FUNCTION_BLOCK ModbusConversion

(eLLabUtyLib_C050) Modbus Ascii/RTU conversion

```
VAR_INPUT
Enable : BOOL; (* FB enable *)
SpyOn : BOOL; (* Spy active *)
FpAscii : FILEP; (* File pointer (Modbus Ascii) *)
FpRTU : FILEP; (* File pointer (Modbus RTU) *)
IFTime : UDINT; (* Interframe time (uS) *)
Timeout : UDINT; (* Timeout time (mS) *)
END_VAR

VAR_OUTPUT
Enabled : BOOL; (* FB enabled *)
Fault : BOOL; (* FB fault *)
END_VAR

VAR
DFAscii : ARRAY[ 0..269 ] OF USINT; (* Data frame (Modbus Ascii) *)
i : INT; (* Auxiliary variable *)
TimeBf : ARRAY[ 0..2 ] OF UDINT; (* Time buffer (uS) *)
GetCRC : SysGetCrc; (* Get CRC *)
CaseNr : ARRAY[ 0..1 ] OF USINT; (* Case programma *)
IDx : ARRAY[ 0..1 ] OF UINT; (* Data frame index *)
Ch : USINT; (* Character buffer *)
FFlop : BOOL; (* Flip/Flop *)
DFRTU : ARRAY[ 0..269 ] OF USINT; (* Data frame (Modbus RTU) *)
ErrorNr : UDINT; (* Error number *)
END_VAR
```

```
(* ********************************************************************** *)
(* FUNCTION BLOCK "ModbusConversion"                                      *)
(* ********************************************************************** *)
(* Questa FB esegue la conversione tra il modbus Ascii ed il modbus RTU.  *)
(*                                                                        *)
(* Parametri in ingresso:                                                 *)
(* Enable: Abilitazione FB                                                *)
(* SpyOn: Attiva spionaggio dati                                          *)
(* FpAscii: File pointer (Stream Modbus Ascii)                            *)
(* FpRTU: File pointer (Stream Modbus RTU)                                *)
(* IFTime: Tempo attesa carattere                                         *)
(* Timeout: Tempo timeout risposta (mS)                                   *)
(*                                                                        *)
(* Parametri in uscita:                                                   *)
(* Enabled: FB abilitata                                                  *)
(* Fault: Fault                                                           *)
(* ErrorNr: Numero di errore                                              *)
(* ---------------------------------------------------------------------- *)
(* Spy TFlags                                                             *)
(* 16#00000001, 'Ax' Frame Modbus Ascii ricevuto                          *)
(* 16#00000002, 'Rx' Frame Modbus RTU ricevuto                            *)
(* 16#40000000, 'Er' Errori di esecuzione                                 *)
(* ---------------------------------------------------------------------- *)

    (* ---------------------------------------------------------------- *)
    (* INIZIALIZZAZIONI                                                 *)
    (* ---------------------------------------------------------------- *)
    (* Eseguo Spy per averlo attivo da Telnet. Azzero bit di fault. *)

```

```
30      IF (SpyOn) THEN i:=SysSpyData(0, 0, 0, 0); END_IF;
31      IF (Fault) THEN Fault:=FALSE; ErrorNr:=0; END_IF;
32
33      (* Eseguo gestione report errore. *)
34
35      IF (ErrorNr <> 0) THEN
36          IF (SpyOn) THEN
37              IF NOT(SysSpyData(0, 0, 0, 0)) THEN RETURN; END_IF;
38              i:=TO_UINT(SysVarsnprintf(ADR(DFAscii), SIZEOF(DFAscii), 'Error:%08d', UDINT_TYPE, ADR(Erro
    rNr)));
39              i:=SysSpyData(SPY_ASCII, 16#40000000, ADR('Er'), ADR(DFAscii));
40          END_IF;
41
42          (* Set error code ed impulso di "Fault". *)
43
44          Fault:=TRUE; (* Execution fault*)
45          CaseNr[0]:=0; (* Case programma *)
46          CaseNr[1]:=0; (* Case programma *)
47          RETURN; (* Esco per propagare impulso di "Fault" *)
48      END_IF;
49
50      (* ------------------------------------------------------------------ *)
51      (* GESTIONE ABILITAZIONE                                              *)
52      (* ------------------------------------------------------------------ *)
53      (* Gestione abilitazione. *)
54
55      IF NOT(Enable) THEN Enabled:=FALSE; RETURN; END_IF;
56
57      (* One shot abilitazione FB.*)
58
59      IF NOT(Enabled) THEN Enabled:=TRUE; CaseNr[0]:=0; CaseNr[1]:=0; END_IF;
60
61      (* ================================================================== *)
62      (* CONVERSIONE DA MODBUS ASCII A RTU                                  *)
63      (* ================================================================== *)
64
65      (* ------------------------------------------------------------------ *)
66      (* GESTIONE TIMEOUT SEQUENZA                                          *)
67      (* ------------------------------------------------------------------ *)
68      (* Gestione timeout, l'intero comando deve chiudersi nel tempo definito. *)
69
70      IF (CaseNr[0] = 0) THEN TimeBf[0]:=SysGetSysTime(TRUE); END_IF;
71      IF ((SysGetSysTime(TRUE)-TimeBf[0]) >= (Timeout*1000)) THEN
72          ErrorNr:=10069100; (* Error number *)
73          RETURN;
74      END_IF;
75
76      (* ------------------------------------------------------------------ *)
77      (* GESTIONE CASES                                                     *)
78      (* ------------------------------------------------------------------ *)
79      (* Gestione cases conversione da Ascii ad RTU. *)
80
81      WHILE (TRUE) DO
82          CASE (CaseNr[0]) OF
83
84              (* ---------------------------------------------------------- *)
85              (* RICEZIONE PACCHETTO ASCII                                  *)
86              (* ---------------------------------------------------------- *)
87              (* Controllo se ricevuto carattere ":" di start, loop su WHILE. *)
88
```

```
 89            0:
 90            IF NOT(TO_BOOL(SysGetIChars(FpAscii))) THEN EXIT; END_IF;
 91            IF (TO_USINT(Sysfgetc(FpAscii)) = 16#3A) THEN
 92                FFlop:=FALSE; (* Flip/Flop *)
 93                IDx[0]:=0; (* Data frame index *)
 94                CaseNr[0]:=CaseNr[0]+1; (* Case programma *)
 95            END_IF;
 96
 97            (* ------------------------------------------------------------ *)
 98            (* Controllo se ricevo caratteri frame ascii. *)
 99
100            1:
101            IF NOT(TO_BOOL(SysGetIChars(FpAscii))) THEN EXIT; END_IF;
102            Ch:=TO_USINT(Sysfgetc(FpAscii)); (* Character buffer *)
103
104            (* Eseguo controllo se <CR>, fine stringa. *)
105
106            IF (Ch = 16#0D) THEN CaseNr[0]:=CaseNr[0]+1; RETURN; END_IF;
107
108            (* Eseguo controllo dato Ascii in esadecimale. *)
109
110            IF (Ch < 16#30) OR (Ch > 16#46) OR ((Ch > 16#39) AND (Ch < 16#41)) THEN
111                ErrorNr:=10069110; (* Error number *)
112                RETURN;
113            END_IF;
114
115            (* Eseguo conversione dato Ascii in esadecimale. *)
116
117            IF (Ch <= 16#39) THEN Ch:=Ch-16#30; ELSE Ch:=Ch-16#37; END_IF;
118
119            (* Carico carattere ricevuto in stringa ricezione. *)
120
121            IF NOT(FFlop) THEN
122                FFlop:=TRUE; (* Flip/Flop *)
123                DFAscii[IDx[0]]:=Ch*16; (* Data frame *)
124            ELSE
125                FFlop:=FALSE; (* Flip/Flop *)
126                DFAscii[IDx[0]]:=DFAscii[IDx[0]]+Ch; (* Data frame *)
127                IDx[0]:=IDx[0]+1; (* Data frame index *)
128            END_IF;
129
130            (* Controllo limite caratteri, lascio spazio per 2 byte di CRC. *)
131
132            IF (IDx[0] >= (SIZEOF(DFAscii)-2)) THEN
133                ErrorNr:=10069150; (* Error number *)
134                RETURN;
135            END_IF;
136
137            (* ------------------------------------------------------------ *)
138            (* Controllo se ricevuto carattere "LF" di fine, loop su WHILE. *)
139
140            2:
141            IF NOT(TO_BOOL(SysGetIChars(FpAscii))) THEN EXIT; END_IF;
142            IF (TO_USINT(Sysfgetc(FpAscii)) <> 16#0A) THEN ErrorNr:=10069160; RETURN; END_IF;
143            CaseNr[0]:=CaseNr[0]+1; (* Case programma *)
144
145            (* ------------------------------------------------------------ *)
146            (* Se attivo eseguo spy frame ricevuto. Limito spy a 63 caratteri *)
147            (* per evitare roll over su somma. *)
148
```

```
149             3:
150             IF (SpyOn) THEN
151                 IF NOT(SysSpyData(0, 0, 0, 0)) THEN EXIT; END_IF;
152                 i:=SysSpyData(SPY_BINARY+TO_USINT(LIMIT(IDx[0], 0, 63)), 16#00000001, ADR('Ax'), ADR(DF
    Ascii));
153             END_IF;
154             CaseNr[0]:=CaseNr[0]+1; (* Case programma *)
155
156             (* ------------------------------------------------------------ *)
157             (* Controllo numero minimo caratteri ricevuti, almeno 5. *)
158             (* [Nodo], [Funzione], 2 di dato, [LRC]. *)
159
160             4:
161             IF (IDx[0] < 5) THEN ErrorNr:=10069170; RETURN; END_IF;
162
163             (* Eseguo controllo LRC su frame ricevuto. *)
164
165             Ch:=0; (* Character buffer *)
166             FOR i:=0 TO TO_INT(IDx[0]-1) DO Ch:=Ch+DFAscii[i]; END_FOR;
167
168             (* Errore LRC frame ricevuto, il risultato deve essere "0". *)
169
170             IF (Ch <> 0) THEN ErrorNr:=10069180; RETURN; END_IF;
171
172             (* Elimino da lunghezza frame LRC. *)
173
174             IDx[0]:=IDx[0]-1; (* Data frame index *)
175             CaseNr[0]:=10; (* Case programma *)
176
177             (* ------------------------------------------------------------ *)
178             (* TRASMISSIONE FRAME RTU                                       *)
179             (* ------------------------------------------------------------ *)
180             (* Controllo se spazio in buffer di trasmissione. *)
181
182             10:
183             IF (TO_UINT(SysGetOSpace(FpRTU)) < (IDx[0]+2)) THEN EXIT; END_IF;
184
185             (* Calcolo CRC frame da trasmettere. *)
186
187             GetCRC.Buf:=ADR(DFAscii); (* Buffer address *)
188             GetCRC.ByteNr:=IDx[0]; (* Byte number *)
189             GetCRC.CrcIni:=16#FFFF; (* CRC ini value *)
190             GetCRC(); (* Calculate CRC *)
191             DFAscii[IDx[0]]:=TO_USINT(GetCRC.Crc/256); (* MSB CRC *)
192             DFAscii[IDx[0]+1]:=TO_USINT(GetCRC.Crc); (* LSB CRC *)
193
194             (* Prima di trasmettere frame pulisco buffer di ricezione Ascii. *)
195
196             i:=SysFIBfClear(FpAscii);
197             i:=Sysfwrite(ADR(DFAscii), TO_INT(IDx[0]+2), 1, FpRTU);
198
199             (* Eseguo flush, così su socket la stringa esce immediatamente. *)
200
201             IF NOT(SysFOBfFlush(FpRTU)) THEN ErrorNr:=10069200; RETURN; END_IF;
202             CaseNr[0]:=CaseNr[0]+1; (* Case programma *)
203
204             (* ------------------------------------------------------------ *)
205             (* Eseguo attesa fine trasmissione frame. *)
206
207             11:
```

```
208             IF (SysGetOSpace(FpRTU) <> TO_INT(SysGetTxBSize(FpRTU))) THEN EXIT; END_IF;
209             CaseNr[0]:=0; (* Case programma *)
210         ELSE
211             CaseNr[0]:=0; (* Case programma *)
212         END_CASE;
213     END_WHILE;
214
215     (* ================================================================== *)
216     (* CONVERSIONE DA MODBUS RTU A ASCII                                  *)
217     (* ================================================================== *)
218
219     (* ------------------------------------------------------------------ *)
220     (* GESTIONE TIMEOUT SEQUENZA                                          *)
221     (* ------------------------------------------------------------------ *)
222     (* Gestione timeout, l'intero comando deve chiudersi nel tempo definito. *)
223
224     IF (CaseNr[1] = 0) THEN TimeBf[1]:=SysGetSysTime(TRUE); END_IF;
225     IF ((SysGetSysTime(TRUE)-TimeBf[1]) >= (Timeout*1000)) THEN
226         ErrorNr:=10069300; (* Error number *)
227         RETURN;
228     END_IF;
229
230     (* ------------------------------------------------------------------ *)
231     (* GESTIONE CASES                                                     *)
232     (* ------------------------------------------------------------------ *)
233     (* Gestione cases conversione da Ascii ad RTU. *)
234
235     WHILE (TRUE) DO
236         CASE (CaseNr[1]) OF
237
238             (* ------------------------------------------------------------ *)
239             (* RICEZIONE PACCHETTO RTU                                      *)
240             (* ------------------------------------------------------------ *)
241             (* Controllo se ricevuto caratteri e salvo tempo. *)
242
243             0:
244             IF NOT(TO_BOOL(SysGetIChars(FpRTU))) THEN EXIT; END_IF;
245             IDx[1]:=0; (* Data frame index *)
246             TimeBf[2]:=SysGetSysTime(TRUE); (* Time buffer (uS) *)
247             CaseNr[1]:=CaseNr[1]+1; (* Case programma *)
248
249             (* ------------------------------------------------------------ *)
250             (* Controllo se pausa ricezione frame ricevuto. *)
251
252             1:
253             IF NOT(TO_BOOL(SysGetIChars(FpRTU))) THEN
254                 IF ((SysGetSysTime(TRUE)-TimeBf[2]) > IFTime) THEN CaseNr[1]:=CaseNr[1]+1; END_IF;
255                 EXIT; (* Esco dal ciclo While *)
256             END_IF;
257
258             (* Carico carattere ricevuto in stringa ricezione. *)
259
260             TimeBf[2]:=SysGetSysTime(TRUE); (* Time buffer *)
261             DFRTU[IDx[1]]:=TO_USINT(Sysfgetc(FpRTU)); (* Rx string *)
262             IDx[1]:=IDx[1]+1; (* Data frame index *)
263
264             (* Controllo limite caratteri, lascio spazio per byte di LRC. *)
265
266             IF (IDx[1] >= (SIZEOF(DFRTU)-1)) THEN
267                 ErrorNr:=10069350; (* Error number *)
```

```
268                 RETURN;
269             END_IF;
270
271             (* ------------------------------------------------------------ *)
272             (* Se attivo eseguo spy frame ricevuto. Limito spy a 63 caratteri *)
273             (* per evitare roll over su somma. *)
274
275             2:
276             IF (SpyOn) THEN
277                 IF NOT(SysSpyData(0, 0, 0, 0)) THEN EXIT; END_IF;
278                 i:=SysSpyData(SPY_BINARY+TO_USINT(LIMIT(IDx[1], 0, 63)), 16#00000002, ADR('Rx'), ADR(DF
    RTU));
279             END_IF;
280             CaseNr[1]:=CaseNr[1]+1; (* Case programma *)
281
282             (* ------------------------------------------------------------ *)
283             (* Controllo numero minimo caratteri ricevuti, almeno 6. *)
284             (* [Nodo], [Funzione], 2 di dato, [CRC]. *)
285
286             3:
287             IF (IDx[1] < 6) THEN ErrorNr:=10069370; RETURN; END_IF;
288
289             (* Eseguo controllo CRC su frame ricevuto. *)
290
291             GetCRC.Buf:=ADR(DFRTU); (* Buffer address *)
292             GetCRC.ByteNr:=TO_UINT(IDx[1]); (* Byte number *)
293             GetCRC.CrcIni:=16#FFFF; (* CRC ini value *)
294             GetCRC(); (* Calculate CRC *)
295
296             (* Gestione errore CRC frame ricevuto. *)
297
298             IF (GetCRC.Crc <> 0) THEN ErrorNr:=10069380; RETURN; END_IF;
299
300             (* Elimino da lunghezza frame CRC. *)
301
302             IDx[1]:=IDx[1]-2; (* Data frame index *)
303             CaseNr[1]:=10; (* Case programma *)
304
305             (* ------------------------------------------------------------ *)
306             (* TRASMISSIONE FRAME ASCII                                     *)
307             (* ------------------------------------------------------------ *)
308             (* Controllo se spazio in buffer di trasmissione. Per ogni byte *)
309             (* RTU ricevuto sono trasmessi 2 caratteri Ascii, in più vi è il *)
310             (* carattere di start, i 2 caratteri di LRC ed il <CR><LF>. *)
311
312             10:
313             IF (TO_UINT(SysGetOSpace(FpAscii)) < ((IDx[1]*2)+1+2+2)) THEN EXIT; END_IF;
314
315             (* Calcolo LRC del frame e lo aggiungo al termine dei dati. *)
316
317             Ch:=0; (* Character buffer *)
318             FOR i:=0 TO TO_INT(IDx[1]-1) DO Ch:=Ch+DFRTU[i]; END_FOR;
319             DFRTU[IDx[1]]:=0-Ch; (* LRC value *)
320             IDx[1]:=IDx[1]+1; (* Data frame index *)
321             CaseNr[1]:=CaseNr[1]+1; (* Case programma *)
322
323             (* Trasmetto start frame ":". *)
324
325             IF (Sysfputc(TO_INT(16#3A), FpAscii) = EOF) THEN ErrorNr:=10069400; RETURN; END_IF;
326
```

```
327             (* Loop conversione dato esadecimale in Ascii e trasmissione. *)
328
329             FOR i:=0 TO TO_INT(IDx[1]-1) DO
330
331                 (* Conversione dato esadecimale in Ascii e lo trasmetto. *)
332
333                 Ch:=DFRTU[i]/16; (* Character buffer *)
334                 IF (Ch <= 16#09) THEN Ch:=Ch+16#30; ELSE Ch:=Ch+16#37; END_IF;
335                 IF (Sysfputc(TO_INT(Ch), FpAscii) = EOF) THEN ErrorNr:=10069401; RETURN; END_IF;
336
337                 Ch:=DFRTU[i]&16#0F; (* Character buffer *)
338                 IF (Ch <= 16#09) THEN Ch:=Ch+16#30; ELSE Ch:=Ch+16#37; END_IF;
339                 IF (Sysfputc(TO_INT(Ch), FpAscii) = EOF) THEN ErrorNr:=10069402; RETURN; END_IF;
340             END_FOR;
341
342             (* Eseguo trasmissione <CR> e <LF>. *)
343
344             IF (Sysfputc(TO_INT(16#0D), FpAscii) = EOF) THEN ErrorNr:=10069403; RETURN; END_IF;
345             IF (Sysfputc(TO_INT(16#0A), FpAscii) = EOF) THEN ErrorNr:=10069404; RETURN; END_IF;
346
347             (* Eseguo flush, così su socket la stringa esce immediatamente. *)
348
349             IF NOT(SysFOBfFlush(FpAscii)) THEN ErrorNr:=10069405; RETURN; END_IF;
350             CaseNr[1]:=CaseNr[1]+1; (* Case programma *)
351
352             (* ------------------------------------------------------------ *)
353             (* Eseguo attesa fine trasmissione frame. *)
354
355             11:
356             IF (SysGetOSpace(FpAscii) <> TO_INT(SysGetTxBSize(FpAscii))) THEN EXIT; END_IF;
357             CaseNr[1]:=0; (* Case programma *)
358         ELSE
359             CaseNr[1]:=0; (* Case programma *)
360         END_CASE;
361     END_WHILE;
362
363 (* [End of file] *)
364
365
```