PROJEKTARBEIT

Smarter LSS

Autor:
SCHAUER Philipp - REIGL Julian

Version:
-V260125x2150

INHALTSVERZEICHNIS

1 URHEBERRECHT

Die Nutzung und Vervielfältigung dieser Projektarbeit unterliegt den Bestimmungen des österreichischen Urheberrechts, insbesondere § 42 Abs. 4 UrhG:

> (4) Jede natürliche Person darf von einem Werk einzelne Vervielfältigungsstücke auf anderen als den in Abs. 1 genannten Trägern [Papier oder einem ähnlichen Träger, die jedermann zum eigenen Gebrauch freistehen] zum privaten Gebrauch und weder für unmittelbare noch mittelbare kommerzielle Zwecke herstellen.

Gemäß § 42 Abs. 5 UrhG dürfen diese Vervielfältigungsstücke nicht dazu verwendet werden, das Werk der Öffentlichkeit zugänglich zu machen.

Da die Erstellung dieser Projektarbeit mit erheblichem Zeit- und Arbeitsaufwand verbunden war, ist es nur fair und angemessen, dass Personen, die keinen direkten Zugang zu uns haben, sich mit ihrem Anliegen direkt an uns wenden. Jegliche unautorisierte Weitergabe oder Veröffentlichung der Projektarbeit ist untersagt.

Bei Anfragen stehen wir unter *julian.reigl@outlook.de* oder *phili.schauer@gmx.at* gerne zur Verfügung.

2 PROJEKTTAGEBUCH

2.1 07.OKT 2025

REI und SCP haben die benötigten Teile für das Projekt erhalten und gemeinsam die *Projektdefinition* ausgearbeitet. Dabei wurden die Ziele, der Aufbau und die geplante Vorgehensweise festgelegt.

2.2 14.OKT 2025

Zu Beginn hat SCP die Beschreibungen und Anleitungen zu den Komponenten vorgelesen. REI hat anschließend ein kleines Demonetzwerk aufgebaut. Dafür wurde eine Fritz!Box als Router verwendet, an die ein *PAC2200* und ein *POC1100* angeschlossen wurden.

Um das Testen und Arbeiten zu erleichtern, haben beide alle Komponenten auf eine Hutschiene montiert und passend verdrahtet. Danach hat REI das Netzwerk konfiguriert.

Anschließend haben REI und SCP überprüft, ob die Geräte im Netzwerk erreichbar sind – beispielsweise über die Weboberfläche oder die Powerconfig-App von Siemens.

2.3 21.OKT 2025

Heute war SCP krank, sodass nur allein gearbeitet werden konnte. REI hat sich in den gesamten drei Stunden intensiv mit dem *Modbus*-Protokoll beschäftigt. Zunächst wurde die Funktionsweise des Protokolls analysiert, um ein besseres Verständnis für die Kommunikation zwischen den Geräten zu bekommen.

Anschließend wurden die ersten Verbindungsversuche mit den einzelnen Geräten durchgeführt, wobei Daten ausgelesen und die Reaktionsfähigkeit der Geräte überprüft wurden.

2.4 04.NOV 2025

Es wurde festgelegt, dass ein C#-Webserver entwickelt wird, da dieser umfangreiche Möglichkeiten bietet und viel Freiheit im Design erlaubt. Zudem wurde der Grundstein des Projekts gelegt und die grundlegende Struktur erstellt.

Im weiteren Verlauf wurde der Dashboard-Controller programmiert und das Routing bzw. Mapping eingerichtet, um die Verbindungen zwischen den einzelnen Seiten und Funktionen herzustellen.

2.5 11.NOV 2025

Es wurde eine C#-Applikation entwickelt, die über Modbus-TCP die Daten aus unseren Geräten ausliest. Dafür haben wir zunächst die notwendigen Klassen, Modelle und Properties definiert, um die ausgelesenen Werte strukturiert verarbeiten zu können.

Die Kommunikation mit den Geräten wurde mithilfe von TcpClient und NetworkStream umgesetzt. Unsere Anwendung sendet einen eigenen Bitstream an das jeweilige Gerät, empfängt die Antwort über den Stream und filtert anschließend gezielt die relevanten Daten heraus. Danach wurden die Ergebnisse passend formatiert, damit sie übersichtlich dargestellt und weiterverarbeitet werden können.

2.6 18.Nov 2025

Heute wurde weiter am Modbus-System gearbeitet. Das gesamte System wurde umfassend umstrukturiert, um den Server schneller und insgesamt leistungsfähiger zu machen.

Im Zuge der Optimierung wurde die komplette Speicherstruktur überarbeitet. Der Server holt nun automatisch alle zwei Minuten die aktuellen Daten von den Geräten ab und lädt diese in einen temporären Zwischenspeicher.

Wenn anschließend eine Seite aufgerufen wird, greift sie nicht mehr direkt auf die Geräte zu, sondern liest die Daten aus dem temporären Speicher aus. Dadurch wird die Datenanzeige deutlich schneller und die Geräte werden entlastet.

2.7 25.Nov 2025

Heute haben wir das Design für das Dashboard ausgearbeitet. Zunächst haben wir gemeinsam die Aufteilung der Oberfläche festgelegt und entschieden, welche Werte an welcher Position angezeigt werden sollen. Dabei haben wir uns für eine Variante auf Basis eines einfachen HTML-Grundgerüsts entschieden, da diese Lösung eine unkomplizierte und flexible Möglichkeit bietet, das Dashboard übersichtlich zu gestalten.

Im Anschluss haben wir die einzelnen Tiles vorbereitet, die später die verschiedenen Messwerte darstellen werden. Damit ist bereits die Grundlage geschaffen, um beim nächsten Mal die echten Daten im Dashboard anzeigen zu lassen.

2.8 02.Dec 2025

Am 02. Dezember haben wir das Dashboard „smart“ gemacht, indem wir die Möglichkeit geschaffen haben, eine JSON-Datei anzulegen. In dieser Datei kann die Struktur und Anordnung des Dashboards konfiguriert werden, sodass es flexibel an unterschiedliche Anforderungen angepasst werden kann.

Durch diese Lösung lässt sich festlegen, welche Elemente angezeigt werden, wie sie angeordnet sind und welche Werte dargestellt werden sollen. Dadurch kann das Dashboard je nach Anwendung optimal gestaltet werden, ohne den Quellcode direkt ändern zu müssen.

2.9 09.Dec 2025

Am 09. Dezember wurde ein neuer Tab eingeführt, der Geräte-Tab. Dieser dient dazu, alle Geräte im Netzwerk übersichtlich anzuzeigen, ohne diese direkt im Dashboard darstellen zu müssen.

Im Geräte-Tab werden die Stammdaten der einzelnen Geräte angezeigt. Dazu gehören unter anderem die IP-Adressen, die Modbus-IDs sowie der Online- bzw. Offline-Status der Geräte. Dadurch ist es möglich, den aktuellen Zustand des Netzwerks schnell zu überblicken und bei Bedarf gezielt Informationen zu einzelnen Geräten abzurufen.

2.10 16.Dec 2025

Am 16. Dezember hat SCP das Plakat für den Tag der offenen Tür gestaltet. Zusätzlich wurden alle notwendigen Vorbereitungen für die Präsentation getroffen. Dazu zählte unter anderem die inhaltliche Abstimmung, die visuelle Aufbereitung sowie die Planung des Präsentationsablaufs, um das Projekt verständlich und ansprechend vorstellen zu können.

2.11 23.Dec 2025

Es wurde der aktuelle Projektstatus präsentiert. Dabei wurden der bisherige Fortschritt, die umgesetzten Funktionen sowie der aktuelle Stand des Systems vorgestellt und erläutert.

2.12 13.Jan 2026

Wir haben einen Rittal-Verteiler besorgt, um unseren Demoaufbau für den Tag der offenen Tür zu erweitern und anschaulicher darstellen zu können, womit wir uns beschäftigen.

Zunächst wurde die Montageplatte mechanisch vorbereitet. Dabei haben wir die notwendigen Bohrungen durchgeführt sowie Hutschienen und Verdrahtungskanäle montiert. Damit wurde die Grundlage für den weiteren elektrischen Aufbau und die übersichtliche Präsentation der Komponenten geschaffen.

2.13 16.Jan 2026

Wir begonnen, den zuvor mechanisch aufgebauten Verteiler zu verdrahten. Zunächst wurde der Verteiler in EPLAN gezeichnet, um einen übersichtlichen und korrekten Schaltplan zu erstellen.

Im Anschluss an die Planung erfolgte die Verdrahtung gemäß dem erstellten Plan, sodass der Verteiler strukturiert und nachvollziehbar aufgebaut wurde. Dadurch konnte sichergestellt werden, dass alle Anschlüsse korrekt ausgeführt und für spätere Erweiterungen gut dokumentiert sind.

2.14 17.Jan 2026

Verdrahtung des Verteilers war vollständig abgeschlossen. Anschließend haben wir mit den Tests begonnen. Dazu wurden Lasten angeschlossen, um zu überprüfen, welche Werte die Leitungsschutzschalter (LSS) tatsächlich erfassen und melden.

Zusätzlich wurde die Visualisierung einem Stresstest unterzogen, um sicherzustellen, dass auch bei höherer Belastung alle Daten korrekt verarbeitet und angezeigt werden. Dabei konnte festgestellt werden, dass das System stabil läuft und die Software fehlerfrei programmiert ist.

2.15 19.Jan 2026

Visualisierung wurde in ihrer finalen Form fertiggestellt und die zugehörige Dokumentation geschrieben. Diese Arbeiten haben die gesamten drei Unterrichtsstunden in Anspruch genommen. In der nächsten Zeit steht das vollständige Abschließen des Projekts an, damit im neuen Semester mit einem neuen, größeren Projekt begonnen werden kann.

3 PROJEKTDEFINITION

3.1 PROJEKTBESCHREIBUNG

Es soll untersucht werden, wie der Leitungsschutzschalter mit integriertem Fehlerlichtbogenschutz vom Typ Siemens *5SV6016-7MC16* funktioniert und welche Eigenschaften und Schutzmechanismen er bietet. Es soll ermittelt werden, welche Messgrößen verfügbar sind und in welchen Anwendungsbereichen das Gerät sinnvoll eingesetzt werden kann. Darüber hinaus soll geprüft werden, wie der Schalter mit einem drahtlosen Kommunikationsmodul verbunden werden kann, um seine Daten weiterzuleiten und auszuwerten.

3.2 ZIELSETZUNG

Es soll das Ziel verfolgt werden, ein System zu erstellen, das den AFDD/LS-Schalter, das Connection-Modul (*7KN1111-0MC00*) und das Power Monitoring Device (*7KM2200-2EA40-1EA1*) miteinander verbindet. Es soll überprüft werden, wie die drahtlose Kommunikation eingerichtet werden kann, welche Daten über Modbus TCP übertragen werden können und wie diese Daten visualisiert werden können. Es soll eine praxisnahe Lösung entstehen, die die Schutzfunktionen, Schaltzustände und elektrischen Messgrößen erfasst und darstellt.

3.3 PROJEKTUMFANG

Es soll analysiert werden, welche Funktionen der AFDD/LS-Schalter bereitstellt und wie das Connection-Modul zur drahtlosen Übertragung der Daten eingesetzt werden kann. Es soll untersucht werden, wie die Modbus TCP-Kommunikation aufgebaut wird und welche Daten ausgelesen werden können. Es soll geprüft werden, wie das Power Monitoring Device in das Netzwerk integriert werden kann, um zusätzliche Messwerte zu erfassen. Außerdem soll festgehalten werden, welche Schritte notwendig sind, um die Geräte zu verbinden und die Daten in einer Visualisierung darzustellen.

3.4 ERWARTETES ERGEBNIS

Es soll ein funktionsfähiges System entstehen, das aus dem Leitungsschutzschalter, dem Connection-Modul und dem Power Monitoring Device besteht. Es soll die Daten drahtlos übertragen und über Modbus TCP verfügbar machen. Es soll möglich sein, Schaltzustände, Messwerte und Schutzfunktionen in einer Visualisierung darzustellen. Das Projekt soll aufzeigen, wie moderne elektrische Schutz- und Überwachungsgeräte miteinander vernetzt werden können.

3.5 DOKUMENTATION

Es soll eine ausführliche Dokumentation erstellt werden, die den Aufbau der Geräte, die Einrichtung der drahtlosen Verbindung, die Modbus-Adressen und die Konfiguration der Visualisierung beschreibt. Es soll eine Schritt-für-Schritt-Anleitung entstehen, die nachvollziehbar macht, wie die Geräte verbunden werden, wie die Daten übertragen werden und wie die Visualisierung umgesetzt wird.

4 DOKUMENTATION

4.1 SENTRON, MESSGERÄT, 7KM PAC2200 (7KM2200-2EA40-1EA1)

Siemens Industry Mall

4.1.1 PRODUKT BESCHREIBUNG [1]

Das SENTRON PAC2200 (7KM2200-2EA40-1EA1) ist ein elektronisches Energie- und Leistungsmessgerät von Siemens, das für den Einsatz in dreiphasigen Niederspannungsnetzen konzipiert ist. Es wird typischerweise in Schaltschränken oder Unterverteilungen auf einer 35-mm-Hutschiene montiert und dient dort zur kontinuierlichen Erfassung, Anzeige und Weitergabe elektrischer Mess- und Energiedaten. Sein Hauptzweck liegt in der transparenten Überwachung von Energieflüssen, der Analyse des Verbrauchs sowie der Unterstützung von Energiemanagement- und Automatisierungssystemen.

Das Gerät ist in der Lage, Spannungen, Ströme und daraus abgeleitete Leistungsgrößen in Echtzeit zu messen. Bei der hier genannten Variante können Ströme bis 65 A direkt gemessen werden, ohne dass externe Stromwandler erforderlich sind. Für größere Anlagen ist alternativ auch der Betrieb mit externen Stromwandlern möglich, wodurch das Messgerät flexibel in unterschiedlichsten Netzgrößen eingesetzt werden kann. Es unterstützt klassische dreiphasige Netzformen und erfasst sowohl Leiter-Leiter- als auch Leiter-Neutral-Spannungen, wodurch eine vollständige Netzbewertung möglich ist.

Aus den gemessenen Grundgrößen berechnet das PAC2200 fortlaufend Wirkleistung, Blindleistung und Scheinleistung, sowohl phasenweise als auch als Gesamtsumme. Zusätzlich werden Wirk-, Blind- und Scheinenergie über die Zeit integriert und als Energiezählerstände gespeichert. Weitere wichtige Netzparameter wie Frequenz und Leistungsfaktor (cos φ) werden ebenfalls erfasst. Die Messung erfolgt als True-RMS-Messung, wodurch auch verzerrte oder nicht sinusförmige Strom- und Spannungsverläufe korrekt ausgewertet werden können – ein entscheidender Vorteil in modernen Anlagen mit Frequenzumrichtern, Schaltnetzteilen oder nichtlinearen Lasten.

Zur lokalen Bedienung und Kontrolle verfügt das Gerät über ein integriertes LCD-Display, auf dem die aktuellen Messwerte, Energiezählerstände und Statusinformationen übersichtlich dargestellt werden. Damit eignet sich das PAC2200 nicht nur für die zentrale Datenerfassung, sondern auch für die direkte Vor-Ort-Diagnose durch Instandhaltung oder Betriebspersonal. Die Parametrierung und Navigation erfolgt über Tasten am Gerät oder über angebundene Software-Systeme.

Ein wesentliches Merkmal des PAC2200 ist seine Kommunikationsfähigkeit. Das Gerät besitzt eine integrierte Ethernet-Schnittstelle mit Modbus-TCP, über die Messwerte zyklisch an übergeordnete Systeme übertragen werden können. Dadurch lässt es sich problemlos in SCADA-Systeme, Gebäudeleittechnik, Energiemanagement-Software oder SPS-Umgebungen integrieren. In solchen Systemen dienen die erfassten Daten unter anderem zur Lastüberwachung, zur Identifikation von Energieverbrauchern, zur Analyse von Lastspitzen oder zur langfristigen Optimierung des Energieeinsatzes.

Konstruktiv ist das PAC2200 kompakt aufgebaut und für den dauerhaften Einsatz in Verteilungen ausgelegt. Die Front besitzt in der Regel eine Schutzart IP40, ausreichend für den Schaltschrankbetrieb. Die Versorgung des Geräts erfolgt direkt aus dem zu messenden Netz, sodass keine separate Hilfsspannung erforderlich ist. Die Anschlüsse sind als Schraubklemmen ausgeführt, was eine sichere und normgerechte Verdrahtung ermöglicht.

Im Inneren eines solchen Messgeräts befinden sich typischerweise präzise Spannungs- und Strommesseingänge, die über analoge Messschaltungen an A/D-Wandler angebunden sind. Ein Mikrocontroller oder DSP verarbeitet die digitalisierten Signale, berechnet die elektrischen Kenngrößen und verwaltet die Energiezähler. Zusätzlich sind Speicherbausteine für Zählerstände und Parameter vorhanden sowie Kommunikationsmodule für Ethernet und gegebenenfalls weitere

Schnittstellen. Schutzbeschaltungen sorgen dafür, dass das Gerät gegen Überspannungen und Störeinflüsse aus dem Netz abgesichert ist.

In der Praxis wird das SENTRON PAC2200 vor allem in Industrieanlagen, Zweck- und Bürogebäuden sowie größeren Wohnanlagen eingesetzt. Dort übernimmt es Aufgaben wie Sub-Metering, Energieverbrauchsüberwachung einzelner Anlagenteile, Unterstützung bei Energieaudits oder die Bereitstellung von Messdaten für ein übergeordnetes Energiemanagement. Insgesamt stellt es ein vielseitiges und technisch ausgereiftes Messgerät dar, das sowohl für die lokale Anzeige als auch für die systemweite Energieanalyse geeignet ist.

4.1.2 Modbus Register

4.1.2.1 Basisgrößen

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zugriff
0x0001	1	2	Spannung UL1-N	Float	V	-	-	-	-	R
0x0003	3	2	Spannung UL2-N	Float	V	-	-	-	-	R
0x0005	5	2	Spannung UL3-N	Float	V	-	-	-	-	R
0x0007	7	2	Spannung UL1-L2	Float	V	-	-	-	-	R
0x000D	13	2	Strom L1	Float	A	-	-	-	-	R
0x0019	25	2	Wirkleistung L1	Float	W	-	-	-	-	R
0x001B	27	2	Wirkleistung L2	Float	W	-	-	-	-	R
0x001D	29	2	Wirkleistung L3	Float	W	-	-	-	-	R
0x0025	37	2	Leistungsfaktor L1	Float	-	-	-	-	-	R
0x0037	55	2	Frequenz	Float	Hz	-	-	-	-	R
0x0041	65	2	Wirkleistung Summe	Float	W	-	-	-	-	R

4.1.2.2 Energiezähler

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zugriff
0x0321	801	2	Wirkenergie Bezug T1	Float	Wh	-	-	-	-	R
0x0323	803	2	Wirkenergie Bezug T2	Float	Wh	-	-	-	-	R
0x0325	805	2	Wirkenergie Abgabe T1	Float	Wh	-	-	-	-	R
0x0327	807	2	Wirkenergie Abgabe T2	Float	Wh	-	-	-	-	R
0x00D7	215	2	Universalzähler	U32	-	-	-	-	-	RW

4.1.2.3 Einstellungen und Parameter

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zugriff
0xC34F	49999	2	Nennstrom Anzeige	U32	A	1	10000	-	-	RW
0xC351	50001	2	Anschlussart	U32	W	-	-	-	0=3P4W 4=1P2W	RW
0xC35B	50011	2	Primärstrom (Wandler)	U32	A	1	99999	-	-	RW
0xC35D	50013	2	Sekundärstrom (Wandler)	U32	A	-	-	-	1A oder 5A	RW
0xC37F	50047	2	Dialogsprache	U32	-	-	-	-	0=DE, 1=EN	RW
0xC43B	50235	2	Zeitzone	S32	min	-	-	-	Offset in Minuten	RW
0xC43D	50237	2	Ausgangs-Impulsteiler	U32	kWh	0	1	-	0=1kWh 1=10kWh	RW

4.1.2.4 Kommandos und Status

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zugriff
0xEA61	60001	1	Geräte-Reset	S8	-	-	-	-	Startet das Gerät neu	W
0xEA66	60006	1	Tarif umschalten	S8	-	-	-	-	0=Haupt- 1=Nebentarif	W
0xEA6A	60010	1	Reset Zähler	S8	-	-	-	-	Tages-/Monatsspeicher	W
0x006C	108	0*	Parametrierung aktiv	S8	-	-	-	-	Bit-Zugriff (FC 0x02)	R

4.2 Powercenter 1100 (7KN1111-0MC00)

Siemens Industry Mall

4.2.1 Produkt Beschreibung [2]

Das SENTRON Powercenter 1100 (Bestellnummer 7KN1111-0MC00) ist ein Daten-Transceiver von Siemens, der speziell dazu entwickelt wurde, kommunikations- und messfähige SENTRON-Geräte in elektrischen Verteilungen zu vernetzen, zu sammeln und auszuwerten. Anders als klassische Messgeräte wie das PAC2200, die direkt elektrische Größen erfassen, dient das Powercenter als

zentrale Datensammel- und Kommunikationsschnittstelle, über die Mess- und Statusdaten aus unterschiedlichsten Schutz-, Mess- und Schalteinrichtungen gebündelt, gespeichert und weitergereicht werden können.

Mechanisch ist das Powercenter ebenfalls als DIN-Hutschienen-Modul ausgeführt, das in Schaltschränken oder Verteilungen installiert wird. Es ist sehr kompakt (ca. 18 mm breit, 90 mm hoch, 70 mm tief) und besitzt eine Schutzart IP20, was seiner Eignung für den gewöhnlichen Schaltschrankbetrieb entspricht. Die Versorgung erfolgt mit 24 V DC (SELV), was in industriellen Installationen üblich ist, um Sicherheit und Stabilität der elektronischen Komponenten zu gewährleisten.

Für die Kommunikation verfügt das Powercenter über mehrere Hardware-Schnittstellen: Es besitzt z. B. zwei Ethernet-Ports, über die Daten nach außen übertragen werden können, sowie Bluetooth-Funktechnik für die drahtlose Kommunikation direkt vor Ort. Die drahtlose Verbindung nutzt moderne Bluetooth-Standards (z. B. Bluetooth 5.1) und ermöglicht es, Daten von bis zu 24 angeschlossenen SENTRON-COM-Geräten innerhalb einer Verteilung zu empfangen. Diese Geräte können z. B. messtechnisch erweiterte Schutzschalter, Fernantriebe oder Hilfsschalter sein, die über Funk ihre Mess- und Statusinformationen an das Powercenter senden.

Im Betrieb übernimmt das Powercenter die Erfassung, Zwischenspeicherung und Weiterleitung von Mess- und Statusdaten über einen Zeitraum von Tagen. Die gespeicherten Messwerte können lokal über ein mobiles Endgerät per Bluetooth abgerufen werden oder über Industrie-Kommunikationsprotokolle wie Modbus TCP und EtherNet/IP an übergeordnete Systeme (z. B. SCADA, Energiemanagement-Software oder Gebäudeautomationssysteme) weitergeleitet werden. Diese Protokollunterstützung ermöglicht eine zentrale Auswertung, Visualisierung und Archivierung von Zustandsdaten über alle angeschlossenen Geräte hinweg.

Ein typisches Einsatzszenario ist die Integration vieler einzelner Mess- und Schutzgeräte in einer einheitlichen Kommunikationsstruktur: statt jedes Gerät einzeln auszulesen oder verdrahtet zu überwachen, fungiert das Powercenter als Kommunikationshub, der drahtlos Daten von Schutzschaltern oder Messpunkten sammelt und sie zentral verfügbar macht. Dadurch lassen sich Leistungs- und Zustandsdaten von ganzen Verteilernetzen oder Unterverteilungen effizienter überwachen und verarbeiten.

Im Inneren des Powercenters befinden sich neben den Funk- und Ethernet-Modulen typischerweise ein Prozessor zur Datenverarbeitung, ein Speicher für zwischengespeicherte Messdaten und Konfigurationsinformationen, sowie Schnittstellen-Controller, die die drahtlose und kabelgebundene Kommunikation steuern. Sicherheitsfunktionen wie verschlüsselte Kommunikation, Schreibschutz gegen unerwünschte Änderungen und rollenbasierte Zugriffskontrolle sorgen für den sicheren Betrieb im industriellen Umfeld.

Zusammengefasst ist das Powercenter 1100 ein zentrales Kommunikations- und Datenerfassungsgerät für verteilte Schutz- und Messgeräte in modernen Niederspannungsnetzen. Es erweitert elektrische Verteilungen um eine schichtübergreifende Kommunikationsfähigkeit, die lokale Mess- und Statusdaten sammelt, speichert und netzwerkfähig bereitstellt – ein wichtiges Element im Energiemanagement, Monitoring und in der digitalen Automatisierung elektrischer Anlagen.

4.2.2 MODBUS REGISTER

4.2.2.1 MESSWERTE

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zugriff
0x0A00	2560	2	Alarm Zustand	U32	-	-	-	0	Bitwert 0 = inaktiv, 1 = aktiv. Bitfeld: 0: Alarm Betriebsstunden mit Belastungsstrom 1: Alarm Betriebsstunden,	RO

									2: Alarm Schaltspiele 3: Alarm Auslösezähler, 4: Alarm Temperaturüberschreitung 5: Alarm 1 Überstrom, 6: Alarm 2 Überstrom 7: Alarm 1 Unterstrom, 8: Alarm 2 Unterstrom 9: Alarm 1 Überspannung, 10: Alarm 2 Überspannung 11 : Alarm 1 Unterspannung, 12 : Alarm 2 Unterspannung 13: Schalter ausgelöst, 14: Auslösung Fehlerlichtbogen 15 : Auslösung Überspannung, 16: Alarm Kurzschlussauslösezähler 17: Alarm: AFDD Tripschwelle Unterschreitung. 18: Selbsttest fehlgeschlagen 19: Auslösung Unterspannung, 20: Auslösung Fehlerstrom 24-31: Geräte spezifisch. RCA: 24: RCD Test Zähler Alarm, 25: ARD Fehler, 26: RCD Test fehlgeschlagen, 27: IR Test fehlgeschlagen, 28: IR Test Warnung MCB RCM: 24: RCM AC Voralarm, 25: RCM AC Alarm, 26: RCM RMS Voralarm, 27: RCM RMS Alarm MSP: 24: Überlastauslösung Alarm, 25: Kurzschlussauslösung Alarm ECPD: 24: RCM Vorlarm, 25: RCM Alarm, 26: Verzögerte Auslösung, 27: Unverzögerte Auslösung, 28: ARD fehlgeschlagen, 29: Übertemperaturabschaltung, 30: Alarm für verzögerte Auslösungen, 31: Alarm EIN blockiert	
0x0A12	2578	4	Betriebsstundenzähler gesamt	FP64	s	-	-	0	-	RO
0x0A3D	2621	1	BLE Empfangssignalstärke RSSI	S16	dBm	-127	20	-	-	RO
0x0A45	2629	4	Zeit- und Synchronisationsstatus	U8[8]	-	-	-	946684800	{uint32_t PARAM_DATE_TIME, uint32_t SYNC_STATUS}	RO
0x0A49	2633	1	Aktiver Funkkanal	U16	-	0	26	-	-	RO
0x0C00	2683	2	Temperatur	FP32	°C	0	0	NaN	-	RO
0x0C02	3072	2	Mittelwert Temperatur	FP32	°C	0	0	NaN	-	RO

4.2.2.2 PARAMETER

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zugriff
0x0002	2	1	Hersteller ID	U16	-	-	-	0x2A	= 42 = Siemens	RO
0x0003	3	10	Artikelnummer	UCHAR [20]	-	-	-	-	ASCII-Zeichen	RO
0x000D	13	8	Seriennummer	UCHAR [16]	-	-	-	-	ASCII-Zeichen	RO
0x0015	21	1	Hardware Version	U16	-	-	-	-	-	RO
0x0016	22	2	Software Version	UCHAR [4]	-	-	-	-	ASCII-Zeichen	RO
0x001D	29	12	Anlagenkennzeichen (Name)	UCHAR [32]	-	-	-	-	ASCII-Zeichen	RW
0x002D	45	11	Einbauort	UCHAR [22]	-	-	-	-	ASCII-Zeichen	RW
0x0038	56	8	Installationsdatum	UCHAR [16]	-	-	-	-	ASCII-Zeichen	RW
0x005E	94	1	Zielmarkt	U16	-	1	3	1	1= IEC, 2=UL, 3=IEC+UL	RO
0x0061	97	1	Blinkmodus zur Grätelokalisierung	U16	-	0	1	0	0 = Blinken stoppen, 1 = Blinken für 10 Sek	CMD
0x006D	109	1	Hardware Ausgabestand	U16	-	0	10	0	0 = nicht verfügbar, 1 = Stand 1 usw.	RO
0x0070	112	1	Gerätevariante	U16	0	0xFFFE	0	0	0 = unbekannt, 1 = 5ST3 COM AS+FC, 2 = 3NA COM Fuse, 3 = 5SL6 COM MCB, 4 = 5SV6 COM AFDD, 5 = 5ST3 COM RCA Standard, 6 = 5ST3 COM RCA mit RCD/IR Test, 7 = 5SL6 COM MCB RCM, 8 = 5SV8 COM RCM,	RO

								9 = 3RV2 COM MSP, 10 = 5SV8 COM RCM (1 Kanal), 11= 5TY1 COM ECPD, 14 = POC1100, 16 = POC2000, 18 = 5TT4 COM DIDO	
0x0A16	2582	1	Alarm der Betriebsstunden ein/aus	U16	-	0	1	0	0 = aus, 1 = ein	RW
0x0A17	2583	4	Grenzwert Betriebsstunden	FP64	sec	60	9E+08	1,58E+08	-	RW
0x0E05	3589	2	Grenzwert Temperaturüberschreitung	FP32	°C	20	110 3NA COM: 120 Powercenter: 85	105 3NA COM: 115 Powercenter: 80	-	RW
0x0E07	3591	2	Hysterese Temperatur Alarm	FP32	%	0	10	10	-	RW
0x0E09	3593	2	Letzte Diagnose Log OID	U32	sec	-	-	-	-	RO
0x0E58	3672	2	Letzte Message Log OID	U32	-	-	-	-	-	RO

4.2.2.3 KOMMUNIKATION

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zugriff
0x0200	512	3	Ethernet MAC Adresse	UCHAR [6]	-	-	-	!	-	RO
0x0203	515	2	DHCP (automatische IP Vergabe) ein/aus	U32	-	0	1	1	0 = DHCP aus, 1 = DHCP ein	RW
0x0205	517	2	SNTP Server IP-Adresse	U32	-	0	0xFFFFFFFF	0.0.0.0		RW
0x0207	519	2	SNTP Client Mode	U32	-	0	2	0	0 = aus, 1 = aktiv, 2 = broadcast	RW
0x0208	521	2	Firewall ein/aus	U32	-	0	1	0	0 = aus, 1 = ein	RW
0x0209	523	2	IP Port Nummer	U32	-	0	0xFFFF	502	-	RW
0x0210	525	2	Statische IP-Adresse	U32	-	0	0xFFFFFFFF	0.0.0.0	-	RW
0x0211	527	2	Statische Subnetzmaske	U32	-	0	0xFFFFFFFF	255.255.255.0	-	RW
0x0212	529	2	Statisches Gateway	U32	-	0	0xFFFFFFFF	0.0.0.0	-	RW
0x029D	669	2	Aktuelle IP-Adresse	U32	-	-	-	0.0.0.0	-	RO
0x029F	671	2	Aktuelle Subnetzmaske	U32	-	-	-	0.0.0.0	-	RO
0x02A1	673	2	Aktuelle Gateway Adresse	U32	-	-	-	0.0.0.0	-	RO
0x0300	768	1	Bluetooth Steuerung	U16	-	1	3	!	1= BLE ein, 2 = BLE aus, 3 = Passkey zurückse	CMD
0x0301	769	1	Bluetooth Status	U16	-	-	-	-	-	RO
0x0302	770	1	Bluetooth Sendeleistung	S16	dBm	-18	4	2	-	RW
0x0303	771	3	Bluetooth Geräteadresse	U8[6]	-	-	-	-	-	RO
0x0306	774	2	Bluetooth Passkey	U32	-	-	-	-	-	RW
0x0400	1024	2	Datum/Zeit (UTC)	U32	-	0	0xFFFFFFFF	9,5E+08	UNIX_TS seit 01.01.1970	RW
0x041A	1050	1	Funk Sendeleistung	S13	dBm	-18	2	0	-	RW
0x044C	1100	1	Zähler Parameteränderung Endgerät 1	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x044D	1101	1	Zähler Parameteränderung Endgerät 2	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x044E	1102	1	Zähler Parameteränderung Endgerät 3	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x044F	1103	1	Zähler Parameteränderung Endgerät 4	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0450	1104	1	Zähler Parameteränderung Endgerät 5	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0451	1105	1	Zähler Parameteränderung Endgerät 6	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0452	1106	1	Zähler Parameteränderung Endgerät 7	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0453	1107	1	Zähler Parameteränderung Endgerät 8	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0454	1108	1	Zähler Parameteränderung Endgerät 9	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0455	1109	1	Zähler Parameteränderung Endgerät 10	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0456	1110	1	Zähler Parameteränderung Endgerät 11	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0457	1111	1	Zähler Parameteränderung Endgerät 12	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0458	1112	1	Zähler Parameteränderung Endgerät 13	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0459	1113	1	Zähler Parameteränderung Endgerät 14	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x045A	1114	1	Zähler Parameteränderung Endgerät 15	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x045B	1115	1	Zähler Parameteränderung Endgerät 16	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x045C	1116	1	Zähler Parameteränderung Endgerät 17	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO

0x045D	1117	1	Zähler Parameteränderung Endgerät 18	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x045E	1118	1	Zähler Parameteränderung Endgerät 19	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x045F	1119	1	Zähler Parameteränderung Endgerät 20	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0460	1120	1	Zähler Parameteränderung Endgerät 21	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0461	1121	1	Zähler Parameteränderung Endgerät 22	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0462	1122	1	Zähler Parameteränderung Endgerät 23	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0463	1123	1	Zähler Parameteränderung Endgerät 24	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Endgerät einen Parameter verändert wird	RO
0x0464	1124	1	Zähler Parameteränderung Powercenter	U16	-	0	0xFFFF	0	Zählt hoch, wenn am Powercenter einen Parame	RO
0x04B1	1201	1	Schalter Status Endgerät 1	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04B2	1202	1	Schalter Status Endgerät 2	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04B3	1203	1	Schalter Status Endgerät 3	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04B4	1204	1	Schalter Status Endgerät 4	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04B5	1205	1	Schalter Status Endgerät 5	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04B6	1206	1	Schalter Status Endgerät 6	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04B7	1207	1	Schalter Status Endgerät 7	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04B8	1208	1	Schalter Status Endgerät 8	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04B9	1209	1	Schalter Status Endgerät 9	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04BA	1210	1	Schalter Status Endgerät 10	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04BB	1211	1	Schalter Status Endgerät 11	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04BC	1212	1	Schalter Status Endgerät 12	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO

0x04BD	1213	1	Schalter Status Endgerät 13	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04BE	1214	1	Schalter Status Endgerät 14	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04BF	1215	1	Schalter Status Endgerät 15	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04C0	1216	1	Schalter Status Endgerät 16	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04C1	1217	1	Schalter Status Endgerät 17	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04C2	1218	1	Schalter Status Endgerät 18	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04C3	1219	1	Schalter Status Endgerät 19	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04C4	1220	1	Schalter Status Endgerät 20	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04C5	1221	1	Schalter Status Endgerät 21	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
	1222	1	Schalter Status Endgerät 22	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04C6	1223	1	Schalter Status Endgerät 23	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x04C7	1224	1	Schalter Status Endgerät 24	U16	-	-	-	-	0 = unbekannt 1 = AUS 2 = EIN 3 = Ausgelöst 4= reserve 5 = Standby	RO
0x0744	1860	1	Manuelle Funkkanalauswahl ein/aus	U16	-	0	1	0	0 = aus 1 = ein Nur vor 1. Pairing einstellbar Ab Version 2.0 verfügbar	RW
0x0745	1861	1	Funkkanalauswahl	U16	-	11	26	11	Nur vor 1. Pairing einstellbar Ab Version 2.0 verfügbar	RW
0x4000	16384	1	Pairing Status (1)	U16	-	-	-	-	0 = nicht verfügbar (MAC/Installationscode fehlen) 1 = Pairing läuft 2 = Pairing abgeschlossen 4 = Pairing fehlgeschlagen 5 = Pairing Timeout 9 = Entfernen läuft 11 = Entfernen timeout	RO
0x4001	16385	1	Pairing Status (2)	U16	-	-	-	-	siehe Registernr. 16384	RO

0x4002	16386	1	Pairing Status (3)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4003	16387	1	Pairing Status (4)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4004	16388	1	Pairing Status (5)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4005	16389	1	Pairing Status (6)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4006	16390	1	Pairing Status (7)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4007	16391	1	Pairing Status (8)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4008	16392	1	Pairing Status (9)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4009	16393	1	Pairing Status (10)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x400A	16394	1	Pairing Status (11)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x400B	16395	1	Pairing Status (12)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x400C	16396	1	Pairing Status (13)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x400D	16397	1	Pairing Status (14)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x400E	16398	1	Pairing Status (15)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x400F	16399	1	Pairing Status (16)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4010	16400	1	Pairing Status (17)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4011	16401	1	Pairing Status (18)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4012	16402	1	Pairing Status (19)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4013	16403	1	Pairing Status (20)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4014	16404	1	Pairing Status (21)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4015	16405	1	Pairing Status (22)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4016	16406	1	Pairing Status (23)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4017	16407	1	Pairing Status (24)	U16	-	-	-	-	siehe Registernr. 16384	RO
0x4064	16484	1	Device Status (1)	U16	-	-	-	-	0=IDLE (Kein Endgerät gepaired) 1 Offline, 2 = Verbinden läuft, 3 = Verbunden	RO
0x4065	16485	1	Device Status (2)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4066	16486	1	Device Status (3)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4067	16487	1	Device Status (4)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4068	16488	1	Device Status (5)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4069	16489	1	Device Status (6)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x406A	16490	1	Device Status (7)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x406B	16491	1	Device Status (8)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x406C	16492	1	Device Status (9)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x406D	16493	1	Device Status (10)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x406E	16494	1	Device Status (11)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x406F	16495	1	Device Status (12)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4070	16496	1	Device Status (13)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4071	16497	1	Device Status (14)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4072	16498	1	Device Status (15)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4073	16499	1	Device Status (16)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4074	16500	1	Device Status (17)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4075	16501	1	Device Status (18)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4076	16502	1	Device Status (19)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4077	16503	1	Device Status (20)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4078	16504	1	Device Status (21)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x4079	16505	1	Device Status (22)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x407A	16506	1	Device Status (23)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x407B	16507	1	Device Status (24)	U16	-	-	-	-	siehe Registernr. 16484	RO
0x45DC	17884	50	Koppeln/Entfernen des Endgeräts	U8[100]	-	0	3	!	0 = Keine Aktion, 1 = Start Pairing, 3 = Gerät entfernen/Pairing abbrechen (Beispiel: Pairing starten für End Gerät 1 und 3: Byte 1 & 3 auf den Wert 1 setzen, die anderen sind 0)	CMD
0x460E	17934	2	Identifikation Status (1)	U32	-	-	-	-	0 = Initial, 1 = Gerät unbekannt, 2 = Fehler bitte Gerät entfernen, 3 = Gerät erkannt	RO
0x4610	17936	2	Identifikation Status (2)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4612	17938	2	Identifikation Status (3)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4614	17940	2	Identifikation Status (4)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4616	17942	2	Identifikation Status (5)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4618	17944	2	Identifikation Status (6)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x461A	17946	2	Identifikation Status (7)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x461C	17948	2	Identifikation Status (8)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x461E	17950	2	Identifikation Status (9)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4620	17952	2	Identifikation Status (10)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4622	17954	2	Identifikation Status (11)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4624	17956	2	Identifikation Status (12)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4626	17958	2	Identifikation Status (13)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4628	17960	2	Identifikation Status (14)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x462A	17962	2	Identifikation Status (15)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x462C	17964	2	Identifikation Status (16)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x462E	17966	2	Identifikation Status (17)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4630	17968	2	Identifikation Status (18)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4632	17970	2	Identifikation Status (19)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4634	17972	2	Identifikation Status (20)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4636	17974	2	Identifikation Status (21)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x4638	17976	2	Identifikation Status (22)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x463A	17978	2	Identifikation Status (23)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x463C	17980	2	Identifikation Status (24)	U32	-	-	-	-	siehe Registernr. 17934	RO
0x1000	4096	2	Status der aktuellen Delayed-Acknowledge-Anfrage	U32	-	-	-	-	0x00000000: Idle. Kein Schreib-Prozess laufend, Warten auf Prozess	RW

									0x00000001: Läuft. Endgerät beschäftigt. Schhreibanfrage wird abgelehnt. 0x00000002: Erfolg: Schhreibanfrage wird abgelehnt bis in IDLE State versetzt. 0x00000003: Fehlgeschlagen. Schhreibanfrage wird abgelehnt bis in IDLE State versetzt.	
0x1002	4098	121	Delayed Response Log Information	UCHAR[242]	-	-	-	-		RO
0x107B	4219	2	Status der aktuellen Delayed-Acknowledge Read Log Anfrage	U32	-	-	-	-	0x00000000: Idle. Kein Lese-Prozess laufend, Warten auf Prozess 0x00000001: Läuft. Endgerät beschäftigt. Schhreibanfrage wird abgelehnt. 0x00000002: Erfolg: Schhreibanfrage wird abgelehnt bis in IDLE State versetzt. 0x00000003: Fehlgeschlagen. Schhreibanfrage wird abgelehnt bis in IDLE State versetzt.	RW
0x4A00	18944	1	Status lokales Usermanagement	U8	-	-	-	-	Bitwert: 0=nein, 1=ja Bitfeld: 0: Mind. 1 Superuser angelegt?	RO
0x4A02	18946	1	Schreibschutz Status	U8	-	-	-	-	Bitwert: 0=deaktiviert, 1=aktiviert Bitfeld: 0: Mechanischer Schreibschutz	RO

4.3 BRANDSCHUTZSCHALTER-LS-KOMBI MESSFUNKTION (5SV6016-7MC16)

Siemens Industry Mall

4.3.1 PRODUKT BESCHREIBUNG [3]

Das Siemens 5SV6016-7MC16 ist ein moderner Brandschutzschalter bzw. kombinierter Leitungsschutzschalter mit integrierter Mess- und Kommunikationsfunktion aus der SENTRON-Serie von Siemens. Dieser Schalter vereint in einem kompakten, 1-TE-Breiten-Gerät mehrere Schutz- und Monitoring-Funktionen, wie sie in klassischen Niederspannungs-Haupt- und Unterverteilungen gefordert werden.

Als Kombinationsgerät schützt er elektrische Stromkreise vor Kurzschluss und Überlast wie ein herkömmlicher Leitungsschutzschalter, nutzt dabei aber zusätzlich eine eingebaute Brandschutz-Erkennung für sogenannte Fehlerlichtbögen (AFDD). Diese Technologie erkennt gefährliche Lichtbögen im Stromkreis, die bei herkömmlichen Schutzschaltern oft nicht geortet werden, und unterbricht den Strom bevor ein Brand entstehen kann. Das schließt eine Schutzlücke, die bei normalen Leitungsschutzschaltern bestehen würde.

Das Gerät ist 1+N-polig ausgeführt und für 230 V-Wechselstromnetze mit einem Bemessungsstrom von 16 A ausgelegt; es besitzt die Auslöse-Charakteristik C, was bedeutet, dass es bei Kurzschluss- und Überlastzuständen entsprechend der C-Kennlinie anspricht. Das Bemessungsschaltvermögen liegt bei ca. 6 kA, was typisch für Schutzgeräte in Wohn- und Gewerbeinstallationen ist.

Neben den Schutzfunktionen bringt der 5SV6016-7MC16 eine Messfunktion mit, die es erlaubt, elektrische Größen wie Strom oder eventuell über ausgewählte Parameter auch Energie- bzw. Leistungswerte zu erfassen. Diese Messdaten können lokal genutzt oder über einen Kommunikationskanal weitergegeben werden. Die Kommunikation ist in diesem Gerät drahtlos möglich (z. B. Funk), sodass es z. B. in ein zentrales Energiemanagement- oder Monitoring-System eingebunden werden kann. Für den Aufbau solcher Kommunikationsstrukturen wird meist ein Datentransceiver wie das oben beschriebene Powercenter eingesetzt, das mehrere dieser Schutz- und Messgeräte vernetzen kann.

Mechanisch entspricht der Schalter dem Standard für Schaltschränke und Verteilungen: er wird auf einer Hutschiene montiert, besitzt eine Schutzart, die für den Schaltschrankinnenraum geeignet ist (typisch IP20) und nimmt mit einer Breite von nur eine Teilungseinheit sehr wenig Platz ein. Zur flexiblen Erweiterung stehen Zubehör-Module wie Hilfs- oder Fehlersignalschalter zur Verfügung, die zusätzliche Schalt- oder Signalwege realisieren können.

Zusammengefasst stellt dieser Brandschutzschalter-LS-Kombi ein multifunktionales Schutz- und Messgerät dar, das über die reine Überstrom-Absicherung hinausgeht, indem es frühzeitig Brandrisiken erkennt und gleichzeitig Mess- und Kommunikationsfunktionen integriert, die in vernetzten, modernen Installationen zur Analyse und Steuerung elektrischer Anlagen wichtig sind.

4.3.2 MODBUS REGISTER

4.3.2.1 MESSWERTE

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zu-griff
0x0A00	2560	2	Alarm Zustand	U32	-	-	-	0	Bitwert 0 = inaktiv, 1 = aktiv. Bitfeld: 0: Alarm Betriebsstunden mit Belastungsstrom 1: Alarm Betriebsstunden, 2: Alarm Schaltspiele 3: Alarm Auslösezähler, 4: Alarm Temperaturüberschreitung 5: Alarm 1 Überstrom, 6: Alarm 2 Überstrom 7: Alarm 1 Unterstrom, 8: Alarm 2 Unterstrom	RO

									9: Alarm 1 Überspannung, 10: Alarm 2 Überspannung 11 : Alarm 1 Unterspannung, 12 : Alarm 2 Unterspannung 13: Schalter ausgelöst, 14: Auslösung Fehlerlichtbogen 15 : Auslösung Überspannung, 16: Alarm Kurzschlussauslösezähler 17: Alarm: AFDD Tripschwelle Unterschreitung. 18: Selbsttest fehlgeschlagen 19: Auslösung Unterspannung, 20: Auslösung Fehlerstrom 24-31: Geräte spezifisch. RCA: 24: RCD Test Zähler Alarm, 25: ARD Fehler, 26: RCD Test fehlgeschlagen, 27: IR Test fehlgeschlagen, 28: IR Test Warnung MCB RCM: 24: RCM AC Voralarm, 25: RCM AC Alarm, 26: RCM RMS Voralarm, 27: RCM RMS Alarm MSP: 24: Überlastauslösung Alarm, 25: Kurzschlussauslösung Alarm ECPD: 24: RCM Vorlarm, 25: RCM Alarm, 26: Verzögerte Auslösung, 27: Unverzögerte Auslösung, 28: ARD fehlgeschlagen, 29: Übertemperaturabschaltung, 30: Alarm für verzögerte Auslösungen, 31: Alarm EIN blockiert	
0x0A02	2562	4	Betriebsstundenzähler mit Belastungsstrom	FP64	s	-	-	0	-	RO
0x0A12	2578	4	Betriebsstundenzähler gesamt	FP64	s	-	-	0	-	RO
0x0A21	2593	2	Anzahl mechanischer Schaltspiele	FP32	-	-	-	0	-	RO
0x0A2A	2602	2	Anzahl der Auslösungen	FP32	-	-	-	0	-	RO
0x0A3E	2622	1	Funk Empfangssignalstärke RSSI	S16	dBm	-127	20	-	-	RO
0x0A40	2624	2	Anzahl der Kurzschlussauslösungen	FP32	-	-	-	-	-	RO
0x0C00	3072	2	Temperatur	FP32	°C	-	-	NaN	-	RO
0x0C02	3074	2	Mittelwert Temperatur	FP32	°C	-	-	NaN	-	RO
0x0C04	3076	2	Strom	FP32	A	-	-	NaN	-	RO
0x0C06	3078	2	Mittelwert Strom	FP32	A	-	-	NaN	-	RO
0x0C08	3080	2	Maximalwert Strom	FP32	A	-	-	NaN	-	RO
0x0C0A	3082	2	Spannung	FP32	V	-	-	NaN	-	RO
0x0C0C	3084	2	Netzfrequenz	FP32	Hz	-	-	NaN	-	RO
0x0C0E	3086	2	Wirkleistung	FP32	W	-	-	NaN	-	RO
0x0C10	3088	2	Scheinleistung	FP32	VA	-	-	NaN	-	RO
0x0C12	3090	2	Gesamtblindleistung Qtot	FP32	var	-	-	NaN	-	RO
0x0C14	3092	2	Leistungsfaktor	FP32	-	-	-	NaN	-	RO
0x0C16	3094	4	Bezogene Wirkenergie	FP64	Wh	-	-	NaN	-	RO
0x0C1A	3098	4	Abgegebene Wirkenergie	FP64	Wh	-	-	NaN	-	RO
0x0C1E	3102	4	Bezogene Blindenergie	FP64	varh	-	-	NaN	-	RO
0x0C22	3106	4	Abgegebene Blindenergie	FP64	varh	-	-	NaN	-	RO
0x0C26	3110	1	Schalter Status	U16	-	-	-	0	0 = Status unbekannt 1 = Aus ohne Auslösung 2 = Ein 3 =Ausgelöst 4 =Ausgelöst, aber Hebel ein/blockiert 5 = Standby (für ECPD) 6 = Standby tripped (für ECPD)	RO

4.3.2.2 PARAMETER

Register (Hex)	**Register (dezimal)**	**Register Länge**	**Bezeichnung**	**Format**	**Einheit**	**Min**	**Max**	**Default**	**Bemerkung**	**Zugriff**
0x0002	2	1	Hersteller ID	U16	-	-	-	0x2A	= 42 = Siemens	RO
0x0003	3	10	Artikelnummer	UCHAR [20]	-	-	-	-	ASCII-Zeichen	RO
0x000D	13	8	Seriennummer	UCHAR [16]	-	-	-	-	ASCII-Zeichen	RO
0x0015	21	1	Hardware Version	U16	-	-	-	-	-	RO
0x0016	22	2	Software Version	UCHAR [4]	-	-	-	-	ASCII-Zeichen	RO
0x001D	29	12	Anlagenkennzeichen (Name)	UCHAR	-	-	-	-	ASCII-Zeichen	RW

				[32]						
0x002D	45	11	Einbauort	UCHAR [22]	-	-	-	-	ASCII-Zeichen	RW
0x0038	56	8	Installationsdatum	UCHAR [16]	-	-	-	-	ASCII-Zeichen	RW
0x005E	94	1	Zielmarkt	U16	-	1	3	1	1= IEC 2=UL 3=IEC+UL	RO
0x005F	95	1	Bemessungsstrom In	U16	A	-	-	-	-	RO
0x0060	96	1	Auslösekennlinie	U16	-	0	3	0	0 = Nicht definiert 1 = Characteristic A 2 = Characteristic B 3 = Characteristic C	RO
0x0061	97	1	Blinkmodus zur Grätelokalisierung	U16	-	0	1	0	0 = Blinken stoppen, 1 = Blinke	CMD
0x006D	109	1	Hardware Ausgabestand	U16	-	0	10	0	0 = nicht verfügbar, 1 = Stand 1	RO
0x0070	112	1	Gerätevariante	U16	0	0xFFFE	0	0	0 = unbekannt, 1 = 5ST3 COM AS+FC, 2 = 3NA COM Fuse, 3 = 5SL6 COM MCB, 4 = 5SV6 COM AFDD, 5 = 5ST3 COM RCA Standard, 6 = 5ST3 COM RCA mit RCD/I 7 = 5SL6 COM MCB RCM, 8 = 5SV8 COM RCM, 9 = 3RV2 COM MSP, 10 = 5SV8 COM RCM (1 Kana 11= 5TY1 COM ECPD, 14 = POC1100, 16 = POC2000, 18 = 5TT4 COM DIDO	RO
0x0091	145	1	Phasen Information	U16	-	0	3	0	0 = nicht verfügbar 1 = L1 2 = L2 3 = L3	RW
0x0A06	2566	1	Betriebsstunden mit Belastungsstrom Alarm ein/aus	U16	-	0	1	0	0 = aus 1 = ein	RW
0x0A07	2567	1	Minimaler Belastungsstrom ab dem der Betriebsstundenzähler aktiv wird	U16	%	5	90	70	-	RW
0x0A08	2568	4	Grenzwert Betriebsstunden mit Belastungsstrom	FP64	s				-	RW
0x0A16	2582	1	Alarm der Betriebsstunden ein/aus	U16	-	0	1	0	0 = aus 1 = ein	RW
0x0A17	2583	4	Grenzwert Betriebsstunden	FP64	sec	60	9E+08	1,58E+08	-	RW
0x0A23	2595	1	Alarm für Schaltspiele ein/aus	U16	-	0	1	0	0 = aus, 1 = ein	RW
0x0A24	2596	2	Grenzwert mechanische Schaltspiele	FP32	-	0	10000	1000		RW
0x0A2C	2604	1	Auslösezähler Alarm ein/aus	U16	-	0	1	0	0 = aus, 1 = ein	RW
0x0A2D	2605	2	Grenzwert Auslösezähler	FP32	-	1	10000	30		RW
0x0A42	2626	1	Kurzschlussauslösezähler Alarm ein/aus	U16	-	0	1	1	0 = aus, 1 = ein	RW
0x0A43	2627	2	Grenzwert Kurzschlussauslösungen	FP32	-	1	50	6 3RV2 COM: 3		RW
0x0E02	3586	2	Zeitperiode für die Mittelwertbildung der Temperatur	U32	sec	60	3600	600		RW
0x0E04	3588	1	Temperatur Alarm ein/aus	U16	-	0	1	1	0 = aus 1 = ein	RW
0x0E05	3589	2	Grenzwert Temperaturüberschreitung	FP32	°C	20	110 3NA COM: 120 Powercenter: 85	105 3NA COM: 115 Powercenter: 80	-	RW
0x0E07	3591	2	Hysterese Temperatur Alarm	FP32	%	0	10	10	-	RW
0x0E09	3593	2	Letzte Diagnose Log OID	U32	sec	-	-	-	-	RO
0x0E0D	3597	1	Zeitperiode für die Mittelwertbildung des Stroms	U16	s	3	3600	10	-	RW
0x0E0E	3598	1	Stromüberschreitung Alarm1 ein/aus	U16		0	1	1	0 = aus 1 = ein	RW
0x0E0F	3599	2	Grenzwert Stromüberschreitung Alarm1	FP32	%	0	150	80	In % vom Nennstrom In	RW
0x0E11	3601	2	Hysterese Stromüberschreitung Alarm1	FP32	%	0	10	5	-	RW
0x0E17	3607	1	Stromüberschreitung Alarm2 ein/aus	U16	-	0	1	0	0 = aus 1 = ein	RW
0x0E18	3608	2	Grenzwert Stromüberschreitung Alarm2	FP32	%	0	150	90	-	RW
0x0E1A	3610	2	Hysterese Stromüberschreitung Alarm2	FP32	%	0	10	5	-	RW
0x0E20	3616	1	Stromunterschreitung Alarm1 ein/aus	U16	-	0	1	0	0 = aus 1 = ein	RW
0x0E21	3617	2	Grenzwert Stromunterschreitung Alarm1	FP32	%	0	105	10	-	RW
0x0E23	3619	2	Hysterese Stromunterschreitung Alarm1	FP32	%	0	10	5	-	RW
0x0E29	3625	1	Stromunterschreitung Alarm2 ein/aus	U16	-	0	1	0	0 = aus 1 = ein	RW
0x0E2A	3626	2	Grenzwert Stromunterschreitung Alarm2	FP32	%	0	105	5	In % vom Nennstrom In	RW
0x0E2C	3628	2	Hysterese Stromunterschreitung Alarm2	FP32	%	0	10	5	-	RW
0x0E32	3634	1	Spannungsüberschreitung Alarm1 ein/aus	U16	-	0	1	0	0 = aus 1 = ein	RW

0x0E33	3635	2	Grenzwert Spannungsüberschreitung Alarm1	FP32	V	220	270	250	-	RW
0x0E35	3637	2	Hysterese Spannungsüberschreitung Alarm1	FP32	%	0	10	10	-	RW
0x0E3B	3643	1	Spannungsüberschreitung Alarm2 ein/aus	U16	-	0	1	0	0 = aus 1 = ein	RW
0x0E3C	3644	2	Grenzwert Spannungsüberschreitung Alarm2	FP32	V	220	270	270	-	RW
0x0E3E	3646	2	Hysterese Spannungsüberschreitung Alarm2	FP32	%	0	10	10	-	RW
0x0E44	3652	1	Spannungsunterschreitung Alarm1 ein/aus	U16	-	0	1	0	0 = aus 1 = ein	RW
0x0E45	3653	2	Grenzwert Spannungsunterschreitung Alarm1	FP32	V	150	230	195	-	RW
0x0E47	3655	2	Hysterese Spannungsunterschreitung Alarm1	FP32	%	0	10	10	-	RW
0x0E4D	3661	1	Spannungsunterschreitung Alarm2 ein/aus	U16		0	1	0	0 = aus 1 = ein	RW
0x0E4E	3662	2	Grenzwert Spannungsunterschreitung Alarm2	FP32	V	150	230	185	-	RW
0x0E50	3664	2	Hysterese Spannungsunterschreitung Alarm2	FP32	%	0	10	10	-	RW
0x0E56	3670	1	Rücksetzen der Energiezähler	U16	-	0x0815	0x0815	!	0x0815 void	CMD
0x0E57	3671	1	Letzte Trip Log OID	U16	-	-	-	-	-	RO
0x0E58	3672	2	Letzte Message Log OID	U32	-	-	-	-	-	RO
0x0E5A	3674	1	Energieflussrichtung	U16	-	0	1	0	0 = unten -> oben, 1 = oben -> unten Beeinflusst Leistung, Energie und Leistungsfaktor	RW
0x0E5B	3675	1	Rücksetzen der Extremwerte	U16	-	0x0815	0x0815	!	0x0815 void	CMD

4.3.2.3 KOMMUNIKATION

Register (Hex)	Register (dezimal)	Register Länge	Bezeichnung	Format	Einheit	Min	Max	Default	Bemerkung	Zugriff
0x041A	1050	1	Funk Sendeleistung	S16	dBm	-18	2	0	-	RW

5 VISUALISIERUNG

5.1 C# ASP.NET MVC

Ein C# ASP.NET-Webserver mit der **Model-View-Controller -Schematik** (kurz: **MVC**) ist ein modernes, leistungsfähiges Framework von Microsoft, mit dem sich dynamische, datenbankgestützte Webanwendungen entwickeln lassen.

ASP.NET MVC trennt den Aufbau einer Webanwendung in drei klar definierte Schichten:
Model, View und Controller. Diese Trennung, auch bekannt als Separation of Concerns, sorgt dafür, dass Logik, Darstellung und Daten unabhängig voneinander entwickelt und gewartet werden können.

5.1.1 MODEL (DATEN- UND GESCHÄFTSLOGIK)

Das Model repräsentiert die Daten und die Regeln, nach denen diese verarbeitet werden. Es enthält Klassen, die zum Beispiel Entitäten (z. B. User, Product, Order) darstellen, sowie die Geschäftslogik, Validierungen und Datenbankzugriffe. In einer typischen ASP.NET-Anwendung wird das Model häufig mit Entity Framework (EF) verbunden. EF ist ein sogenannter ORM (Object-Relational Mapper), der Objekte automatisch mit Datenbanktabellen verknüpft. Dadurch kann man in C# mit Objekten arbeiten, während EF im Hintergrund SQL-Befehle generiert.

C#
```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

Diese Klasse kann direkt mit einer Datenbanktabelle Products verbunden werden. Änderungen am Objekt werden über EF automatisch in der Datenbank gespeichert.

5.1.2 CONTROLLER (STEUERLOGIK UND ANWENDUNGSFLUSS)

Der Controller ist die zentrale Steuereinheit einer ASP.NET MVC-Anwendung. Er empfängt Anfragen (Requests) vom Webbrowser, entscheidet, was passieren soll, ruft ggf. Daten aus dem Model ab, verarbeitet sie und gibt sie an eine View weiter. Jeder Controller ist eine C#-Klasse, die von der Basisklasse Controller erbt, und besteht aus sogenannten Actions – Methoden, die eine bestimmte Funktion oder Seite repräsentieren.

C#
```
public class ProductController : Controller
{
    public IActionResult Index()
    {
        var products = _context.Products.ToList();
        return View(products);
    }

    public IActionResult Details(int id)
    {
        var product = _context.Products.Find(id);
        return View(product);
    }
}
```

Der Controller ist also die Brücke zwischen Daten (Model) und Darstellung (View).

5.1.3 VIEW (BENUTZEROBERFLÄCHE / DARSTELLUNG)

Die View ist für die Darstellung der Inhalte im Browser zuständig. Sie enthält HTML, CSS und oft auch Razor-Syntax eine Mischung aus HTML und C#, die serverseitig verarbeitet wird, bevor die Seite an den Browser gesendet wird.

C#

```
@model IEnumerable<Product>

<h1>Produkte</h1>
<ul>
@foreach (var p in Model)
{
    <li>@p.Name – @p.Price €</li>
}
</ul>
```

Die View erhält vom Controller eine Liste von Produkten (Model) und zeigt sie dynamisch an.
Das bedeutet: Änderungen in der Datenbank erscheinen automatisch in der Weboberfläche, ohne dass man die HTML-Datei selbst anpassen muss.

5.2 GREENTECLAB VISUALISIERUNG (C#)

5.2.1 FRONTEND

HTML

```
<!doctype html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>GreenTecLab Device Manager </title>
    <link rel="stylesheet" href="~/css/dashboard/layout.css" />
    @RenderSection("Stylesheet", required: false)
</head>
<body>

    <div class="dashboard-wrapper">
        <header>
            <div class="brand">
                <svg width="24" height="24" viewBox="0 0 24 24" fill="none" stroke="currentColor"
stroke-width="2" stroke-linecap="round" stroke-linejoin="round" style="color: var(--primary)">
                    <rect x="2" y="2" width="20" height="8" rx="2" ry="2"></rect>
                    <rect x="2" y="14" width="20" height="8" rx="2" ry="2"></rect>
                    <line x1="6" y1="6" x2="6.01" y2="6"></line>
                    <line x1="6" y1="18" x2="6.01" y2="18"></line>
                </svg>
                <span>GreenTecLab Device Manager</span>
            </div>
            <div id="timestamp" style="color: var(--text-muted); font-size: 12px;"></div>
        </header>

        @RenderBody()

    </div>

    @RenderSection("Scripts", required: false)
</body>
</html>
```

Der gezeigte Code ist ein HTML-Grundgerüst für ein Dashboard, das speziell für ASP.NET Core bzw. Razor Pages entwickelt wurde. Es beginnt mit der Deklaration des Dokuments als HTML5 und legt die Sprache auf Deutsch fest, was für Barrierefreiheit und Suchmaschinen relevant ist. Im <head>-Bereich wird die Zeichencodierung auf UTF-8 gesetzt, sodass Umlaute und Sonderzeichen korrekt dargestellt werden, und ein viewport-Meta-Tag sorgt dafür, dass die Seite auf mobilen Geräten korrekt skaliert wird. Außerdem wird der Seitentitel definiert und eine zentrale CSS-Datei eingebunden, die das Layout des Dashboards steuert. Durch die Razor-Direktive @RenderSection("Stylesheet", required: false) können einzelne Seiten zusätzliche Stylesheets hinzufügen, ohne dass sie für alle Seiten verpflichtend sind.

Im <body>-Bereich wird die gesamte Seite von einem Container dashboard-wrapper umschlossen, der das Layout zusammenhält. Direkt am Anfang befindet sich ein Header, der das Branding enthält: ein SVG-Icon, das aus zwei Rechtecken und Linien besteht, kombiniert mit dem Text „GreenTecLab Device Manager“. Das Icon und der Text nutzen CSS-Variablen, um Farben konsistent aus dem Designsystem zu übernehmen. Zusätzlich gibt es ein Element für einen Zeitstempel, das aktuell leer ist, aber später vermutlich per JavaScript gefüllt wird. Der zentrale Inhaltsbereich der Seite wird über die Razor-Direktive @RenderBody() eingebunden, sodass jede einzelne View ihren eigenen Inhalt in diesem Layout darstellen kann. Abschließend ermöglicht @RenderSection("Scripts", required: false) das Einfügen zusätzlicher JavaScript-Dateien nur auf den Seiten, die sie benötigen, wodurch das Layout flexibel und wiederverwendbar bleibt.

Insgesamt stellt dieser Code eine saubere, modulare Basis für ein Dashboard dar, bei dem Header, Styles und Scripts zentral verwaltet werden, während der dynamische Inhalt der jeweiligen Seiten über Razor-Platzhalter eingebunden wird. Das Layout ist responsiv, visuell ansprechend durch SVG-Icons und CSS-Variablen, und auf Erweiterbarkeit ausgelegt, da einzelne Views problemlos zusätzliche Styles oder Scripts hinzufügen können.

CSS

```
:root {
    --primary: #16a34a;
    --primary-light: #f0fdf4;
    --border-color: #e2e8f0;
    --text-dark: #0f172a;
    --text-muted: #64748b;
    --bg-main: #f8fafc;
    --white: #ffffff;
}

:root {
    --m-primary: #16a34a;
    --m-primary-dark: #15803d;
    --m-bg-main: #f1f5f9;
    --m-surface: #ffffff;
    --m-border: #cbd5e1;
    --m-text: #1e293b;
    --m-text-light: #64748b;
    --m-code-bg: #1e293b;
}

* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
    font-family: "Inter", -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif;
}

body {
    background-color: var(--bg-main);
    color: var(--text-dark);
    font-size: 14px;
}

/* Layout */
.dashboard-wrapper {
    max-width: 1400px;
    margin: 0 auto;
    padding: 20px;
}

header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 24px;
    padding-bottom: 16px;
    border-bottom: 1px solid var(--border-color);
}

.brand {
    display: flex;
    align-items: center;
    gap: 10px;
    font-weight: 700;
```

```
    letter-spacing: -0.5px;
    text-transform: uppercase;
}
```

C#

```
@using SmarterLeitungsschutzschalter.ModBus
@model dashboardModel
@{
    Layout = "_DashboardLayout";
}

@section Stylesheet {
    <link rel="stylesheet" href="~/css/dashboard/index.css" />
}

@section Scripts {
    <script src="~/js/dashboard/index.js"></script>
}
```

Dieser Block bindet den Modbus-Namespace ein, damit Geräte und deren Daten verwendet werden können. Mit @model dashboardModel wird das Datenmodell für die View festgelegt, das alle Geräteinformationen enthält. Layout = "_DashboardLayout" sorgt dafür, dass die View in das zentrale Dashboard-Layout eingebettet wird, inklusive Header, Styles und Scripts.

Es wird ein spezifisches CSS für diese Seite eingebunden. Über Razor Sections können Styles nur für diese View geladen werden, ohne das Hauptlayout zu verändern.

Sowie ein Abschnitt der JavaScript-Dateien lädt, die für interaktive Funktionen wie Suchfilter, Modals oder dynamische Updates nötig sind. Auch hier werden die Scripts nur auf dieser Seite ausgeführt.

C#

```
<div class="toolbar">
    <input type="text" id="searchInput" class="search-input" placeholder="Nach Gerätename suchen...">
    <div style="flex-grow: 1;"></div>
    <div style="font-size: 12px; color: var(--text-muted); align-self: center;">
        Total: <span id="count" style="color: var(--text-dark); font-weight: bold;">0</span> Units
    </div>
</div>

<div class="table-container">
    <table>
        <thead> ... </thead>
        <tbody>
            @foreach(ModbusTcpDevice device in Model.ModbusDevices)
            {
                // Zeile für jedes Gerät
            }
        </tbody>
    </table>
</div>
```

Die Toolbar enthält ein Suchfeld für Gerätesuche und zeigt rechts die Gesamtanzahl der Geräte an. Flexbox sorgt dafür, dass das Suchfeld links bleibt und die Zählung rechts ausgerichtet ist.

In dieser Tabelle werden alle Geräte aus dem Model aufgelistet. Jede Zeile zeigt die UID, den Gerätenamen, den Arbeitsplatz (als Text), den Status (online/offline), den letzten Sync-Zeitpunkt und einen Button zum Öffnen des Modals mit Details. Statuspunkte werden farblich hervorgehoben, die UID ist monospace für bessere Lesbarkeit, und die Buttons enthalten kleine SVG-Icons.

C#

```
<div id="@uuid" class="modal">
```

```
    <div class="modal-content">
        <div class="modal-header"> ... </div>
        <div class="tab-bar"> ... </div>
        <div class="modal-body" id="modalDataContent">
            @foreach(var dp in device.DeviceData)
            {
                <div><span class="tech-key">@Html.Raw(dp.DataName)</span>
                     <span class="tech-val">@Html.Raw(dp.Value + " " + dp.Unit)</span>
                </div>
            }
        </div>
    </div>
</div>
```

Für jedes Gerät wird ein Modal-Fenster erzeugt, das beim Klick auf „VIEW DATA“ geöffnet wird. Es zeigt den Gerätenamen, die UID und alle technischen Datenpunkte in einer übersichtlichen Liste. Das Modal enthält einen Close-Button mit SVG-Icon. Tabs (aktuell nur „GENERAL“) erlauben später die Erweiterung um weitere Kategorien.

JavaScript

```
function openDetails(uuid) {
    // alle Modals holen
    const modals = document.querySelectorAll('.modal');

    // zuerst alle schließen
    modals.forEach(modal => {
        modal.style.display = 'none';
    });

    // gewünschtes Modal öffnen
    const targetModal = document.getElementById(uuid);
    if (targetModal) {
        targetModal.style.display = 'flex';
    } else {
        console.warn('Modal nicht gefunden:', uuid);
    }
}

function closeModal() {
    const modals = document.querySelectorAll('.modal');
    modals.forEach(modal => {
        modal.style.display = 'none';
    });
}

document.addEventListener('DOMContentLoaded', () => {
    const searchInput = document.getElementById('searchInput');
    const rows = document.querySelectorAll('.table-container tbody tr');
    const countElement = document.getElementById('count');

    function filterTable() {
        const query = searchInput.value.toLowerCase().trim();
        let visibleCount = 0;

        rows.forEach(row => {
            // NUR Gerätebezeichnung (2. Spalte)
            const deviceName = row.cells[1].textContent.toLowerCase();

            if (deviceName.includes(query)) {
                row.style.display = '';
                visibleCount++;
            } else {
                row.style.display = 'none';
```

```
            }
        });

        countElement.textContent = visibleCount;
    }

    // Initiale Anzeige
    countElement.textContent = rows.length;

    // Live-Suche
    searchInput.addEventListener('input', filterTable);
});
```

Es gibt die Funktion openDetails(uuid). Sie wird aufgerufen, wenn ein Nutzer den „VIEW DATA“-Button einer Gerätetabelle anklickt. Zuerst werden alle Modals auf der Seite abgefragt (document.querySelectorAll('.modal')) und geschlossen, um sicherzustellen, dass nur ein Modal gleichzeitig sichtbar ist. Anschließend wird das Modal mit der übergebenen uuid gesucht. Wenn das passende Modal existiert, wird es angezeigt, indem sein display-Stil auf flex gesetzt wird. Ist das Modal nicht vorhanden, wird eine Warnung in der Konsole ausgegeben.

Die Funktion closeModal() dient dazu, alle geöffneten Modals wieder zu schließen. Sie greift ebenfalls auf alle .modal-Elemente zu und setzt deren display-Stil auf none. Das sorgt dafür, dass der Nutzer jederzeit die Detailansicht schließen kann, ohne dass ein spezielles Modal gezielt adressiert werden muss.

Der Hauptteil des Codes läuft innerhalb eines DOMContentLoaded-Events, also nachdem die Seite vollständig geladen ist. Hier werden das Suchfeld (searchInput), alle Tabellenzeilen (rows) und das Element für die Anzeige der Gesamtanzahl (countElement) abgefragt.

Die Funktion filterTable() sorgt für die Live-Suche in der Tabelle. Sie liest den aktuellen Suchbegriff ein, wandelt ihn in Kleinbuchstaben um und entfernt überflüssige Leerzeichen. Dann durchläuft sie jede Tabellenzeile und prüft, ob der Gerätename in der zweiten Spalte den Suchbegriff enthält. Passt er, bleibt die Zeile sichtbar; passt er nicht, wird sie ausgeblendet (display: 'none'). Gleichzeitig wird gezählt, wie viele Geräte aktuell angezeigt werden, und die Gesamtanzeige (countElement) entsprechend aktualisiert.

Zum Schluss wird die Tabelle initial gezählt (countElement.textContent = rows.length) und ein Event-Listener auf das Suchfeld gesetzt, sodass die Filterung live bei jeder Eingabe durchgeführt wird.

5.2.2 BACKEND

5.2.2.1 MODBUS TCP AUSLESEN

Der gesamte Code ist asynchron, damit UI-Threads oder Steuerlogik nicht blockiert werden.

C#

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Sockets;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Tasks;

namespace SmarterLeitungsschutzschalter.ModBus
{
    public class ModbusTcpDevice : IDisposable
    {
```

```
    }
}
```

Dieser Abschnitt bindet alle .NET-Bibliotheken ein, die für die Implementierung notwendig sind. System.Net.Sockets stellt die TCP-Kommunikation bereit, auf der Modbus TCP vollständig basiert. System.Text.Json wird genutzt, um Gerätekonfigurationen und Registerdefinitionen als JSON zu laden oder zu speichern. System.Threading.Tasks ist essenziell, da alle Netzwerkoperationen asynchron ausgeführt werden, um Blockierungen zu vermeiden. Die übrigen Namespaces liefern Basistypen, Collections und Fehlerklassen, die im weiteren Verlauf benötigt werden.

Der Namespace ordnet den Code klar einem Modbus-Modul innerhalb eines größeren Projekts zu. Dadurch wird die Modbus-Kommunikation logisch von anderen Bereichen wie UI, Logik oder Datenbank getrennt. Das ist besonders wichtig in technischen Projekten, da Protokollcode oft unabhängig wiederverwendet oder ausgetauscht wird.

Diese Klasse stellt ein einzelnes Modbus-TCP-Gerät dar. Sie übernimmt alle Aufgaben von der Verbindungsverwaltung bis zum Lesen der Register. Die Implementierung von IDisposable zeigt, dass die Klasse Ressourcen besitzt, die explizit freigegeben werden müssen – konkret eine TCP-Verbindung und einen Netzwerkstream.

C#

```
[JsonIgnore]
public string DeviceName { get; set; }

public string Ip { get; set; }
public int Port { get; set; }
public byte UnitId { get; set; }
public int Workplace { get; set; }
```

Hier werden die grundlegenden Eigenschaften des Modbus-Geräts definiert. Die IP-Adresse und der Port bestimmen das Netzwerkziel, wobei Port 502 der Standard für Modbus TCP ist. Die UnitId ist die Modbus-Slave-Adresse, die vor allem bei Gateways relevant ist. DeviceName und Workplace dienen rein der logischen Organisation innerhalb der Anwendung und werden nicht serialisiert, da sie nichts mit dem Protokoll selbst zu tun haben.

C#

```
[JsonIgnore]
private TcpClient? client;

[JsonIgnore]
private NetworkStream? stream;
```

Diese beiden Felder repräsentieren die aktive TCP-Verbindung. Sie werden bewusst nicht serialisiert, da eine offene Verbindung kein persistenter Zustand ist. Der TcpClient kapselt den Socket, während der NetworkStream den eigentlichen Datenstrom für Lese- und Schreiboperationen bereitstellt.

C#

```
public List<ModbusDataPoint> DataPoints { get; set; } = new();
```

Diese Liste beschreibt, welche Register aus dem Gerät gelesen werden sollen. Jeder Eintrag enthält Startadresse, Länge, Datenformat und Einheit. Technisch handelt es sich um ein Mapping zwischen Modbus-Registeradressen und semantischen Messwerten der Anwendung.

C#

```
[JsonIgnore]
public string DataPointsJson
{
    get
    {
        return JsonSerializer.Serialize(
```

```
            DataPoints,
            new JsonSerializerOptions { WriteIndented = true }
        );
    }
}
```

Diese Eigenschaft erzeugt zur Laufzeit eine JSON-Darstellung der aktuellen Presets. Sie ist hilfreich für Debugging, Logging oder das Weiterreichen der Konfiguration an andere Komponenten, ohne die komplette Geräteklasse zu serialisieren.

C#

```
[JsonIgnore]
public List<ModbusDeviceReceivingData> DeviceData = new();
```

Hier werden die tatsächlich gelesenen und bereits interpretierten Werte gespeichert. Diese Daten sind flüchtig und ändern sich ständig, weshalb sie ebenfalls nicht serialisiert werden.

C#

```
[JsonIgnore]
public bool IsConnected => client?.Connected ?? false;
```

Diese Eigenschaft prüft, ob aktuell eine TCP-Verbindung besteht. Sie greift direkt auf den Status des TcpClient zu und liefert einen sicheren booleschen Wert, auch wenn noch keine Verbindung existiert.

C#

```
public ModbusTcpDevice(string ip = "0.0.0.0", int port = 502, byte unitId = 1)
{
    this.Ip = ip;
    this.Port = port;
    this.UnitId = unitId;
}
```

Der Konstruktor setzt lediglich Standardwerte und baut keine Verbindung auf. Das ist technisch korrekt, da Netzwerkoperationen fehlschlagen können und explizit gestartet werden sollten.

C#

```
public void LoadPresetFromJson(string json)
{
    List<ModbusDataPoint> dataPoints =
        JsonSerializer.Deserialize<List<ModbusDataPoint>>(json);

    if (dataPoints == null)
        throw new InvalidDataException("Preset JSON could not be loaded.");

    DataPoints = dataPoints;
}
```

Dieser Code lädt eine komplette Registerdefinition aus JSON. Schlägt die Deserialisierung fehl, wird sofort ein Fehler ausgelöst, sodass keine inkonsistente Konfiguration verwendet werden kann.

C#

```
public async Task<object> ReadPresetValueAsync(string dataPointName)
{
    if (DataPoints.Count == 0)
        throw new InvalidOperationException("No preset loaded.");
    if (stream == null)
        throw new InvalidOperationException("Not connected. Call ConnectAsync() first.");

    ModbusDataPoint dp = DataPoints.Find(p => p.Title.Equals(dataPointName,
StringComparison.OrdinalIgnoreCase));
    if (dp == null)
        throw new KeyNotFoundException($"No data point named '{dataPointName}' found.");

    ushort[] registers = await ReadHoldingRegistersAsync(dp.StartRegister, dp.Length);
```

```
    return ModbusDataConverter.Convert(registers, dp.Format);
}
```

Diese Methode sucht einen Datenpunkt anhand seines Namens, liest die zugehörigen Register und übergibt die Rohdaten an einen Konverter. Die Kommunikationslogik bleibt dabei strikt von der Dateninterpretation getrennt.

C#

```
public async Task ConnectAsync()
{
    if (IsConnected)
        return;

    try
    {
        client = new TcpClient();
        await client.ConnectAsync(Ip, Port);
        stream = client.GetStream();
    }
    catch (Exception ex)
    {
        // Wrap the low-level exception with contextual information.
        throw new Exception($"Error connecting to {Ip}:{Port}: {ex.Message}", ex);
    }
}
```

Hier wird der TCP-Socket geöffnet und der Netzwerkstream erzeugt. Die Verbindung bleibt bestehen und wird für alle folgenden Modbus-Anfragen genutzt, was die Performance deutlich verbessert.

C#

```
public void Disconnect()
{
    stream?.Close();
    client?.Close();
    stream = null;
    client = null;
}
```

Dieser Abschnitt sorgt für eine saubere Freigabe aller Netzwerkressourcen. Er ist wichtig für Reconnects, Programmende oder Fehlerfälle.

C#

```
public async Task<ushort[]> ReadHoldingRegistersAsync(ushort startAddress, ushort numRegisters)
{
    if (stream == null)
        throw new InvalidOperationException("Not connected. Call ConnectAsync() first.");

    byte[] request = CreateReadHoldingRegistersRequest(UnitId, startAddress, numRegisters);
    await stream.WriteAsync(request, 0, request.Length);

    // Read MBAP + PDU header (9 bytes) first
    byte[] header = new byte[9];
    await ReadExactAsync(stream, header, 0, header.Length);

    // The 9th byte (index 8) is the byte count for the data part
    byte byteCount = header[8];
    int expectedLength = 9 + byteCount;

    // Read the remainder of the response and merge with header
    byte[] response = new byte[expectedLength];
    Array.Copy(header, response, header.Length);
    await ReadExactAsync(stream, response, header.Length, expectedLength - header.Length);
```

```
    // Convert response bytes into ushort registers
    ushort[] registers = new ushort[numRegisters];
    for (int i = 0; i < numRegisters; i++)
    {
        int index = 9 + i * 2;
        if (index + 1 >= response.Length)
            break;
        registers[i] = (ushort)((response[index] << 8) | response[index + 1]);
    }

    return registers;
}
```

Hier wird ein vollständiger Modbus-TCP-Frame erzeugt und über den Stream gesendet. Anschließend wird die Antwort in mehreren Schritten zuverlässig eingelesen.

C#

```
private static async Task ReadExactAsync(NetworkStream stream, byte[] buffer, int offset, int count)
{
    int read = 0;
    while (read < count)
    {
        int bytes = await stream.ReadAsync(buffer, offset + read, count - read);
        if (bytes == 0)
            throw new Exception("Connection unexpectedly closed.");
        read += bytes;
    }
}
```

Diese Methode stellt sicher, dass exakt die erwartete Anzahl an Bytes gelesen wird. Sie ist notwendig, da TCP die Daten fragmentiert liefern kann und ReadAsync keine Garantie für vollständige Frames gibt.

C#

```
private static byte[] CreateReadHoldingRegistersRequest(byte unitId, ushort startAddress, ushort
numRegisters)
{
    byte functionCode = 0x03;
    ushort transactionId = 1;

    byte[] frame = new byte[12];
    // MBAP header
    frame[0] = (byte)(transactionId >> 8);
    frame[1] = (byte)(transactionId & 0xFF);
    frame[2] = 0x00;
    frame[3] = 0x00;
    frame[4] = 0x00;
    frame[5] = 0x06;
    // PDU
    frame[6] = unitId;
    frame[7] = functionCode;
    frame[8] = (byte)(startAddress >> 8);
    frame[9] = (byte)(startAddress & 0xFF);
    frame[10] = (byte)(numRegisters >> 8);
    frame[11] = (byte)(numRegisters & 0xFF);
    return frame;
}
```

Diese Methode erzeugt einen vollständigen Modbus-TCP-Request zum Lesen von Holding-Registern gemäß der offiziellen Modbus-TCP-Spezifikation. Das Ergebnis ist ein exakt 12 Byte langer Frame, der aus zwei logisch getrennten Bereichen besteht: dem MBAP-Header (Modbus Application Protocol Header) und der PDU (Protocol Data Unit). Modbus TCP kapselt das eigentliche Modbus-Protokoll vollständig in TCP, weshalb keine CRC benötigt wird – die Integrität übernimmt TCP selbst.

Der Parameter unitId legt fest, welches Modbus-Gerät adressiert wird. In reinem Modbus TCP ist dieser Wert oft 1 oder 0xFF, er wird aber zwingend benötigt, wenn ein TCP-Gateway mehrere serielle Modbus-RTU-Slaves bedient. Die Parameter startAddress und numRegisters definieren den Registerbereich, der aus dem Holding-Register-Adressraum gelesen werden soll.

Der functionCode wird auf 0x03 gesetzt. Dieser Wert ist nicht beliebig, sondern fest im Modbus-Standard definiert. Funktionscode 3 bedeutet „Read Holding Registers“. Andere Codes wie 0x01 oder 0x04 würden andere Speicherbereiche adressieren, beispielsweise Coils oder Input Registers. Dass hier explizit 0x03 verwendet wird, zeigt, dass ausschließlich Holding-Register gelesen werden sollen, also der typische 4xxxx-Adressraum.

Die transactionId ist ein 16-Bit-Wert, der zur Zuordnung von Anfrage und Antwort dient. In Modbus TCP kann ein Client mehrere Anfragen parallel senden. Der Server kopiert diese Transaction ID unverändert in die Antwort zurück, sodass der Client weiß, zu welcher Anfrage die Antwort gehört. In diesem Code ist sie fest auf 1 gesetzt, was technisch korrekt ist, solange keine parallelen Requests gesendet werden. Würde man mehrere gleichzeitige Anfragen unterstützen, müsste diese ID inkrementiert oder verwaltet werden.

Das Byte-Array frame wird mit einer Länge von 12 Bytes angelegt, da ein Modbus-TCP-Read-Request immer genau diese Länge hat: 7 Bytes MBAP-Header plus 5 Bytes PDU.

Die ersten beiden Bytes frame[0] und frame[1] enthalten die Transaction ID im Big-Endian-Format. Das bedeutet, dass das höherwertige Byte zuerst übertragen wird. Durch das Rechts-Shift um 8 Bits wird das High-Byte extrahiert, während das Maskieren mit 0xFF das Low-Byte liefert. Diese Byte-Reihenfolge ist durch die Modbus-Spezifikation zwingend vorgegeben.

Die Bytes frame[2] und frame[3] bilden die sogenannte Protocol ID. Dieser Wert ist bei Modbus TCP immer 0x0000. Er existiert ausschließlich, um theoretisch andere Protokolle über denselben Mechanismus zu kapseln. In der Praxis wird er immer auf Null gesetzt, und jedes andere Gerät erwartet exakt diesen Wert.

Die Bytes frame[4] und frame[5] definieren das Length-Feld des MBAP-Headers. Dieses Feld gibt an, wie viele Bytes nach diesem Feld noch folgen. In diesem Fall sind das exakt 6 Bytes: 1 Byte Unit ID plus 5 Bytes PDU. Deshalb wird hier der Wert 0x0006 gesetzt. Auch dieses Feld ist Big-Endian codiert, weshalb zuerst 0x00 und danach 0x06 geschrieben wird. Dieses Längenfeld ist entscheidend, damit der Empfänger weiß, wie viele Bytes er aus dem TCP-Stream lesen muss.

Ab frame[6] beginnt die eigentliche Modbus-PDU. Das erste Byte der PDU ist die unitId. Sie identifiziert das Zielgerät hinter dem TCP-Endpunkt. Auch bei reinem Modbus TCP ohne Gateway darf dieses Feld nicht fehlen, da es Teil des Protokolls ist.

Das nächste Byte frame[7] enthält den zuvor definierten Funktionscode 0x03. Der Server wertet dieses Byte aus, um zu entscheiden, welche Aktion auszuführen ist. Falls ein Gerät diesen Funktionscode nicht unterstützt oder ein Fehler auftritt, wird in der Antwort ein Fehlercode zurückgegeben, bei dem das höchstwertige Bit des Funktionscodes gesetzt ist.

Die Bytes frame[8] und frame[9] enthalten die Startadresse der Register. Auch hier wird wieder Big-Endian verwendet. Das bedeutet, dass zuerst das High-Byte der Adresse gesendet wird, danach das Low-Byte. Die Adresse bezieht sich nicht auf die 4xxxx-Notation, sondern auf den nullbasierten Registerindex, wie er im Modbus-Protokoll definiert ist. Das ist eine häufige Fehlerquelle, weshalb die exakte Byte-Aufteilung hier entscheidend ist.

Die letzten beiden Bytes frame[10] und frame[11] definieren die Anzahl der zu lesenden Register. Auch dieser Wert ist ein 16-Bit-Unsigned-Integer im Big-Endian-Format. Der Server nutzt diesen Wert, um zu bestimmen, wie viele Register er aus seinem Speicherbereich lesen und in der Antwort

zurücksenden soll. Die maximale Anzahl ist durch den Modbus-Standard begrenzt und wird vom Server validiert.

Am Ende gibt die Methode das vollständig aufgebaute Byte-Array zurück. Dieses Array kann direkt über einen TCP-Stream gesendet werden, ohne weitere Verarbeitung. Jeder einzelne Bit-Shift, jede Byte-Position und jeder feste Wert ist dabei direkt durch die Modbus-TCP-Spezifikation vorgegeben und nicht austauschbar, ohne die Protokollkonformität zu verlieren.

C#
```
public void Dispose() => Disconnect();
```
Damit wird sichergestellt, dass die Klasse auch bei Verwendung in einem using-Block ihre Ressourcen korrekt freigibt.

C#
```
// DTO describing a data point in the device preset (title, register start, length, format, unit).
public class ModbusDataPoint
{
    public string Title { get; set; } = string.Empty;
    public ushort StartRegister { get; set; }
    public ushort Length { get; set; }
    public string Format { get; set; } = string.Empty;
    public string Unit { get; set; } = string.Empty;
}
```

C#
```
// DTO used to hold a received data point value and metadata after reading from a device.
public class ModbusDeviceReceivingData
{
    public string DeviceName {  get; set; } = string.Empty;
    public string DataName { get; set; } = string.Empty;
    public object Value { get; set; }
    public string Unit { get; set; } = string.Empty;
}
```
Diese Klassen enthalten keine Logik, sondern beschreiben ausschließlich Datenstrukturen. Sie trennen Konfiguration (was wird gelesen) von Ergebnis (was wurde gelesen) und sind damit essenziell für eine saubere Architektur.

5.2.2.2 JSON-File

JSON
```
{
  "Leitungsschutzschalter 1": {
    "Workplace": 1,
    "Ip": "192.168.0.111",
    "Port": 502,
    "UnitId": 1,
    "DataPoints": [
      {
        "Title": "Temperatur",
        "StartRegister": 3071,
        "Length": 2,
        "Format": "FP32",
        "Unit": "°C"
      },
      {
        "Title": "Mittelwert Temperatur",
        "StartRegister": 3073,
```

```
        "Length": 2,
        "Format": "FP32",
        "Unit": "°C"
      },
          ...
    ]
  },
  "Leitungsschutzschalter 2": {
    "Workplace": 1,
    "Ip": "192.168.0.111",
    "Port": 502,
    "UnitId": 2,
    "DataPoints": [...]
  },
  "Leitungsschutzschalter 3": {
    "Workplace": 1,
    "Ip": "192.168.0.111",
    "Port": 502,
    "UnitId": 3,
    "DataPoints": [...]
  },
  "POC1100 1": {
    "Workplace": 1,
    "Ip": "192.168.0.111",
    "Port": 502,
    "UnitId": 255,
    "DataPoints": [...]
  },
  "PAC2200 1": {
    "Workplace": 1,
    "Ip": "192.168.0.112",
    "Port": 502,
    "UnitId": 1,
    "DataPoints": [...]
  },
  "Leitungsschutzschalter 1": {
    "Workplace": 2,
    "Ip": "192.168.0.121",
    "Port": 502,
    "UnitId": 2,
    "DataPoints": [...]
  },
  "Leitungsschutzschalter 2": {
    "Workplace": 2,
    "Ip": "192.168.0.121",
    "Port": 502,
    "UnitId": 2,
    "DataPoints": [...]
  },
  "Leitungsschutzschalter 3": {
    "Workplace": 2,
    "Ip": "192.168.0.121",
    "Port": 502,
    "UnitId": 2,
    "DataPoints": [...]
  },
  "POC1100 2": {
    "Workplace": 2,
    "Ip": "192.168.0.121",
    "Port": 502,
    "UnitId": 255,
    "DataPoints": [...]
  },
```

```
  "PAC2200 2": {
    "Workplace": 2,
    "Ip": "192.168.0.122",
    "Port": 502,
    "UnitId": 1,
    "DataPoints": [...]
  },
  ...
}
```

Die JSON-Datei ist im Grunde eine hierarchische Struktur, in der jedes Gerät in einem Netzwerk als eigener Eintrag organisiert ist. Auf der obersten Ebene stehen die Gerätenamen wie „Leitungsschutzschalter 1“, „PAC2200 1“ oder „POC1100 1“ als Schlüssel, hinter denen jeweils ein Objekt liegt, das alle relevanten Informationen zu diesem Gerät enthält. Jedes dieser Geräteobjekte enthält zunächst allgemeine Angaben wie den Arbeitsplatz (Workplace), an dem es installiert ist, die IP-Adresse (Ip), über die es erreichbar ist, den Kommunikationsport (Port), meistens 502, was auf Modbus TCP hindeutet, sowie die Geräte-ID (UnitId), die typischerweise als Modbus-Slave-Adresse dient.

Ein besonders wichtiger Teil ist das Feld DataPoints, das ein Array von Messgrößen oder Datenpunkten enthält, die das Gerät liefert. Jeder Datenpunkt ist wiederum ein Objekt mit einem Titel (Title), der den Messwert beschreibt, einer Startadresse (StartRegister), die angibt, ab welchem Register der Wert gelesen wird, einer Länge (Length) in Registern, die den Speicherbereich definiert, einem Datenformat (Format), zum Beispiel FP32 für 32-Bit-Fließkommazahlen, und einer Einheit (Unit), etwa °C für Temperatur.

Diese Struktur wiederholt sich für alle Geräte in allen Arbeitsplätzen. In der vorliegenden JSON tauchen manche Gerätenamen mehrfach auf, etwa „Leitungsschutzschalter 1“ für verschiedene Arbeitsplätze, was technisch gesehen in JSON nicht zulässig ist, weil Schlüssel eindeutig sein müssen. Um Konflikte zu vermeiden, müssten die Namen eindeutig gemacht werden, zum Beispiel durch Hinzufügen des Arbeitsplatzes in den Namen. Insgesamt ist die JSON modular aufgebaut, wodurch sie sich gut erweitern lässt, um beliebig viele Geräte und Messwerte zu verwalten, und sie eignet sich besonders für die strukturierte Abfrage von Messdaten über ein Netzwerkprotokoll wie Modbus TCP.

6 MODBUS

Das Kommunikationsprotokoll Modbus wurde ursprünglich im Jahr 1979 von der Firma Modicon (heute Teil von Schneider Electric) entwickelt. Der Hintergrund war die zunehmende Verbreitung von speicherprogrammierbaren Steuerungen (SPS, engl. PLC) in industriellen Anlagen. Diese Steuerungen mussten Daten mit Sensoren, Aktoren, Messgeräten und übergeordneten Leitsystemen (SCADA-Systemen) austauschen, doch damals gab es kaum standardisierte Protokolle.

Modicon wollte eine Lösung schaffen, die einfach, zuverlässig und offen dokumentiert war, damit verschiedene Geräte verschiedener Hersteller miteinander kommunizieren konnten. So entstand Modbus, das zu einem der ältesten noch heute genutzten Industrieprotokolle wurde.

Die ursprüngliche Form – Modbus RTU (Remote Terminal Unit) – basierte auf serieller Kommunikation über RS-232 oder RS-485. Diese physikalischen Schnittstellen waren robust und störsicher, aber sie erlaubten nur punkt-zu-punkt oder busförmige Kommunikation mit relativ niedriger Übertragungsgeschwindigkeit (typisch 9,6 kbit/s bis 115,2 kbit/s).

Mit der fortschreitenden Vernetzung industrieller Anlagen, insbesondere durch den Einzug von Ethernet in den 1990er-Jahren, wurde die Idee geboren, Modbus auch über IP-basierte Netzwerke zu übertragen. Im Jahr 1999 wurde schließlich der offizielle Standard Modbus TCP/IP (kurz: Modbus-TCP) veröffentlicht. Er verwendet die gleiche logische Struktur wie Modbus RTU, jedoch mit einem völlig anderen physikalischen Übertragungsmedium (Ethernet) und Transportprotokoll (TCP/IP).

Eigenschaft	**Modbus RTU**	**Modbus-TCP**
Physikalische Schicht	*RS-485 / RS-232*	*Ethernet*
Max. Geschwindigkeit	*ca. 115 kbit/s*	*Bis 1 Gbit/s (abhängig vom Netzwerk)*
Teilnehmerzahl	*Max. 32 (ohne Repeater)*	*Theoretisch unbegrenzt*
Fehlererkennung	*CRC-16 im Telegramm*	*Durch TCP-Prüfsummen*
Adressierung	*Slave-Adresse (1–247)*	*IP-Adresse + Unit-ID*
Telegrammstruktur	*Start-/Stop-Bits + CRC*	*MBAP-Header + TCP-Segment*
Kommunikationsform	*Halbduplex (Master-Slave)*	*Vollduplex (Client-Server)*

6.1 MODBUS-RTU

Modbus RTU ist ein serielles Kommunikationsprotokoll, das für den Datenaustausch zwischen elektronischen Geräten in industriellen Netzwerken entwickelt wurde. Es basiert auf einem Master-Slave-Prinzip und arbeitet in der Regel auf physikalischer Ebene mit RS-485 oder RS-232. Der Begriff „RTU“ steht für Remote Terminal Unit, was so viel bedeutet wie „entfernte Steuereinheit“. Das beschreibt bereits den ursprünglichen Zweck: Die Kommunikation zwischen einer zentralen Steuerungseinheit (z. B. einer SPS Speicherprogrammierbaren Steuerung) und verteilten Sensoren, Aktoren oder Messmodulen über ein einziges, einfaches Bus-System.

Das Modbus-Protokoll ist binär codiert, was eine kompakte und effiziente Datenübertragung ermöglicht. Im Gegensatz zur ASCII-Variante, die lesbar, aber langsamer ist, ist Modbus RTU auf maximale Effizienz und Geschwindigkeit optimiert. Es eignet sich deshalb hervorragend für Systeme, in denen regelmäßig und zuverlässig kleine Datenmengen übertragen werden müssen – etwa Messwerte, Schaltzustände oder Steuerkommandos.

Das Modbus-Protokoll wurde 1979 von der Firma Modicon entwickelt, einem US-amerikanischen Hersteller von Speicherprogrammierbaren Steuerungen, der später von Schneider Electric übernommen wurde. Damals war die industrielle Kommunikation noch sehr herstellerspezifisch und proprietär. Jeder Automatisierungshersteller nutzte eigene Kommunikationsmethoden, was die Integration verschiedener Systeme erschwerte. Modbus war eines der ersten offenen Protokolle, das

öffentlich dokumentiert und frei verfügbar war. Dadurch konnte jeder Hersteller eigene Geräte entwickeln, die mit anderen Modbus-fähigen Geräten kommunizieren konnten.

Der ursprüngliche Zweck war einfach: Modicon wollte eine einfache Möglichkeit schaffen, Daten zwischen einer zentralen Steuerung (Master) und mehreren dezentralen Geräten (Slaves) auszutauschen. Die Anforderungen waren dabei klar:

- Das Protokoll sollte robust gegenüber elektrischen Störungen sein.
- Es sollte mit einfachen Mikrocontrollern realisierbar sein.
- Es sollte eine minimale Bandbreite benötigen.

Diese Philosophie spiegelt sich bis heute in Modbus RTU wider. Trotz neuerer Bus-Systeme wie PROFIBUS, CANopen oder EtherCAT bleibt Modbus RTU aufgrund seiner Einfachheit, Robustheit und weiten Verbreitung ein Industriestandard, der nach wie vor in neuen Geräten implementiert wird.

6.1.1 KOMMUNIKATION

Modbus RTU ist ein **Master-Slave-Kommunikationssystem**. Das bedeutet, dass nur ein Gerät (der Master) die Kommunikation kontrolliert und Datenanfragen initiiert. Die übrigen Geräte (Slaves) antworten ausschließlich, wenn sie vom Master angesprochen werden. Sie können weder selbstständig senden noch den Bus aktiv übernehmen. Dieses Prinzip hat den Vorteil, dass es keine Kollisionen oder Konflikte auf dem Bus gibt – es kommuniziert immer nur ein Gerät zur selben Zeit.

Ein typisches Modbus-Netzwerk besteht aus einem Master, beispielsweise einer SPS oder einem PC mit Modbus-Schnittstelle, und mehreren Slaves, etwa Temperaturreglern, Stromzählern oder I/O-Modulen. Jedes Slave-Gerät hat eine eindeutige Adresse zwischen 1 und 247, wobei 0 für Broadcasts (Nachrichten an alle Slaves) reserviert ist. Broadcast-Kommandos werden zwar von allen Slaves ausgeführt, aber nicht beantwortet, um Busüberlastung zu vermeiden.

Die Kommunikation verläuft streng sequenziell. Der Master sendet eine Anfrage, die das Zielgerät, den gewünschten Funktionscode (also die Art der Aktion, z. B. „Register lesen“), die Datenadresse und ggf. Parameter enthält. Der angesprochene Slave verarbeitet die Anfrage, führt die geforderte Aktion aus und sendet eine Antwort zurück. Diese enthält entweder die angeforderten Daten oder eine Bestätigung der erfolgreichen Ausführung. Falls ein Fehler auftritt (z. B. ungültige Adresse, falscher Funktionscode oder CRC-Fehler), antwortet der Slave mit einem sogenannten Exception Frame, in dem der Funktionscode um 0x80 erhöht ist und ein Fehlercode angegeben wird.

6.1.2 KOMMUNIKATIONSSTAPELS

Die physikalische Übertragungsschicht beschreibt, wie die elektrischen Signale auf der Leitung tatsächlich übertragen werden – also welche Spannungen, Pegel, Kabeltypen, Topologien und elektrischen Eigenschaften das Modbus-System nutzt.
Da Modbus RTU ausschließlich auf seriellen Schnittstellen basiert, erfolgt die Kommunikation über asynchrone serielle Datenübertragung, typischerweise mit RS-485, seltener mit RS-232 oder RS-422.

Der physikalische Layer ist entscheidend für die Zuverlässigkeit, Reichweite und Störfestigkeit der Modbus-Kommunikation. Obwohl das Protokoll selbst hardwareunabhängig ist, haben sich RS-485 und RS-232 als Standardtransportschichten etabliert, weil sie in der Industrie äußerst robust und bewährt sind.

6.1.2.1 RS-485

RS-485 (EIA-485) ist die am häufigsten verwendete physikalische Schnittstelle für Modbus RTU. Sie wurde speziell für industrielle Umgebungen entwickelt, in denen elektromagnetische Störungen, lange Leitungswege und mehrere Geräte auf einem Bus üblich sind.

RS-485 arbeitet differenziell, d. h. jedes Signal wird auf zwei Leitungen (A und B) übertragen, die entgegengesetzte Spannungen führen.
Ein logisches Signal ergibt sich aus der Spannungsdifferenz zwischen diesen beiden Leitungen:

- Logische „1“ (Mark): Leitung A < Leitung B (Differenzspannung kleiner als –200 mV)
- Logische „0“ (Space): Leitung A > Leitung B (Differenzspannung größer als +200 mV)

Der Empfänger wertet also nicht den absoluten Pegel einer Leitung, sondern die Differenz zwischen beiden aus.
Das hat den großen Vorteil, dass Gleichtaktstörungen – also Störungen, die beide Leitungen gleichzeitig beeinflussen – herausfallen, weil sich die Differenz kaum ändert.
Dies ist der Schlüssel zur hohen Störsicherheit von RS-485, besonders bei langen Leitungen oder in Umgebungen mit starker elektromagnetischer Beeinflussung (z. B. Motoren, Frequenzumrichter, Schütze).

Die RS-485-Norm definiert, dass die Differenzspannung (A–B) mindestens ±200 mV betragen muss, damit ein Zustand eindeutig erkannt werden kann. Der Treiber kann Differenzspannungen von bis zu ±5 V erzeugen. Die Eingänge müssen Gleichtaktspannungen zwischen –7 V und +12 V tolerieren können, damit keine Fehlinterpretationen durch Spannungsverschiebungen im Bus entstehen.

RS-485 wird typischerweise in Daisy-Chain-Topologie (Linientopologie) aufgebaut. Das bedeutet, alle Geräte sind entlang einer einzigen verdrillten Zweidrahtleitung miteinander verbunden. Diese Leitung darf nicht sternförmig verzweigt werden, weil Signalreflexionen entstehen würden, die das Signal verzerren und Telegramme unlesbar machen können.

An den beiden physikalischen Enden der Leitung werden sogenannte Abschlusswiderstände (Termination Resistors) angebracht – meist 120 Ω. Diese Widerstände entsprechen in etwa dem Wellenwiderstand der verdrillten A/B-Leitung und verhindern Reflexionen, die besonders bei höheren Baudraten gravierende Signalstörungen verursachen.

Oft wird zusätzlich ein Schirm (Abschirmung) verwendet, der an einer Seite, meist nur auf Master-Seite, mit Masse verbunden ist. Dies reduziert elektromagnetische Einstrahlung (EMI) und schützt vor Störfeldern.

Ein unbelegter RS-485-Bus (also wenn gerade niemand sendet) kann sich in einem undefinierten Zustand befinden, da alle Treiber inaktiv sind. Die Leitung schwebt („floating“) und elektrische Rauschimpulse können fälschlich als gültige Bits erkannt werden. Um das zu verhindern, wird eine Vorspannung (Biasing) verwendet. Ein Widerstandspaar (typisch 680 Ω–1 kΩ) zieht Leitung A leicht auf +5 V und Leitung B leicht auf 0 V. Damit ergibt sich im Ruhezustand eine definierte Differenzspannung (z. B. –200 mV), die als logische „1“ interpretiert wird. Diese Widerstände befinden sich meist im Mastergerät, manchmal auch zusätzlich in Repeatern oder Gateways.

Typische Kabelparameter:

- Wellenwiderstand: ca. 120 Ω
- Aderquerschnitt: 0,25–0,5 mm² (je nach Länge)

- Maximale Buslänge: 1200 m bei 9600 Baud
- Bei höheren Baudraten (z. B. 115200 Baud) verringert sich die mögliche Länge deutlich, oft unter 100 m.

Eine Faustregel lautet:

- Je höher die Baudrate, desto kürzer darf die Leitung sein.

Die Leitungskapazität und -induktivität beeinflussen das Signalverhalten stark. Lange Leitungen wirken wie Tiefpassfilter und können die Flanken der Rechtecksignale verformen. Deshalb werden in langen Netzen Repeater oder Signalverstärker eingesetzt.

RS-485 kann halbduplex (Half-Duplex) oder vollduplex (Full-Duplex) betrieben werden.
Modbus RTU verwendet fast ausschließlich den Half-Duplex-Modus, um die Verkabelung zu minimieren. Dabei werden dieselben A/B-Leitungen sowohl zum Senden als auch zum Empfangen verwendet. Das Protokoll stellt sicher, dass zu jedem Zeitpunkt nur ein Gerät (meist der Master oder ein gerade antwortender Slave) sendet – so entstehen keine Kollisionen.

In seltenen Fällen wird Full-Duplex (vieradrig) eingesetzt, wenn gleichzeitig gesendet und empfangen werden soll (z. B. bei Modbus-Gateways oder Diagnosegeräten).

6.1.2.2 RS-232

Während RS-485 für Mehrpunktverbindungen optimiert ist, ist RS-232 eine Punkt-zu-Punkt-Schnittstelle. Das bedeutet, dass hier nur ein Master und ein Slave direkt miteinander verbunden sein können. Sie eignet sich daher für kurze, störungsarme Verbindungen – etwa zwischen PC und SPS oder für Konfigurationstools.

RS-232 verwendet single-ended-Signale, d. h. die Signale werden gegen Masse übertragen. Der logische Zustand ergibt sich also direkt aus der Spannung zwischen Signalleitung (z. B. TX) und Bezugspotential (GND).

Die Signalpegel nach der EIA-232-Norm liegen zwischen +3 V bis +15 V für eine logische 0 (Space) und –3 V bis –15 V für eine logische 1 (Mark). Das bedeutet, dass RS-232 mit invertierter Logik arbeitet – im Gegensatz zu TTL- oder CMOS-Signalen, wo „1“ typischerweise +5 V bedeutet.

Da RS-232 keine differentielle Übertragung nutzt, ist es anfällig für Erdpotentialunterschiede und elektrische Störfelder. Die maximale Leitungslänge liegt bei ca. 15 m, bei niedrigen Baudraten bis 19,2 kBaud manchmal etwas mehr. Dafür ist RS-232 äußerst einfach aufzubauen, benötigt keine Abschlusswiderstände und ist in nahezu jedem Mikrocontroller oder PC-Interface integriert.

6.1.2.3 ZUSAMMENFASSUNG

Merkmal	**RS-485**	**RS-232**
Signalart	*Differenziell*	*Single-ended*
Leitungen	*2 (A/B) + GND*	*TX, RX, GND*
Kommunikation	*Halbduplex / Vollduplex*	*Punkt-zu-Punkt*
Typische Baudraten	*1200–115200 Baud*	*1200–115200 Baud*
Maximale Kabellänge	*ca. 1200 m @ 9600 Baud*	*ca. 15 m*
Anzahl der Geräte	*Bis zu 32 pro Bus*	*Nur 2*
Störfestigkeit	*Hoch*	*Niedrig*
Abschlusswiderstände	*Erforderlich (120 Ω)*	*Nicht erforderlich*
Bias-Widerstände	*Empfohlen*	*Nicht nötig*

6.1.3 DATENÜBERTRAGUNG UND TELEGRAMMSTRUKTUR

Die Datenübertragung bei Modbus RTU ist asynchron und basiert auf der UART-Kommunikation. Das bedeutet, dass die Daten byteweise übertragen werden, wobei jedes Byte aus 1 Startbit, 8 Datenbits, optional 1 Paritätsbit und 1 oder 2 Stopbits besteht. Üblich sind Konfigurationen wie 8N1 (8 Datenbits, keine Parität, 1 Stopbit) oder 8E1 (8 Datenbits, gerade Parität, 1 Stopbit).

Ein Modbus-RTU-Telegramm (auch „Frame“ genannt) besteht aus mehreren aufeinanderfolgenden Bytes, die zeitlich sehr genau definiert sind. Es gibt keine Start- oder Endemarker im Datenstrom – die Zeitlücken zwischen Telegrammen sind entscheidend. Wenn zwischen zwei Bytes eine Pause länger als 1,5 Zeichenzeiten auftritt, wird das aktuelle Telegramm als beendet betrachtet. Zwischen zwei vollständigen Telegrammen muss eine Pause von mindestens 3,5 Zeichenzeiten liegen. Diese zeitliche Trennung ist essenziell, um den Anfang und das Ende einer Nachricht zu erkennen.

Ein vollständiger RTU-Frame hat folgende Struktur:

| Slave-Adresse (1 Byte) | Funktionscode (1 Byte) | Daten (n Bytes) | CRC-Prüfsumme (2 Bytes) |

Die Länge des Datenfeldes hängt von der Art der Anfrage ab. Bei einer Leseanforderung enthält es beispielsweise Startadresse und Anzahl der zu lesenden Register; bei einer Schreibanforderung enthält es die Zieladresse und die zu schreibenden Werte.

6.1.3.1 REGISTERSYSTEM

Das Herzstück des Modbus-Protokolls bilden die sogenannten Funktionscodes, die definieren, welche Art von Operation durchgeführt wird. Sie reichen von einfachen Lese- und Schreibbefehlen bis zu komplexeren Diagnosen.
Die wichtigsten Funktionscodes sind:

- 01 (Read Coils): Liest digitale Ausgänge, also Schaltzustände (Bits).
- 02 (Read Discrete Inputs): Liest digitale Eingänge.
- 03 (Read Holding Registers): Liest 16-Bit-Holding-Register (typisch für Messwerte, Sollwerte oder Parameter).
- 04 (Read Input Registers): Liest 16-Bit-Input-Register (typisch für analoge Eingänge).
- 05 (Write Single Coil): Schreibt einen einzelnen digitalen Ausgang.
- 06 (Write Single Register): Schreibt ein einzelnes Holding-Register.
- 15 (Write Multiple Coils): Schreibt mehrere digitale Ausgänge gleichzeitig.
- 16 (Write Multiple Registers): Schreibt mehrere Holding-Register.

Das Datenmodell von Modbus ist streng in vier Speicherbereiche unterteilt:

- Coils (0xxxx): Schreib- und lesbare digitale Ausgänge (1 Bit).
- Discrete Inputs (1xxxx): Nur lesbare digitale Eingänge (1 Bit).
- Input Registers (3xxxx): Nur lesbare analoge Werte (16 Bit).
- Holding Registers (4xxxx): Schreib- und lesbare analoge Werte oder Parameter (16 Bit).

Intern arbeitet Modbus RTU grundsätzlich mit 16-Bit-Registern. Größere Datentypen wie 32-Bit-Integer oder 32-Bit-Floats werden auf zwei Register aufgeteilt. Da Modbus keine Standarddefinition für die Reihenfolge der Bytes oder Register hat, muss bei Mehrwort-Daten immer bekannt sein, ob das Gerät Big-Endian oder Little-Endian arbeitet (dies wird oft als „Word Order“ bezeichnet).

6.1.3.2 FEHLERERKENNUNG

Modbus RTU verwendet zur Fehlererkennung eine zyklische Redundanzprüfung (CRC-16).
Diese Prüfsumme deckt alle Bytes des Telegramms ab, außer die beiden CRC-Bytes selbst.

Der Algorithmus arbeitet folgendermaßen:

- Das CRC-Register wird mit 0xFFFF initialisiert.
- Jedes Byte der Nachricht wird mit dem CRC-Register durch XOR verknüpft.
- Für jedes der 8 Bits wird geprüft, ob das niederwertigste Bit gesetzt ist.
    - Wenn ja, wird das Register nach rechts geschoben und mit dem Polynom 0xA001 verknüpft.
    - Wenn nein, wird nur nach rechts geschoben.
- Am Ende ergibt sich eine 16-Bit-Prüfsumme, die in Little-Endian-Form (Low-Byte zuerst) an das Telegramm angehängt wird.

Wenn der Empfänger dieselbe Berechnung durchführt und das Ergebnis nicht null ist, gilt das Telegramm als fehlerhaft und wird verworfen.

6.1.3.3 BEISPIEL

Angenommen, der Master möchte von Slave 1 die Werte zweier Holding-Register (Adresse 40001 und 40002) lesen.
Der Master sendet folgendes Telegramm:

```
01 03 00 00 00 02 C4 0B
```

Das bedeutet:

- 01 → Slave-Adresse 1
- 03 → Funktionscode „Read Holding Registers“
- 00 00 → Startadresse 0 (entspricht 40001)
- 00 02 → Anzahl der Register (2 Stück)
- C4 0B → CRC-Prüfsumme (Low-Byte zuerst)

Der Slave antwortet:

```
01 03 04 00 0A 00 14 F9 6C
```

Bedeutung:

- 01 → Antwort von Slave 1
- 03 → Bestätigung Funktionscode 03
- 04 → Anzahl der folgenden Datenbytes (2 Register = 4 Byte)
- 00 0A → Wert des ersten Registers (10)
- 00 14 → Wert des zweiten Registers (20)
- F9 6C → CRC-Prüfsumme

6.2 MODBUS-TCP

6.2.1 BEDEUTUNG TCP

Das Transmission Control Protocol (TCP) ist eines der zentralen Protokolle der Internetprotokollfamilie (TCP/IP-Stack). Es arbeitet auf der Transportschicht (Layer 4) des OSI-Modells und stellt eine zuverlässige, verbindungsorientierte Kommunikation zwischen zwei Endpunkten bereit.

Im Gegensatz zu einfacheren, verbindungslosen Protokollen wie UDP (User Datagram Protocol) garantiert TCP, dass:

- alle gesendeten Datenpakete vollständig und in der richtigen Reihenfolge beim Empfänger ankommen,
- keine Duplikate oder Verluste auftreten,
- und eventuelle Übertragungsfehler automatisch erkannt und korrigiert werden.

Das geschieht durch Sequenznummern, Bestätigungen (ACKs), Timeout-Mechanismen und Wiederholungen.
Vor jeder Datenübertragung baut TCP eine logische Verbindung zwischen Client und Server auf, den sogenannten TCP-Handshake (ein dreistufiger Verbindungsaufbau: SYN → SYN/ACK → ACK). Erst danach beginnt die eigentliche Datenübertragung.

Modbus-TCP kann sich auf eine zuverlässige Übertragungsschicht verlassen. Fehlererkennung und -korrektur, die bei Modbus RTU noch durch eine CRC-Prüfsumme im Telegramm realisiert werden mussten, sind bei Modbus-TCP durch TCP bereits gewährleistet.
Dadurch wird die Kommunikation einfacher und effizienter, ohne dass sie an Sicherheit verliert.

6.2.2 KOMMUNIKATIONSSTAPELS

Modbus-TCP integriert sich direkt in den Ethernet- und IP-Kommunikationsstapel.
Man kann sich das in vier Hauptschichten vorstellen:

1. **Physikalische Schicht (Layer 1):**
   Hier findet die elektrische Signalübertragung über Twisted-Pair-Kabel (z. B. Cat 5e, Cat 6) oder Lichtwellenleiter statt. Diese Ebene ist identisch mit der von herkömmlichen Ethernet-Netzwerken (IEEE 802.3-Standard).

2. **Sicherungsschicht (Layer 2):**
   Ethernet-Frames werden gebildet, MAC-Adressen dienen der Adressierung der Geräte im lokalen Netzwerk. Die Daten werden in Ethernet-Frames gekapselt.

3. **Netzwerkschicht (Layer 3):**
   Diese Schicht ist durch das Internet Protocol (IP) realisiert. Jedes Gerät im Netzwerk besitzt eine eindeutige IP-Adresse, wodurch auch Kommunikation über Router und größere Netzwerke hinweg möglich wird.

4. **Transportschicht (Layer 4):**
   Hier arbeitet TCP, das einen logischen Datenkanal zwischen zwei Geräten aufbaut.
   TCP verwendet standardmäßig Port 502 für Modbus-Kommunikation.
   Jeder Modbus-TCP-Server (z. B. eine SPS) lauscht also auf Port 502 auf eingehende Verbindungen, während der Client (z. B. ein HMI oder PC-Programm) Anfragen an diesen Port sendet.

5. **Anwendungsschicht (Layer 7):**
   Hier befindet sich das eigentliche Modbus-Anwendungsprotokoll (Modbus Application Protocol, kurz MAP), das definiert, welche Daten übertragen werden und wie sie interpretiert werden müssen.

TCP/IP transportiert die Daten, Modbus definiert ihren Inhalt.

6.2.3 DATENÜBERTRAGUNG UND TELEGRAMMSTRUKTUR

Ein Modbus-TCP-Telegramm besteht im Wesentlichen aus zwei Teilen: dem MBAP-Header (Modbus Application Protocol Header) und der PDU (Protocol Data Unit).

6.2.3.1 MBAP-HEADER

Der MBAP-Header umfasst genau 7 Byte und ist eine Erweiterung gegenüber dem Modbus-RTU-Protokoll.
Er enthält Verwaltungsinformationen, die für die Zuordnung und Identifikation der Nachrichten innerhalb einer TCP-Verbindung notwendig sind.

Byte	Bezeichnung	Länge	Beschreibung
0-1	Transaction Identifier	2 Byte	Vom Client gesetzt, damit Antwort eindeutig zugeordnet werden kann
2-3	Protocol Identifier	2 Byte	Immer 0x0000 für Modbus; andere Werte reserviert
4-5	Length	2 Byte	Gibt die Länge der nachfolgenden Daten (inkl. Unit-Identifier) an
6	Unit Identifier	1 Byte	Entspricht der Slave-Adresse bei RTU; dient bei Gateways zur Zuordnung

Die Transaction-ID ist besonders wichtig, wenn mehrere Anfragen parallel über eine einzige TCP-Verbindung laufen (sogenanntes Pipelining).
Dadurch kann der Client auch dann die Antworten korrekt zuordnen, wenn sie in anderer Reihenfolge zurückkommen.

Der Unit Identifier ist vor allem relevant, wenn ein Modbus-TCP/RTU-Gateway verwendet wird.
Das Gateway empfängt TCP-Nachrichten und leitet sie seriell an die angeschlossenen RTU-Geräte weiter. In diesem Fall ist die Unit-ID die RTU-Slave-Adresse.
Wenn keine seriellen Geräte beteiligt sind (also reine TCP-Kommunikation), wird meist einfach der Wert 0xFF oder 0x01 verwendet.

6.2.3.2 DIE PDU

Die PDU ist der Teil, der das eigentliche Modbus-Protokoll abbildet.
Sie besteht aus einem Funktionscode (1 Byte) und den zugehörigen Daten.

Der Funktionscode bestimmt, welche Art von Zugriff oder Aktion ausgeführt werden soll.
Zum Beispiel:

- 0x01 → Lesen von digitalen Ausgängen (Coils)
- 0x02 → Lesen von digitalen Eingängen
- 0x03 → Lesen von Holding-Registern (z. B. Messwerte oder Sollwerte)
- 0x04 → Lesen von Input-Registern (z. B. analoge Eingänge)

- 0x05 → Schreiben eines einzelnen Coils
- 0x06 → Schreiben eines einzelnen Registers
- 0x10 → Schreiben mehrerer Register

Die Datenfelder in der PDU enthalten Adressen, Mengenangaben und ggf. Werte, die übertragen werden sollen.
Ein Holding-Register umfasst immer 16 Bit (2 Byte), und die Adressierung beginnt traditionell bei 40001, was aber nur eine logische Darstellung ist – im Telegramm selbst wird die Adresse 0-basiert übertragen.

6.2.3.3 BEISPIEL

Nehmen wir an, ein Client möchte die Holding-Register 40001 und 40002 eines Geräts mit Unit-ID 1 lesen.

Request (Hexadezimal):

```
00 01 00 00 00 06 01 03 00 00 00 02
```

Aufgeschlüsselt:

- 00 01 → Transaction-ID = 1
- 00 00 → Protocol-ID = 0 (Modbus)
- 00 06 → Länge der nachfolgenden Bytes (6 Byte)
- 01 → Unit-Identifier (Geräteadresse 1)
- 03 → Funktionscode „Read Holding Registers“
- 00 00 → Startadresse 0 (entspricht Register 40001)
- 00 02 → Anzahl Register 2

Response (Beispiel):

```
00 01 00 00 00 07 01 03 04 00 0A 00 14
```

Das Gerät antwortet mit:

- denselben Transaction- und Unit-IDs,
- dem Funktionscode 03,
- der Angabe, dass 4 Byte Daten folgen (04),
- und den zwei Registerwerten: 0x000A (= 10) und 0x0014 (= 20).

LITERATURVERZEICHNIS

[1] Siemens , „Siemens Industry Mall - 7KM2200-2EA40-1EA1," Siemens , 20 10 2025. [Online]. Available: https://mall.industry.siemens.com/mall/de/oeii/Catalog/Product/7KM2200-2EA40-1EA1. [Zugriff am 20 10 2025].

[2] Siemens, „Siemens Industry Mall - 7KN1111-0MC00," Siemens, 20 10 2025. [Online]. Available: https://mall.industry.siemens.com/mall/de/oeii/Catalog/Product/7KN1111-0MC00. [Zugriff am 20 10 2025].

[3] Siemens, „Siemens Industry Mall - 5SV6016-7MC16," Siemens, 20 10 2025. [Online]. Available: https://mall.industry.siemens.com/mall/de/oeii/Catalog/Product/5SV6016-7MC16. [Zugriff am 20 10 2025].

[4] Siemens , „Siemens SiePortal Support - Modbus Register für SENTRON COM System," 20 10 2025. [Online]. Available: https://support.industry.siemens.com/cs/document/109973540/modbus-register-für-sentron-com-system. [Zugriff am 20 10 2025].

ABBILDUNGSVERZEICHNIS

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.