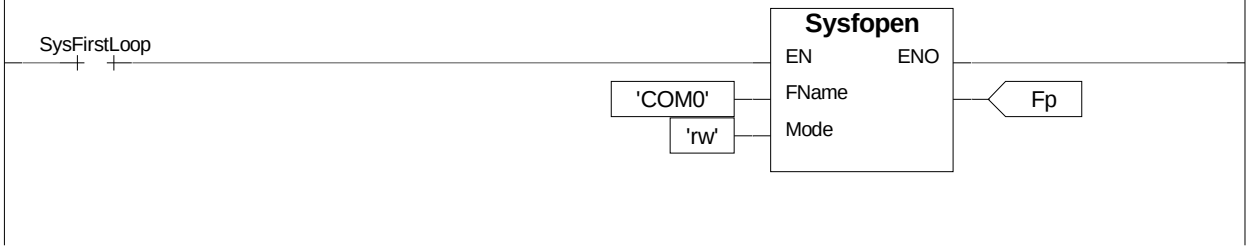
```
VAR_GLOBAL
Di00CPU : BOOL; (* Digital input 00 on CPU *)
Di01CPU : BOOL; (* Digital input 01 on CPU *)
Do00CPU : BOOL; (* Digital output 00 on CPU *)
Do01CPU : BOOL; (* Digital output 01 on CPU *)
CLIPRxd : BOOL; (* CLIP received *)
CLIPNumber : STRING[ 16 ]; (* CLIP number *)
END_VAR
```

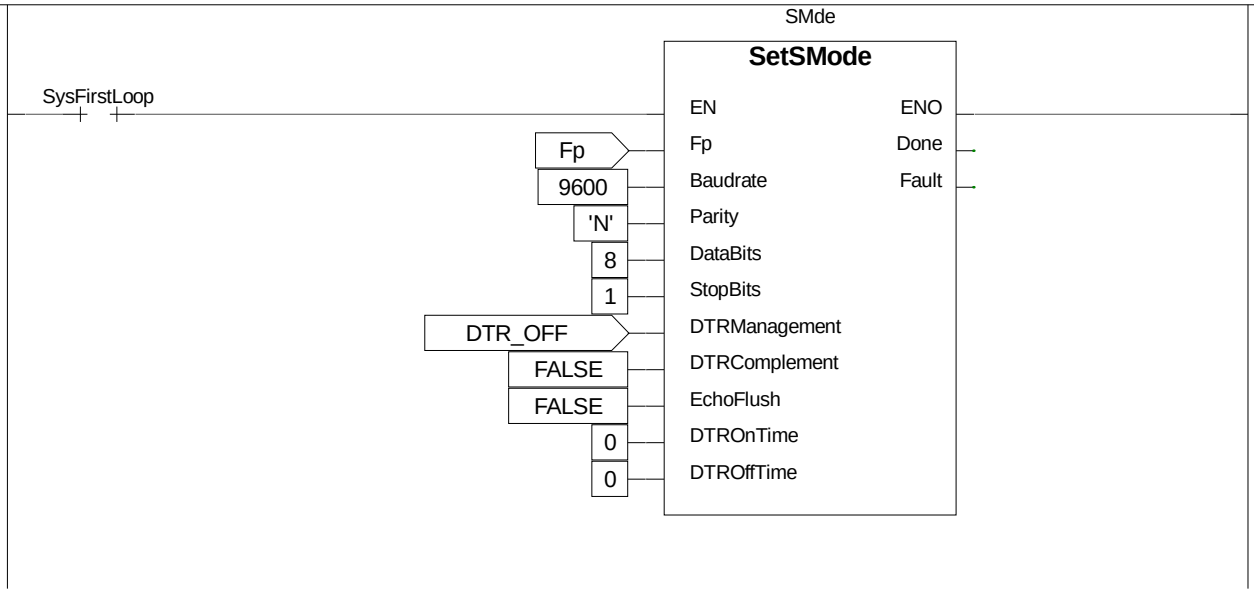
PROGRAM ModemMng

```
VAR
Rssi : USINT; (* Received signal strength indication  *)
ModemID : UDINT; (* Modem ID *)
ModemRx : STRING[ 32 ]; (* Modem Rx string *)
ModemTx : STRING[ 32 ]; (* Modem Tx string *)
Operator : STRING[ 32 ]; (* GSM operator *)
Fp : FILEP; (* File pointer *)
Mdm : ModemCore; (* Modem core FB *)
SMde : SetSMode; (* Set serial mode FB *)
Dummy : BOOL; (* Dummy variable *)
END_VAR
```

1

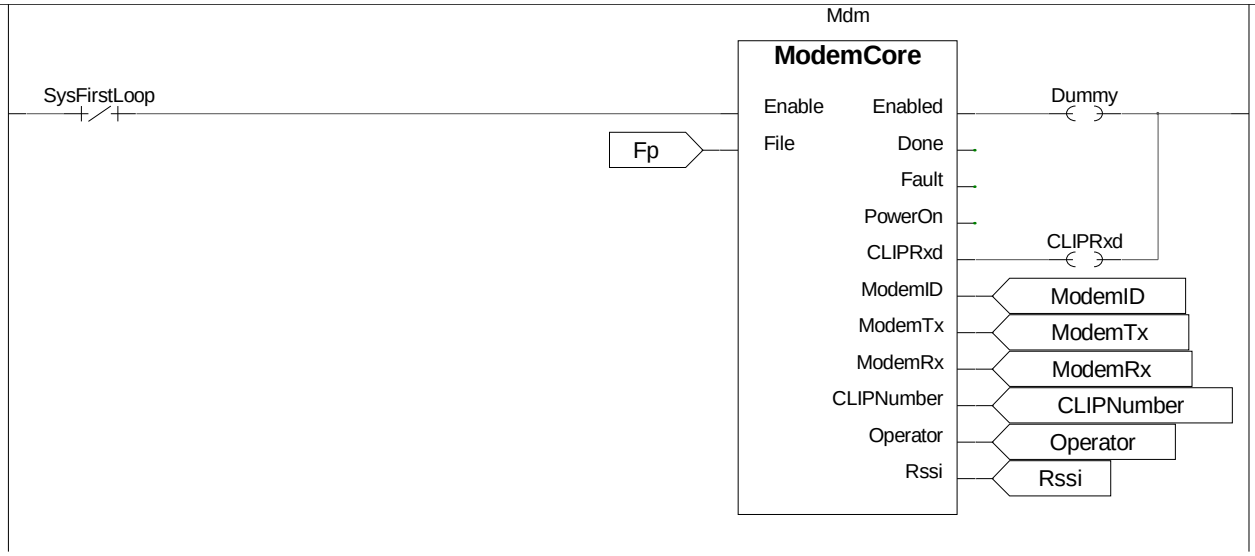


2



	Project : GSMDoorOpen	
	PROGRAM : ModemMng	
	Release : GSMDoorOpe	Ver :1.00
	Author :	Date:18/04/2012
	Note :	Page:1 of 2

3



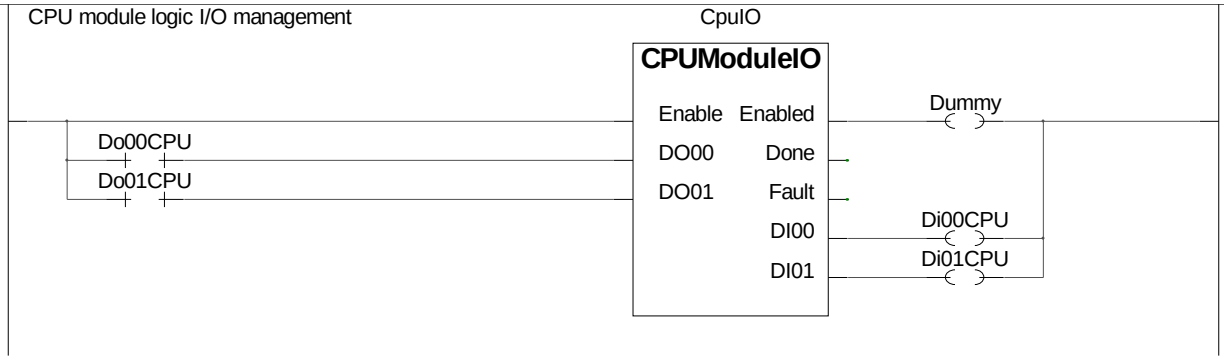
	Project : GSMDoorOpen	
	PROGRAM : ModemMng	
	Release : GSMDoorOpe	Ver :1.00
	Author :	Date:18/04/2012
	Note :	Page:2 of 2

PROGRAM Logic

```
VAR
CpuIO : CPUModuleIO; (* CPU logic I/O management FB *)
Dummy : BOOL; (* Dummy variable *)
CFind : FindCLIPOnFile; (* Find CLIP on file FB *)
SearchOn : BOOL; (* Search CLIP on a file *)
CLIPFound : BOOL; (* CLIP number found in the file *)
DoorNr : USINT; (* Door to open *)
DoorCmdDelay : eTON;
END_VAR
```

1

CPU module logic I/O management



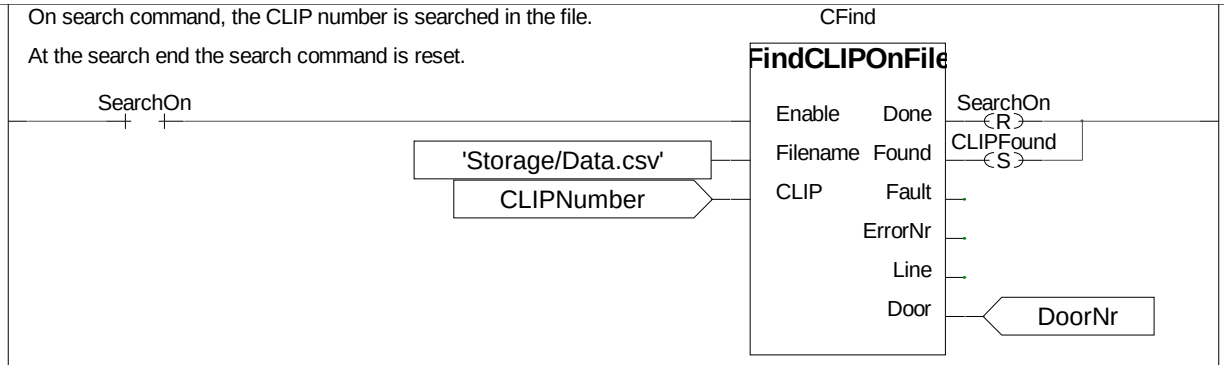
2

On CLIP detection (A call is arrived) the search sequence is started.

3

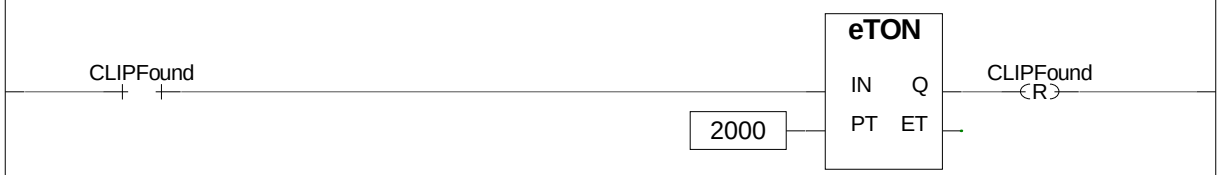
On search command, the CLIP number is searched in the file.

At the search end the search command is reset.



4

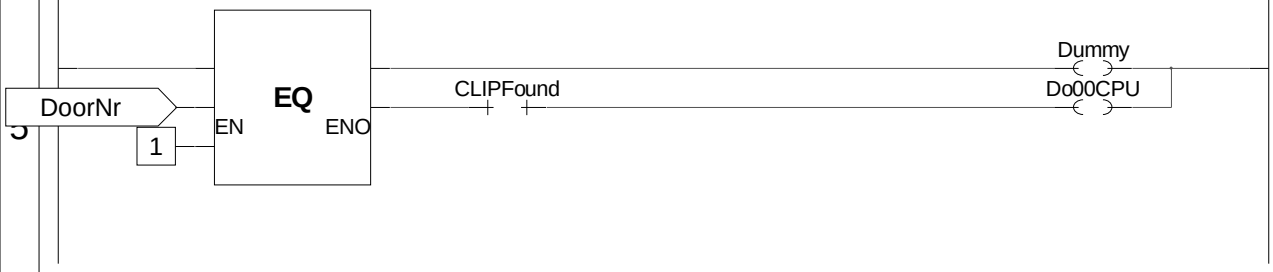
The door open command is delayed.



	Project : GSMDoorOpen	
	PROGRAM : Logic	
	Release : GSMDoorOpe	Ver :1.00
	Author :	Date:18/04/2012
	Note :	Page:1 of 2

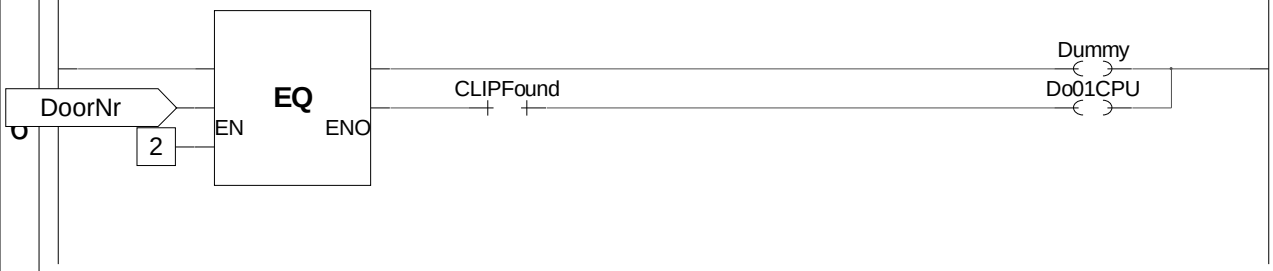
5

Command the logic output to open door Nr 1.



6

Command the logic output to open door Nr 2.



	Project : GSMDoorOpen	
	PROGRAM : Logic	
	Release : GSMDoorOpe	Ver :1.00
	Author :	Date:18/04/2012
	Note :	Page:2 of 2

FUNCTION_BLOCK SetSMode

(SFR054B000) Set the serial communication mode
ENCRYPTED CODE

```
VAR_INPUT
Fp : FILEP;
Baudrate : UDINT; (* Baudrate *)
Parity : STRING[ 1 ]; (* Parity type *)
DataBits : USINT; (* Nr of data bits *)
StopBits : USINT; (* Nr of stop bits *)
DTRManagement : USINT; (* DTR management type *)
DTRComplement : BOOL; (* Complement the DTR signal *)
EchoFlush : BOOL; (* Flush the echo *)
DTROnTime : UINT; (* DTR On wait time *)
DTROffTime : UINT; (* DTR Off wait time *)
END_VAR

VAR_OUTPUT
Done : BOOL; (* Execution done *)
Fault : BOOL; (* Execution fault *)
END_VAR
```

1

	Project : GSMDoorOpen	
	FUNCTION BLOCK : SetSMode	
	Release : GSMDoorOpe	Ver :1.00
	Author :	Date:18/04/2012
	Note :	Page:1 of 1

FUNCTION_BLOCK CPUModuleIO

(SFR054B000) Manages the logic I/O on the CPU module
ENCRYPTED CODE

```
VAR_INPUT
Enable : BOOL; (* Function enable *)
DO00 : BOOL; (* Digital output 0 *)
DO01 : BOOL; (* Digital output 1 *)
END_VAR

VAR_OUTPUT
Enabled : BOOL; (* Function enabled *)
Done : BOOL; (* Function done *)
Fault : BOOL; (* Function fault *)
DI00 : BOOL; (* Digital input 0 *)
DI01 : BOOL; (* Digital input 0 *)
END_VAR
```

1

	Project : GSMDoorOpen	
	FUNCTION BLOCK : CPUModuleIO	
	Release : GSMDoorOpe	Ver :1.00
	Author :	Date:18/04/2012
	Note :	Page:1 of 1

```
VAR_INPUT
Enable : BOOL; (* FB enable *)
Filename : STRING[ 32 ]; (* Path and name of file *)
CLIP : STRING[ 32 ]; (* CLIP to find *)
END_VAR

VAR_OUTPUT
Done : BOOL; (* Execution done *)
Found : BOOL; (* Code found *)
Fault : BOOL; (* Execution fault *)
ErrorNr : USINT; (* Error number *)
Line : UDINT; (* File line where FB ends *)
Door : USINT; (* Door number *)
END_VAR

VAR
Enabled : BOOL; (* FB enabled *)
FPos : UDINT; (* File position *)
FString : STRING[ 64 ]; (* Find string *)
Fp : FILEP; (* File pointer *)
i : INT; (* Auxiliary var *)
j : UDINT; (* Auxiliary var *)
Ptr : @USINT; (* General pointer *)
AString : STRING[ 32 ]; (* Auxiliary string *)
END_VAR
```

```
 1 (* ********************************************************************** *)
 2 (* FUNCTION BLOCK "FindCLIPOnFile"                                        *)
 3 (* ********************************************************************** *)
 4 (* Questo blocco funzione esegue ricerca in un file CSV del CLIP indicato.    *)
 5 (* Viene ritornato il numero di riga in cui il CLIP è stato trovato e il     *)
 6 (* numero di porta che può aprire.                                        *)
 7 (*                                                                        *)
 8 (* Il file CSV deve essere del tipo:                                      *)
 9 (* CLIP;       Door<CR><LF>                                               *)
10 (* +391234567; 1<CR><LF>                                                 *)
11 (* +395678907; 2<CR><LF>                                                 *)
12 (* +395671234; 1<CR><LF>                                                 *)
13 (*                                                                        *)
14 (* I parametri in ingresso sono:                                          *)
15 (* Enable: Attivandolo viene eseguita la ricerca del codice.              *)
16 (* Filename: Percorso e nome del file in cui eseguire la ricerca.         *)
17 (* CLIP: CLIP da ricercare.                                               *)
18 (*                                                                        *)
19 (* I parametri in uscita sono:                                            *)
20 (* Done: Ricerca terminata.                                               *)
21 (* Found: CLIP trovato (One shot).                                        *)
22 (* Fault: Errore nella ricerca (One shot).                                *)
23 (* ErrorNr: Numero errore riscontrato.                                    *)
24 (* Line: Linea del file in cui termina controllo.                         *)
25 (* Door: Numero di porta abilitata.                                       *)
26 (* ---------------------------------------------------------------------- *)
27
28     (* ---------------------------------------------------------------- *)
29     (* ABILITAZIONE                                                     *)
30     (* ---------------------------------------------------------------- *)
31     (* Eseguo controllo se FB abilitata. *)
```

```
32
33      IF NOT(Enable) THEN Enabled:=FALSE; Done:=FALSE; Found:=FALSE; Fault:=FALSE; RETURN; END_IF;
34
35      (* Eseguo inizializzazioni su abilitazione. *)
36
37      IF NOT(Enabled) THEN
38          Enabled:=TRUE; (* FB enabled *)
39          Line:=0; (* File line where FB ends *)
40          ErrorNr:=0; (* Error number *)
41          Door:=0; (* Door number *)
42          FPos:=0; (* File position *)
43      END_IF;
44
45      (* Se attivo Done esco, occorre disabilitare e poi riabilitare l'FB per *)
46      (* eseguire una nuova ricerca. *)
47
48      IF (Done) THEN RETURN; END_IF;
49
50      (* ------------------------------------------------------------------- *)
51      (* INIZIALIZZAZIONI                                                    *)
52      (* ------------------------------------------------------------------- *)
53      (* Apertura file in read. Se non esiste errore, il file rimane aperto *)
54      (* solo durante l'esecuzione della FB, uscendo viene sempre chiuso. *)
55
56      Fp:=Sysfopen(Filename, 'r'); (* File pointer *)
57      IF (Fp = NULL) THEN
58          Fault:=TRUE; (* Execution fault *)
59          Done:=TRUE; (* Execution done *)
60          ErrorNr:=10; (* Error number *)
61          RETURN;
62      END_IF;
63
64      (* ------------------------------------------------------------------- *)
65      (* ESEGUO LETTURA RIGA DA FILE                                         *)
66      (* ------------------------------------------------------------------- *)
67      (* Imposto posizione in cui leggere nel file. *)
68
69      IF (Sysfseek(Fp, TO_DINT(FPos), ID_SEEK_SET) = EOF) THEN
70          Fault:=TRUE; (* Execution fault *)
71          Done:=TRUE; (* Execution done *)
72          ErrorNr:=20; (* Error number *)
73          i:=Sysfclose(Fp); (* Eseguo chiusura file *)
74          RETURN;
75      END_IF;
76
77      (* Leggo dal file stringa di dimensione sufficente per contenere almeno *)
78      (* una riga, su fine file funzione torna meno caratteri dei richiesti. *)
79
80      i:=Sysfread(ADR(FString), 1, 64, Fp); (* Received characters *)
81      j:=Sysfclose(Fp); (* Eseguo chiusura file *)
82
83      (* Se non sono letti dati il file è terminato. *)
84
85      IF (i = 0) THEN Done:=TRUE; RETURN; END_IF;
86
87      (* Ho letto sicuramente almeno una riga. La riga termina con <CR><LF> *)
88      (* controllo se presenti i terminatori di riga. *)
89
90      j:=FIND(FString, '$r$n');
91
```

```
 92     IF (j = 0) THEN
 93         Fault:=TRUE; (* Execution fault *)
 94         Done:=TRUE; (* Execution done *)
 95         ErrorNr:=30; (* Error number *)
 96         RETURN;
 97     END_IF;
 98
 99     (* Inserisco codice tappo dopo ultimo carattere di riga, sul <CR>. *)
100
101     Ptr:=ADR(FString)+j; (* General pointer *)
102     @Ptr:=0; (* Inserisco codice tappo *)
103
104     (* Posiziono file su inizio riga seguente, +1 per puntare dopo <CR><LF>. *)
105
106     FPos:=FPos+j+1; (* File position *)
107     Line:=Line+1; (* File line where FB ends *)
108
109     (* ---------------------------------------------------------------------- *)
110     (* ESEGUO CONTROLLO CLIP                                                  *)
111     (* ---------------------------------------------------------------------- *)
112     (* Eseguo acquisizione CLIP. Se funzione non trova stringa di CLIP non *)
113     (* segnalo errore, per eseguire skip di campi vuoti o non numerici. *)
114
115     IF NOT(SysVarsscanf(ADR(FString), '%s', STRING_TYPE, ADR(AString))) THEN RETURN; END_IF;
116
117     (* Eseguo verifica se CLIP cercato. *)
118
119     IF (FIND(AString, CLIP) = 0) THEN RETURN; END_IF;
120
121     (* CLIP trovato, ritorno numero porta, cerco separatore. *)
122
123     j:=FIND(FString, ';');
124
125     IF (j = 0) THEN
126         Fault:=TRUE; (* Execution fault *)
127         Done:=TRUE; (* Execution done *)
128         ErrorNr:=40; (* Error number *)
129         RETURN;
130     END_IF;
131
132     (* Copio numero porta su variabile di ritorno. *)
133
134     IF NOT(SysVarsscanf(ADR(FString)+j, '%d', USINT_TYPE, ADR(Door))) THEN
135         Fault:=TRUE; (* Execution fault *)
136         Done:=TRUE; (* Execution done *)
137         ErrorNr:=50; (* Error number *)
138         RETURN;
139     END_IF;
140
141     (* Trovata voce in file per comandare l'apertura porta. *)
142
143     Found:=TRUE; (* Code found *)
144     Done:=TRUE; (* Execution done *)
145
146 (* [End of file] *)
147
148
```

FUNCTION_BLOCK eTON

(SFR053A500) On delay timer
ENCRYPTED CODE

```
VAR_INPUT
IN : BOOL; (* Timer input *)
PT : UDINT; (* Preset time value (mS) *)
END_VAR

VAR_OUTPUT
Q : BOOL; (* Delayed output *)
ET : UDINT; (* Executing time (mS) *)
END_VAR

VAR_EXTERNAL
SysTime : UDINT; (* System time (mS) *)
END_VAR
```

1

	Project : GSMDoorOpen	
	FUNCTION BLOCK : eTON	
	Release : GSMDoorOpe	Ver :1.00
	Author :	Date:18/04/2012
	Note :	Page:1 of 1