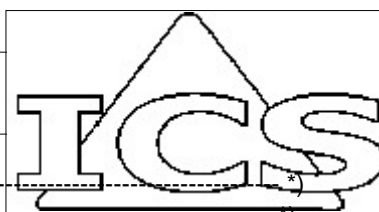


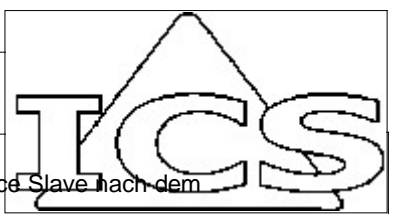
Dateiname: Example_Counter.pro
Verzeichnis: D:\Andreas_Schiff\ICS GmbH\ICS\internet\internet-PDFs
Geändert am: 1.3.05 14:43:23 / V2.2
Bezeichnung: Example_Counter
Autor: A. Schiff, ICS GmbH
Version: 0.7 vom 27.01.2005
Beschreibung: Das Programm kommuniziert mit dem Counter Module CM001, liest die Eingangsvariablen und schreibt die Parameter



```

0001 PROGRAM PLC_PRG
0002 (* -----*)
0003 (*           Parameter-Ausleseprogramm für AS-Interface CM001, 2004
0004 (* ICS GmbH, Tettngang, Andreas Schiff           Version 0.7           27.01.2005           *)
0005 (* -----*)
0006 VAR
0007 (* Variablen für E/As *)
0008 CounterModuleDI0 AT %IX1.5.0:   BOOL;      (* Ausgang DI0 des CounterModules *)
0009 CounterModuleDI1 AT %IX1.5.1:   BOOL;      (* Ausgang DI1 des CounterModules *)
0010 ResetZ AT %QX1.5.2:  BOOL;          (* Eingang des CounterModules zum Rücksetzen des Zählers *)
0011 DI2 AT %IX1.5.2:  BOOL;          (* Ein- und Ausgänge für die serielle Kommunikation *)
0012 DI3 AT %IX1.5.3:  BOOL;
0013 DO0 AT %QX1.5.0:  BOOL;
0014 DO1 AT %QX1.5.1:  BOOL;
0015 (* Hilfsvariablen *)
0016 Limit1:WORD;
0017 Limit2:WORD;
0018 Parameter: ARRAY[1..8] OF BYTE:=2,0,0,0,0,0,0,0;
0019 CounterMod: SerComCM;
0020 iw:INT;
0021 SW2O: WORD;
0022 SW1U: WORD;
0023 Product_ID: WORD;
0024 Vendor_ID: WORD;
0025 Comp: BYTE;
0026 Vers: BYTE;
0027 CMode: BYTE;
0028 CDamp: BYTE;
0029 Diagnose:BYTE;
0030 END_VAR
0031
0032 (* Ende des Deklarationsteils *)
0001 Parameter[3]:=INT_TO_BYTE(Limit1/256);
0002 Parameter[4]:=INT_TO_BYTE(Limit1-Parameter[3]*256);
0003 Parameter[5]:=INT_TO_BYTE(Limit2/256);
0004 Parameter[6]:=INT_TO_BYTE(Limit2-Parameter[5]*256);
0005
0006 CounterMod(DI2:=DI2, DI3:=DI3, Para_DatIn:=Parameter);
0007 DO1:=CounterMod.DO1;
0008 DO0:=CounterMod.DO0;
0009 iw:=CounterMod.InputWord;
0010 Vendor_ID:=CounterMod.ID_Data[0]*256+CounterMod.ID_Data[1];
0011 Product_ID:=CounterMod.ID_Data[2]*256+CounterMod.ID_Data[3];
0012 Comp:=CounterMod.ID_Data[4];
0013 Vers:=CounterMod.ID_Data[5];
0014 CMode:=CounterMod.Para_DatOut[1];
0015 CDamp:=CounterMod.Para_DatOut[2];
0016 Diagnose:=CounterMod.Diagnosis;
0017 SW2O:=CounterMod.Para_DatOut[5]*256+CounterMod.Para_DatOut[6];
0018 SW1U:=CounterMod.Para_DatOut[3]*256+CounterMod.Para_DatOut[4];

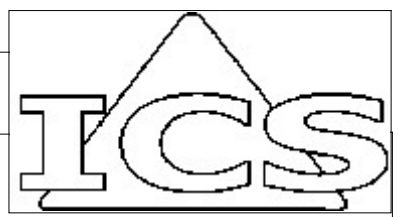
```



```

0001 FUNCTION_BLOCK SerComCM
0002 (* Dieser Funktionsbaustein wickelt die gesamte serielle Kommunikation zu einem AS-Interface Slave nach dem
0003 Profil S-7.A.5 ab.
0004 Rücksetzen der Rückmeldebits vom Slave ergänzt Datum: 10.01.2005 *)
0005
0006 VAR_INPUT
0007     DI2:      BOOL;          (* Eingänge für serielle Kommunikation *)
0008     DI3:      BOOL;
0009     Para_DatIn: ARRAY [1..8] OF BYTE :=2,0,7,208,3,232; (* Parameter-Solldaten *)
0010 END_VAR
0011
0012 VAR_OUTPUT
0013     InputWord: INT;          (* Eingabe-Datenfeld: Zählerstand *)
0014     Diagnosis: BYTE;        (* Diagnose-Daten: Bit 7: Slave nicht im seriellen Modus
0015                               Bit 6: ID nicht korrekt
0016                               Bit 5: ---
0017                               Bit 4: Slave busy
0018                               Diagnose vom Slave: Bit 3: ---
0019                               Diagnose vom Slave: Bit 2: ---
0020                               Diagnose vom Slave: Bit 1: Request not implemented
0021                               Diagnose vom Slave: Bit 0: Illegal Index *)
0022     ID_Data:   ARRAY [0..15] OF BYTE; (* ID-Daten des Slaves *)
0023     Para_DatOut: ARRAY [0..8] OF BYTE; (* Parameter-Istdaten des Slaves *)
0024     DO1:      BOOL;          (* Ausgänge für serielle Kommunikation *)
0025     DO0:      BOOL;
0026 END_VAR
0027
0028 VAR
0029     InputD:   ARRAY [0..15] OF BYTE; (* Eingabe-Datenfeld *)
0030     InputData: ARRAY [0..15] OF BYTE; (* Eingabe-Datenfeld *)
0031     OutputData: ARRAY [0..15] OF BYTE; (* Ausgabe-Datenfeld *)
0032     OutputLen: BYTE;          (* Länge der Ausgabedaten in Byte; Zahlenbereich 1..15 *)
0033     InputL:   BYTE;          (* Länge der Eingabedaten in Byte; Zahlenbereich 1..15 *)
0034     InputLen: BYTE;
0035     Taktalt:  BOOL;
0036     In:      BYTE;          (* aktuell empfangene Nachricht *)
0037     Inalt:   BYTE;          (* letzte empfangene Nachricht *)
0038     Out:     BYTE:= 3;
0039     Watchdog: BOOL;
0040     Timeout: BYTE;
0041     LastIn:  BYTE;          (* letztes gültiges empfangenes Informationsbit *)
0042     i:      BYTE;
0043     IBitzeiger: BYTE;
0044     Outalt:  BYTE;
0045     Outzeiger: BYTE;
0046     OBitzeiger: BYTE;
0047     x:      BYTE;
0048     Send:   BOOL;
0049     y:      BYTE;
0050     NewData: BOOL;
0051     SSW:    WORD;
0052     Inst1:  TON;
0053     Inst2:  TON;
0054     Takt:   BOOL;
0055     Var2:   BOOL;
0056     Acyclic: WORD;
0057     Para_Data: ARRAY [0..8] OF BYTE;
0058     MerkiPara: BOOL;
0059     SCSlave: BOOL;
0060     Schluss: BOOL;

```



```

0061 Timeout2: BYTE;
0062 ARS0Request: BOOL;
0063 ARSResponse_0: BOOL;
0064 ARS1Request: BOOL;
0065 ARSResponse_1: BOOL;
0066 AWS2Request: BOOL;
0067 ARSResponse_2: BOOL;
0068 AWSResponse_2: BOOL;
0069 ARS2Request: BOOL;
0070 END_VAR
0071
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
0007 (* Eingänge des Slaves lesen, ausmaskieren und auf Veränderung prüfen *)
0008 IF DI2=FALSE AND DI3=FALSE THEN In:=0; END_IF;
0009 IF DI2=FALSE AND DI3=TRUE THEN In:=2; END_IF;
0010 IF DI2=TRUE AND DI3=FALSE THEN In:=1; END_IF;
0011 IF DI2=TRUE AND DI3=TRUE THEN In:=3; END_IF;
0012
0013 IF In<>Inalt THEN
0014     (* eine Antwort ist eingetroffen *)
0015     IF Out=1 AND In=1 THEN
0016         (* die eine der erwarteten Antworten ist eingetroffen: Separator *)
0017         Watchdog:=FALSE;
0018         SCSlave:=TRUE;
0019         Timeout:=0;
0020         Lastin:=Inalt;
0021         (* Aktion 2: Trenn-Nachricht empfangen; Master sendet Information oder Idle *)
0022         IF Send=TRUE THEN
0023             IF Outalt=3 THEN
0024                 (* Ausgabe initialisieren *)
0025                 Outzeiger:=0;
0026                 OBitzeiger:=0;
0027                 y:=OutputData[0];
0028             END_IF;
0029             IF Schluss=FALSE THEN
0030                 Outalt:=Out;
0031                 x:=y AND 2#1000_0000;
0032                 IF x<>0 THEN
0033                     Out:=2;
0034                 ELSE
0035                     Out:=0;
0036                 END_IF;
0037                 y:=SHL(y,1);
0038                 OBitzeiger:=OBitzeiger+1;
0039                 (* ein Byte fertig? Zeiger auf nächstes Byte stellen *)
0040                 IF OBitzeiger>7 THEN
0041                     Outzeiger:=OBitzeiger+1;
0042                     y:=OutputData[Outzeiger];
0043                     OBitzeiger:=0;
0044                 IF Outzeiger>=OutputLen OR Outzeiger>=19 THEN
0045                     Schluss:=TRUE;
0046                     SCSlave:=TRUE;
0047                 END_IF;
0048             END_IF;
0049             ELSE

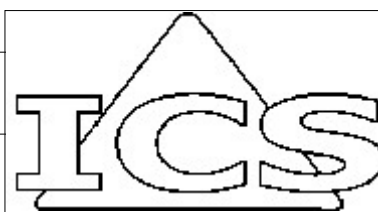
```



```

0050      Outalt:=Out;
0051      Out:=3; (* zum Abschluss einer Nachricht wird mindestens 1 Idle gesendet! *)
0052      Send:=FALSE;
0053      Schluss:=FALSE;
0054      END_IF;
0055      ELSE
0056      Outalt:=Out;
0057      Out:=3;
0058      END_IF;
0059      IF Out=0 THEN DO0:=TRUE; DO1:=TRUE; END_IF;
0060      IF Out=1 THEN DO0:=FALSE; DO1:=TRUE; END_IF;
0061      IF Out=2 THEN DO0:=TRUE; DO1:=FALSE; END_IF;
0062      IF Out=3 THEN DO0:=FALSE; DO1:=FALSE; END_IF;
0063      ELSIF Out<>1 AND In<>1 THEN
0064      (* die andere der erwarteten Antworten ist eingetroffen: Idle oder Daten *)
0065      Watchdog:=FALSE;
0066      SCSSlave:=TRUE;
0067      Timeout:=0;
0068      IF (In=0 AND Inalt<>2) OR (In=2 AND Inalt<>0) THEN
0069      (* Aktion 0/1: Slave hat Informationsbit gesendet: abspeichern; Master sendet Trenn-Nachricht *)
0070      IF Lastin=3 THEN
0071      (* Beginn einer Nachricht: Zeiger und Feld Initialisieren, wenn Feld bereit *)
0072      NewData:=FALSE;
0073      InputL:=0;
0074      IBitzeiger:=0;
0075      FOR i:=0 TO 19 DO
0076      InputD[i]:=0;
0077      END_FOR;
0078      END_IF;
0079      InputD[InputL]:=SHL(InputD[InputL],1);
0080      IF In=2 THEN
0081      InputD[InputL]:=InputD[InputL]+1;
0082      END_IF;
0083      IBitzeiger:=IBitzeiger+1;
0084      IF IBitzeiger>7 THEN
0085      InputL:=InputL+1;
0086      IF InputL>19 THEN
0087      InputL:=19;
0088      END_IF;
0089      IBitzeiger:=0;
0090      END_IF;
0091      Outalt:=Out;
0092      Out:=1;
0093      IF Out=0 THEN DO0:=TRUE; DO1:=TRUE; END_IF;
0094      IF Out=1 THEN DO0:=FALSE; DO1:=TRUE; END_IF;
0095      IF Out=2 THEN DO0:=TRUE; DO1:=FALSE; END_IF;
0096      IF Out=3 THEN DO0:=FALSE; DO1:=FALSE; END_IF;
0097      ELSE
0098      (* Aktion 3: Slave hat Idle gesendet: Abschluss einer Nachricht?; Master sendet Trenn-Nachricht *)
0099      IF Lastin<>3 THEN
0100      (* Abschluss einer Nachricht; wenn diese nicht vollständig (also ganzzahlige Anzahl von Bytes),
0101      dann wird sie verworfen! *)
0102      IF IBitzeiger:=0 THEN
0103      FOR i:=0 TO InputL DO
0104      InputData[i]:=InputD[i];
0105      END_FOR;
0106      NewData:=TRUE;
0107      InputLen:=InputL;
0108      SCSSlave:=TRUE;
0109      END_IF;

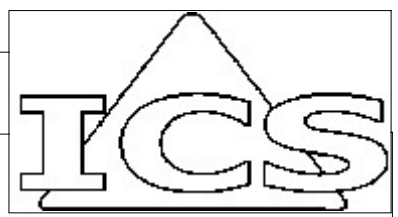
```



```

0110     END_IF;
0111     Outalt:=Out;
0112     Out:=1;
0113     IF Out=0 THEN DO0:=TRUE; DO1:=TRUE; END_IF;
0114     IF Out=1 THEN DO0:=FALSE; DO1:=TRUE; END_IF;
0115     IF Out=2 THEN DO0:=TRUE; DO1:=FALSE; END_IF;
0116     IF Out=3 THEN DO0:=FALSE; DO1:=FALSE; END_IF;
0117     END_IF;
0118 END_IF;
0119 ELSIF (Takt<>Taktalt AND Takt=TRUE) THEN
0120 (* Slave alle 120ms testen, ob er den seriellen Kommunikationsmodus unterstützt, dient auch zum Anlauf
0121 der Kommunikation *)
0122 Timeout:=Timeout+1;
0123 IF Timeout>4 THEN
0124     Outalt:=Out;
0125     IF (Out=0 OR Out=3) THEN
0126         Out:=1;
0127     ELSIF (Out=1 OR Out=2) THEN
0128         Out:=3;
0129     END_IF;
0130     Timeout:=0;
0131     SCSSlave:=FALSE;
0132     SSW:=0;
0133     IF Out=0 THEN DO0:=TRUE; DO1:=TRUE; END_IF;
0134     IF Out=1 THEN DO0:=FALSE; DO1:=TRUE; END_IF;
0135     IF Out=2 THEN DO0:=TRUE; DO1:=FALSE; END_IF;
0136     IF Out=3 THEN DO0:=FALSE; DO1:=FALSE; END_IF;
0137     END_IF;
0138 END_IF;
0139
0140 (* und hier läuft die übergeordnete Steuerung des seriellen Slaves ab. Zyklus: 16s *)
0141 IF Taktalt<>Takt AND Takt=FALSE AND Acyclic<>0 THEN
0142     Timeout2:=Timeout2+1;
0143     IF Timeout2>151 THEN
0144         Timeout2:=0;
0145         Acyclic:=0;
0146         Diagnosis:=Diagnosis AND 16#EF;
0147     END_IF;
0148 END_IF;
0149
0150 IF SCSSlave=FALSE THEN
0151     Diagnosis:=Diagnosis OR 16#80;
0152     ARS0Request:=FALSE;
0153     ARSResponse_0:=FALSE;
0154     ARS1Request:=FALSE;
0155     ARSResponse_1:=FALSE;
0156     AWS2Request:=FALSE;
0157     ID_Data[1]:=0;
0158     FOR i:=0 TO 8 DO
0159         Para_Data[i]:=0;
0160         Para_DatOut[i]:=0;
0161     END_FOR;
0162 ELSE
0163     Diagnosis:=Diagnosis AND 16#7F;
0164 END_IF;
0165
0166 (* Kommandoanfrage: ID des Slaves schon bekannt? *)
0167 IF ID_Data[1]=0 AND Acyclic=0 AND Send=FALSE AND SCSSlave=TRUE THEN
0168 (* ID des Slaves anfragen *)
0169     OutputData[0]:=16;

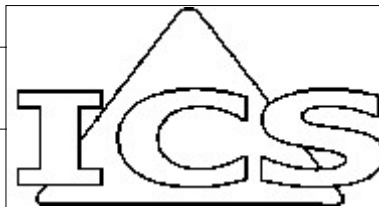
```



```

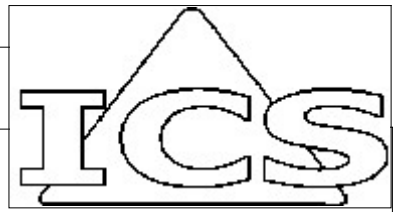
0170 OutputData[1]:=0;
0171 OutputData[2]:=6;
0172 OutputLen:=3;
0173 Send:=TRUE;
0174 ARS0Request:=TRUE;
0175 Diagnosis:=Diagnosis OR 16#10;
0176 Acyclic:=100;
0177 END_IF;
0178
0179 (* Zyklisch das Diagnose-Byte abfragen *)
0180 IF ARSResponse_0=TRUE AND ARSResponse_1=FALSE AND Acyclic=0 AND Send=FALSE AND SC Slave=TRUE THEN
0181 (* Diagnose des Slaves abfragen *)
0182 OutputData[0]:=16;
0183 OutputData[1]:=1;
0184 OutputData[2]:=1;
0185 OutputLen:=3;
0186 Send:=TRUE;
0187 ARS1Request:=TRUE;
0188 Diagnosis:=Diagnosis OR 16#10;
0189 Acyclic:=101;
0190 END_IF;
0191
0192 (* Parameter des Slaves schon bekannt? *)
0193 IF ARSResponse_2=FALSE AND Acyclic=0 AND Send=FALSE AND SC Slave=TRUE THEN
0194 OutputData[0]:=16;
0195 OutputData[1]:=2;
0196 OutputData[2]:=6;
0197 OutputLen:=3;
0198 Send:=TRUE;
0199 ARS2Request:=TRUE;
0200 Acyclic:=102;
0201 Diagnosis:=Diagnosis OR 16#10;
0202 END_IF;
0203
0204 (* Wenn Veränderung, dann Parameter schreiben *)
0205 IF ARSResponse_2=TRUE AND SC Slave=TRUE AND Acyclic=0 AND Send=FALSE THEN
0206 MerkPara:=FALSE;
0207 FOR i:=1 TO 6 DO
0208 IF Para_DatIn[i]<>Para_Data[i] THEN
0209 MerkPara:=TRUE;
0210 END_IF;
0211 END_FOR;
0212 IF MerkPara=TRUE THEN
0213 OutputData[0]:=17;
0214 OutputData[1]:=2;
0215 OutputData[2]:=6;
0216 FOR i:=1 TO 6 DO
0217 OutputData[i+2]:=Para_DatIn[i];
0218 END_FOR;
0219 OutputLen:=9;
0220 AWS2Request:=TRUE;
0221 ARSResponse_2:=FALSE;
0222 Send:=TRUE;
0223 Acyclic:=103;
0224 END_IF;
0225 END_IF;
0226
0227 IF Newdata=TRUE THEN
0228 (* ID-Daten sind angekommen *)
0229 IF InputData[0]=80 AND Acyclic=100 THEN

```



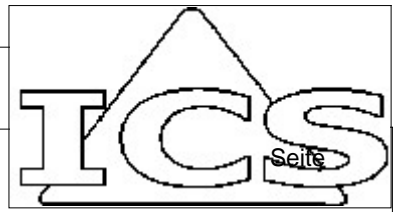
```

0230   FOR i:=1 TO 6 DO
0231       ID_Data[i]:=InputData[i+1];
0232   END_FOR;
0233   NewData:=FALSE;
0234   ARSResponse_0:=TRUE;
0235   Timeout2:=0;
0236   Acyclic:=0;
0237   IF ID_Data[1]<>1 OR ID_Data[3]<>4 THEN      (* richtiger Slave? *)
0238       Diagnosis:=Diagnosis OR 16#40;
0239   ELSE
0240       Diagnosis:=Diagnosis AND 16#BF;
0241   END_IF;
0242   Diagnosis:=Diagnosis AND 16#E0;
0243 END_IF;
0244 (* Diagnosedaten sind angekommen *)
0245 IF InputData[0]=80 AND Acyclic=101 THEN
0246     InputData[1]:=InputData[1] AND 16#0F;
0247     Diagnosis:=Diagnosis AND 16#F0;
0248     Diagnosis:=Diagnosis OR InputData[1];
0249     NewData:=FALSE;
0250     ARSResponse_1:=TRUE;
0251     Acyclic:=0;
0252     Diagnosis:=Diagnosis AND 16#EF;
0253 END_IF;
0254 (* Parameter-Istdaten sind angekommen *)
0255 IF InputData[0]=80 AND Acyclic=102 THEN
0256     FOR i:=1 TO 7 DO
0257         Para_DatOut[i]:=InputData[i];
0258         Para_Data[i]:=InputData[i];
0259     END_FOR;
0260     NewData:=FALSE;
0261     ARSResponse_2:=TRUE;
0262     Timeout2:=0;
0263     Diagnosis:=Diagnosis AND 16#E0;
0264     Acyclic:=0;
0265 END_IF;
0266 (* Acknowledge for Parameter Schreiben ist angekommen *)
0267 IF InputData[0]=81 AND Acyclic=103 THEN
0268     NewData:=FALSE;
0269     AWSResponse_2:=TRUE;
0270     Timeout2:=0;
0271     Acyclic:=0;
0272     Diagnosis:=Diagnosis AND 16#E0;
0273 END_IF;
0274 (* Messdaten sind angekommen *)
0275 IF InputData[0]=0 AND SCSSlave=TRUE THEN
0276     InputWord:=256*InputData[1]+InputData[2];
0277     NewData:=FALSE;
0278 END_IF;
0279 (* Aufruf nicht ok *)
0280 IF InputData[0]=144 OR InputData[0]=145 THEN
0281     InputData[1]:=InputData[1] AND 16#0F;
0282     Diagnosis:=Diagnosis AND 16#F0;
0283     Diagnosis:=Diagnosis OR InputData[1];
0284     NewData:=FALSE;
0285     Timeout2:=0;
0286     Diagnosis:=Diagnosis AND 16#EF;
0287     Acyclic:=0;
0288 END_IF;
0289 (* irgendein Fehler ist eingetroffen *)
  
```



```
0290 IF (InputData[0]=81 OR InputData[0]=0) AND Acyclic=0 THEN
0291     NewData:=FALSE;
0292     Diagnosis:=Diagnosis AND 16#EF;
0293 END_IF;
0294 END_IF;
0295
0296 Taktalt:=Takt;
0297 Inalt:=In;
```

Beispielprogramm



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

A
1-1
2-1

Beispielprogramm