

Dateiname: Example_BCD.pro

Verzeichnis: D:\Andreas_Schiff\ICSGmbH\ICS\Internet\Internet-PDFs\Deutsch

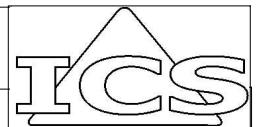
Geändert am: 19.11.05 19:28:24 / V2.2

Bezeichnung: Example_BCD

Autor: Andreas Schiff, ICS GmbH

Version: 1.2 vom 14.10.2004

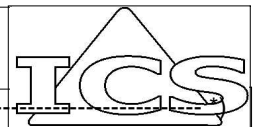
Beschreibung: Das Programm steuert das AS-Interface Anzeigemodul AM004 an und gibt im Sekundentakt eine um 1 erhöhte Zahl aus.



```

0001 PROGRAM PLC_PRG
0002 (* ----- *)
0003 (*           Anzeige-Text-Beispielprogramm           *)
0004 (* ----- *)
0005 (* ICS GmbH, Hopfenstraße 1, 88069 Tettnang, Autor: Andreas Schiff           Version 1.2 *)
0006 (* ----- *)
0007 (* Dieses Beispielprogramm dient nur zur Demonstration der Funktion des AM004. Es kann keine Gewähr dafür *)
0008 (* übernommen werden, dass dieses Programm in einem realen Automatisierungsprojekt fehlerfrei läuft. *)
0009 (* ----- *)
0010
0011 VAR
0012     AnzeigeB: SerComBCD; (* Funktionsbaustein für BCD-Wandlung und Ausgabe *)
0013     Taktgeber3: TON;
0014     Takt2: BOOL;
0015     HB2: BOOL;
0016     Taktgeber4: TON;
0017     Takt1: BOOL;
0018     Zaehler: INT; (* Zahl, die angezeigt wird *)
0019     Plus: BYTE:=0; (* Pluszeichen soll ausgegeben werden *)
0020     Text: ARRAY [0..7] OF BYTE:=16#26, 16#26, 16#26, 16#26, 16#6D, 16#62, 16#61, 16#72; (* Textformatierung
0021     für unterlagerten Text: Die Zeichen &&&& werden von der Zahl
0022     überschrieben *)
0023     Diagnose: BYTE;
0024     DI2 AT %IX1.5.2: BOOL; (* Im Beispiel hat das Anzeigemodul die Adresse 5 *)
0025     DI3 AT %IX1.5.3: BOOL;
0026     DO0 AT %QX1.5.0: BOOL;
0027     DO1 AT %QX1.5.1: BOOL;
0028 END_VAR
0029
0030 (* Ende des Deklarationsteils *)
-----
0001
0002 (* ----- Hauptprogramm ----- *)
0003 (* In diesem Programm wird alle 1s die Variable "Zähler" um 1 erhöht. Die Variable wird anschließend auf
0004 dem einzeiligen Textdisplay ausgegeben. Daraus kann die Zeit, die für die Aktualisierung der Anzeige benötigt wird,
0005 abgeschätzt werden. *)
0006
0007 (* Takt: 1000ms *)
0008 Taktgeber3(IN:=Takt2,PT:=t#100ms);
0009 HB2:=NOT Taktgeber3.Q;
0010 Taktgeber4(IN:=HB2,PT:=t#900ms);
0011 Takt2:=Taktgeber4.Q;
0012
0013 AnzeigeB(DI2:=DI2, DI3:=DI3, Variable:=Zaehler, Format:=Plus,Text:=Text);
0014 Diagnose:=AnzeigeB.Diagnosis;
0015 DO0:=AnzeigeB.DO0;
0016 DO1:=AnzeigeB.DO1;
0017
0018 IF Takt1<>Takt2 AND Takt2=TRUE THEN
0019     Zaehler:=Zaehler+1;
0020     IF (Zaehler>999 OR Zaehler<-999) THEN
0021         Zaehler:=-999;
0022     END_IF;
0023 END_IF;
0024
0025 Takt1:=Takt2;
-----
SerComBCD (FB-ST)
0001 FUNCTION_BLOCK SerComBCD

```

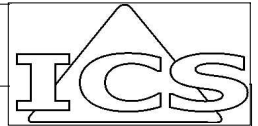


0002	(* ----- *)		
0003	(* Dieser Funktionsbaustein wickelt die gesamte serielle Kommunikation zu einem AS-Interface Slave nach dem *)		
0004	(* Profil S-B.A.5 ab. Gilt nur für einzeilige Ausgabe mit der alternativen Möglichkeit der zyklischen Ausgabe im *)		
0005	(* BCD-Code! Der Funktionsbaustein schreibt eine Zeile ASCII-Text im azyklischen Mode und dann die 4stellige *)		
0006	(* Zahl, die im BCD-Code übergeben werden muss. *)		
0007	(* *)		
0008	(* ICS GmbH, Hopfenstraße 1, 88069 Tettngang, Autor: Andreas Schiff		Version 0.4 Datum: 08.10.2004 *)
0009	(* verbesserter Abschluss eines gesendeten Telegramms *)		
0010	(* ----- *)		
0011			
0012	VAR_INPUT		
0013	Variable:	INT;	(* auszugebene Zahl nach Maßgabe der Formatierung *)
0014	Format:	BYTE;	(* Formatierung der Variablen *)
0015	(* =0: Ganze Zahl im Bereich von -999 bis 9999 *)		
0016	(* =1: wie vor, Pluszeichen wird zusätzlich ausgegeben: -999 bis +999 *)		
0017	(* =2: ausgegebene Zahl hat eine Nachkommastelle: -9,9 bis +9,9 *)		
0018	(* =3: wie vor, Pluszeichen wird zusätzlich ausgegeben: -9,9 bis +9,9 *)		
0019	(* =6: wie 2, Zahl hat eine Nachkommastelle und Punkt statt Komma *)		
0020	(* =7: wie 3, Zahl hat eine Nachkommastelle und Punkt statt Komma *)		
0021	Text:	ARRAY [0..7] OF BYTE:=16#7E,16#20,16#26,16#20,16#20,16#20,16#20,16#7F;	
0022	DI2:	BOOL;	(* Eingang serielles Protokoll *)
0023	DI3:	BOOL;	(* Eingang serielles Protokoll *)
0024	END_VAR		
0025			
0026	VAR_OUTPUT		
0027	Diagnosis:	BYTE;	(* Diagnose-Daten: Bit 7: Slave nicht im seriellen Modus Bit 6: ID nicht korrekt (Soll: Vendor-ID: 1, Product-ID: 5) Bit 5: --- Bit 4: Slave busy Bit 0..3: Diagnoseinfo vom Slave *)
0028			
0029			
0030			
0031			
0032	DO0:	BOOL;	(* Ausgang serielles Protokoll *)
0033	DO1:	BOOL;	(* Ausgang serielles Protokoll *)
0034	END_VAR		
0035			
0036	VAR		
0037	DatIn:	ARRAY [0..7] OF BYTE; (* Zyklischen Daten im BCD-Code *)	
0038	ID_Data:	ARRAY [0..15] OF BYTE; (* ID-Daten des Slaves *)	
0039	InputD:	ARRAY [0..20] OF BYTE;	
0040	InputData:	ARRAY [0..20] OF BYTE; (* Eingabe-Datenfeld *)	
0041	OutputData:	ARRAY [0..20] OF BYTE; (* Ausgabe-Datenfeld *)	
0042	OutputLen:	BYTE;	(* Länge der Ausgabedaten in Byte; Zahlenbereich 1..15 *)
0043	InputL:	BYTE;	(* Länge der Eingabedaten in Byte; Zahlenbereich 1..15 *)
0044	InputLen:	BYTE;	
0045	Taktalt:	BOOL;	
0046	In:	BYTE;	(* aktuell empfangene Nachricht *)
0047	Inalt:	BYTE;	(* letzte empfangene Nachricht *)
0048	Out:	BYTE:= 3;	
0049	Watchdog:	BOOL;	
0050	Timeout:	BYTE;	
0051	Lastin:	BYTE;	(* letztes gültiges empfangenes Informationsbit *)
0052	i:	BYTE;	
0053	IBitzeiger:	BYTE;	
0054	Outalt:	BYTE;	
0055	Outzeiger:	BYTE;	
0056	OBitzeiger:	BYTE;	
0057	x:	BYTE;	
0058	Send:	BOOL;	
0059	y:	BYTE;	
0060	NewData:	BOOL;	

```

0061 SCSlave: BOOL;
0062 SSW: WORD;
0063 Inst1:TON;
0064 Inst2:TON;
0065 Takt:BOOL;
0066 Var2: BOOL;
0067 Acyclic: WORD;
0068 Para_DatIn: ARRAY [0..20] OF BYTE;
0069 DatInAlt: ARRAY[0..20] OF BYTE;
0070 Timeout2: BYTE;
0071 MerkDat1: BOOL;
0072 MerkDat2: BOOL;
0073 Schluss: BOOL;
0074 ARS0Request: BOOL;
0075 ARSResponse_0: BOOL;
0076 ARS1Request: BOOL;
0077 ARSResponse_1: BOOL;
0078 ARS2Request: BOOL;
0079 ARSResponse_2: BOOL;
0080 AWS2Request: BOOL;
0081 AWSResponse_2: BOOL;
0082 Hunderter: INT;
0083 Einser: INT;
0084 Punkt: BOOL;
0085 Tausender: INT;
0086 Zehner: INT;
0087 Komma: BYTE;
0088 Steuerbyte:BYTE;
0089 Varalt: INT;
0090 END_VAR
0091
0092 (* -----Ende des Deklarationsteils----- *)
0001 (* Hier wird ein Takt mit einer Zykluszeit von 20ms generiert *)
0002 Inst1(IN:=Takt, PT:=t#10ms);
0003 Var2:=NOT Inst1.Q;
0004 Inst2(IN:=Var2, PT:=t#10ms);
0005 Takt:=Inst2.Q;
0006 (* ----- Hier wird die auszugebende Zahl in das richtige Format gebracht ----- *)
0007 (* Steuerbyte analysieren *)
0008 Steuerbyte:=Format;
0009 IF (Steuerbyte AND 16#2)<>0 THEN
0010     Punkt:=TRUE;
0011     IF (Steuerbyte AND 16#4)<>0 THEN
0012         Komma:=16#0E;
0013         Steuerbyte:=Steuerbyte AND 16#03;
0014     ELSE
0015         Komma:=16#0C;
0016     END_IF;
0017 ELSE
0018     Punkt:=FALSE;
0019 END_IF;
0020
0021 IF Variable<>Varalt THEN
0022     IF (Variable<10000 AND Variable>=0 AND Steuerbyte=0) OR (Variable<1000 AND Variable>=0 AND (Steuerbyte=1
0023         OR Steuerbyte=2)) OR (Variable<100 AND Variable>=0 AND Steuerbyte=3) THEN
0024         (* Ermitteln der max. 4 BCD-Stellen *)
0025         Tausender:=0;
0026         Hunderter:=Variable/100;
0027         Zehner:=0;

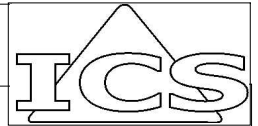
```



```

0028   Einser:=Variable-Hunderter*100;
0029   DatIn[0]:=0;
0030   DatIn[1]:=0;
0031   IF Hunderter>9 THEN
0032       Tausender:=Hunderter/10;
0033       Hunderter:=Hunderter-Tausender*10;
0034   END_IF;
0035   IF Einser>9 THEN
0036       Zehner:=Einser/10;
0037       Einser:=Einser-Zehner*10;
0038   END_IF;
0039   (* Eintragen der BCD-Stellen *)
0040   IF Punkt=TRUE THEN
0041       DatIn[0]:=SHL(INT_TO_BYTE(Hunderter),4)+INT_TO_BYTE(Zehner);
0042       DatIn[1]:=SHL(Komma,4)+INT_TO_BYTE(Einser);
0043   ELSE
0044       DatIn[0]:=SHL(INT_TO_BYTE(Tausender),4)+INT_TO_BYTE(Hunderter);
0045       DatIn[1]:=SHL(INT_TO_BYTE(Zehner),4)+INT_TO_BYTE(Einser);
0046   END_IF;
0047   (* Führende Nullen unterdrücken, ggf. Pluszeichen eintragen *)
0048   IF DatIn[0]<10 AND DatIn[0]>0 THEN
0049       DatIn[0]:=DatIn[0]+160;
0050       IF (Steuerbyte AND 16#1)=1 THEN
0051           DatIn[0]:=DatIn[0]+16;
0052       END_IF;
0053   END_IF;
0054   IF DatIn[0]=0 AND DatIn[1]>9 THEN
0055       IF Punkt=FALSE THEN
0056           DatIn[0]:=170;
0057           IF (Steuerbyte AND 16#1)=1 THEN
0058               DatIn[0]:=DatIn[0]+1;
0059           END_IF;
0060       ELSE
0061           DatIn[0]:=160;
0062           IF (Steuerbyte AND 16#1)=1 THEN
0063               DatIn[0]:=DatIn[0]+16;
0064           END_IF;
0065       END_IF;
0066
0067   END_IF;
0068   IF DatIn[0]=0 AND DatIn[1]<10 THEN
0069       DatIn[0]:=170;
0070       DatIn[1]:=DatIn[1]+160;
0071       IF (Steuerbyte AND 16#1)=1 THEN
0072           DatIn[1]:=DatIn[1]+16;
0073       END_IF;
0074   END_IF;
0075   ELSIF (Variable>-1000 AND Variable<0 AND (Steuerbyte=0 OR Steuerbyte=1)) OR (Variable>-100 AND
0076       Variable<0 AND (Steuerbyte=2 OR Steuerbyte=3)) THEN
0077       Hunderter:=-Variable/100;
0078       Zehner:=0;
0079       Einser:=-Variable-Hunderter*100;
0080       IF Einser>9 THEN
0081           Zehner:=Einser/10;
0082           Einser:=Einser-Zehner*10;
0083       END_IF;
0084       (* Eintragen der BCD-Stellen *)
0085       IF Punkt=TRUE THEN
0086           DatIn[0]:=SHL(INT_TO_BYTE(Hunderter),4)+INT_TO_BYTE(Zehner);

```



```

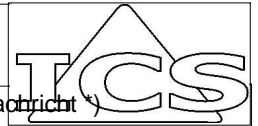
0087   DatIn[1]:=SHL(Komma,4)+INT_TO_BYTE(Einser);
0088   ELSE
0089   DatIn[0]:=INT_TO_BYTE(Hunderter);
0090   DatIn[1]:=SHL(INT_TO_BYTE(Zehner),4)+INT_TO_BYTE(Einser);
0091   END_IF;
0092   (* Minuszeichen einbauen; führende Nullen unterdrücken *)
0093   IF Punkt=FALSE THEN
0094     IF DatIn[0]<10 AND DatIn[0]>0 THEN
0095       DatIn[0]:=DatIn[0]+208;
0096     END_IF;
0097     IF DatIn[0]=0 AND DatIn[1]>9 THEN
0098       DatIn[0]:=173;
0099     END_IF;
0100     IF DatIn[0]=0 AND DatIn[1]<10 THEN
0101       DatIn[0]:=170;
0102       DatIn[1]:=DatIn[1]+208;
0103     END_IF;
0104   ELSE
0105     DatIn[0]:=DatIn[0]+208;
0106   END_IF;
0107   ELSE
0108     DatIn[0]:=255;
0109     DatIn[1]:=255;
0110   END_IF;
0111 END_IF;
0112 (* ----- Hier wird das serielle Protokoll bedient ----- *)
0113 (* Eingänge des Slaves lesen, ausmaskieren und auf Veränderung prüfen *)
0114 (* Hier wird umcodiert: In=3: Idle; In=1: Separator; In=0: DATA0; In=2: DATA1 *)
0115 IF DI2=FALSE AND DI3=FALSE THEN In:=0; END_IF;
0116 IF DI2=FALSE AND DI3=TRUE THEN In:=2; END_IF;
0117 IF DI2=TRUE AND DI3=FALSE THEN In:=1; END_IF;
0118 IF DI2=TRUE AND DI3=TRUE THEN In:=3; END_IF;
0119
0120 IF In<>Inalt THEN
0121   (* eine Antwort ist eingetroffen *)
0122   IF Out=1 AND In=1 THEN
0123     (* die eine der erwarteten Antworten ist eingetroffen: Separator *)
0124     Watchdog:=FALSE;
0125     SCSSlave:=TRUE;
0126     Timeout:=0;
0127     Lastin:=Inalt;
0128     (* Aktion 2: Trenn-Nachricht empfangen; Master sendet Information oder Idle *)
0129     IF Send=TRUE THEN
0130       IF Outalt=3 THEN
0131         (* Ausgabe initialisieren *)
0132         Outzeiger:=0;
0133         OBitzeiger:=0;
0134         y:=OutputData[0];
0135       END_IF;
0136       IF Schluss=FALSE THEN
0137         Outalt:=Out;
0138         x:=y AND 2#1000_0000;
0139         IF x<>0 THEN
0140           Out:=2;
0141         ELSE
0142           Out:=0;
0143         END_IF;
0144         y:=SHL(y,1);
0145         OBitzeiger:=OBitzeiger+1;

```

```

0146      (* ein Byte fertig? Zeiger auf nächstes Byte stellen *)
0147      IF OBitzeiger>7 THEN
0148          Outzeiger:=Outzeiger+1;
0149          y:=OutputData[Outzeiger];
0150          OBitzeiger:=0;
0151          IF Outzeiger>=OutputLen OR Outzeiger>=19 THEN
0152              Schluss:=TRUE;
0153              SCSSlave:=TRUE;
0154          END_IF;
0155      END_IF;
0156      ELSE
0157          Outalt:=Out;
0158          Out:=3;
0159          Send:=FALSE;
0160          Schluss:=FALSE;
0161      END_IF;
0162      ELSE
0163          Outalt:=Out;
0164          Out:=3;
0165      END_IF;
0166      IF Out=0 THEN DO0:=TRUE; DO1:=TRUE; END_IF;
0167      IF Out=1 THEN DO0:=FALSE; DO1:=TRUE; END_IF;
0168      IF Out=2 THEN DO0:=TRUE; DO1:=FALSE; END_IF;
0169      IF Out=3 THEN DO0:=FALSE; DO1:=FALSE; END_IF;
0170      ELSIF Out<>1 AND In<>1 THEN
0171          (* die andere der erwarteten Antworten ist eingetroffen: Idle oder Daten *)
0172          Watchdog:=FALSE;
0173          SCSSlave:=TRUE;
0174          Timeout:=0;
0175          IF (In=0 AND Inalt<>2) OR (In=2 AND Inalt<>0) THEN
0176              (* Aktion 0/1: Slave hat Informationsbit gesendet: abspeichern; Master sendet Trenn-Nachricht *)
0177              IF Lastin=3 THEN
0178                  (* Beginn einer Nachricht: Zeiger und Feld Initialisieren, wenn Feld bereit *)
0179                  NewData:=FALSE;
0180                  InputL:=0;
0181                  IBitzeiger:=0;
0182                  FOR i:=0 TO 19 DO
0183                      InputD[i]:=0;
0184                  END_FOR;
0185              END_IF;
0186              InputD[InputL]:=SHL(InputD[InputL],1);
0187              IF In=2 THEN
0188                  InputD[InputL]:=InputD[InputL]+1;
0189              END_IF;
0190              IBitzeiger:=IBitzeiger+1;
0191              IF IBitzeiger>7 THEN
0192                  InputL:=InputL+1;
0193                  IF InputL>19 THEN
0194                      InputL:=19;
0195                  END_IF;
0196                  IBitzeiger:=0;
0197              END_IF;
0198              Outalt:=Out;
0199              Out:=1;
0200              IF Out=0 THEN DO0:=TRUE; DO1:=TRUE; END_IF;
0201              IF Out=1 THEN DO0:=FALSE; DO1:=TRUE; END_IF;
0202              IF Out=2 THEN DO0:=TRUE; DO1:=FALSE; END_IF;
0203              IF Out=3 THEN DO0:=FALSE; DO1:=FALSE; END_IF;
0204          ELSE

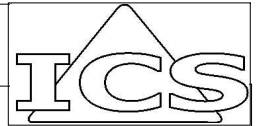
```



```

0205 (* Aktion 3: Slave hat Idle gesendet: Abschluss einer Nachricht?; Master sendet Trenn-Nachricht *)
0206 IF Lastin<>3 THEN
0207     (* Abschluss einer Nachricht; wenn diese nicht vollständig (also ganzzahlige Anzahl von Bytes),
0208     dann wird sie verworfen! *)
0209     IF IBitzeiger =0 THEN
0210         FOR i:=0 TO InputL DO
0211             InputData[i]:=InputD[i];
0212         END_FOR;
0213         NewData:=TRUE;
0214         InputLen:=InputL;
0215         SCSSlave:=TRUE;
0216     END_IF;
0217 END_IF;
0218 Outalt:=Out;
0219 Out:=1;
0220 IF Out=0 THEN DO0:=TRUE; DO1:=TRUE; END_IF;
0221 IF Out=1 THEN DO0:=FALSE; DO1:=TRUE; END_IF;
0222 IF Out=2 THEN DO0:=TRUE; DO1:=FALSE; END_IF;
0223 IF Out=3 THEN DO0:=FALSE; DO1:=FALSE; END_IF;
0224 END_IF;
0225 END_IF;
0226 ELSIF (Takt<>Taktalt AND Takt=TRUE) THEN
0227     (* Slave alle 120ms testen, ob er den seriellen Kommunikationsmodus unterstützt, dient auch zum Anlauf
0228     der Kommunikation *)
0229     Timeout:=Timeout+1;
0230     IF Timeout>4 THEN
0231         Outalt:=Out;
0232         IF (Out=0 OR Out=3) THEN
0233             Out:=1;
0234         ELSIF (Out=1 OR Out=2) THEN
0235             Out:=3;
0236         END_IF;
0237         Timeout:=0;
0238         SCSSlave:=FALSE;
0239         SSW:=0;
0240         Send:=FALSE;
0241         IF Out=0 THEN DO0:=TRUE; DO1:=TRUE; END_IF;
0242         IF Out=1 THEN DO0:=FALSE; DO1:=TRUE; END_IF;
0243         IF Out=2 THEN DO0:=TRUE; DO1:=FALSE; END_IF;
0244         IF Out=3 THEN DO0:=FALSE; DO1:=FALSE; END_IF;
0245     END_IF;
0246 END_IF;
0247
0248 (* Hier läuft die Timeoutüberwachung für azyklische Dienste ab: 1s *)
0249 IF Taktalt<>Takt AND Takt=FALSE AND Acyclic<>0 THEN
0250     Timeout2:=Timeout2+1;
0251     IF Timeout2>151 THEN
0252         Acyclic:=0;
0253         Timeout2:=0;
0254         MerkDat1:=FALSE;
0255         MerkDat2:=FALSE;
0256         Diagnosis:=Diagnosis AND 16#EF;
0257     END_IF;
0258 END_IF;
0259
0260 IF SCSSlave=FALSE THEN
0261     Diagnosis:=Diagnosis OR 16#80;
0262     ARS0Request:=FALSE;
0263     ARSResponse_0:=FALSE;

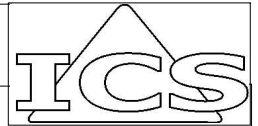
```



```

0264 ARS1Request:=FALSE;
0265 ARSResponse_1:=FALSE;
0266 AWS2Request:=FALSE;
0267 ID_Data[1]:=0;
0268 ELSE
0269     Diagnosis:=Diagnosis AND 16#7F;
0270 END_IF;
0271
0272 (* Kommandoaufrufe: ID des Slaves schon bekannt? *)
0273 IF ID_Data[1]=0 AND Acyclic=0 AND Send=FALSE AND SCSSlave=TRUE THEN
0274 (* ID des Slaves abfragen *)
0275     OutputData[0]:=16;
0276     OutputData[1]:=0;
0277     OutputData[2]:=6;
0278     OutputLen:=3;
0279     Send:=TRUE;
0280     ARS0Request:=TRUE;
0281     Acyclic:=100;
0282     Diagnosis:=Diagnosis OR 16#10;
0283 END_IF;
0284
0285 (* Kommandoaufrufe: Diagnose vom Slave zyklisch abrufen *)
0286 IF ARSResponse_0=TRUE AND ARSResponse_1=FALSE AND Acyclic=0 AND Send=FALSE AND
0287     SCSSlave=TRUE THEN
0288 (* Diagnose des Slaves abfragen *)
0289     OutputData[0]:=16;
0290     OutputData[1]:=1;
0291     OutputData[2]:=1;
0292     OutputLen:=3;
0293     Send:=TRUE;
0294     ARS1Request:=TRUE;
0295     Acyclic:=101;
0296     Diagnosis:=Diagnosis OR 16#10;
0297 END_IF;
0298
0299 (* Kommandoaufrufe: Parameter des Slaves schon bekannt? *)
0300 IF AWS2Request=FALSE AND Acyclic=0 AND Send=FALSE AND SCSSlave=TRUE THEN
0301 (* Parameter an Slave senden *)
0302     OutputData[0]:=17;
0303     OutputData[1]:=2;
0304     OutputData[2]:=8;
0305     FOR i:=0 TO 7 DO
0306         OutputData[i+3]:=Text[i];
0307     END_FOR;
0308     OutputLen:=11;
0309     Send:=TRUE;
0310     AWS2Request:=TRUE;
0311     ARSResponse_2:=FALSE;
0312     Acyclic:=103;
0313     Diagnosis:=Diagnosis OR 16#10;
0314 END_IF;
0315
0316 (* Kommandoaufrufe: Parameter des Slaves schon bekannt? *)
0317 IF ARSResponse_2=FALSE AND Acyclic=0 AND Send=FALSE AND SCSSlave=TRUE THEN
0318 (* Parameter des Slaves abfragen *)
0319     OutputData[0]:=16;
0320     OutputData[1]:=2;
0321     OutputData[2]:=8;
0322     OutputLen:=3;

```



```

0323 Send:=TRUE;
0324 ARS2Request:=TRUE;
0325 Acyclic:=102;
0326 Diagnosis:=Diagnosis OR 16#10;
0327 END_IF;
0328
0329 (* Wenn einzeliges Display, dann zyklische Ausgabe! *)
0330 IF Send=FALSE AND SCSSlave=TRUE AND ID_Data[4]=208 AND (DatInalt[1]<>DatIn[1] OR DatInalt[0]<>DatIn[0]) THEN
0331   OutputData[0]:=1;
0332   OutputData[1]:=DatIn[0];
0333   OutputData[2]:=DatIn[1];
0334   DatInalt[0]:=DatIn[0];
0335   DatInalt[1]:=DatIn[1];
0336   OutputLen:=3;
0337   Send:=TRUE;
0338 END_IF;
0339
0340 IF Newdata=TRUE THEN
0341 (* ID-Daten sind angekommen *)
0342   IF InputData[0]=80 AND Acyclic=100 THEN
0343     FOR i:=0 TO 5 DO
0344       ID_Data[i]:=InputData[i+1];
0345     END_FOR;
0346     NewData:=FALSE;
0347     ARSResponse_0:=TRUE;
0348     Timeout2:=0;
0349     Acyclic:=0;
0350     IF ID_Data[1]<>1 OR ID_Data[3]<>5 THEN
0351       Diagnosis:=Diagnosis OR 16#40;
0352     ELSE
0353       Diagnosis:=Diagnosis AND 16#BF;
0354     END_IF;
0355     Diagnosis:=Diagnosis AND 16#EF;
0356   END_IF;
0357 (* Diagnosedaten sind angekommen *)
0358   IF InputData[0]=80 AND Acyclic=101 THEN
0359     InputData[1]:=InputData[1] AND 16#0F;
0360     Diagnosis:=Diagnosis AND 16#F0;
0361     Diagnosis:=Diagnosis OR InputData[1];
0362     NewData:=FALSE;
0363     ARSResponse_1:=TRUE;
0364     Timeout2:=0;
0365     Acyclic:=0;
0366     Diagnosis:=Diagnosis AND 16#EF;
0367   END_IF;
0368 (* Parameter-Istdaten sind angekommen *)
0369   IF InputData[0]=80 AND Acyclic=102 THEN
0370     NewData:=FALSE;
0371     ARSResponse_2:=TRUE;
0372     Timeout2:=0;
0373     Acyclic:=0;
0374     FOR i:=0 TO 7 DO
0375       Para_DatIn[i]:=InputData[i+1];
0376     END_FOR;
0377     Diagnosis:=Diagnosis AND 16#EF;
0378   END_IF;
0379 (* Acknowledge für Parameter Schreiben ist angekommen *)
0380   IF InputData[0]=81 AND Acyclic=103 THEN
0381     NewData:=FALSE;

```

```
0382   AWSResponse_2:=TRUE;
0383   Timeout2:=0;
0384   Acyclic:=0;
0385   Diagnosis:=Diagnosis AND 16#EF;
0386   END_IF;
0387
0388 (* Fehlerhafte Aufrufe*)
0389   IF InputData[0]=145 OR InputData[0]=144 THEN
0390     InputData[1]:=InputData[1] AND 16#0F;
0391     Diagnosis:=Diagnosis AND 16#F0;
0392     Diagnosis:=Diagnosis OR InputData[1];
0393     NewData:=FALSE;
0394     Timeout2:=0;
0395     Acyclic:=0;
0396     Diagnosis:=Diagnosis AND 16#EF;
0397   END_IF;
0398 (* irgendein Fehler ist angekommen *)
0399   IF (InputData[0]=81 OR InputData[0]=0) AND Acyclic=0 THEN
0400     NewData:=FALSE;
0401     Diagnosis:=Diagnosis AND 16#EF;
0402   END_IF;
0403 END_IF;
0404
0405 Taktalt:=Takt;
0406 Inalt:=In;
0407 Varalt:=Variable;
```

Projektinformationen
PLC_PRG (PRG-ST)
SerComBCD (FB-ST)

A
1
1

Beispielprogramm