Fachhochschule
Dortmund
University of Applied Sciences and Arts

FB Informations- und Elektrotechnik

Bachelor Thesis
zur Erlangung des Akademischen Grades
Bachelor of Engineering

Flexible Programmierung einer Bürogebäudesteuerung mit der WAGO-SPS

Carsten Butz
Matrikel Nr. 7089503

Erstprüfer: Prof. Dr. Bernd Aschendorf

Zweitprüfer: Dipl. Ing. Sandra Stahlberg

Abgabedatum: Dortmund, 21.08.2015

Zusammenfassung (Deutsch)

Unter dem Namen WAGO-I/O-System vermarktet WAGO ein modulares Automatisierungssystem bestehend aus verschiedenen Speicherprogrammierbaren Steuerungen, die mit unterschiedlichen Ein- und Ausgangsklemmen gekoppelt auf Hutschienen montiert werden. Grundsätzlich ist das System für beliebige Automatisierungsaufgaben geeignet, damit auch für Aufgaben der Gebäudeautomation. Insbesondere werden Steuerungen mit Schnittstellen und separate Kommunikationsklemmen für verschiedene im Gebäudebereich verbreiteter Bussysteme produziert, so u.a. ETHERNET, KNX, BACnet, LONWORKS, EnOcean und DALI.

In der vorliegenden Arbeit wird die Steuerung einer Büroetage mit 40 Raumachsen programmiert, wobei eine Raumachse den Raum zwischen Außenwand und Flur beschreibt in der Breite eines Fensters. Jede Raumachse ist mir Lampen, Anwesenheitssensor, Außen- und Innenjalousien und mit Heizung und Kühlung ausgestattet. Raumachsen können flexibel zu Büro zusammengefasst werden und werden je nach Bedarf mit EnOcean-Tastern zur Jalousie- und Lichtsteuerung ausgestattet sowie mit EnOcean Raumthermostaten.

Die Herausforderungen an die Programmierung sind dreifach. Zum Einen betrachten wir, bei Vollausstattung der Raumachsen und Büros, ca. 50 Datenpunkte pro Raumachse und damit ca. 2000 Datenpunkte insgesamt, die sich kaum noch durch einfache Ein- und Ausgangsklemmen bewältigen lassen. Bei der Kommunikation greifen wir daher auf EnOcean zurück, bei der Lichtsteuerung auf den DALI-Bus. Zum Zweiten soll die Zuordnung von Raumachsen zu Räumen und damit auch die Ausstattung von Räumen flexibel sein. Insbesondere lassen sich Räume und deren Ausstattung zur Laufzeit des SPS-Programms ändern, ohne das Programm neu- bzw. umschreiben zu müssen. Zum Dritten muss der gesamte Flur über eine Applikation mit grafischer Benutzeroberfläche überwacht und parametriert werden, wobei die Bordmittel der Programmierumgebung CoDeSys mit integrierter Visualisierung nur ca. 100 Datenpunkte gleichzeitig überwachen und anzeigen können.

Abstract (English): Flexible Programming of Office Building Controls Using a Programmable WAGO Field Bus Controller

WAGO-I/O-SYSTEM is the name of a modular rail-mounted automation system produced and marketed by WAGO where a programmable field bus controller is combined with different modules handling digital and analogue in- and output and communication. Originally targeted at industrial applications the system has become more and more prominent in building automation, partly due to the great number of available communication interfaces to bus systems used in building automation like Ethernet, KNX, BACnet, LONWORKS, EnOcean or DALI, but also due to the flexibility gained by the ability to program almost every function needed in building automation.

In this thesis we develop and document the control of the floor of an office building. Offices are flexibly arranged from 40 segments where a segment is the space between the outer wall and the corridor of width one window. Each segment is equipped with lights, motion sensor, outside roller blinds and inside sun blinds, heating and cooling. Sensors are added according to need and consist of 2-channel 2-rocker switches to control blinds, shades and light, and room operating panels to control the room climate (heating and cooling).

The challenges are threefold. First, fully equipped segments contain about 50 data points and thus we have in total about 2000 data points that have to be maintained, which can only be reasonably achieved using communication busses. In this project we use EnOcean technology to handle communication between the field bus controller and sensors and the digital addressable lighting interface (DALI) to control lighting functions. Second, we need to be flexible about the set-up of offices and the amount and function of sensors (rocker switches) added to each office. In particular, the sensors attached to offices and their functionality should be alterable at run-time without the need to reprogram the software. Finally, the floor has to be monitored and operated (global functions, rearranging of offices, addition or replacement of sensors) using an application with graphical user interface, whereupon the visualization tool of the development kit CoDeSys only allows to monitor and visualize about 100 data points simultaneously.

Inhalt

1. Einleitung

Die Komponenten aus dem modularen Automatisierungssystem WAGO-I/O-System[1] sind, insbesondere durch umfangreiche Schnittstellen für verschiedene im Gebäudebereich verbreiteter Bussystem wie Ethernet, KNX, BACnet, LONWORKS und EnOcean, hervorragend geeignet für Aufgaben der Gebäudesteuerung und Gebäudeautomation. Ein sogenannter Feldbusknoten besteht aus einem Controller (speicherprogrammierbare Steuerung, SPS), digitalen Ein- und Ausgangsklemmen und z.B. Kommunikationsklemmen, die mit unterschiedlichen Bussystemen kommunizieren können. Betrachtet man die Leistungsfähigkeit der Controller, aber auch die technischen Beschränkungen, so sollte ein Feldbusknoten in der Lage sein, alle relevanten Steuerungsaufgaben in komfortabel ausgestatteten Eigenheimen zu übernehmen, aber auch die Steuerungsaufgaben von Büroetagen kleiner und mittlerer Größe in kommerziell genutzten Gebäuden.

In dieser Arbeit programmieren wir einfache Steuerungsaufgaben, wie sie in einer typischen Büroetage vorkommen: Licht, Jalousiefunktionen, Heizung und Kühlung. Um die Möglichkeiten und Grenzen einer Etagensteuerung mit einer WAGO-SPS auszuloten, gehen wir in mindestens drei Aspekten an die Grenzen der Leistungsfähigkeit der WAGO-SPS.

(1) Wir betrachten eine Büroetage mit 40 Büroachsen, wobei eine Büroachse dem Raum in der Breite eines Fensters zwischen Außenwand und Flur entspricht. Je nach Ausstattung kann eine Büroachse zwischen 30 und 50 Datenpunkte haben, und wir sprechen damit über 1200 bis 2000 Datenpunkte nur für die aus Büroachsen zusammengesetzten Büros eines Flurs.
(2) Durch Umbaumaßnahmen können Büroachsen zu Büros unterschiedlicher Breite zusammengefasst werden. Büros werden mit EnOcean-Sensoren ausgestattet und die Definition der Funktionen muss flexibel während der Laufzeit Steuerung möglich sein (Parametrierung statt Programmierung).
(3) Managementaufgaben wie die Definition von Büros (begrenzt durch zwei Trennwände), Hinzufügen von Tastern und ihren Funktionen, Monitoring-Aufgaben, etc. sollen einfach zu bewerkstelligen sein.

Aufgrund der großen Anzahl von Datenpunkten muss ein wesentlicher Teil der Datenpunkte über ein Bussystem abgefragt bzw. geschrieben werden. Zusammen mit der geforderten Flexibilität bietet sich EnOcean als Technologie an, und wir werden die Räume flexibel mit EnOcean Raumthermostaten und Tastern ausstatten.

Auch bei der (An-)Steuerung der Lichtaktoren bietet sich die Verwendung eines Bussystems an, und wir verwenden Aktoren mit DALI Schnittstelle. Prinzipiell wäre es auch möglich, die Jalousiefunktionen über ein Bussystem zu steuern, da wir aber keine entsprechende Schnittstelle zum Testen der Grundfunktionen zur Verfügung hatten, haben wir darauf verzichtet.[2]

[1] Die folgenden Ausführungen basieren auf dem Produktkatalog (WAGO, Produktkatalog) und den Projektierungshinweisen (WAGO, WAGO-I/O-System 750 - Projektierungshinweise, Version 2.0.1, 2007).
[2] Der Bus wäre in diesem Fall der Standard Motor Interface Bus, kurz SMI-Bus, der mit einer WAGO-SPS und einer SMI-Schnittstelle, angeschlossen an eine serielle Kommunikationsklemme, realisiert werden kann und pro Schnittstelle bis zu 16 Jalousiemotoren steuern kann, vgl. (Aschendorf B. , 2014) S. 746ff und S. 806, und (WAGO, Bausteinbeschreibungen für den Interface_Baustein RS-232/SMI, 2014).

2. Szenario

Focus dieser Arbeit ist die flexible Programmierung von Funktionen der Gebäudesteuerung und Gebäudeautomation auf einer WAGO-SPS. Um die programmiertechnischen Konzepte besser zu verdeutlichen, arbeiten wir mit einem konkreten Szenario:

2.1 Szenario: Variabel zu gestaltende Büroetage

Zu programmieren ist die Steuerung einer Büroetage bestehend aus einem Flur (2 Lichtstromkreise, Bewegungssensoren), einer Teeküche (1 Lichtstromkreise, Lichtschalter) und zwei Toiletten (je 1 Lichtstromkreis, Heizung mit Heizungsregelung). Es gibt insgesamt 40 Büroachsen, wobei eine Büroachse aus dem Bereich zwischen Flur und Außenfassade besteht, und genau eine Fensterbreite hat. Jede Büroachse hat 3 dimmbare Lichtstromkreise (Deckenbeleuchtung), 2 Steckdosenstromkreise, Heizung und Kühlung, und Innen- und Außenjalousie. In jeder Achse ist ein Anwesenheitssensor untergebracht.

Büroachsen können durch Umbau zu einzelnen Büros unterschiedlicher Breite zusammengefasst werden. Die Büros werden dann mit Tastern (Licht, Jalousie) und Raumthermostaten ausgestattet, die über eine WAGO-SPS die entsprechenden Bürofunktionen steuern.

Auf einem Flur befinden sich damit Büros unterschiedlicher Größe, die auch an verschiedene Kunden vermietet sein können.

2.2 Klassische Implementierung

Wir werden, ohne auf Details einzugehen, darlegen, wie man die Programmierung der variablen Anteile in einer klassischen Hochsprache (z.B. Java, C++ oder C#) implementieren würde.

- Es gibt eine Konstante für die Anzahl der Büroachsen.
- Der Datentyp Kunde besteht aus Kundendaten und einer Liste (möglicherweise leer) aus (derzeit) angemieteten Räumen:

```
class Kunde {
Daten: Kundendaten;
Mietobjekt: LIST OF Raum;
}
```

Code 1: Klasse Kunde

- Ein Raum besteht aus mehreren nebeneinanderliegenden Raumachsen und einer Raumsteuerung:

```
class Raum {
linkeAchse, rechteAchse: INT;
Steuerung: Raumsteuerung;
}
```

Code 2: Klasse Raum

  Gegebenenfalls kann auch bei jedem Raum der Kunde hinterlegt sein.
- Eine Raumsteuerung ist eine Liste von Sensoren (Schalter, Taster, Anwesenheitssensoren):

```
class Raumsteuerung {
Sensoren: LIST OF Sensor;
}
```

Code 3: Klasse Raumsteuerung

- Schlussendlich wird ein Sensor beschrieben durch einen Typ (1-fach Taster, Mehrfachtaster), ID (z.B. EnOcean ID) und einem oder mehreren Eventlistenern, bei denen der Sensor registriert ist. Werden Events ausgelöst (Taster gedrückt, Bewegung im Raum, Raumtemperatur verändert, etc.), so werden die Eventlistener aufgerufen und führen die entsprechende Aufgabe (Licht ein, Anwesenheit registrieren, Heizung regeln, etc.) aus.

```
class Sensor {
ID: INT;
TYP: Sensortyp; (* hier ist man flexible *)
```

```
        Steuerung: LIST OF Eventlistener;
        }
```
Code 4: Klasse Sensor

Zu jeder einzelnen Klasse gibt es dann Methoden die zum Beispiel erlauben, zu einem Raum einen neuen Sensor hinzufügen. Über eine grafische Benutzeroberfläche (GUI, Graphical User Interface) hätte man dann die Möglichkeit, einen neuen Raum über seine linke und rechte Grenze zu definieren (der Konstruktor muss prüfen, dass es keine Überlappung mit bereits definierten Räumen gibt), und der Raumsteuerung Sensoren zuzufügen, wobei die entsprechenden Objekte zur Laufzeit des Programms erzeugt werden und, werden Sensoren wieder weggenommen, auch wieder gelöscht werden, d.h., der Platz im Speicher wird wieder freigegeben.

2.3 Diskussion

Mit der Überarbeitung der Norm DIN EN 61131-3[3] stehen seit Ende 2013 auch objektorientierte Programmierparadigma auf Normenebene zur Verfügung, hierzu gehört insbesondere die Möglichkeit Klassen und Methoden zu definieren und damit auch die Möglichkeit, interne Variablen und Implementierungsalgorithmen zu kapseln. Objektorientierte Methoden werden, in verschiedenem Umfang, in CoDeSys V3 unterstützt[4], WAGO arbeitet aber immer noch mit CoDeSys V2.3,[5] und die objektorientierten Methoden stehen bei der Programmierung von WAGO Controllern *nicht* zur Verfügung. Hiermit entfällt derzeit auch die Möglichkeit, Objekte bzw. Funktionsbausteine während der Laufzeit dynamisch zu erzeugen, wie dies in modernen objektorientierten Programmiersprachen möglich ist. Damit sind auch weiterhin alle Funktionen global.

Eine weitere Herausforderung ist das Fehlen von dynamischen (rekursiven) Datenstrukturen. Insbesondere gibt es keinen Datentyp für Listen und der Versuch, rekursive Datentypen zu definieren, wird mit einer Fehlermeldung quittiert.[6] Selbst bei Feldern fester Länge (Arrays) muss die Länge bei der Übersetzung festgelegt sein. Der Versuch, einen Datentyp

```
Laenge: INT := 40;
Raumachsen: ARRAY[1..Laenge] OF INT;
```
Code 5: CoDeSys: Fehlerhafte Deklaration eines Arrays

nur zur Definition der Raumachsennummern zu deklarieren führt zu einer Fehlermeldung[7], da die Länge innerhalb der Deklaration des Arrays entweder als Zahl (40) fest vorgegeben oder aber die Variable `Laenge` explizit als Konstante deklariert sein muss.

[3] DIN EN 61131-3 Speicherprogrammierbare Steuerungen – Teil 3: Programmiersprachen (IEC 61131-3:2013) als Ersatz für DIN EN 61131-3: 2003-12.
[4] Aktuell (Juli 2015) ist die Version V3.5 SP 6 Patch 4 vom 28.05.2015 (CoDeSys).
[5] Aktuell (Juli 2015) ist die Version V2.3.9.47 vom 02.04.2015 (CoDeSys), welche, als Version V2.3 der Software WAGO-I/O-Pro zugrunde liegt (WAGO, Produktkatalog).
[6] Wir werden später zeigen, wie man die Überprüfung auf Rekursion durch den Compiler aushebeln kann, werden dies aber nicht weiter verfolgen.
[7] Fehler 3750: [Baustein oder Programmname](Zeilennummer): Obergrenze ‚LAENGE' unbekannt.

2.4 Datenpunkte und Funktionsbeschreibung

Im Folgenden werden wir die Etage ausplanen um zu einer sinnvollen Funktionsbeschreibung zu kommen. Mit F benannte Funktionen wurden implementiert, mit G benannte Funktionen wurden nicht implementiert, wären aber wünschenswert. Zu den Datenpunkten verweisen wir insbesondere auf (Aschendorf B. , Datenpunktaufstellung Bürobereich), wo mögliche Datenpunkte und Funktionen beschrieben werden.

Flur

Datenpunkte:

- 2 Lichtstromkreise
- 2 Einfachtaster an den Enden des Flurs
- 4 Bewegungsmelder

Funktion:

F1: Taster oder Bewegungsmelder schalten das Licht für 1 Minute ein.
G2: Über eine Kalenderfunktion werden zu Bürozeiten (Montag bis Freitag, 7:00 bis 16:00) die Sensoren abgeschaltet und die Beleuchtung ist dauerhaft eingeschaltet.

Teeküche:

Datenpunkte:

- 1 Lichtstromkreis
- 1 Taster zum Ein- bzw. Ausschalten des Lichts

Funktion:

F1: Der Taster schaltet das Licht ein und aus (Umschalt-/Toggle-Funktion).

Toiletten (jeweils eine Damen- und eine Herrentoilette):

Datenpunkte:

- 1 Lichtstromkreis
- 1 Taster zum Ein- bzw. Ausschalten des Lichts
- 1 Anwesenheitssensor

Funktion:

F1: Der Taster schaltet das Licht ein bzw. aus.
F2: Bei Abwesenheit wird das Licht nach 5 Minuten ausgeschaltet.

Büroachse (Breite: 1 Fenster):

Datenpunkte:

- 3 Lichtstromkreise (DALI)
- 1 Anwesenheitssensor
- Außen- und Innenjalousie (je 1 mit Jalousie auf und Jalousie ab)
- Heizungs- und Kühlungsaktoren (je 1 mit Heizung und Kühlung)

Funktion:

Keine

Büro (Zusammenschluss mehrerer Büroachsen):

Datenpunkte:

- X 4-fach Taster (EnOcean) zur Jalousiesteuerung (außen, innen), maximal 2 4-fach Taster pro zugeordneter Büroachse
- X Temperaturfühler durch zugeordnetes Raumthermostat (EnOcean), maximal 1 Raumthermostat pro Büro
- X Anwesenheitstaster durch zugeordnetes Raumthermostat (EnOcean), maximal 1 Raumthermostat pro Büro
- X Sollwertverstellung durch zugeordnetes Raumthermostat (EnOcean), maximal 1 Raumthermostat pro Büro
- X 4-fach Taster (EnOcean) zur Lichtsteuerung (DALI), maximal 2 4-fach Taster pro zugeordneter Büroachse

Funktion:

F1: Jalousiesteuerung (außen, innen) durch zugeordnete EnOcean 4-fach Taster (inklusive Lamellenverstellung außen).
F2: Klimasteuerung (Einzelraumsteuerung Heizen, Kühlen) durch zugeordnetes EnOcean Raumthermostat.
F3: Komfortbetrieb bei Anwesenheit.
F4: Lichtsteuerung durch zugeordnete EnOcean 4-fach Taster; alle Lichtstromkreise sind dimmbar und die Dimmfunktionen werden über eine Wippe realisiert. Darüber hinaus gibt es die Möglichkeit, Szenen abzurufen.

Management (Bürodefinition):

Datenpunkte:

- 41 (Trenn-)Wände für 40 Büroachsen
- X Kunden-IDs, maximal 40

Funktion:

F1: Trennwand vorhanden/nicht vorhanden.
F2: Zuordnung von Kunden-ID zu Büro („Mieter“).
F3: Zuordnung von Raumthermostat (1 pro Büro).
F4: Zuordnung von Jalousietastern (EnOcean 4-fach Taster) für Innen- und Außenjalousien (maximal 2 pro zugeordneter Raumachse); die Taster steuern gleichzeitig alle Jalousien der zuordneten Achsen.
F5: Zuordnung von Lichttastern (EnOcean 4-fach Taster) und Definition der Dimm- oder Schaltfunktion (maximal 2 4-fach Taster pro zugeordneter Raumachse).
G10: Definition des Basissollwerts der Raumtemperatur individuell für jedes Büro.

Management (Überwachung, Betrieb):

Datenpunkte:

- Keine weiteren Datenpunkte

Funktion:

F1: Status der Lampenaktoren.
F2: Anfahren der Sicherheitsposition und Verriegelung zum Schutz bei Unwetter oder für Wartungs- oder Reinigungsarbeiten der Innenjalousien.
F3: Anfahren der Sicherheitsposition und Verriegelung zum Schutz bei Unwetter oder für Wartungs- oder Reinigungsarbeiten der Außenjalousien.
F4: Anfahren der Beschattungsposition der Innenjalousien.
F5: Anfahren der Beschattungsposition der Außenjalousien.
F6: Aktivieren und Deaktivieren des Nachtmodus für Klima- und Heizungsanlage.
F7: Zentrales Ausschalten des Lichts (manuell)

Bereits an dieser Stelle muss man sich klar machen, das die vorgesehenen Aufgaben die WAGO-SPS an ihre Grenzen bringen werden: Bei maximaler Auslastung (jede Büroachse ein eigenes Büro) haben wir pro Achse im Bereich
Klima

- 2 Datenpunkte für Stellantrieb Heizen,
- 2 Datenpunkte für Stellantrieb Kühlen,
- 5 Datenpunkte für das Raumthermostat (Ist- und Solltemperatur, Sollwertabweichung, Anwesenheit, EnOcean-ID des Thermostats),

Jalousien

- 4 Datenpunkte für die Stellantriebe (innen und außen jeweils auf und ab),
- 10 Datenpunkte für die 2 Taster (jeweils eine EnOcean-ID und 4 Wippen),

Licht

- 3 Datenpunkte für die Lichtstromkreise (z.B. aktueller Dimmwert),
- 10 Datenpunkte für die 2 Taster, vgl. oben,
- X weitere Datenpunkte für DALI-Klemmen-ID, Zustand der Lampen bzw. Aktoren,

Sonstige

- 1 Datenpunkt für den Anwesenheitssensor,

insgesamt also mindestens 40 Datenpunkte pro Raumachse[8], 1600 pro Flur. Hierzu kommen noch die Datenpunkte von Flur, Teeküche, Toiletten, etc. Insbesondere die Managementfunktionen lassen sich damit nicht mehr global über eine Visualisierung steuern, da Visualisierungen nur ca. 100 Datenpunkte gleichzeitig überwachen können.[9] Andererseits kann eine WAGO-SPS auch nur ca. 1000 Funktionsbausteine verwalten,[10] also haben wir nur 25 Funktionsbausteine pro Büroachse für die Steuerungsaufgaben zur Verfügung. Auch wenn dies im erste Augenblick viel erscheint - 5 Bausteine für die 4 4-fach Taster und das Raumthermostat, 2 Bausteine für die Jalousiesteuerung, 1 Baustein für die Klimasteuerung, 3 Bausteine für die Dimmfunktion der Lampen, etc. gibt noch Luft - so sollte man doch mit weiteren Bausteinen geizen da einige der verwendeten Baustein intern andere Funktionsbausteine benutzen.

Bei den digitalen Ausgangsklemmen muss auf Klemmen mit vielen Anschlüssen zurückgegriffen werden: Bei mindestens 6 digitalen Ausgangssignalen pro Raumachse (Stellantriebe Heizen, Kühlen Innen- und Außenjalousien auf und ab) benötigen wir mindestens 240 digitale Ausgangskanäle.[11]

In den folgenden Kapiteln dieser Arbeit werden wir Technologien (DALI, EnOcean) und Methoden (flexible Programmierung, Visualisierung) vorstellen und diskutieren, welche Bedeutung und welche Einschränkung diese für unser Projekt haben.

[8] (Aschendorf B. , Datenpunktaufstellung Bürobereich) nennt 40-50 Datenpunkte pro Raumachse für durchaus realistisch.

[9] Es gibt weder bei CoDeSys noch bei WAGO genaue Angaben über die maximal mögliche Anzahl von überwachten Variablen in einer Visualisierung. Die Zahl entspricht unseren Erfahrungswerten.

[10] Der (einstellbare) Standardwert für die maximale Anzahl der Funktionsbausteine bei der Auswahl des Zielsystems ist 1024 und sollte zumindest als ein Maß dafür verstanden werden, welchen Funktionsumfang die SPS noch sinnvoll bearbeiten kann.

[11] An eine WAGO-SPS lassen sich mit Klemmbusverlängerung bis zu 10 (weitere) Teilknoten anbinden, insgesamt lassen sich 256 Klemmen anschließen, wobei vermutlich die Klemmbusverlängerungen und die Endklemmen 750-600 mitgezählt werden.

3. Digital Addressable Lighting Interface (DALI)

3.1 Systembeschreibung DALI

Das Digital Addressable Lighting Interface (DALI) ist ein Interface und Protokoll[12] zur Steuerung von zu Beleuchtungszwecken eingesetzten Steuergeräten wie elektronischen Vorschaltgeräten oder Dimmern (DALI AG, DALI Handbuch, 2. Auflage, 2002). Zur Kommunikation werden Telegramme auf einem asynchronen, bidirektionalen Bus (DALI-Bus) mit Manchester-Codierung mit 1200 Bit/s verschickt. Die Spannungspegel sind 0 V ± 4,5 V) für den Low-Pegel, und 16 V ± 6,5 V für den High-Pegel, jeweils auf Empfängerseite, bei einem Spannungsabfall von maximal 2 V. Der Bus kann stern- oder baumförmig aufgebaut sein (keine Ringe), und kann bis zu 64[13] Teilnehmer adressieren bei einer maximalen Stromaufnahme aller Teilnehmer von 250 mA.[14]

Telegramme werden durch ein Startbit und zwei Stoppbits umrahmt, und bestehen aus zwei Datenbytes (Kommunikation Zentrale – Teilnehmer)[15] oder einem Datenbyte (Kommunikation Teilnehmer – Zentrale).

Die Buskommunikation folgt dem Master-Slave-Prinzip. Abhängig davon, ob der Bus von einer Kontrolleinheit mit angeschlossenen Sensoren, oder von mehreren Kontrolleinheiten mit jeweils angeschlossenen Sensoren gesteuert wird, wird das Single-Master und Multi-Master Prinzip unterschieden.

Ähnlich wie KNX kennt DALI auch Gruppenadressen zur gemeinsamen Steuerung von Funktionen, wobei jeder Teilnehmer zu bis zu 16 Gruppen gehören kann. Jeder Teilnehmer kann bis zu 16 Lichtszenen speichern, die auch über Gruppenadressen abgerufen werden können.

Ein einzelner DALI-Bus steuert, aufgrund seiner geringen Größe, die Lichtszenen einzelner Büro-/Konferenzräume, Flure oder kleine Etagen. Bei großen Installationen müssen die einzelnen DALI-Busse über andere Bussysteme gekoppelt werden.[16]

---

[12] Das Protokoll ist in der IEC 62386, Digital adressierbare Schnittstelle für die Beleuchtung, genormt.

[13] Die Adressen zwischen 0 und 63 können entweder vor der Installation vergeben werden, oder aber nach der Inbetriebnahme, manuell oder automatisch. Neben der Kurzadresse verfügen alle DALI Betriebsgeräte über eine 24-Bit Langadresse, die zur eindeutigen Identifizierung von Teilnehmer benutzt wird. Sollten zufälligerweise zwei Langadressen in einem DALI-Bus übereinstimmen, so können Steuergeräte in Aktoren neue Langadressen erzeugen, die die alten, im Herstellungsprozess vergebenen Langadressen, überschreiben.

[14] Da die Spannung auf dem Bus gemäß Norm keine sichere Kleinspannung (Safety Extra Low Voltage, SELV) darstellt, werden Steuergeräte üblicherweise mit einem Standardinstallationskabel 5x1,5 mm^2 angeschlossen, wobei 3 Adern die Versorgungsleitung bilden (Netzanschluss), und zwei weitere als Steuerleitungen (Bus) verwendet werden.

[15]
Bit 15:	0 für Geräteadresse (Individualadresse), 1 für Gruppenadresse
Bit 14 - 9:	6-Bit Geräte- oder Gruppenadresse
Bit 8:	0 für Parameter, 1 für Befehl
Bit 7 – 0:	Parameter oder Befehl

[16] Der Gewinner des DALI Awards 2014 im Bereich Anwendung ist das World Trade Centre in Abu Dhabi, U.A.E., mit 60000 Individualadressen für die Licht- und Jalousiesteuerung (ca. 36000 Leuchten), vgl. (DALI AG, Winner DALI Award 2014).

3.2 Aufbau eines DALI-Busses mit der WAGO Klemme 750-641 (DALI-Masterklemme)

Mit der WAGO Klemme 750-641 lässt sich, gesteuert durch eine WAGO SPS, ein Single-Master DALI-Bus steuern.[17] Mit der Bibliothek `DALI_02_d.lib` steht eine umfangreiche Bibliothek zur Verfügung. Im Folgenden beschreiben wir kurz eine Auswahl der wichtigsten Bausteine. Bei allen Bausteinen gilt, dass die Adresse `bAddress` als Teilnehmeradresse bei `xGroup = FALSE` und als Gruppenadresse bei `xGroup = TRUE` interpretiert wird.

- Der Baustein `FbDALI_Joblist` verwaltet die Kommunikation mit dem DALI Bus.

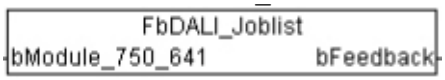


Abbildung 1: Funktionsbaustein `FbDALI_Joblist`

Der Eingang `bModule_750_641` bekommt als Wert, wie üblich bei der Kommunikation zwischen einer WAGO-SPS und einem Bus, die Nummer der Klemme im Klemmbus, wobei die erste Klemme die Nummer 1 erhält, die zweite die Nummer 2, usw. Der Ausgang `bFeedback` gibt einen Statuscode aus[18].

Der Baustein sollte vor allen anderen DALI Funktionsbausteinen aufgerufen werden und darf, pro Klemmbaustein 750-641 auch nur einmal vorhanden sein.[19]

- Die sogenannte Basisbausteine `FBDALI_Master` und `FbDALI_Master_Adv` schreiben nach DIN IEC 60929 spezifizierte DALI-Befehle auf den DALI-Bus.
- Die Bausteine `FbDALI_DimmSingleButton`, `FbDALI_DimmEasy` dienen dem Dimmen mittels Einfachtaster, der Baustein `FbDALI_DimmDoubleButton` dem Dimmen mittels Zweifachtaster:

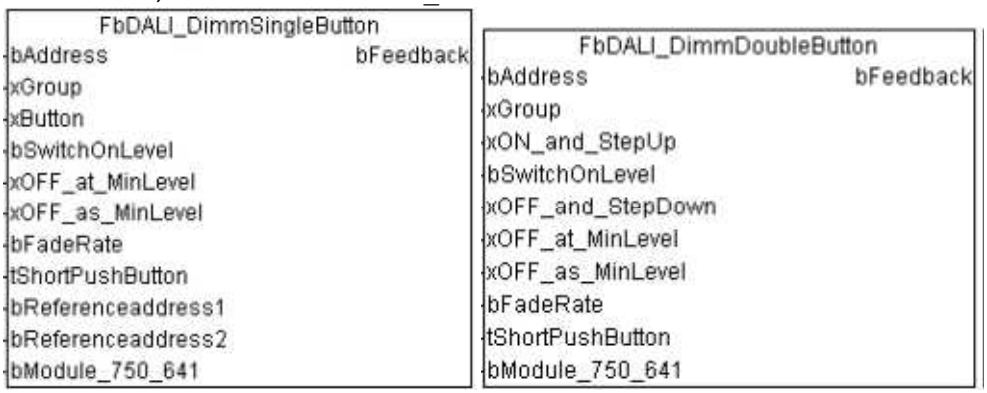


Abbildung 2: Funktionsbausteine `FbDALI_DimmSingleButton` und `FbDALI_DimmDoubleButton`

- Der Baustein `FbDALI_LatchingRelay` ist ein Stromstoßschalter, der Baustein `FbDALI_SwitchValue` setzt einen Teilnehmer oder eine Gruppe auf einen bestimmten (Dimm-)Wert:

[17] Zur Spannungsversorgung des DALI-Busses ist eine separate Spannungsversorgung nötig. WAGO selbst bietet unter der Nummer 288-895 einen DC/DC-Wandler mit 18 V (sic!) Ausgangsspannung an, der bis zu drei vollständig ausgebaute DALI-Busse mit Strom versorgen kann, vgl. (WAGO, Handbuch zur DALI/DSI-Masterklemme 750-641, 2011).

[18] Vgl. Tabelle 6 auf S. 57 in (WAGO, DALI_02_d, Bausteinbeschreibungen für die DALI-Masterklemme 750-641, 2008). Beispiele sind (hexadezimal) 00 (Kein Fehler), 03 (Jobliste im Controller ist voll), 09 (DALI Bus Fehler), 0D (Falsche Szenenummer), 13 (Adresse unbekannt), 15 (Adresse vergeben), 16 (DALI Modul wurde nicht erkannt).

[19] (WAGO, DALI_02_d, Bausteinbeschreibungen für die DALI-Masterklemme 750-641, 2008), S. 7.

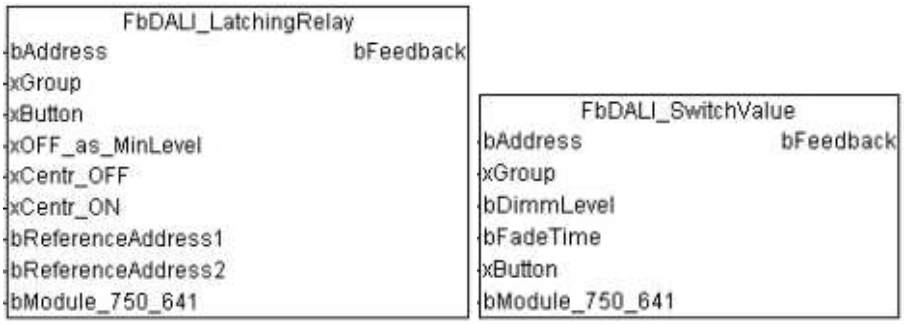


Abbildung 3: Funktionsbausteine `FbDALI_LatchingRelay` und `FbDALI_SwitchValue`

- Der Baustein `FbDALI_ConstantLightControl` realisiert eine Konstantlichtregelung für einen Teilnehmer bzw. eine Gruppe in Verbindung mit einem Helligkeitssensor.

Abbildung 4: Funktionsbaustein `FbDALI_ConstantLightControl`

- Der Baustein `FbDALI_RecallScene` ruft, bei einer positiven Flanke an `xSceneButton` die Szene `bSceneNo` ab:

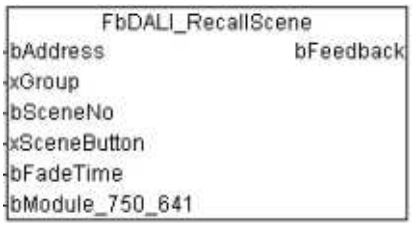


Abbildung 5: Funktionsbaustein `FbDALI_RecallScene`

Im einfachsten Fall benötigt man nur die Jobliste und die Dimmer- bzw. Szenebausteine.

Beispiel: Im Beispiel `Test-Koffer-WAGO-mit DALI.pro` werden an einem einfachen DALI-Bus zwei Gruppen über 1-fach Taster gedimmt bzw. zwei vordefinierte Szenen aufgerufen.[20] Als Taster dient der EnOcean Taster mit der ID 1140678, die linke Wippe dimmt oben Gruppe 1 und wählt unten Szene 1 aus, die rechte Wippe dimmt oben Gruppe 2 und wählt unten Szene 2 aus.

[20] Zum Testen wurde der DALI-Bus im Labor A-112 verwendet, an dem 5 elektronische Vorschaltgeräte vom Typ PCA 1/36 Excel 220-240V 50/60/0Hz der Firma Tridonic angeschlossen sind. Programmiert wurden zwei Gruppen (Gruppe 1: 1-3, Gruppe 2: 4-5) und zwei Szenen (Szene 1 und Szene 2 mit vordefinierten Werten), vgl. den folgenden Abschnitt.

Der Programmcode in diesem einfachen Beispiel besteht aus der Deklaration der Variablen für die Jobliste, die Antwort, den Bausteinen für Gruppen und Szenen, und (globalen) externen Variablen für die Taster

```
VAR
	Joblist_1: FbDALI_Joblist;
	Antwort_1: BYTE;
	Dimmen_Gruppe_1: FbDALI_DimmSingleButton;
	Dimmen_Gruppe_2: FbDALI_DimmSingleButton;

	Szene_Gruppe_1: FbDALI_RecallScene;
	Szene_Gruppe_2: FbDALI_RecallScene;
END_VAR
VAR_EXTERNAL
	xUL,xLL,xUR,xLR: BOOL;
END_VAR
```

Die Ausführung ruft einfach nur die entsprechenden Bausteine auf:

```
(* Kommunikationsschnittstelle für die erste DALI-Klemme  750-641 *)
Joblist_1(bModule_750_641:=1);

Dimmen_Gruppe_1(bAddress :=1,xGroup := TRUE,
	bReferenceaddress1 :=1,
	xButton := xUL,
	bSwitchOnLevel := 50, xOff_at_MinLevel := TRUE,
	bModule_750_641 := 1);
Dimmen_Gruppe_2(bAddress:=2,xGroup := TRUE,
	bReferenceaddress1 :=4,
	xButton := xUR,
	bSwitchOnLevel := 50, xOff_at_MinLevel := TRUE,
	bModule_750_641 := 1);

Szene_Gruppe_1(bAddress :=1, xGroup := TRUE,
	xSceneButton := xLL, bSceneNo := 1,
	bFadeTime := 2, bModule_750_641 := 1);
Szene_Gruppe_2(bAddress :=2, xGroup := TRUE,
	xSceneButton := xLR, bSceneNo := 2,
	bFadeTime := 2, bModule_750_641 := 1);
```

Code 6: Beispiel einer einfachen Steuerung von DALI-Akoren

3.3 Konfiguration eines DALI-Busses mit der WAGO Klemme 750-641 (DALI-Masterklemme)

Zur Konfiguration des DALI-Busses stehen zwei Varianten zur Verfügung: Das DALI Konfigurationstool ist ein Funktionsbaustein, der über die Visualisierung `StartseiteDALI`

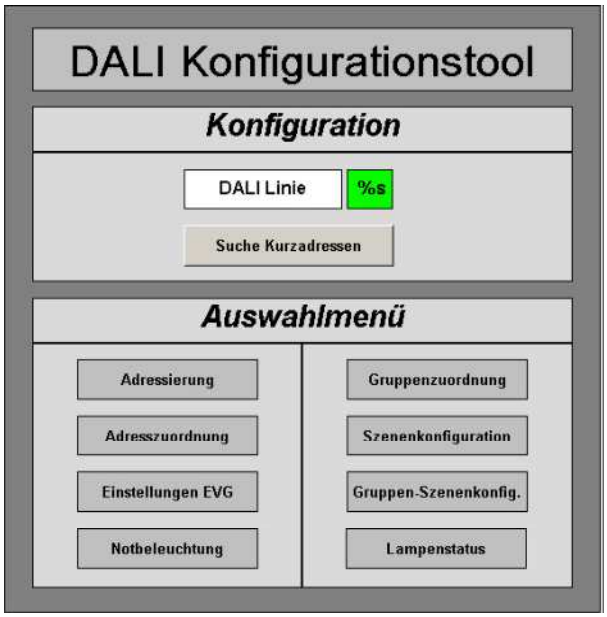


Abbildung 6: Startseite des DALI Konfigurationstools als Visualisierung

den Bus mit einer Art Management-Tool konfiguriert[21]: Adressen der elektronischen Vorschaltgeräte (EVG) können erkannt und von Hand sortiert werden[22], Einstellungen der EVGs können geändert werden und es können bis zu 16 Gruppen definiert werden. Szenen werden entweder für einzelne EVGs oder aber für alle Teilnehmer einer Gruppe gemeinsam definiert.

Zur freien Programmierung der Konfiguration stellt die Bibliothek `DALI_02_d.lib` unter Anderem folgende Bausteine zur Verfügung:

- Der Baustein `FbDALI_ConfigScene` dient zur Abspeicherung neuer Szenen: Die ersten zwei Eingänge dienen zur Adressauswahl, `bSceneNo` zur Auswahl der Szenennummer und `bDimmValue` zur Definition des Dimmwerts. Eine steigende Flanke an `xSet` bewirkt die Speicherung der Szene. Soll der aktuelle Wert als Szenewert gespeichert werden, so wird eine steigende Flanke am Eingang `xActualValueAsScene` benötigt.

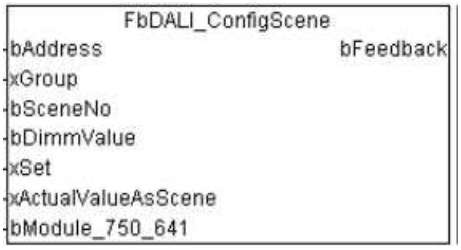


Abbildung 7: Funktionsbaustein `FbDALI_ConfigScene`

- Die Bausteine FbDALI_ConfigShortAddress und FbDALI_ShowShortAdr dienen zur Adressierung bzw. zur Suche von Kurzadressen.

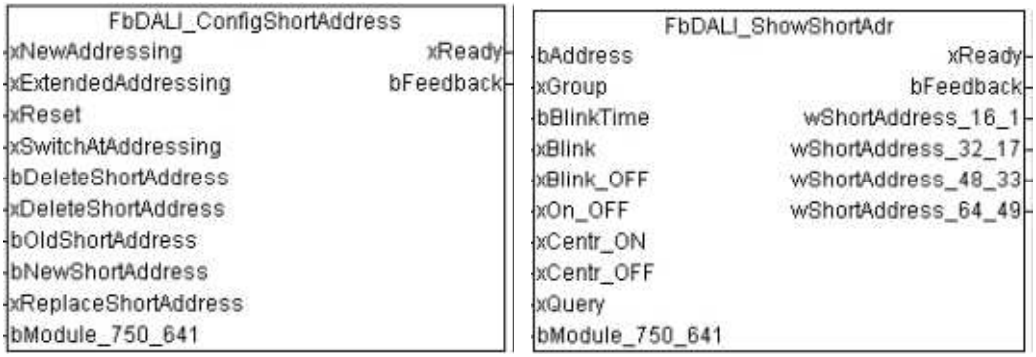


Abbildung 8: Funktionsbausteine `FbDALI_ConfigShortAddress` und `FbDALI_ShowShortAdr`

- Mit dem Baustein FbDALI_ConfigDevice können die Parameter einzelner DALI-Geräte konfiguriert werden.

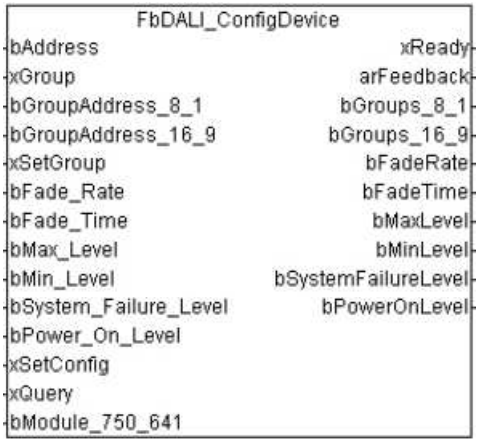


Abbildung 9: Funktionsbaustein `FbDALI_ConfigDevice`

[21] (WAGO, Konfiguration eines DALI-Beleuchtungssystems über die Visualisierung der WAGO-I/O-PRO, 2007). Für die Konfiguration eines DALI-Busses über eine Klemme 750-647 DALI-Multi-Master-Klemme steht ein etwas komfortableres Stand-alone Tool zur Verfügung, der WAGO DALI Configurator.

[22] Sollen die gefundenen Adressen sortiert werden, und dies wird in der Praxis fast immer der Fall sein, so können nur 63 Teilnehmer an einem DALI-Bus adressiert werden, da bei der Sortierung den Teilnehmern immer nur eine *freie* Adresse zwischen 1 und 64 zugewiesen werden kann.

3.4 Anwendung im Projekt

Insgesamt haben wir, lässt man die Nebenschauplätze Flur, Teeküche, Toiletten, etc. außen vor, 3 Lichtstromkreise/Lampen pro Büroachse, also insgesamt 120 Lampenaktoren. Hierfür reichen 2 DALI-Busse aus, ein weiterer kann für die Ansteuerung von Flur, Teeküche, Toiletten etc. verwendet werden. Um die Belastung des Datenbusses zu reduzieren und um für gemeinsames Einschalten von Lampengruppen auf Gruppen und Szenen zurückgegriffen zu können, werden mehr als die insgesamt 32 verfügbare Gruppen bei 2 DALI-Bussen (die Anzahl der verfügbaren Szenen ist weniger wichtig) benötigt, denn es sollen mindestens alle 3 Lichtstromkreise einer Büroachse in einer individuellen Gruppe zusammengefasst werden. Damit sind, aus Symmetriegründen, mindestens 4 DALI-Busse aufzubauen.

In der Realisierung des Projekts haben wir uns für 5 Busse entschieden, und steuern jeweils die Lichtfunktionen von 8 Raumachsen über einen DALI-Bus. Hiermit stehen pro Raumachse (mindestens) 2 Gruppenadressen und 2 Szenen zur Verfügung. Hierbei wird eine Gruppenadresse fest für die Zusammenfassung der 3 Lichtstromkreise der Raumachse vergeben, die anderen Gruppenadressen sind frei verfügbar.

Da eine WAGO-SPS nur maximal 5 DALI-Klemmen verwalten kann, können wir damit die Lichtsteuerung der verbleibenden Räume nicht mit DALI realisieren, sondern müssen konventionell arbeiten (d.h., Ansteuerung von Schaltaktoren über digitale Ausgangsklemmen, Nutzen von 1-10 V Schnittstellen bei Dimmfunktionen oder Einsatz einer weiteren SPS mit DALI-Bus zur Steuerung der restlichen Lichtfunktionen, gegebenenfalls für mehrere Etagen).

Zur Definition von Gruppen und Szenen und zum Monitoring greifen wir erst mal auf die vorhandene WAGO-Visualisierung zurück. Die Definition der Schaltfunktion (welche Wippe löst welche DALI-Befehle auf welchen Bussen aus) geschieht im eigenen Managementtool.
Folgende Funktionen stehen zur Verfügung:

- Dimmen (einzeln oder Gruppe)
- Aufruf Szene (einzeln oder Gruppe)

Da bei Dimmfunktionen die Länge des Tastendrucks entscheidend ist, wir aber nicht wissen, wie viele der Wippen (insgesamt 80 Taster mal 4 Wippen) eine Dimmfunktion steuern, müssten wir maximal 320 Funktionsbausteine für Dimmfunktionen bereitstellen[23], was sich etwas reduzieren lässt: Wir fügen zu jedem Lichtstromkreis eine Dimmfunktion hinzu (`FbDALI_DimmSingleButton`) und eine Liste der Wippen, die diesen Lichtstromkreis ansteuern.[24] Um das Problem der inkonsistenten Ansteuerung bei Gruppendimmen zumindest teilweise zu umgehen, fügen wir jeder Büroachse noch einen Dimmerbaustein hinzu, der alle 3 Lichtstromkreise gemeinsam dimmt.

Der Aufruf von Szenen gestaltet sich hingegen ziemlich einfach, da Szenenaufrufe nur kurze Schaltbefehle sind: Hierzu reicht es, für jede der 5 DALI-Klemmen einen Baustein `FbDALI_RecallScene` zu haben, der bei entsprechendem Tastendruck das Kommando für die Szenenauswahl auf den DALI-Bus schickt.[25]

---

[23] 40 Büroachsen x 2 4-fach Taster.

[24] Die hier gewählte Ansteuerung funktioniert nicht immer wie erwartet: Ist in einem Büro bestehend aus einer Büroachse ein 4-fach Taster hinterlegt mit 4 Dimmfunktionen (dreimal dimmen der einzelnen Lichtstromkreise, einmal gemeinsames Dimmen), so kann es zu folgender „Fehlfunktion“ kommen: Die drei Lichtstromkreise werden gemeinsam eingeschalten, Lichtstromkreis 1 wird herunter gedimmt, und nun möchte man alle drei Lichtstromkreise gemeinsam herunter dimmen: Während die Lichtstromkreise 2 und 3 korrekt herunter gedimmt werden, wird Lichtstromkreis 1 herauf gedimmt: Dies liegt daran das sich der Funktionsbaustein `FbDALI_DimmSingleButton`, da wir mit Einzeltasten dimmen, merken muss, ob beim nächsten Tastendruck auf- oder abgedimmt werden soll. Neben dem pragmatischen Einsatz der Funktionen gibt es nur „teure“ Abhilfen: Zusätzliche Dimmbausteine für alle Gruppen, Dimmbausteine für alle Wippen (dies ist „teurer“ als nur die oben angedeuteten 160 Bausteine, da eine Wippe unter Umständen Schaltfunktionen in mehreren DALI-Bussen auslösen muss); alternativ müssen die jeweiligen Zustände der einzelnen betroffenen Aktoren bei Gruppen-Dimmbefehlen erste synchronisiert werden. Eine weitere (einfache) Abhilfe ist die Verwendung von Doppelwippen für Dimmfunktionen (Dimmen auf und Dimmen ab mit separaten Wippen). Der Preis hier ist die Anzahl der benötigten Lichttaster.

[25] Auch hier gibt es ein kleines Problem: Da der Baustein flankengesteuert ist, gleichzeitig aber viele Wippen bedienen soll, muss der Eingang noch innerhalb eines Zyklus nach jeder Auswertung einer Wippe mit

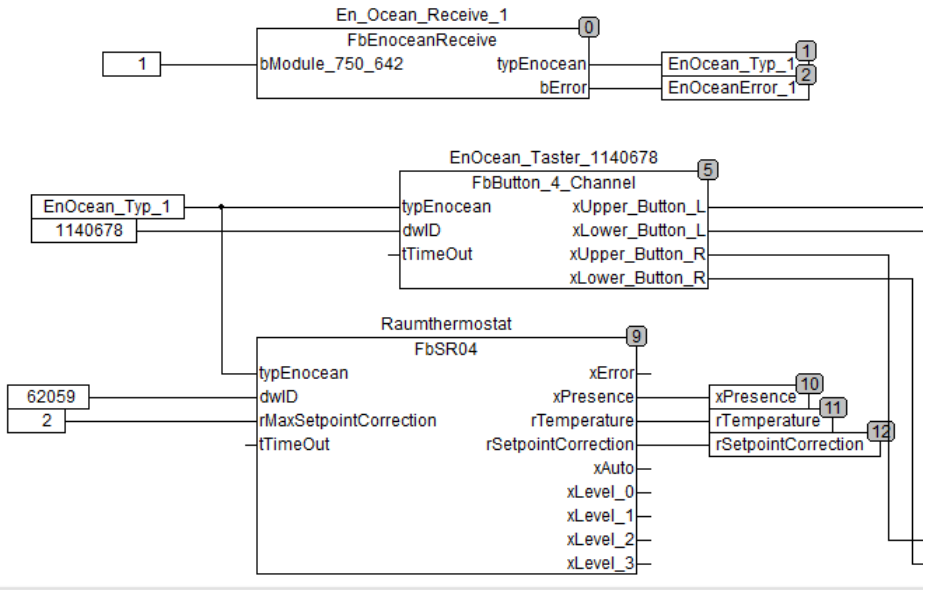
Szenefunktion auf 0 gesetzt werden (damit bei der Auswertung der nächsten Wippe eine steigende Flanke erkannt werden kann). Wird nun eine Wippe mit Szenefunktion längere Zeit gedrückt, so wird in jedem Zyklus ein Telegramm zum Setzten der Szene geschickt (d.h., durch das Überladen des Eingangs kann der Flankenanstieg nicht mehr erkannt werden, und der Baustein reagiert auf eine 1 am Eingang, d.h., eine gedrückte Taste mit Szenefunktion. Entweder man ignoriert dies, oder man schaltet zwischen dem Eingang des Bausteins `FbDALI-RecallScene` einen Baustein, der auf steigende Flanken nur für einen Zyklus eine 1 ausgibt (`R_TRIG`), bzw., um die Anzahl der Bausteine klein zu halten, baut, für jede Lichtschalterwippe, einen solchen Triggerbaustein nach indem man den Wippenzustand des letzten Zyklus speichert und auswertet.

4. EnOcean

4.1 Anbindung von EnOcean an die WAGO SPS

Mit der Funkempfänger-Busklemme 750-642 können EnOcean Funktelegramme empfangen und mit einer WAGO SPS ausgewertet werden. Pro Klemme können ca. 100 Sender (Sensoren) verwaltet werden.[26] Zur Kommunikation und Auswertung der Datentelegramme stehen zwei sehr unterschiedliche Bibliotheken zur Verfügung: Die Programmbibliothek `EnOcean_04.lib`[27] ist die ältere der Bibliotheken. Die Struktur der Bibliothek ist geräteorientiert, d.h., zu jedem (komplexeren) Gerät eines Herstellers gibt es einen Funktionsbaustein, der genau die gesendeten Daten dieses Gerätes bereitstellt.

Die neue Bibliothek `EnOcean_05.lib`[28] von 2013 orientiert sich an sogenannten EnOcean Equipment Profilen (EEP)[29], welche jedem Gerät zugeordnet sind. Für alle (wichtigen) EEPs gibt es dann einen Funktionsbaustein, der die gesendeten Daten des Geräts bereitstellt.



Code 7: Auswertung eines 4-fach Tasters und eines Raumthermostats SR04PT mit der Programmbibliothek `EnOcean_04.lib`

Die Abbildung zeigt beispielhaft bei Verwendung der Bibliothek `EnOcean_04.lib` die Auswertung eines 4-fach Tasters (ID 1140678) und eines Raumthermostats SR04PT[30] (ID 62059) der Firma thermokon mit Ist-Temperatur, Drehregler zur Beeinflussung der Solltemperatur und Präsenstaste.

[26] (WAGO, Datenblatt zu 750-642, Funkempfänger-Busklemme, 2015).
[27] (WAGO, Bausteinbeschreibungen für Enocean Funkempfänger 750-642, 2008).
[28] (WAGO, EnOcean Equipment Profile (EEP) - Anbindung von EnOcean-Funk-Sensoren/Aktoren unter Verwendung WAGO-EnOcean-Bibliothek, 2013)
[29] Ein EEP besteht aus dem Telegrammtyp (ORG bzw. RORG) mit Werten zwischen (hexadezimal) 00 und FF, der Funktionalität (FUNC) mit Werten zwischen 00 und 3F, und den Eigenschaften des Gerätetyps (TYPE) mit Werten zwischen 00 und 7F, vgl. (WAGO, EnOcean Equipment Profile (EEP) - Anbindung von EnOcean-Funk-Sensoren/Aktoren unter Verwendung WAGO-EnOcean-Bibliothek, 2013) S. 8.
[30] (thermokron GmbH, SR04 Funk-Raumtemperaturfühler).

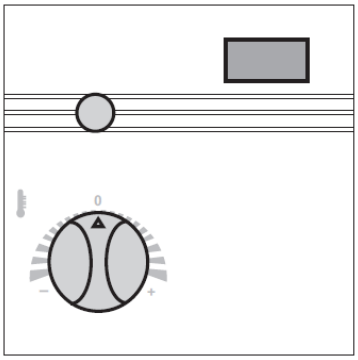
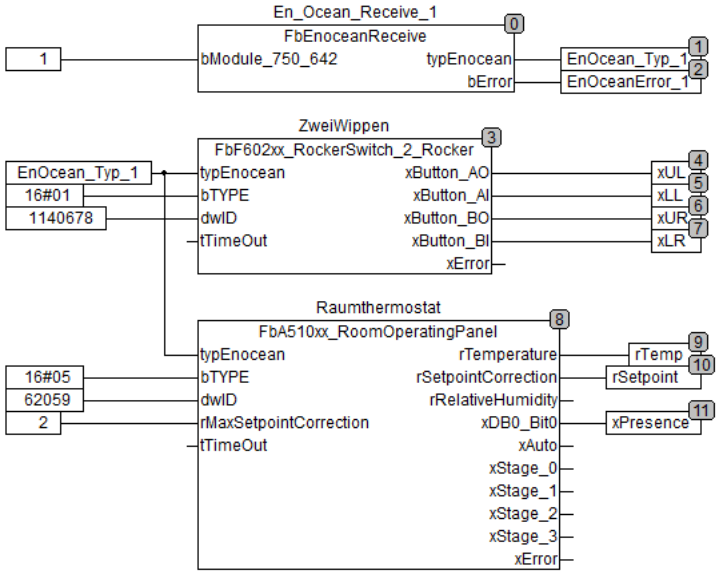


Abbildung 10: Skizze Raumtemperaturfühler SR04PT, Quelle: Datenblatt zu SR04, vgl. Literatur- und Quellenangaben

Die Kommunikation mit der ersten WAGO-Klemme (`bModule_750_642 = 1`) übernimmt der Baustein `FbEnoceanReceive`. Die Auswertung der zwei Sensoren übernehmen die Bausteine `FbButton_4_Channel` für den 4-fach Taster, und der Baustein `FbSR04` für das Raumthermostat. Der Ausgang `xPresence` gibt den Zustand des Anwesenheitstasters wieder, `rTemperature` ist die gemessene Raumtemperatur in Grad Celsius, und `rSetPointCorrection` gibt die Stellung des Drehreglers wieder, skaliert auf ±`rMaxSetpointCorrection`, hier also ±2. Wie im Code deutlich wird, trägt der Funktionsbaustein (nicht seine Instanz) den Namen FbSR04, und ist damit nur zur Auswertung von Raumthermostaten der Serie SR04 der Firma thermokon geeignet.

Bei Benutzung der Bibliothek `EnOcean_05.lib` sieht die Auswertung sehr ähnlich aus,



Code 8: Auswertung eines 4-fach Tasters und eines Raumthermostats SR04PT mit der Programmbibliothek `EnOcean_05.lib`

allerdings orientieren sich die Namen der Funktionsbausteine jetzt am EnOcean Equipment Profile (EEP), und der Typ muss des Sensors muss entweder aus Datenblättern oder unter Zuhilfenahme der Beschreibung der Bibliothek `EnOcean_05.lib` ermittelt werden[31].

4.2 Anwendung im Projekt

Wir werden sowohl die lokale Jalousiesteuerung wie auch die lokale Lichtsteuerung über EnOcean 4-fach Taster realisieren, für die Einzelraumsteuerung verwenden wir Raumthermostate, die dem EnOcean Equipment Profil[32] RORG = A5, FUNC = 10 entsprechen. Die Geräte dieses Typs sind Raumthermostate mit unterschiedlichen Funktionen und Temperaturbereichen. Der Typ 05 entspricht Raumthermostaten mit Anwesenheitstaster und einem Temperaturfühler für den Bereich zwischen 0

[31] (WAGO, EnOcean Equipment Profile (EEP) - Anbindung von EnOcean-Funk-Sensoren/Aktoren unter Verwendung WAGO-EnOcean-Bibliothek, 2013), S. 9ff.
[32] Vgl. (EnOcean Alliance 2011).

und 40°C, und einer skalierbaren Sollwertverstellung (Drehknopf). Durch Verwendung der Programmbibliothek `EnOcean_05.lib` bleibt man in Grenzen Hersteller-unabhängig, solange das entsprechende EEP unterstützt wird.

Wir beschränken uns, wie in Abschnitt 0 beschrieben, auf maximal
- 1 Raumthermostat pro Büro (und damit auch pro Raumachse)
- 2 4-fach Taster für die Jalousiesteuerung pro Raumachse
- 2 4-fach Taster für die Lichtsteuerung pro Raumachse

Damit gibt es ca. 200 EnOcean Sensoren und wir benötigen mindestens 2 EnOcean Klemmen. Um noch eine gewisse Skalierbarkeit zu erhalten, werden bei der Initialisierung die ersten 20 Raumachsen der ersten Klemme, die zweiten 20 Raumachsen der zweiten Klemme zugeordnet.

Nach Umbaumaßnahmen oder bei Bedarf wird im Managementtool ein EnOcean Sensor aktiviert durch Eingabe seiner EnOcean-ID. Jalousietaster bedienen damit automatisch alle Jalousien (linke Wippen außen, rechte Wippen innen), die zum Büro gehören, dem die Büroachse zugeordnet ist. Bei Lichttastern muss im Managementtool zusätzlich zu jeder der 4 Wippen festgelegt werden, welche Lichtfunktion ausgeführt werden soll. Mögliche Lichtfunktionen einer Wippe sind
- (Dimmen Einzellampe)*
- (Dimmen Gruppe)*
- (Schalten Szene)*

wobei der * andeutet, das eine Funktion einer Wippe aus mehreren (gleichartigen) Einzelfunktionen zusammengesetzt sein kann, wobei sich die Einzelfunktionen auch auf mehrere DALI-Busse beziehen können.

Beim Abspeichern der Programme im Managementtool werden die Programme analysiert und es werden den jeweiligen Dimmaktoren die Taster und Wippen zugeordnet, die den Dimmaktor beeinflussen. Beim Abarbeiten der Dimmfunktion werden die dem Dimmerbaustein zugeordneten Wippen der Reihe nach abgefragt und, als gemeinsames logisches Oder, als Steuerbefehl verwendet.

Lichtfunktionen (einer Wippe zugeordnete Programme) haben eine der drei folgenden Formen (Dimmen einfach, Dimmen Gruppe, Setzten Szene):
- DE:(Adresse.DALI_Bus;)*
- DG:(Adresse.DALI_Bus;)*
- SS:(Adresse.xGruppe.Szene.DALI_Bus;)*

wobei
- Adresse immer zweistellig zwischen 01 und 64 ist (bei xGruppe = 1 zwischen 01 und 16),
- xGruppe 0 oder 1 ist,
- Szene immer zweistellig zwischen 01 und 16 ist,
- DALI_Bus immer einstellig zwischen 1 und 5 ist.

Ein Szene-Programmteil hat damit immer 10 Stellen, ein Dimm-Programmteil 5 Stellen.

Bei der Initialisierung erhalten die Lichtstromkreise die Nummern (für k = 1 .. 40)
- 8*((k-1)%8)+1, 8((k-1)%8)+2, 8((k-1)%8)+3 auf dem DALI_Bus (k-1)/5+1, wobei die Division bzw. Modulsbildung als Integer-Operationen zu verstehen sind.[33]

Die in den Achsen zusammengefassten Lampenaktoren gehören, für k = 1..40, zur Gruppe
- 2*((k-1)%8)+1 auf dem DALI_Bus (k-1)/5+1.[34]

Beim Speichern der Wippenfunktion werden die Wippen den entsprechenden Dimmerbausteinen zugeordnet bzw. als Szene-Wippen global gespeichert.

[33] Dies sind die Adressen 8*((1-1)%8)+1 = 1, 2, 3, 8*((2-1)%8)+1 = 9, 10, 11, 8*((3-1)%8)+1 = 17, 18, 19, etc. auf den Bussen 1 bis 5.

[34] Dies sind die Gruppenadressen 2*((1-1)%8)+1 = 1, 2*((2-1)%8)+1 = 3, …, 2*((7-1)%8)+1 = 13, 2*((8-1)%8)+1 = 15, 2*((9-1)%8)+1 = 1, etc.

5. Flexible Programmierung

5.1 Pointer

Zeiger oder Pointer sind Variablen, die Speicheradressen halten (Referenzen). Pointer haben immer einen Typ, in CoDeSys V2.3 zum Beispiel

```
iINT1,iINT1: INT;
pINT1,pINT2: POINTER TO INT;
…
pINT1 := ADR(iINT1); (* Adresszuweisung *)
iINT2 := pINT2^; (* Dereferenzieren auf den Inhalt *)
```

Code 9: Syntax und Gebrauch von Pointer-Variablen in CoDeSys

Während bei den grundständigen Datentypen Variablen „echte“ Variablen sind, so sind in modernen Programmiersprachen Klassenvariablen immer Pointervariablen, auch wenn sich dies in der Syntax nicht unbedingt wiederspiegelt.

Pointer können[35] nicht remament gespeichert werden, da der Inhalt der Pointervariable eine Speicheradresse ist und es keine Garantie gibt, dass eine Variable bei einem Neustart eines Programms wieder an derselben Stelle im Speicher liegt.

5.2 Dynamische Speicherverwaltung

Auch mit beschränkten Mitteln ist es eigentlich relativ einfach, eine dynamische Speicherverwaltung selbst zu schreiben. Die einfachste dynamische Speicherverwaltung ist für homogene Daten, z.B. Zeichenketten (STRING), und besteht aus dem Reservieren eines genügend großen Segments im Speicher in der Form zweier Arrays,

```
Daten: ARRAY[1..Size] OF STRING;
Frei: ARRAY[1..Size] OF BOOL;
```

Code 10: Einfache Speicherverwaltung (homogene Daten), Datendeklaration

wobei das erste Array die Daten hält und das zweite Array als Merker funktioniert, welche Zellen vergeben wurden. Wird neue Speicher angefragt, so wird in `Frei` nach der ersten freien (nicht-belegten) Zelle gesucht, und z.B. die Adresse des entsprechenden Datensegments zurückgegeben. Etwas feiner und schneller geht es, wenn man zusätzlich eine (verkettete) Liste aktualisiert, die die freien Zellen enthält.

```
TYPE FreeListElement :
STRUCT
      xFrei: BOOL;
      iNext: INT; (* "POINTER" TO ListElement*)
END_STRUCT
END_TYPE

VAR
Daten: ARRAY[1..Size] OF STRING;
Frei: ARRAY[1..Size] OF FreeListElement;
First: iListElement; (* Pointer auf erstes freie Listelement *)
END_VAR
```

Code 11: Einfache Speicherverwaltung (homogene Daten), Listen als Arrays

Bei der Initialisierung ist `First = 1` und für alle Listelemente gilt `Frei[n].iNext = n+1`. Außerdem gilt `Frei[n].xFrei := TRUE`. Die Vergabe von Speicherplatz läuft nach dem Schema

[35] Können im Sinne von sollen bzw. im Sinne von „es macht wenig Sinn“.

```
IF First = 0 THEN RETURN 0: (* Null-Pointer *)
ELSE
       iReturn := First;
       First := First.iNext;
       Frei[iReturn].xFrei = FALSE;
       Return ADR(Daten[iReturn]);
END_IF
```
Code 12: Einfache Speicherverwaltung (homogene Daten), Vergabe von Speicher

Umgekehrt verläuft die Freigabe von Speicher: Zur Speicheradresse `pFreeMe` ist der passende Index im Array Daten zu berechnen, und die Speicherzelle wird wieder freigegeben:

```
iIndex := ADR(Daten[0]) + (ADR(pFreeMe)-ADR(Daten[0]))/Size(Daten[0]);
IF pFreeME = ADR(Daten[iIndex]) THEN Frei[iIndex].xFrei := TRUE;
ELSE (* Error handling *)
END_IF
```
Code 13: Einfache Speicherverwaltung (homogene Daten), Freigabe von Speicher

Bei dieser Implementierung gibt es zwei Probleme: Zum einen lassen sich nur Daten gleicher Größe (und gleichen Typs) in den Zellen ablegen, insbesondere ist der Bereich des Speichers für Daten anderer Größe nicht nutzbar, zum anderen gibt es keinen Mechanismus, nicht benötigten Speicher, der vergessen wurde freizugeben, automatisch wieder zu befreien.[36] Ein Vorteil der obigen Implementierung (genauer, der Homogenität der Daten) ist, dass der Speicher unmöglich fragmentieren kann.

Will man Speicher für Daten unterschiedlicher Größe verwalten, so arbeitet man am besten mit einer Adressierung auf zwei Ebenen: Es wird ein (jetzt unstrukturierter) Datenbereich reserviert, und es gibt zwei Listen von Pointern:

```
aDataPointer: ARRAY[0..iMaxDataPointer] OF DWORD; (* Array von Pointern *)
aFrei: ARRAY[0..iMaxDataPointer] OF BOOL;
Data: ARRAY[0..iMaxData] OF BOOL; (* reservierter Datenspeicher *)
```
Code 14: Datentyp für Speicherverwaltung (inhomogene Daten) mit Möglichkeit zur Defragmentierung

Über `iMaxData` wird die Speichergröße definiert, Zellen werden aber nie direkt angesprochen. Im Array `aDataPointer` stehen die Pointer auf Adressen in Data. Der Zugriff auf Daten erfolgt über *doppelte Referenzierung*, d.h., ein Pointer zeigt auf eine Speicherzelle in `aDataPointer`, und hier steht der Pointer auf ein Datensegment in Data.
Mit dieser Struktur ist es möglich, den Speicherbereich Data zu defragmentieren, da zum Defragmentieren nur die Adressen im Array `aDataPointer` geändert werden müssen.

5.3 (Verkettete) Listen

Eine Liste mit Listelementen aus einer Menge M ist eine Implementierung des Modells beschreiben durch die Gleichung

List ::= ε | m List

d.h., eine Liste ist entweder leer (ε) oder besteht aus einem Element m aus M und, daran angehängt, einer weiteren Liste.[37] In modernen Programmiersprachen gibt es entweder einen Datentyp Liste, der über einen anderen Datentyp instanziiert werden kann, oder Listen werden als einfach- oder zweifach verkettete Listen in Eigenregie implementiert.[38] Bei der einfach-verketteten Liste geschieht dies durch die Klasse

```
class Listelement{
data: M;                    (* Wir bleiben hier bei der Menge der Daten M *)
```

[36] Das Stichwort hier ist automatic garbage collection, wie es zum Beispiel in Laufzeitumgebungen der Programmiersprache Java implementiert ist. Automatic garbage collection ist hochgradig nicht-trivial, da anhand von (automatischen) Code-Analysen festgestellt werden muss, ob ein bestimmter Pointer nochmal aufgerufen werden kann oder nicht.
[37] Gleichungen dieser Art werden in der Semantik von Programmiersprachen über sogenannten Domänen (engl. Domains) gelöst, vgl. (Stoltenberg-Hansen & Lindström, 1994) oder (Winskel, 1993).
[38] Vgl. zum Beispiel (Cormen, Leiserson, Rivest, & Stein, 2001), Kapitel 10.2.

```
next: POINTER TO Listelement;     (* Zeiger auf das nächste Listelement *)
…                                 (* Methoden *)
}
```

Code 15: Implementierung einer verketteten Liste mit Pointern

wobei eine Liste ein Pointer `first` auf das erste Listelement ist. Ist der Pointer `first` der Null-Pointer, so ist die Liste leer.

Verkettete Listen sind in CoDeSys verboten, und der Versuch, eine verkette Liste über den Integern als

```
TYPE ListElement :
STRUCT
      iData: INT;
      pNext: POINTER TO ListElement;
END_STRUCT
END_TYPE
```

Code 16: CoDeSys ListElement-Implementierung, die zu einer Fehlermeldung führt

zu definieren, führt zu einer Fehlermeldung[39]. Es ist allerdings möglich, CoDeSys zu überlisten und mit einem Trick verkettete Listen doch zu implementieren.[40]

Gerade bei vergleichsweise kurzen Listen ist die Dynamik der Listen nicht unbedingt ein Kriterium für die Auswahl des Datentyps Liste, und oftmals werden Listen deren maximale Länge bekannt ist auch als Felder (Arrays) implementiert. Der folgende Code zeigt die Implementierung einer verketteten Liste durch ein Array, wobei die Listenelemente in beliebigen Zellen des Arrays in beliebiger Reihenfolge abgelegt werden können, hier, zur deutlichen Trennung zwischen dem Zellinhalt (`data`) und der Zeigerfunktion (`next`) als Liste von Strings:

```
TYPE ListElement :
STRUCT
      sData: STRING;
      pNext: INT; (* Pointer-Funktion *)
END_STRUCT
END_TYPE

VAR CONSTANT
iMaxLength: INT := 64; (* Beispiel *)
END_VAR

VAR
aListOfString := ARRAY[1..iMaxLength] OF ListElement;
pFirst: INT;
END_VAR
```

Code 17: CoDeSys Listenimplementierung mit Array

Die Variable `pFirst` hält also die Feldnummer des Arrays, in der das erste Element der Liste steht. Eine 0 zeigt an, dass der „Pointer“ der „Null-Pointer“ ist und die Liste leer ist. Ist die Liste gefüllt, und ist `le` ein „Pointer“ (eine Integerzahl) auf ein Listenelement, so enthält `aListOfString[le].sData` die Daten (eine Zeichenkette), und `aListOfString[le].next` die Adresse (Feldnummer, „Pointer“) des

[39] Fehler 3704: Datenrekursion: LISTELEMENT -> LISTELEMENT.

[40] Auf 32-Bit Maschinen werden Pointer, egal welchen Typs, als DWORD realisiert. Daher kann eine POINTER-Variable auch als DWORD-Variable deklariert werden, d.h., die Implementierung

```
TYPE ListElement :
STRUCT
      iData: INT;
      pNext: DWORD; (* anstatt POINTER TO ListElement *)
END_STRUCT
END_TYPE
```

ist eine Implementierung einer verketteten Liste, die auch funktioniert. Auf 64-Bit Maschinen muss `pNext` entsprechend als `LWORD` deklariert werden.
Die hier vorgestellte Implementierung sollte aber vermieden werden.

nächsten Listenelements. Mit solchen Datenstrukturen lassen sich sogar gleichzeitig mehrere verkettete Listen verwalten falls jedes Datenelement nur in maximal einer Liste auftauchen kann.[41]

5.4 Anwendung im Projekt

Wir haben uns am Ende *gegen* die Nutzung der oben vorgestellten Methoden der flexiblen Programmierung entschieden: Eine eigene Speicherverwaltung käme nur in Frage, wenn der reservierte Speicherbereich im Remanent-Speicher liegen würde. Aber auch dann ist der Vorteil gegenüber der gewählten Implementierung gering: Auch bei der eigenen Speicherverwaltung muss der Speicher so groß sein, das er alle Daten potentieller Geräte aufnehmen kann. Dann kann man die maximale Anzahl der Geräte aber auch im Voraus definieren und dann nur den benötigten Teil benutzen. Extra Flexibilität würde sich noch bei der Zuordnung der Geräte zu den Räumen ergeben: Bei einer flexibleren Programmierung wäre die Gesamtzahl der Geräte (z.B. 80 Jalousietaster) konstant und unveränderbar, aber man hätte volle Flexibilität, wie viele Geräte man einem (kleinen) Büro zuordnet. Da die Implementierung die einzelnen Büroachsen aber durchaus üppig ausstattet, wird diese Extraflexibilität nicht wirklich benötigt.

Im folgenden Kapitel werden wir die einzelnen Steuerungsaufgaben

- Klima (Heizung, Kühlung)
- Jalousien (innen, außen)
- Licht

behandeln und zeigen, wie diese innerhalb der gewählten Datenstrukturen realisiert werden können.

[41] Ein Beispiel wäre eine Liste (ein Pool) von Schaltern, wobei ein Schalter maximal einem Büro zugeordnet ist, d.h., jedes Büro hat eine lokale Variable `pFirst`, die ein Pointer (Zellindex) des ersten dem Büro zugeordneten Schalters ist.

6. Implementierung der Steuerung

Die Daten der Büroetage werden in einem Funktionsbaustein `FB_Bueros_und_Steuerung` gehalten, der die einzelnen Daten für Gang (Flur), Teeküche, Toiletten und dem Büroflur bestehend aus 40 Achsen zusammenfaßt. Über globale Konstanten werde die grundsätzlichen Parameter definiert, unter anderem[42]

```
VAR_GLOBAL CONSTANT
	iAnzahlBueroachsen: INT := 40;
	iAnzahlJalousieTaster: INT := 80;
	iAnzahlLichtTaster: INT := 80;
	iAnzahlEnOceanKlemmen: INT := 2;
	iAnzahlDALIKlemmen: INT := 5;		(* Achtung: Maximal 5 *)
END_VAR
```

Code 18: Globale Konstanten

Für alle Bereiche (Gang, Teeküche, Toiletten, Büroflur) gibt es Funktionen zur Initialisierung und zur Steuerung. Den Funktionen wird jeweils ein Pointer (Zeiger) auf die Daten übergeben um diese zu bearbeiten. Durch die Übergabe eines Pointers wird zum Einen die Größe der übergebenen Daten reduziert, zum Anderen sind Änderungen an den Daten auch nach Beendigung der Funktionen vorhanden.

Die Bausteine und Funktionen sind in einer Bibliothek zusammengefasst, und der Baustein `FB_Bueros_und_Steuerung` enthält sowohl die Daten als auch Initialisierung und Abarbeitung der Steuerung. Das eigentliche Programm besteht damit nur aus einer einzigen Instanz dieses Bausteins

```
PROGRAM PLC_PRG
VAR
	main: FB_Bueros_und_Steuerung;
END_VAR
main();
```

Code 19: Struktur des Hauptprogramms

Zur Dokumentation der einzelnen Bausteine und Funktionen verweisen wir auf den Anhang.

6.1 Steuerung der Klima-Funktionen

Jede Raumachse ist mit Stellantrieben für Heizung bzw. Klima ausgestattet. Jedem Büro wird ein Raumthermostat zugeordnet, und über dieses Thermostat werden alle Stellantriebe eines Büros gesteuert. Globale Variablen steuern den Tag/Nachtbetrieb, der Anwesenheitstaster am Raumthermostat steuert den Komfortbetrieb. Für die Einzelraumsteuerung greifen wir auf den Baustein Fb2PointSingleRoomController der WAGO Bibliothek `Building_HVAC_03.lib` zurück, der sowohl Heizung- und Kühlaktoren ansteuert, und die Betriebsarten Komfort, Stand-By, Nacht und Frost bereitstellt.[43]

In der Steuerung werden die Büroachsen von der höchsten zur niedrigsten Nummer durchlaufen und es werden die Klimafunktionen für Heizung und Kühlung bearbeitet. Die Klimadaten stammen von einem Raumthermostat, welches jeweils nach einer Trennwand – dies entspricht dem Beginn eines neuen Büros – ausgewertet wird.

```
FUNCTION SteuerungKlima : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n: INT;
	pRaumthermostat: POINTER TO TRaumthermostat;
	rTemperature: REAL;
	rSetpointCorrection: REAL;
	xComfortStandby: BOOL;
END_VAR
```

[42] Eigentlich sollte es genügen, die Anzahl der Büroachsen zu definieren, und die anderen Konstanten abzuleiten (zu berechnen). Leider ist es mit diesen abgeleiteten Konstanten nicht möglich, Arrays zu definieren.
[43] Vgl. (WAGO, Bausteinbeschreibungen für HLK-Funktionen - Bibliothek und Dokumentation, 2013).

```
n:= iAnzahlBueroachsen;

WHILE n > 0 DO
	IF pBueroFlur^.aTrennwaende[n] THEN
		(* Trennwand vorhanden, d.h., Index enhaelt Buerodaten *)
		pRaumthermostat := ADR(pBueroFlur^.aRaumthermostat[n]);
		(* Raumthermostat auslesen *)
		pRaumthermostat^.Raumthermostat(
		typEnocean :=
pBueroFlur^.aEnOceanReceive[pBueroFlur^.aBueroAchse[n].bModule_750_642].typEnoce
an,
		bType := pRaumthermostat^.bType,
		dwID := pRaumthermostat^.dwEnOceanID,
		rMaxSetpointCorrection :=
			pRaumthermostat^.rMaxSetpointCorrection);

		rTemperature := pRaumthermostat^.Raumthermostat.rTemperature;
		rSetpointCorrection :=
			pRaumthermostat^.Raumthermostat.rSetpointCorrection;
		xComfortStandby := pRaumthermostat^.Raumthermostat.xDB0_Bit0;
		(* Komfortbetrieb bei Anwesenheit *)
	END_IF;
	(* Controller auswerten und Aktoren setzten *)
	pBueroFlur^.aBueroAchse[n].KlimaController(
	rRoomTemperature := rTemperature,
	rSetpointCorrection := rSetpointCorrection,
	xComfortStandby := xComfortStandby,
	xNightMode := xGlobalNightMode
	);
	pBueroFlur^.aBueroAchse[n].xCooling :=
		pBueroFlur^.aBueroAchse[n].KlimaController.xCooling;
	pBueroFlur^.aBueroAchse[n].xHeating :=
		pBueroFlur^.aBueroAchse[n].KlimaController.xHeating;
	n:= n-1;
END_WHILE
```

Code 20: Implementierung der Klima-Funktion

6.2 Steuerung der Jalousie-Funktionen

Jede Raumachse hat die Breite eines Fensters, und das Fenster ist mit Außen- und mit Innenjalousien ausgestattet. Folgende Funktionen werden implementiert:

- Alle Außen- bzw. Innenjalousien eines Büros werden immer gemeinsam gesteuert.
- Lokal in den Büros ist die Ansteuerung der Jalousien immer über einen oder mehrere 4-fach Taster, wobei das linke Tastenpaar die Außenjalousien, das rechte die Innenjalousien steuert.
- Die Innenjalousien können zentral nicht gesteuert werden.
- Die Außenjalousien können zentral in die Sicherheitsposition gefahren werden oder zentral in eine vorgegebene Beschattungsposition.
- Sowohl Außen- wie auch Innenjalousien können zentral für Reinigungs- und Wartungsarbeiten getrennt in Sicherheitsposition gefahren werden und werden dort bis zur Freigabe gehalten.

Für die Implementierung werden jeder Büroachse zwei Funktionsbausteine `FbJalousie` der Bibliothek `Gebaeude_allgemein.lib` zugeordnet zur Steuerung der Jalousien.

Darüber hinaus gibt es globale Variablen zum Anfahren der Sicherheitsposition der Außenjalousien und der Innenjalousien, die entweder manuell gesetzt werden können (Innen- oder/und Außenjalousie) oder automatisch (Außenjalousie, z.B. gesteuert durch eine Wetterstation). Außerdem können die Innenjalousien, auch durch eine globale Variable gesteuert, in die Beschattungsposition gefahren werden.

Zur Steuerung der Jalousiefunktion eines Büros werden die Büroachsen von 40 bis 1 gleichzeitig zweimal parallel durchlaufen: Die erste Schleife wertet alle Taster aus und verknüpft diese mit einem logischen ODER bis eine Trennwand gefunden wird (damit werden alle Taster eines Büros bzw. deren Wippen zusammengefasst), die zweite Schleife läuft nun nach, und steuert in den in allen dem Büro zugeordneten Büroachsen die Stellantriebe für Innen- und Außenjalousien.

```
FUNCTION SteuerungJalousie : BOOL
VAR_INPUT
```

```
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n,n1: INT;
	m: INT;

	pJT: POINTER TO TJalousieTaster;
	pRS: POINTER TO FbF602xx_RockerSwitch_2_Rocker;
	xAup, xAdown, xIup, xIdown: BOOL;
END_VAR

n:= iAnzahlBueroachsen;
n1:= n;

WHILE n >= 0 DO
	IF pBueroFlur^.aTrennwaende[n] THEN
		(* Trennwand vorhanden, d.h., Index enthaelt Buerodaten *)
		(* letztes Buero Jalousie steuern *)
		IF n < iAnzahlBueroachsen THEN
			WHILE n1 >  n DO
			pBueroFlur^.aBueroAchse[n1].AJalousie(
				xJalousieTasterAuf := xAup,
				xJalousieTasterAb := xAdown,
				xBeschattungsPosAnfahren :=xABeschattungsPosAnfahren,
				xSicherheit := xASicherheit);

			pBueroFlur^.aBueroAchse[n1].IJalousie(
				xJalousieTasterAuf := xIup,
				xJalousieTasterAb := xIdown,
				xBeschattungsPosAnfahren :=xIBeschattungsPosAnfahren,
				xSicherheit := xISicherheit);

			pBueroFlur^.aBueroAchse[n1].xDoAJalousieAb :=
			pBueroFlur^.aBueroAchse[n1].AJalousie.xDoJalousieAb;
			pBueroFlur^.aBueroAchse[n1].xDoAJalousieAuf :=
			pBueroFlur^.aBueroAchse[n1].AJalousie.xDoJalousieAuf;
			pBueroFlur^.aBueroAchse[n1].xDoIJalousieAb :=
			pBueroFlur^.aBueroAchse[n1].IJalousie.xDoJalousieAb;
			pBueroFlur^.aBueroAchse[n1].xDoIJalousieAuf :=
			pBueroFlur^.aBueroAchse[n1].IJalousie.xDoJalousieAuf;
				n1 := n1-1;
			END_WHILE
		END_IF;
		xAup := FALSE;
		xAdown := FALSE;
		xIup := FALSE;
		xIdown := FALSE;
	END_IF;

	FOR m := 1 TO 2 DO
		pJT := ADR(pBueroFlur^.aJalousieTaster[n,m]);

		IF pJT^.xAngemeldet THEN
			pRS := ADR(pJT^.JalousieTaster);
			pBueroFlur^.aEnOceanReceive[n](
			bModule_750_642 :=
				pBueroFlur^.aBueroAchse[n].bModule_750_642);

			pRS^(typEnocean :=
				pBueroFlur^.aEnOceanReceive[n].typEnocean,
			bTYPE :=pJT^.bType,
			dwID := pJT^.dwEnOceanID
			);
			xAup := xAup OR pRS^.xButton_AO;
			xAdown := xAdown OR pRS^.xButton_AI;
			xIup := xIup OR pRS^.xButton_BO;
			xIdown := xIdown OR pRS^.xButton_BI;
		END_IF;
	END_FOR;

	n := n-1;
END_WHILE
```

Code 21: Steuerung der Jalousie-Funktion

6.3 Steuerung der Licht-Funktionen

Die Steuerung der Lichtfunktionen ist auf drei Funktionen verteilt: Dimmen ganzer Büroachsen, Dimmen einzelner Lampen, und der Aufruf von Lichtszenen.

Bei der Steuerung der Dimmfunktion ganzer Büroachsen ist jeder Büroachse ein Dimmerbaustein zugeordnet und eine Liste der Tasterwippen, die diese die Achse dimmen. Die Liste der Wippen ist ein String bestehend aus Folgen von 7 Zeichen und Ziffern,

Nummer Raumachse.Nummer Taster.Nummer Wippe;

wobei jeder Raumachse maximal zwei 4-fach Taster zugeordnet sind.[44] Die einzelnen Wippen werden abgerufen, mit ODER verknüpft, und auf den entsprechenden Dimmerbaustein als Eingang gelegt.

```
FUNCTION SteuerungLicht_DimmenBueroAchsen : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n: INT;
	xPush: BOOL;
	sListe: STRING;

	p: UINT; (* Anzahl angemeldeter Tasten *)
	NrT: INT;
	NrTT: INT;
	NrWippe: INT;
	pLT: POINTER TO TLichtTasterMitProgramm;
END_VAR

n:=1;

WHILE n <= iAnzahlBueroachsen DO
	IF pBueroFlur^.aBueroAchse[n].DimmenGruppe.sListe <>'' THEN
		sListe := pBueroFlur^.aBueroAchse[n].DimmenGruppe.sListe;
		xPush := FALSE;
		FOR p := 1 TO LEN(sListe)/7 DO(* Anzahl der angemeldeten Tasten *)
			NrT := STRING_TO_INT(MID(sListe,2,(p-1)*7+1));
			NrTT := STRING_TO_INT(MID(sListe,1,(p-1)*7+4));
			NrWippe := STRING_TO_INT(MID(sListe,1,(p-1)*7+6));
			(* Auswertung Taster und Wippe *)
			pLT := ADR(pBueroFlur^.aLichtTaster[NrT,NrTT]);
			pLT^.LichtTaster(typEnocean :=
pBueroFlur^.aEnOceanReceive[pBueroFlur^.aBueroAchse[NrT].bModule_750_642].typEnO
cean,
			bTYPE := pLT^.bType,
			dwID :=pLT^.dwEnOceanID
			);
			CASE NrWippe OF
			1: xPush := xPush OR pLT^.LichtTaster.xButton_AO;
			2: xPush := xPush OR pLT^.LichtTaster.xButton_AI;
			3: xPush := xPush OR pLT^.LichtTaster.xButton_BO;
			4: xPush := xPush OR pLT^.LichtTaster.xButton_BI;
			END_CASE
		END_FOR
		pBueroFlur^.aBueroAchse[n].DimmenGruppe.DimmAktor(
			xButton := xPush);
	END_IF

n := n+1;
END_WHILE
```

Code 22: Steuerung der Licht-Funktion: Dimmen von Büroachsen

[44] Die 80 vorhandenen Lichttaster mit jeweils 4 Wippen sind paarweise den 40 Raumachsen zugeordnet. Die Programme können aber beliebig sein, und selbst bei 40 Büros der Breite einer Raumachse ist es möglich, einer Wippe mit einen Taster der im Programm der Raumachse 20 zugeordnet ist Funktionen der Büros 1-4 zu steuern, und den Taster, der ja ein EnOcean-Taster ist, im Flur oder in Raumachse 40 zu platzieren. – Ob diese Platzierung in der Realität Sinn macht, ist eine andere Frage.

Die Zuordnung der Wippen zu den Dimmerbausteinen wird beim Abspeichern der Funktionen im Management-Tool vorgenommen, wobei im Management-Tool jeder *Wippe* eine Funktion zugeordnet wird (Dimmen Einzellampen, Dimmen Raumachsen, Aufruf Szenen). Der Vorteil dieser Zuordnung ist, das wir im Management-Tool natürlicherweise in „Funktionen von Wippen“ denken, während wir bei der Abarbeitung der Steuerung genau die Wippen auswerten, die benötigt werden, d.h., mit angemeldet und mit Schaltfunktionen belegt sind.

Die Implementierung der Dimmfunktion von Einzellampen ist analog, der Aufruf von Szenen ist sehr ähnlich: Jedem DALI-Bus ist ein Szenebaustein zugeordnet, der die eigentlichen Schaltbefehle auf den jeweiligen Bus schickt. Dem Baustein sind wieder Wippen zugeordnet, die Szenenaufrufe auslösen. Die Unterschiede zum Dimmen sind nun zweifach:

- Beim Szenenaufruf kommt es nicht auf die Dauer an, sondern das Programm reagiert auf positive Flanken der einzelnen Wippen.
- Den Szenebausteinen sind wieder Wippen zugeordnet, und für jede einzelne Wippe muß, bei positiver Flanke, nun das bei der Wippe hinterlegte Szeneprogramm abgearbeitet werden. Insgesamt können pro DALI-BUS 64 Wippen mit Szenefunktion definiert werden.

```
FUNCTION SteuerungLicht_SchaltenSzene : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n,m,o: INT;
	sWippe: STRING;
	NrT: INT;
	NrTT: INT;
	NrWippe: INT;
	pLT: POINTER TO TLichtTasterMitProgramm;
	xStatus: BOOL;
	Programm: STRING;
	bProgAddress: BYTE;
	xProgGroup: BOOL;
	bProgSceneNo: BYTE;
	bProgModule_750_641: BYTE;
	iProgModule_750_641: INT;
END_VAR

FOR n := 1 TO iAnzahlDALIKlemmen DO
	FOR m := 1 TO 64 DO
		sWippe := pBueroFlur^.aRegistrierteSzeneWippe[n,m];
		IF sWippe <> '' THEN
			NrT := STRING_TO_INT(MID(sWippe,2,1));
			NrTT := STRING_TO_INT(MID(sWippe,1,4));
			NrWippe := STRING_TO_INT(MID(sWippe,1,6));
			(* Auswertung Taster und Wippe *)
			pLT := ADR(pBueroFlur^.aLichtTaster[NrT,NrTT]);
			pLT^.LichtTaster(typEnocean :=
pBueroFlur^.aEnOceanReceive[pBueroFlur^.aBueroAchse[NrT].bModule_750_642].typEnO
cean,
				bTYPE := pLT^.bType,
				dwID :=pLT^.dwEnOceanID
				);
			CASE NrWippe OF
				1: xStatus := pLT^.LichtTaster.xButton_AO;
				2: xStatus := pLT^.LichtTaster.xButton_AI;
				3: xStatus := pLT^.LichtTaster.xButton_BO;
				4: xStatus := pLT^.LichtTaster.xButton_BI;
			END_CASE
			IF xStatus AND NOT pLT^.aAlterSchaltzustand[NrWippe] THEN
				(* Jetzt positive Flanke *)
				(* Also Programm abarbeiten *)
				Programm := pLT^.sProg[NrWippe];

				FOR o := 1 TO LEN(Programm)/10 DO
					bProgAddress :=
					STRING_TO_BYTE(MID(Programm,2,(o-1)*10+1));
					xProgGroup :=
```

```
                                    STRING_TO_BOOL(MID(Programm,1,(o-1)*10+4));
                                    bProgSceneNo :=
                                    STRING_TO_BYTE(MID(Programm,2,(o-1)*10+6));
                                    bProgModule_750_641 :=
                                    STRING_TO_BYTE(MID(Programm,1,(o-1)*10+9));
                                    iProgModule_750_641 :=
                                    BYTE_TO_INT(bProgModule_750_641);

                                    pBueroFlur^.aSzene[iProgModule_750_641](
                                    bAddress := bProgAddress,
                                    xGroup := xProgGroup,
                                    bSceneNo := bProgSceneNo,
                                    xSceneButton := TRUE,
                                    bModule_750_641 := bProgModule_750_641
                                    );
                                    pBueroFlur^.aSzene[iProgModule_750_641](
                                    xSceneButton := FALSE,
                                    bModule_750_641 := bProgModule_750_641
                                    );
                                END_FOR
                            END_IF
                            pLT^.aAlterSchaltzustand[NrWippe] := xStatus;
                        ELSE m:= 64;
                        END_IF
                    END_FOR
                END_FOR
```

6.4 Initialisierung

Für jeden der Bereiche Büroflur, Gang, Teeküche und Toiletten stehen Funktionen zur Initialisierung zur Verfügung. Im Wesentlichen werden hier globale, feste Parameter gesetzt (z.B. Nummern der zugeordneten DALI- und EnOcean-Klemmen, Zeitparameter für die Flur- und Toilettenbeleuchtung, etc.).

6.5 Steuerung von Flur, Teeküche und Toiletten

Zur Steuerung von Flur, d.h., dem Gangareal vor den Büroräumen, Teeküche und Toiletten verweisen wir auf die im Anhang wiedergegebene Dokumentation. An dieser Stelle sei nur so viel gesagt, dass der Flur mit Bewegungsmeldern und Tastern an den Enden des Flurs ausgestattet ist, die Teeküche einen einfachen Lichtstromkreis enthält, und die Toiletten auch beispielhaft mit jeweils einem Lichtstromkreis ausgestattet sind. Flur und Toiletten werden über Zeitschaltuhren ausgeschalten, bei den Toiletten wird das Ausschalten gegebenenfalls durch einen Anwesenheitssensor (Bewegungsmelder) unterbunden.

7. Visualisierung/Management Tool

Verbunden mit Programmierung ist die Aufgabe, ein Interface bereitzustellen, um zum Einen die laufende Anlage zu überwachen und um zum Anderen nach Umbaumaßnahmen die Raumthermostate und Taster für Licht und Jalousien neu zuordnen zu können.
Betrachtet man die 40 bis 50 Datenpunkte pro Büroachse und die Limitierung der WAGO-SPS bzw. von CoDeSys, nur ca. 100 Programmvariablen gleichzeitig überwachen zu können, so ist klar, dass man hier kreativ werden muss:

1. Man kann immer nur eine kleine Zahl von Büroachsen gleichzeitig visualisieren, unter Umständen nur eine oder zwei. Hierzu bietet sich ein „Sliding Window“ an, dass immer eine feste oder variable aber begrenzte Anzahl von nebeneinanderliegenden Büroachsen anzeigt.
2. Da man mindestens ein Büro mit all seinen Büroachsen gleichzeitig visualisieren möchte, kann man nicht alle Dimensionen (Klima, Licht, Jalousien) gleichzeitig anzeigen, sondern muss zwischen den Anzeigen wechseln.
3. Variablen zur Steuerung der Ansicht zählen natürlich auch zu den überwachten Variablen und reduzieren weiter die Zahl der darstellbaren Datenpunkte der Etage.
4. Werden Anzeigeelemente durch Steuervariablen ausgeblendet, so werden die in diesen Anzeigeelementen verwendeten Variablen immer noch überwacht, d.h., es müssen *separate Visualisierungen* für Klima, Licht und Jalousien erstellt werden.

Die folgende Abbildung zeigt den prinzipiellen Aufbau der Visualisierung, die sich aber aufgrund des Datenvolumens so nicht realisieren lässt:

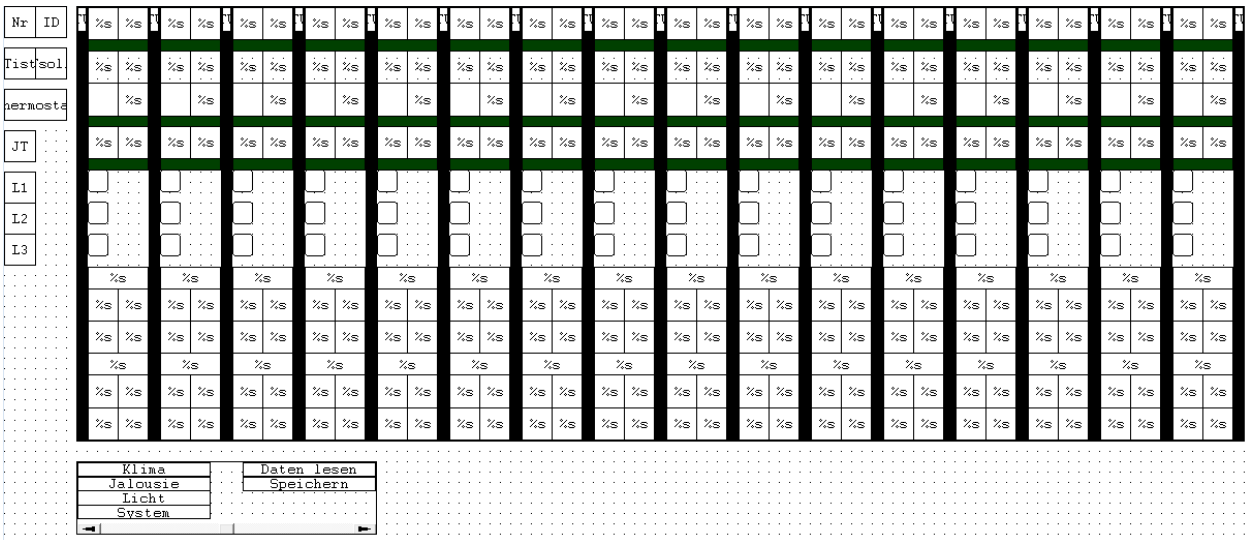


Abbildung 11: Visualisierung, grundlegendes Design

Dargestellt sind 16 Raumachsen und 17 Trennwände, links eine kleine Legende, und unten ein Steuerpanel. Die Trennwände lassen sich durch Auswahl ausblenden, um so Raumachsen zu Räumen zusammenzufassen. In den jeweiligen Blöcken der Raumachsen können folgende Parameter abgerufen bzw. eingegeben werden:

- IDs: Nummer der Raumachse (1-40) und Kunden-ID. Pro Raum erscheint das Feld der Kunden-ID nur einmal.
- Klimasteuerung: Angezeigt werden Ist-Temperatur, Soll-Temperatur, und die EnOcean-ID des Raumthermostats. Auch hier erscheinen die Felder nur einmal pro Raum.
- Jalousiesteuerung: Eingegeben bzw. angezeigt werden die Enocean-IDs der Taster mit Doppelwippe. Pro Raumachse können maximal 2 Taster zugeordnet werden. Alle Taster, die bei einer Raumachse eines Raumes angemeldet sind steuern unabhängig alle Jalousien des Raumes.
- Lichtsteuerung: Angezeigt werden die Dimmwerte der 3 Lichtstromkreise der Raumachse, und zwei EnOcean 4-fach-Taster mit EnOcean-ID und den Programmen der 4 Wippen. Auch wenn die Visualisierung dies suggeriert so betonen wir nochmal, dass, da die Programme der Wippen beliebig sind, die Taster und sogar die einzelnen Wippen Lichtfunktionen in beliebigen Raumachsen oder Büros ausführen können.

Im Panel zur Steuerung der Visualisierung gibt es zum einen Funktionen zum Lesen der Daten (EnOcean IDs, Programme der Lichtfunktionen, und Kunden-IDs) aus der SPS und, umgekehrt, die Funktion zum Schreiben von gegebenenfalls geänderten Daten. Es gibt die Möglichkeit, sich nur einen Ausschnitt der Daten anzeigen zu lassen (Klima, Jalousie, Licht oder Systemsteuerung) und mittels des Sliders den gezeigten Ausschnitt zu verschieben.

7.1 Ansteuerung der einzelnen Visualisierungen: Das Steuer-Panel

Es gibt insgesamt vier Visualisierungen zu den Themen Jalousie, Klima, Licht und System, wobei jede der Visualisierungen insgesamt 6 Raumachsen darstellt. In jeder der Visualisierungen wird, neben den Nummern der Raumachsen, den Nummern der Trennwände, und den Kunden-IDs, jeweils nur Steuerelemente der jeweiligen Funktion angezeigt, vgl. die Abbildung unten.

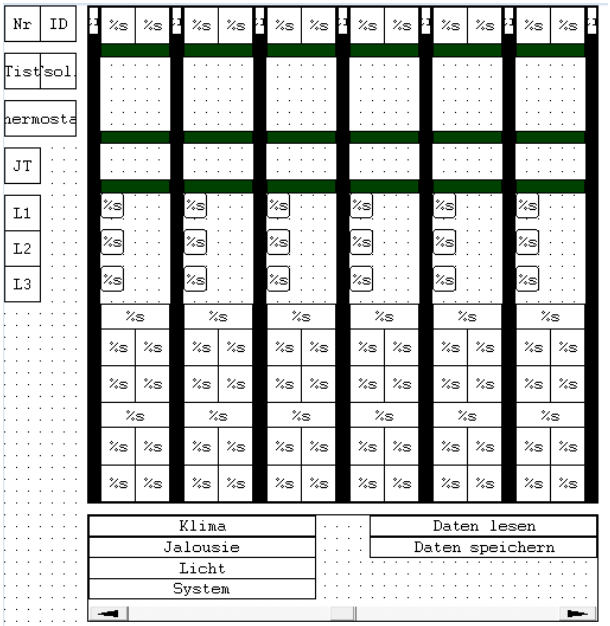


Abbildung 12: Beispiel: Visualisierung der Lichtfunktion

Über die Tasten Klima, Jalousie, Licht und System kann zwischen den Visualisierungen gewechselt werden. Gleichzeitig wird die Auswahl auch lokal in einer Variablen gespeichert und kann bei Bedarf oder bei Erweiterungen verwendet werden.

Der Slider am unteren Ende des Panels wählt den sichtbaren Bereich aus indem der Wert der linken Raumachse (zwischen 1 und 34) in eine lokale Variable geschrieben wird. Über „Daten lesen“ werden Kopien SPS-Variablen angelegt, damit diese in der Visualisierung modifiziert werden können. Die Änderungen haben aber erst Auswirkungen nach dem Abspeichern („Daten speichern“).

7.2 Systemfunktionen

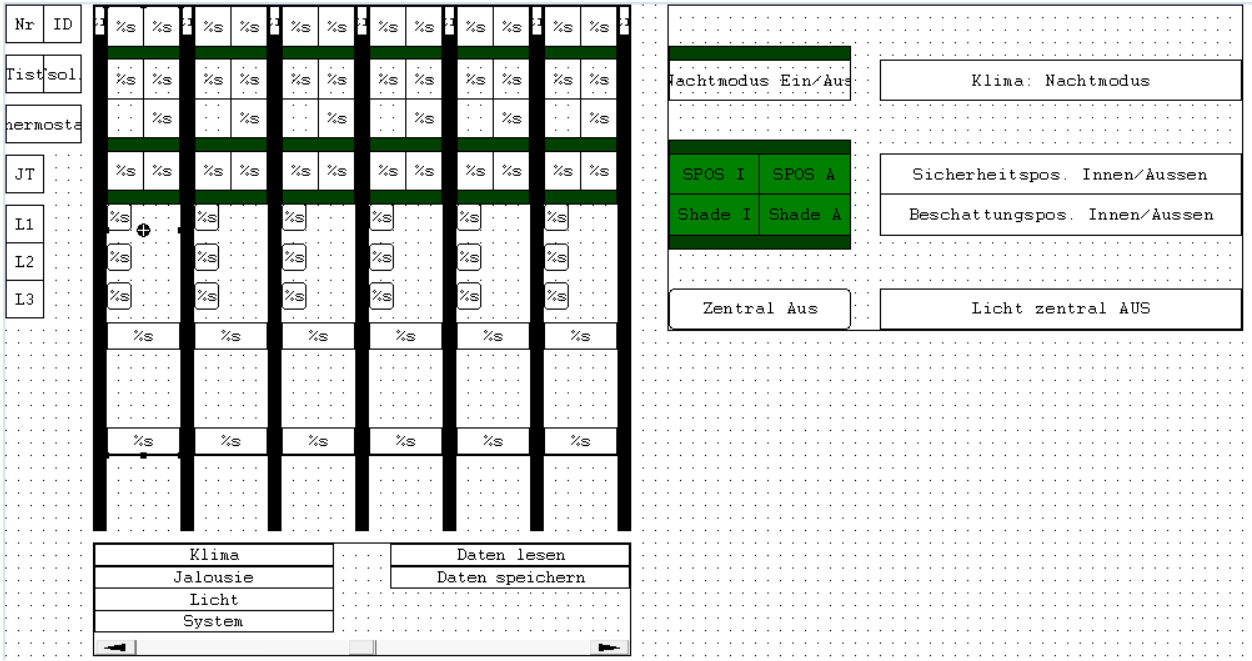


Abbildung 13: Beispiel: Visualisierung der Systemfunktionen

Unter den Systemfunktionen werden die Klimadaten angezeigt, die Dimmwerte der Lampen, und die EnOcean-IDs der Jalousie- und Lampentaster. Über ein weiteres Panel können Globalfunktionen ausgewählt werden, wie Anfahren der Sicherheitsposition oder der Beschattungsfunktion der Innen- und Außenjalousien, Ausschalten der Beleuchtung, und der Betrieb der Klimafunktionen im Nachtmodus (wobei diese Funktion natürlich über einen Kalender bzw. eine Uhr gesteuert werden müsste). Die Taster im linken Teil der Menüsteuerung wechseln entsprechend des Schaltzustands die Farbe.

8. Alternativen zur CoDeSys-Visualisierung: Zugriff auf interne Variablen

Mit CoDeSys erstelle Visualisierungen lassen sich, sofern eine Lizenz vorliegt, mittels CoDeSys am PC darstellen oder aber auf dem SPS-internen Webserver ablegen, um vom dort mittels Webbrowser oder App aufgerufen zu werden.[45] Da sowohl die graphische Darstellung als auch die Performance eher dürftig sind, wollen wir im Folgenden zeigen, wie man aus anderen Programmen heraus auf Variablen der Steuerung zugreifen kann. Allen Methoden gemeinsam ist, dass auf den direkt zugänglichen Speicherbereich der Eingänge, Ausgänge und Merker zugegriffen wird.

8.1 Direkt zugänglicher Speicherbereich nach IEC 61131-3

Innerhalb der WAGO SPS gibt es drei Speicherbereiche, auf die direkt zugegriffen werden kann: den Speicherbereich der Eingänge (I), den Speicherbereich der Ausgänge (Q), und den Speicherbereich der Merker (Q). Die jeweiligen Speicherbereiche sind zusammenhängend und werden symbolisch bit- (X) oder byteweise (B) angesprochen, wobei das erste Byte immer die Adresse 0 hat. Es können auch mehrere Bytes zusammengefasst werden als Wort (W, 2 Bytes) oder Doppelwort (D, 4 Bytes), wobei jeweils die Adresse des niedrigsten Bytes verwendet wird.

%IB0	Erstes Byte im Speicherbereich der Eingänge
%IW0	Erstes Wort im Speicherbereich der Eingänge
%IW1	Zweites Wort im Speicherbereich der Eingänge
%IX0.0	Erstes Bit (Position 0) im ersten Wort im Speicherbereich der Eingänge
%QB20	Zwanzigstes Byte im Speicherbereich der Ausgänge
%MD20	Zwanzigstes Doppelwort im Speicherbereich der Merker

Abbildung 14: Beispiele zur Adressierung der direkt zugänglichen Speicherbereiche nach IEC 61131-3

Die drei Speicherbereiche werden auf unterschiedliche Art erzeugt: Der Merkerbereich bzw. dessen Größe wird bei den Zielsystemeinstellungen festgelegt, vgl. die Abbildung unten, wobei die Zeile Memory den Speicherbereich der Merker definiert, hier 8192 Byte (16#2000), beginnend mit der physikalischen Adresse 50331648 (16#3000 0000).

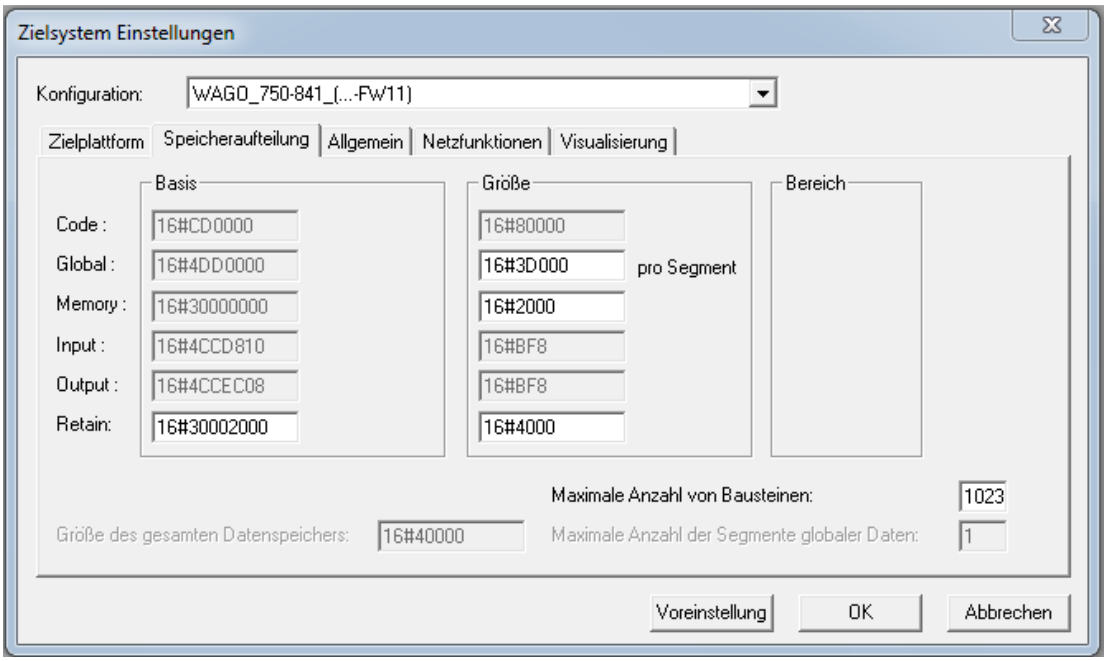


Abbildung 15: Speicheraufteilung des Zielsystems, hier eine WAGO 750-841, unter CoDeSys 2.3

[45] Genauer handelt es sich um die WAGO-WebVisu-App, die unter CoDeSys Version 2 erstellte Webseiten direkt auf Smartphones und Tablets darstellen kann. Verfügbar ist die App für iPad, iPhone und Geräte, die mit dem Android-Betriebssystem laufen, vgl. (WAGO, WebVisu Bedienungsanleitung, 2015), (WAGO, WebVisu Anwendungshinweis, 2015).

Die Bereiche des Eingangs- und des Ausgangsspeichers ergeben sich aus dem Knotenaufbau[46], und bestehen aus dem Ein- und Ausgangsprozessabbild des Knotens: Die Daten der Module, die den Knoten bilden, werden der Reihenfolge nach im Eingangs- und Ausgangsspeicher abgelegt, wobei jeweils zuerst die komplexen Ein- und Ausgänge der Klemmen (inklusive der Kommunikationsklemmen), danach die nicht-komplexen abgebildet werden. Hierbei sind komplexe Ein- und Ausgänge solche, deren Typ mehr als ein Byte benötigt.

Werden komplexe Daten (z.B. vom Typ Word, 2 Bytes) abgespeichert, so ist die Byte-Reihenfolge Little-Endian (Intel-Format), d.h., das Byte mit der niedrigeren Wertigkeit wird im Feld mit der niedrigeren Adresse gespeichert, das Byte mit der höheren Wertigkeit im Feld mit der höheren Adresse.

[46] Vgl. (WAGO, Modbuskommunikaton zwischen WAGO Ethernet Kopplern und Controllern - Anwendungshinweis, Verson 1.2.1), Abschnitt 2.2.

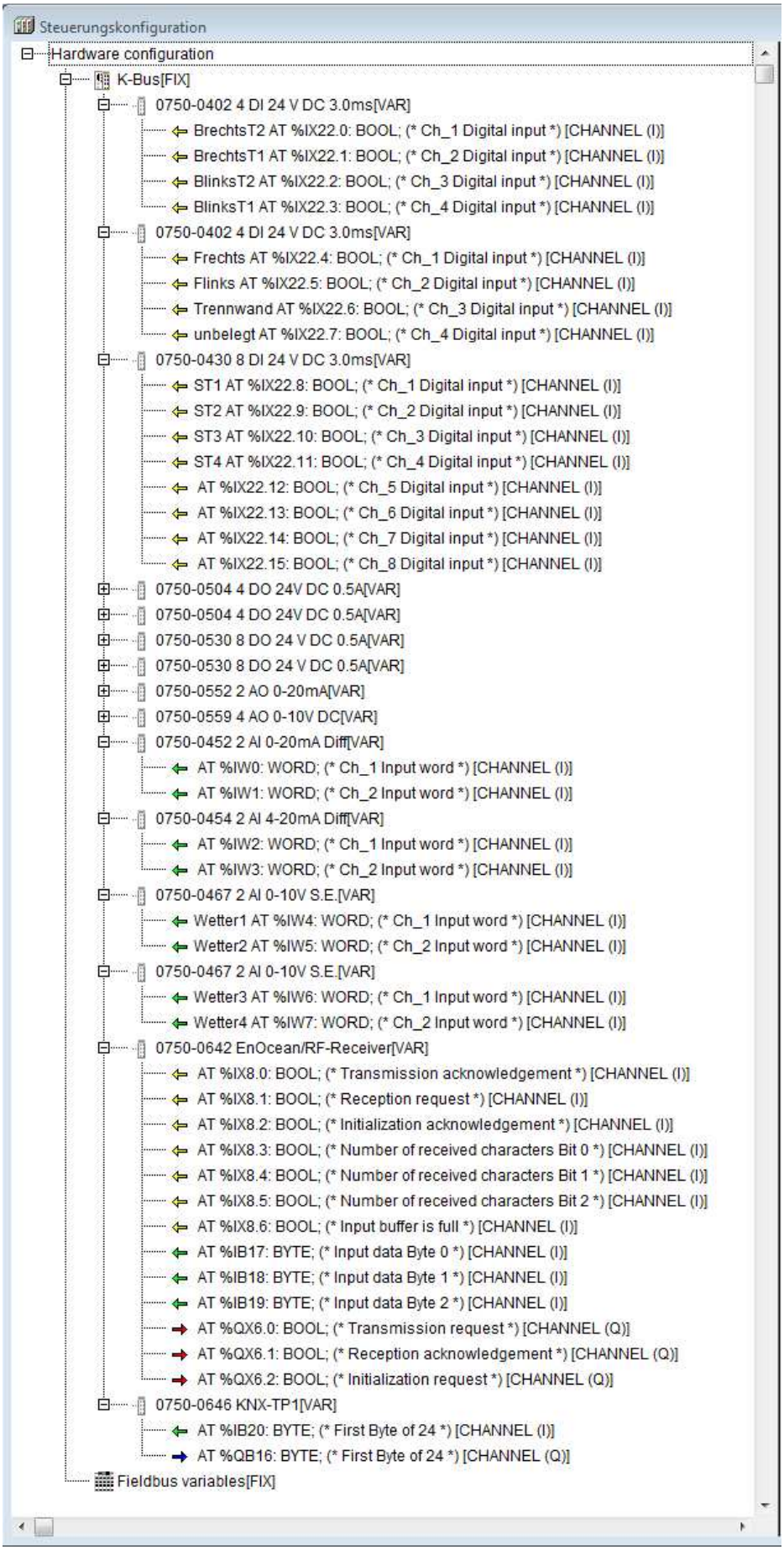


Abbildung 16: Prozessabbild eines WAGO Knotens, aufgeklappt sind alle Module mit Eingangsvariablen

Die Abbildung zeigt einen Teil des Prozessabbilds eines WAGO Knotens.[47] Im Eingangsbereich ist die erste nicht-digitale Klemme die Klemme 750-552 mit zwei Analogen Eingangssignalen, die jeweils ein Wort belegen. Belegt werden die Wörter 0 und 1, d.h., die Bytes 0 bis 3. Wort 0 belegt hierbei die Bytes IB0 (niedriges Byte) und IB1 (höherwertiges Byte). Die nächste Klemme mit komplexen (nicht-digitalen) Eingangsklemmen ist die Klemme 750-454, die die Wörter 2 und 3 im Speicherbereich der Eingänge belegt, d.h., Bytes 4 bis 7, etc. Die folgende Tabelle gibt die vollständige Liste wieder.[48]

Klemme	Eingang	Byte im Eingangsbereich
750-452	%IW0	1 und 0
	%IW1	3 und 2
750-454	%IW2	5 und 4
	%IW3	7 und 6
750-467	%IW4	9 und 8
	%IW5	11 und 10
750-467	%IW6	13 und 12
	%IW7	15 und 14
750-642	%IX8.0 bis %IX8.6	16
	%IB17	17
	%IB18	18
	%IB19	19
	%IB20	20
750-402	%IX22.0 bis %IX22.3	44
750-402	%IX22.4 bis %IX22.7	44
750-430	%IX22.8 bis %IX22.15	45

Abbildung 17: Klemmen, IEC 61131-3 Namen und Belegung im Eingangsbereich eines Knotens

Soll der Zustand der Büroetage und dessen Steuerung über Merker abgebildet werden, so benötigen wir einen nicht unerheblichen Speicherraum:

Jalousietaster:
- EnOceanID: DWORD (4 Byte)
- Zustand der 4 Wippen: 4 Bit

Lichttaster:
- EnOceanID: DWORD
- Zustand der 4 Wippen: 4 Bit
- 4 Programme: 4x80 Byte

Raumthermostat:
- EnOceanID: DWORD (4 Byte)
- Isttemperatur: REAL (4 Byte)
- Solltemperatur: REAL (4 Byte)
- Anwesenheit: 1 Bit

Anwesenheitssensor:
- 1 Bit

Lampen:
- Dimmwert: Byte

Klimaaktoren:
- Ausgänge Heizung: 1 Bit
- Ausgänge Klima: 1 Bit

Jalousieaktoren:

[47] Gezeigt wird die Steuerungskonfiguration eines WAGO Koffers, wie er im Labor der Fachhochschule Dortmund verwendet wird.

[48] Warum bei Bits das Offset in 2-er Gruppen von Bytes angegeben wird, ist aus der Dokumentation nicht ersichtlich. Auch bleibt unklar, warum die Daten nicht in fortlaufende Bytes geschrieben werden, wie eigentlich aus der Dokumentation zu erwarten wäre. Die hier beschriebene Belegung wurde empirisch durch Anzeigen der Bytes im Inputspeicher ermittelt.

- Ausgänge Innenjalousie: 2 Bit
- Ausgänge Außenjalousie: 2 Bit

Raumachse:

- ID: INT (2 Byte)
- Kunden-ID: INT

Trennwand:

- Vorhanden (ja/nein): 1 Bit

Zentrale Funktionen:

- Diverse

Zusammenfassend wird also mindestens folgender Merkerbereich benötigt, wobei teilweise auch auf den Ausgangsbereich ausgewichen werden könnte:

Sensor/Aktor	**Pro Raumachse (maximal), in Bytes**	**Pro Flur (maximal), in Bytes**
Trennwand	1/8	41/8 = 5 1/8
Raumachse: ID, KundenID	4	160
Anwesenheitssensor	1/8	5
Jalousiefunktion	4/8	20
Jalousietaster	2x4 4/8	40*9 = 360
Lichtzustand (Dimmwert)	1	40
Lichttaster	2x(4*80 + 4 + 4*1/8)	40*649 = 25940
Klimaaktoren	2*1/8	10
Raumthermostat	12 1/8	512
Gesamt, ohne Lichtprogramme		**1451 1/8**
Gesamt, Lichtprogramme		**25600**

Abbildung 18: Speicherbedarf Merker, ohne weitere (zentrale) Funktionen

Damit wird ca. 1kByte Speicherplatz für Merkerspeicherplatz für die grundlegenden Funktionen und weitere 24kByte Merkerspeicherplatz für die Programme der Lichtsteuerung benötigt. Da dies den üblichen Merkerbereich von 8kByte deutlich überschreitet[49], aber auch den zulässigen Merkerbereich von 24k, die Lichtprogramme aber höchst selten abgerufen bzw. geändert werden, könnte man die Lichtprogramme über einen einfachen Bus abrufen, wobei die WAGO-SPS als Slave fungiert:

- Datenbreite 320 Byte (alle Programme eines Lichttasters mit 4 Wippen)
- Breite der Steuersignale 2 Byte:
  - Tasten-ID: 1 Byte (wird nur durch den Master beschrieben)
  - Steuerbefehle: 1 Byte

  Mögliche Steuerbefehle[50] wären
  - 0: Quittierung durch den Slave
  - 1: Request (Anforderung durch Master)
  - 2: Write (Aufforderung an den Slave, das Programm zu speichern)
  - Etc.

[49] Die Größe des Retain Memories (Merkerbereich) liegt z.B. bei der WAGO SPS 750-841 zwischen 8 und 24kB.

[50] Eine mögliche Kommunikation läuft folgendermaßen ab: Der Master schreibt die ID eines Tasters und den Befehl 1 in die Steuerbytes. Der Slave (WAGO-SPS) schreibt die Programme der 4 Wippen in die Datenbytes und quittiert mit 0, der Master holt die Datenbytes ab.

8.2 Zugriff über HTTP und Server Side Includes (SSI)

Server Side Includes (SSI) sind kleine Programme bzw. Befehle, die in HTML oder XML Dateien eingebettet werden. Sind auf einem Web-Server SSIs zugelassen, so werden die kleinen Programme ausgeführt bevor eine Datei einem Request übergeben wird.[51] Die WAGO SPS unterstützt unter anderen die Befehle `READPI` und `WRITEPI`.[52] Zum Lesen von Daten wird auf dem Webserver der WAGO-SPS ein XML, HTML oder SHTML File abgelegt, welches zum Beispiel die Zeile[53]

```
<!—#READPI ADR=MW0 & FORMAT=%d-->
```

Code 23: Lesen von Registerinhalten mittels READPI

enthält. Bevor die Datei an den Request-Service zurückgegeben wird, wird der Befehl `READPI` ausgeführt, und die obige Zeile durch den Wert des Registers `MW0` ersetzt, dargestellt hier als Dezimalzahl. Analog lassen sich auch Input- und Outputregister (I, Q) lesen.

Zum Schreiben über http wird einfach der Befehl WRITEPI über http bzw. https abgesetzt,

```
https://user.password@IP_der_SPS/WRITEPI?ADR1=MW0&VALUE1=12&FORMAT1=%d
```

Code 24: Schreiben von Registerinhalten mittels WRITEPI

wobei https benutzt werden muss um sicherzustellen, dass die Login-Informationen auch verschlüsselt werden. [54,55] Im Beispiel wird also der (Dezimal-)Wert 12 in das (Wort-)Register MW0 geschrieben.

[51] Sicherheitstechnisch ist die Verwendung von SSI nicht ganz unproblematisch, und insbesondere der Befehl exec, der beliebige andere Programme starten kann, sollte explizit deaktiviert sein. Für die weiteren Ausführungen verweisen wir auch auf (World Wide Web Consortium (W3C), 2011).

[52] PI steht hier für Process Information. Während es im Internet unzählige (gleichartige) Beispiele gibt, wie man diese beiden Funktionen verwendet, so scheint es aber keine Dokumentation der beiden Befehle zu geben.

[53] In der Praxis wird die XML- oder HTML-Datei weitere Strukturinformationen beinhalten, und entweder direkt in einem Webbrowser angezeigt werden, oder aber die Daten werden durch einen XML-Parser extrahiert und weiterverarbeitet.

[54] Es ist zu beachten, dass verschiedene Versionen des Internet-Explorers die Übergabe von User-Credentials (Login-Informationen) nicht unterstützen.

[55] Zum Lesen eines einzigen Registers kann natürlich auch READPI über einen http-Request aufgerufen werden. Die Antwort auf den Request enthält dann genau das eine Datum.

8.3 Zugriff über Modbus/TCP

Modbus ist ein einfaches Industrieprotokoll das es ermöglicht, Daten byte- und bitweise zu lesen und zu schreiben.[56] Das Modbus Protokoll kennt digitale Dienste (Lesen und Schreiben digitaler Ein- und Ausgänge) und das Lesen und Schreiben von Registern (16 Bit Ein- und Ausgangsdaten). Da das Protokoll nur Adressen zwischen (hexadezimal) 00 00 und FF FF ansprechen kann, werden Modbus-Adressen durch die WAGO-SPS auf andere Speicherbereiche, insbesondere Eingangsspeicher, Ausgangsspeicher und Merkerspeicher, gemappt. Das Adressmapping ist teilweise abhängig vom auszuführenden Modbus Befehl. Beispielhaft zeigen wir in den folgenden zwei Tabellen die unterstützten Modbus-Dienste der WAGO SPS 705-841[57] und das Mapping der Modbus-Adressen für Register-Dienste und digitale Dienste[58]

Funktionscode (FC)	Name	Beschreibung
FC1	Read coils	Rücklesen mehrerer digitaler Ausgänge
FC2	Read inputs discrete	Lesen mehrerer digitaler Eingänge
FC3	Read holding registers	Lesen mehrerer analoger Eingänge(und Ausgänge)
FC4	Read input registers	Lesen mehrerer analoger Eingänge(und Ausgänge)
FC5	Write coil	Schreiben eines einzelnen digitalen Ausgangs
FC6	Write single register	Schreiben eines einzelnen analogen Ausgangs
FC11	Get communication event counter	Kommunikationsereigniszähler
FC15	Force multiple coils	Schreiben mehrerer digitaler Ausgänge
FC16	Write multiple registers	Schreiben mehrerer analoger Ausgänge
FC22	Mask write	Manipulation einzelner Bits eines Registers
FC23	Read/Write multiple registers	Schreib-Lese-Operation auf analoge Ein/Ausgänge

Abbildung 19: Tabelle der durch die WAGO SPS 750-841 unterstützten Modbus-Dienste

Modbus Adresse	IEC 61131 Adresse	Beschreibung	Register Dienste
0000 – 00FF	%IW0 – %IW255	Erste 256 Worte Input Daten	FC3, FC4, FC23, FC6, FC16, FC22, FC23
0100 – 01FF	%QW 256 – %QW255	PFC-OUT: Flüchtige SPS Ausgangsvariablen	FC3, FC4, FC23, FC6, FC16, FC22, FC23
0200 – 02FF	%QW0 – %QW255	Erste 256 Worte Output Daten	FC3, FC4, FC23, FC6, FC16, FC22, FC23
0300 – 03FF	%IW256 – %IW511	PFC-IN: Flüchtige SPS Eingangsvariablen	FC3, FC4, FC23, FC6, FC16, FC22, FC23
0400 – 04FF		Modbus Exception „Illegal data address“	
0500 – 09FF		Nicht verwendet	
1000 – 2FFF		Konfiguration Register	FC3, FC4, FC23, FC6, FC16, FC22, FC23
3000 – 5FFF	%MW0 – %MW12287	Merkerspeicher	FC3, FC4, FC23, FC6, FC16, FC22, FC23
6000 – 62FC	%IW512 – %IW1275	Weitere 764 Worte	FC3, FC4, FC23,

[56] Vgl. (IEC 61158 - Digital data communication for measurement and control - Fieldbus for use in industrial control systems, 1999-2013) und speziell (WAGO, Modbuskommunikaton zwischen WAGO Ethernet Kopplern und Controllern - Anwendungshinweis, Verson 1.2.1).

[57] Vgl. die Tabelle (WAGO, Modbuskommunikaton zwischen WAGO Ethernet Kopplern und Controllern - Anwendungshinweis, Verson 1.2.1), S. 52.

[58] Vgl. die verschiedenen Tabellen in (WAGO, Modbuskommunikaton zwischen WAGO Ethernet Kopplern und Controllern - Anwendungshinweis, Verson 1.2.1), Abschnitt 5.4.1. Insbesondere für neuere, Modbus-fähige WAGO Controller finden sich die Mapping-Tabellen in den jeweiligen Handbüchern.

		Input Daten	FC6, FC16, FC22, FC23
62FD – 6FFF		Modbus Exception „Illegal data address“	
7000 – 72FC	%QW512 – %QW1275	Weitere 764 Worte Output Daten	FC3, FC4, FC23, FC6, FC16, FC22, FC23
72FD – FFFF			

Abbildung 20: Tabelle der durch die WAGO SPS 750-841 unterstützten Modbus Register-Dienste

Modbus Adresse	**IEC 61131 Adresse**	**Beschreibung**	**Digitale Dienste**
0000 – 001FF	%IX0.0 – %IX31.15	Erste 512 digitale Inputs	FC1, FC2, FC5, FC15
02FF – 03FF	%Qx0.0 – %QX31.15	Erste 512 digitale Outputs	FC1, FC2, FC5, FC15
04FF – 0FFF		Modbus Exception „Illegal data address“	
1000 – 1FFFF	%QX256.0 – %QX511.15	PFC-OUT: Flüchtige SPS-Ausgangsvariablen	FC1, FC2, FC5, FC15
2000 – 2FFFF	%IX256.0 – %IX511.15	PFC-IN: Flüchtige SPS-Eingangsvariablen	FC1, FC2, FC5, FC15
3000 – 7FFFF	%MX0.0 – %MX1279.15	Merkerspeicher	FC1, FC2, FC5, FC15
8000 – 85F7		Weitere digitale Inputs von 513 bis 2039	FC1, FC2, FC5, FC15
85F8 – 8FFF		Modbus Exception „Illegal data address“	
9000 – 95F7		Weitere digitale Outputs von 513 bis 2039	FC1, FC2, FC5, FC15
95F8 – FFFF		Modbus Exception „Illegal data address“	

Abbildung 21: Tabelle der durch die WAGO SPS 750-841 unterstützten digitale Modbus-Dienste

Die Kommunikation mit der WAGO-SPS verläuft über Modbus/TCP, Port 502, d.h., das Modbus-Protokoll wird in das TCP-Protokoll[59] eingebettet. Für verschiedene Betriebssysteme bzw. Programmiersprachen existieren Implementierungen des Modbus-Protokolls:

- Jamod für Java[60]
- phpmodbus[61] für PHP
- WSMBT Modbus Master TCP/IP für .NET (kommerziell)[62]

Da das Modbus-Protokoll relativ einfach ist, genügt es allerdings auch, Zugriff auf eine TCP-Implementierung zu haben, und hierin das Modbus-Protokoll selbst einzubetten.[63]

[59] Transmission Control Protocol (TCP). Es ist auch möglich, über das verbindungslose (nicht-verbindungsorientierte) User Datagram Protocol (UDP) zu kommunizieren.
[60] http://jamod.sourceforge.net/
[61] http://github.com/krakorj/phpmodbus/
[62] http://www.modbustools.com/. Auf der gleichen Webseite werden auch Modbus ActiveX Controls angeboten. Hiermit lassen sich Modbus-Slaves u.a. aus Excel heraus ansprechen.
[63] TCP Sockets lassen sich relativ einfach implementieren, in PHP zum Beispiel durch den folgenden Code, wobei zur Vereinfachung die Fehlererkennung weggelassen wurde, vgl. http://php.net/manual/en/book.sockets.php/:

```
$local_ip = ‚10.3.34.19‘; // Lokale IP-Adresse
$remote_ip = ‚10.3.34.100‘; // Remote IP
```

9. Mögliche Verbesserungen

Die hier vorgestellt Steuerung einer Büroetage wird, mit Ausnahme kleinerer Fehler die erst in einem Testlauf zum Vorschein kommen, sicher funktionieren. Problematisch dürfte allerdings die Zykluszeit sein, d.h., die Zeit, die zum Durchlauf eines Zyklus benötigt wird. Es gibt einige Stellen, an denen die Durchlaufzeit durch Umprogrammierung reduziert werden kann, z.B.

- Auslagerung der Klima-Funktionen in eigenen Thread
  Die Regelung der Klima-Funktionen ist naturgemäß sehr träge, und solange kein direktes Feedback benötigt wird (z.B. Anzeige der Soll-Temperatur oder der Ist-Temperatur in einem Panel), können die Klima-Funktionen in einen eigenen Thread ausgelagert werden, der zum Beispiel nur einmal alle 30 Sekunden läuft.
- Optimierung der Szeneaufrufe
  Derzeit können jedem Baustein zur Steuerung der Lichtszenen bis zu 64 Wippen zugeordnet werden, die dann eine oder mehrere Lichtszenen aufrufen. Es wird derzeit nicht gespeichert, wie viele Wippen tatsächlich jeweils zugeordnet sind. Gerade wenn wenig mit Lichtszenen gearbeitet wird lässt sich hier Programmlaufzeit einsparen.

Weiterhin fehlen noch mindestens folgende weitere, für eine marktreife Anwendung wesentliche, Funktionen:

- Einstellmöglichkeiten für weitere Parameter, wie z.B. die Laufzeiten der Beleuchtung auf dem Flur und in den Toiletten.
- Rollback-Funktionen bei der Eingabe, gegebenenfalls mit der Möglichkeit zum Test der Funktionen vor dem Speichern.
- Es findet derzeit keine Überprüfung auf syntaktische Korrektheit der Lichtprogramme statt.
- Verbesserung des Management-Tools, insbesondere Verwendung einer GUI-Programmierung, die nicht auf die von CoDeSys bereitgestellten Web-Interface beruht, sondern direkt auf die Daten der SPS über Modbus/TCP zugreift.

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP); // IPv4, Stream, TCP-
Protokoll
socket_bind($socket, $local_ip, 502); // Binding
socket_connect($socket, remote_ip, 502); // Verbindungsaufbau
// Befehle wie socket_write, socket_send, bzw. socket_read und socket_recv
socket_close($socket); // Schliessen der Verbindung
```

10. Zusammenfassung und Ausblick

In dieser Bachelor Thesis wurden die wesentlichen Funktionen – Licht, Jalousien und Klima – einer Büroetage bestehend aus einer Teeküche, Flur, Toilettenbereich und 40 Büroachsen programmiert. Jede Büroachse ist mit Innen- und Außenjalousien, Heizung und Kühlung ausgestattet. Die Büroachsen lassen sich flexible zu Büros unterschiedlicher Größe zusammenfassen und werden dann mit EnOcean-Sensoren, also Tastern und Raumthermostaten, ausgestattet, über die die Funktionen der Büros gesteuert werden. Die Definition der Taster, deren Wippen, und der Raumthermostate erfolgt flexibel und zur Laufzeit der eingesetzten WAGO-SPS über eine Visualisierung. Die Steuerung der Beleuchtung wurde mittels DALI realisiert, wobei den Wippen der EnOcean-Taster zur Laufzeit kleine Programme zugeordnet werden, die die Beleuchtung entsprechend den Programmen dimmen und schalten.

Zur Steuerung der Büroetage wurde auch eine Visualisierung erstellt, die, neben der Monitoring-Funktion, die Möglichkeit bietet, EnOcean-Raumthermostate und EnOcean-Taster zur Steuerung der Bürofunktionen zu definieren. Hierbei wird der WAGO-SPS der Sensor über seine EnOcean-ID bekannt gemacht, bei Lichttastern wird darüber hinaus zu jeder Wippe ein Programm hinterlegt, welches die Funktion der Wippe definiert.

Die mit CoDeSys erstellte Visualisierung ist jedoch in einigen Aspekten unbefriedigend: Neben der Optik einer solchen Visualisierung, die heutigen Ansprüchen eigentlich nicht mehr genügt, ist auch die Anzahl der darstellbaren Datenpunkte sehr begrenzt. Wünschenswert wäre hier eine Visualisierung bzw. ein Management-Tool basierend auf einer anderen Technologie, z.B. eine Management-Software mit grafischer Benutzeroberfläche programmiert in einer Hochsprache wie Java oder C++. Insbesondere ließen sich hier dann auch relativ einfach elementare Bedienfunktionen wie Roll-Back, Editierfunktionen, Datensicherung der Lichtprogramme, etc. realisieren. Als Ausblick auf ein solches Management-Tool widmen wir uns im letzten Kapitel dieser Arbeit dann auch dem Lesen und Schreiben von Daten der SPS über Modbus/TCP.

Abbildungsverzeichnis

Verzeichnis der Codefragmente

Literatur und Quellen

Aschendorf, B. (kein Datum). *Datenpunktaufstellung Bürobereich.* Abgerufen am 30.07.2015 von Homepage der FH Dortmund: http://www.fh-dortmund.de/

Aschendorf, B. (2014). *Energiemanagement durch Gebäudeautomation: Grundlagen - Technologien - Anwendungen.* Heidelberg: Springer Vieweg.

Aschendorf, B. (kein Datum). *Vorlesung zur Dezentralen Gebäudesystemtechnik, Kapitel 16: WAGO-SPS.* Abgerufen am 01.07.2015 von http://www.fh-dortmund.de/de/fb/3/personen/lehr/aschendorf/lehre/Lehrartikel_EG/EG_Vorlesungsmaterial.php

Butz, C. (20.05.2015). *Schalten und Walten: Die WAGO-SPS in der Gebäudeautomation. Praktikumsbericht zum Modul Betriebliche Praxis.* Fachhochschule Dortmund, Fachbereich Informations- und Elektrotechnik, Dortmund.

CoDeSys. (kein Datum). *Homepage*. Abgerufen am 23.07.2015 von http://www.codesys.de

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms, 2nd edition.* MIT Press.

DALI AG. (2. Auflage, 2002). *DALI Handbuch.* Frankfurt: DALI AG (Digital Addressable Lighting Interface Activity Group, Fachverband Elektroleuchten im ZVEI).

DALI AG (kein Datum). *Winner DALI Award 2014*. Abgerufen am 15.07.2015 von Homepage der DALI AG: http://www.dali-ag.org/dali-award/winner-dali-award-2014/details/news/detail/News/trust-tower-world-trade-centre-abu-dhabi-uae.html

EnOcean. (kein Datum). *EnOcean Homepage*. Abgerufen am 13.07.2015 von http://www.enocean.de/

EnOcean Alliance (20.01.2011). *EnOcean Equipment Profiles (EEP), Version: 2.1.* Abgerufen am 22.07.2015 von EnOcean Homepage: http://www.enocean-alliance.org/fileadmin/redaktion/enocean_alliance/pdf/EnOcean_Equipment_Profiles_EEP2.1.pdf

IEC 61131-3 mit CoDeSys V3: Ein Praxisbuch für SPS-Programmierer. (2011). CoDeSys Eigenverlag.

IEC 61158 - Digital data communication for measurement and control - Fieldbus for use in industrial control systems. (1999-2013)

Stoltenberg-Hansen, V., & Lindström, I. (1994). *Mathematical Theory of Domains.* Cambridge: Cambridge University Press.

thermokron GmbH (12.02.2014). *SR04 Funk-Raumtemperaturfühler; DE-Datenblatt.* Abgerufen am 08.07.2015 von thermokon Homepage: http://www.thermokon.de

WAGO. (26.02.2009). *Bausteinbeschreibungen für allgemeine Gebäudefunktionen, Programmbibliothek und Dokumentation.* Abgerufen am 01.07.2015 von http://www.wago.com

WAGO. (02.05.2014). *Bausteinbeschreibungen für den Interface_Baustein RS-232/SMI.* Abgerufen am 07.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (23.01.2008). *Bausteinbeschreibungen für Enocean Funkempfänger 750-642.* Abgerufen am 08.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (08.07.2013). *Bausteinbeschreibungen für HLK-Funktionen - Bibliothek und Dokumentation.* Abgerufen am 07.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (18.04.2008). *DALI_02_d, Bausteinbeschreibungen für die DALI-Masterklemme 750-641.* Abgerufen am 01.06.2015 von Homepage WAGO: http://www.wago.de/

WAGO. (05.01.2015). *Datenblatt zu 750-642, Funkempfänger-Busklemme.* Abgerufen am 02.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (29.04.2013). *EnOcean Equipment Profile (EEP) - Anbindung von EnOcean-Funk-Sensoren/Aktoren unter Verwendung WAGO-EnOcean-Bibliothek.* Abgerufen am 02.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (2011). *Handbuch zur DALI/DSI-Masterklemme 750-641.* Abgerufen am 02.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (14.11.2007). *Konfiguration eines DALI-Beleuchtungssystems über die Visualisierung der WAGO-I/O-PRO.* Abgerufen am 01.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (kein Datum). *Modbuskommunikaton zwischen WAGO Ethernet Kopplern und Controllern - Anwendungshinweis, Verson 1.2.1.* Abgerufen am 20.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (kein Datum). *Produktkatalog*. Abgerufen am 02.07.2015 von http://www.wago.de/

WAGO. (2007). *WAGO-I/O-System 750 - Projektierungshinweise, Version 2.0.1.*

WAGO. (2015). *WebVisu Anwendungshinweis.* Abgerufen am 20.07.2015 von WAGO Homepage: http://www.wago.de/

WAGO. (2015). *WebVisu Bedienungsanleitung.* Abgerufen am 20.07.2015 von WAGO Homepage: http://www.wago.de/

Wellenreuther, G., & Zastrow, D. (2015 (6. Auflage)). *Automatisieren mit SPS - Theorie und Praxis.* Wiesbaden: Springer Vieweg.

Winskel, G. (1993). *The Formal Semantics of Programming Languages: An Introduction.* MIT Press.

World Wide Web Consortium (W3C) (12. 03 2001). *Server Side Include Commands*. Abgerufen am 24.07.2015 von http://www.w3.org/Jigsaw/Doc/User/SSI.html

Anhang A: Datentypen der Bibliothek Bueroetage.lib

A.1 Daten ETC (sonstige)

Daten für den Gang (Flur)

Kategorie:	Daten_ETC	
Name:	TGang	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
xTasterEin1, xTasterEin2	BOOL	Taster für Flurbeleuchtung
xSensor1, xSensor2, xSensor3, xSensor4	BOOL	Anwesenheitssensoren
Ausgang (extern):	**Datentyp:**	**Kommentar:**
xDoLS1, xDoLS2	BOOL	Ausgänge für zwei Lichtstromkreise

Beschreibung:

Der Datentyp sammelt die (externen) Eingänge der Flurtaster, die (externen) Eingänge der Sensoren, die (externen) Ausgänge der Lichtstromkreise, und die Zeitschaltfunktion der Flurbeleuchtung.

Implementierung:

```
TYPE TGang :
STRUCT
        xTasterEin1, xTasterEin2: BOOL; (* Tastereingaenge, auf Klemmen legen *)
        xSensor1, xSensor2, xSensor3, xSensor4: BOOL; (* Sensoren, auf Klemmen legen *)
        xSensor1_old, xSensor2_old, xSensor3_old, xSensor4_old: BOOL; (* Sensoren, alt *)
        xOn: BOOL; (* Globales Einschalten mit Schaltuhr *)
        xDoLS1, xDoLS2: BOOL; (* Ausgaenge für Lichtstromkreise, auf Klemmen legen *)
        TreppenLicht: Fb_Treppe1;
END_STRUCT
END_TYPE
```

Daten für den Teeküche

Kategorie:	Daten_ETC	
Name:	TTeeKueche	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
xToggle	BOOL	Taster für Beleuchtung der Teeküche
Ausgang (extern):	**Datentyp:**	**Kommentar:**
xDoLicht	BOOL	Ausgang für einen Lichtstromkreis

Beschreibung:

Der Datentyp sammelt die (externen) Eingänge der Teeküche (ein Taster) und den Ausgang des Lichtstromkreises.

Implementierung:

```
TYPE TTeeKueche :
STRUCT
        xToggle: BOOL; (* Auf Klemme legen *)
        LichtAktor: Fb_Stromstoss;
        xDoLicht: BOOL; (* Auf Klemme legen *)
END_STRUCT
END_TYPE
```

Daten für den Toiletten

Kategorie:	Daten_ETC	
Name:	TToilette	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
xTaster	BOOL	Taster für Beleuchtung
xSensor	BOOL	Anwesenheitssensor
Ausgang (extern):	**Datentyp:**	**Kommentar:**
xDoLichtAktor	BOOL	Ausgänge für Beleuchtung
Beschreibung:		
Der Datentyp sammelt die (externen) Eingänge der Toiletten (ein Taster und ein Anwesenheitssensor) und den Ausgang des Lichtstromkreises.		
Implementierung:		

```
TYPE TToilette :
STRUCT
	xTaster: BOOL; (* Taster, auf Klemmen legen *)
	xSensor: BOOL; (* Sensor, auf Klemmen legen *)
	xSensor_old: BOOL;
	ToilettenLicht: Fb_Treppe1;
	xDoLichtAktor: BOOL; (* Auf Klemmen legen *)
END_STRUCT
END_TYPE
```

A.2 Daten Steuerung

Daten für Büroachse

Kategorie:	Daten_Steuerung	
Name:	BueroAchse	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
Ausgang (extern):	**Datentyp:**	**Kommentar:**
xHeating	BOOL	Schaltsignal Heizen
xCooling	BOOL	Schaltsignal Kühlen
xDoAJalousieAuf, xDoAJalousieAb	BOOL	Schaltsignale Außenjalousie
xDoIJalousieAuf, xDoIJalousieAb	BOOL	Schaltsignale Innenjalousie

Beschreibung:

Der Datentyp sammelt die Daten einer Büroachse, insbesondere

- Nummer der zugeordneten EnOcean Klemme
- Nummer des zugeordneten DALI-Busses
- Nummern der Lichtstromkreise auf dem DALI-Bus
- Dimmwerte der drei Lichtstromkreise inklusive der Funktionsbausteine zur Abfrage der Werte
- Dimmaktoren für die einzelnen Lampen und für die Gruppe bestehend aus drei Lampen
- Jalousieaktoren für Innen- und Außenjalousien
- Funktionsbaustein zur Klimaregelung (Heizung und Kühlung)

Implementierung:

```
TYPE BueroAchse :
STRUCT
(* EnOcean Klemme *)
	bModule_750_642: BYTE; (* Nummer der EnOcean-Klemme *)

(* Lichtstromkreise *)
	bDaliBus: BYTE;			(* Nummer des DaliBusses *)
	bLS1: BYTE;			(* Nummer des Lichtstromkreises 1 *)
	bLS2: BYTE;			(* Nummer des Lichtstromkreises 2 *)
	bLS3: BYTE;			(* Nummer des Lichtstromkreises 3 *)
	bHelligkeit1: BYTE;	(* Helligkeit von Lichtstromkreis 1 *)
	bHelligkeit2: BYTE;	(* Helligkeit von Lichtstromkreis 2 *)
	bHelligkeit3: BYTE;	(* Helligkeit von Lichtstromkreis 3 *)
	StatusDimmwert1: FbDALI_StatusDimmValue;
	StatusDimmwert2: FbDALI_StatusDimmValue;
	StatusDimmwert3: FbDALI_StatusDimmValue;

(* Dimmaktoren *)
	DimmenGruppe: TDimmAktor;
	DimmenEinzeln: ARRAY[1..3] OF TDimmAktor;

(* Stromkreise *)

(* Jalousie *)
	AJalousie: FbJalousie;(* Steuerung  Außenjalousie *)
	IJalousie: FbJalousie;(* Steuerung  Innenjalousie *)
			(* Taster global *)

(* Anwesenheit *)
	xAnwesend: BOOL := FALSE;

(* Heizungssteuerung *)
	KlimaController: Fb2PointSingleRoomController; (* 2-Punkt Einzelraumregler *)

(* Datenpunkte, auflegen auf Klemmen! *)
	xHeating: BOOL; (* Schaltsignal Heizen *)
	xCooling: BOOL; (* Schaltsignal Kühlen *)
	xDoAJalousieAuf: BOOL; (*Schaltsignal Jalousie AUF, Aussenjalousie *)
	xDoAJalousieAb: BOOL; (* Schaltsignal Jalousie AB, Aussenjalousie *)
	xDoIJalousieAuf: BOOL; (*Schaltsignal Jalousie AUF, Innenjalousie *)
```

```
        xDoIJalousieAb: BOOL; (* Schaltsignal Jalousie AB, Innenjalousie *)
END_STRUCT
END_TYPE
```

Daten für Büroflur bestehend aus Büroachsen

Kategorie:	Daten_Steuerung	
Name:	BueroFlur	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
Ausgang (extern):	**Datentyp:**	**Kommentar:**

Beschreibung:

Der Datentyp sammelt die Daten des Büroflurs bestehend aus 40 Büroachsen: Er besteht aus

- 41 Trennwänden, wobei die erste und letzte immer vorhanden sind;
- (bis zu) 41 Büros, wobei bei vorhandener Trennwand n die Bürodaten im Feld n stehen;
- 2 Funktionsbausteinen zum Empfang von EnOcean-Telegrammen;
- Maximal 2x40 Jalousietaster;
- Maximal 40 Raumthermostate – eins für jeden Raum;
- 5 DALI-Joblisten für 5 DALI-Busse;
- 5 Szenebausteinen, einer für jeden Bus;
- 5x64 Wippen, die mit Szenefunktionen hinterlegt sein können;
- 2x40 Lichttaster mit 4-fach Wippe

Implementierung:

```
TYPE BueroFlur :
STRUCT
	aBueroAchse: ARRAY[1..iAnzahlBueroachsen] OF BueroAchse;
	aTrennwaende: ARRAY[0..iAnzahlBueroachsen] OF BOOL;
	aBuero: ARRAY[1..iAnzahlBueroachsen] OF TBuero;
(* Kommunikation: EnOcean *)
	aEnOceanReceive: ARRAY[1..iAnzahlEnOceanKlemmen] OF FbEnoceanReceive;
(* Taster Jalousie *)
	aJalousieTaster: ARRAY[1..iAnzahlBueroachsen,1..2] OF TJalousieTaster;
(* Raumthermostat - Klima *)
	aRaumthermostat: ARRAY[1..iAnzahlBueroachsen] OF TRaumthermostat;

(* Lichtsteuerung *)
(*                                  *)
	aDALIJobListe: ARRAY[1..iAnzahlDALIKlemmen] OF FbDALI_Joblist;

(* Szeneaktoren DALI *)
	aSzene: ARRAY[1..iAnzahlDALIKlemmen] OF FbDALI_RecallScene;
	aRegistrierteSzeneWippe: ARRAY[1..iAnzahlDALIKlemmen,1..64] OF STRING(7);

	aLichtTaster: ARRAY[1..iAnzahlBueroachsen,1..2] OF TLichtTasterMitProgramm;
END_STRUCT
END_TYPE
```

Daten für Büros

Kategorie:	Daten_Steuerung	
Name:	TBuero	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
Ausgang (extern):	**Datentyp:**	**Kommentar:**
Beschreibung:		
Der Datentyp hält die Charakteristika eines Büros, hier nur die ID, welche eine Kunden-ID ist.		
Implementierung:		

```
TYPE TBuero :
STRUCT
        sID: STRING; (* ID *)
        sIDGUI: STRING;
END_STRUCT
END_TYPE
```

Dimmaktoren

Kategorie:	Daten_Steuerung	
Name:	TDimmAktor	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
Ausgang (extern):	**Datentyp:**	**Kommentar:**
Beschreibung:		
Der Datentyp enthält einen Funktionsbaustein zur Ansteuerung eines Dimmaktors und eine Liste der Wippen, die den Dimmaktor ansteuern.		
Implementierung:		

```
TYPE TDimmAktor :
STRUCT
        DimmAktor: FbDALI_DimmSingleButton;
        sListe: STRING;
END_STRUCT
END_TYPE
```

Jalousietaster

Kategorie:	Daten_Steuerung	
Name:	TJalousieTaster	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
Ausgang (extern):	**Datentyp:**	**Kommentar:**
Beschreibung:		
Der Datentyp sammelt die notwendigen Daten für einen Jalousietaster, insbesondere seine EnOcean-ID. Die Variable `xAngemeldet` wird nicht (mehr) verwendet.		
Implementierung:		

```
TYPE TJalousieTaster :
STRUCT
	JalousieTaster: FbF602xx_RockerSwitch_2_Rocker;
	bType: BYTE := 16#01;
	dwEnOceanID: DWORD;
	dwEnOceanIDGUI: DWORD;
	xAngemeldet: BOOL;
END_STRUCT
END_TYPE
```

Lichttaster mit Programm

Kategorie:	Daten_Steuerung	
Name:	TLichtTasterMitProgramm	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
Ausgang (extern):	**Datentyp:**	**Kommentar:**
Beschreibung:		
Der Datentyp sammelt die notwendigen Daten für einen Lichttaster mit 4 Wippen, insbesondere die EnOcean-ID und die 4 Programme, die bei den einzelnen Wippen hinterlegt sind.		
Implementierung:		

```
TYPE TLichtTasterMitProgramm :
STRUCT
	LichtTaster: FbF602xx_RockerSwitch_2_Rocker;
	bType: BYTE := 16#05;
	dwEnOceanID: DWORD;
	sProg: ARRAY[1..4] OF STRING; (* 4x Programme *)

	dwEnOceanIDGUI: DWORD;
	sProgGUI: ARRAY[1..4] OF STRING;
	aAlterSchaltzustand: ARRAY[1..4] OF BOOL; (* Szene reagiert auf positive Flanke *)
END_STRUCT
END_TYPE
```

Raumthermostat

Kategorie:	Daten_Steuerung	
Name:	TRaumthermostat	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
Ausgang (extern):	**Datentyp:**	**Kommentar:**
Beschreibung:		
Der Datentyp sammelt die notwendigen Daten für ein Raumthermostat, insbesondere die EnOcean-ID.		
Implementierung:		

```
TYPE TRaumthermostat :
STRUCT
        Raumthermostat: FbA510xx_RoomOperatingPanel;
        bType: BYTE := 16#05;
        dwEnOceanID: DWORD;
        dwEnOceanIDGUI: DWORD;
        rMaxSetpointCorrection: REAL := 2;
END_STRUCT
END_TYPE
```

A.3 Daten Visualisierung

Daten für die Steuerung

Kategorie:	Daten_Visualisierung	
Name:	T_ETL	
Typ:	Datentyp (Struct)	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingang (extern):	**Datentyp:**	**Kommentar:**
Ausgang (extern):	**Datentyp:**	**Kommentar:**
Beschreibung:		
Der Datentyp sammelt die Daten zur Kontrolle der Visualisierung. Die Variablen `xExtract` und `xLoad` definieren (flankengesteuert) das Auslesen der Lichtprogramme bzw. das Speichern der Lichtprogramme. Die Variablen in `aSteuerung` definieren die Auswahl der vier Visualisierungen, im Array `aAnzeigen` stehen die 6 (mit Trennwand 7) Nummern der 6 angezeigten Trennwände.		
Implementierung:		

```
TYPE BueroAchse :
STRUCT
	xExtract: BOOL := FALSE;
	xLoad: BOOL := FALSE;
	xExtract_Old: BOOL := FALSE;
	xLoad_Old: BOOL := FALSE;
	aSteuerung: ARRAY[1..4] OF BOOL;
	aSteuerung_alt: ARRAY[1..4] OF BOOL;
	iSlider: INT := 0;

	aAnzeigen: ARRAY[0..6] OF INT; (* Nummern der 6 angezeigten Achsen *)
(* Klima = 1 *)
(* Jalousie = 2 *)
(* Licht = 3 *)
(* System = 4 *)
	aNAnzeigen: ARRAY[1..4] OF BOOL;
END_STRUCT
END_TYPE
```

Anhang B: Funktionen der Bibliothek Bueroetage.lib

B.1 Initialisierung

Büroflur initialisieren

Kategorie:	Initialisierung	
Name:	BueroFlur_Initialisieren	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
`pBueroFlur`	`POINTER TO BueroFlur`	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
`BueroFlur_Initialisieren`	`BOOL`	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Initialisierung der 40 Büroachsen, insbesondere zuordnen der DALI- und EnOcean Kommunikationsklemmen und definieren der DALI-Nummern der Lichtstromkreise.		

Implementierung:

```
FUNCTION BueroFlur_Initialisieren : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	iAchse: INT := 0;
	n,m,o: INT;
END_VAR

(* DALI-Nummern vergeben *)
(* 8 Achsen pro DALI Klemme *)
FOR iAchse := 1 TO iAnzahlBueroachsen DO
	pBueroFlur^.aBueroAchse[iAchse].bDaliBus := INT_TO_BYTE((iAchse / 8)+1);

	pBueroFlur^.aBueroAchse[iAchse].bLS1 := INT_TO_BYTE(1+8*( (iAchse - 1) MOD 8));
	pBueroFlur^.aBueroAchse[iAchse].bLS2 := pBueroFlur^.aBueroAchse[iAchse].bLS1 +1;
	pBueroFlur^.aBueroAchse[iAchse].bLS3 := pBueroFlur^.aBueroAchse[iAchse].bLS2 +1;
END_FOR

(* FbDALI_StatusDimmValue initialisieren *)
FOR iAchse := 1 TO iAnzahlBueroachsen DO
	pBueroFlur^.aBueroAchse[iAchse].StatusDimmwert1(
		bShortAddress := pBueroFlur^.aBueroAchse[iAchse].bLS1,
		bModule_750_641 := pBueroFlur^.aBueroAchse[iAchse].bDaliBus);
END_FOR

(* Gruppen-Dimmen Achsen *)
FOR n := 1 TO iAnzahlDALIKlemmen DO
	FOR m:= 1 TO 8 DO
		pBueroFlur^.aBueroAchse[n].DimmenGruppe.DimmAktor(
		bAddress := INT_TO_BYTE(m),
		xGroup := TRUE,
		bReferenceaddress1 := INT_TO_BYTE((m-1)*8+1),
		bSwitchOnLevel := 50,
		xOff_at_MinLevel := TRUE,
		bModule_750_641 := INT_TO_BYTE(n)
		);
	END_FOR
END_FOR

(* Einzel-Dimmen Lampen *)
FOR n := 1 TO iAnzahlDALIKlemmen DO
	FOR m:= 1 TO 8 DO
		FOR o:= 1 TO 3 DO
			pBueroFlur^.aBueroAchse[n*m].DimmenEinzeln[o].DimmAktor(
			bAddress := INT_TO_BYTE(o+(m-1)*8),
			xGroup := FALSE,
			bSwitchOnLevel := 50,
			xOff_at_MinLevel := TRUE,
			bModule_750_641 := INT_TO_BYTE(n)
			);
		END_FOR
	END_FOR
END_FOR

(* Szenebausteine initialisieren *)
FOR n := 1 TO iAnzahlDALIKlemmen DO
	pBueroFlur^.aSzene[n](bModule_750_641:= INT_TO_BYTE(n));
END_FOR

(* EnOcean-Nummern vergeben *)
FOR iAchse := 1 TO iAnzahlBueroachsen DO
	IF iAchse <= 20 THEN pBueroFlur^.aBueroAchse[iAchse].bModule_750_642 := 1;
	ELSE pBueroFlur^.aBueroAchse[iAchse].bModule_750_642 := 2;
	END_IF
END_FOR
```

Gang initialisieren

Kategorie:	Initialisierung	
Name:	Gang_Initialisieren	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pGang	POINTER TO TGang	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
Gang_Initialisieren	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Initialisierung des Flurbereichs (Gang).		
Implementierung:		

```
FUNCTION Gang_Initialisieren : BOOL
VAR_INPUT
	pGang: POINTER TO TGang;
END_VAR
VAR
END_VAR

(* Zeitkonstante setzten *)
pGang^.TreppenLicht(dwT_10tel_s := 600);
```

Teeküche initialisieren

Kategorie:	Initialisierung	
Name:	TeeKueche_Initialisieren	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pTeeKueche	POINTER TO TTeekueche	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
TeeKueche_Initialisieren	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Initialisierung der Teeküche: Ausschalten des Lichts.		
Implementierung:		

```
FUNCTION TeeKueche_Initialisieren : BOOL
VAR_INPUT
	pTeeKueche: POINTER TO TTeeKueche;
END_VAR
VAR
END_VAR

pTeeKueche^.LichtAktor(xZenAus := TRUE);
pTeeKueche^.LichtAktor(xZenAus := FALSE);
```

Toiletten initialisieren

Kategorie:	Initialisierung	
Name:	Toiletten_Initialisieren	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pToiletten	POINTER TO TToiletten	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
Toiletten_Initialisieren	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Initialisierung der Toiletten: Zeitkonstante setzten.		
Implementierung:		

```
FUNCTION Toiletten_Initialisieren : BOOL
VAR_INPUT
	pToiletten: POINTER TO TToiletten;
END_VAR
VAR
END_VAR

(* Zeitkonstante setzten *)
pToiletten^.ToilettenLicht(dwT_10tel_s := 3000);
```

B.2 Kommunikation

DALI Joblisten

Kategorie:	Kommunikation	
Name:	DALI_Joblist	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pBueroFlur	POINTER TO BueroFlur	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
DALI_Joblist	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Initialisieren der DALI-Joblisten: Vergabe der DALI-Klemmennummer.		
Implementierung:		

```
FUNCTION DALI_Joblist : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n: INT;
END_VAR

FOR n := 1 TO iAnzahlDaliKlemmen DO
	pBueroFlur^.aDALIJobListe[n](bModule_750_641 := INT_TO_BYTE(n));
END_FOR
```

EnOcean Kommunikation

Kategorie:	Kommunikation	
Name:	EnOceanReceive	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pBueroFlur	POINTER TO BueroFlur	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
EnOceanReceive	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Initialisieren der EnOcean Kommunikation: Vergabe der Klemmennummer.		
Implementierung:		

```
FUNCTION EnOceanReceive : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n: INT;
END_VAR

FOR n := 1 TO iAnzahlEnOceanKlemmenKlemmen DO
	pBueroFlur^.aEnOceanReceive[n](bModule_750_642:= INT_TO_BYTE(n));
END_FOR
```

B.3 Steuerung

Steuerung der Jalousie-Funktion

Kategorie:	Steuerung	
Name:	SteuerungJalousie	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
`pBueroFlur`	`POINTER TO BueroFlur`	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
`SteuerungJalousie`	`BOOL`	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Steuerung der Jalousiefunktion. Globale Variablen steuern die Sicherheitsposition bzw. lassen die Beschattungsposition anfahren.		

Implementierung:

```
FUNCTION SteuerungJalousie : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n,n1: INT;
	m: INT;

	pJT: POINTER TO TJalousieTaster;
	pRS: POINTER TO FbF602xx_RockerSwitch_2_Rocker;
	xAup, xAdown, xIup, xIdown: BOOL;
END_VAR

n:= iAnzahlBueroachsen;
n1:= n;

WHILE n >= 0 DO
	IF pBueroFlur^.aTrennwaende[n] THEN (* Trennwand vorhanden, d.h., Index enthaelt
Buerodaten *)
		(* letztes Buero Jalousie steuern *)
		IF n < iAnzahlBueroachsen THEN
			WHILE n1 >  n DO
				pBueroFlur^.aBueroAchse[n1].AJalousie(xJalousieTasterAuf := xAup,
xJalousieTasterAb := xAdown,
				xBeschattungsPosAnfahren := xABeschattungsPosAnfahren, xSicherheit
:= xASicherheit);
				pBueroFlur^.aBueroAchse[n1].IJalousie(xJalousieTasterAuf := xIup,
xJalousieTasterAb := xIdown,
				xBeschattungsPosAnfahren := xIBeschattungsPosAnfahren, xSicherheit
:= xISicherheit);

				pBueroFlur^.aBueroAchse[n1].xDoAJalousieAb :=
pBueroFlur^.aBueroAchse[n1].AJalousie.xDoJalousieAb;
				pBueroFlur^.aBueroAchse[n1].xDoAJalousieAuf :=
pBueroFlur^.aBueroAchse[n1].AJalousie.xDoJalousieAuf;
				pBueroFlur^.aBueroAchse[n1].xDoIJalousieAb :=
pBueroFlur^.aBueroAchse[n1].IJalousie.xDoJalousieAb;
				pBueroFlur^.aBueroAchse[n1].xDoIJalousieAuf :=
pBueroFlur^.aBueroAchse[n1].IJalousie.xDoJalousieAuf;
				n1 := n1-1;
			END_WHILE
		END_IF;
		xAup := FALSE;
		xAdown := FALSE;
		xIup := FALSE;
		xIdown := FALSE;
	END_IF;

	FOR m := 1 TO 2 DO
		pJT := ADR(pBueroFlur^.aJalousieTaster[n,m]);

		IF pJT^.xAngemeldet THEN
			pRS := ADR(pJT^.JalousieTaster);
			pBueroFlur^.aEnOceanReceive[n](bModule_750_642 :=
pBueroFlur^.aBueroAchse[n].bModule_750_642);

			pRS^(typEnocean := pBueroFlur^.aEnOceanReceive[n].typEnocean,
			bTYPE :=pJT^.bType,
			dwID := pJT^.dwEnOceanID
			);
			xAup := xAup OR pRS^.xButton_AO;
			xAdown := xAdown OR pRS^.xButton_AI;
			xIup := xIup OR pRS^.xButton_BO;
			xIdown := xIdown OR pRS^.xButton_BI;
		END_IF;
	END_FOR;

	n := n-1;
END_WHILE
```

Steuerung der Klima-Funktion

Kategorie:	Steuerung	
Name:	SteuerungKlima	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pBueroFlur	POINTER TO BueroFlur	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
SteuerungKlima	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Steuerung der Klima-Funktion.		
Implementierung:		

```
FUNCTION SteuerungKlima : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n: INT;
	pRaumthermostat: POINTER TO TRaumthermostat;
	rTemperature: REAL;
	rSetpointCorrection: REAL;
	xComfortStandby: BOOL;
END_VAR

n:= iAnzahlBueroachsen;

WHILE n > 0 DO
	IF pBueroFlur^.aTrennwaende[n] THEN (* Trennwand vorhanden, d.h., Index enhaelt Buerodaten
*)
		pRaumthermostat := ADR(pBueroFlur^.aRaumthermostat[n]);
		(* Raumthermostat auslesen *)
		pRaumthermostat^.Raumthermostat(
		typEnocean :=
pBueroFlur^.aEnOceanReceive[pBueroFlur^.aBueroAchse[n].bModule_750_642].typEnocean,
		bType := pRaumthermostat^.bType,
		dwID := pRaumthermostat^.dwEnOceanID,
		rMaxSetpointCorrection := pRaumthermostat^.rMaxSetpointCorrection);

		rTemperature := pRaumthermostat^.Raumthermostat.rTemperature;
		rSetpointCorrection := pRaumthermostat^.Raumthermostat.rSetpointCorrection;
		xComfortStandby := pRaumthermostat^.Raumthermostat.xDB0_Bit0; (* Komfortbetrieb
bei Anwesenheit *)
	END_IF;
	(* Controller auswerten und Aktoren setzten *)
	pBueroFlur^.aBueroAchse[n].KlimaController(
	rRoomTemperature := rTemperature,
	rSetpointCorrection := rSetpointCorrection,
	xComfortStandby := xComfortStandby,
	xNightMode := xGlobalNightMode
	);
	pBueroFlur^.aBueroAchse[n].xCooling :=
pBueroFlur^.aBueroAchse[n].KlimaController.xCooling;
	pBueroFlur^.aBueroAchse[n].xHeating :=
pBueroFlur^.aBueroAchse[n].KlimaController.xHeating;
	n:= n-1;
END_WHILE
```

Steuerung Licht: Dimmen einer gesamten Büroachse

Kategorie:	Steuerung	
Name:	SteuerungLicht_DimmenBueroAchsen	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pBueroFlur	POINTER TO BueroFlur	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
SteuerungLicht_DimmenBueroAchsen	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Die Funktion ist ein Teil der Steuerung der Lichtfunktion und steuert die Dimmfunktion einer gesamten Büroachse.		
Implementierung:		

```
FUNCTION SteuerungLicht_DimmenBueroAchsen : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n: INT;
	xPush: BOOL;
	sListe: STRING;

	p: UINT; (* Anzahl angemeldeter Tasten *)
	NrT: INT;
	NrTT: INT;
	NrWippe: INT;
	pLT: POINTER TO TLichtTasterMitProgramm;
END_VAR

n:=1;

WHILE n <= iAnzahlBueroachsen DO
	IF pBueroFlur^.aBueroAchse[n].DimmenGruppe.sListe <>'' THEN
		sListe := pBueroFlur^.aBueroAchse[n].DimmenGruppe.sListe;
		xPush := FALSE;
		FOR p := 1 TO LEN(sListe)/7 DO	(* Anzahl der angemeldeten Tasten *)
			NrT := STRING_TO_INT(MID(sListe,2,(p-1)*7+1));
			NrTT := STRING_TO_INT(MID(sListe,1,(p-1)*7+4));
			NrWippe := STRING_TO_INT(MID(sListe,1,(p-1)*7+6));
			(* Auswertung Taster und Wippe *)
			pLT := ADR(pBueroFlur^.aLichtTaster[NrT,NrTT]);
			pLT^.LichtTaster(typEnocean :=
pBueroFlur^.aEnOceanReceive[pBueroFlur^.aBueroAchse[NrT].bModule_750_642].typEnOcean,
			bTYPE := pLT^.bType,
			dwID :=pLT^.dwEnOceanID
			);
			CASE NrWippe OF
			1: xPush := xPush OR pLT^.LichtTaster.xButton_AO;
			2: xPush := xPush OR pLT^.LichtTaster.xButton_AI;
			3: xPush := xPush OR pLT^.LichtTaster.xButton_BO;
			4: xPush := xPush OR pLT^.LichtTaster.xButton_BI;
			END_CASE
		END_FOR
		pBueroFlur^.aBueroAchse[n].DimmenGruppe.DimmAktor(xButton := xPush);
	END_IF

n := n+1;
END_WHILE
```

Steuerung Licht: Dimmen Lampen

Kategorie:	Steuerung
Name:	SteuerungLicht_DimmenLampen
Typ:	Funktion
Name der Bibliothek:	Bueroetage.lib
Anwendbar für:	Alle programmierbaren Feldbus-Controller
Verwendet:	

Eingangsparameter:	**Datentyp**	**Kommentar:**
pBueroFlur	POINTER TO BueroFlur	Pointer auf Datensatz

Rückgabewert:	**Datentyp**	**Kommentar:**
SteuerungLicht_DimmenLampen	BOOL	Wird nicht verwendet

Grafische Darstellung:

Funktionsbeschreibung:

Die Funktion ist ein Teil der Steuerung der Lichtfunktion und steuert die Dimmfunktion der einzelnen Lampen der Büroachsen.

Implementierung:

```
FUNCTION SteuerungLicht_DimmenLampen : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n,m: INT;
	xPush: BOOL;
	sListe: STRING;

	p: UINT; (* Anzahl angemeldeter Tasten *)
	NrT: INT;
	NrTT: INT;
	NrWippe: INT;
	pLT: POINTER TO TLichtTasterMitProgramm;
END_VAR

n:=1;

FOR m := 1 TO 3 DO
	WHILE n <= iAnzahlBueroachsen DO
		IF pBueroFlur^.aBueroAchse[n].DimmenEinzeln[m].sListe <> '' THEN
			sListe := pBueroFlur^.aBueroAchse[n].DimmenEinzeln[m].sListe;
			xPush := FALSE;
			FOR p := 1 TO LEN(sListe)/7 DO	(* Anzahl der angemeldeten Tasten *)
				NrT := STRING_TO_INT(MID(sListe,2,(p-1)*7+1));
				NrTT := STRING_TO_INT(MID(sListe,1,(p-1)*7+4));
				NrWippe := STRING_TO_INT(MID(sListe,1,(p-1)*7+6));
				(* Auswertung Taster und Wippe *)
				pLT := ADR(pBueroFlur^.aLichtTaster[NrT,NrTT]);
				pLT^.LichtTaster(typEnocean :=
pBueroFlur^.aEnOceanReceive[pBueroFlur^.aBueroAchse[NrT].bModule_750_642].typEnOcean,
				bTYPE := pLT^.bType,
				dwID :=pLT^.dwEnOceanID
				);
				CASE NrWippe OF
				1: xPush := xPush OR pLT^.LichtTaster.xButton_AO;
				2: xPush := xPush OR pLT^.LichtTaster.xButton_AI;
				3: xPush := xPush OR pLT^.LichtTaster.xButton_BO;
				4: xPush := xPush OR pLT^.LichtTaster.xButton_BI;
				END_CASE
			END_FOR
			pBueroFlur^.aBueroAchse[n].DimmenEinzeln[m].DimmAktor(xButton := xPush);
		END_IF

	n := n+1;
	END_WHILE
END_FOR
```

Steuerung Licht: Szene-Funktion

Kategorie:	Steuerung	
Name:	SteuerungLicht_SchaltenSzene	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
`pBueroFlur`	`POINTER TO BueroFlur`	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
`SteuerungLicht_SchaltenSzene`	`BOOL`	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Die Funktion ist ein Teil der Steuerung der Lichtfunktion und steuert Lichtszenen.		
Implementierung:		

```
FUNCTION SteuerungLicht_SchaltenSzene : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n,m,o: INT;
	sWippe: STRING;
	NrT: INT;
	NrTT: INT;
	NrWippe: INT;
	pLT: POINTER TO TLichtTasterMitProgramm;
	xStatus: BOOL;
	Programm: STRING;
	bProgAddress: BYTE;
	xProgGroup: BOOL;
	bProgSceneNo: BYTE;
	bProgModule_750_641: BYTE;
	iProgModule_750_641: INT;
END_VAR

FOR n := 1 TO iAnzahlDALIKlemmen DO
	FOR m := 1 TO 64 DO
		sWippe := pBueroFlur^.aRegistrierteSzeneWippe[n,m];
		IF sWippe <> '' THEN
			NrT := STRING_TO_INT(MID(sWippe,2,1));
			NrTT := STRING_TO_INT(MID(sWippe,1,4));
			NrWippe := STRING_TO_INT(MID(sWippe,1,6));
			(* Auswertung Taster und Wippe *)
			pLT := ADR(pBueroFlur^.aLichtTaster[NrT,NrTT]);
			pLT^.LichtTaster(typEnocean :=
pBueroFlur^.aEnOceanReceive[pBueroFlur^.aBueroAchse[NrT].bModule_750_642].typEnOcean,
				bTYPE := pLT^.bType,
				dwID :=pLT^.dwEnOceanID
				);
			CASE NrWippe OF
				1: xStatus := pLT^.LichtTaster.xButton_AO;
				2: xStatus := pLT^.LichtTaster.xButton_AI;
				3: xStatus := pLT^.LichtTaster.xButton_BO;
				4: xStatus := pLT^.LichtTaster.xButton_BI;
			END_CASE
			IF xStatus AND NOT pLT^.aAlterSchaltzustand[NrWippe] THEN
				(* Jetzt positive Flanke *)
				(* Also Programm abarbeiten *)
				Programm := pLT^.sProg[NrWippe];

				FOR o := 1 TO LEN(Programm)/10 DO
					bProgAddress := STRING_TO_BYTE(MID(Programm,2,(o-1)*10+1));
					xProgGroup := STRING_TO_BOOL(MID(Programm,1,(o-1)*10+4));
					bProgSceneNo := STRING_TO_BYTE(MID(Programm,2,(o-1)*10+6));
					bProgModule_750_641 := STRING_TO_BYTE(MID(Programm,1,(o-
1)*10+9));
					iProgModule_750_641 := BYTE_TO_INT(bProgModule_750_641);

					pBueroFlur^.aSzene[iProgModule_750_641](
					bAddress := bProgAddress,
					xGroup := xProgGroup,
					bSceneNo := bProgSceneNo,
					xSceneButton := TRUE,
					bModule_750_641 := bProgModule_750_641
					);
					pBueroFlur^.aSzene[iProgModule_750_641](
					xSceneButton := FALSE,
					bModule_750_641 := bProgModule_750_641
					);
				END_FOR
			END_IF
			pLT^.aAlterSchaltzustand[NrWippe] := xStatus;
		ELSE m:= 64;
		END_IF
	END_FOR
END_FOR
```

B.4 Steuerung der weiteren Etagenfunktionen

Steuerung Gang

Kategorie:	Steuerung_ETC	
Name:	SteuerungGang	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pGang	POINTER TO TGang	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
SteuerungGang	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Die Funktion implementiert die Steuerung der Lichtfunktion auf dem Flur (Gang).		
Implementierung:		

```
FUNCTION SteuerungGang : BOOL
VAR_INPUT
	pGang: POINTER TO TGang;
END_VAR
VAR
	xSignal: BOOL;
END_VAR

IF pGang^.xOn THEN
	pGang^.xDoLS1 := TRUE;
	pGang^. xDoLS2 := TRUE;
ELSE
	xSignal := pGang^.xTasterEin1 OR pGang^.xTasterEin2;
	xSignal := xSignal OR (pGang^.xSensor1 AND NOT pGang^.xSensor1_old);
	xSignal := xSignal OR (pGang^.xSensor2 AND NOT pGang^.xSensor2_old);
	xSignal := xSignal OR (pGang^.xSensor3 AND NOT pGang^.xSensor3_old);
	xSignal := xSignal OR (pGang^.xSensor4 AND NOT pGang^.xSensor4_old);
	pGang^.TreppenLicht(xTaster := xSignal);
	pGang^.xDoLS1 := pGang^.TreppenLicht.xAktor;
	pGang^.xDoLS2 := pGang^.TreppenLicht.xAktor;
END_IF

pGang^.xSensor1_old := pGang^.xSensor1;
pGang^.xSensor2_old := pGang^.xSensor2;
pGang^.xSensor3_old := pGang^.xSensor3;
pGang^.xSensor4_old := pGang^.xSensor4;
```

Steuerung Teeküche

Kategorie:	Steuerung_ETC	
Name:	SteuerungTeeKueche	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pTeeKueche	POINTER TO TTeeKueche	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
SteuerungTeeKueche	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Die Funktion implementiert die Steuerung der Lichtfunktion der Teeküche.		
Implementierung:		

```
FUNCTION SteuerungTeeKueche : BOOL
VAR_INPUT
	pTeeKueche: POINTER TO TTeeKueche;
END_VAR
VAR
END_VAR

pTeeKueche^.LichtAktor(xTaster:= pTeeKueche^.xToggle, xZenAus := xLichtZentralAus);
pTeeKueche^.xDoLicht := pTeeKueche^.LichtAktor.xAktor;
```

Steuerung Toiletten

Kategorie:	Steuerung_ETC	
Name:	SteuerungToiletten	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pToiletten	POINTER TO TToiletten	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
SteuerungToiletten	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Die Funktion implementiert die Steuerung der Lichtfunktion der Toiletten.		
Implementierung:		

```
FUNCTION SteuerungToilettene : BOOL
VAR_INPUT
	pToiletten: POINTER TO TToiletten;
END_VAR
VAR
END_VAR

xSignal := pToiletten^.xTaster OR pToiletten^.xSensor;
pToiletten^.ToilettenLicht(xTaster := xSignal);

pToiletten^.xDoLichtAktor := pToiletten^.ToilettenLicht.xAktor;

pToiletten^.xSensor_old := pToiletten^.xSensor;
```

B.5 Visualisierung

VISU_ReadData

Kategorie:	Visualsisierung	
Name:	VISU_ReadData	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pBueroFlur	POINTER TO BueroFlur	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
VISU_ReadData	BOOL	Wird nicht verwendet
Grafische Darstellung:		
Funktionsbeschreibung:		
Die Funktion liest aktuelle Daten der Steuerung aus und speichert eine Kopie der Daten, die dann in der Visualisierung angezeigt und manipuliert werden können.		
Implementierung:		

```
FUNCTION VISU_ReadData : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n,m, k: INT;
END_VAR

FOR n := 1 TO iAnzahlBueroAchsen DO
	(* Buero IDs *)
	pBueroFlur^.aBuero[n].sIDGUI := pBueroFlur^.aBuero[n].sID;

	(* Raumthermostate *)
	pBueroFlur^.aRaumthermostat[n].dwEnOceanIDGUI :=
pBueroFlur^.aRaumthermostat[n].dwEnOceanID;

	FOR m := 1 TO 2 DO
		(* Jalousie Taster *)
		pBueroFlur^.aJalousieTaster[n,m].dwEnOceanIDGUI :=
pBueroFlur^.aJalousieTaster[n,m].dwEnOceanID;
		(* Licht Taster *)
		pBueroFlur^.aLichtTaster[n,m].dwEnOceanIDGUI :=
pBueroFlur^.aLichtTaster[n,m].dwEnOceanID;
		FOR k := 1 TO 4 DO
			pBueroFlur^.aLichtTaster[n,m].sProgGUI[k] :=
pBueroFlur^.aLichtTaster[n,m].sProg[k];
		END_FOR
	END_FOR
END_FOR
```

VISU_StoreData

Kategorie:	Visualsisierung	
Name:	VISU_StoreData	
Typ:	Funktion	
Name der Bibliothek:	Bueroetage.lib	
Anwendbar für:	Alle programmierbaren Feldbus-Controller	
Verwendet:		
Eingangsparameter:	**Datentyp**	**Kommentar:**
pBueroFlur	POINTER TO BueroFlur	Pointer auf Datensatz
Rückgabewert:	**Datentyp**	**Kommentar:**
VISU_StoreData	BOOL	Wird nicht verwendet

Grafische Darstellung:

Funktionsbeschreibung:

Die Funktion schreibt die Daten aus der Visualisierung in die einzelnen Komponenten der Etagensteuerung. Hierbei werden insbesondere die Programme der Wippen der Taster analysiert und den jeweiligen Dimmaktoren zugeordnet.

Implementierung:

```
FUNCTION VISU_StoreData : BOOL
VAR_INPUT
	pBueroFlur: POINTER TO BueroFlur;
END_VAR
VAR
	n,m, k,l: INT;
	o: INT;
	sProgramm: STRING;
	sProgrammType: STRING;
	aSSCounter: ARRAY[1..iAnzahlDALIKlemmen] OF INT;

	sProgrammShort: STRING;
	iProgModule_750_641: INT;
	iProgrammAdresse: INT;
	iProgrammAchse: INT;
	iProgrammLampe: INT;
	sGruppe: STRING;
	iDALIBus : INT;
	xGruppe: BOOL;
	sProg: STRING;
END_VAR

FOR n := 1 TO iAnzahlBueroAchsen DO
	(* Buero IDs *)
	pBueroFlur^.aBuero[n].sID := pBueroFlur^.aBuero[n].sIDGUI;

	(* Raumthermostate *)
	pBueroFlur^.aRaumthermostat[n].dwEnOceanID :=
pBueroFlur^.aRaumthermostat[n].dwEnOceanIDGUI;

	FOR m := 1 TO 2 DO
		(* Jalousie Taster *)
		pBueroFlur^.aJalousieTaster[n,m].dwEnOceanID :=
pBueroFlur^.aJalousieTaster[n,m].dwEnOceanIDGUI;
		(* Licht Taster *)
		pBueroFlur^.aLichtTaster[n,m].dwEnOceanID :=
pBueroFlur^.aLichtTaster[n,m].dwEnOceanIDGUI;
		FOR k := 1 TO 4 DO
			pBueroFlur^.aLichtTaster[n,m].sProg[k] :=
pBueroFlur^.aLichtTaster[n,m].sProgGUI[k];
		END_FOR
	END_FOR
END_FOR

(* Alte Licht-Programm löschen *)
FOR n := 1 TO iAnzahlBueroAchsen DO
	pBueroFlur^.aBueroAchse[n].DimmenGruppe.sListe := ''; (* Gruppen-Dimmen *)
	FOR k := 1 TO iAnzahlDALIKlemmen DO
		FOR l := 1 TO 64 DO
			pBueroFlur^.aRegistrierteSzeneWippe[k,l] := ''; (* Szenen *)
		END_FOR
	END_FOR
	FOR k := 1 TO 3 DO
```

```
                pBueroFlur^.aBueroAchse[n].DimmenEinzeln[k].sListe := ''; (* Einzel-Dimmen *)
        END_FOR
END_FOR

(* Licht-Programme Speichern *)
FOR n := 1 TO iAnzahlDALIKlemmen DO
        aSSCounter[n] := 1;
END_FOR

(* Programme Schreiben *)
FOR n := 1 TO iAnzahlBueroAchsen DO
        FOR m := 1 TO 2 DO
                FOR k := 1 TO 4 DO
                        sProgramm := pBueroFlur^.aLichtTaster[n,m].sProg[k];
                        sProgrammType := MID(sProgramm,3,1);

                        IF sProgrammType = 'SS:' THEN
                                FOR o := 1 TO (LEN(sProgramm)-3)/10 DO
                                        sProgrammShort := MID(sProgramm,7,3+(o-1)*10);
                                        iProgModule_750_641 := STRING_TO_INT(MID(sProgramm,2,3+(o-
1)*10+7));

        pBueroFlur^.aRegistrierteSzeneWippe[iProgModule_750_641,aSSCounter[iProgModule_750_641]]
:= sProgrammShort;
                                        aSSCounter[iProgModule_750_641] :=
aSSCounter[iProgModule_750_641] + 1;
                                END_FOR
                        END_IF

                        IF sProgrammType = 'DE:' OR sProgrammType = 'DG:' THEN
                                xGruppe := sProgrammType = 'DG';
                                sGruppe := BOOL_TO_STRING(xGruppe);
                                FOR o := 1 TO (LEN(sProgramm)-3)/5 DO
                                        (* DE: Achse, 1,2, Wippe *)
                                        (* DG: Analog *)
                                        iProgrammAdresse := STRING_TO_INT(MID(sProgramm,2,(3+(o-
1)*5+1)));
                                        iDALIBus := STRING_TO_INT(MID(sProgramm,2,(3+(o-1)*5+4)));
                                        iProgrammLampe := 1+(iProgrammAdresse-1) MOD 3;
                                        iProgrammAchse := (iDALIBus -1)*8+ (iProgrammLampe-1)/8 +1;

                                        sProg := CONCAT(INT_TO_STRING(iProgrammAchse),'.');
                                        sProg := CONCAT(sProg,INT_TO_STRING(m));
                                        sProg := CONCAT(sProg,'.');
                                        sProg := CONCAT(sProg,INT_TO_STRING(k));
                                        sProg := CONCAT(sProg,';');

                                        IF xGruppe THEN
                                                (* Besser: Grenzen Überprüfen *)
                                                pBueroFlur^.aBueroAchse[n].DimmenGruppe.sListe :=
CONCAT(pBueroFlur^.aBueroAchse[n].DimmenGruppe.sListe,sProg);
                                        ELSE

        pBueroFlur^.aBueroAchse[n].DimmenEinzeln[iProgrammLampe].sListe :=
CONCAT(pBueroFlur^.aBueroAchse[n].DimmenEinzeln[iProgrammLampe].sListe,sProg);;
                                        END_IF
                                END_FOR
                        END_IF
                END_FOR
        END_FOR
END_FOR
```

Erklärung nach §17 Abs. 5 der Bachelor-Prüfungsordnung

Hiermit versichere ich, dass die von mir vorgelegte Prüfungsleistung selbstständig und ohne unzulässige fremde Hilfe erstellt worden ist. Alle verwendeten Quellen sind in der Arbeit so aufgeführt, dass Art und Umfang der Verwendung nachvollziehbar sind.

Dortmund, 21.08.2015