . 69
11.4.7.4 Schreib ins Objektverzeichnis (SDO SOV) . . . 70
11.4.7.5 Lies aus dem Objektverzeichnis (SDO LOV) . . . 71
11.4.7.6 Schreib indiziert ins Objektverzeichnis (SDO SOi) . . . 72
11.4.7.7 Lies indiziert aus dem Objektverzeichnis (SDO LOi) . . . 73
11.5 Daten-Verschiebe Bausteine . . . 74
11.5.1 Swapp Accu-Byte (SwAB) . . . 74
11.5.2 Swapp Accu-Word (SwAW) . . . 74
11.5.3 Kopiere Register nach Register (RtoR) . . . 74
11.5.4 Kopiere Register nach Register indiziert (RtoRi) . . . 75
11.5.5 Kopiere Register byteweise mit Offset (BtoB) . . . 75
11.5.6 Kopiere Register nach EEPROM-Register (RtoER) . . . 76
11.5.7 Kopiere Register nach EEPROM-Register indiziert (RtoER_i) . . . 76
11.6 Anzeige und Testhilfe-Bausteine . . . 77
11.6.1 Uhrzeit (Uhr) . . . 77
11.6.2 Systemstatus Baustein (Status) . . . 77
11.6.3 Register-CheckSumme (CHKS) . . . 78
11.6.4 Kommentartext . . . 78
11.6.5 Analog-Schreiber (Disp) . . . 78
11.6.6 Numerisches Analog-Display (AnaDisp) . . . 79
11.6.7 Numerisches Integer-Display (IntDisp) . . . 79
11.6.8 Boolescher Hand-Schalter (Hand) . . . 80
11.6.9 LCD Display: Text-Ausgabe . . . 80
11.6.10 LCD Display: Wert-Ausgabe . . . 81
11.6.11 Programm-fluss Steuerung (PFS) . . . 82
11.6.12 Variablen-Bausteine (VAR) . . . 82

Kapitel 1

Einleitung

Der in die ***IDE*** eingebundene ***G****rafische****P****rogramm****E****ditor* ***GPE*** ermöglicht das Erzeugen von grafischen SPS-Programmen. Der ***GPE*** enthält einen Simulator, mit dem diese SPS-Programme (vor)getestet werden können.

Die grafisch erzeugten Programme werden dann durch den ***GPE*** in unsere bestehende hausinterne ***SPS-Makro-Sprache*** übersetzt, bevor sie in der ***IDE*** assembliert und in die Zielhardware geladen werden.

Kapitel 2

Bildschirm-Layout

Die Oberfläche des ***GPE's*** besteht aus (max. 99) voneinander unabhängigen Bildschirmseiten. Durch Umschaltfunktionen kann zwischen diesen Seiten beliebig gewechselt werden.

Jede dieser Bildschirmseiten ist in 5 x 5 (=25) Arbeitsblätter unterteilt. Diese Unterteilung dient nur der besseren Übersicht bzw. Einteilbarkeit und hat keinen Einfluss auf die erzeugten Programme. (..Programmstrukturen können ohne Rücksicht auf Arbeitsblattgrenzen erstellt werden.)

Zur Anpassung an unterschiedliche Gegebenheiten (..verwendeter Monitor) und Aufgaben (..Übersicht verschaffen) kann die Bildschirmdarstellung in einem weiten Bereich vergrößert oder verkleinert werden (***..zoom)***.

Kapitel 3

Programmaufruf-Parameter

Wenn dieses Programm aus der IDE aufgerufen wird, dann werden alle Aufrufparameter von der IDE entsprechend gesetzt. Wird es direkt aufgerufen, dann können Parameter über die Kommandozeile eingegeben werden.

Kommandozeilenparameter:

1. Name der zu bearbeitenden Datei (..Name.GP2) Die Extention 'GP2' muss dabei nicht mit angegeben werden.
2. Zykluszeit (in Millisekunden)
3. Prozess-Verzeichnis-Pfad für Grafik- und Makro-Programmdateien (..Name.GP2, Name.a)
4. Verzeichnis-Pfad für Übergeordnete Datenstrukturen (SPS Variablen- und SPS Konstanten-Dateien)
5. IP-Adresse für Online-Betrieb (nur bei ZM432, bei der ZKG500 ist dieser Parameter ein Dummy)
6. Prozess-Zyklus-Nr. (0=Hauptzyklus, -1=undefiniert) ..Mit Hilfe dieser Nummer kann in der Betriebsart 'Online' aus der Zielhardware die verbrauchte Rechenzeit des Prozesses abgefragt und in Prozent angezeigt werden.
7. Kennung ob im '***Grafik-Mode***'(default-Einstellung) oder im '***nur still compilieren***' Mode gestartet werden soll. Wenn dieser Parameter den String 'leer' enthält, dann wird das angegebene Programm nur 'still compiliert'. Diese Betriebsart ist vorgesehen, um aus der IDE heraus alle Programme gleichzeitig automatisch compilieren zu können.
8. Das Zielsystem: '0'=ZM432, '1'=MIC500, ...
9. Der aktuelle Projektname
10. Der aktive ComPort: 'COM1' bis 'COM6'
11. Die Nummer eines gesuchten Bausteins (-1, wenn kein Baustein gesucht werden soll)

Kapitel 4

Tastenbelegung

Um die Arbeit zu erleichtern, sind einige Tasten mit Funktionen belegt:

Taste	—— Funktionsbelegung ——
Z	Undo (letzte Änderungen rückgängig machen)
Y	Redo (letztes ***Undo*** rückgängig machen)
S	Sichert aktuellen Stand, den man mit ***Undo*** wieder zurückholen kann
X	Cut („Rahmen" Ausschneiden)
C	Copy („Rahmen" Kopieren)
V	Paste (Ausschnitt oder Kopie wieder einfügen)
F	Finde (Baustein suchen)
Pfeil	Scrollt Bildschirm in entsprechende Richtung
Strg+Pfeil	Scrollt Bildschirm um ***Bildschirmgröße*** in entsprechende Richtung
+	ZOOM In (..näher ran)
-	ZOOM Out (..weiter weg)
Bild auf	Nächste Seite (..falls vorhanden)
Bild ab	Vorherige Seite (..falls vorhanden)
Pos 1	Bei allen selektierten Bausteinen (..Rahmen) die Baustein-Nr. um eins ***erhöhen***
Ende	Bei allen selektierten Bausteinen (..Rahmen) die Baustein-Nr. um eins ***runter-zählen***
F7	Singel-Step / Einzelschritt-Simulation (..nur in Betriebsart Simulation)

Die folgenden Funktionen (Signalkommentare für Ausgangs-Pins und Verfolgung von Verbindungslinien zu anderen Seiten) sind durch eine Kombination von Tasten-druck und Maus-klick erreichbar.

Taste + linke Maustaste klicken	—— Funktionsbelegung ——
Strg + Verbindungspunkt anklicken	Signalkommentar für Ausgang erzeugen
Alt + Verbindungspunkt anklicken	Signalkommentar für Ausgang ein/aus-schalten (..toggelt)
Shift + Verbindungspunkt anklicken	Verbindung zu anderen Seiten folgen (..falls vorhanden)

Kapitel 5

Menüs

5.1 Menü Datei

- ***ArbeitsBlätter Löschen:*** ... Alle Arbeitsblätter werden gelöscht, danach gibt es nur noch eine leere Seite.
- ***Programm Laden:*** ... Eine Programmdatei „Name.GP2" kann geladen werden.
- ***Programm Sichern:*** ... Der aktuelle Programmstand wird unter dem aktuellen Programmnamen gesichert.
- ***Programm Sichern unter:*** ... Der aktuelle Programmstand kann unter einem beliebigen Namen gesichert werden.
- ***Rahmen Sichern:*** ... Der Inhalt des Select-Rahmens kann unter einem beliebigen Namen „Name.GP3" gesichert werden.
- ***Rahmen Laden:*** ... Ein vorher gesicherter „Rahmen" kann wieder in das Programm eingefügt werden.
- ***Makro Laden:*** ... Abgespeicherte Makros „Name.GP4" können wieder in das Programm eingefügt werden.
- ***Automat Laden:*** ... Abgespeicherte Zustandsautomaten „Name.GP5" können wieder in das Programm eingefügt werden.
- ***Drucker-Setup:*** ... Drucker einrichten.
- ***Vorschau (s/w):*** ... Der Bildschirm wird in schwarz/weiß dargestellt (..Funktion toggelt)
- ***Drucke (Bildschirm s/w):*** ... Der aktuelle Bildschirminhalt wird schwarz/weiß ausgedruckt (..screendump)
- ***Drucke (alle Seiten s/w):*** ... Alle Seiten werden schwarz/weiß ausgedruckt. (Inhalte von Makros etc. werden nicht mit ausgedruckt)
- ***Bitmap (aktuelle Seite):*** ... Die aktuelle Seite wird als Bitmap-Datei gespeichert: „Name.bmp" .

- ***Ende ohne Sicherung (Quitt):*** ... Das Programm wird beendet, ohne Änderungen zu sichern.
- ***Ende mit Sicherung (Exit):*** ... Das Programm wird mit Sicherung der Änderungen beendet.

5.2 Menü Bausteine

Über diesen Menüpunkt können alle SPS-Bausteinarten erzeugt und in das aktuelle Programm eingefügt werden.

Siehe Kapitel **?? *„Bausteine".***

5.3 Menü Bearbeiten

- ***Undo*** ... (letzte Änderungen rückgängig machen)
- ***Redo*** ... (letztes „Undo" rückgängig machen)
- ***Cut*** ... („Rahmen" Ausschneiden)
- **Copy** ... („Rahmen" Kopieren)
- **Paste** ... (Ausschnitt oder Kopie wieder einfügen)

5.4 Menü Anzeigen

- ***Anzeigen>Meldungs-Fenster: ...*** Dieser Menüpunkt macht das Meldungsfenster sichtbar, in welchem Fehler und Diagnosemeldungen angezeigt werden. Durch klicken mit der rechten Maustaste in das Fenster, kann dieses wieder geschlossen werden. Ein erneutes auswählen dieses Menüs schließt das Fenster ebenfalls.
- ***Anzeigen>Variablen-Liste: ...*** Dieser Menüpunkt erzeugt eine Liste aller Prozess-Variablen und -Konstanten. Wird eine Variable mit *'Doppelklick'* oder *'Return'* ausgewählt, dann wird zu der entsprechenden Stelle Verzweigt.
- ***Anzeigen>Seiten-Nummerierung: ...*** Dieser Menüpunkt versieht alle 25 Seiten in der rechten unteren Ecke mit einer Seitennummer. Die 1. Seite erhält die Bezeichnung „A1" die letzte Seite erhält die Bezeichnung „E5". Ein erneutes auswählen dieses Menüpunktes entfernt die Seiten-Nummerierung wieder.
- ***Anzeigen>Alle Parameter: ...*** Dieser Menüpunkt bewirkt das bei jedem Baustein alle Baustein-Parameter angezeigt werden, auch wenn sie im Parameter-Dialog als „unsichtbar" gekennzeichnet sind.
- ***Anzeigen>Keine Parameter: ...*** Dieser Menüpunkt bewirkt das keine Baustein-Parameter angezeigt werden, auch wenn sie im Parameter-Dialog als „sichtbar" gekennzeichnet sind.
- ***Anzeigen>Selektierte Parameter: ...*** Dieser Menüpunkt bewirkt das nur die Baustein-Parameter angezeigt werden, die im Parameter-Dialog als „sichtbar" gekennzeichnet sind.

- ***Anzeigen>Projekt-Information: ...*** Über diesen Menüpunkt gelangt man in ein Dialogfenster, in welchem alle aktuellen Projektinformationen angezeigt und geändert werden können:

ProjektName	= Name des Projektes
ProjektVersion	= VersionsNummer des Projektes
Bearbeiter	= Name des Bearbeiters
SaveDatum	= Datum des letzten Abspeicherung des Projektes
CompileDatum	= Datum des letzten Übersetzens
ZielSystem	= ZM432, ZKG500, SRK01, SRK02, X-ZKG500, ZKG512, ...
CompileMode	= optimieren, nicht optimieren
Zykluszeit	= Prozess-Zykluszeit in mSec
Display(Sec/Div)	= Zeitmaßstab für „Schreiber-Baustein". Einheit: Sekunden/Teilung
Kommentar	= In dieses Feld kann ein beliebiger Kommentar eingegeben werden.

- ***Anzeigen>Grafik immer im Vordergrund: ...*** Das Anwählen dieses Menüpunktes bewirkt, daß das Editorfenster immer im Vordergrund ist. Das kann hilfreich sein, wenn mehrere Anwendungen auf dem Monitor angezeigt werden.
- ***Anzeigen>Deutsch...Englisch..: ...*** Das Anwählen dieser Menüpunkte bewirkt, daß alle Texte in der gewählten Sprache angezeigt werden.
- ***Anzeigen>Suche Baustein..: ...*** Über diesen Menüpunkt können Bausteine anhand der Baustein-Nr. gesucht werden.

5.5 Menü Übersetzen

Das Menü ***Übersetzen*** enthält die Untermenüpunkte „Code erzeugen" und „Programm prüfen".

Code erzeugen Über diesen Untermenüpunkt wird das aktuelle Programm in SPS-Makro-Dateien (Name.a, Name.KF, Name.VB und Name.VF) übersetzt. Wenn beim Übersetzen Fehler auftreten, dann werden entsprechende Meldungen (Warnungen oder Fehler) über Dialogfenster angezeigt. Wenn diese Fehler bzw. Warnungen Bausteinen zugeordnet werden können, dann werden diese Bausteine farblich gekennzeichnet (Rot=Fehler, Gelb=Warnung).

Alle Fehler, Warnungen und Diagnosemeldungen werden zusätzlich im „Meldungsfenster" ausgegeben und können nach dem Übersetzen über den Menüpunkt: „Anzeigen.Meldungsfenster" angezeigt werden. Wenn beim Übersetzen Fehler auftreten, dann wird das Meldungsfenster sofort angezeigt.

Programm prüfen Über diesen Untermenüpunkt wird das aktuelle Programm ebenfalls übersetzt und es werden dieselben Prüfungen wie bei „Code Erzeugen" durchgeführt. Hier werden aber **keine** SPS-Makro-Dateien erzeugt.

5.6 Menü Seiten

Ein Programm (eine Programmdatei) kann auf mehrere Arbeitsblattebenen aufgeteilt werden. Diese Ebenen werden im folgenden als „Seiten" bezeichnet. Mit Hilfe der folgenden Untermenüs können die (max.99) Seiten eines Programms verwaltet werden:

- ***Bezeichnung ändern:...*** Die Bezeichnung der aktuellen (der sichtbaren) Seite kann geändert werden.
- ***Neue Seite:...*** Es wird eine neue (noch leere) Seite erzeugt. Dazu muss auch der Name der neuer Seite angegeben werden. Nach Ausführung dieser Aktion wird die neue Seite zur aktuellen Seite.
- ***Seite löschen:...*** Die aktuelle Seite wird gelöscht. Wenn es nur eine Seite gibt, dann werden Inhalt und Name der aktuellen Seite gelöscht.
- ***Seite anheben:...*** Die aktuelle Seite wird gegenüber allen anderen Seiten „angehoben". War sie z.B. „Seite 2 von 5", dann ist sie nach dem Anheben „Seite 3 von 5".
- ***Seite absenken:...*** Die aktuelle Seite wird gegenüber allen anderen Seiten „abgesenkt". War sie z.B. „Seite 4 von 5", dann ist sie nach dem Absenken „Seite 3 von 5".
- ***Nächste Seite:...*** Wenn es eine nächste Seite gibt, dann wird die nächste Seite zur aktuellen Seite. Diese Funktion kann auch drücken der Taste „Bild ↑" angewählt werden.
- ***Vorherige Seite:...*** Wenn es eine vorherige Seite gibt, dann wird die vorherige Seite zur aktuellen Seite. Diese Auswahl kann auch durch drücken der Taste „Bild ↓" erfolgen.

Verbindungen zu anderen Seiten können über selbst-definierte Variable oder über Verbindungslinien hergestellt werden. Wenn eine Verbindungslinie zwei Seiten miteinander verbinden soll, dann geht man folgendermaßen vor:

- Mit der linken Maustaste einen Verbindungspunkt anklicken.
- Über die Menüs „Nächste/Vorige"-Seite oder über die „Bild auf/ab"-Tasten eine andere Seite anwählen.
- Die immer noch mit dem Mauszeiger geführte Verbindungslinie nun entsprechend weiter verbinden.

Verbindungen direkt von Baustein-Pins zu anderen Seiten sind nicht erlaubt. Man muss in diesem Fall einen Zwischenpunkt setzen und kann dann von diesem in die anderen Seiten gehen. Verbindungspunkte die in andere Seiten führen, sind durch eine rote Umrandung und durch einen Ziel-Text (Seite.Baustein.Pin) gekennzeichnet.

5.7 Menü Zoom

Über diesen Menüpunkt kann die Bildschirmdarstellung „gezoomt" werden. Es gibt folgende Untermenüpunkte:

- ***Übersicht:...*** Das ganze Arbeitsblatt (5x5=25 Teilarbeitsblätter) wird angezeigt. So kann man sich eine Übersicht verschaffen. Diese Ansicht ist auch dann nützlich, wenn man ganze Programmteile auf dem Arbeitsblatt verschieben will.
- ***Zentrieren:...*** Der Ausschnitt des Arbeitsblattes, auf dem sich Bausteine befinden, wird in maximaler Größe angezeigt.
- ***Select-Rahmen:...*** Der Ausschnitt des Arbeitsblattes, der durch den „Select-Rahmen" markiert ist, wird in maximaler Größe angezeigt.
- ***Arbeitsblatt:...*** Das Teilarbeitsblatt (eines von 25) das sich in der Mitte des Bildschirms befindet, wird in maximaler Größe angezeigt.

5.8 Menü Betriebsarten

Der ***GPE*** unterstützt folgende Betriebsarten:

- ***Edit*** ... (Programmerstellung)
- ***Simulation*** ... (Programmtest)
- ***Online*** ... (Online-Test/Bediensystem)

Unabhängig von diesen Betriebsarten kann der Menüpunkt ***read-only*** aktiviert werden. Dadurch wird in allen Betriebsarten ein (unabsichtliches) verändern des SPS-Programms verhindert.

Betriebsart Edit Dieses ist die voreingestellte Betriebsart nach Programmstart. Hier können SPS-Bausteine platziert, parametriert und miteinander verknüpft werden.

Betriebsart Simulation In dieser Betriebsart können (formal) fehlerfreie SPS-Programme simuliert werden. Es stehen die Befehle ***Start***, ***Stopp***, ***Einzelschritt*** und ***Reset*** zur Verfügung. Wenn nicht Einzelschritt gewählt ist, dann wird das Programm mit der eingestellten Zykluszeit simuliert.

Register und Baustein-Verbindungen über SPS-Variable bleiben bei der Simulation unberücksichtigt. Analoge Bausteineingänge die nur über Variable angeschlossen sind, können beim Simulieren mit der linken Maustaste angeklickt und dann über ein Dialogfenster mit einem Simulationswert versorgt werden. Bei digitalen Bausteineingängen bewirkt das Anklicken der Eingangsvariablen beim Simulieren eine unmittelbare Invertierung des Eingangswertes.

Betriebsart Online Diese Betriebsart stellt das (online) Bediensystem dar. Hier kann das SPS-Programm auf der Zielhardware gestartet, gestoppt und beobachtet werden. Dafür werden alle benötigten SPS-Variable über eine ***Netzwerkverbindung***[1] an den ***GPE*** übertragen und in der SPS-Grafik animiert dargestellt. Wenn ein SPS-Bausteinausgang keine Ausgangs-variable besitzt, dann wird der Wert dieses Ausgangs durch den GPE berechnet(nach-simuliert) und ebenfalls animiert dargestellt.

In dieser Betriebsart können SPS-Variable(im RAM) und SPS-Konstanten(im EEPROM) verändert werden. Geänderte SPS-Konstanten werden automatisch in die aktuelle „Projekt-Datei" übertragen.

Zu Beginn jeder „Online Sitzung" werden selbsttätig alle Konstanten der Anlage in das aktuelle Projekt übertragen. Dadurch wird sichergestellt, dass auf jedem Rechner immer die aktuellen SPS-Daten[2] vorhanden sind.

5.9 Menü Hilfe

Über diesen Menüpunkt kann man Informationen über den ***G****rafischen****P****rogramm****E****ditor* erhalten.

5.10 Menü Exit

Dieser Menüpunkt bewirkt ein Beenden des Programms unter Abspeicherung aller Änderungen.

[1] TCP/IP über Ethernet beim ZM432, RS-232 bei ZKG500
[2] Daten, die sich auf dem Zielrechner befinden

Kapitel 6

Sonderfunktionen

6.1 SPS-Code passivieren (auskommentieren)

Wenn SPS-Programmteile aktuell nicht benutzt werden sollen, dann können sie „**passiviert"** werden.

Dazu wird ein Rahmen (linke Maustaste) um den betroffenen Programmteil gezogen. Dann wählt man aus dem Kontexmenü (rechte Maustaste) den Menüpunkt „***Passiviere selektierte Bausteine***". Die selektierten Bausteine werden daraufhin dunkelgrün dargestellt.

Über den Menüpunkt **„Aktiviere selektiert Bausteine**" kann die ***Passivierung*** wieder rückgängig gemacht werden.

Für ***passivierte*** Baustein wird kein Programmcode erzeugt. Baustein-variable werden zwar angelegt, ihnen wird aber (wegen der Passivierung der Bausteine) kein Wert zugewiesen.

Kapitel 7

Datentypen

Zur Zeit werden die Datentypen ***Boolean*** und ***Festpunkt*** unterstützt.

7.1 Datentyp Boolean

Dieser Datentyp kennt nur die zwei Zustande ***false*** und ***true*** bzw. ***0*** und ***1***. Der Speicherplatzbedarf für diesen Datentyp ist 32 Bit bzw. 4 Byte. Die eigentliche digitale Status-Information steckt im obersten Byte, die restlichen drei Byte können anders genutzt werden. Zeit-Bausteine (Monoflops u.s.w.) nutzen diese Bytes als Zähler-variable.

In der animierten Darstellung (Betriebsarten ***Simulation*** oder ***Online***) werden Signal-Leitungen mit dem Wert „true" ***grün*** dargestellt. Signale mit dem Wert „false" werden ***rot*** dargestellt.

7.2 Datentyp Festpunkt

Dieser Datentyp (engl. ***FRACTION***) hat einen Wertebereich von -1 bis +1. Beim ZM432 (32 Bit Variable) hat dieser Datentyp eine Auflösung von $1/2^{31} = 0.46 * 10^{-9} = 0.000\ 000\ 000\ 46$. Bei Systemen mit 16 Bit Variable hat dieser Datentyp eine Auflösung von $1/2^{15} = 30 * 10^{-6} = 0.000\ 03$

Bei arithmetischen Operationen die einen Überlauf bewirken können (z.B. Addition) wird das Ergebnis normalerweise auf +/-1 begrenzt. Wird der Festpunktwert als Winkel interpretiert (+/-1 entspricht +/-pi) dann wird bei den entsprechenden arithmetischen Funktionen ADDW(Add Winkel) und SUBW(Sub Winkel) keine Überlaufbegrenzung vorgenommen:

(0.5+1.0 => -0.5 entsprechen 90° + 180° => -90°).

Signalleitungen des Datentyps „Festpunkt" werden in allen Betriebsarten ***gelb*** dargestellt.

In den Betriebsarten „Simulation" und „Online" kann über ein Kontexmenü (Pin-Name oder Wert-Anzeige mit der rechten Maustaste anklicken) der Wert von Ausgangsvariablen oder externen Eingangsvariablen auf verschiedene Arten angezeigt werden:

- ***als Festpunktwert***: . . . -1.0 bis +1.0

- ***als Winkel***: . . . -180 bis +180 Grad
- ***als Dezimalwert***: . . . -2 147 483 648 bis +2 147 483 647 bei 32-Bit Auflösung. -32768 bis +32767 bei 16-Bit Auflösung
- ***als Hexadezimal-wert***: . . . $8000 0000 bis $7FFF FFFF bei 32-Bit Auflösung $8000 bis $7FFF bei 16-Bit Auflösung
- ***Variablen-Name:*** . . . Der Name der Variable wird angezeigt
- ***als Zeit***: . . . in Sekunden anzeigen (Zeit=Wert*Zykluszeit)

Kapitel 8

Register

Register sind 2-Byte Variable welche als Array strukturiert sind. Das ZM432 enthält 32K Register. Dem Anwender steht der Bereich 0 bis 29999 zur Verfügung. Für „nichtflüchtiges" Speichern ist der untere Registerbereich (4k-Byte EEPROM beim ZM432) reserviert.

Bei der ZKG500 stehen dem Anwender 2000 RAM-Register zur Verfügung. Der untere Registerbereich (0 bis 199) ist EEPROM-Speicher.

Die X-ZKG500 hat 3500 RAM-Register und 500 EEPROM-Register

Der Datenaustausch mit Fremdgeräten erfolgt über serielle Schnittstellen, welche Register beschreiben oder auslesen können. Der Datenaustausch zwischen Registern und SPS-Variablen erfolgt durch spezielle ***Register- bzw. Schnittstellen-Bausteine***.

Auf den EEPROM-Registerbereich ist im allgemeinen nur lesender Zugriff möglich. Wenn in diesen Bereich geschrieben werden soll, dann müssen dafür die Bausteine ***RegToEEPROM*** oder ***RegToEEPROM_idx*** benutzt werden. Da EEPROMS nicht beliebig oft beschrieben werden können (einige 10 000 mal), sollten zyklische Schreiboperationen auf „einmal je Stunde" begrenzt werden.

Kapitel 9

Variable

Jeder SPS-Baustein enthält in der Regel für jeden Ausgang entweder eine Logische- oder eine Festpunkt-Variable. In diesen Variablen wird das Ergebnis der SPS-Operationen abgelegt. Das Ergebnis jeder SPS-Operationen wird außerdem im ***logischen-*** bzw. im ***Festpunkt-Akkumulator*** gespeichert. Der nächste SPS-Baustein in der Abarbeitungsreihenfolge kann ggf. den Wert aus dem Akkumulator benutzen, so daß in vielen Fällen keine Ausgangsvariable benötigt, und dann auch nicht erzeugt wird. (..spart Rechenzeit und Variablen-Speicherplatz).

In der Betriebsart ***Online*** muss jeder Ausgangswert angezeigt werden können. Wenn ein Ausgang keine Variable besitzt, dann versucht der GPE den Wert dieses Ausganges zu berechnen bzw. nachzusimulieren. Wenn das nicht möglich ist, dann wird der Ausgangswert ohne Wert in „weiß" angezeigt.

Benötigte SPS-Variable werden beim Übersetzen automatisch angelegt. Wenn das nicht gewünscht ist, weil man den Variablen-Namen selbst bestimmen möchte, oder weil eine bereits bestehende lokale oder externe Variable genutzt werden soll (Referenz), dann ist folgendermaßen vorzugehen:

- Mit der rechten Maustaste auf den Pin-Namen klicken, und aus dem dann erscheinenden Kontext-Menü: „***Lokale Variable erzeugen***" oder „***Bestehende Variable nutzen (Referenz)***" wählen.

Wenn der Eingangswert eines Bausteins nicht über Verbindungslinien sondern über den Namen einer Variable (Referenz) bestimmt werden soll, dann ist folgendermaßen vorzugehen:

- Mit der rechten Maustaste auf den Eingangs-Pin-Namen klicken, und aus dem Kontext-Menü: „***Referenz erzeugen***" wählen.

9.1 Temporäre Variable

In Ziel-Systemen mit wenig RAM (GPM500) kann es vorkommen, das mehr SPS-Variable gebraucht werden, als zur Verfügung stehen. In solchen Fällen besteht eine Lösung darin, Variable mehrfach zu nutzen. Eine Variable die mehrfach genutzt werden soll, muss zuerst lokal oder extern erzeugt werden. Bausteine die diese Variable als Ausgangsvariable benutzen sollen, müssen dann eine Referenz auf diese Variable erzeugen.

Der Name von temporären Variablen muss mit einem Unterstrich „_Name" beginnen. Der Unterstrich signalisiert dem GPE das es sich um temporäre Variablen handelt.

Temporäre Variable werden vom GPE folgendermaßen unterstützt:

- Es wird keine Warnung erzeugt, wenn eine Variable mehrfach (als Ausgangs-Referenz) vorkommt.
- Die Abarbeitungsreihenfolge der Bausteine wird so gewählt, das keine Konflikte durch temporäre Variable entstehen. Wenn das nicht möglich ist, dann wird eine Fehlermeldung erzeugt.
- Eingangswerte für SPS-Bausteine die über Referenzen/Variablennamen (und nicht über Verbindungslinien) festgelegt sind, dürfen keine temporären Variable sein, da sonst die Konfliktfreiheit nicht überprüft werden kann. (..wird beim Übersetzen vom GPE z.Z. **nicht** geprüft, da Funktionsbausteine „Temporäre Variable" als Eingangsparameter benutzen.)
- SPS-Ausgangs-Variable die im nächsten Zyklus von ihrem eigenen Baustein zur Berechnung benötigen werden (z.B. Integratoren oder Zeitbausteine) dürfen nicht temporär sein. (..wird beim Übersetzen vom GPE geprüft)
- Rückgekoppelte Ausgangs-Variable (SPS-Variable die von vorgelagerten Bausteinen zur Berechnung benötigen werden) dürfen ebenfalls nicht temporär sein. (..wird beim Übersetzen vom GPE geprüft)

Da temporäre Variable typischerweise mehrfach (an verschiedenen Programmstellen) benutzt werden, kann in der Betriebsart ***ONLINE*** für temporäre Variable kein gültiger Wert ausgelesen werden. In diesen Fällen versucht der ***GPE*** den Wert dieser Variablen zu berechnen. Wenn das nicht möglich ist, dann wird die Variable ***Weiß*** dargestellt.

Kapitel 10

Konstanten

Über das Menü „Bausteine" können analoge(Festpunkt, Integer, Zeit) und digitale(true, false) Konstanten-Bausteine erzeugt werden. Bei den analogen Konstanten kann über das Parameterformular gewählt werden, ob der Konstanten-wert nur im 'Flash' abgelegt wird, oder ob eine SPS-Konstante (EEPROM+RAM) erzeugt werden soll. SPS-Konstanten haben den Vorteil, daß sie zur Laufzeit geändert werden können. FLASH-Konstanten haben den Vorteil, daß sie kein „RAM" verbrauchen.

Der ***GPE*** unterstützt für analoge Eingänge auch so genannte ***Eingangskonstanten.*** Durch anklicken des Eingangs-Pinnamens mit der rechten Maustaste erscheint ein Kontexmenü, über welches man eine Eingangskonstante erzeugen kann. Beim Erzeugen der Konstante muss noch gewählt werden in welchem Format der Konstantenwert dargestellt werden soll. Zur Zeit werden die Anzeige-Formate: ***Festpunkt***, ***Dezimal***, ***Hexadezimal***, ***Zeit*** und ***Winkel*** unterstützt.

Eingangskonstanten sind **immer** FLASH-Konstanten. Sie sind zur Laufzeit nicht änderbar.

Kapitel 11

Bausteine

Über das Menü „Bausteine" können Programm-Bausteine ausgewählt und in die SPS-Grafik eingefügt werden.

Jeder Baustein enthält im oberen Bereich eine Typ-Bezeichnung und im unteren Bereich eine Nummer. Wenn ein Baustein Parameter enthält, dann sind diese Parameter (Name und Wert) unter dem Baustein zu sehen.

Durch drücken eines Bausteins mit der linken Maustaste wird dieser Baustein gehalten und kann verschoben werden. Durch drücken eines Bausteines mit der rechten Maustaste wird ein Baustein-Menü aktiviert, mit dessen Hilfe der Baustein modifiziert werden kann (..spiegeln, löschen, Parameter ändern usw.)

Analoge Bausteinanschlüsse (Eingänge oder Ausgänge) werden „gelb", digitale Anschlüsse „blau" dargestellt. Da eigentlich ein Bit ausreichen würde, benutzen einige digitale SPS-Bausteine die restlichen Bits für weitere interne Informationen (z.B. Zähler-variable für Zeitbausteine).Beim Verbinden von Anschlüssen durch Linien können nur gleichartige Anschlüsse (digital mit digital) oder (analog mit analog) miteinander verknüpft werden.

Zur Zeit werden folgende Bausteinarten unterstützt:

- ***Analoge Standardbausteine:*** ... (Analog_In, Analog_Out, ADD, SUB, MUL, DIV ...)
- ***Digitale Standardbausteine:*** ... (Digital_In, Digital_Out, AND, OR, NOT, ...)
- ***Telegrammbausteine:*** ... STN-Protokoll(RS-485), CAN, CAN-OPEN, TP2(Bedien-Tableau), MODBUS(Slave)
- ***Makrobausteine:*** ... SPS-Strukturen werden durch einzelne Bausteine repräsentiert
- ***Sub-Prozesse:*** ... SPS-Programme(Dateien) werden durch einzelne Bausteine repräsentiert
- ***Funktionsblöcke:*** ... SPS-Unterprogramme
- ***Digitale Formelbausteine:*** ... Der Wert der logischen Ausgangs-Variable wird durch boolsche Verknüpfung der Eingangs-Variablen berechnet.
- ***Analoge Formelbausteine:*** ... Der Wert der analogen Ausgangs-Variable wird durch arithmetische Verknüpfung der Eingangs-Variablen berechnet.

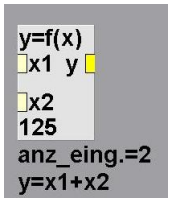
- ***Strukturierter Text:*** ... Die Funktion des Bausteins wird in „Pascal" ähnlicher Syntax angegeben.
- ***Zustands-Automaten:*** ... Der Wert der Ausgangsvariablen wird über einen (ebenfalls grafisch erstellten) Zustandsautomaten berechnet.
- ***Sonstige:*** ... Textbausteine(Kommentar) und Displaybausteine zum Darstellen von analogen Signalverläufen.

11.1 Komplexe Bausteine

11.1.1 Formelbausteine

Bei den Formelbausteinen wird zwischen analogen und digitalen Bausteinen unterschieden. Diese Bausteine verknüpfen die Werte ihrer Eingänge nach einer parametrierbaren Berechnungsvorschrift (Formel) und geben das Ergebnis am Bausteinausgang aus. Jede Berechnungsvorschrift wird von links nach rechts abgearbeitet. Formeln können Klammern mit beliebiger Schachtelungstiefe enthalten.

Analoge Formeln:



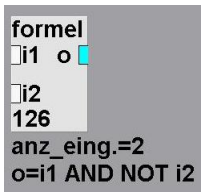
Erlaubte Operatoren in analogen Formeln sind: + - * / sin cos tan cot sqr sqrt abs pw(+ Winkel) mw(- Winkel)

Erlaubte Werte in analogen Formeln sind die Namen der Eingänge(x1, x2..) und konstante Festpunktwerte (z.B. 0.32 ..0.77). Wenn mit Konstanten multipliziert wird, dann darf der Wert dieser Konstanten auch größer als 1 sein (z.B. x1*3.7). Konstanten >1 dürfen nicht am Anfang einer Multiplikation stehen, da der Festpunktakku dann mit einem Wert >1.0 geladen werden müsste: also nicht (2.5*x1) sondern (x1*2.5).

Beispiel einer analogen Formel: y=x1+x2-0.11+(sin(x3 pw 0.5)*3.1415926)

Bei der Benutzung von Formeln muss beachtet werden, daß die verwendete Festpunktarithmetik einen Wertebereich von +/-1 hat. Auch Zwischenergebnisse werden auf +/-1 begrenzt. Bei der Winkel-addition(pw) bzw. -subtraktion(mw) gibt es keine Begrenzung, stattdessen werden die als Winkel interpretierten Argumente (+/-1 entspricht +/-pi) „mit Übertrag" addiert (z.B. 0.9+0.6 = -0.5).

Digitale Formeln:

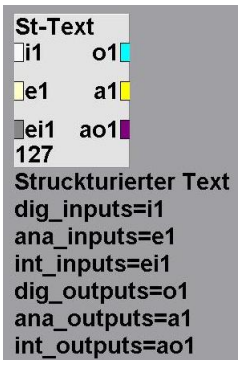


Erlaubte Operatoren in digitalen Formeln sind: AND & OR | NOT /

Erlaubte Werte in digitalen Formeln sind nur die Namen der Eingänge(i1, i2..)

Beispiel einer digitalen Formel: o=(i1 or i2) and not(i1 & /i3)

11.1.2 Strukturierter Text



Mit ***„Strukturiertem-Text"*** können die Funktionen von SPS-Bausteinen in einer Programmiersprache mit ***Pascal*** ähnlichen Syntax programmiert werden.

Erlaubte VariablenNamen für ***Ein-*** und ***Ausgangs-variable*** sind dabei die parametrierbaren Namen der ***Ein-*** und ***Ausgänge*** des Bausteins. Außerdem können ***Lokale***- und ***Temporäre***-Variable deklariert werden. Lesender Zugriff ist für alle Variable erlaubt. Schreibender Zugriff auf ***Eingangs-variable*** ist verboten (nicht möglich).

Neben den Datentypen ***FRACT***(Festpunkt) und ***BOOL***(Logisch/Digital) wird beim Strukturierten-Text auch der Datentyp ***INTEGER***(ganzeZahl) unterstützt. INTEGER Variable werden z.B. benötigt wenn indizierten auf Register zugegriffen werden soll, oder wenn Schleifen mit Lauf-variablen programmiert werden sollen.

Lokale Variable (VAR_LOCAL) haben Initial (nach Programmstart) den Wert „0". Wenn das nicht passt, dann können lokale Variable im Deklarationsteil auch initialisiert werden (...var1: FRACT := 0.32;). Lokale Variable behalten ihren Wert bis zum nächsten Zyklus/Programm-Aufruf. Sie haben ein „Gedächtnis".

Temporäre Variable (VAR_TEMP) haben kein „Gedächtnis". Ihr Wert ist bei jedem Programm-Aufruf undefiniert, weil sich alle ***St-Text Bausteine*** den selben Satz von temporären Variablen teilen.

Am Anfang des Deklarationsteils können Konstanten vereinbart werden. Enthält der Zahlenwert einer Konstante einen Punkt, dann wird dieser Konstanten der Datentyp ***FRACT***(Festpunkt) zugeordnet. Enthält eine Zahl keinen Punkt, dann wird der zugehörigen Konstante der Datentyp ***INTEGER*** zugeordnet.

Als Programmieranweisung gibt es die ***„Zuweisung (var := Ausdruck)"*** , die ***"IF THEN ELSE"*** Fallunterscheidung, die ***„WHILE"*** Schleife, die ***„REPEAT"*** Schleife und die ***„FOR"*** Schleife.

Bei der Auswertung von Ausdrücken wird die Regel „Punktrechnung geht vor Strichrechnung" beachtet. Außerdem können Ausdrücke durch Klammern beliebig geschachtelt werden.

Beispiel1: 4+2*3=10

Beispiel2:(4+2)*3=18

Kommentare können folgendermaßen geschrieben werden:

1. (* Kommentar *)
2. /* Kommentar */
3. // Kommentar bis Zeilenende

Ein vollständiges „ST-Text" Programm kann z.B. so aussehen:

```
PROGRAM ProgrammName; (* ProgrammName = BausteinName *)
CONST
    c1=21; Cons2=-0.73;
VAR_TEMP // Variable ohne Gedächtnis
    i, k :INTEGER;
VAR_LOCAL // Variable mit Gedächtnis
    var1 : INTEGER;
    var2 : FRACT := 0.99; //..initialisierte Variable
BEGIN_PROGRAM
    da1 := NOT(de1 & de2);  // Zuweisung
    IF ae1<Cons2 THEN // if then else...
       ia1:= ie1 DIV 5;
    ELSIF ae1<0.23 THEN
       aa1 := 10.41 * ae1 + SIN(ae2 - 0.33);
    ELSE
       da3 := da2 OR da3;
    END_IF;
    k :=27;
    FOR i:=1 TO k+3 BY 2 DO // For-Schleife
       ok := fillBit(800,900,i); // Reg. 800, bit i auf TRUE setzen
    END_FOR;
END_PROGRAM /* Ende St-Text */
```

11.1.2.1 Zugriff auf Register

Für den Zugriff auf Register steht folgender Satz von Funktionen zur Verfügung:

1. ok := **getBit**(RegAnf,RegEnd,BitNr,BoolVar); Der Wert des Bits „BitNr" des Registers „RegAnf" wird in die Variable BoolVar geschrieben.
2. ok := **setBit**(RegAnf,RegEnd,BitNr,BoolVar); Der Wert der Variablen „BoolVar" wird in das Bit „BitNr" des Registers „RegAnf" geschrieben.
3. ok := **fillBit**(RegAnf,RegEnd,BitNr); Das Bit „BitNr" des Registers „RegAnf" wird auf ***TRUE*** gesetzt.
4. ok := **clrBit**(RegAnf,RegEnd,BitNr); Das Bit „BitNr" des Registers „RegAnf" wird auf ***FALSE*** gesetzt.
5. ok := **setHiVal**(RegAnf,RegEnd,Offset,Var); Das höherwertige Wort der Variablen „Var" wird in das Register „RegAnf+Offset" geschrieben.
6. ok := **setLoVal**(RegAnf,RegEnd,Offset,Var); Das niederwertige Wort der Variablen „Var" wird in das Register „RegAnf+Offset" geschrieben.
7. ok := **setVal**(RegAnf,RegEnd,Offset,Var); Der Wert der Variablen „Var" wird in das Register „RegAnf+Offset" und „RegAnf+Offset+1" geschrieben.
8. ok := **getHiVal**(RegAnf,RegEnd,Offset,Var); Das höherwertige Wort der Variablen „Var" wird mit dem Inhalt des Registers „RegAnf+Offset" beschrieben.
9. ok := **getLoVal**(RegAnf,RegEnd,Offset,Var); Das niederwertige Wort der Variablen „Var" wird mit dem Inhalt des Registers „RegAnf+Offset" beschrieben.
10. ok := **getVal**(RegAnf,RegEnd,Offset,Var); Die Variable „Var" wird mit dem Inhalt der Registers „RegAnf+Offset" und „RegAnf+Offset+1" beschrieben.

Die Parameter „***RegAnf***" und „***RegEnd***" sind die (zur Laufzeit) überwachte untere- und obere Grenze für Registerzugriffe.

Das Funktionsergebnis(ok) wird dann auf „FALSE" gesetzt, wenn versucht wird auf Register außerhalb des erlaubten Bereiches ***(RegAnf..RegEnd-1)*** zuzugreifen.

Für lesende Register-Zugriffe (bei der ZKG512) ist der maximal erlaubte Registerbereich: 0..2600.

Für schreibende Register-Zugriffe (bei der ZKG512) ist der maximal erlaubte Registerbereich: 600..2600.

Der Parameter „***BitNr***" darf auch größer als 15 sein. Es wird dann auf Bits der folgenden Register zugegriffen. Dadurch können Bitfelder (beginnend mit ***RegAnf***) bearbeitet werden.

11.1.2.2 Integer Funktionen

Für Integer Variable stehen zusätzlich zur Arithmetik (+, -, * und DIV) folgende Funktionen zur Verfügung:

1. **INC**(Var,val); Die Variable „Var" wird um den Wert „val" erhöht. Wenn val nicht angegeben ist, dann wird um „1" erhöht.
2. **DEC**(Var,val); Die Variable „Var" wird um den Wert „val" vermindert. Wenn val nicht angegeben ist, dann wird um „1" vermindert.
3. **MOD** oder **%** : Es wird der (ganzzahlige) Rest einer Integerdivision berechnet. Beispiel: 13 MOD 10 = 3

Integer Multiplikation und Division ist hier nur (...für Adress-Berechnungen) als ***Unsigned_16Bit*** implementiert.

11.1.2.3 Festpunkt Funktionen

Neben den Grundrechenarten (+ , - , * , /) werden folgende Festpunkt-funktionen unterstützt:

1. **Addiere Winkel**: +< ; Beispiel: aa1:=aa1 +< 0.9; // Addition ohne Begrenzung, wie z.B ..180°+90°= -90°
2. **Subtrahiere Winkel**: -< ; Beispiel: aa1:=aa1 -< 0.9; // Subtraktion ohne Begrenzung, wie z.B ..-160°- 30°= +170°
3. **Absolutwert**: ABS; Beispiel: aa1:=ABS(-0.84); // Ergebnis ist +0,84
4. **Quadratwurzel**: SQRT; Beispiel: aa1:=SQRT(0.25); // Ergebnis ist +0,50;
   Achtung: SQRT(-x) => -SQRT(x)
5. **Winkelfunktionen**: SIN, COS, TAN, COT, ARCSIN; Beispiel: aa1 := SIN(ae1);

Es werden folgende Relationen unterstützt:

1. **Gleich**: =; Beispiel: IF ae1=0.71 THEN ...
2. **Ungleich**: <>, #; Beispiel: IF (ae1#ae2) OR (ae1<>ae3) THEN ...
3. **Größer**: >; Beispiel: IF ae1 > ae1 THEN ...
4. **Größer oder gleich**: >=; Beispiel: IF ae1 >= ae1 THEN ...
5. **Kleiner**: <; Beispiel: IF ae1 < ae1 THEN ...
6. **Kleiner oder gleich**: <=; Beispiel: IF ae1 <= ae1 THEN ...

11.1.2.4 Boolesche Funktionen

Es werden folgende Boolesche bzw. Digitale Funktionen unterstützt:

1. **Und** Verknüpfung: AND bzw. &; Beispiel: da1:= de1 & de2 AND de3;
2. **Oder** Verknüpfung: OR bzw. | ; Beispiel: da1:= de1 | de2 OR de3;
3. **Negierung**: NOT bzw. ! oder ~ ; Beispiel: da1:= NOT (~ (! de1)); // 3-fache Negierung

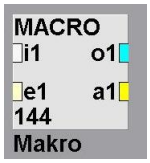
Es werden folgende Boolesche bzw. Digitale Relationen unterstützt:

1. **Gleich**: = ; Beispiel: „IF de1=TRUE THEN ...
2. **Ungleich**: <>, # ; Beispiel: IF de1#de2 THEN...

11.1.2.5 Strukturierter-Text Klone-Bausteine

Über den Baustein-Menüpunkt ***„klonen"*** können Strukturierter-Text **Klone-Bausteine** erzeugt werden. Änderungen an einem Klon werden (automatisch) auf alle Klone gleichen Namens übertragen.

11.1.3 Makros



Makros sind selbst-definierte SPS-Bausteine und können beliebige SPS-Strukturen enthalten. Makros verbergen komplexe Strukturen und erhöhen so die Übersichtlichkeit eines Programms.

Makros können, wie andere Bausteine auch, kopiert werden. Änderungen am Original wirken sich nicht auf Kopien aus. Um das Projekt wartbar zu halten, wird aber (beim Übersetzen und beim Schließen von Makros) geprüft, ob es Makros gleichen Namens aber mit verschiedenen Inhalten gibt. Makros die sich nur durch die Parameter ihrer Bausteine unterscheiden gelten als gleich.

Über den Baustein-Menüpunkt „Öffnen" gelangt man in die innere Struktur eines Makros und kann hier (im Edit-mode) auch Änderungen vornehmen.

Über den Baustein-Menüpunkt(..Kontexmenü) „sichern" kann man Makros abspeichern, und sich so eine „Makrobibliothek" anlegen. Über das Menü „Datei/Makro-Laden" können abgespeicherte Makros wieder geladen werden.

Neue Makros erzeugen: Aus dem Menü „Bausteine" wählt man „Makro" und platziert dieses Makro auf dem Arbeitsblatt. Über den Baustein-Menüpunkt „öffnen" gelangt man ins Innere des noch leeren Makros. Hier kann man über das „Bausteine" Menü analoge oder digitale Makroanschlüsse (..PortIn, PortOut) erzeugen. Diese Makroanschlüsse müssen mit Anschluss-Name(3 Zeichen) und Anschluss-Nr. parametriert werden. Die Anschluss-Nr. bestimmt die Position des Anschlusses an der Makro-Hülle (Nr.1 ist immer ganz oben).

Damit diese Anschlüsse auch auf der äußeren Makro-Hülle sichtbar werden, muss jeder Anschluss über den Baustein-Menüpunkt „einschalten" aktiviert werden. Beim „einschalten" wird geprüft, ob Anschluss-Name und Nr. gültige Werte enthalten bzw. ob diese Namen und Nummern schon in anderen (aktivierten)

Anschlüssen des Makros benutzt wurden. In solch einem Fall wird eine Fehlermeldung erzeugt, und der Anschluss wird nicht aktiviert. Wenn kein Fehler vorhanden ist, dann wird der Anschluss auch auf der äußeren Makro-Hülle sichtbar, und kann benutzt werden.

Abschließend erzeugt man im Makro SPS-Strukturen, welche mit den Anschlüssen des Makros beliebig verbunden werden können. Es ist auch erlaubt Makros in Makros zu benutzen (Rekursionen sind nicht möglich).

Makros aktualisieren: Mit dem Baustein-Menüpunkt ***updateMacros*** kann erreicht werden, dass Änderungen an einem Makro auf alle andere Makros gleichen Namens übertragen werden.

Makros verriegeln: Makros können durch Passwörter[1] gegen Änderungen durch Unbefugte geschützt werden. Das ist z.B. dann sinnvoll, wenn weitergegebene Makros sicherheitsrelevante Schutzfunktionen enthalten, die nicht verändert werden dürfen. Zum Verriegeln wählt man das Kontexmenü ***Verriegeln*** und muß dann ein Passwort (zweimal) eingeben. Zum Öffnen des Makros muss dann immer dieses Passwort eingegeben werden. Zum ***Entriegeln*** eines Makros muß das gleiche Passwort eingegeben werden. Wenn man das Passwort vergessen hat, dann kann mit Hilfe des ***Makro-Namens*** ein ***Freischaltcode***[2] berechnet werden, mit dem das Makro entriegelt werden kann.

Zum entriegeln eines Makros mittels ***Freischaltcode*** geht man folgendermaßen vor:

- Im Passwortdialog nichts eingeben, sondern nur OK drücken.
- In dem daraufhin erscheinenden Dialog den ***Freischaltcode*** eingeben und OK drücken.

[1] Passwörter werden nach dem MD5-Algorithmus verschlüsselt.
[2] Der Freischaltcode kann bei der Fachabteilung unter Angabe des Makro-Namens erfragt werden.

11.1.4 Digitale Filter

Mit Hilfe von SPS-Bausteinen können auch „***Digitale Filter***" realisiert werden.

Die Koeffizienten der Übertragungsfunktion H(z) Digitaler-filter können mit Hilfe von Filterentwurfs-programmen ermittelt werden.

Zum Beispiel mit dem *freien* Filterentwurfs-programm ***DFCGEN***
Siehe: http://home.arcor.de/dfcgen/

Oder mit Hilfe der ***„Signal-processing"*** Funktionen des Programmes ***OCTAVE***[3].
Dokumentation siehe: http://octave.sourceforge.net/signal

Im folgenden Beispiel soll ein Tiefpass 8.Ordnung (Tschebyscheff II) mit einer Grenzfrequenz von 49 Hz und einer Dämpfung von 60 dB im Sprerrbereich berechnet werden. Die Abtastfrequenz Fs sei 1000 Hz.

Mit dem **OCTAV** Aufruf: **[b,a]=cheby2(n,Rs,Wn,)** werden die Zähler(b) und Nenner(a) Koeffizienten der Übertragungsfunktion H(z) ermittelt. Dabei ist:

- **n** = Die Ordnung des Filters
- **Rs** = Die Dämpfung (in dB) im Sprerrbereich
- **Wn** = Die Grenzfrequenz im Sperrbereich

Die Grenzfrequenz Wn liegen im Bereich 0..1 und bezieht sich auf die halbe Abtastfrequenz (Fs/2=1.0).

also: Rs=60; Wn=49/500; [b,a]=cheby2(8,Rs,Wn)

Durch Aufruf der OCTAVE-Funktion ***freqz(b,a)*** kann der Frequenzgang des Filters grafisch dargestellt werden.

Da Filter höherer Ordnung eine hohe Koeffizientengenauigkeit erfordern und zu numerischen Instabilitäten neigen, ist es besser solche Filter durch eine Reihenschaltung aus Filtern 2. Ordnung zu realisieren.

Die Berechnung der Koeffizienten für Filterblöcke 2. Ordnung, aus einem Filter höherer Ordnung, erfolgt mit dem **OCTAVE**-Kommando: **[sos,g]=tf2sos(b,a)**. **Sos** steht für „**second order system**", **g**(gain) ist die benötigte Gesamtverstärkung über alle Blöcke.

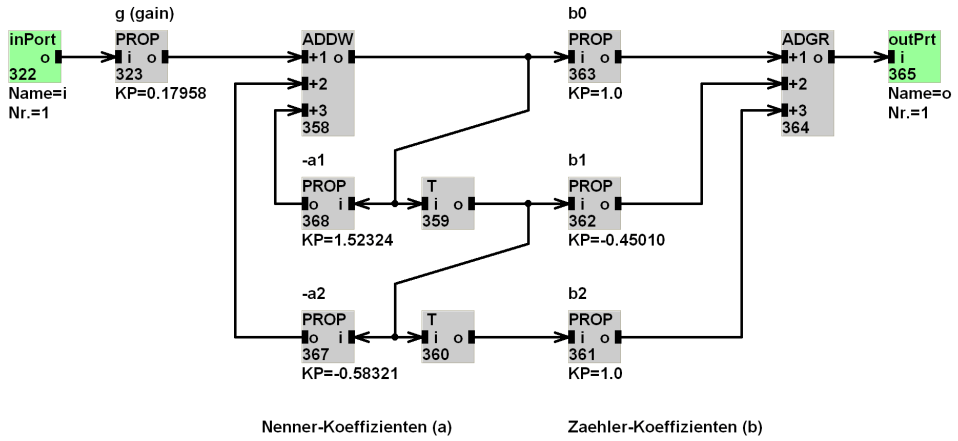
Ausgabeformat der Ergebnisses von ***tf2sos:***

```
b01 b11 b21 1 a11 a21      1.0 -0.45010 1.0 1 -1.52324 0.58321
b02 b12 b22 1 a12 a22      1.0 -1.71056 1.0 1 -1.63642 0.68917
b03 b13 b23 1 a13 a23      1.0 -1.86539 1.0 1 -1.77510 0.82065
b04 b14 b24 1 a14 a24      1.0 -1.90233 1.0 1 -1.89855 0.94113
   g = verstärkung(gain)          0.0010401
```

Die Zähler(b) und Nenner(a) Koeffizienten werden als Parameter(KP) in die **PROP**ortional Bausteine der Filterblöcke eingetragen.

[3] ein freies Programm für numerische Berechnungen (ähnlich wie MATLAB)

Abbildung 11.1: Filterblock 2.Ordnung

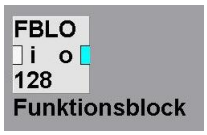


Filterblock 2.Ordnung realisiert durch SPS-Bausteine:

Proportional-bausteine mit einer Verstärkung von 1 (KP=1.0) können auch weggelassen und durch eine Verbindungslinie ersetzt werden.

Wenn es beim Testen zu Überläufen kommt (Begrenzung der Ausgangswerte auf +/-1.0), dann müssen die Verstärkungsfaktoren der jeweiligen Blöcke herunter-gesetzt werden. Nach dem letzten Filterblock muss dann ein weiterer Proportional-baustein eingefügt werden, damit die Gesamtverstärkung (g) den geforderten Wert hat.

11.1.5 Funktionsblöcke



Funktionsblöcke (FBLO's) sind SPS-Programme die nur dann ausgeführt werden, wenn an ihren Eingängen der Wert *TRUE* anliegt. Der Parameter „Name" eines Funktionsblockbausteins gibt den Namen eines zuvor erzeugten Funktionsblock-Programms an. Ein Funktionsblock kann auch mehrfach innerhalb eines Programms/Prozesses eingefügt werden. Der Programm-Code dieses Blockes wird dann in jedem SPS-Zyklus ggf. auch mehrfach durchlaufen. Die Anzahl unterschiedlicher Funktionsblöcke ist z.Z. auf „32" begrenzt.

Am Ausgang eines Funktionsblockes liegt immer der Wert des Eingangs an (der Eingangswert wird auf den Ausgang übertragen).

Wenn ein Funktionsblock mehrfach in einem Programm benutzt wird, dann muss man beachten, daß dieser Fkt-Block jedes mal mit dem selben Variablensatz arbeitet.

Zeitabhängige Bausteine (z.B. Anzugsverzögerungen) in Funktionsblöcken sind problematisch, weil die Zeit in Funtionsblöcken *stehen bleibt*, wenn sie nicht *enabled* sind, d.h. wenn sie nicht ausgeführt werden. Wird ein Funktionsblock mehrfach in einem Programm ausgeführt, dann vergeht die Zeit in diesem Funktionsblock sogar schneller.

11.1.6 Sub-Prozesse



Sub-Prozesse sind eigenständige SPS-Prozessdateien, welche durch Sub-Prozess Bausteine repräsentiert werden.

Der einzige Parameter dieser Bausteine ist der Name der zugeordneten SPS-Programmdatei. Wenn diese SPS-Programmdatei beim Öffnen des Sub-Prozesses noch nicht existieren, dann wird sie (beim Öffnen) erzeugt.

Sub-Prozesse besitzen „dummy-Anschlüsse". Über diese Anschlüsse werden keine Daten übertragen. Sie können aber zur Festlegung der Baustein-Abarbeitungsreihenfolge genutzt werden.

Beim Übersetzen wird der Code der Sub-Prozesse mittels „include" Mechanismus in übergeordnete SPS-Prozesse eingebunden. Der Datenaustausch zwischen Prozess und Sub-Prozess erfolgt (falls erforderlich) über selbst-definierte SPS-Variable oder über Register.

11.1.7 Zustandsautomaten

Mit Zustandsautomaten können komplexe logische zusammenhänge übersichtlich programmiert und dargestellt werden. Ähnlich wie Makros können auch selbstdefinierte Zustandsautomat-Bausteine erstellt werden. Nach außen ist nur die Hülle des Automaten mit logischen Ein- und Ausgängen sichtbar. Der Zustand der logischen Ausgänge eines Automaten ist abhängig vom Zustand der logischen Eingänge und vom internen Zustand des Automaten. Der Wert des State-Nr-Ausgangs(Integer) eines Automaten ist nur vom internen Zustand des Automaten abhängig.

Zustandsautomaten werden aus folgenden Bausteinen zusammengesetzt:

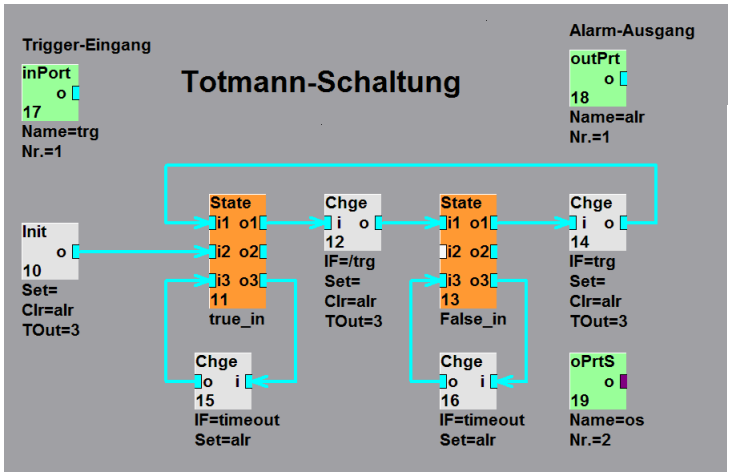
- **Zustände (State)**: Zustände haben einen Namen, eine optionale State-Nr. sowie ein- und Ausgänge. Von allen Zustands-Bausteinen eines Automaten ist immer genau einer der ***aktive Zustand***. In der Simulation oder im Online-betrieb wird der aktive Zustandsbaustein durch die Farbe ***gelb*** gekennzeichnet. Für den Parameter sNr.(State-Nr.) kann eine beliebige Integer-Zahl eingetragen werden. Die ***State-Nr.*** des aktiven Zustands wird in den Betriebsarten Simulation oder Online auf den 'State-AusgangsPin' übertragen.
- **Zustandsübergänge (StateChange)**: Verbinden Ausgänge von Zuständen(State Bausteine) mit Eingängen von anderen Zuständen. Zustandsübergänge werden mit einer Übergangsbedingung und mit Übergangsaktionen parametriert. Die Übergangsbedingung kann eine boolsche Formel sein, welche die Eingänge des Automaten verknüpft, sie kann aber auch aus den Schlüsselwörtern ***true, false*** oder ***timeout*** bestehen. Wenn die Übergangsbedingung wahr(true) ist, dann werden die parametrierten Übergangsaktionen ausgeführt, und der Folgezustand wird zum aktiven Zustand. Die Übergangsaktionen werden durch drei Parameterzeilen festgelegt. In der Parameterzeile „Set" werden alle Ausgänge des Automaten eingetragen, die beim Zustandsübergang auf logisch_1=true gesetzt werden sollen. In der Parameterzeile „Clr" werden alle Ausgänge des Automaten eingegeben, die beim Zustandsübergang auf logisch_0=false gesetzt werden sollen. In der Parameterzeile „TOut" wird das TimeOut in Sekunden angegeben, das für den nächsten aktiven Zustand gelten soll (TimeOut=0 bedeutet: ohne TimeOut Überwachung. Timeout=-1 bedeutet den aktuellen Wert des Timeout-zählers nicht zu verändern.).
- **Initialisierung (InitStateChange)**: Dieser Initial Zustandsübergangs-Baustein (ohne Übergangsbedingung) wird beim Start einmal ausgeführt und initialisiert den Automat durch seine Übergangsaktionen. Der Folgebaustein des Init-Bausteins wird nach dem Start zum „aktiven Zustand".

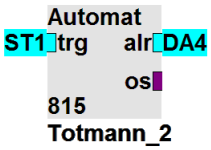
- **Ein- und Ausgänge (InPort, OutPort, OutPortS):** Diese Bausteine repräsentieren die logischen Ein- bzw. Ausgangsvariable des Automaten. Nachdem sie (wie bei Makros) eingeschaltet bzw. aktiviert worden sind, wechselt ihre Farbe von Dunkelgrün nach Hellgrün, und sie sind auch auf der äußeren Automatenhülle sichtbar. Der Baustein OutPortS (..OutPortState) ist vom Typ Festpunkt(Integer) und enthält immer die (frei wählbare)Zustands-Nr. des aktuellen Zustands.

Hat ein Zustand(State) mehrere Ausgänge mit Zustandsübergängen, dann werden diese Übergänge von ***oben*** nach ***unten*** abgearbeitet, bis eine Übergangsbedingung gefunden wird welche den wert ***true*** liefert. Liefert kein Zustandsübergang den Wert ***true,*** dann findet auch kein Zustandsübergang statt. Der *alte* „aktive Zustand" ist der *neue* „aktive Zustand".

Beim Erstellen eines Zustandsautomaten sollte man besondere Sorgfalt auf das Verlegen der Verbindungslinien (möglichst wenig Kreuzungen) legen, da das Zustandsdiagramm ansonsten schnell unübersichtlich wird.

Beispiel: Zustandsdiagramm mit übergeordnetem Zustandsautomat-Baustein.





11.2 Analoge Standardbausteine

Das Ergebnis von Analog(Festpunkt)-operationen wird immer im internen Festpunktakkumulator gespeichert. Wenn der jeweilige Bausteinausgang eine Variable besitzt, dann wird das Ergebnis auch in diese Variable geschrieben.

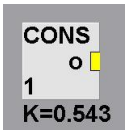
Immer wenn im weiteren Text der Festpunktwert +1.0 genannt wird, dann ist damit eigentlich der Wert +0.99999999953 gemeint, weil im verwendeten 32-Bit-Festpunktformat(ZM432) der Zahlenbereich von -1.0 bis +1.0 - $(1/2^{31})$ geht. Wenn das 16-Bit Festpunktformat verwendet wird, dann ist die größte positive Zahl = 1.0 - $1/2^{15}$ = 0.99996948.

11.2.1 Konstanten-Bausteine (CONS, xCONS, tCONS)

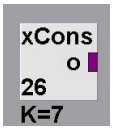
Mit Hilfe dieser Bausteine können Konstanten erzeugt werden. Im Parameterformular dieser Bausteine kann der gewünschte Wert eingegeben werden.

Durch Anklicken des letzten Eintrages der Parameterzeile(...) kann gewählt werden, ob der Parameter im EEPROM gespeichert und dadurch zur Laufzeit(Betriebsart Online) änderbar ist, oder ob der Parameter im Flash abgelegt und dadurch zur Laufzeit nicht änderbar ist. Der Vorteil der Flash-Speicherung liegt darin, dass weniger EEPROM und SPS-Variable verbraucht werden.

Zur Zeit gibt es drei verschiedene Arten von Analog-Konstanten, welche sich in der Art der Werteinterpretation bzw. in der Art der Zahlendarstellung unterschieden:

- 

Festpunkt-Konstanten (CONS): Werte dieser Konstanten werden als Festpunktzahl interpretiert. Als Parameter können Wert von (-1.0 bis +1.0) eingegeben werden.

- 

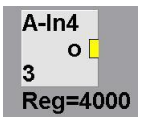
Ganzzahl-Konstanten (xCONS): Werte dieser Konstanten werden als Integerzahl interpretiert. Es können Werte von (MinInt bis MaxInt) in dezimaler- oder in hexa-dezimaler[4] Schreibweise eingegeben werden.

- 

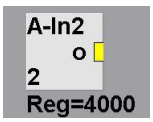
Zeit-Konstanten (tCONS): Werte dieser Konstanten werden als Zeit in Sekunden angezeigt und eingegeben. Intern werden diese Zeiten in Vielfache der Zykluszeit umgerechnet. Der Wertebereich dafür ist auf 16 Bit begrenzt, so das die größte darstellbare Zeit den Wert: $T = Zykluszeit * 2^{16}$ besitzt.

11.2.2 Analog-In Bausteine (A-In2, A-In4, AIn2-Indx, AIn4-Indx)

Bausteine dieses Typs kopieren ***Register-Inhalte*** in ***SPS-Variable***. Zur Zeit gibt es 4 verschiedene ***Analog-In*** Bausteine:

- 

Analog_In (4 Byte): Der Inhalt des parametrierten Quell-Register(hi-word) und des darauf folgenden Register(lo-word) wird in die 4-Byte Ausgangsvariable des Bausteins kopiert.

- 

Analog_In (2 Byte): Der Inhalt des parametrierten Quell-Registers wird in die höherwertigen 2-Byte der Ausgangsvariable des Bausteins kopiert.

[4] hexa-dezimalen Zahlen wird ein „$" Zeichen vorangestellt

- 

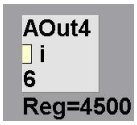
Analog_In (4 Byte) indexd: Der Eingangs-wert ***„idx***" des Bausteins wird als Quell-Register Adress-Offset interpretiert. Die Inhalte des Quell-Registers[***min_reg + idx***] (hi-word) und des darauf folgenden Registers (lo-word) werden in die 4-Byte Ausgangsvariable des Bausteins kopiert. Die Parameter ***„min_reg"*** und ***„max_reg"*** geben den erlaubten Bereich der Register-Adresse an. Wenn die Register-Adresse außerhalb dieses erlaubten Bereiches liegt, dann wird die Fehler-Ausgangs-Variable „***err***" des Bausteins gesetzt.

- 

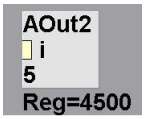
Analog_In (2 Byte) indexd: Der Eingangs-wert ***„idx***" des Bausteins wird als Quell-Register Adress-Offset interpretiert. Der Inhalt des Quell-Registers [***min_reg + idx***] wird in die höherwertigen 2-Byte der Ausgangsvariable des Bausteins kopiert. Die Parameter ***„min_reg"*** und ***„max_reg"*** geben den erlaubten Bereich der Register-Adresse an. Wenn die Register-Adresse außerhalb dieses erlaubten Bereiches liegt, dann wird die Fehler-Ausgangs-Variable „***err***" des Bausteins gesetzt.

11.2.3 Analog-Out Bausteine (AOut2, AOut4, AOut2-Indx, AOut4-Indx)

Bausteine dieses Typs kopieren ***Baustein-Eingangswerte*** in ***Register***. Zur Zeit gibt es 4 verschiedene ***Analog-Out*** Bausteine:

- 

Analog_Out (4 Byte): Der Baustein-Eingangswert(4 Byte) wird in das parametrierte Ziel-Register(hi-word) und in das darauf folgende Register(lo-word) kopiert.

- 

Analog_Out (2 Byte): Die höherwertigen 2-Byte des Baustein-Eingangswertes werden in das parametrierte Ziel-Register kopiert.

- 

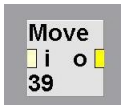
Analog_Out (4 Byte) indexd: Der Eingangswert ***„idx***" des Bausteins wird als Ziel-Register Adress-Offset interpretiert. Der Baustein-Eingangswert „***i***" (4 Byte) wird in dieses Ziel-Register[***min_reg + idx***] (hi-word) und in das darauf folgende Register(lo-word) kopiert. Die Parameter ***„min_reg"*** und ***„max_reg"*** geben den erlaubten Bereich der Register-Adresse an.

Wenn die Register-Adresse außerhalb des erlaubten Bereiches liegt, dann wird die Fehler-Ausgangs-Variable „***err***" des Bausteins gesetzt.

- 

Analog_Out (2 Byte) indexd: Der Eingangswert ***„idx***" des Bausteins wird als Ziel-Register Adress-Offset interpretiert. Die höherwertigen 2-Byte des Baustein-Eingangswertes „***i***" werden in das Ziel-Register[***min_reg + idx***] kopiert. Die Parameter ***„min_reg"*** und ***„max_reg"*** geben den erlaubten Bereich der Register-Adresse an. Wenn die Register-Adresse außerhalb des erlaubten Bereiches liegt, dann wird die Fehler-Ausgangs-Variable „***err***" des Bausteins gesetzt.

11.2.4 Move Bausteine (Move)

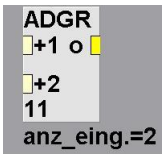


Move Bausteine kopieren ihren Eingangswert direkt in ihre (immer vorhandene) Ausgangsvariable.

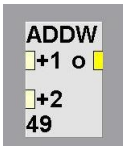
Diese Bausteine können z.B. zur Signalvervielfachung eingesetzt werden, wenn ein Ausgang mit mehr als 32 Eingängen verbunden werden soll. Diese Bausteine können auch dann eingesetzt werden, wenn man Sicherstellen will, das bestimmte Signale in Variable gespeichert werden.

11.2.5 Addierer (ADGR, ADDW)

Addierer können bis zu 32 Eingänge haben. Sie laden den ersten Eingangswert und addieren die anderen Eingangswerte (falls vorhanden) hinzu. Es werden zwei Arten Addierer unterstützt:

- 

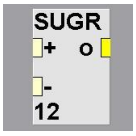
Addierer mit Begrenzung(ADGR): Das Ergebnis von Additionen wird immer auf +/-1.0 Begrenzt. ***Beispiel: 0.9 + 0.6 = 1.0***

- 

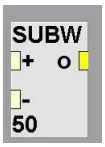
Addierer ohne Begrenzung(ADDW): Das Ergebnis von Additionen wird NICHT Begrenzt, sondern es wird im „2er-Komplement" addiert. ***Beispiel: 0.9 + 0.6 = -0.5*** Diese Art der Addition ist beim addieren von Winkeln nützlich (+/-1.0 wird als +/-180 Grad interpretiert). Deshalb auch der Name ADDW (ADDiere Winkel).

11.2.6 Subtrahierer (SUGR, SUBW)

Subtrahierer haben 2 Eingänge. Sie laden den ersten Eingangswert und subtrahieren davon den zweiten Eingangswert. Es werden zwei Arten Subtrahierer unterstützt:

- 

Subtrahierer mit Begrenzung(SUGR): Das Ergebnis von Subtraktionen wird immer auf +/-1.0 Begrenzt. ***Beispiel: -0.9 - 0.6 = -1.0***

- 

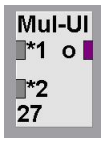
Subtrahierer ohne Begrenzung(SUBW): Das Ergebnis von Subtraktionen wird NICHT Begrenzt sondern es wird im „2er-Komplement" subtrahiert. ***Beispiel: -0.9 - 0.6 = +0.5*** Diese Art der Subtraktion ist beim subtrahieren von Winkeln nützlich (+/-1.0 wird als +/-180 Grad interpretiert). Deshalb auch der Name SUBW (SUBtrahiere Winkel).

11.2.7 Multiplizierer (MUL)



Multiplizierer können bis zu 32 Eingänge haben. Sie laden den ersten Eingangswert und multiplizieren damit die anderen Eingangswerte. Überläufe kommen dabei (bis auf eine Ausnahme) nicht vor. Die Ausnahme stellt das multiplizieren von -1.0 * -1.0 dar. Das korrekte Ergebnis +1.0 ist im verwendeten 32Bit-Festpunktformat nicht darstellbar, weil der Zahlenbereich hier nur von -1.0 bis +1.0 - $(1/2^{31})$ = -1.0 bis +0.99999999953 geht.

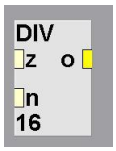
11.2.8 Multiplizierer Unsigned-Integer (MUL_UI)



Dieser Multiplizierer hat 2 Eingänge. Die Eingangswerte werden als vorzeichenlose 16-Bit Zahlen interpretiert. Das Multiplikationsergebnis ist eine vorzeichenlose 32-Bit Zahl. Überläufe kommen nicht vor. Die Aufgabe dieses Multiplizierer ist hauptsächlich das Berechnen von Adressen.

Beispiel: 3*4=12

11.2.9 Dividierer (DIVI)



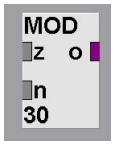
Dividierer haben zwei Eingänge. Sie laden den Zähler-Eingangswert(z) und dividieren ihn durch den Nenner-Eingangswert(n). Das Ergebnis(der Quotient) wird im Festpunktakkumulator und (falls vorhanden) auch in der Ausgangsvariable gespeichert. Bei Überläufen wird das Ergebnis auf +/-1.0 begrenzt.

Beispiele: 0.4/0.5=0.8 0.4/0.1=1.0

Sonderfälle:

- Division durch 0 ergibt +/-1.0 entsprechen dem Vorzeichen des Zählers.
- Null / Null ergibt Null

11.2.10 Modulo (MOD)



Beim Modulo Baustein wird der *Zähler*-Eingangswert(z) durch den *Nenner*-Eingangswert(n) geteilt. Der Divisions-Rest wird im Festpunktakkumulator und (falls vorhanden) auch in der Ausgangsvariable gespeichert.

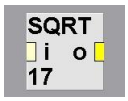
Eingangswerte und Ergebnis sind „Ganze-Zahlen" *(INTEGER)*.

Beispiele: 10 MOD 7 = 3 4 MOD 2 = 0

Sonderfälle:

- ... MOD 0 ergibt 0

11.2.11 Quadratwurzel (SQRT)

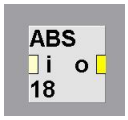


Quadratwurzel-Bausteine laden ihren Eingangswert und ziehen daraus die Wurzel. Das Ergebnis wird in den Festpunktakkumulator und (falls vorhanden) auch in die Ausgangsvariable geschrieben.

Achtung: Negative Eingangswerte liefern auch negative Ergebnisse!

Beispiele: SQRT(0.25)=0.5

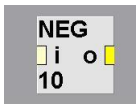
11.2.12 Betrag/Absolutwert (ABS)



Betrags-Bausteine laden ihren Eingangswert und bilden daraus den Absolutwert. Das Ergebnis wird in den Festpunktakkumulator und (falls vorhanden) auch in die Ausgangsvariable geschrieben.

Beispiele: ABS(-0.25) = +0.25 ABS(+0.25) = +0.25

11.2.13 Negierer (NEG)



Negierer laden ihren Eingangswert und ändern das Vorzeichen. Das Ergebnis wird in den Festpunktakkumulator und (falls vorhanden) auch in die Ausgangsvariable geschrieben.

Beispiele: NEG(+0.32) = -0.32 NEG(-0.32) = +0.32

11.2.14 Verdoppler (DOPP)

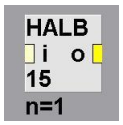


Verdoppler laden ihren Eingangswert und verdoppeln ihn, entsprechend der parametrierten Anzahl(n), n-mal. Das Ergebnis wird in den Festpunktakkumulator und (falls vorhanden) auch in die Ausgangsvariable geschrieben. Bei Überlauf wird das Ergebnis auf +/-1 begrenzt. Im ZielSystem wird verdoppeln durch „***links schieben***" realisiert.

Ausgangswert := Eingangswert * (2^n)

Beispiel: Eingang=0.1, n=3 => 0.8

11.2.15 Halbierer (HALB)

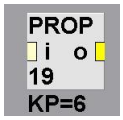


Halbierer laden ihren Eingangswert und halbieren ihn, entsprechend der parametrierten Anzahl(n), n-mal. Das Ergebnis wird in den Festpunktakkumulator und (falls vorhanden) auch in die Ausgangsvariable geschrieben. Bei Überlauf wird das Ergebnis auf +/-1 begrenzt. Im ZielSystem wird halbieren durch „***rechts schieben***" realisiert.

Ausgangswert := Eingangswert / (2^n)

Beispiel: Eingang=0.8, n=3 => 0.1

11.2.16 Proportional-Glied (PROP)

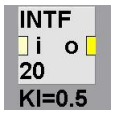


Proportional-Glieder laden ihren Eingangswert und multiplizieren ihn mit dem parametrierten Faktor(KP). Das Ergebnis wird in den Festpunktakkumulator und (falls vorhanden) auch in die Ausgangsvariable geschrieben. Bei Überlauf wird das Ergebnis auf +/-1 begrenzt. Als Proportionalitätsfaktor(KP)sind auch Werte >1.0 erlaubt.

Ausgangswert := Eingangswert * KP

Beispiele: Eingang=0.2, KP=0.5 => 0.1 Eingang=0.2, KP=4.5 => 0.9

11.2.17 Integrator (INTF) mit konstantem Integrationsfaktor

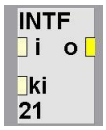


Integratoren laden ihren Eingangswert und integrieren damit ihren Ausgangswert entsprechend dem parametrierten Integrationsfaktor (KI) auf. Das Ergebnis wird in den Festpunktakkumulator und in die (hier immer vorhandene) Ausgangsvariable geschrieben. Bei Überlauf wird das Ergebnis auf +/-1 begrenzt. Als Integrationsfaktor(KI) sind auch Werte >1.0 erlaubt.

Ausgangswert := Ausgangswert + (Eingangswert * KI * Zykluszeit)

Nach der Zeit „T = 1 / KI" hat der Ausgangswert den (konstanten)Eingangswert erreicht. Unabhängig von der gewählten Zykluszeit.

11.2.18 Integrator (INTF) mit variablem Integrationsfaktor

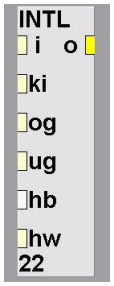


Diese Integratoren laden ihren Eingangswert(i), multiplizieren ihn mit dem Eingangswert(ki) und addieren das Ergebnis zum Ausgangswert. Das Endergebnis wird in den Festpunktakkumulator und in die (hier immer vorhandene) Ausgangsvariable geschrieben. Bei Überlauf wird das Ergebnis auf +/-1 begrenzt.

Ausgangswert := Ausgangswert + Eingangswert(i) * Eingangswert(ki)

Wenn die Integrationszeitkonstante unabhängig von der Zykluszeit sein soll, dann muss der Eingangswert(i) oder der Eingangswert(ki) vor dem integrieren noch mit der Zykluszeit(in Sekunden) multipliziert werden.

11.2.19 Integrator (INTL) mit Begrenzung und Wertvorgabe



Dieser Integrator verhält sich genauso wie der Integrator mit „variablem Integrationsfaktor" (siehe oben).

Zusätzlich besitzt dieser Integrator einen oberen(og) und einen unteren(ug) Grenzwerteingang. Mit diesen Grenzwerten wird der Integrator-Ausgangswert begrenzt.

Außerdem ist ein digitaler „Hand-betrieb" Eingang und ein analoger „Hand-wert" Eingang vorhanden. Immer wenn der „Hand-betrieb" Eingang den Wert ***true*** hat, dann übernimmt der Integrator-Ausgang den Wert des „Hand-wert" Einganges.

11.2.20 Tiefpass (PT1)



Dieser Tiefpass(1.Ordnung) wird mit einer Zeitkonstanten (Zeitangabe in Sekunden) parametriert. Wegen der einfachen (aber schnellen) Berechnungsvorschrift sind nur Zeitkonstanten möglich, welche sich aus Verdoppelungen der Zykluszeit ergeben.

Das Ergebnis wird in den Festpunktakkumulator und in die (hier immer vorhandene) Ausgangsvariable geschrieben. Bei Überlauf wird das Ergebnis auf +/-1 begrenzt.

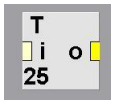
11.2.21 Mittelwertbildung (AVER) über n Messwerte



Dieser Mittelwert(average)-Baustein speichert die letzten „n" Messwerte und berechnet daraus den Mittelwert. Der parametrierbare Wert „n" muss im Bereich von 1 bis 64 liegen.

Messwertglättungen nach dieser Methode haben den Vorteil gegenüber einer Tiefpass-Messwertglättung, dass der Ausgangswert spätestens nach der Zeit ***T=n*Zykluszeit,*** den Endwert erreicht hat.

11.2.22 Verzögerung (T) um eine Zykluszeit



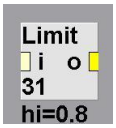
Dieser Baustein liest den Eingangswert, speichert ihn, und gibt ihn im nächsten SPS-Zyklus am Ausgang wieder aus.

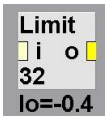
Mit Hilfe dieser Verzögerungs-Bausteine ist es u.a. möglich ***Digitale-Filter 11.1.4*** zu realisieren.

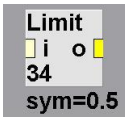
11.2.23 Grenzwert-Bausteine (Limit)

Limit- oder Grenzwert-Bausteine lesen ihren Eingangswert, begrenzen ihn, und geben den begrenzten Wert am Ausgang aus.

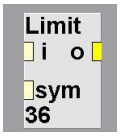
Zur Zeit werden folgende Grenzwert-Bausteine unterstützt:

- 

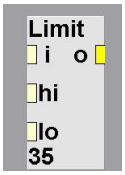
Oberer Grenzwert: Der Baustein besitzt einen oberen, parametrierbaren Grenzwert.
- 

Unterer Grenzwert: Der Baustein besitzt einen unteren, parametrierbaren Grenzwert.
- 

Symmetrischer Grenzwert: Der Baustein besitzt einen parametrierbaren Grenzwert., welcher den Ausgangswert symmetrisch begrenzt. Der Grenzwert muss positiv sein.
- 

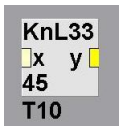
Oberer und Unterer Grenzwert: Der Ausgangswert wird nach oben und nach unten, über zwei unabhängig voneinander parametrierbare Grenzwerte, begrenzt.
- 

Symmetrischer (variabler) Grenzwert: Der Baustein besitzt einen Grenzwert-Eingang. Der Wert dieses Eingangs begrenzt den Ausgangswert symmetrisch. Der Grenzwert kann zur Laufzeit verändert werden. Der Grenzwert muss positiv sein.



- ***Oberer und Unterer Grenzwert (mit variablen Grenzen):*** Der Ausgangswert wird nach oben und unten über zwei Grenzwerteigänge begrenzt. Bei Bausteinen dieser Art können die Grenzwerte zur Laufzeit geändert werden.

11.2.24 Kennlinie mit 33 Stützpunkten (KnL33)

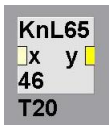


Die 33 Kennlinien-Stützpunkte(Y-Werte) sind im Bereich -1.0 bis +1.0 gleichmäßig über die X-Achse verteilt (äquidistant). Zur Laufzeit wird aus dem Eingangswert(x) des Bausteins, und aus den zugehörigen Y-Werten der Kennlinie, der Ausgangswert(y) berechnet. Werte zwischen den Stützpunkten werden linear interpoliert.

Im Parameterformular dieses Bausteines muss der Name der Kennlinie eingegeben werden. In der Datei TK33.a muss diese Kennlinie mittels „TKF1 Kennlinien-name" und 33 zugehörigen Y-Werten eingetragen werden. Diese Y-Werte können dezimal oder Hexadezimal eingegeben werden.

In der Datei TK33.a ist das Format bzw. die Syntax der Kennliniendefinition beschrieben. Hier kann man auch Beispiel-Kennlinien finden.

11.2.25 Kennlinie mit 65 Stützpunkten (KnL65)

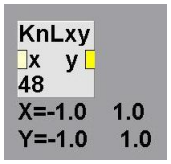


Die 65 Kennlinien-Stützpunkte(Y-Werte) sind im Bereich -1.0 bis +1.0 gleichmäßig über die X-Achse verteilt (äquidistant). Zur Laufzeit wird aus dem Eingangswert(x) des Bausteins, und aus den zugehörigen Y-Werten der Kennlinie, der Ausgangswert(y) berechnet. Werte zwischen den Stützpunkten werden linear interpoliert.

Im Parameterformular dieses Bausteines muss der Name der Kennlinie eingegeben werden. In der Datei TK65.a muss diese Kennlinie mittels „TKF2 Kennlinienname" und 65 zugehörigen Y-Werten eingetragen werden. Diese Y-Werte können dezimal oder Hexadezimal eingegeben werden.

In der Datei TK65.a ist das Format bzw. die Syntax der Kennliniendefinition beschrieben. Hier kann man auch Beispiel-Kennlinien finden.

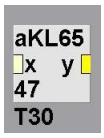
11.2.26 Kennlinie aus (X,Y) Wertepaaren



Mit Hilfe dieses Bausteines kann eine Kennlinie aus Max. 12 (X,Y)-Wertepaaren konstruiert werden. Die Abstände mussen nicht gleichmässig über die X-Achse verteilt sein.

Im Parameterformular dieses Bausteines können maximal 12 Wertepaare (X,Y) eingegeben werden. Der 1. X-wert mus -1, der letzte muss +1 sein. Das ist notwendig, damit die Kennlinie keine undefinierten Bereiche enthält.

11.2.27 Kennlinie mit 65 Stützpunkten (aKL65) ...nur positive X-Werte

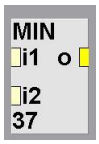


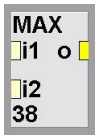
Die 65 Kennlinien-Stützpunkte(Y-Werte) sind im Bereich 0.0 bis +1.0 gleichmäßig über die X-Achse verteilt (äquidistant). Zur Laufzeit wird aus dem Eingangswert(x) des Bausteins, und aus den zugehörigen Y-Werten der Kennlinie, der Ausgangswert(y) berechnet. Werte zwischen den Stützpunkten werden linear interpoliert. Wenn der Eingangswert(x) negativ ist, dann wird als Ergebnis der erste definierte Kennlinienwert y=f(x0) genommen.

Im Parameterformular dieses Bausteins muss der Name der Kennlinie eingegeben werden. In der Datei TK65.a muss diese Kennlinie mittels „TKF2 Kennlinienname" und 65 zugehörigen Y-Werten eingetragen werden. Die Werte-eingabe kann dabei dezimal oder hexadezimal erfolgen.

In der Datei TK65.a ist das Format bzw. die Syntax der Kennliniendefinition beschrieben. Hier kann man auch Beispiel-Kennlinien finden.

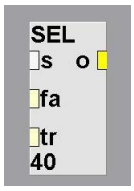
11.2.28 Minimum und Maximum Bausteine (MIN, MAX)





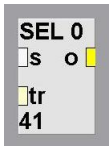
Diese Bausteine wählen aus ihren Eingängen den größten bzw. den kleinsten Wert und geben ihn am Ausgang aus.

11.2.29 Analoger Umschalter (SEL)



Dieser Baustein wählt aus zwei Analogeingängen (**tr**ue-Eingang und **fa**lse-Eingang) in Abhängigkeit von einem digitalen **sel**ect-Eingang, den Ausgangswert. Wenn ***s***=true ist, dann wird der ***tr***-Eingang durchgeschaltet, sonst wird der ***fa***-Eingang zum Ausgang durchgeschaltet.

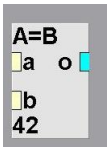
11.2.30 Analoger Einschalter (SEL 0)

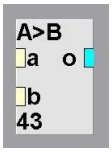


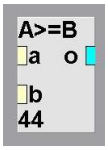
Wenn der Select-Eingang(s)=true ist, dann wird der Wert des ***tr***-Eingangs durchgeschaltet, sonst wird der Wert ***„0.0"*** zum Ausgang durchgeschaltet.

11.2.31 Komparator(Vergleicher) Bausteine (A=B, A>B, A>=B)

Zur Zeit werden drei Komparator-Bausteine unterstützt:

- 

A=B: Wenn der Wert des A-Eingangs gleich dem Wert des B-Eingangs ist, dann wird der Ausgang auf true gesetzt
- 

A>B: Wenn der Wert des A-Eingangs größer als der Wert des B-Eingangs ist, dann wird der Ausgang auf true gesetzt
- 

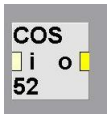
A>=B: Wenn der Wert des A-Eingangs gleich oder größer als der Wert des B-Eingangs ist, dann wird der Ausgang auf true gesetzt

11.2.32 Sinus (SIN)



Der Sinus-Baustein interpretiert das Eingangssignal -1.0 bis +1.0 als Winkel im Bereich von -180 bis +180 Grad, und berechnet daraus die Sinusfunktion. Beispiel: SIN(0.5)=1.0

11.2.33 Cosinus (COS)



Der Cosinus-Baustein interpretiert das Eingangssignal -1.0 bis +1.0 als Winkel im Bereich von -180 bis +180 Grad, und berechnet daraus die Cosinusfunktion. Beispiel: COS(0.5)=0.0

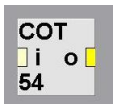
11.2.34 Tangens (TAN)



Der Tangens-Baustein interpretiert das Eingangssignal -1.0 bis +1.0 als Winkel im Bereich von -180 bis +180 Grad, und berechnet daraus die Tangensfunktion. Beispiel: TAN(0.25)=1.0

Das Ergebnis wird immer auf +/-1.0 begrenzt. Das bedeutet, daß das Funktionsergebnis im Bereich von z.B. 45 Grad bis 135 Grad beschnitten wird.

11.2.35 Cotangens (COT)



Der Cotangens-Baustein interpretiert das Eingangssignal -1.0 bis +1.0 als Winkel im Bereich von -180 bis +180 Grad, und berechnet daraus die Cotagensfunktion. Beispiel: COT(0.25)=1.0

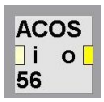
Das Ergebnis wird immer auf +/-1.0 begrenzt. Das bedeutet, daß das Funktionsergebnis im Bereich von z.B. -45 Grad bis +45 Grad beschnitten wird.

11.2.36 ArcusSinus (ASIN)



Der ArcusSinus-Baustein (...die Sinus Umkehrfunktion) berechnet aus dem Eingangswert (-1.0 bis +1.0) die ArcusSinus-Funktion im Bereich von (-90 bis +90 Grad) entsprechend einem Festpunktwert von (-0.5 bis +0.5).

11.2.37 ArcusCosinus (ACOS)

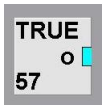


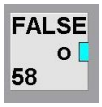
Der ArcusCosinus-Baustein (...die Cosinus Umkehrfunktion) berechnet aus dem Eingangswert (-1.0 bis +1.0) die ArcusCosinus-Funktion im Bereich von (0 bis +180 Grad) entsprechend einem Festpunktwert von (0.0 bis +1.0).

11.3 Digitale Standardbausteine

Das Ergebnis von logischen Operationen wird immer im internen logischen-Arbeitsregister gespeichert. Wenn der jeweilige (digitale)Bausteinausgang eine Variable besitzt, dann wird das Ergebnis auch in diese Variable geschrieben.

11.3.1 Konstanten-Bausteine (TRUE, FALSE)

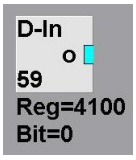




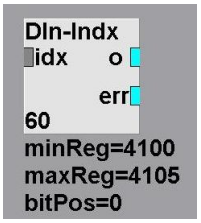
Mit Hilfe dieser Bausteine werden Konstanten erzeugt, die immer den Wert ***TRUE*** bzw. ***FALSE*** haben.

11.3.2 Digital-In Bausteine (D-In, DIn-Indx)

Bausteine dieses Types kopieren den Wert eines ***Register-Bit's*** in ihre ***Ausgangs-Variable***. Zur Zeit gibt es 2 verschiedene ***Digital-In*** Bausteine:

- 

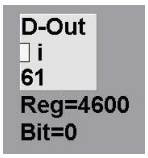
Digital_In: Der Wert des parametrierten Quell-Registers-Bit's wird in die digitale Ausgangsvariable des Bausteins kopiert.

- 

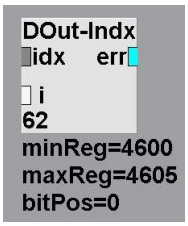
Digital_In (indexd): Der Eingangswert ***„idx"*** des Bausteins wird als Quell-Register-Adresse interpretiert. Der Wert des parametrierten Bits(Bit-Nr.) dieses Quell-Registers wird in die (digitale)Ausgangsvariable des Bausteins kopiert. Die Baustein-Parameter ***„anf_reg"*** und ***„end_reg"*** geben den erlaubten Bereich der Register-Adresse an. Wenn die Register-Adresse außerhalb des erlaubten Bereiches liegt, dann wird die Baustein-Ausgangs-Variable „***err***" gesetzt.

11.3.3 Digital-Out Bausteine (D-Out, DOut-Indx)

Bausteine dieses Types kopieren (digitale) ***Baustein-Eingangswerte*** nach ***Register-Bits***. Zur Zeit gibt es 2 verschiedene ***Digital-Out*** Bausteine:

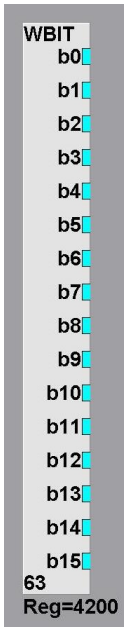
- 

Digital_Out : Der Baustein-Eingangswert wird in das parametrierte Ziel-Register-Bit kopiert

- 

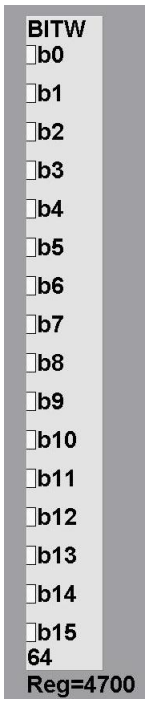
Digital_Out (indexd): Der Eingangswert „***idx***" des Bausteins wird als Ziel-Register-Adresse interpretiert. Der Wert des (digitalen) Baustein-Eingangs „***i***" wird in das parametrierte Bit(Bit-Nr.) des Ziel-Registers kopiert. Die Baustein-Parameter ***„min_reg"*** und ***„max_reg"*** geben den erlaubten Bereich der Register-Adresse an. Wenn die Register-Adresse außerhalb des erlaubten Bereiches liegt, dann wird die Baustein-Ausgangs-Variable „***err***" gesetzt.

11.3.4 Register_To_Bits (WBIT)



Reg_To_Bits- (oder auch ***Word_To_Bits***)-Bausteine kopieren alle 16 Bit des parametrierten Quell-Registers in die ihre 16 digitalen Ausgangsvariablen. Über ein Kontexmenü (mit rechter Maus-Taste auf Pin-Namen klicken) kann der Name der Ausgangsvariable bestimmt werden, oder es kann eine Referenz zu einer bereits bestehenden Variable erzeugt werden (in diesem Fall wird keine eigene Variable erzeugt).

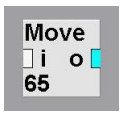
11.3.5 Bits_To_Register (BITW)



Bits_To_Reg- (oder auch ***Bits_To_Word***)-Bausteine kopieren alle ihre 16 digitalen Eingangswerte in die 16 Bit des parametrierten Ziel-Registers. Über ein Kontexmenü (mit rechter Maus-Taste auf Pin-Namen klicken) kann bestimmt werden, ob der jeweilige Eingang invertiert werden soll, und ob der Eingangswert über eine Variablenreferenz(Variablenname) ermittelt werden soll.

Wenn nicht alle 16 Eingänge angeschlossen sind, dann muss an die unbenutzten Eingänge die virtuelle Variable „_nil" angeschlossen werden, um Fehlermeldungen beim Übersetzen zu unterdrücken.

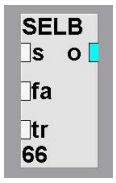
11.3.6 Move digital (Move)



Move(digital) Bausteine kopieren ihre Eingangswert direkt in ihre (immer vorhandenen) Ausgangsvariable.

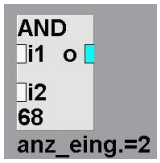
Diese Bausteine können z.B. zur Signalvervielfachung eingesetzt werden, wenn ein Ausgang mit mehr als 32 Eingängen verbunden werden soll. Diese Bausteine können auch dann eingesetzt werden, wenn man Sicherstellen will, das bestimmte Signale in Variable gespeichert werden.

11.3.7 Digitaler Umschalter (SELB)



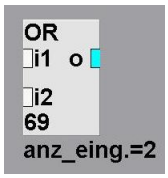
Dieser Baustein wählt aus zwei digitalen Eingängen (**tr**ue-Eingang und **fa**lse-Eingang) in Abhängigkeit von einem digitalen **s**elect-Eingang, den Ausgangswert. Wenn ***s***=true ist, dann wird der ***tr***(ue)-Eingang durchgeschaltet, sonst wird der ***fa***(lse)-Eingang zum Ausgang(***o***) durchgeschaltet.

11.3.8 UND-Baustein (AND)



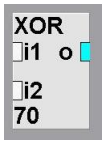
Der ***AND***-Baustein berechnet die UND-Verknüpfung aller seiner Eingänge(Anzahl ist parametrierbar, max=32) und speichert das Ergebnis im logischen Arbeitsregister. Wenn eine Ausgangsvariable vorhanden ist, dann wird sie auch entsprechend gesetzt. Über ein Kontexmenü (mit rechter Maus-Taste auf Eingangs-Pin-Namen klicken) kann bestimmt werden, ob der jeweilige Eingang invertiert werden soll, und ob der Eingangswert über eine Variablenreferenz(Variablenname) geholt werden soll.

11.3.9 ODER-Baustein (OR)



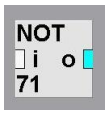
Der ***OR***-Baustein berechnet die ODER-Verknüpfung aller seiner Eingänge(Anzahl ist parametrierbar, max=32) und speichert das Ergebnis im logischen Arbeitsregister. Wenn eine Ausgangsvariable vorhanden ist, dann wird sie auch entsprechend gesetzt. Über ein Kontexmenü (mit rechter Maus-Taste auf Eingangs-Pin-Namen klicken) kann bestimmt werden, ob der jeweilige Eingang invertiert werden soll, und ob der Eingangswert über eine Variablenreferenz(Variablenname) geholt werden soll.

11.3.10 Exklusiv-ODER Baustein (XOR)



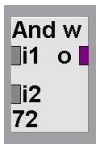
Der ***XOR***-Baustein berechnet die Exklusiv-ODER-Verknüpfung(ungleich-Verknüpfung) seiner beiden Eingänge und speichert das Ergebnis im logischen Arbeitsregister. Wenn eine Ausgangsvariable vorhanden ist, dann wird sie auch entsprechend gesetzt.

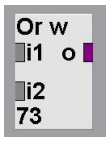
11.3.11 Nicht-Baustein (NOT)

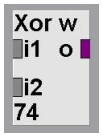


Der ***NOT***-Baustein invertiert seinen Eingangswert und speichert das Ergebnis im logischen Arbeitsregister. Wenn eine Ausgangsvariable vorhanden ist, dann wird sie auch entsprechend gesetzt.

11.3.12 Wortweise logische Verknüpfungen (AND_W, OR_W, XOR_W)

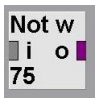






Diese Bausteine führen die jeweilige logische Verknüpfung (and, or, xor) ***bitweise*** mit ihren beiden (Festpunkt/Integer)- Eingangsvariablen durch. Das Ergebnis wird im logischen Arbeitsregister abgelegt. Wenn eine Ausgangsvariable vorhanden ist, dann wird das Ergebnis auch dort abgelegt.

11.3.13 Wortweise NOT Verknüpfung (NOT_W)



Dieser Baustein lädt seinen (Festpunkt/Integer)-Eingangswert, invertiert alle Bits, und speichert das Ergebnis im logischen Arbeitsregister. Wenn eine Ausgangsvariable vorhanden ist, dann wird das Ergebnis auch dort abgelegt.

11.3.14 Logischer Generator (LGEN)



Dieser Baustein erzeugt ein Rechtecksignal an seinem digitalen Ausgang. Die parametrierbare Zykluszeit (T/2) muss in Sekunden angegeben werden.

Beispiel: Zeit=2.0 Sekunden; alle 2 Sekunden ändert der digitale Ausgang seinen Wert (true..false).

11.3.15 Frequenzteiler (FRQD)



Bei jeder ***n***-ten positiven Signalflanke am Eingang ändert dieser Baustein seinen digitalen Ausgangswert. Der Teilerfaktor (***n***) ist über das Parameterformular einstellbar.

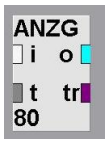
11.3.16 Konstante(k) Anzugverzögerung (ANZG)



Dieser Baustein verzögert das Durchschalten von positiven Signalflanken (false->true) am Eingang um die parametrierte Verzögerungszeit. Negative Signalflanken (true->false) werden nicht verzögert. Die Verzögerungszeit muss in Sekunden eingegeben werden.

Achtung: Wenn bei Systemstart das Eingangssignal TRUE ist, dann wird der Ausgang ***unverzögert*** auf TRUE gesetzt.

11.3.17 Variable Anzugverzögerung (ANZG)



Dieser Baustein verzögert das Durchschalten von positiven Signalflanken (false->true) am Eingang. Die Verzögerungszeit wird über den am Eingang (t) anliegenden Wert bestimmt. Dadurch kann die Verzögerungszeit zur Laufzeit verändert werden. Die Verzögerungszeit Tv ist abhängig von der wirksamen Zykluszeit(dT) und dem (Integer)Wert am Eingang (t) des Bausteins: Tv = dT * t. Negative Signalflanken (true->false) werden nicht verzögert. Am Ausgang (tr) wird immer die noch verbleibende Restzeit ausgegeben. Über ein Kontexmenü (mit rechter Maustaste auf Ausgangs-Pin-Namen oder Ausgangs-Wert klicken) kann man einstellen, daß der Wert der Restzeit in Sekunden angezeigt werden soll.

Die Verzögerungszeit am (t)-Eingang des Bausteins muss in vielfache der SPS-Zykluszeit angegeben werden. Außerdem muss dieser Wert(aus Kompatibilitätsgründen) in die *oberen* 16-bit einer 32-bit Variable eingetragen werden. Beispiel: Bei einer Zykluszeit von 20 mSec soll eine Verzögerung von 1.0 Sec eingestellt werden. t=1000mSec/20mSec=50 => Wert(hex)=$0032 0000. Die Verschiebung ins obere Wort kann mit dem Baustein *„Swapp Akku-Word"* durchgeführt werden. Für konstante Zeiten kann auch der Konstantenbaustein *„tCons"* benutzt werden. Er übernimmt alle erforderlichen Skalierungen und hat als Parameter die Verzögerungszeit in Sekunden.

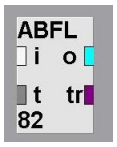
Achtung: Wenn bei Systemstart das Eingangssignal TRUE ist, dann wird der Ausgang ***unverzögert*** auf TRUE gesetzt.

11.3.18 Konstante(k) Abfallverzögerung (ABFL)



Dieser Baustein verzögert das Durchschalten von negativen Signalflanken (true->false) am Eingang um die parametrierte Verzögerungszeit. Positive Signalflanken (false->true) werden nicht verzögert. Die Verzögerungszeit muss in Sekunden eingegeben werden.

11.3.19 Variable Abfallverzögerung (ABFL)



Dieser Baustein verzögert das Durchschalten von negativen Signalflanken (true->false) am Eingang. Die Verzögerungszeit wird über den am Eingang (t) anliegenden Wert bestimmt. Dadurch kann die Verzögerungszeit zur Laufzeit verändert werden. Die Verzögerungszeit Tv ist abhängig von der wirksamen Zykluszeit(dT) und dem (Integer)Wert am Eingang (t) des Bausteins: Tv = dT * t. Positive Signalflanken (false->true) werden nicht verzögert. Am Ausgang (tr) wird immer die noch verbleibende Restzeit ausgegeben. Über ein Kontexmenü (mit rechter Maustaste auf Ausgangs-Pin-Namen oder Ausgangs-Wert klicken) kann man einstellen, daß der Wert der Restzeit in Sekunden angezeigt werden soll.

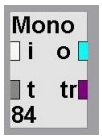
Die Verzögerungszeit am (t)-Eingang des Bausteins muss in vielfache der SPS-Zykluszeit angegeben werden. Außerdem muss dieser Wert(aus Kompatibilitätsgründen) in die *oberen* 16-bit einer 32-bit Variable eingetragen werden. Beispiel: Bei einer Zykluszeit von 20 mSec soll eine Verzögerung von 1.0 Sec eingestellt werden. t=1000mSec/20mSec=50 => Wert(hex)=$0032 0000. Die Verschiebung ins obere Wort kann mit dem Baustein „*Swapp Akku-Word*" durchgeführt werden. Für konstante Zeiten kann auch der Konstantenbaustein „*tCons*" benutzt werden. Er übernimmt alle erforderlichen Skalierungen und hat als Parameter die Verzögerungszeit in Sekunden.

11.3.20 Monoflop(Mono) mit konstanter(k) Impulsdauer



Dieser Baustein erzeugt bei positiven Eingangs-Signalflanken (false->true) am Ausgang(o) einen Rechteckimpuls(Wert=true) mit der parametrierten Impulsdauer. Dieses Monoflop ist nicht Retriggerbar (wenn die Zeit einmal läuft, dann kann sie nicht mehr zurückgesetzt werden). Die Impulsdauer wird in Sekunden eingegeben.

11.3.21 Monoflop(Mono) mit variabler Impulsdauer



Dieser Baustein erzeugt bei positiven Eingangs-Signalflanken (false->true) am Ausgang(o) einen Rechteckimpuls(Wert=true). Die Impulsdauer wird über den am Eingang (t) anliegenden Wert bestimmt. Dadurch kann die Impulsdauer zur Laufzeit verändert werden. Die Impulsdauer T ist abhängig von der wirksamen Zykluszeit(dT) und dem (Integer)Wert am Eingang (t) des Bausteins: T = dT * t. Dieses Monoflop ist nicht Retriggerbar (wenn die Zeit einmal läuft, dann kann sie nicht mehr zurückgesetzt werden). Am Ausgang (tr) wird immer die noch verbleibende Restzeit ausgegeben. Über ein Kontexmenü (mit rechter Maustaste auf Ausgangs-Pin-Namen oder Ausgangs-Wert klicken) kann man einstellen, daß der Wert der Restzeit in Sekunden angezeigt wird.

Die Impulsdauer am (t)-Eingang des Bausteins muss in vielfache der SPS-Zykluszeit angegeben werden. Außerdem muss dieser Wert(aus Kompatibilitätsgründen) in den *oberen* 16-bit einer 32-bit Variable eingetragen werden. Beispiel: Bei einer Zykluszeit von 20 mSec soll eine Impulsdauer von 1.0 Sec eingestellt werden. t=1000mSec/20mSec=50 => Wert(hex)=$0032 0000. Die Verschiebung ins obere Wort kann mit dem Baustein „*Swapp Akku-Word*" durchgeführt werden. Für konstante Zeiten/Impulsdauer kann auch der Konstantenbaustein „*tCons*" benutzt werden. Er übernimmt alle erforderlichen Skalierungen und hat als Parameter die Impulsdauer in Sekunden.

11.3.22 Raising edge Trigger (r-trigg)



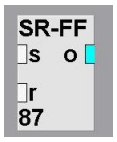
Dieser Baustein setzt bei positiven Eingangs-Signalflanken (false->true), den Wert seines Ausgangs für die Dauer einer Zykluszeit auf true.

11.3.23 Falling edge Trigger (f-trigg)



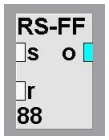
Dieser Baustein setzt bei negativen Eingangs-Signalflanken (true->false), den Wert seines Ausgangs für die Dauer einer Zykluszeit auf true.

11.3.24 Set/Reset Speicher (SR-FF)



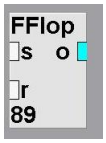
Dieser Baustein hat einen ***Setzen***- und einen ***Rücksetzen***-Eingang. Wenn nur der ***Set***-Eingang true ist, dann wird der Ausgang(auf ***true***) gesetzt. Führt nur der ***Reset***-Eingang true-Signal, dann wird der Ausgang zurückgesetzt (auf ***false***). Wenn der Wert beider Eingänge true ist, dann bleibt der Ausgangswert auf ***true***. Der Setzen-Eingang hat also Vorrang.

11.3.25 Reset/Set Speicher (RS-FF)



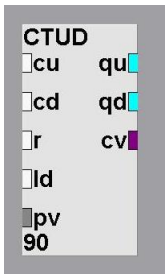
Dieser Baustein hat einen ***Setzen***- und einen ***Rücksetzen***-Eingang. Wenn nur der ***Set***-Eingang true ist, dann wird der Ausgang(auf ***true***) gesetzt. Führt nur der ***Reset***-Eingang true-Signal, dann wird der Ausgang zurückgesetzt (auf ***false***). Wenn der Wert beider Eingänge true ist, dann bleibt der Ausgangswert auf ***false***. Der Rücksetzen-Eingang hat also Vorrang.

11.3.26 Reset/Set Flip-Flop (FFlop)



Dieser Baustein hat einen ***Setzen***- und einen ***Rücksetzen***-Eingang. Wenn der Wert des ***Set***-Eingangs von false nach true wechselt (positive Flanke), dann wird der Ausgang(auf ***true***) gesetzt. Führt der ***Reset***-Eingang true-Signal, dann wird der Ausgang zurückgesetzt (auf ***false***). Wenn der Reset-Eingang true ist, und am Set-Eingang eine positive Flanke auftritt, dann bleibt der Ausgangswert auf ***false***. Der Rücksetzen-Eingang hat also Vorrang.
Dieser Baustein verhält sich wie der Reset/Set-Speicher mit dem Unterschied, das der Set-Eingang nur auf (positive)***Flanken*** reagiert und nicht auf statische „true-Werte".

11.3.27 Zähler vor/rück (CTUD)



Dieser vor/rück-Zähler Baustein hat 6 Eingänge und 3 Ausgänge:

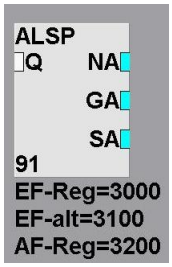
Eingänge:

- ***cu(count up)***: Bei jeder positiven Flanke an diesem Eingang wird der Zählerstand erhöht (+1).
- ***cd(count down)***: Bei jeder positiven Flanke an diesem Eingang wird der Zählerstand erniedrigt (-1).
- ***r(reset)***: Wenn dieser Eingang den Wert true hat, dann wird der Zählerstand auf 0 gesetzt.
- ***ld(load)***: Wenn dieser Eingang den Wert true hat (und ***r***eset=false ist), dann wird der Zählerstand auf den Wert des Eingangs(***pv***) gesetzt.
- ***pv(preset value)***: Dieser (Festpunkt/Integer)Eingangswert wird zum Setzen und Vergleichen des Zählerstands genutzt.

Ausgänge:

- ***qu(..up)***: Dieser (digitale)Ausgang ist true, wenn der Zählerstand ***größer*** ist als der Wert des ***pv***-Eingang.
- ***qd(..down)***: Dieser (digitale)Ausgang ist true, wenn der Zählerstand ***kleiner*** ist als der Wert des ***pv***-Eingang.
- ***cv(count value)***: Der Wert dieses (Festpunkt/Integer)Ausgangs ist der ***aktuelle Zählerstand***.

11.3.28 Alarmspeicher (ALSP)



Mit diesem Alarmspeicher können 16 Alarme (Registerbits) parallel bearbeitet werden.

Der Eingang des Bausteins ist ein Register mit 16 Alarm-informationen. Die Ausgangssignale sind: NA=neuer Alarm, GA=gegangener Alarm und SA=stehender Alarm. Über den Quitt-Eingang(Q) können die Alarme quittiert werden. Ein Ausgangsregister übernimmt nach einer Quittung das Eingangsregister:

Über das Parameterformular müssen folgende Parameter eingegeben werden:

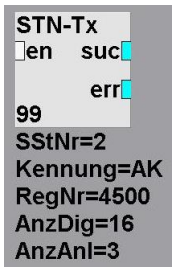
- ***EF-Register:*** Eingangsregister
- ***EF-RegAlt:*** altes Eingangsregister
- ***AF-Register:*** af-Register ????????

11.4 Telegramm-Bausteine

In diesem Kapitel werden Bausteine beschrieben welche zum Senden und Empfangen von Nachrichten über Serielle-Verbindungen benötigt werden. Zur Zeit werden RS-485/RS-422- und CAN-basierende Protokolle unterstützt. Auf dem ZM432 wird auch ModBus/UDP unterstützt.

Bausteine, die zum Aufbereiten von Telegrammdaten benötigt werden (z.B. Register-Kopier-Bausteine), werden im nächsten Kapitel beschrieben.

11.4.1 STN-Telegramm auf RS-485/422 senden (STN-Tx)



Dieser Baustein sendet immer dann, wenn der Freigabe-Eingang(***en***able) true ist, ein STN-Telegramm entsprechend den aktuellen Sende-Parametern. Nach jeder erfolgreichen Sendung wird der Erfolgs-Ausgang(***suc***cess) auf true gesetzt. Nach Sendefehlern wird der Fehlerausgang(***err***or) auf true gesetzt.

Über das Parameterformular müssen folgende Sendeparameter eingestellt werden:

- ***SStNr(Schnittstellen-Nr.)***: Die physikalische Schnittstellennummer (ZielSystem-abhängig) muss hier eingegeben werden.
- ***Kennung:*** Die Telegramm-Kennung (2 Zeichen) z.B. 'ST' muss hier eingegeben werden. Alternativ kann die Kennung auch Hexadezimal, mit führendem $-Zeichen eingegeben werden (z.B. $07FF).
- ***RegNr(Register-Nr.):*** Die Nr. des Registers, in dem die Kennung des Telegramms steht. Die Nutzdaten stehen in den darauf folgenden Registern.
- ***AnzDig(Anzahl dig. Werte):*** Anzahl digitaler Werte (Bits) die mit dem Telegramm übertragen werden sollen. Die Anzahl der Register die zum speichern der digitalen Werte nötig sind, wird durch Aufrunden von ***AnzDig*** auf vielfache von 16 ermittelt.
  AnzDigRegister=(AnzDigBits+15) DIV 16
- ***AnzAnl(Anzahl analoge Werte):*** Anzahl analoger (16-Bit)Werte (=Anzahl Register) die gesendet werden sollen.

Die *Telegrammkennung* und die *Nutzdatenanzahl* wird automatisch in die ersten beiden Register (RegNr+0 und RegNr+1) kopiert.

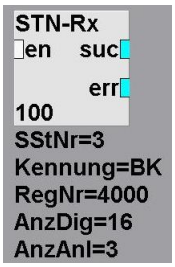
Die Nutzdaten dieses Telegramms müssen vor dem Senden in den Registerbereich geschrieben werden, der mit ***„RegNr.+2"*** beginnt. In den ersten Registern (entsprechend ***AnzDig***-Parameter) stehen die Bits der digitalen Werte. In den folgenden (***AnzAnl***) Registern stehen die zu sendenden Analogwerte.

Die Menge der Nutzdaten(Digitale + Analoge) ist auf 200 Byte(=100 Register) begrenzt.

Beispiel Registerbelegung für STN-Telegramm (12 digitale + 18 analoge Werte):

Register-Nr.	Register-Inhalt:
RegNr + 0	Telegramm-Kennung(z.B. 'AB')
RegNr + 1	AnzDig(hi-Byte)=12, AnzAnl(lo-Byte)=18
RegNr + 2	Digitale Nutzdaten (obere 12 bit)
RegNr + 3	1. Analogwert
RegNr + 20	18. (letzter) Analogwert

11.4.2 STN-Telegramme empfangen (STN-Rx)



Dieser Baustein erwartet immer dann, wenn der Freigabe-Eingang(***en***able) true ist, ein STN-Telegramm entsprechend den aktuellen Empfangs-Parametern. Nach jedem erfolgreichen Empfang wird der Erfolgs-Ausgang(***suc***cess) auf true gesetzt. Nach Empfangsfehlern wird der Fehlerausgang(***err***or) auf true gesetzt.

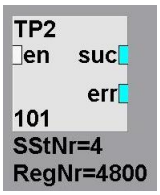
Über das Parameterformular müssen folgende Empfangsparameter eingestellt werden:

- ***SStNr(Schnittstellen-Nr.)***: Die physikalische Schnittstellennummer (ZielSystem-abhängig) muss hier eingegeben werden.
- ***Kennung:*** Die erwartete Telegramm-Kennung (2 Zeichen) z.B. 'ST' muss hier eingegeben werden. Alternativ kann die Kennung auch Hexadezimal, mit führendem $-Zeichen eingegeben werden (z.B. $07FF).
- ***RegNr(Register-Nr.):*** Die Nr. des 1. Registers, in das die Nutzdaten des Telegramms geschrieben werden sollen.
- ***RegNr(Register-Nr.):*** Die Nr. des Registers, in das die Kennung des Telegramms eingetragen werden soll. Die Nutzdaten werden in die darauf folgenden Register eingetragen.
- ***AnzDig(Anzahl dig. Werte):*** Anzahl digitaler Werte (Bit's) die das Telegramm enthalten soll. Die Anzahl der Register die zum Empfang der digitalen Werte nötig sind, wird durch Aufrunden von ***AnzDig*** auf vielfache von 16 berechnet. AnzDigRegister=(AnzDigBits+15) DIV 16
- ***AnzAnl(Anzahl analoge Werte):*** Anzahl analoger (16-Bit)Werte (=Anzahl Register) die das Telegramm enthalten soll.

Die Nutzdaten dieses Telegramms werden nach dem Empfang in einen Registerbereich geschrieben, der mit der Nr. des Parameters ***RegNr.*** beginnt. In die ersten Register (entsprechend ***AnzDig***) kommen die Bits der digitalen Werte. In die folgenden (***AnzAnl***) Register kommen die empfangenen Analogwerte.

Die Menge der Nutzdaten(Digitale + Analoge) ist auf 200 Byte(=100 Register) begrenzt.

11.4.3 TP2-Slave (TP2)



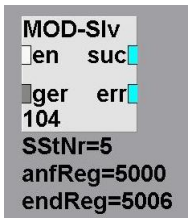
Dieser Baustein bearbeitet alle Kommandos/Anfragen eines „Gallileo"-Bedientableaus (über RS-485 Schnittstelle), wenn der Freigabe-Eingang(***en***able) =true ist. Nach jedem erfolgreichen Daten-Empfang wird der Erfolgs-Ausgang(***suc***cess) auf true gesetzt. Nach Empfangsfehlern wird der Fehlerausgang(***err***or) auf true gesetzt.

Über das Parameterformular müssen folgende Empfangsparameter eingestellt werden:

- ***SStNr(Schnittstellen-Nr.)***: Die physikalische Schnittstellennummer (ZielSystem-abhängig) muss hier eingegeben werden
- ***RegNr(Register-Nr.):*** Die Nr. des 1. Registers, in dem die TP2-Daten stehen.

11.4.4 RS-485 MODBUS Bausteine

11.4.4.1 RS-485 MODBUS-Slave (MOD slv)



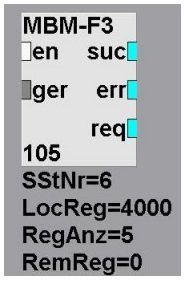
Dieser Baustein bearbeitet alle Register-Kommandos/Anfragen(MBM F3 F6 F16) eines MODBUS-Masters (über RS-485 Schnittstelle), wenn der Freigabe-Eingang(***en***able) =true ist. Nach jedem erfolgreichen Daten-Empfang wird der Erfolgs-Ausgang(***suc***cess) auf true gesetzt. Nach Empfangsfehlern wird der Fehlerausgang(***err***or) auf true gesetzt.

Jede SPS darf je Schnittstelle(RS-485 Bus) nur einen ***Modbus-Slave*** Baustein enthalten.

Über das Parameterformular müssen folgende Parameter eingestellt werden:

- ***SStNr(Schnittstellen-Nr.)***: Die physikalische Schnittstellen-nummer (ZielSystem-Abhängig) muss hier eingegeben werden
- ***G.Adr(Geräte-Adresse):*** Die Geräte-Nr.(Adresse) dieses Modbus-Slave's.
- ***Anf-Reg(Anfangs-Register-Nr.):*** Die Nr. des ersten Registers(Adr.=0), auf das MODBUS-Master zugreifen dürfen.
- ***End-Reg(End-Register-Nr.):*** Die Nr. des letzten Registers +1, auf das MODBUS-Master zugreifen dürfen.

11.4.4.2 RS-485 MODBUS-Master: Read-Multiple-Register (MBM F3)



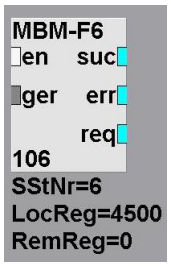
Wenn der Freigabe-Eingang(***en***able)=true ist, dann sendet dieser Modbus-Master-Baustein eine lese-Anfrage (read-request) an das adressierte Modbus-Gerät (der Wert im (high-word) des Eingangs ***„ger"*** ist die Geräte-Adresse auf dem RS-485 Bus) und schreibt die Antwort-Daten in die parametrierten Ziel-Register.

Solange ein Read-Request-Dialog(FC3) aktiv ist, solange ist auch der (***req***uest) Ausgang „true" gesetzt. Nach einem erfolgreichen Dialog wird der (***suc***cess) Ausgang „true" gesetzt. Nach einem Fehler wird der (***err***or) Ausgang „true" gesetzt.

Über das Parameterformular müssen folgende Parameter eingestellt werden:

- ***SStNr(Schnittstellen-Nr.)***: Die physikalische Schnittstellennummer (ZielSystem-Abhängig) muss hier eingegeben werden.
- ***LocReg(Locale Register-Nr.):*** Die Register-Nummer, ab der die angeforderten Registerinhalte abgelegt werden sollen.
- ***RegAnz(Register-Anzahl):*** Die Anzahl der angeforderten Register(max. 125).
- ***RemReg(Remote-Register-Nr.):*** Die Register-Nummer (im Slave), ab der die Register ausgelesen werden sollen.

11.4.4.3 RS-485 MODBUS-Master: Write-Register (MBM F6)



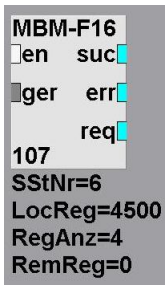
Wenn der Freigabe-Eingang(***en***able)=true ist, dann sendet dieser Modbus-Master-Baustein ein Register (Write-request) an das adressierte Modbus-Gerät. Der Wert des Eingangs ***„ger"*** (high-word) stellt die Ziel-Geräte-Adresse auf dem RS-485 Bus dar.

Solange ein Write-Request-Dialog(FC6) aktiv ist, solange ist auch der (***req***uest) Ausgang „true" gesetzt. Nach einem erfolgreichen Dialog wird der (***suc***cess) Ausgang „true" gesetzt. Nach einem Fehler wird der (***err***or) Ausgang „true" gesetzt.

Über das Parameterformular müssen folgende Parameter eingestellt werden:

- ***SStNr(Schnittstellen-Nr.)***: Die physikalische Schnittstellennummer (ZielSystem-Abhängig) muss hier eingegeben werden.
- ***LocReg(Locale Register-Nr.):*** Die Nummer des Registers, das übertragen werden sollen.
- ***RemReg(Remote-Register-Nr.):*** Die Register-Nummer (im Slave), an der das übertragenen Register eingeschrieben werden sollen.

11.4.4.4 RS-485 MODBUS-Master: Write-Multiple-Register (MBM F16)



Wenn der Freigabe-Eingang(***en***able)=true ist, dann sendet dieser Modbus-Master-Baustein die parametrierte Anzahl von Register (Write-request) an das adressierte Modbus-Gerät. Der Wert des Eingangs ***„ger"*** (high-word) stellt die Ziel-Geräte-Adresse auf dem RS-485 Bus dar.

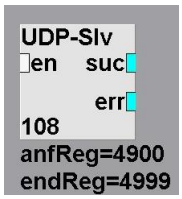
Solange ein Write-Request-Dialog(FC16) aktiv ist, solange ist auch der (***req***uest) Ausgang „true" gesetzt. Nach einem erfolgreichen Dialog wird der (***suc***cess) Ausgang „true" gesetzt. Nach einem Fehler wird der (***err***or) Ausgang „true" gesetzt.

Über das Parameterformular müssen folgende Parameter eingestellt werden:

- ***SStNr(Schnittstellen-Nr.)***: Die physikalische Schnittstellennummer (ZielSystem-Abhängig) muss hier eingegeben werden.
- ***LocReg(Locale Register-Nr.):*** Die Register-Nummer, ab der die Register übertragen werden sollen.
- ***RegAnz(Register-Anzahl):*** Die Anzahl der zu übertragenden Register.
- ***RemReg(Remote-Register-Nr.):*** Die Register-Nummer (im Slave), ab der die übertragenen Register eingeschrieben werden sollen.

11.4.5 UDP MODBUS Bausteine

11.4.5.1 UDP MODBUS-Slave (MOD slv)



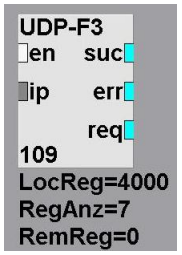
Dieser Baustein bearbeitet Modbus-Aufträge (F3 F6 F16) eines MODBUS-Masters (über UDP/IP), wenn der Freigabe-Eingang(***en***able) =true ist. Nach jedem erfolgreich bearbeiteten Auftrag(ModBus Request) wird der Erfolgs-Ausgang(***suc***cess) auf true gesetzt. Bei Fehlern wird der Fehlerausgang(***err***or) auf true gesetzt.

Jede SPS darf nur **einen *UDP-Modbus-Slave*** Baustein enthalten. In jedem SPS-Zyklus wird vom Slave max. ein ModBus-Request bearbeitet. Abweichend vom ModBus-Standard (125 bzw. 123 Register) können hier bis zu 512 Register je Telegramm übertragen werden.

Über das Parameterformular müssen folgende Parameter eingestellt werden:

- ***Anf-Reg(Anfangs-Register-Nr.):*** Die Nr. des ersten Registers (Request-Adr. = 0), auf das MODBUS-Master zugreifen dürfen.
- ***End-Reg(End-Register-Nr.):*** Die Nr. des letzten Registers, auf das MODBUS-Master zugreifen dürfen.

11.4.5.2 UDP MODBUS-Master: Read-Multiple-Register (MBM F3)



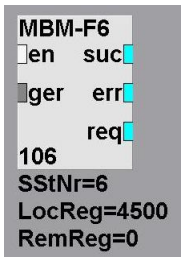
Wenn der Freigabe-Eingang(***en***able)=true ist, dann sendet dieser Modbus-Master-Baustein eine lese-Anfrage (read-request) an die Ziel-SPS (der Wert am ***„ip"*** Eingang ist die IP-Adresse der Ziel-SPS) und schreibt die Antwort-Daten in die parametrierten Ziel-Register.

Solange ein Read-Request-Dialog(FC3) aktiv ist, solange ist auch der (***req***uest) Ausgang „true" gesetzt. Nach einem erfolgreichen Dialog wird der (***suc***cess) Ausgang „true" gesetzt. Nach einem Fehler wird der (***err***or) Ausgang „true" gesetzt.

Über das Parameterformular müssen folgende Parameter eingestellt werden:

- ***LocReg(Locale Register-Nr.):*** Die Register-Nummer, ab der die angeforderten Registerinhalte abgelegt werden sollen.
- ***RegAnz(Register-Anzahl):*** Die Anzahl der angeforderten Register (max. 125).
- ***RemReg(Remote-Register-Nr.):*** Die Register-Nummer (der Ziel-SPS), ab der die Register ausgelesen werden sollen.

11.4.5.3 UDP MODBUS-Master: Write-Register (MBM F6)



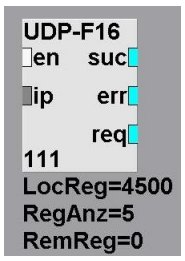
Wenn der Freigabe-Eingang(***en***able)=true ist, dann sendet dieser Modbus-Master-Baustein ein Register (Write-request) an die Ziel-SPS. Der Wert am ***„ip"*** Eingang ist die IP-Adresse der Ziel-SPS.

Solange ein Write-Request-Dialog(FC6) aktiv ist, solange ist auch der (***req***uest) Ausgang „true" gesetzt. Nach einem erfolgreichen Dialog wird der (***suc***cess) Ausgang „true" gesetzt. Nach einem Fehler wird der (***err***or) Ausgang „true" gesetzt.

Über das Parameterformular müssen folgende Parameter eingestellt werden:

- ***LocReg(Locale Register-Nr.):*** Die Nummer des Registers, das übertragen werden sollen.
- ***RemReg(Remote-Register-Nr.):*** Die Register-Nummer (der Ziel-SPS), an der der übergebenen Registerwert eingetragen werden soll.

11.4.5.4 UDP MODBUS-Master: Write-Multiple-Register (MBM F16)



Wenn der Freigabe-Eingang(***en***able)=true ist, dann sendet dieser Modbus-Master-Baustein die parametrierte Anzahl von Register (Write-request) an die Ziel-SPS. Der Wert am ***„ip"*** Eingang ist die IP-Adresse der Ziel-SPS.

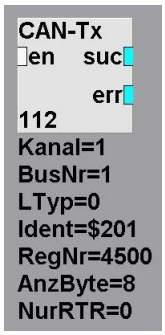
Solange ein Write-Request-Dialog(FC16) aktiv ist, solange ist auch der (***req***uest) Ausgang „true" gesetzt. Nach einem erfolgreichen Dialog wird der (***suc***cess) Ausgang „true" gesetzt. Nach einem Fehler wird der (***err***or) Ausgang „true" gesetzt.

Über das Parameterformular müssen folgende Parameter eingestellt werden:

- ***LocReg(Locale Register-Nr.):*** Die Register-Nummer, ab der die Register übertragen werden sollen.
- ***RegAnz(Register-Anzahl):*** Die Anzahl der zu übertragenden Register (max. 123).
- ***RemReg(Remote-Register-Nr.):*** Die Register-Nummer (der Ziel-SPS), ab der die gesendeten Registerwerte eingetragen werden sollen.

11.4.6 CAN Bausteine

11.4.6.1 CAN Sende (CAN-Tx)

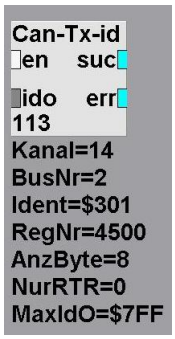


Dieser Baustein sendet CAN Telegramme, wenn der Wert seines ***en***able-Eingangs ***true*** ist. Nach jeder erfolgreichen Sendung wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Sendefehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. (z.B. wenn der Sende-Fehlerzähler des CAN-Controlers eine bestimmte Grenze überschreitet)

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 14 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von einem Sendebaustein benutzt werden. Kanal 14 ist ein „gepufferter" Sendekanal und kann gleichzeitig von mehreren Bausteinen benutzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controler, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Auf den internen CAN-Bus(0) kann dieser Baustein nicht zugreifen.
- ***LTyp(LängenTyp):*** Dieser Parameter bestimmt, ob ein CAN-Telegramm mit kurzem(11 Bit) oder mit langem(29 Bit) Identifier gesendet werten soll. Wenn „0" eingetragen wird, dann wird ein kurzer Identifier verwendet. Trägt man „1" ein, dann wird ein langer Identifier verwendet.
- ***Ident(Identifier):*** Hier muss der gewünschte Identifier (11 oder 29 Bit) eingegeben werden.
- ***RegNr(Register-Nr.):*** Der Inhalte des Registers(hi-Byte) mit dieser Nr. ist das erste Byte der Sende-Nutzdaten.
- ***AnzByte(Anzahl Byte):*** Dieser Wert (0 bis 8) bestimmt, wie-viele Nutzdaten(Byte) gesendet werden sollen.
- ***NurRTR(Nur RTR=Remote Transmitt Request):*** Wenn dieser Parameter den Wert „1" bekommt, dann wird nicht aktiv gesendet, sondern dieses Telegramm wird nur dann gesendet, wenn für diesen Kanal ein ***RTR-Telegramm*** eintrifft. Wenn dieser Parameter eine „0" enthält, dann wird dieses Telegramm bei Aktivierung(en=true) auch sofort gesendet.

11.4.6.2 CAN Sende mit Identifier-Offset (CAN-Tx-Id)



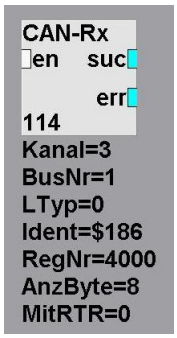
Dieser Baustein sendet CAN Telegramme, wenn der Wert seines ***en***able-Eingangs ***true*** ist. Nach jeder erfolgreichen Sendung wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Sendefehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. (z.B. wenn der Sende-Fehlerzähler des CAN-Controlers eine bestimmte Grenze überschreitet).

Der wirksame Identifier kann zur Laufzeit über den ***ido***-Eingang (Identifier-Offset) modifiziert werden. (Id = BasisIdentifier + IdentifierOffset)

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 14 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von einem Sendebaustein benutzt werden. Kanal 14 ist ein „gepufferter" Sendekanal und kann gleichzeitig von mehreren Bausteinen benutzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controler, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Auf den internen CAN-Bus(0) kann dieser Baustein nicht zugreifen.
- ***LTyp(LängenTyp):*** Dieser Parameter bestimmt, ob ein CAN-Telegramm mit kurzem(11 Bit) oder mit langem(29 Bit) Identifier gesendet werten soll. Wenn „0" eingetragen wird, dann wird ein kurzer Identifier verwendet. Trägt man „1" ein, dann wird ein langer Identifier verwendet.
- ***Ident(Identifier):*** Hier muss der gewünschte Basis-Identifier (11 oder 29 Bit) eingegeben werden.
- ***RegNr(Register-Nr.):*** Der Inhalte des Registers(hi-Byte) mit dieser Nr. ist das erste Byte der Sende-Nutzdaten.
- ***AnzByte(Anzahl Byte):*** Dieser Wert (0 bis 8) bestimmt, wie-viele Nutzdaten(Byte) gesendet werden sollen.
- ***NurRTR(Nur RTR=Remote Transmitt Request):*** Wenn dieser Parameter den Wert „1" bekommt, dann wird nicht aktiv gesendet, sondern dieses Telegramm wird nur dann gesendet, wenn für diesen Kanal ein ***RTR-Telegramm*** eintrifft. Wenn dieser Parameter eine „0" enthält, dann wird dieses Telegramm bei Aktivierung(en=true) auch sofort gesendet.
- ***MaxIdO(Maximaler Identifier-Offset):*** Hier muss der maximal erlaubtet Identifier-Offset eingetragen werden.

11.4.6.3 CAN Empfang (CAN-Rx)

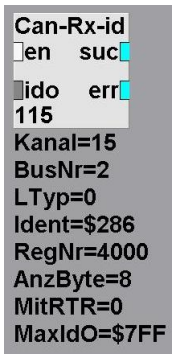


Dieser Baustein erwartet CAN Telegramme, wenn der Wert seines ***en***able-Eingangs ***true*** ist. Nach jedem erfolgreichen Empfang wird der Wert des ***success***-Ausgangs auf true gesetzt. Nach Empfangsfehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. Empfangsfehler liegen vor, wenn die Anmeldedaten und die Empfangsdaten nicht übereinstimmen (z.B. die Länge der Nutzdaten). Der Fehlerausgang wird auch dann gesetzt, wenn der Empfangs-Fehlerzähler des CAN-Controlers eine bestimmte Grenze überschreitet.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von einem CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen genutzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controler, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern: 1, 2 und 4. Auf den internen CAN-Bus(0) kann dieser Baustein nicht zugreifen.
- ***LTyp(LängenTyp):*** Dieser Parameter bestimmt, ob ein CAN-Telegramm mit kurzem(11 Bit) oder mit langem(29 Bit) Identifier empfangen werten soll. Eine „0" eintragen bedeutet: 11-Bit Identifier verwenden. Eine „1" eintragen bedeutet: 29-Bit Identifier verwenden.
- ***Ident(Identifier):*** Hier muss der gewünschte Identifier (11 oder 29 Bit) eingegeben werden.
- ***RegNr(Register-Nr.):*** Die Empfangsdaten werden byteweise in das Register(RegNr.) und in die folgenden Register eingetragen.
- ***AnzByte(Anzahl Byte):*** Dieser Wert (0 bis 8) bestimmt, wie viele Nutzdaten(Byte) empfangen werden sollen.
- ***MitRTR(Mit RTR=Remote Transmitt Request):*** Wenn dieser Parameter den Wert „1" bekommt, dann fordert dieser Sendekanal mittels eines ***RTR-Telegramms*** die Empfangsdaten aktiv an. Wenn dieser Parameter eine „0" enthält, dann wartet dieser Kanal nur auf das parametrierte CAN-Telegramm, ohne es aktiv anzufordern. Kanal 15 kann nicht mit ***RTR*** arbeiten.

11.4.6.4 CAN Empfange mit Identifier-Offset (CAN-Rx-Id)



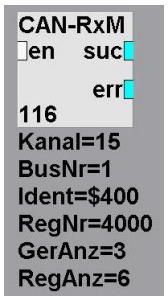
Dieser Baustein erwartet CAN Telegramme, wenn der Wert seines ***en***able-Eingangs ***true*** ist. Nach jedem erfolgreichen Empfang wird der Wert des ***success***-Ausgangs auf true gesetzt. Nach Empfangsfehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. Empfangsfehler liegen vor, wenn die Anmeldedaten und die Empfangsdaten nicht übereinstimmen (z.B. die Länge der Nutzdaten). Der Fehlerausgang wird auch dann gesetzt, wenn der Empfangs-Fehlerzähler des CAN-Controlers eine bestimmte Grenze überschreitet.

Der wirksame Identifier kann zur Laufzeit über den ***ido***-Eingang (Identifier-Offset) modifiziert werden. (Id = BasisIdentifier + IdentifierOffset)

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von einem CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen genutzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controler, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern: 1, 2 und 4. Auf den internen CAN-Bus(0) kann dieser Baustein nicht zugreifen.
- ***LTyp(LängenTyp):*** Dieser Parameter bestimmt, ob ein CAN-Telegramm mit kurzem(11 Bit) oder mit langem(29 Bit) Identifier empfangen werten soll. Eine „0" eintragen bedeutet: 11-Bit Identifier verwenden. Eine „1" eintragen bedeutet: 29-Bit Identifier verwenden.
- ***Ident(Identifier):*** Hier muss der gewünschte Identifier (11 oder 29 Bit) eingegeben werden.
- ***RegNr(Register-Nr.):*** Die Empfangsdaten werden byteweise in das Register(RegNr.) und in die folgenden Register eingetragen.
- ***AnzByte(Anzahl Byte):*** Dieser Wert (0 bis 8) bestimmt, wie viele Nutzdaten(Byte) empfangen werden sollen.
- ***MitRTR(Mit RTR=Remote Transmitt Request):*** Wenn dieser Parameter den Wert „1" bekommt, dann fordert dieser Sendekanal mittels eines ***RTR-Telegramms*** die Empfangsdaten aktiv an. Wenn dieser Parameter eine „0" enthält, dann wartet dieser Kanal nur auf das parametrierte CAN-Telegramm, ohne es aktiv anzufordern. Kanal 15 kann nicht mit ***RTR*** arbeiten.
- ***MaxIdO(Maximaler Identifier-Offset):*** Hier muss der maximal erlaubte Identifier-Offset eingetragen werden.

11.4.6.5 CAN Empfange multiple (CAN-RxM)



Dieser Baustein kann Telegramme von mehreren Geräten am angegebenen Bus empfangen. Die unteren 6-Bit des verwendeten (11-Bit) Identifiers repräsentieren die Geräteadresse(1 bis 63) des Absenders. Wenn zwei dieser Bausteine auf unterschiedlichen Bussen „horchen", sonst aber gleiche Parameter haben, dann kann damit ein ***„Redundantes CAN-Bus-System"*** realisiert werden.

Je System können maximal 16 ***CAN-RxM*** Bausteine verwendet werden. Wenn mehrere dieser Bausteine (auf unterschiedlichen Kanälen und mit unterschiedlichen Basis-Identifiern) angemeldet werden, dann können auf diese Weise Gerätegruppen angesprochen werden.

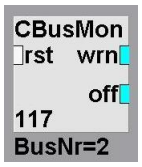
Für jeden angemeldeten CAN-Bus-Teilnehmer (anz.Geräte) werden entsprechend dem Parameter „Register/Gerät" entsprechend viele Register benötigt. Die Gesamtmenge der benötigten Register(ab RegNr.) ist: ***AnzahlRegister = AnzahlGeräte * (Register_je_Gerät + 1).*** Jeder dieser Registerbereiche beginnt mit einem Statusregister.

Dieser Baustein erwartet CAN Telegramme, wenn der Wert seines ***en***able-Eingangs ***true*** ist. Nach jedem erfolgreichen Empfang eines der möglichen Telegramme wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Empfangsfehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von einem CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen benutzt werden. Die Kanäle 1 bis 13 verbrauchen weniger Rechenzeit, und sollten deshalb bevorzugt eingesetzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controler, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Der interne CAN-Bus(0) und kann hier nicht benutzt werden.
- ***Ident(Identifier):*** Hier muss der gewünschte Basis-Identifier (11 Bit) eingegeben werden. Die unteren 6-Bit(Geräte-Absender-Adresse) sollten dabei leer (=0) gelassen werden ($040 bis $7C0).
- ***RegNr(Register-Nr.):*** Ab diesem Register werden die empfangenen Daten(Register) eingeschrieben.
  Der beanspruchte Registerbereich geht bis: **RegNr+GerAnz*(RegAnz+1) -1**
- ***GerAnz(Geräte Anzahl):*** Dieser Parameter bestimmt, welche Absender-Adressen akzeptiert werden und wie viele Empfangsregister reserviert werden müssen.
- ***RegAnz(Anzahl Register/Gerät):*** Dieser Wert gibt an, wie viele Register-Nutzdaten ein Gerät sendet. Jedes CAN-RxM Telegramm überträgt 8 Byte. Byte 1 und 2 enthalten den Ziel-Offset. Die Bytes 3 bis 8 enthalten die Daten der 3 übertragenen Register. Der Parameter ***RegAnz*** muss deshalb ein vielfaches von 3 sein.

11.4.6.6 CAN Bus-Monitor (CBusMon)



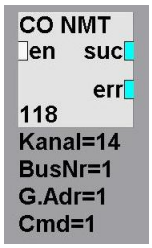
Dieser Baustein wertet Status-informationen der CAN-Controler aus und zeigt aktuelle Fehlerzustände an zwei digitalen Bausteinausgängen an.

- ***wrn(Warnung):*** Dieser Ausgang signalisiert, das der interne Fehlerzähler des CAN-Controlers (für den parametrierten Bus) einen bestimmten Wert überschritten hat, der Controler aber immer noch aktiv ist (senden ist möglich).
- ***off(Abgeschaltet):*** Dieser Ausgang signalisiert, das der interne Fehlerzähler (für den parametrierten Bus) eine obere Grenze überschritten hat, und deshalb der CAN-Bus Sender abgeschaltet wurde. (Der Sender geht in einen hochohmigen Zustand)

Über das Parameterformular muss die Nummer des zu überwachenden CAN-Busses eingegeben werden. Wenn der Bausteineigang ***rst***(reset) auf true gesetzt wird, dann werden die internen Fehlerzustände des CAN-Controlers zurückgesetzt, und senden ist wieder möglich. Der ***rst***-Eingang kann auch permanent auf true gesetzt werden. In diesem Fall werden auftretende Fehler sofort zurückgesetzt.

11.4.7 CAN-OPEN Bausteine

11.4.7.1 Send-NMT(CO NMT)



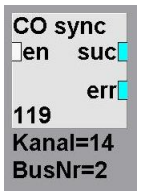
Dieser CAN-OPEN Baustein sendet parametrierte NMT(Network ManagemenT) Kommandos, wenn sein ***en***able-Eingang den Wert true hat.

Folgende NMT Kommandos können parametriert bzw. gesendet werden:

- ***Start_Remote_Node (Code=1):*** Dieses Kommando versetzt einen CAN-Open Slave aus dem „Pre-Operational_State" (Zustand nach dem einschalten eines Gerätes) in den „Operational_State" (aktiver/normaler Betriebszustand).
- ***Stopp_Remote_Node (Code=2):*** Dieses Kommando versetzt einen CAN-Open Slave in den „Prepared_State" (PDO/SDO/Error- Kommunikation ist ausgeschaltet).
- ***Enter_Pre-Operational_State (Code=$80):*** Dieses Kommando versetzt einen CAN-Open Slave in den „Pre-Operational_State"
  (In diesem Zustand sind nur Initialisierungs-Aktionen/Telegramme erlaubt).
- ***Reset_Node (Code=$81):*** Dieses Kommando löst bei dem angesprochenen Salve einen Rest aus.
- ***Reset_Communication (Code=$82):*** Dieses Kommando löst bei dem angesprochenen Salve einen Rest aus.

Über die parametrierte Geräteadresse (Node-Id) 1..127 wird ein einzelnes Gerät gewählt. Über die Geräteadresse 0 werden alle CAN-Open Geräte an einem Bus selektiert.

11.4.7.2 Send-SYNC (CO sync)

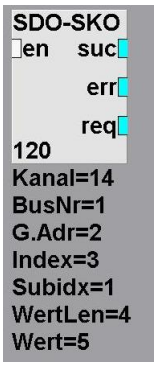


Dieser CAN-OPEN Baustein sendet ein SYNC-Telegramm auf dem parametrierten Bus, wenn sein ***en***able-Eingang den Wert ***true*** hat. SYNC-Telegramme dienen der Synchronisation von „Prozess-Aufgaben" z.B. dem zyklischen anfordern von Messwerten. Nach jeder erfolgreichen Sendung eines SYNC-Telegramms wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Sendefehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von einem CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen benutzt werden. Die Kanäle 1 bis 13 verbrauchen weniger Rechenzeit, und sollten deshalb bevorzugt eingesetzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controller, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Der interne CAN-Bus(0) und kann hier nicht benutzt werden.

11.4.7.3 Schreib Konstante ins Objektverzeichnis (SDO SKO)

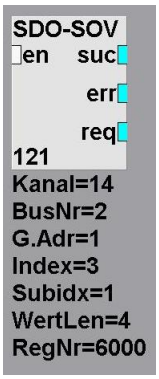


Mit SDO-Telegrammen können Werte in das ObjektVerzeichnis ***anderer*** CAN-OPEN Geräte eintragen werden. Dieser CAN-OPEN Baustein sendet einen parametrierbaren konstanten Wert über ein SDO-Telegramm an das adressierte CAN-OPEN Gerät, wenn sein ***en***able-Eingang den Wert ***true*** hat. Nach jedem erfolgreichen SDO-Dialog wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Sende- oder Empfangs-fehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. Solange ein SDO-Dialog aktiv ist (..bis eine Antwort eingetroffen ist), solange führt der ***req***uest-Ausgang den Wert true.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von **einem** CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen benutzt werden. Die Kanäle 1 bis 13 verbrauchen weniger Rechenzeit, und sollten deshalb bevorzugt eingesetzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controller, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Der interne CAN-Bus(0) und kann hier nicht benutzt werden.
- ***GeräteAdresse***: Hier wird die Adresse des Ziel-Gerätes eingetragen.
- ***Index***: Hier wird der Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***SubIndex***: Hier wird der Sub-Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***WertLen***: Hier wird die Länge der zu übertragenden Konstanten in Byte angegeben (1, 2, oder 4).
- ***Wert***: Hier wird der Wert der zu übertragenden Konstante eingetragen.

11.4.7.4 Schreib ins Objektverzeichnis (SDO SOV)

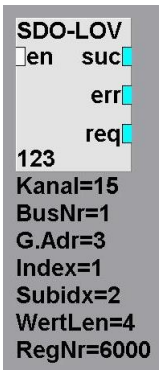


Mit SDO-Telegrammen können Werte in das ObjektVerzeichnis ***anderer*** CAN-OPEN Geräte eintragen werden. Dieser CAN-OPEN Baustein sendet Registerinhalte über ein SDO-Telegramm an das adressierte CAN-OPEN Gerät, wenn sein ***en***able-Eingang ***true*** ist. Nach jedem erfolgreichen SDO-Dialog wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Sende- oder Empfangs-fehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. Solange ein SDO-Dialog aktiv ist (..bis eine Antwort eingetroffen ist), solange führt der ***req***uest-Ausgang den Wert true.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von **einem** CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen benutzt werden. Die Kanäle 1 bis 13 verbrauchen weniger Rechenzeit, und sollten deshalb bevorzugt eingesetzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controller, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Der interne CAN-Bus(0) und kann hier nicht benutzt werden.
- ***GeräteAdresse***: Hier wird die CAN-OPEN-Adresse des Ziel-Gerätes eingetragen.
- ***Index***: Hier wird der Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***SubIndex***: Hier wird der Sub-Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***WertLen***: Hier wird die Länge der zu übertragenden Daten(Registerinhalte) in Byte angegeben (1, 2 oder 4).
- ***RegNr***: Hier wird die Nr. des 1. Registers eingetragen, ab dem die Registerinhalte übertragen werden sollen.

11.4.7.5 Lies aus dem Objektverzeichnis (SDO LOV)

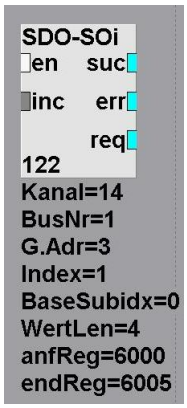


Mit SDO-Telegrammen kann man Werte aus dem ObjektVerzeichnis ***anderer*** CAN-OPEN Geräte auslesen. Dieser CAN-OPEN Baustein liest aus dem ObjektVerzeichnis des adressierte CAN-OPEN Gerätes und trägt den gelesenen Wert in das Zielregister und ggf. in folgende Register ein, wenn sein ***en***able-Eingang ***true*** ist. Nach jedem erfolgreichen SDO-Dialog wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Sende- oder Empfangs-fehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. Solange ein SDO-Dialog aktiv ist (..bis eine Antwort eingetroffen ist), solange führt der ***req***uest-Ausgang den Wert true.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von **einem** CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen benutzt werden. Die Kanäle 1 bis 13 verbrauchen weniger Rechenzeit, und sollten deshalb bevorzugt eingesetzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controller, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Der interne CAN-Bus(0) und kann hier nicht benutzt werden.
- ***Geräte-Adresse***: Hier wird die CAN-OPEN-Adresse des Ziel-Gerätes eingetragen.
- ***Index***: Hier wird der Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***SubIndex***: Hier wird der Sub-Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***WertLen***: Hier wird die Länge der auszulesenden Daten in Byte angegeben (1, 2 oder 4).
- ***RegNr***: Hier wird die Nr. des 1. Registers eingetragen, ab dem die empfangenen Daten eingeschrieben werden sollen.

11.4.7.6 Schreib indiziert ins Objektverzeichnis (SDO SOi)



Mit SDO-Telegrammen kann man Werte in das ObjektVerzeichnis ***anderer*** CAN-OPEN Geräte eintragen. Dieser CAN-OPEN Baustein sendet Inhalte von indiziert adressierten Registern über ein SDO-Telegramm an das adressierte CAN-OPEN Gerät, wenn sein ***en***able-Eingang ***true*** ist. Nach jedem erfolgreichen SDO-Dialog wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Sende- oder Empfangs-fehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. Solange ein SDO-Dialog aktiv ist (..bis eine Antwort eingetroffen ist), solange führt der ***req***uest-Ausgang den Wert true.

Der Wert des ***inc***-Eingangs wird als Offset für den Zielgeräte-Subindex und als Offset für die Start-Register-Nr. (anfReg) der zu sendenden Daten interpretiert. Der Subindex setzt sich aus dem parametrierten ***BaseSubIndex*** und aus dem Wert des ***inc***-Eingangs zusammen: *Subindex = BaseSubIndex + Wert(**inc**-Eingang)*

Die Quell-Register-Nr. errechnet sich folgendermaßen: *RegNr = anfReg + 2*Wert(**inc**-Eingang)*

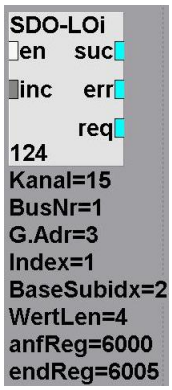
Das bedeutet, dass über den ***inc***-Eingang nur Register bzw. Daten adressiert werden können die 4 Byte auseinander liegen, auch wenn jeweils nur ein Byte übertragen werden soll.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von **einem** CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen benutzt werden. Die Kanäle 1 bis 13 verbrauchen weniger Rechenzeit, und sollten deshalb bevorzugt eingesetzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controller, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Der interne CAN-Bus(0) und kann hier nicht benutzt werden.
- ***GeräteAdresse***: Hier wird die CAN-OPEN-Adresse des Ziel-Gerätes eingetragen.
- ***Index***: Hier wird der Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***BaseSubIndex***: Hier wird der BasisSub-Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***WertLen***: Hier wird die Länge der zu übertragenden Daten in Byte angegeben (1,2 oder 4).
- ***AnfReg***: Hier wird die Start-Register-Nr. eingetragen, ab der Registerinhalte übertragen werden dürfen.
- ***EndReg***: Hier wird die End-Register-Nr. eingetragen, bis zu der Registerinhalte übertragen werden dürfen.

Wenn zur Laufzeit eine Quell-Register-Nr. größer als ***endReg*** ist, dann wird der Fehlerausgang auf true gesetzt.

11.4.7.7 Lies indiziert aus dem Objektverzeichnis (SDO LOi)



Mit SDO-Telegrammen kann man Werte aus dem ObjektVerzeichnis ***anderer*** CAN-OPEN Geräte lesen. Dieser CAN-OPEN Baustein liest indiziert(Subindex-Offset) Daten aus dem Objektverzeichnis des adressierten CAN-OPEN Gerätes, wenn sein ***en***able-Eingang ***true*** ist. Nach jedem erfolgreichen SDO-Dialog wird der Wert des ***success***-Ausgangs für einen Zyklus auf true gesetzt. Nach Sende- oder Empfangs-fehlern wird der Wert des ***err***or-Ausgangs auf true gesetzt. Solange ein SDO-Dialog aktiv ist (..bis eine Antwort eingetroffen ist), solange führt der ***req***uest-Ausgang den Wert true.

Der Wert des ***inc***-Eingangs wird als Offset für den Subindex des CAN-OPEN Gerätes und als Offset für die Ziel-Register-Nr. (anfReg) der zu übertragenden Daten interpretiert. Der Subindex setzt sich aus dem parametrierten ***BaseSubIndex*** und aus dem Wert des ***inc***-Eingangs zusammen: *Subindex = BaseSubIndex + Wert(**inc**-Eingang)*

Die Ziel-Register-Nr. errechnet sich folgendermaßen: *RegNr = anfReg + 2*Wert(**inc**-Eingang)*

Das bedeutet, dass über den ***inc***-Eingang nur Register bzw. Daten adressiert werden können die 4 Byte auseinander liegen, auch wenn jeweils nur ein Byte übertragen werden soll.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

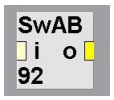
- ***Kanal-Nr.:*** Es werden 15 Kanäle unterstützt. Jeder Kanal ist entweder ein Sende- oder ein Empfangs-kanal. Die Kanäle 1 bis 13 dürfen jeweils immer nur von **einem** CAN-Baustein benutzt werden. Kanal 14 ist ein „gepufferter" *Sendekanal* und kann nicht zum Empfangen benutzt werden. Kanal 15 ist ein „gepufferter" *Empfangskanal* und kann gleichzeitig von mehreren Bausteinen benutzt werden. Die Kanäle 1 bis 13 verbrauchen weniger Rechenzeit, und sollten deshalb bevorzugt eingesetzt werden.
- ***Bus-Nr.:*** Das ZM432 hat 2 CAN-Controller, deshalb kann hier die Bus-Nr.1 oder die Bus-Nr.2 eingegeben werden. Die ZKG500 hat 3 externe CAN-Busse mit den Nummern:1, 2 und 4. Der interne CAN-Bus(0) und kann hier nicht benutzt werden.
- ***GeräteAdresse***: Hier wird die CAN-OPEN-Adresse des Ziel-Gerätes eingetragen.
- ***Index***: Hier wird der Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***BaseSubIndex***: Hier wird der BasisSub-Index des Objektverzeichnises des adressierten Gerätes eingetragen.
- ***WertLen***: Hier wird die Länge der zu übertragenden Daten in Byte angegeben (1,2 oder 4).
- ***AnfReg***: Hier wird die Start-Register-Nr. eingetragen, ab der Registerinhalte eingetragen werden dürfen.
- ***AndReg***: Hier wird die End-Register-Nr. eingetragen, bis zu der Registerinhalte eingetragen werden dürfen.

Wenn zur Laufzeit eine Ziel-Register-Nr. größer als ***endReg*** ist, dann wird der Fehlerausgang auf true gesetzt.

11.5 Daten-Verschiebe Bausteine

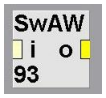
Mit den folgenden Bausteinen ist es möglich Daten (Register und SPS-Variable) zu verschieben bzw. zu manipulieren. Dieses wird hauptsächlich zum Aufbereiten von Telegramm-Sende und -Empfangsdaten benötigt.

11.5.1 Swapp Accu-Byte (SwAB)



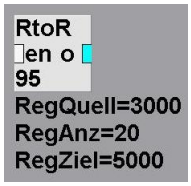
Dieser Baustein liest seinen Festpunkt-Eingangswert, tauscht die oberen 2 Byte des Wertes und lösch die niederwertigen 2 Byte. Das Ergebnis dieser Operation gibt der Baustein an seinem Ausgang wieder aus. Diese Funktion wird speziell für die Aufbereitung von TP2-Telegrammen benötigt.

11.5.2 Swapp Accu-Word (SwAW)



Dieser Baustein liest seinen Eingangswert(32-Bit Festpunkt), tauscht das obere Wort mit dem unteren Wort, und gibt das Ergebnis an seinem Ausgang wieder aus.

11.5.3 Kopiere Register nach Register (RtoR)



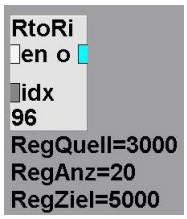
Mit Hilfe dieses Bausteins können die Inhalte ganzer Registerbereiche (Quell-Register nach Ziel-Register) kopiert werden.

Dieser Baustein kopiert ***RegAnz*** Register von ***RegQuelle*** nach ***RegZiel,*** wenn sein ***en***able Eingang ***true*** ist.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***RegQuelle: . . .*** Hier muss die Quell-Reg.Nr. des ersten zu kopierenden Registers eingetragen werden.
- ***RegAnz: . . .*** Hier muss die Anzahl der zu kopierenden Register eingetragen werden.
- ***RegZiel: . . .*** Hier muss die erste Reg.Nr. des Reg.Zielbereiches eingetragen werden.

11.5.4 Kopiere Register nach Register indiziert (RtoRi)



Mit Hilfe dieses Bausteins können die Inhalte ganzer Registerbereiche (Quell-Register nach Ziel-Register + idx) kopiert werden.

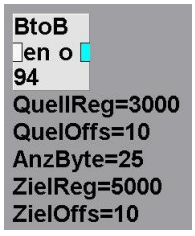
Dieser Baustein kopiert ***RegAnz*** Register von ***RegQuelle*** nach ***RegZiel+idx,*** wenn sein ***en***able Eingang ***true*** ist.

Der Wert des ***idx***-Eingang wird als Offset für das Register-Ziel benutzt.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***RegQuelle: . . .*** Hier muss die Quell-Reg.Nr. des ersten zu kopierenden Registers eingetragen werden.
- ***RegAnz: . . .*** Hier muss die Anzahl der zu kopierenden Register eingetragen werden.
- ***RegZiel: . . .*** Hier muss die erste Reg.Nr. des Reg.Zielbereiches eingetragen werden. (..Offset 'idx' berücksichtigen)

11.5.5 Kopiere Register byteweise mit Offset (BtoB)



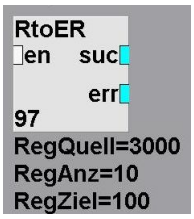
Mit Hilfe dieses Bausteins können Registerbereiche byteweise mit Quell- und Ziel-Offset kopiert werden. Die Offsetangaben sind Byte-Offsets, so dass als Quelle bzw. als Ziel auch das niederwertige Byte eines Registers adressiert werden kann.

Dieser Baustein kopiert ***AnzByte*** Byte von ***QuellReg+QuellByteOffset*** nach ***ZielReg+ZielByteOffset,*** wenn sein ***en***able Eingang ***true*** ist.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***QuellReg: . . .*** Hier muss die Quell-Reg.Nr. des ersten zu kopierenden Registers eingetragen werden.
- ***QuellOffs: . . .*** Hier muss der Quell-Reg. Byte-Offset (0..255) eingetragen werden.
- ***AnzByte: . . .*** Hier muss die Anzahl der zu kopierenden Bytes (0..200) eingetragen werden.
- ***ZielReg: . . .*** Hier muss die erste Reg.Nr. des Reg.Zielbereiches eingetragen werden.
- ***ZielOffs: . . .*** Hier muss der Ziel-Reg. Byte-Offset (0..255) eingetragen werden.

11.5.6 Kopiere Register nach EEPROM-Register (RtoER)



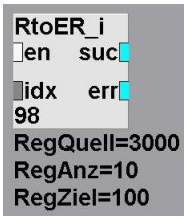
Mit Hilfe dieses Bausteins können die Inhalte ganzer Registerbereiche in EEPROM-Registerbereiche kopiert werden. Da das EEPROM nur eine begrenzte Anzahl von Schreibzyklen besitzt (10 000 .. 100 000) soll zwischen zwei Kopieroperationen eine Wartezeit von mindestens einer Stunde eingehalten werden.

Dieser Baustein kopiert ***RegAnz*** Register von ***RegQuelle*** nach ***RegZiel,*** wenn sein ***en***able Eingang ***true*** ist. Nach einem erfolgreichen Kopiervorgang wird der ***suc***cess-Ausgang für eine Zykluszeit auf true gesetzt. Nach einem erfolglosen Kopiervorgang (z.B. weil das EEPROM defekt ist) wird der ***err***or-Ausgang auf true gesetzt.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***RegQuelle: . . .*** Hier muss die Quell-Reg.Nr. des ersten zu kopierenden Registers eingetragen werden.
- ***RegAnz: . . .*** Hier muss die Anzahl (0..99) der zu kopierenden Register eingetragen werden.
- ***RegZiel: . . .*** Hier muss die erste Reg.Nr. des EEPROM-Reg.Zielbereiches (ZM432: 0..2047 bzw. ZKG500: 0..199) eingetragen werden.

11.5.7 Kopiere Register nach EEPROM-Register indiziert (RtoER_i)



Mit Hilfe dieses Bausteins können die Inhalte ganzer Registerbereiche in EEPROM-Registerbereiche (mit Ziel-Offset) kopiert werden. Da das EEPROM nur eine begrenzte Anzahl von Schreibzyklen besitzt (10 000 .. 100 000) soll zwischen zwei Kopieroperationen eine Wartezeit von mindestens einer Stunde eingehalten werden.

Dieser Baustein kopiert ***RegAnz*** Register von ***RegQuelle*** nach ***RegZiel+idx,*** wenn sein ***en***able Eingang ***true*** ist. Nach einem erfolgreichen Kopiervorgang wird der ***suc***cess-Ausgang für eine Zykluszeit auf true gesetzt. Nach einem erfolglosen Kopiervorgang (z.B. weil das EEPROM defekt ist) wird der ***err***or-Ausgang auf true gesetzt.

Der Wert des ***idx***-Eingang wird als Offset für das Register-Ziel benutzt.

Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***RegQuelle: . . .*** Hier muss die Quell-Reg.Nr. des ersten zu kopierenden Registers eingetragen werden.
- ***RegAnz: . . .*** Hier muss die Anzahl (0..99) der zu kopierenden Register eingetragen werden.
- ***RegZiel: . . .*** Hier muss die erste Reg.Nr. des EEPROM-Reg.Zielbereiches (ZM432: 0..2047 bzw. ZKG500: 0..199) eingetragen werden. (..Offset 'idx' berücksichtigen)

11.6 Anzeige und Testhilfe-Bausteine

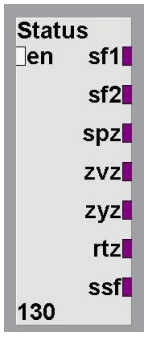
11.6.1 Uhrzeit (Uhr)

Wenn der ***err***-Ausgang ***false*** ist, dann gibt dieser Baustein an seinen Ausgängen folgende Zeiten aus:

- ***yea:*** . . . Das aktuelle Jahr z.B. 05
- ***mon:*** . . . Der aktuelle Monat: 1..12
- ***day:*** . . . Der aktuelle Monatstag: 1..31
- ***hou:*** . . . Die aktuelle Stunde: 0..23
- ***min:*** . . . Die aktuelle Minute: 0..59
- ***sec:*** . . . Die aktuelle Sekunde: 0..59
- ***wdy:*** . . . Der aktuellen Wochentag: 1..7 (Sonntag = 1)

Der Uhr-Baustein liefert erst dann gültige Zeitwerte(err=false), nachdem er ein gültiges ***Uhr-Stellen*** Telegramm erhalten hat. Zur Zeit kann dieses Telegramm nur durch Anklicken des Uhr-Bausteins mit der rechten Maustaste in der Betriebsart Online erzeugt werden.

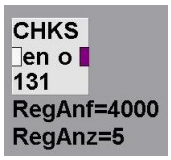
11.6.2 Systemstatus Baustein (Status)



Wenn sein ***en***able Eingang ***true*** ist, dann gibt dieser Baustein an seinen Ausgängen folgende Systeminformationen aus:

- ***SF1: . . .*** Slot-Fehler 1 (FP-Variable): SF1=0, wenn alle angeschlossenen IO-Karten antworten.
- ***SF2: . . .*** Slot-Fehler 2 (FP-Variable): SF2=0, wenn alle angeschlossenen IO-Karten antworten.
- ***SPZ: . . .*** Zeitverbrauch (Integerwert) des kompletten SPS-Programms in Mikrosekunden.
- ***ZVZ: . . .*** Zeitverbrauch des aktuellen SPS-Zyklusses in Mikrosekunden.
- ***ZYZ: . . .*** Die Zykluszeit in Millisekunden.
- ***RTZ: . . .*** Die Restzeit in Mikrosekunden (RTZ=Zeitscheibe SPZ)
- ***SSF: . . .*** Schnittstellen-Fehler (Festpunktvariable): SSF=0, wenn CAN1, CAN2, CAN4, RS485 fehlerfrei sind, und die SLE500 mit allen Telegrammen antwortet.

11.6.3 Register-CheckSumme (CHKS)



Wenn sein ***en***able Eingang ***true*** ist, dann berechnet dieser Baustein über einen parametrierbaren Registerbereich eine Check-Summe und gibt diese Check-Summe an seinem Ausgang als Festpunktwert(Integer) aus.

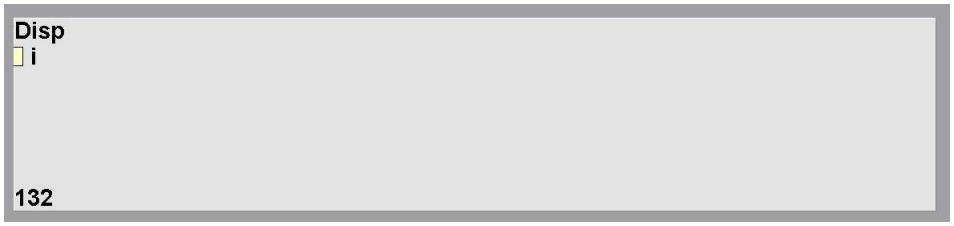
Über das Parameterformular müssen folgende Parameter eingegeben werden:

- ***RegAnf: . . .*** Hier muss die Reg.Nr. des ersten Registers des Reg.-Bereichs eingetragen werden.
- ***RegAnz: . . .*** Hier muss die Anzahl der Register des Reg.-Bereichs eingetragen werden.

11.6.4 Kommentartext

Kommentartext kann überall in die SPS-Grafik eingefügt werden. Er ist in erster Linie dazu bestimmt das SPS-Programm zu kommentieren bzw. verständlicher zu gestalten. Über ein Kontexmenü (..mit rechter Maustaste auf Text drücken) kann Inhalt und Größe der Kommentartexte verändert werden.

11.6.5 Analog-Schreiber (Disp)



Mit diesen Bausteinen kann der zeitliche Verlauf von analogen Festpunktwerten dargestellt werden. Der an den Eingang „i” angeschlossene Festpunktwert wird in den Betriebsarten Simulation und Online als Zeit-verlauf dargestellt. Der Y-Bereich des Schreibers beträgt +/- 1.0 wie bei Festpunktwerten üblich. Die Zeitspreizung (Sekunden/Teilung bzw. Sec/Div) kann über das Menü: „Anzeigen.Projektinformationen. Display(Sec/Div)” oder über das Menü: „Betriebsart.Display-Zeitbasis” geändert werden.

11.6.6 Numerisches Analog-Display (AnaDisp)



Mit diesen Bausteinen kann der aktuelle Wert von analogen Festpunktwerten als Festkommazahl dargestellt werden. Der an den Eingang „i" angeschlossene Festpunktwert wird mit einem parametrierbaren Skalierfaktor multipliziert und entsprechend der parametrierten Anzahl von Nachkommastellen angezeigt.

Über das Parameterformular können folgende Parameter eingegeben werden:

- ***Name: ...*** Hier kann ein Name für den angezeigten Wert eingegeben werden. z.B.: „Ist-Drehzahl"
- ***DimText: ...*** Hier kann ein Dimensions-Text für den angezeigten Werte eingegeben werden. z.B.: „U/Min"
- ***SkalFaktor: ...*** Hier muss ein Skalierfaktor für den angezeigten Wert eingegeben werden. z.B.: „1000"
- ***NachKomma: ...*** Hier muss die Anzahl der Nachkommastellen für den angezeigten Wert eingegeben werden. z.B.: „2"

11.6.7 Numerisches Integer-Display (IntDisp)

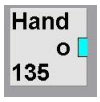


Mit diesen Bausteinen kann der aktuelle Wert von analogen Bausteinen als ganze (integer)Zahl dargestellt werden. Der an den Eingang „i" angeschlossene Wert wird mit einem parametrierbaren Skalierfaktor multipliziert und entsprechend der parametrierten Anzahl von Stellen (ggf. mit führenden Nullen) angezeigt.

Über das Parameterformular können folgende Parameter eingegeben werden:

- ***Name: ...*** Hier kann ein Name für den angezeigten Wert eingegeben werden. z.B.: „Betriebsstunden"
- ***DimText: ...*** Hier kann ein Dimensions-Text für den angezeigten Werte eingegeben werden. z.B.: „Stunden"
- ***SkalFaktor: ...*** Hier muss ein Skalierfaktor für den angezeigten Wert eingegeben werden. z.B.: „1"
- ***StellenAnz: ...*** Hier muss die Anzahl der Stellen für den angezeigten Wert eingegeben werden. z.B.: „4"

11.6.8 Boolescher Hand-Schalter (Hand)



In den Betriebsarten ***Simulation*** und ***Online*** kann der digitale Ausgangswert dieses „**Hand-Schalters**" durch drücken mit der linken Maus-Taste auf den Baustein, umgeschaltet werden.

Über das Parameterformular kann ein Name für den Baustein eingegeben werden (..dient nur der Kommentierung).

11.6.9 LCD Display: Text-Ausgabe

Mit Hilfe dieser Bausteine können (nur für die Zielsysteme SRK01/02/03 und MCC601/602) Texte auf einem LCD-Display (4x12 Zeichen bzw 1x8 Zeichen) ausgegeben werden. Wenn die Enable-Eingänge der Bausteine den Wert TRUE haben, dann werden ihre parametrierten Texte an das Tableau ausgegeben. Die Ausgangswerte 'o' dieser Bausteine übernehmen immer den Wert der Enable-Eingänge. Diese Ausgangswerte können zur Festlegung der Reihenfolge von LCD-Ausgaben benutzt werden. Dazu müssen nur die Ausgänge mit den Enable-Eingängen der nächsten LCD-Bausteine verbunden werden.

Parameter sind:

- ***Zeile: ...*** Gewünschte Zeilennummer des Ausgabetextes (1..4, bzw. 1)
- ***Spalte: ...*** Gewünschte Spalten-Nr. des Ausgabetextes (1..12, bzw. 1..8)
- ***Text: ...*** Hier wird der gewünschte Ausgabetext eingegeben. Soll der Text am Anfang oder am Ende Leerzeichen(Blanks) enthalten, dann muss der Text mit Hochkommas eingeschlossen werden z.B.: ' Ausgabe'.

Die Ausgabereihenfolge aller LCD-Ausgaben richtet sich nach der Baustein-Nr. Soll ein Text nur einmal und nicht in jedem SPS-Zyklus wieder ausgegeben werden, dann muss das über den Enable-Eingang gesteuert werden (z.B. mit dem Trigger-Baustein ..R-Trig).

Die LCD-Verwaltung sorgt für eine „gerechte" LCD-Ausgabe. Das bedeutet das auch ein Baustein mit konstant gesetztem Enable-Eingang nicht die Ausgabe eines Textes mit größerer Baustein-Nr. und nur einmalig gesetztem Enable-Eingang verhindern kann.

11.6.10 LCD Display: Wert-Ausgabe

Mit Hilfe dieser Bausteine können (für die Zielsysteme SRK01/02/03 und MCC601/602) Werte von SPS-Variablen auf dem LCD-Display ausgegeben werden. Die obersten 16-bit der SPS-Variablen werden als Dezimalzahl ausgegeben. Die Position des Dezimalpunktes kann frei gewählt werden. Die Werte der Ausgabevariablen müssen so Skaliert(mit einem Faktor multipliziert) werden, dass die sich ergebenden Dezimalzahlen dem gewünschten Anzeigebereich entsprechen.

Parameter sind:

- ***Zeile: ...*** Gewünschte Zeilennummer des Ausgabewertes (1..4, bzw. 1)
- ***Spalte: ...*** Gewünschte StartSpalten-Nr. des Ausgabewertes (1..12, bzw. 1..8)
- ***PunkPos: ...*** Hier muss die Position des Dezimalpunktes(=Anzahl der Nachkommastellen) angegeben werden (Pos. 0 ist rechts von der Zahl, das bedeutet in diesem Fall: „kein Punkt")
- ***FeldLen: ...*** Hier muss die Anzahl der Stellen des Ausgabefeldes angegeben werden. Beispiel: -327.68 (PunktPos=2, FeldLen=7). Das Ausgabefeld wird immer von rechts aufgefüllt. Ist die angegebene Feldlänge zu klein, dann werden Zeichen auf der linken Seite weggelassen.

Beispiel:

Es soll eine Spannung im Bereich ± 999,9 V mit einer Nachkommastelle dargestellte werden. Zur Darstellung werden 6 Stellen für Vorzeichen, Zahlenwert und Dezimalpunkt benötigt.

Die Spannungsnormierung sei -999.999V bis +999.999V => -1.0 bis +1.0

Der Multiplikator x ergibt sich somit zu: x = 9999 / 32768 = 0,30515 , PunktPos = 1

11.6.11 Programm-fluss Steuerung (PFS)



Über Anschlüsse dieses Bausteins kann die Abarbeitungsreihenfolge von Bausteinen, die sich sonst nicht über Verbindungslinien verknüpfen lassen, festgelegt werden.

Bausteine dieses Typs erzeugen weder Programmcode noch Variable.

11.6.12 Variablen-Bausteine (VAR)





Diese Bausteine enthalten immer Ausgangsvariablen. Der Wert dieser Variablen wird von den Bausteinen nicht verändert. Über den „virtuellen" Eingang kann aber die Abarbeitungsreihenfolge (wichtig für Folgebausteine) festgelegt werden.

Von diesem Bausteintyp gibt es eine Variante mit Festpunkt- bzw. Integer- und eine mit Boolescher-variable.

Abbildung 11.2: Beispiel für VAR- und PFS-Bausteine:

