



# **PROFIBUS PROJEKTIERUNG IN S7**

## **KEYPILOT VERSION PROFIBUS DP (AL)**

Stand 05/2009  
Firmware Version 2.3

**SysDesign GmbH**  
Säntisstrasse 25  
D-88079 Kressbronn

Telefon: +49 7543 9620-0  
Telefax: +49 7543 9620-22  
Internet: [www.SysDesign.info](http://www.SysDesign.info)

## Inhaltsverzeichnis:

<b>1. EINLEITUNG .....</b>	<b>3</b>
1.1 Zielbestimmung .....	3
1.2 Beispielkonfiguration .....	4
<b>2. BEISPIELE.....</b>	<b>5</b>
2.1 Step-by-Step Projektierung .....	5
2.1.1 Hardware Konfiguration .....	5
2.1.2 Schlüsseldaten auslesen.....	9
2.1.3 LED ansteuern.....	11
2.1.4 Diagnose.....	12
2.2 Einfaches Lesen und Schreiben.....	13
2.2.1 Besonderheiten .....	13
2.2.2 Key-ID Einlesen .....	13
2.2.3 LED Steuern.....	13
2.3 Einfacher Schlüsselabgleich.....	14
2.3.1 Besonderheiten .....	14
2.3.2 UDT1 „KPID“ .....	14
2.3.3 DB1 „KPLIST“ .....	15
2.3.4 FC1 „KPCHECK“ .....	16
2.3.5 Funktionsaufruf.....	19
2.4 Zuordnung von Rechten und Nutzerdaten .....	20
2.4.1 Besonderheiten .....	20
2.4.2 UDT2 „KPUSER“ .....	20
2.4.3 DB2 „KPUSRLST“ .....	21
2.4.4 FB1 „KPCHKUSR“.....	22
2.4.5 Funktionsaufruf.....	28
<b>3. GSD-DATEI .....</b>	<b>29</b>
3.1 Download.....	29
3.2 Siemens HW-Konfig Katalog.....	29
3.3 GSD für ältere Firmwarestände .....	29
3.4 Listing .....	30

## 1. Einleitung

### 1.1 Zielbestimmung

Dieses Dokument soll zusätzliche Informationen für die Projektierung des KeyPilot PDP bzw. KeyPilot PDP AL mit einer SIMATIC S7 Steuerung liefern. Es stellt Beispiele für die Integration des KeyPilot in ein S7-Programm vor. Grundlegende Kenntnisse der Projektierung von S7-Systemen werden vorausgesetzt.

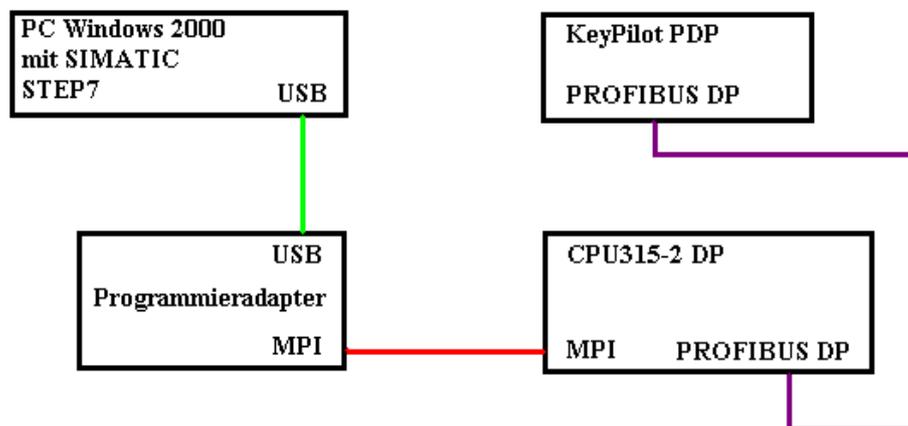
Das Dokument soll die Angaben der KeyPilot-Bedienungsanleitung ergänzen, kann diese jedoch keinesfalls ersetzen. Die Bedienungsanleitung muss in jedem Fall beachtet werden.

## 1.2 Beispielkonfiguration

Für die nachfolgenden Beispiele wurden folgende Komponenten verwendet:

- SIMATIC S7-300 SPS, CPU-Modul CPU315-2 DP 315-2AF03-0AB0 V1.2
- SIMATIC S7 PC Adapter USB6ES7972-0CB20-0XA0
- KeyPilot PDP Slave
- Netzteil PS307 5A DC24V
- Software SIMATIC STEP 7 V5.3 mit Service Pack 2

Systemaufbau der Beispielkonfiguration:



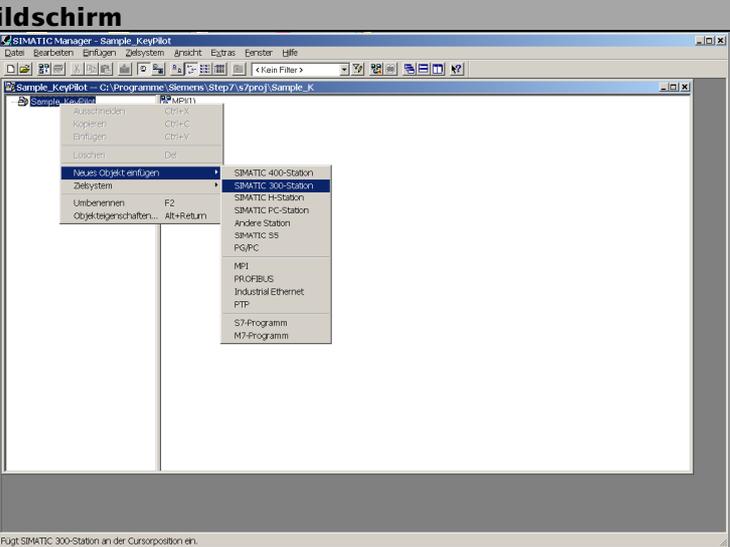
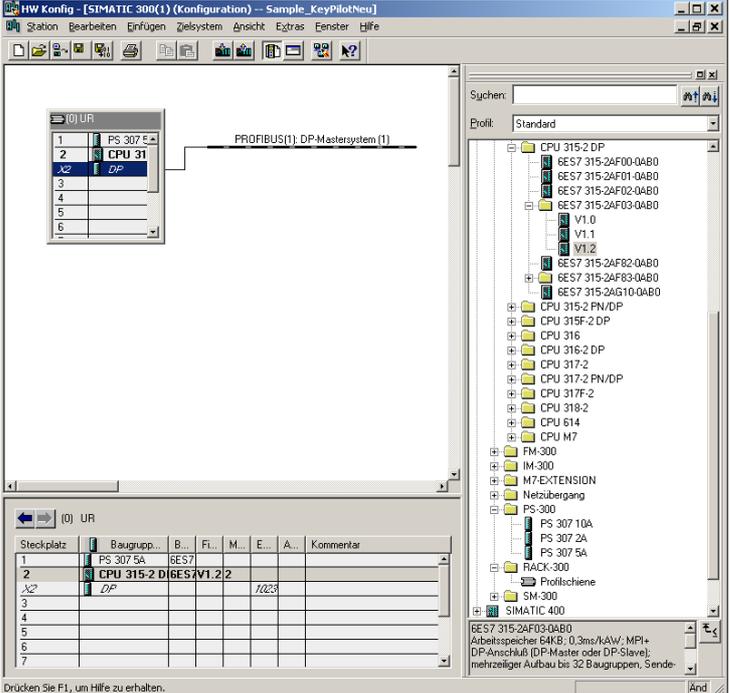
## 2. Beispiele

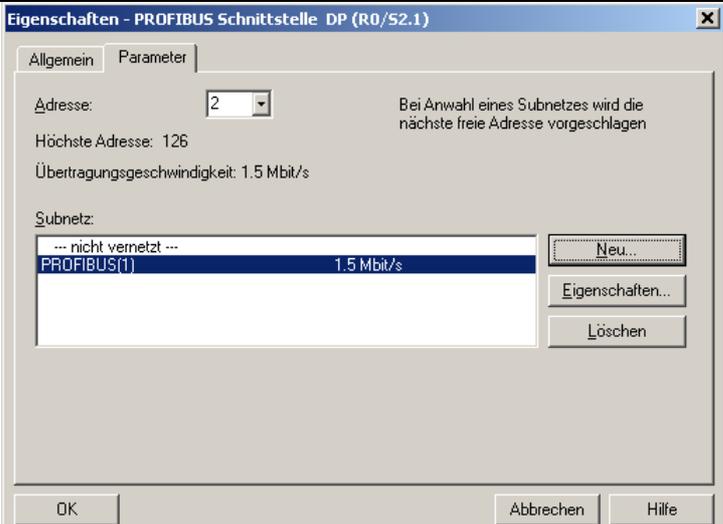
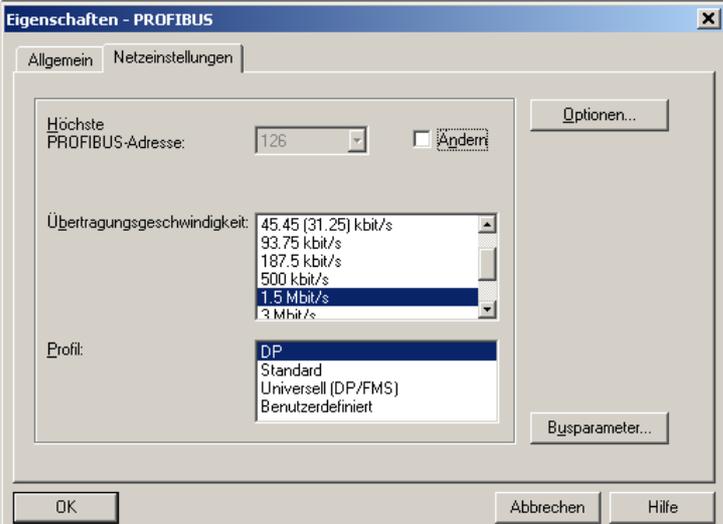
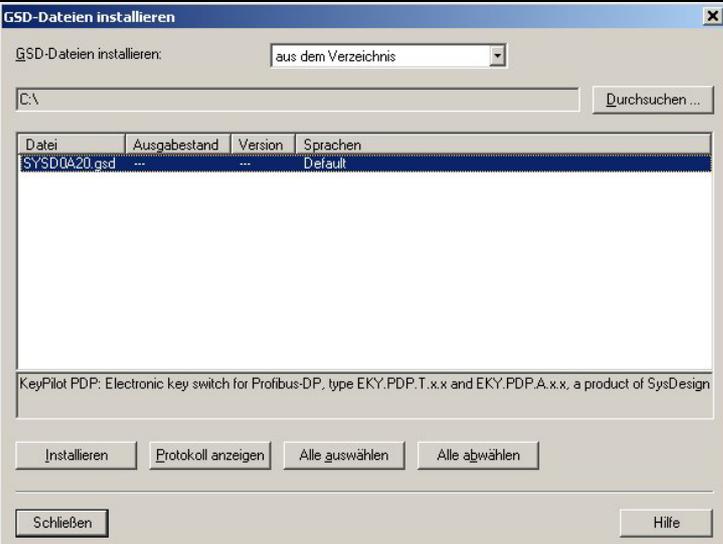
### 2.1 Step-by-Step Projektierung

Dieses Beispiel zeigt schrittweise, anhand der Beispielkonfiguration, die Erstellung eines einfachen S7-Projekts. Das Programm liest die Schlüsselkennung des aktuell am KeyPilot angelegten Schlüssels aus und stellt Möglichkeiten zur Ansteuerung der LED des KeyPilot bereit.

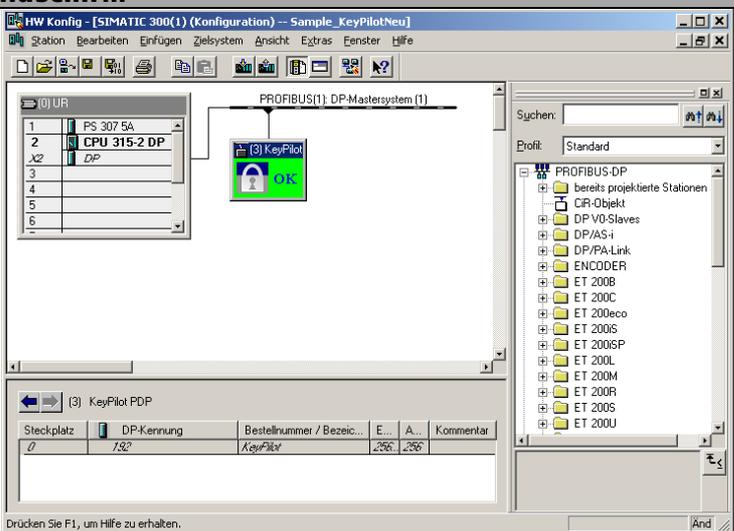
Dieses Beispiel ist auch in der KeyPilot-Bedienungsanleitung enthalten.

#### 2.1.1 Hardware Konfiguration

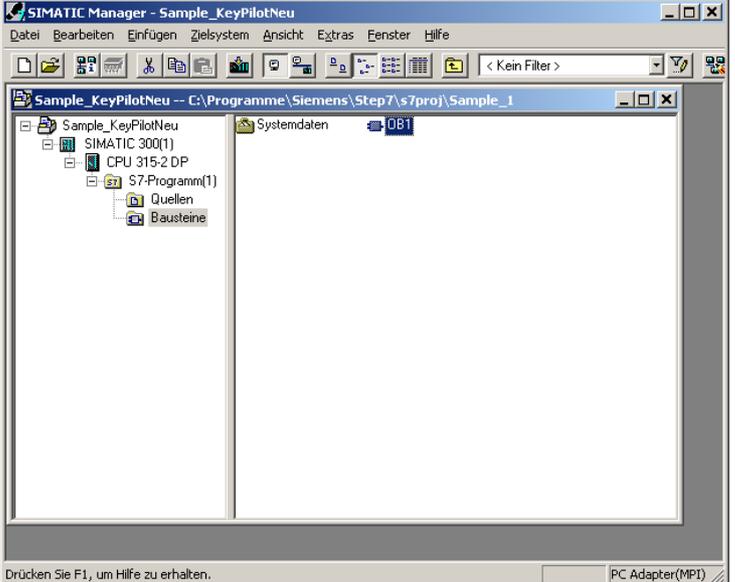
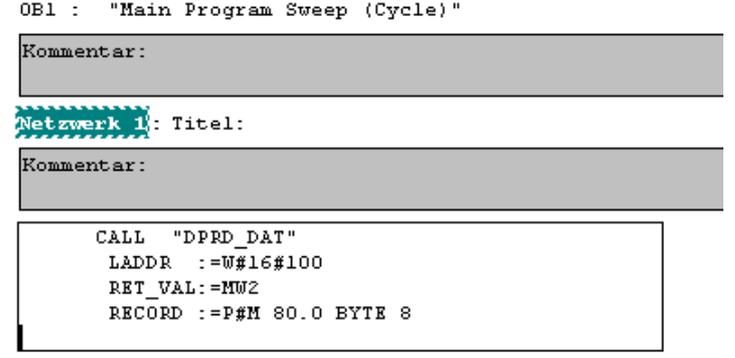
Schritt	Aktion	Bildschirm
1.	Im SIMATIC Manager ein neues Projekt anlegen, eine neue Station einfügen und diese öffnen.	 <p>Fügt SIMATIC 300-Station in der Cursorposition ein.</p>
2.	In der Hardware Konfiguration folgende Komponenten einfügen: Profilschiene, Netzteil PS 307 (nicht zwingend notwendig) und CPU 315-2 DP Beim Einfügen der CPU öffnet sich der Dialog zur Einstellung des PROFIBUS Netzes (siehe nächster Schritt).	 <p>Drücken Sie F1, um Hilfe zu erhalten.</p>

Schritt	Aktion	Bildschirm
3.	Neues Subnetz anlegen, Baudrate und Busadresse einstellen.	 
4.	Um den KeyPilot in der Hardware Konfiguration einfügen zu können, muss die Datei „SYSD0A20.gsd“ einmalig installiert werden. Dazu über „Extras → GSD-Dateien installieren...“ nebenstehenden Dialog aufrufen und die Datei auswählen.	

Schritt	Aktion	Bildschirm
5.	Den KeyPilot Slave in das PROFIBUS Subnetz einfügen und die Slaveadresse festlegen.	<p>The screenshot shows a project tree for PROFIBUS-DP. Under 'bereits projektierte Stationen', there are numerous device types listed, including ET 200 series, SIMATIC, SIMODRIVE, and SIMOVERT. Below the tree, the 'Eigenschaften - PROFIBUS Schnittstelle KeyPilot PDP' dialog box is open. It has two tabs: 'Allgemein' and 'Parameter'. In the 'Allgemein' tab, the 'Adresse' field is set to 3. The 'Übertragungsgeschwindigkeit' is 1.5 Mbit/s. The 'Subnetz' dropdown menu is open, showing 'PROFIBUS(1)' selected. Other buttons like 'Neu...', 'Eigenschaften...', and 'Löschen' are visible. At the bottom of the dialog are 'OK', 'Abbrechen', and 'Hilfe' buttons.</p>

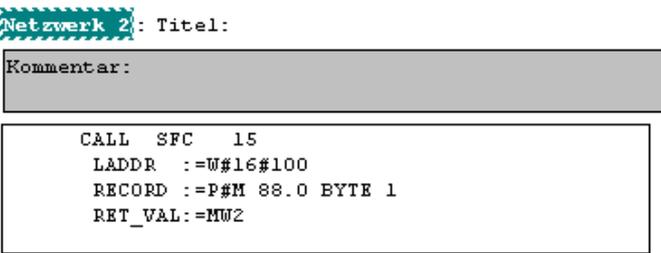
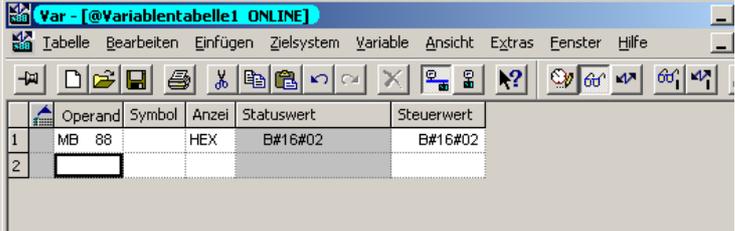
Schritt	Aktion	Bildschirm
6.	Die Konfiguration sollte nun wie folgt aussehen.	
7.	Bei Bedarf weitere Komponenten in der Hardwarekonfiguration einfügen und gegebenenfalls konfigurieren. Danach die Hardware Konfiguration speichern und übersetzen.	

## 2.1.2 Schlüsseldaten auslesen

Schritt	Aktion	Bildschirm
8.	Wechseln Sie wieder in den SIMATIC Manager und öffnen Sie die Ansicht „Bausteine“. Doppelklick auf OB1 öffnet den Dialog „Eigenschaften – Organisationsbaustein“. Diesen mit „OK“ bestätigen. Es öffnet sich die Programmierumgebung „KOP/AWL/FUP“.	 <p>The screenshot shows the SIMATIC Manager interface. The project tree on the left is expanded to show 'OB1' under 'S7-Programm(1)'. The main workspace is currently empty.</p>
9.	„CALL SFC 14“ eintippen (wird in „DPRD_DAT“ umgewandelt). Es erscheinen die Parameter des Bausteinaufrufs. Diese wie gezeigt belegen. Speichern Sie den Baustein.	 <p>The screenshot shows the 'CALL SFC 14' parameter dialog box. The title is 'OB1 : "Main Program Sweep (Cycle)".' There are two 'Kommentar:' fields. The 'Netzwerk 1' field is highlighted in blue and contains the text 'Titel:'. Below the dialog, the generated code is shown:</p> <pre>CALL "DPRD_DAT" LADDR :=W#16#100 RET_VAL:=MW2 RECORD :=P#M 80.0 BYTE 8</pre>
10.	Der SFC 14 dient zum Einlesen der Daten eines DP-Slaves. Der Bezeichner LADDR in obigem Code gibt die Adresse der Eingangsdaten an, an der die Daten des Slaves gelesen werden können (z.B. bedeutet W#16#100 → wortweise von Adresse 256 (100h) lesen). Von welcher Adresse die Daten des Slaves gelesen werden können, ist in der Hardware Konfiguration sichtbar und kann dort auch eingestellt werden. Die Zeile P#M 80.0 BYTE 8 setzt einen Zeiger auf das Merkerbyte 80 und gibt an, dass 8 Byte abgelegt werden sollen.	
11.	Den Drehschalter an der CPU in die Stellung RUN-P bringen. Der Programmieradapter muss auf der MPI-Schnittstelle eingesteckt und richtig konfiguriert sein. Im SIMATIC Manager das Objekt „SIMATIC 300(1)“ markieren und den Button „Laden“ betätigen. Die erscheinenden Dialoge mit „OK“ beantworten und die Frage „Soll die Baugruppe jetzt gestartet werden (Neustart) ?“ mit „Ja“ beantworten. Nach dem Neustart der Baugruppe sollten nur die LED´s DC5V und RUN grün leuchten. Die LED´s SF und BUSF sollten aus sein. Ebenso sollte die rote LED am KeyPilot erloschen sein.	

Schritt	Aktion	Bildschirm																																																							
12.	<p>Im SIMATIC Manager können über den Menüpunkt „Zielsystem → Variable beobachten / steuern,“ die Eingangsdaten des KeyPilot PDP (Key ID) beobachtet werden. In der Variablen-tabelle muss dazu ein Bezug auf die im OB1 verwendeten Merkerbytes hergestellt werden. Dazu sind in der Spalte Operand die Bezeichner „MB 80“ bis „MB 87“ einzutragen (Menü „Einfügen → Bereich“). Anschließend den Button „Statuswerte aktualisieren“ betätigen.</p>	<table border="1"> <thead> <tr> <th></th> <th>Operand</th> <th>Symbol</th> <th>Anzei</th> <th>Statuswert</th> </tr> </thead> <tbody> <tr> <td>1</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>2</td> <td>MB 80</td> <td></td> <td>HEX</td> <td>B#16#01</td> </tr> <tr> <td>3</td> <td>MB 81</td> <td></td> <td>HEX</td> <td>B#16#54</td> </tr> <tr> <td>4</td> <td>MB 82</td> <td></td> <td>HEX</td> <td>B#16#5D</td> </tr> <tr> <td>5</td> <td>MB 83</td> <td></td> <td>HEX</td> <td>B#16#48</td> </tr> <tr> <td>6</td> <td>MB 84</td> <td></td> <td>HEX</td> <td>B#16#0D</td> </tr> <tr> <td>7</td> <td>MB 85</td> <td></td> <td>HEX</td> <td>B#16#00</td> </tr> <tr> <td>8</td> <td>MB 86</td> <td></td> <td>HEX</td> <td>B#16#00</td> </tr> <tr> <td>9</td> <td>MB 87</td> <td></td> <td>HEX</td> <td>B#16#5A</td> </tr> <tr> <td>10</td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>		Operand	Symbol	Anzei	Statuswert	1					2	MB 80		HEX	B#16#01	3	MB 81		HEX	B#16#54	4	MB 82		HEX	B#16#5D	5	MB 83		HEX	B#16#48	6	MB 84		HEX	B#16#0D	7	MB 85		HEX	B#16#00	8	MB 86		HEX	B#16#00	9	MB 87		HEX	B#16#5A	10				
	Operand	Symbol	Anzei	Statuswert																																																					
1																																																									
2	MB 80		HEX	B#16#01																																																					
3	MB 81		HEX	B#16#54																																																					
4	MB 82		HEX	B#16#5D																																																					
5	MB 83		HEX	B#16#48																																																					
6	MB 84		HEX	B#16#0D																																																					
7	MB 85		HEX	B#16#00																																																					
8	MB 86		HEX	B#16#00																																																					
9	MB 87		HEX	B#16#5A																																																					
10																																																									
13.	<p>Das Merkerbyte 80 zeigt den Family Code des anliegenden Schlüssels. Merkerbyte 81 bis 86 zeigen die eigentliche Key-ID. Merkerbyte 87 zeigt einen 8 Bit CRC über die vorherigen Bytes.</p>																																																								

## 2.1.3 LED ansteuern

Schritt	Aktion	Bildschirm																		
14.	Im OB1 ein weiteres Netzwerk einfügen, „CALL SFC 15“ eintippen und den erscheinenden Bausteinaufruf wie gezeigt belegen. OB1 speichern und in die CPU laden.	 <pre> CALL SFC 15 LADDR :=W#16#100 RECORD :=P#M 88.0 BYTE 1 RET_VAL:=MW2 </pre>																		
15.	Der SFC 15 dient zum Senden von Daten an einen DP-Slave. Die Bedeutung von LADDR ist wie beim SFC 14 (siehe Schritt 10) mit dem Unterschied, dass nun die Ausgangsdaten adressiert werden. Die Zeile P#M 88.0 BYTE 1 setzt einen Zeiger auf das Merkerbyte 88 und gibt somit an, dass die zu sendenden Daten diesem Byte entnommen werden sollen.																			
16.	Über „Variable beobachten / steuern“ kann nun die LED am KeyPilot aktiviert werden. Dazu das Merkerbyte 88 eintragen und in der Spalte „Steuerwert“ einen Wert von 0 bis 2 eingeben (B#16#0 wird automatisch ergänzt). Nach Betätigung des Buttons „Steuerwerte aktivieren“ wird der eingegebene Wert an den KeyPilot gesendet. Bedeutung der Werte: 0: LED aus 1: LED rot ein 2: LED grün ein	 <table border="1"> <thead> <tr> <th></th> <th>Operand</th> <th>Symbol</th> <th>Anzei</th> <th>Statuswert</th> <th>Steuerwert</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>MB 88</td> <td></td> <td>HEX</td> <td>B#16#02</td> <td>B#16#02</td> </tr> <tr> <td>2</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>		Operand	Symbol	Anzei	Statuswert	Steuerwert	1	MB 88		HEX	B#16#02	B#16#02	2					
	Operand	Symbol	Anzei	Statuswert	Steuerwert															
1	MB 88		HEX	B#16#02	B#16#02															
2																				

## 2.1.4 Diagnose

Schritt	Aktion	Bildschirm
17.	In die Hardware Konfiguration wechseln und den Button „Offline <-> Online“ drücken. Mit einem Doppelklick auf den Slave kann die Slave Diagnose eingesehen werden	<p>The image shows two screenshots of the KeyPilot software interface. The top screenshot displays the 'DP-Slave Diagnose' window. It includes fields for 'Bezeichnung' (KeyPilot PDP), 'Name' (KeyPilot PDP), 'Systemkennung' (PROFIBUS DP), and 'Version'. A table lists components with columns for 'Bestell-Nr./ Bezeichn.', 'Komponente', and 'Ausgabestand'. Below this, 'DP-Mastersystem' is set to 1 and 'Station' to 3, with 'Adresse' set to E 1022. The status is 'Baugruppe vorhanden und o.k.'. The bottom screenshot shows the same window with a 'Diagnose im Hexadezimalformat' dialog box open, displaying the hex value '0000 : 00 0C 00 02 09 85'.</p>

## 2.2 Einfaches Lesen und Schreiben

### 2.2.1 Besonderheiten

Das vorangegangene Beispiel verwendet für die Ansteuerung der LED die Systemfunktion SFC15 ("DPWR\_DAT"). Diese dient üblicherweise zur konsistenten Übertragung von Datenblöcken, die mehr als vier Byte einnehmen. Blöcke bis einschließlich vier Byte werden bei Profibus grundsätzlich konsistent übertragen.

Da für die Steuerung des KeyPilot nur ein Byte verwendet wird (und von diesem auch nur die beiden niederwertigsten Bits), ist der Einsatz von SFC15 nicht zwingend notwendig. Allerdings wurde der KeyPilot im vorangegangenen Beispiel mit der Standard-Ausgangsadresse 256 konfiguriert, so dass die Steuerausgänge außerhalb des Prozessabbilds der S7-315 liegen. Werden wie weiter unten gezeigt E/A-Adressen verwendet, die vollständig innerhalb des Prozessabbilds der jeweiligen CPU liegen, dann kann die KeyPilot-LED über klassische Ausgangs-Zuweisungen angesteuert werden.

Beim Einlesen der Schlüsselkennung vom KeyPilot empfiehlt sich dagegen aber grundsätzlich die Verwendung der Systemfunktion SFC14 ("DPRD\_DAT"), da hierbei acht Bytes übertragen werden und es somit zu Inkonsistenzen kommen kann.

### 2.2.2 Key-ID Einlesen

Folgendes Netzwerk zeigt beispielhaft den Aufruf der Systemfunktion SFC14. Als Eingangsadresse des KeyPilot wurde 10 gewählt. Die Schlüsselkennung wird in die Merkerbyte MB80 bis MB87 geschrieben.

```
NETWORK
TITLE =read actual key-ID

CALL SFC 14 (
  LADDR          := W#16#A,                //input address 10
  RET_VAL        := MW 2,                  //dummy
  RECORD         := P#M 10.0 BYTE 8);     //buffer for Key-ID
```

### 2.2.3 LED Steuern

Hier wird die Farbe der LED über die Merker M18.0 und M18.1 angefordert. Ausgangsadresse für den KeyPilot ist wieder 10. Somit wird über A10.0 der Zustand „rot“ angefordert und über A10.1 der Zustand „grün“.

```
NETWORK
TITLE =LED control

U   M   18.0;
=   A   10.0; //LED red

U   M   18.1;
=   A   10.1; //LED green
```

## 2.3 Einfacher Schlüsselabgleich

### 2.3.1 Besonderheiten

In diesem Beispiel wird eine Liste mit Schlüsselkennungen nach der Kennung des aktuell am KeyPilot aufgelegten Schlüssels durchsucht. Wenn ein Schlüssel aufgelegt ist, dessen Kennung nicht in der Liste vorkommt, wird die LED auf rot geschaltet. Ist die Kennung in der Liste vorhanden, dann schaltet die LED auf grün.

### 2.3.2 UDT1 „KPID“

Dieser UDT stellt eine Struktur für eine Schlüsselkennung bereit.

```
TYPE UDT 1
AUTHOR : RAmann
FAMILY : KP
NAME : KPID
VERSION : 1.0

STRUCT
  B_FC : BYTE ; //family code
  B_ID1 : BYTE ; //ID byte 1
  B_ID2 : BYTE ; //ID byte 2
  B_ID3 : BYTE ; //ID byte 3
  B_ID4 : BYTE ; //ID byte 4
  B_ID5 : BYTE ; //ID byte 5
  B_ID6 : BYTE ; //ID byte 6
  B_CRC : BYTE ; //CRC byte
END_STRUCT ;
END_TYPE
```

### 2.3.3 DB1 „KPLIST“

Im DB1 ist die zu durchsuchenden Liste abgelegt. Die Liste besteht aus einem Array von Schlüsselkennungen (Typ UDT1, siehe oben).

Im Beispiel enthält die Liste vier Plätze, von denen ein Platz mit dem Schlüssel 0100000CBF2FD5 belegt ist.

```

DATA_BLOCK DB 1
TITLE =
AUTHOR : RAmann
FAMILY : KP
NAME : KPLIST
VERSION : 1.0

STRUCT
  AUDT_KeyList : ARRAY [1 .. 4 ] OF //list of known keys (insert your data here)
  UDT 1;
END_STRUCT ;
BEGIN
  AUDT_KeyList[1].B_FC := B#16#1;
  AUDT_KeyList[1].B_ID1 := B#16#D5;
  AUDT_KeyList[1].B_ID2 := B#16#2F;
  AUDT_KeyList[1].B_ID3 := B#16#BF;
  AUDT_KeyList[1].B_ID4 := B#16#C;
  AUDT_KeyList[1].B_ID5 := B#16#0;
  AUDT_KeyList[1].B_ID6 := B#16#0;
  AUDT_KeyList[1].B_CRC := B#16#35;
  AUDT_KeyList[2].B_FC := B#16#0;
  AUDT_KeyList[2].B_ID1 := B#16#0;
  AUDT_KeyList[2].B_ID2 := B#16#0;
  AUDT_KeyList[2].B_ID3 := B#16#0;
  AUDT_KeyList[2].B_ID4 := B#16#0;
  AUDT_KeyList[2].B_ID5 := B#16#0;
  AUDT_KeyList[2].B_ID6 := B#16#0;
  AUDT_KeyList[2].B_CRC := B#16#0;
  AUDT_KeyList[3].B_FC := B#16#0;
  AUDT_KeyList[3].B_ID1 := B#16#0;
  AUDT_KeyList[3].B_ID2 := B#16#0;
  AUDT_KeyList[3].B_ID3 := B#16#0;
  AUDT_KeyList[3].B_ID4 := B#16#0;
  AUDT_KeyList[3].B_ID5 := B#16#0;
  AUDT_KeyList[3].B_ID6 := B#16#0;
  AUDT_KeyList[3].B_CRC := B#16#0;
  AUDT_KeyList[4].B_FC := B#16#0;
  AUDT_KeyList[4].B_ID1 := B#16#0;
  AUDT_KeyList[4].B_ID2 := B#16#0;
  AUDT_KeyList[4].B_ID3 := B#16#0;
  AUDT_KeyList[4].B_ID4 := B#16#0;
  AUDT_KeyList[4].B_ID5 := B#16#0;
  AUDT_KeyList[4].B_ID6 := B#16#0;
  AUDT_KeyList[4].B_CRC := B#16#0;
END_DATA_BLOCK

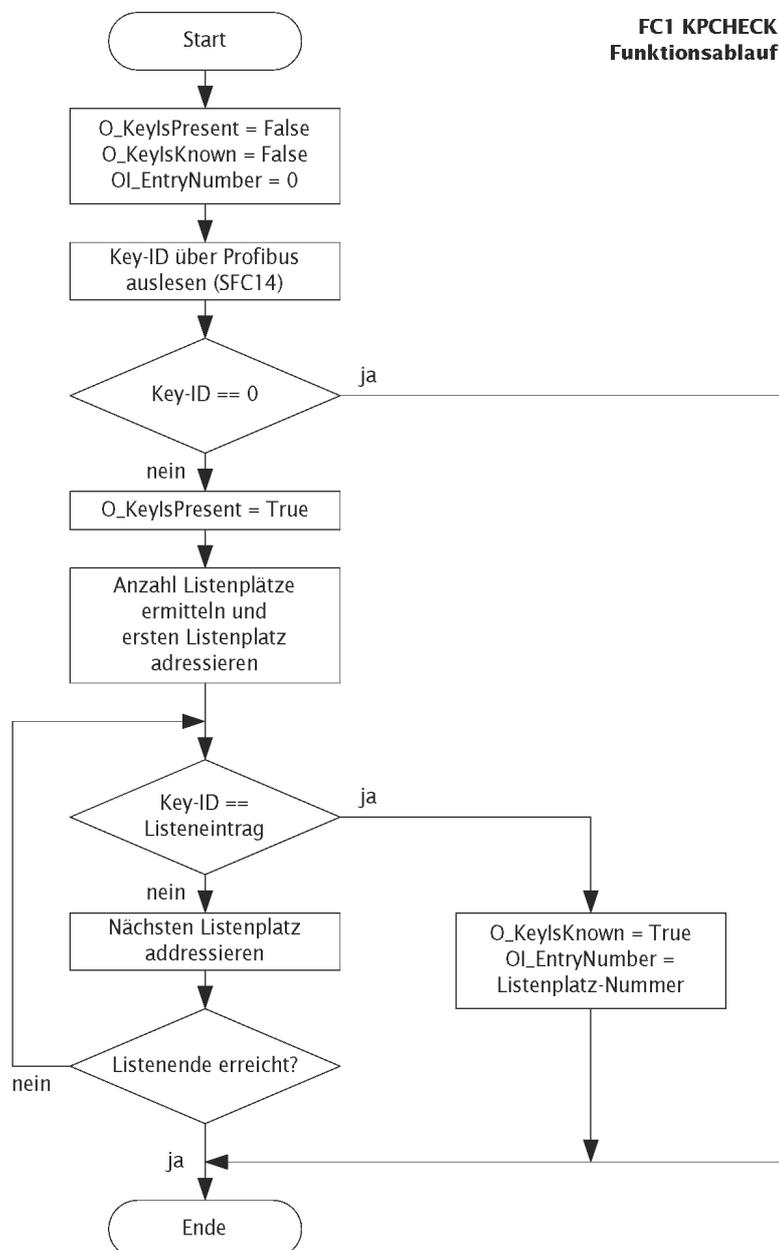
```

## 2.3.4 FC1 „KPCHECK“

Diese Funktion liest die Daten vom KeyPilot ein und führt den Vergleich mit der zu durchsuchenden Liste durch.

Der Baustein benötigt als Eingangsparameter einen Verweis zur Liste inkl. Längenangabe (IANyr\_KeyList) und die Eingangs-Startadresse des KeyPilot (II\_StartAddressIn). Er liefert folgende Signale:

- Schlüssel ist aufgelegt (O\_KeysPresent)
- Schlüssel ist in der Liste vorhanden (O\_KeysKnown)
- Number des übereinstimmenden Listeneintrags (OI\_EntryNumber)



```

FUNCTION FC 1 : VOID
TITLE =Read Key-ID and find matching database entry
AUTHOR : RAmann
FAMILY : KP
NAME : KP CHECK
VERSION : 1.0

VAR_INPUT
  IANYr_KeyList : ANY ; //pointer on table with known keys and user data [length N*UDT1; read]
  II_StartAddressIn : INT ; //input start address of KeyPilot (according to HW-Config)
END_VAR
VAR_OUTPUT
  OI_EntryNumber : INT ; //number of first matching entry in the table of known keys [0=no
matching entry]
  O_KeyIsPresent : BOOL ; //key is corretly applied to reading head [0=no key/1=key applied]
  O_KeyIsKnown : BOOL ; //key is listed in the table of known keys [0=key not found/1=key found]
END_VAR
VAR_TEMP
  LADW_KeyID : ARRAY [1 .. 2 ] OF //buffer for key-id
  DWORD ;
  LDW_AR1 : DWORD ; //buffer for address register 1
  LI_RetVal : INT ; //state information of SFC14 "DPRD_DAT"
  LI_DataNumber : INT ; //numer of keys in table
  LI_LoopCounter : INT ; //loop counter
  LW_Address : WORD ; //input start address from HW-Config
  LW_DB : WORD ; //number of DB with key table
END_VAR
BEGIN
NETWORK
TITLE =backup of address register

  TAR1 #LDW_AR1;

NETWORK
TITLE =clear out values

  CLR ;
  = #O_KeyIsPresent;
  = #O_KeyIsKnown;

  L 0;
  T #OI_EntryNumber;

NETWORK
TITLE =read key id

  L #II_StartAddressIn; // type cast from INT to WORD
  T #LW_Address;

  CALL SFC 14 (// read data from KeyPilot
  LADDR := #LW_Address,
  RET_VAL := #LI_RetVal,
  RECORD := #LADW_KeyID);

NETWORK
TITLE =key applied?

  O( ;
  L #LADW_KeyID[1];
  L 0;
  <>D ;
  ) ;
  O( ;
  L #LADW_KeyID[2];
  L 0;
  <>D ;
  ) ;
  = #O_KeyIsPresent;
  SPBN mend; // jump if no key is applied to reading head

```

```

NETWORK
TITLE =check pointer to data table

// get address of pointer
L    P##IANYr_KeyList;
LAR1 ;

// check data type
L    B [AR1,P#1.0]; // data type stored in ANY-pointer
L    2;
<>I ;
SPB  mend; // jump if data type is not BYTE

// check length of data area
L    W [AR1,P#2.0]; // repeat counter (area length) stored in ANY-pointer
L    8; // size of key-id
/I   ;
T    #LI_DataNumber; // store integral result of division
SRD  16; // move remainder to the right
L    0;
<>I ;
SPB  mend; // jump if length of data area is not a multiple of key-id size

// open DB
L    0;
L    W [AR1,P#4.0]; // number of DB stored in ANY-pointer
==I ;
SPB  mdb1; // jump if data area is not part of a DB
T    #LW_DB;
AUF  DB [#LW_DB];

// get address of data area
mdb1: L    D [AR1,P#6.0]; // Adresse im ANY-Zeiger
LAR1 ;

NETWORK
TITLE =search loop

L    #LI_DataNumber;
mnxt: T    #LI_LoopCounter;

U(   ;
L    #LADW_KeyID[1];
L    D [AR1,P#0.0]; // key-id from data area, part 1
==D ;
)   ;
U(   ;
L    #LADW_KeyID[2];
L    D [AR1,P#4.0]; // key-id from data area, part 2
==D ;
)   ;
SPB  mok;

L    64; // size of one key-id (8) moved 3 bits to the left for addressing (8<<3)
+AR1 ; // get address of next key-id in data area

L    #LI_LoopCounter;
LOOP mnxt;
SPA  mend;

mok: SET ;
=    #O_KeyIsKnown;

L    #LI_DataNumber;
L    #LI_LoopCounter;
-I   ;
L    1;
+I   ;
T    #OI_EntryNumber;

NETWORK

```

```
TITLE =restore address register  
  
mend: LAR1 #LDW_AR1;  
  
END_FUNCTION
```

### 2.3.5 Funktionsaufruf

Beim Aufruf der Funktion FC1 wird der ANY-Pointer zur Kennungsliste im DB1 übergeben. Die Ausgangssignale der Funktion werden dann für die Logik zur Ansteuerung der LED verwendet.

Im Beispiel wurde dem KeyPilot die Ein-/Ausgangs-Startadresse 10 zugeordnet.

```
NETWORK  
TITLE =read key-ID and check database  
  
    CALL FC      1 (  
        IANYr_KeyList      := DB1.AUDT_KeyList,  
        II_StartAddressIn  := 10,  
        OI_EntryNumber    := MW      10,  
        O_KeyIsPresent    := M       12.0,  
        O_KeyIsKnown      := M       12.1);  
  
NETWORK  
TITLE =control LED  
  
// LED red: key present but unknown  
U      M       12.0;  
UN     M       12.1;  
=      A       10.0;  
  
// LED green: key present and known  
U      M       12.0;  
U      M       12.1;  
=      A       10.1;
```

## 2.4 Zuordnung von Rechten und Nutzerdaten

### 2.4.1 Besonderheiten

Hier soll gezeigt werden, wie einem Schlüssel verschiedene zusätzliche Informationen zugeordnet werden können, wie z.B. Zugriffsrechte oder Name des Schlüsselinhabers. Dazu wird die Liste der bekannten Schlüsselkennungen um zusätzliche Daten erweitert.

### 2.4.2 UDT2 „KPUSER“

In die Struktur dieses UDTs werden die benötigten Zusatzinformationen eingehängt. Im Beispiel sind dies folgende Daten:

- Benutzerrechte (I\_AccessLevel)
- Benutzerkennung (I\_UserID)
- Benutzerinformationen (TXT\_UserData)

Sie können die Struktur und Länge dieses UDTs problemlos ändern. Die hier gezeigten Funktionen stellen den Inhalt dieser Struktur anderen Programmteilen zur Verfügung, sind aber selbst nicht davon abhängig. Die Länge des UDTs muss den Funktionen jedoch durch passende ANY-Zeiger übermittelt werden.

Wie weiter unten gezeigt, wird die LED abhängig von I\_AccessLevel gesteuert. Dies lässt sich leicht an andere Anwendungen anpassen.

```

TYPE UDT 2
AUTHOR : RAmann
FAMILY : KP
NAME : KPUSER
VERSION : 1.0

STRUCT
  I_AccessLevel : INT ;          //rights of user
  I_UserID : INT ;              //user identification number
  TXT_UserData : STRING [94 ] ; //user name
END_STRUCT ;
END_TYPE
    
```

## 2.4.3 DB2 „KPUSRLST“

Dieser DB setzt sich aus zwei Teilen zusammen:

- UDT\_ActualUser: In diese einzelne UDT2-Struktur werden die Zusatzdaten kopiert, die dem aktuell aufgelegten Schlüssel zugeordnet wurden. Andere Programmteile können diese Daten für ihre Zwecke nutzen.
- AS\_KeyList: Dieses Array ist eine Erweiterung der im vorangegangenen Beispiel gezeigten Schlüsseliste (siehe 2.3.3 DB1 „KPLIST“). Die einzelnen Feldelemente setzen sich nun jeweils aus dem bereits bekannten UDT1 für die Schlüsselkennung, und dem UDT2 für die Zusatzinformationen zusammen.

Im Beispiel enthält AS\_KeyList drei freie Plätze und einen Eintrag, der dem Schlüssel 0100000CBF2FD5 einen Benutzer „Testuser 1“ mit der Benutzerkennung 100 und Benutzerrechten der Stufe 1 zuordnet.

```

DATA_BLOCK DB 2
TITLE =
AUTHOR : RAmann
FAMILY : KP
NAME : KPUSRLST
VERSION : 1.0

STRUCT
  UDT_ActualUser : UDT 2; //user data corresponding to found key
  AS_KeyList : ARRAY [1 .. 4 ] OF STRUCT
    UDT_Key : UDT 1; //ID of key
    UDT_User : UDT 2; //data of corresponding user
  END_STRUCT ;
END_STRUCT ;
BEGIN
  UDT_ActualUser.I_AccessLevel := 0;
  UDT_ActualUser.I_UserID := 0;
  UDT_ActualUser.TXT_UserData := '';
  AS_KeyList[1].UDT_Key.B_FC := B#16#1;
  AS_KeyList[1].UDT_Key.B_ID1 := B#16#D5;
  AS_KeyList[1].UDT_Key.B_ID2 := B#16#2F;
  AS_KeyList[1].UDT_Key.B_ID3 := B#16#BF;
  AS_KeyList[1].UDT_Key.B_ID4 := B#16#C;
  AS_KeyList[1].UDT_Key.B_ID5 := B#16#0;
  AS_KeyList[1].UDT_Key.B_ID6 := B#16#0;
  AS_KeyList[1].UDT_Key.B_CRC := B#16#35;
  AS_KeyList[1].UDT_User.I_AccessLevel := 1;
  AS_KeyList[1].UDT_User.I_UserID := 100;
  AS_KeyList[1].UDT_User.TXT_UserData := 'Testuser 1';
  AS_KeyList[2].UDT_Key.B_FC := B#16#0;
  AS_KeyList[2].UDT_Key.B_ID1 := B#16#0;
  AS_KeyList[2].UDT_Key.B_ID2 := B#16#0;
  AS_KeyList[2].UDT_Key.B_ID3 := B#16#0;
  AS_KeyList[2].UDT_Key.B_ID4 := B#16#0;
  AS_KeyList[2].UDT_Key.B_ID5 := B#16#0;
  AS_KeyList[2].UDT_Key.B_ID6 := B#16#0;
  AS_KeyList[2].UDT_Key.B_CRC := B#16#0;
  AS_KeyList[2].UDT_User.I_AccessLevel := 0;
  AS_KeyList[2].UDT_User.I_UserID := 0;
  AS_KeyList[2].UDT_User.TXT_UserData := '';
  AS_KeyList[3].UDT_Key.B_FC := B#16#0;
  AS_KeyList[3].UDT_Key.B_ID1 := B#16#0;
  AS_KeyList[3].UDT_Key.B_ID2 := B#16#0;
  AS_KeyList[3].UDT_Key.B_ID3 := B#16#0;
  AS_KeyList[3].UDT_Key.B_ID4 := B#16#0;
  AS_KeyList[3].UDT_Key.B_ID5 := B#16#0;
  AS_KeyList[3].UDT_Key.B_ID6 := B#16#0;
  AS_KeyList[3].UDT_Key.B_CRC := B#16#0;
  AS_KeyList[3].UDT_User.I_AccessLevel := 0;

```

```

AS_KeyList[3].UDT_User.I_UserID := 0;
AS_KeyList[3].UDT_User.TXT_UserData := '';
AS_KeyList[4].UDT_Key.B_FC := B#16#0;
AS_KeyList[4].UDT_Key.B_ID1 := B#16#0;
AS_KeyList[4].UDT_Key.B_ID2 := B#16#0;
AS_KeyList[4].UDT_Key.B_ID3 := B#16#0;
AS_KeyList[4].UDT_Key.B_ID4 := B#16#0;
AS_KeyList[4].UDT_Key.B_ID5 := B#16#0;
AS_KeyList[4].UDT_Key.B_ID6 := B#16#0;
AS_KeyList[4].UDT_Key.B_CRC := B#16#0;
AS_KeyList[4].UDT_User.I_AccessLevel := 0;
AS_KeyList[4].UDT_User.I_UserID := 0;
AS_KeyList[4].UDT_User.TXT_UserData := '';
END_DATA_BLOCK

```

## 2.4.4 FB1 „KPCHKUSR“

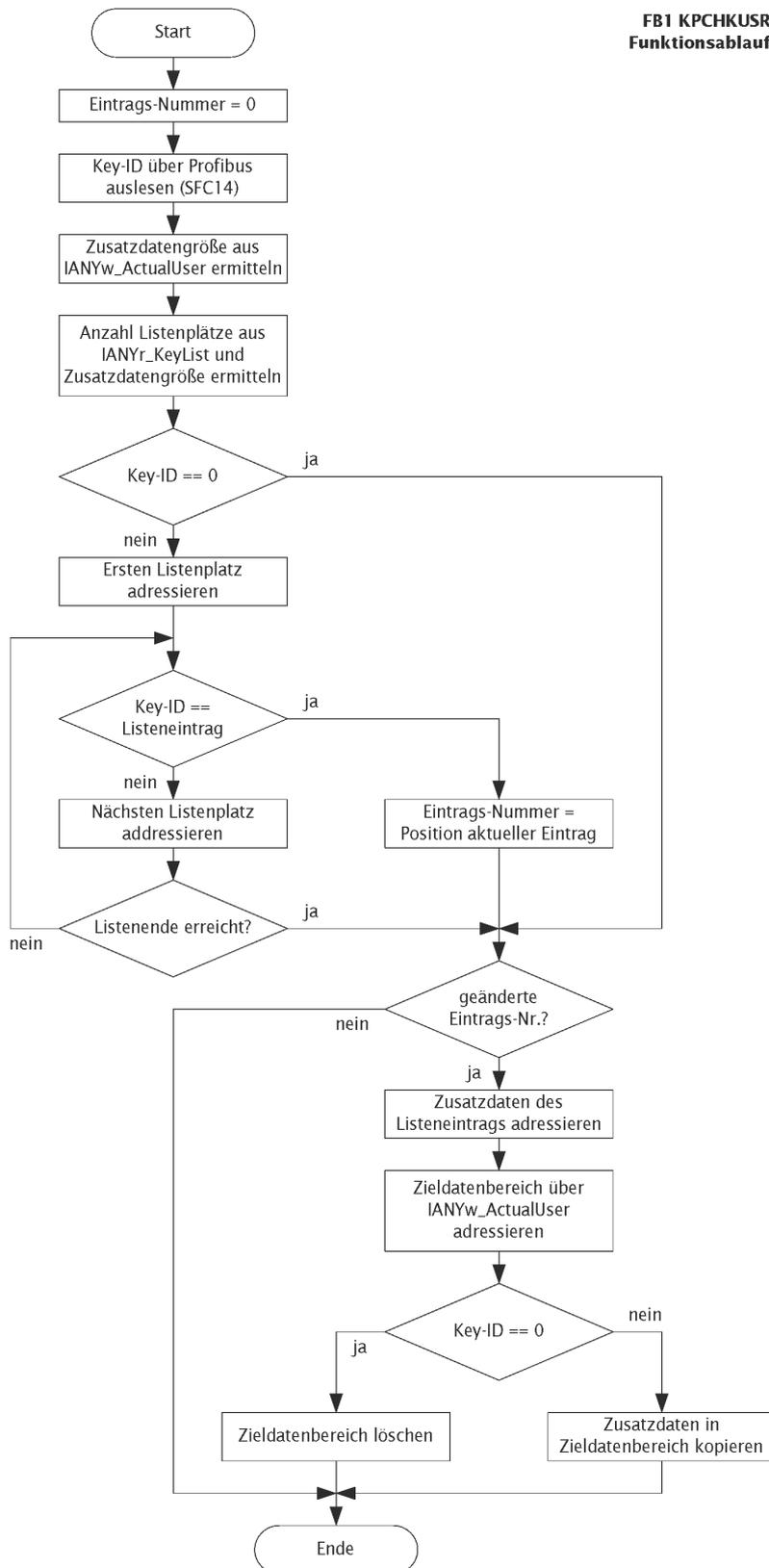
Wie die Funktion FC1 (siehe 2.3.4 FC1 „KPCHECK“) liest der Funktionsbaustein FB1 zunächst die aktuelle Schlüsselkennung über den Profibus ein, vergleicht diese mit 0 und anschließend mit einer Liste bekannter Schlüsselkennungen. Falls es eine Übereinstimmung mit einem Listeneintrag gibt, werden die Zusatzdaten, die diesem Eintrag zugeordnet sind, in einen angegebenen Speicherbereich umkopiert.

Der Baustein benötigt drei Eingangsparameter:

- IANYw\_ActualUser: Zeiger auf den Speicherbereich, in den die Zusatzdaten kopiert werden.
- IANYr\_KeyList: Zeiger auf die Liste der bekannten Schlüsselkennungen und deren Zusatzdaten
- II\_StartAddressIn: Eingangs-Startadresse des KeyPilot entsprechend der Hardware-Konfiguration

Für die korrekte Funktion muss die durch IANYr\_KeyList adressierte Liste Feldelemente enthalten, die sich jeweils aus einem Bereich für eine Schlüsselkennung (acht Byte) und einem Bereich für Zusatzdaten zusammen setzen. Die Größe der Zusatzdaten ist prinzipiell beliebig, muss allerdings genau gleich wie die Größe des durch IANYw\_ActualUser adressierten Speicherbereichs sein (siehe weiter oben Aufbau des DB2). Mit den Längenangaben aus den Zeigern IANYw\_ActualUser und IANYr\_KeyList, sowie der bekannten Größe der Schlüsselkennungen, kann der Funktionsbaustein die Anzahl der Listenelemente ermitteln.

FB1 KPCHKUSR  
Funktionsablauf



```

FUNCTION_BLOCK FB 1
TITLE =Read Key-ID and find corresponding user data
AUTHOR : RAmann
FAMILY : KP
NAME : KPCHKUSR
VERSION : 1.0

VAR_INPUT
  IANYw_ActualUser : ANY ; //pointer on buffer for actual user data [length 1*UDT2; write]
  IANYr_KeyList : ANY ; //pointer on table with known keys and user data [length N*(UDT1+UDT2);
read]
  II_StartAddressIn : INT ; //input start address of KeyPilot (according to HW-Config)
END_VAR
VAR
  SADW_KeyID : ARRAY [1 .. 2 ] OF //buffer for key-id
  DWORD ;
  SDW_EntryAddress : DWORD ; //start address of found entry
  SI_RetVal : INT ; //state information of SFC14 "DPRD_DAT"
  SI_DataNumber : INT ; //number of keys in table
  SI_LoopCounter : INT ; //loop counter
  SI_UserDataSize : INT ; //size of user data
  SI_KeyDataSize : INT ; //size of key data (id + user data)
  SI_EntryNumber : INT ; //number of found entry in list of known keys
  SI_LastEntryNumber : INT ; //number of found entry at last cycle
  SW_StartAddressIn : WORD ; //input start address from HW-Config
  S_KeyApplied : BOOL ; //flag is true if a key is applied to the KeyPilot reading head
END_VAR
VAR_TEMP
  LANYr_FoundUser : ANY ; //pointer to user data corresponding with found key
  LANYw_ActualUser : ANY ; //pointer to user data corresponding with found key
  LDW_AR1 : DWORD ; //buffer for address register 1
  LDW_AR2 : DWORD ; //buffer for address register 2
  LW_DB : WORD ; //number of DB with key table
END_VAR
BEGIN
NETWORK
TITLE =backup of address register

  TAR1 #LDW_AR1;
  TAR2 #LDW_AR2;

NETWORK
TITLE =clear values

  L #SI_EntryNumber;
  T #SI_LastEntryNumber;

  L 0;
  T #SI_EntryNumber;

NETWORK
TITLE =read key id

  L #II_StartAddressIn; // type cast from INT to WORD
  T #SW_StartAddressIn;

  CALL SFC 14 (// read data from KeyPilot
  LADDR := #SW_StartAddressIn,
  RET_VAL := #SI_RetVal,
  RECORD := #SADW_KeyID);

NETWORK
TITLE =check pointer to buffer for actual user

// get address of pointer
  L P##IANYw_ActualUser;
  LAR1 ;

// check data type
  L B [AR1,P#1.0]; // data type stored in ANY-pointer

```

```

    L    2;
    <>I  ;
    SPB  mend; // jump if data type is not BYTE

// get length of data area
    L    W [AR1,P#2.0]; // repeat counter (area length) stored in ANY-pointer
    T    #SI_UserDataSize;
    L    8; // size of key-id
    +I   ;
    T    #SI_KeyDataSize;

NETWORK
TITLE =check pointer to data table

// get address of pointer
    L    P##IANYr_KeyList;
    LAR1 ;

// check data type
    L    B [AR1,P#1.0]; // data type stored in ANY-pointer
    L    2;
    <>I  ;
    SPB  mend; // jump if data type is not BYTE

// check length of data area
    L    W [AR1,P#2.0]; // repeat counter (area length) stored in ANY-pointer
    L    #SI_KeyDataSize;
    /I   ;
    T    #SI_DataNumber; // store integral result of division
    SRD  16; // move remainder to the right
    L    0;
    <>I  ;
    SPB  mend; // jump if length of data area is not a multiple of key-data size

// open DB
    L    0;
    L    W [AR1,P#4.0]; // number of DB stored in ANY-pointer
    ==I  ;
    SPB  mdb1; // jump if data area is not part of a DB
    T    #LW_DB;
    AUF  DB [#LW_DB];

// get address of data area
mdb1: L    D [AR1,P#6.0]; // Adresse im ANY-Zeiger
    LAR1 ;

NETWORK
TITLE =key applied?

    O(   ;
    L    #SADW_KeyID[1];
    L    0;
    <>D  ;
    )    ;
    O(   ;
    L    #SADW_KeyID[2];
    L    0;
    <>D  ;
    )    ;
    =    #S_KeyApplied;
    SPBN mnot; // jump if no key is applied to reading head

NETWORK
TITLE =search loop

    L    #SI_DataNumber;
mnxt: T    #SI_LoopCounter;

    U(   ;
    L    #SADW_KeyID[1];
    L    D [AR1,P#0.0]; // key-id from data area, part 1

```

```

    ==D    ;
    )      ;
    U(     ;
    L      #SADW_KeyID[2];
    L      D [AR1,P#4.0]; // key-id from data area, part 2
    ==D    ;
    )      ;
    SPB    mok;

    L      #SI_KeyDataSize;
    SLW    3;
    +AR1   ; // get address of next key-data in data area

    L      #SI_LoopCounter;
    LOOP   mnxt;
    SPA    mnot;

mok:  TAR1  ;
      T    #SDW_EntryAddress; // store start address of found entry

      L    #SI_DataNumber;
      L    #SI_LoopCounter;
      -I   ;
      L    1;
      +I   ;
      T    #SI_EntryNumber;

NETWORK
TITLE =enable copying of user data

mnot: L    #SI_EntryNumber;
      L    #SI_LastEntryNumber;
      ==I   ;
      SPB   mend;

NETWORK
TITLE =create source pointer

// get address of pointer
      L    P##LANYr_FoundUser;
      LAR1 ;
      L    P##IANYr_KeyList;
      LAR2 ;

// copy data type
      L    W [AR2,P#0.0]; // part of pointer IANYr_KeyList
      T    W [AR1,P#0.0]; // part of pointer LANYr_FoundUser

// copy DB number
      L    W [AR2,P#4.0];
      T    W [AR1,P#4.0];

// restore AR2
      LAR2 #LDW_AR2;

// write data length
      L    #SI_UserDataSize;
      T    W [AR1,P#2.0];

// write address
      L    8; // size of key-id
      SLW  3;
      L    #SDW_EntryAddress;
      +D   ;
      T    D [AR1,P#6.0];

NETWORK
TITLE =create destination pointer

// get address of pointer
      L    P##LANYw_ActualUser;

```

```
LAR1 ;
L P##IANYw_ActualUser;
LAR2 ;

// copy pointer information
L D [AR2,P#0.0]; // part of pointer IANYw_ActualUser
T D [AR1,P#0.0]; // part of pointer LANYw_ActualUser
L D [AR2,P#4.0];
T D [AR1,P#4.0];
L W [AR2,P#8.0];
T W [AR1,P#8.0];

// restore AR2
LAR2 #LDW_AR2;

NETWORK
TITLE =copy/clear user data

U #S_KeyApplied;
SPBN mclr;

CALL SFC 20 (
SRCBLK := #LANYr_FoundUser,
RET_VAL := #SI_RetVal,
DSTBLK := #LANYw_ActualUser);

SPA mend;

mclr: CALL SFC 21 (
BVAL := #SI_EntryNumber, // if function is called, entry number is zero
-> empty fill pattern
RET_VAL := #SI_RetVal,
BLK := #LANYw_ActualUser);

NETWORK
TITLE =restore address register

mend: LAR1 #LDW_AR1;
LAR2 #LDW_AR2;

END_FUNCTION_BLOCK
```

## 2.4.5 Funktionsaufruf

Beim Aufruf des Funktionsbausteins FB1 werden die beiden ANY-Pointer zum Speicherbereich für den aktuellen Zusatzdaten und zur Datenliste, sowie die Eingangsdaten-Startadresse des KeyPilot übergeben.

Im Beispiel wird DB3 als Instanz-DB verwendet. Für die Umschaltung der LED wird ein Feld der aktuellen Zusatzdaten (I\_AccessLevel) verwendet.

```

NETWORK
TITLE =read key-ID and check database

    CALL FB      1 , DB      3 (
        IANYw_ActualUser      := DB2.UDT_ActualUser,
        IANYr_KeyList         := DB2.AS_KeyList,
        II_StartAddressIn     := 10);

NETWORK
TITLE =control LED

// LED control: set LED to green if access level is greater than zero
L      DB2.DBW      0; // DB2.UDT_ActualUser.I_AccessLevel
L      0;
>I    ;
=      A      10.1;
NOT   ;
=      A      10.0;
    
```

## 3. GSD-Datei

### 3.1 Download

Die aktuellste Version der GSD-Datei können Sie unter folgender Adresse herunterladen:

**[www.KeyPilot.de](http://www.KeyPilot.de)**

### 3.2 Siemens HW-Konfig Katalog

Nach der Installation der GSD im SIMATIC Hardware-Konfigurator wird der KeyPilot im Geräte-Katalog an folgender Position geführt:

**PROFIBUS-DP/Weitere Feldgeräte/Allgemein/KeyPilot PDP**

### 3.3 GSD für ältere Firmwarestände

Bitte verwenden Sie für folgende Geräte die aktuellste GSD-Datei:

- ⇒ Kompakt-Geräte ab Version 1.4 (Kennung EKY.PDP.T.1.4 und nachfolgend)
- ⇒ Geräte mit abgesetztem Lesekopf ab Version 2.0 (Kennung EKY.PDP.A.2.0 und nachfolgend)

Falls Sie ältere Geräte einsetzen, wenden Sie sich bitte für eine passende GSD an SysDesign, da der Funktionsumfang des KeyPilot gegenüber diesen älteren Geräten erweitert wurde.

## 3.4 Listing

```

; =====
; PROFIBUS Device Data for: KeyPilot PROFIBUS DP
; Date:                13.05.2009
; Author:              Dipl.-Ing. (FH) Robert Amann
; Language:            English
; =====
; a product of:
; SysDesign GmbH
; Säntisstraße 25
; D-88079 Kressbronn, Germany
; =====
; Support Information:  www.KeyPilot.de
; =====

#Profibus_DP

; General specifications
GSD_Revision      = 3

; Device identification
Vendor_Name       = "SysDesign GmbH"
Model_Name        = "KeyPilot PDP" ; FW-version used to ident
Revision          = "2.1"          ; file revision
Ident_Number      = 0x0A20
Protocol_Ident    = 0              ; DP protocol
Station_Type      = 0              ; Slave device
FMS_supp          = 0              ; FMS not supported
Hardware_Release  = "PDP.A/PDP.T" ; Type of HW
Software_Release  = "V2.3"         ; Firmware-version, see Model_Name
Info_Text         = "Electronic key switch for Profibus-DP, type EKY.PDP.T.x.x and EKY.PDP.A.x.x, a
product of SysDesign"

; Supported baudrates
9.6_supp          = 1
19.2_supp         = 1
45.45_supp        = 1
93.75_supp        = 1
187.5_supp        = 1
500_supp          = 1
1.5M_supp         = 1
3M_supp           = 1
6M_supp           = 1
12M_supp          = 1

; Maximum responder time for supported baudrates
MaxTsdR_9.6       = 60
MaxTsdR_19.2      = 60
MaxTsdR_45.45     = 60
MaxTsdR_93.75     = 60
MaxTsdR_187.5     = 60
MaxTsdR_500       = 100
MaxTsdR_1.5M      = 150
MaxTsdR_3M        = 250
MaxTsdR_6M        = 450
MaxTsdR_12M       = 800

; Supported hardware features
Redundancy        = 0              ; not supported
Repeater_Ctrl_Sig = 0              ; not connected
24V_Pins          = 0              ; not connected

; Implementation

```

```
Implementation_Type = "VPC3+C Solution"

; Used bitmap
Bitmap_Device = "KP_dev"
Bitmap_Diag   = "KP_dia"
Bitmap_SF     = "KP_err"

; Supported DP features
Freeze_Mode_supp = 1      ; supported
Sync_Mode_supp   = 1      ; supported
Auto_Baud_supp   = 1      ; supported
Set_Slave_Add_supp = 1    ; supported

; User parameter data
User_Prm_Data_Len=12
User_Prm_Data=0x00,0x00,0x00,0x55,0xAA,0xAA,0xAA,0xAA,0xF0,0xF0,0xF0,0xF0

; Maximum polling frequency
Min_Slave_Intervall = 6

; Module
; 1 Byte Output Data
; 8 Byte Input Data: |FC|ID1|ID2|ID3|ID4|ID5|ID6|CRC|
Modular_Station = 0      ; not modular
Module          = "EKY.PDP.T.x.x / EKY.PDP.A.x.x" 0xC0, 0x80, 0x87
                23      ; module reference (23 for firmware V2.3)
Info_Text       = "Data module for Key-ID and LED-control. HW-type EKY.PDP.T.1.4 or EKY.PDP.A.2.0
and higher (FW2.3)"
EndModule

; Fail safe mode
Fail_Safe       = 1      ; state CLEAR accepted

; Maximum diagnosis data length
Max_Diag_Data_Len = 22

; Slave family
Slave_Family    = 0      ; General
```