TEKO
SCHWEIZERISCHE FACHSCHULE

Dokumentation Diplomarbeit

Automatisierte Prüfung von Frequenzumrichter Schaltschränken

Diplomarbeit Elektrotechniker 2023

Auftraggeber: Adrian John, Software Engineer Research and Development

Kunde: Haefely AG

Diplomlehrer: Adrian John

Prüfungsexperte: Patrick Grubert

Studierender: Mario Vasilakis

Summary

Die renommierte Firma Haefely AG spezialisiert sich auf die Produktion von Hochspannungsprüfgeräten, die essentiell für namhafte Hersteller von Leistungs-, Verteilungs- und Instrumentenwandlern, Kabelsystemen sowie hochmodernen Schaltanlagen sind. Ein integraler Bestandteil dieser Prüfgeräte sind die Frequenzumrichter, die speziell für Hochspannungsprüfquellen konzipiert sind. Diese komplexen Frequenzumrichter werden durch einen ausgewählten externen Fachbetrieb für Schaltanlagen in speziell entworfene Schaltschränke integriert. Jeder dieser Schaltschränke beherbergt, neben dem zentralen Umrichter, eine Vielzahl von weiteren essentiellen Komponenten. Dazu zählen Hauptschütze, präzise Temperatursensoren, optimierte Lüfter, genaue Strom- und Spannungsmesssysteme, fortschrittliche Netzwerk-Gateways und multifunktionale SPS I/O-Module.

Trotz der hohen Expertise des Schaltanlagenbauers ist es aktuell so, dass die Endkontrolle der Schaltschränke nur eine rudimentäre Prüfung erlaubt. Dies resultiert bedauerlicherweise in Situationen, in denen Schaltschränke bei der finalen Auslieferung an unser HAEFELY-Team nicht den höchsten Qualitätsstandards entsprechen.

Als proaktive Reaktion auf diese Herausforderung haben wir uns entschlossen, eine spezifizierte Prüfsequenz mithilfe einer modernen SPS-Steuerung zu implementieren. Für dieses ambitionierte Vorhaben wird die notwendige Hardware von HAEFELY bereitgestellt.

Mit der Einführung dieser hochmodernen Prüfeinrichtung ist der Schaltanlagenbauer in der Lage, nach Abschluss der Montagearbeiten eine umfassende und detaillierte Funktionsprüfung aller verbauten Komponenten durchzuführen. Dies ermöglicht eine zeitnahe Erkennung und Korrektur jeglicher Unregelmäßigkeiten.

Ergänzend hierzu haben wir eine benutzerfreundliche Software entwickelt, die durch eine intuitive Kontexthilfe nicht nur eine schrittweise Anleitung zur Funktionalität bietet, sondern auch konkrete Unterstützung bei der Fehlerdiagnose und -behebung leistet.

Inhaltsverzeichnis

1. Definitionen und Abkürzungen

FU/FC (Frequenzumrichter / Frequency Converter): Ein technisch ausgereiftes Gerät, das in der Lage ist, aus einer speisenden Wechselspannung eine modifizierte Wechselspannung zu generieren.

SPS (Speicherprogrammierbare Steuerung): Eine hochentwickelte Vorrichtung, die präzise zur Steuerung oder Regelung von technischen Anlagen und Maschinen eingesetzt wird.

B&R (B&R Industrial Automation GmbH - Bernecker + Rainer): Ein renommierter Hersteller im Bereich der Automatisierungstechnik, der sich insbesondere auf innovative Technologien im Segment Steuerungs-, Visualisierungs- und Antriebstechnik fokussiert.

CIF-BOX (Control Interface Box): Eine standardisierte Schnittstellenbox, bereitgestellt von der HAEFELY AG. Sie dient diversen Prüfungsanwendungen. Das Testmodul dieses Projekts basiert ebenfalls auf dem CIF-BOX-Prinzip und erfüllt identische Standards.

GUI (Graphical User Interface): Das visuelle Benutzeroberflächen-System, welches die Interaktion zwischen Benutzer und Computer ermöglicht.

BC/BK (Bus Controller / Buskoppler): Diese technologischen Komponenten dienen der Verbindung modular erweiterbarer elektronischer Elemente von B&R, insbesondere den I/O-Modulen.

PC (Personal Computer): Ein vielseitig einsetzbarer Computer, dessen Konstruktion und Fähigkeiten speziell darauf ausgerichtet sind, den individuellen und persönlichen Anforderungen des Alltags zu genügen.

Bugs: Unvorhergesehene Programmabweichungen oder -fehler, die zu einem unerwünschten Softwareverhalten führen.

DI/Os (Digitale Inputs/Outputs): Spezialisierte Schnittstellen, die zur Übertragung digitaler Signale konzipiert sind.

AI/Os (Analoge Inputs/Outputs): Schnittstellen, die speziell für die Übertragung analoger Signale entwickelt wurden.

2. Qualifikationsprofil und CV Qualifikationsprofil

Elektrotechniker HF

Entscheidungen fällen Prozess 2	Es wurde eine spezifische Richtlinie bereitgestellt, welche die zu prüfenden Komponenten des Frequenzumrichters detailliert auflistet und klare Vorgaben bezüglich der Gestaltung des GUI-Systems macht. Diese Richtlinie wurde erfolgreich umgesetzt und befolgt.
Sich sprachlich verständigen Prozess 4	Es fand eine regelmäßige und konstruktive Kommunikation mit Kollegen statt, die Experten im Bereich der Automation und Softwareentwicklung sind. Die Kommunikationsprozesse waren beidseitig transparent und klar, und sämtliche Gespräche wurden vollumfänglich in deutscher Sprache geführt.
Unternehmensprozesse verstehen und mitgestalten Prozess 6	Die Haefely AG verwendet B&R als ihr primäres SPS-System und setzt für die SPS-Programmierung den Strukturierten Text als bevorzugte Sprache ein. Das gesamte Steuerungssystem wurde auf Basis einer B&R SPS implementiert. Dabei wurden alle funktionalen Aspekte mittels Strukturiertem Text realisiert. Die Steuerungskonfiguration erfolgte durch den Einsatz des „Automation Studio“, während das Benutzeroberflächensystem mithilfe von „mapp View“ gestaltet wurde.
Probleme analysieren und lösen Prozess 9	Während der Produktionsphase des Systems traten diverse Bugs auf. Diese wurden systematisch durch Einsatz von Debugging-Techniken, der Watch-Funktion und speziellen GUI-Visualisierungsmethoden identifiziert und korrigiert.

Sich persönlich weiter entwickeln Prozess 10	Ich habe mich intensiv mit dem Strukturierten Text beschäftigt und fundierte Kenntnisse in den Bereichen B&R SPS, Automation Studio sowie mapp View erworben.
Programme entwickeln Prozess 12	Innerhalb des Frequenzumrichters sind Steuerungskomponenten von B&R integriert. Diese sind mit einer CIF-BOX verbunden, welche die B&R SPS beinhaltet. Aufgrund dieser die Prüfung der Konfiguration wurde ein spezielles Programm im „Automation Studio“ für die Überprüfung der Komponenten entwickelt und „mapp View“ für die anspruchsvolle Visualisierung verwendet.
Elektrotechnische Anlagen unterhalten Prozess 15	Die Software samt dem grafischen Benutzerinterface (GUI) wurde speziell als Prüfwerkzeug für die Kunden entwickelt. Damit können sie nach Fertigstellung des Frequenzumrichters eigenständig gewährleisten, dass dieser korrekt und gemäß den Spezifikationen konstruiert wurde.

LEBENSLAUF

Personalien

Name Vorname	Vasilakis Mario-Otto
Adresse	Güterstrasse 229
	4053 Basel
Telefon	+41 77 942 80 98
E-Mail	vasilakismario@yahoo.gr
Geburtsdatum	30. Januar 1992
Nationalität	Schweiz und Griechenland
Zivilstand, Kinder	ledig, keine
Familie	3 Brüder und Eltern in der Schweiz

Studium

2021 -2023 Studium an der Schweizerischen Fachhochschule TEKO

Diplom: Elektrotechnik HF

- Themen der Projektarbeiten:

  Videospiel programmieren mit Programm GODOT Engine zur Vertiefung der Programmierung.
  Die Erstellung einer Steuerung für automatische Bewässerung auf Basis SPS Schneider ZelioSoft 2.
  Eine Spielzeugmaus mit Raspberry Pi, Webcam, TensorFlow und cv2 erstellen.

- Thema der Diplomarbeit:
  Automatisierte Prüfung von Frequenzumrichter Schaltschränken.

2018 - 2020 Deutschkurs A2 - B2 bei NSH.

2017 - 2018 Deutschkurs A1 bei ECAP.

2010 - 2018 Studium an der Hochschule für Technologische Anwendung Abteilung Elektroingenieur T.E. Kreta

- Begleitendes Praktikum im elektronischen Schullabor mit Elektroinstallationsarbeiten, Anschluss und Messung von

Elektromotoren, Elektrogeneratoren und Elektrotransformatoren,

- Kurse in Ektroplanung, Elektroinstallation, Energieverteilungen, Elektro- und Ektronikmessungen, Erstellen von Elektroschemas, Anfertigen von Elektroplatinen, Solartechnik, Windtechnik, Programmierung von Arduino, Sprache C++, Matlab,
- Diplomarbeit Thema Simulation eines Inverters für Elektrofahrzeug mit Programm MATLAB

Praktika

07.08. – Aktuell — *Praktikum als Elektroingenieur bei Haefely AG*

- *Prüfung der Frequenzumrichter Schrankkomponenten mittels einer Prüfsequenz mit einer SPS-Steuerung.*

02.08 – 22.12.2022 — Praktikum als Elektroingenieur bei Endress + Hauser Flowtec AG Reinach Abteilung RDEH im Bereich Forschung + Entwicklung

- Optimierung einer Versuchs-Anlage für unsere Messgeräte
- Einarbeitung in die Funktionsweise der Anlage
- (Elektrik und Steuerung von Sensoren und Aktoren, Source-Code der Anlagen-Steuerung)
- Verifikation der implementierten Umrechnung zwischen Masse und Volumen anhand der Wassertemperatur in der Anlage
- Erweiterung der Software zur Ausgabe zusätzlicher Informationen (Log-Daten)
- Untersuchung und Vorschlagen von Verbesserungen, um den Arbeitsbereich aus Druck und Durchfluss der Anlage vollständig nutzen zu können (Regel-Algorithmus)
- Erstellen einer Bedienungsanleitung und Dokumentation

01.01. – 30.06. 2021 — Praktikum als Elektroingenieur bei SKAN AG Allschwil

- Digitalisieren von SIT/FT Dokumenten (Software Integration Test / Functional Test)
- SKAN Logoänderungen in Automation Templates wegen SKAN Rebranding
- Aufsetzen von Window Virtual Machine Station mit VMware Workstation
- Installation von diversen Konfigurationstools auf Virtual Machine

Schule und Ausbildung

2007 - 2010	Professional High School Ierapetras Crete (Elektriker)
2004 - 2007	Gymnasium Ierapetra, Kreta
1998 - 2004	Primarschule Ierapetra, Kreta

EDV - Kenntnisse

Microsoft Office	sehr gute Anwenderkenntnisse
C++	gute Kenntnisse
MATLAB	gute Grundkenntnisse
SPS Siemens S7 TIA Portal V17	gute Kenntnisse
SPS Schneider ZelioSoft 2	gute Kenntnisse
SPS B&R Automation Studio	gute Kenntnisse
Arduino	gute Kenntnisse
Raspberry Pi	gute Kenntnisse
AutoCAD	gute Kenntnisse
Altium (Leiterplatten-Software)	gute Kenntnisse
Eagle (Leiterplatten-Software)	gute Kenntnisse
VMware Workstation	gute Kenntnisse
Visio	Grundkenntnisse
Visual Studio Code	gute Kenntnisse
Git	gute Kenntnisse
EPLAN	gute Kenntnisse
QElectroTech	gute Kenntnisse
GODOT Engine	gute Kenntnisse

Sprachkenntnisse

Griechisch	Muttersprache
Deutsch	gute Kenntnisse in Wort und Schrift (B2)
Englisch	gute Kenntnisse in Wort und Schrift

Interessen

Schachturniere, Fitnesstraining

Referenzen

SKAN AG

Matthias Recher

Projektleiter Automation

Tel. +41 61 485 45 20

Mobil +41 79 629 0649

E-Mail: matthias.recher@skan.ch

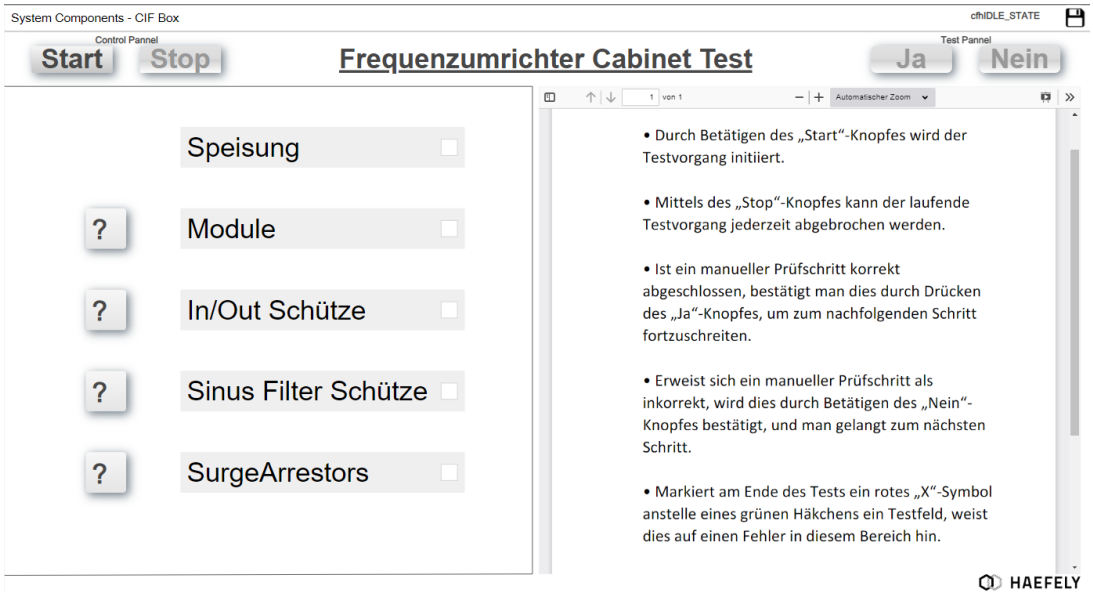
Endress+Hauser Flowtec AG
Thomas Bier
Head of Department | RDEH Utility Products

Phone: +41 61 715 6655

E-Mail: thomas.bier@endress.com

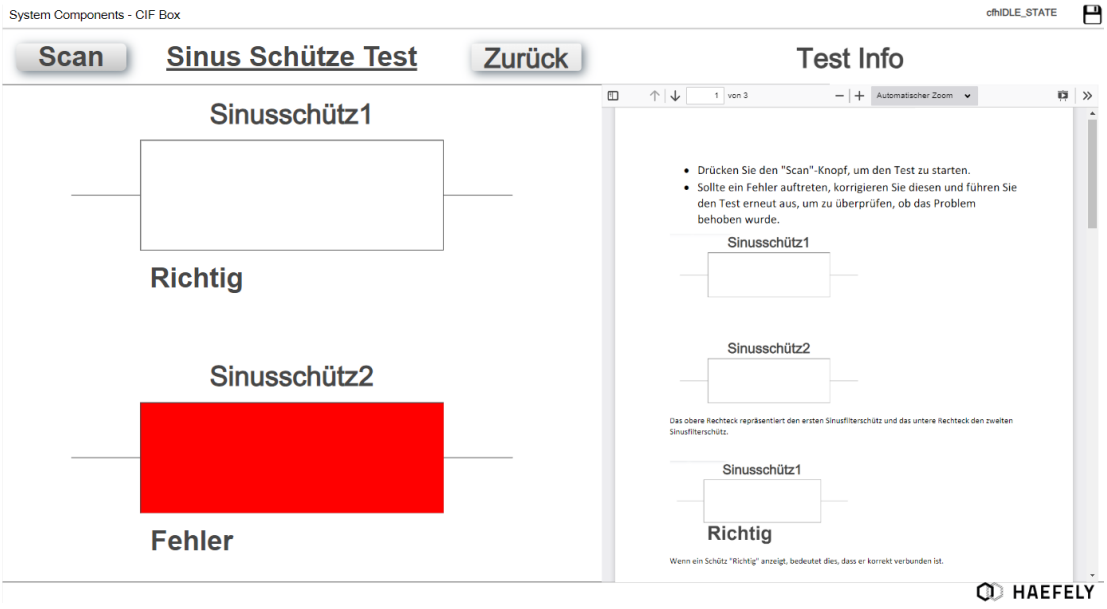
3. Einleitung

Das primäre Ziel dieser Funktionsprüfung besteht darin, die Prüfschritte zur Überprüfung der Funktionalität des Umrichters für den Nutzer so einfach und intuitiv wie möglich zu gestalten. Dies erforderte die Implementierung eines benutzerfreundlichen grafischen Interface-Systems (GUI). Dieses GUI leitet den Nutzer schrittweise durch den Prüfprozess, bietet detaillierte Anleitungen für manuelle Tests und stellt eine Kontexthilfe zur Verfügung, die im Falle eines Fehlers Unterstützung bei dessen Behebung anbietet. Sollte ein automatisierter Test einen Fehler identifizieren, ermöglicht das GUI das Öffnen einer Seite, die genau aufzeigt, wo das Problem liegt. Das GUI-System präsentiert sich wie folgt:



GUI-System (hat die Prüfschritte und Anleitungen)

Um dem Nutzer ein intuitives Verständnis darüber zu vermitteln, ob alle Komponenten korrekt funktionieren, wurden visuelle Implementierungen hinzugefügt. Diese repräsentieren die einzelnen Komponenten des Frequenzumrichters. Durch eine einfache Rotcodierung wird unmittelbar angezeigt, ob ein Fehler vorliegt. Eine detaillierte Darstellung dieser Farbcodierung finden Sie im Bild „Farbcodierung" weiter unten:



Farbcodierung (zeigt mit rot, wenn es einen Fehler gibt)

4. Initialisierung

4.1. Allgemeine Ziele

- Die entwickelte Software soll künftig von Kunden eingesetzt werden, um den Frequenzumrichter nach dessen Fertigstellung eingehend zu überprüfen.
- Bei Vorliegen eines Fehlers ermöglicht die Software dem Anwender, unmittelbar die exakte Problemstelle zu identifizieren.
- Das GUI wird mit integrierten Anleitungen ausgestattet, die dem Nutzer im Fehlerfall konkrete Lösungsansätze präsentieren.

4.2. Spezifische Ziele

- Das GUI-System soll eine intuitive und benutzerfreundliche Handhabung gewährleisten.
- **Speisung Test:** Dem Kunden soll visuell verdeutlicht werden, ob die Speisung korrekt konfiguriert ist.
- **Module Test:** Der Kunde soll in der Lage sein zu überprüfen, ob sämtliche B&R-Module – einschließlich SPS, Buscontroller, DI/Os, AI/Os und dergleichen – nicht nur korrekt gemäß der Softwarekonfiguration installiert sind, sondern auch ob sie an den vorgesehenen Positionen platziert und nicht fehlerhaft vertauscht wurden.
- **Ein-Ausgangsschütze Test:** Der Kunde soll klar erkennen können, ob die B&R DI/Os, welche mit den Eingangsschützen sowie den Ausgangsschützen verbunden sind, korrekt konfiguriert und funktionsfähig sind.
- **Sinusschütze Test:** Dem Kunden soll ermöglicht werden zu prüfen, ob die B&R DI/Os, die mit den Sinusschützen verknüpft sind, adäquat konfiguriert und voll funktionsfähig sind.
- **Surge Arrestors Test:** Der Kunde soll in der Lage sein, zu überprüfen, ob die B&R DI/Os, die in Verbindung mit den Surge Arrestors stehen, korrekt positioniert und funktional sind.

4.3. Anforderungen und Systemvoraussetzung

Als Projektleiter – in diesem Fall meine Person – hatte ich vor Beginn des Projekts keinerlei Erfahrung mit B&R SPS und der SPS-Programmiersprache "Strukturierter Text". Daher war es unabdingbar, rasch Kenntnisse in diesem Bereich zu erwerben. Ein initialer Schritt war das Erstellen eines kleineren Beispielprojekts, um einen schnellen Einstieg und die entsprechende Einarbeitung zu gewährleisten.

Der Frequenzumrichter, ein Produkt der Haefely AG, wurde gemäß spezifischen Normen und Standards hergestellt. Dies impliziert, dass die SPS-Software so konzipiert werden musste, dass sie die bereits existierenden B&R Komponenten im Schaltschrank des Frequenzumrichters nahtlos integriert, ohne dass zusätzliche Komponenten erforderlich sind.

Aufgrund der Tatsache, dass die Haefely AG B&R als primäres SPS-System verwendet, war der Einsatz der zugehörigen Softwarelösungen „Automation Studio“ und „mapp View“ unerlässlich.

Da die Software speziell für Kunden entwickelt wurde, die den Frequenzumrichter selbst herstellen, war es von essentieller Bedeutung, dass sie besonders benutzerfreundlich gestaltet ist. Dies garantiert, dass sie von jedem Anwender, unabhängig von dessen Vorkenntnissen, problemlos genutzt werden kann.

- **Softwareaufbau**
- **GUI-Systemaufbau**

4.4. Aufbau

Es war kein zusätzlicher Hardwareaufbau erforderlich, da die gesamte Hardware bereits von der Haefely AG bereitgestellt wurde. Dies umfasste den Frequenzumrichter sowie die CIF-BOX. Sowohl der Frequenzumrichter als auch die CIF-BOX integrieren B&R Komponenten. Die Software führt die Prüfung des Frequenzumrichters ausschließlich über diese Komponenten durch.

Die Softwareentwicklung erfolgte unter Verwendung von „Automation Studio“ und „mapp View“, wobei als SPS-Programmiersprache der "Strukturierte Text" zum Einsatz kam.

Die nachfolgend aufgeführten Schritte wurden während des Softwareaufbaus befolgt:

- Befolgung der Richtlinie, welche Komponenten des Frequenzumrichters und in welcher Reihenfolge sie geprüft werden sollten.
- Analyse des Elektroschemas des Frequenzumrichters, um Lösungsansätze zur Prüfung der jeweiligen Komponenten zu entwickeln.
- Segmentierung der Frequenzumrichterprüfung in verschiedene Abschnitte, wobei jeder Prüfschritt als eigenständiger Funktionsbaustein definiert wurde.
- Jeder einzelne Prüfschritt folgte einem strukturierten Vorgehen: Programmierung in der SPS, Testen der Software im Demo-System, das den Frequenzumrichter repräsentiert, Fehlerkorrekturen und, sobald ein Schritt erfolgreich abgeschlossen war, Beginn des nächsten Prüfschritts.

4.5. Projektorganisation

Die Projektorganisation war so strukturiert:

Kunde: Haefely AG

Auftraggeber: Adrian John (Haefely AG)

Projektbetreuer: Anna Koziec (Haefely AG)

Diplombetreuer: Patrick Grubert (TEKO)

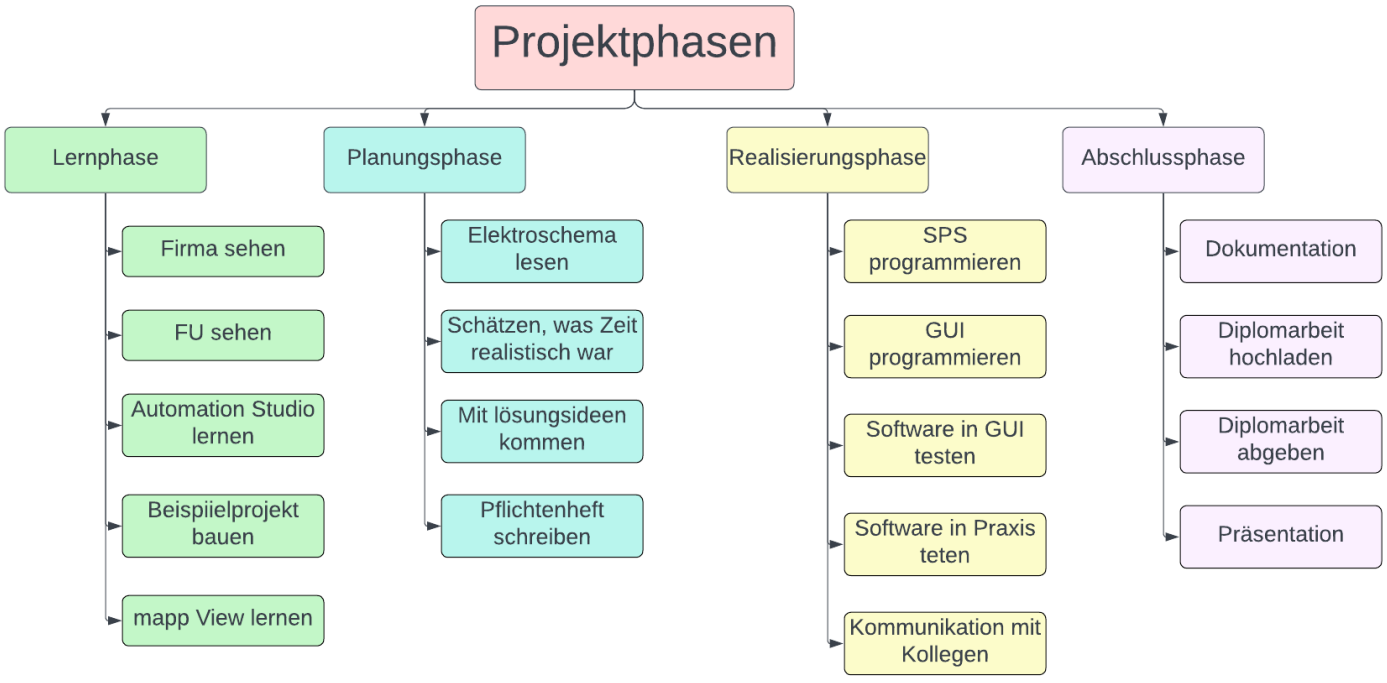
Projektleiter: Mario Vasilakis (Haefely AG)

5. Planung

5.1. Projektverteilung laut Pflichtenheft

Vorbereitungsphase	
Projektantrag	15.06.2023
Pflichtenheft anfangen	17.08.2023
Pflichtenheft beenden	18.08.2023
Hardware und Software lernen (Lernen Phase Anfang)	17.08.2023
Hardware und Software lernen (Lernen Phase Ende)	21.08.2023
Anfang des Projekts	28.08.2023
Realisierung	
Anfang der Programmierung	28.08.2023
Ende der Programmierung	15.09.2023
Dokumentation anfangen	16.09.2023
Dokumentation beenden	27.09.2023
Diplomarbeit online hochladen	28.09.2023
Durchführung	
Projekt Endprodukt	21.09.2023
Projekt letzter Test	21.09.2023
Projekt beenden	27.09.2023
Abgabe des Projekts	28.09.2023
Präsentation	12.10.2023

5.2. Projektphasendiagram

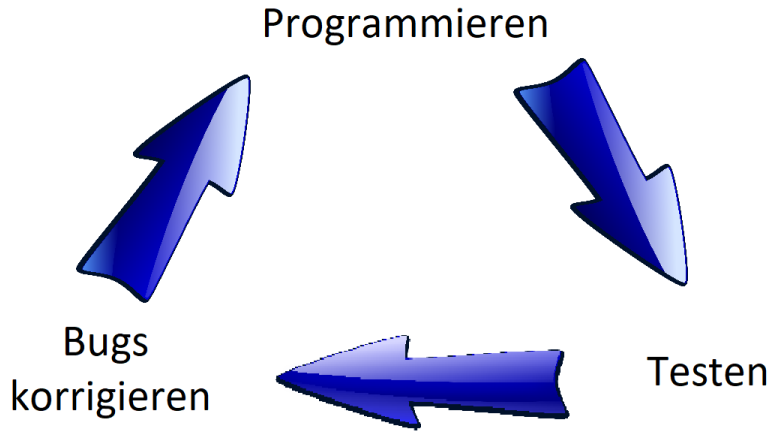


5.3. Projektphasenbeschreibung

Lernphase: Vor Beginn dieses Projekts hatte ich, in meiner Rolle als Projektleiter, noch keine Erfahrungen mit der Haefely AG gesammelt. Daher war eine eingehende Einführung in das Unternehmen sowie eine direkte Begutachtung des Frequenzumrichters vor Ort unerlässlich, um ein umfassendes Verständnis zu entwickeln. Wie bereits erwähnt, war dies meine erste Begegnung mit B&R SPS und dem "Strukturierten Text". In Vorbereitung auf das Hauptprojekt war es sinnvoll, mir eine Woche Zeit zu nehmen, um mich intensiv mit den B&R Komponenten, der Software "Automation Studio" und "mapp View" vertraut zu machen. In dieser Lernphase wurde zudem ein einfaches Beispielprojekt mit B&R SPS, B&R DI/Os, zwei Relais und einem Schaltkomponenten realisiert. Dieses Beispielprojekt diente später als Prototyp zur Überprüfung eines Funktionsblocks, um sicherzustellen, dass die Software die konfigurierten B&R Komponenten im Hardwaresystem korrekt identifizieren kann. Nach erfolgreichem Test wurde diese Funktion, mit einigen Anpassungen, in das endgültige Projekt integriert.

Planungsphase: Im Anschluss an die Lernphase initiierten wir eine ausführliche Planungsphase. In enger Abstimmung mit dem Auftraggeber legten wir fest, welche Komponenten des Frequenzumrichters essentiell für die Prüfung waren und wie viele dieser Elemente innerhalb des vorgegebenen Zeitrahmens effizient programmierbar sind. Nach der Definition der zu prüfenden Komponenten analysierte ich das Elektroschema, um konkrete Lösungsansätze zur Überprüfung der einzelnen Elemente zu entwickeln. Zum Abschluss dieser Phase wurde das Pflichtenheft erstellt.

Realisierungsphase: Diese Phase war zweifellos die intensivste. Ihr Verlauf kann als zyklisch oder, treffender, dreiecksförmig beschrieben werden. Jeder Prüfschritt des Frequenzumrichters folgte einem festgelegten Ablauf: Erstellung des Programmcodes und des GUI, digitale und physische Tests, Feedbacksammlung und Fehlerbehebung. Erst nachdem ein Zyklus erfolgreich und fehlerfrei abgeschlossen war, wurde mit dem nächsten Prüfschritt begonnen. Während dieses Prozesses erwies sich der regelmäßige Austausch mit erfahrenen Kollegen als besonders wertvoll, insbesondere bei technischen Herausforderungen.



Dreieck von Realisierungsphase (Die drei wiederholte Schritte)

Dokumentationsphase: Zum Abschluss des Projekts verblieb eine Woche, um die Dokumentation zu finalisieren und gemeinsam mit den dazugehörigen Anhängen einzureichen.

5.4. Wahre Projektverteilung

Vorbereitungsphase	
Projektantrag	15.06.2023
Pflichtenheft anfangen	17.08.2023
Pflichtenheft beenden	25.08.2023
Hardware und Software lernen (Lernen Phase Anfang)	17.08.2023
Hardware und Software lernen (Lernen Phase Ende)	25.08.2023
Anfang des Projekts	28.08.2023
Realisierung	
Anfang der Programmierung	28.08.2023
Ende der Programmierung	21.09.2023
Dokumentation anfangen	22.09.2023
Dokumentation beenden	27.09.2023
Diplomarbeit online hochladen	28.09.2023
Durchführung	
Projekt Endprodukt	21.09.2023
Projekt letzter Test	21.09.2023
Projekt beenden	27.09.2023
Abgabe des Projekts	28.09.2023
Präsentation	12.10.2023

5.5. Prüfschritteverteilung

Wie bereits erwähnt, wurden fünf Prüfschritte definiert, von denen angenommen wurde, dass ihre Programmierung zeitlich realisierbar ist. Der Programmierzeitraum erstreckte sich, wie bereits angegeben, vom 28.08.2023 bis zum 21.09.2023. Allerdings konnte der Prüfschritt für den Modulentest bereits während der Lernphase abgeschlossen werden. Zum Programmierzeitraum zählte ebenfalls die Entwicklung des GUI. Das Ziel dabei war, ein Nutzer-Interface zu schaffen, das möglichst einfach und intuitiv für die Kunden zu bedienen ist. Die Aufteilung der Prüfschritte gestaltete sich folgendermaßen:

Module Test (Anfang Programmierung)	23.08.2023
Module Test (Ende Programmierung)	25.08.2023
Speisung Test (Anfang Programmierung)	28.08.2023
Speisung Test (Ende Programmierung)	28.08.2023
Sinusschütze Test (Anfang Programmierung)	28.08.2023
Sinusschütze Test (Ende Programmierung)	04.09.2023
Surge Arrestors Test (Anfang Programmierung)	11.09.2023
Surge Arrestors Test (Ende Programmierung)	15.09.2023
Ein-Ausgangsschütze Test (Anfang Programmierung)	18.09.2023
Ein-Ausgangsschütze Test (Ende Programmierung)	22.09.2023
GUI-Entwicklung (Anfang Programmierung)	28.08.2023
GUI-Entwicklung (Ende Programmierung)	22.09.2023

6. Realisierung

6.1. Beispielprojekt

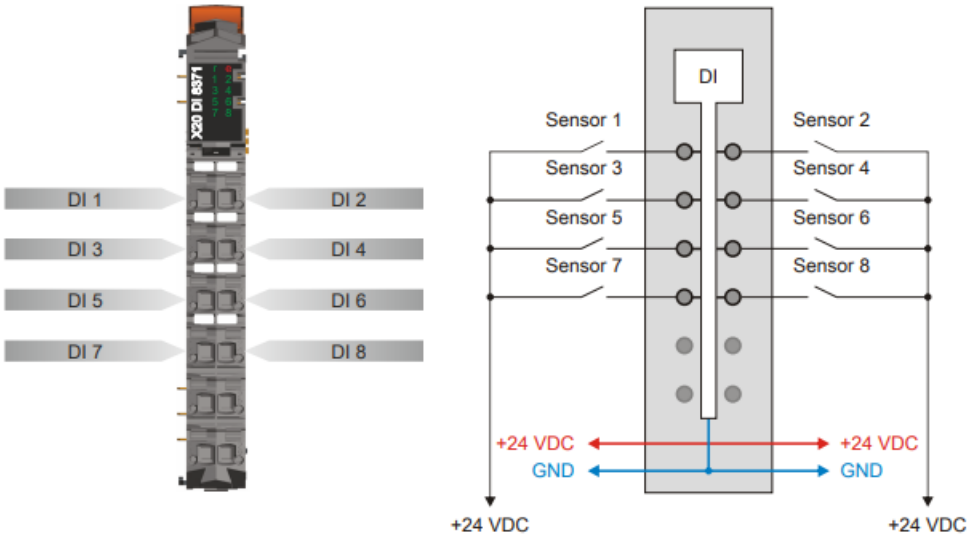
Um mich mit der B&R SPS vertraut zu machen, war es vorteilhaft, ein Beispielprojekt zu erstellen. Zahlreiche Erkenntnisse und Elemente aus diesem Vorprojekt fanden schließlich Eingang in das finale Projekt.

6.1.1. Entscheidung von Komponenten

Die im Beispielprojekt verwendeten Komponenten dienten dazu, die Elemente des Frequenzumrichters zu simulieren. Beispielsweise wurden zwei Relais eingesetzt, welche die Funktion der Schütze des Frequenzumrichters nachahmten. Außerdem kamen B&R-Komponenten zum Einsatz. Hierbei wurde eine Funktion entwickelt, um zu überprüfen, ob die konfigurierten Module (SPS, DI/Os, BC) korrekt integriert waren. Diese Funktion wurde schließlich auch in das Endprodukt übernommen.

6.1.2. Entscheidung von Komponenten

Zur Einarbeitung in die B&R Komponenten waren die bereitgestellten Datenblätter äußerst hilfreich, wie beispielsweise das Datenblatt für den Digital Input X20DI8371:

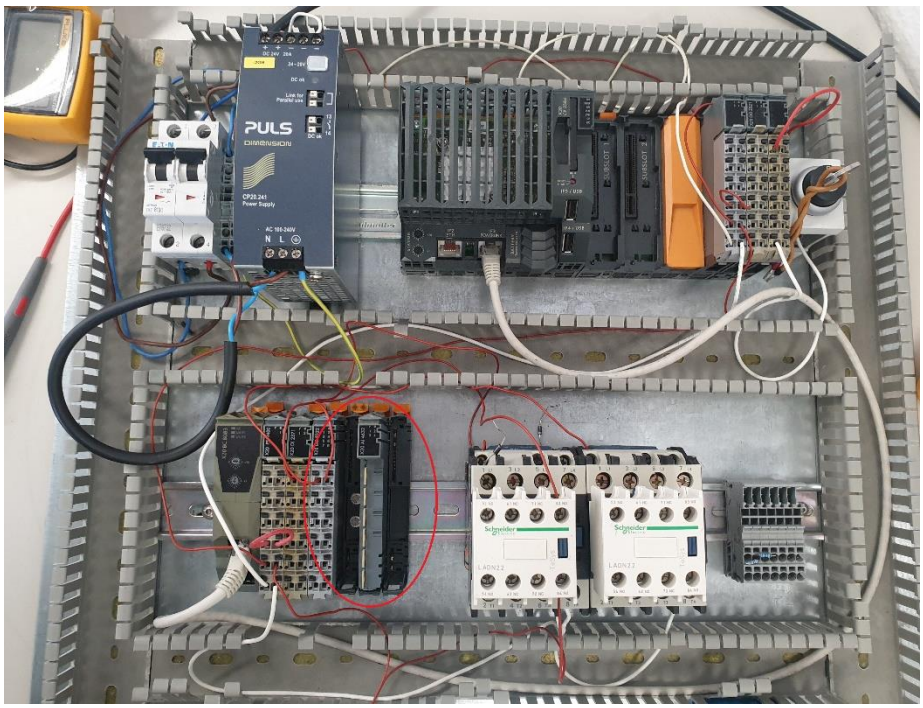


B&R Datasheets (die kann man online finden https://www.br-automation.com/de-ch)

Durch die Datenblätter konnte problemlos ermittelt werden, welcher Pin welche Funktion besitzt.

6.1.3. Beispielprojekt Hardware

Die Hardware des Beispielprojekts war bewusst schlicht gehalten, um bestimmte Funktionen testen zu können. Neben den B&R-Komponenten wurden zwei Relais und eine Schaltkomponente integriert. Die Hardwarekonfiguration gestaltete sich wie folgt:

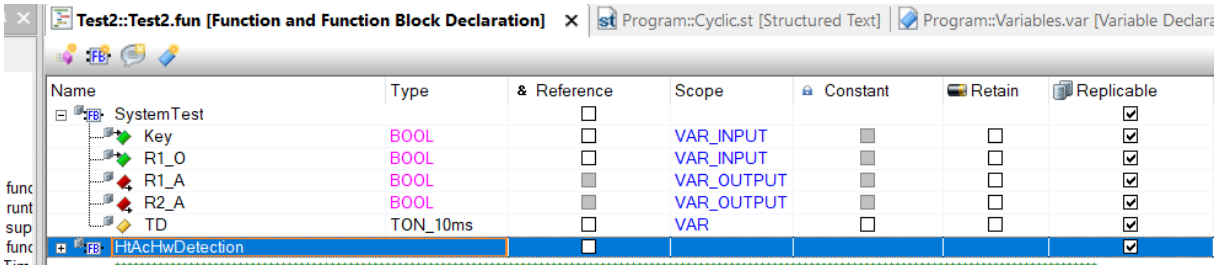


Beispielprojekt Hardware (Innerhalb des roten Kreises befindet sich der Bereich, in dem mit den Modulen experimentiert wurde)

6.1.4. Beispielprojekt Software

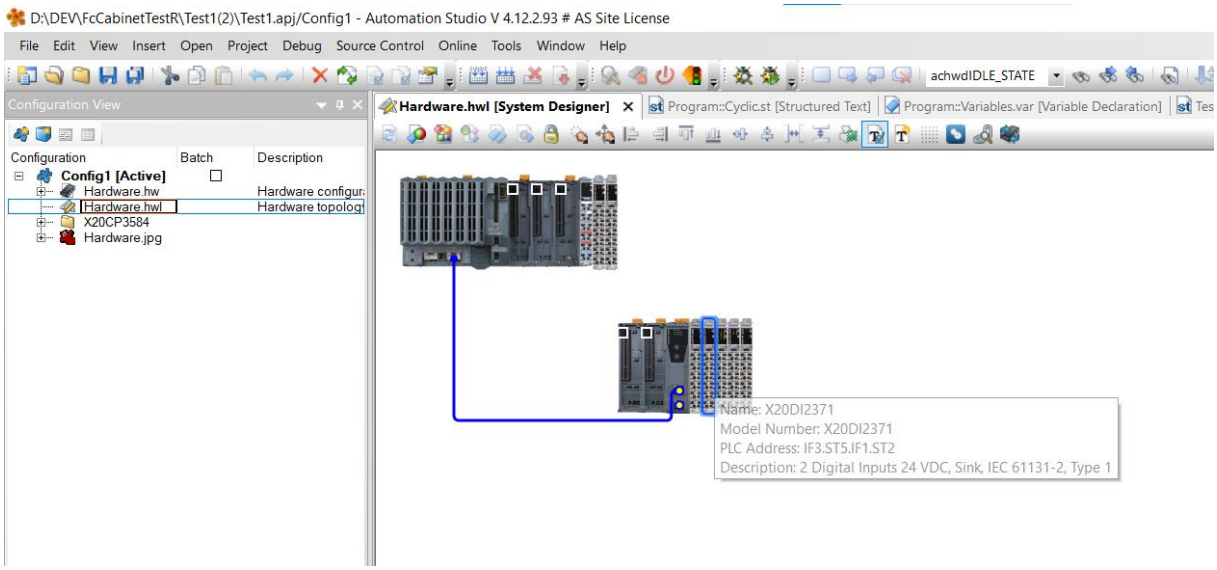
Die Software beinhaltet alle essenziellen Funktionen, die für ein grundlegendes Verständnis der B&R SPS-Programmierung erforderlich sind:

- Wie man Variable definiert



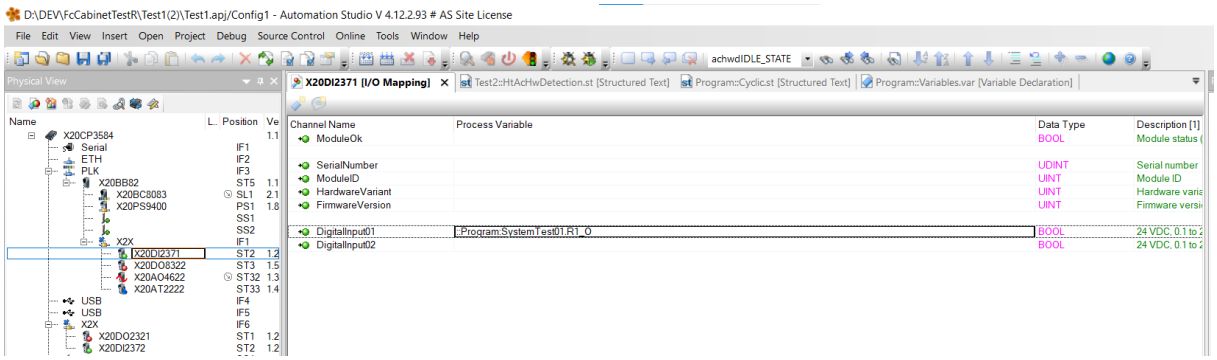
Variable (Es besteht die Möglichkeit, für jede Variable deren Art, wie z.B. Input, Output, und deren Typ, wie BOOL, INT, STRING etc., zu definieren)

- Wie man die B&R Module konfiguriert



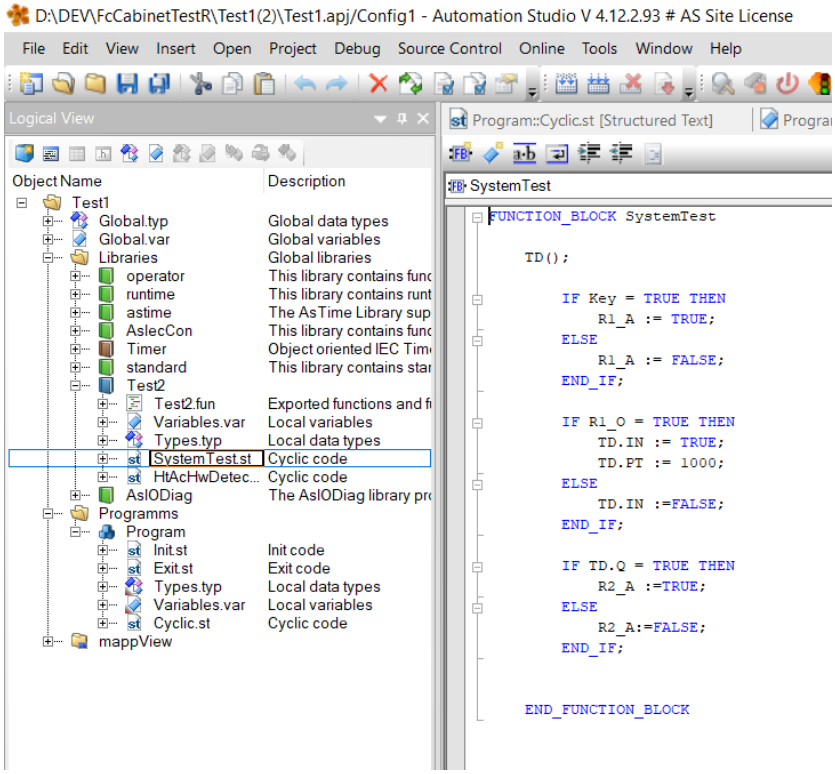
Modulekonfiguration

- Wie man die Variablen mit den Pins der B&R Module verbindet



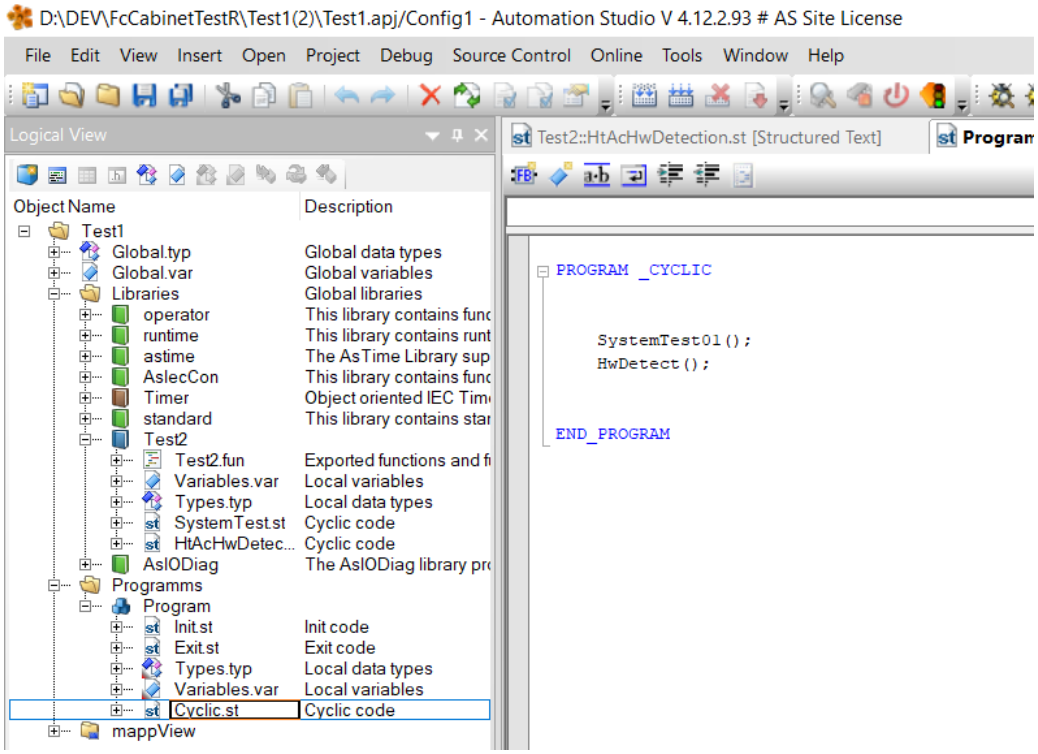
Variablen mit Modulen verbinden

- Wie man ein Funktionsblock baut



Funktionsblocks (die sind notwendig, um ein Programm strukturiert zu halten)

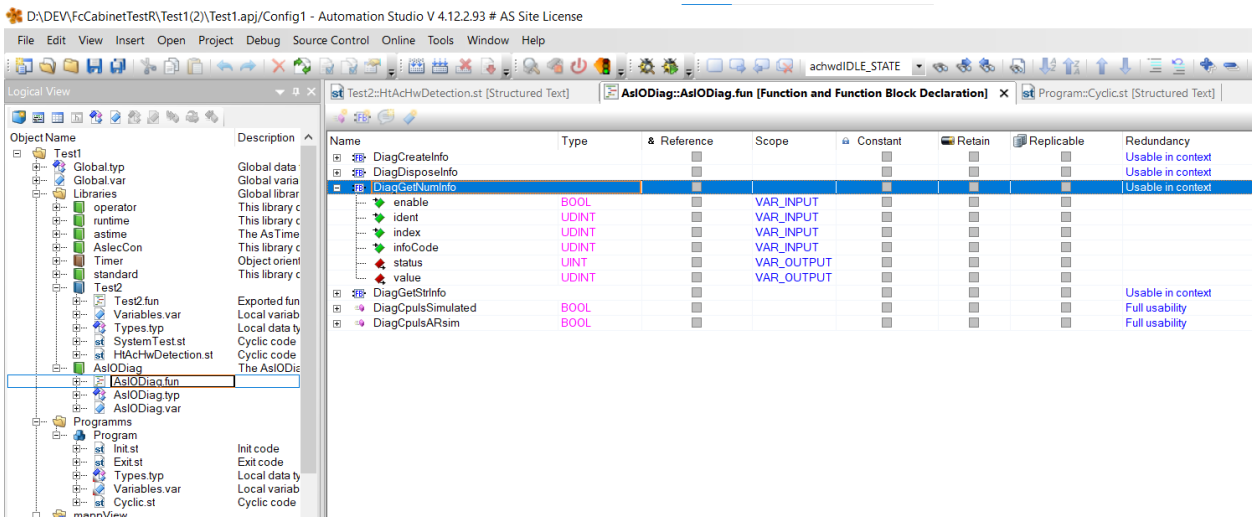
- Wie man sowohl selbst erstellte als auch aus Bibliotheken stammende Funktionsblöcke im Hauptprogramm aufrufen kann



Einbindung von Funktionsblöcken ins Hauptprogramm (zur Vereinfachung des Codes)

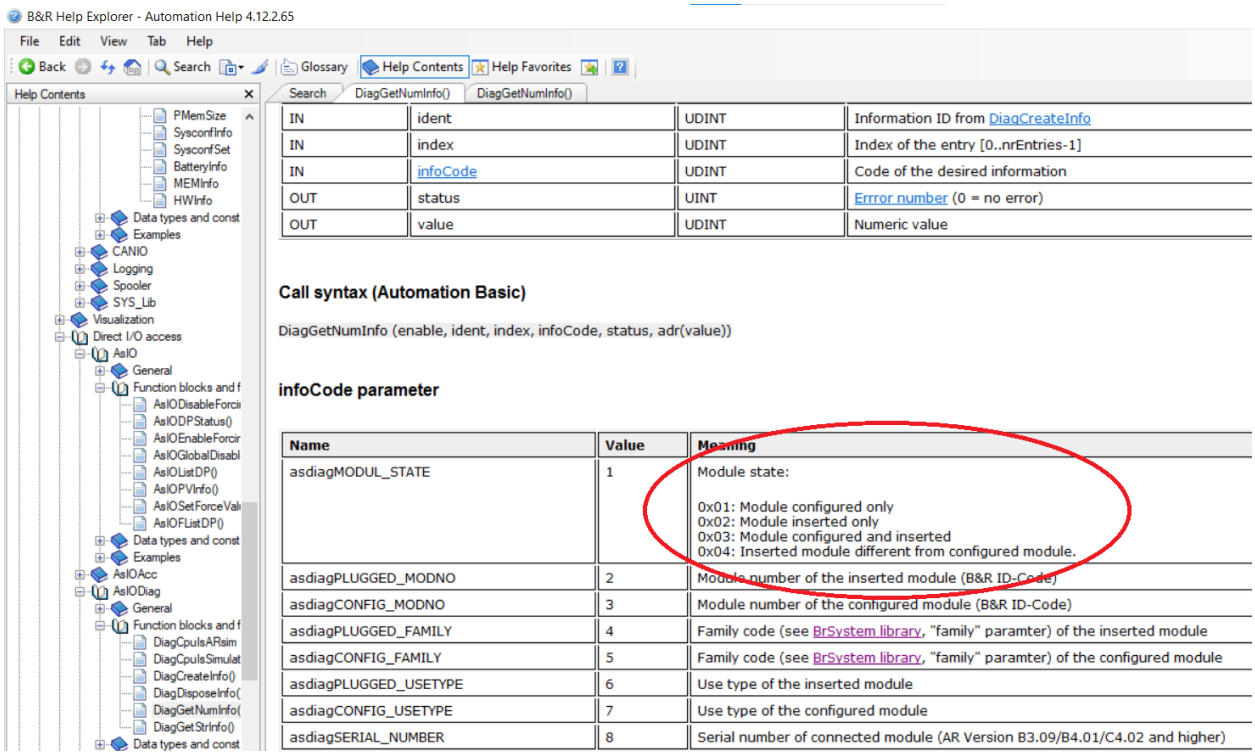
6.1.5. Module Test in Beispielprojekt

Im Beispielprojekt war es möglich, die AsIODiag-Bibliothek hinzuzufügen, wie auf der untenstehenden Abbildung ersichtlich ist:



AsIODiag-Bibliothek (Diese wurde speziell für den Modultest implementiert)

Die AsIODiag-Bibliothek beinhaltet eine Funktion namens "DiagGetNumInfo". Diese Funktion gibt an, in welchem Zustand sich ein Modul befindet, wie im untenstehenden Bild am markierten Bereich im roten Kreis ersichtlich ist:

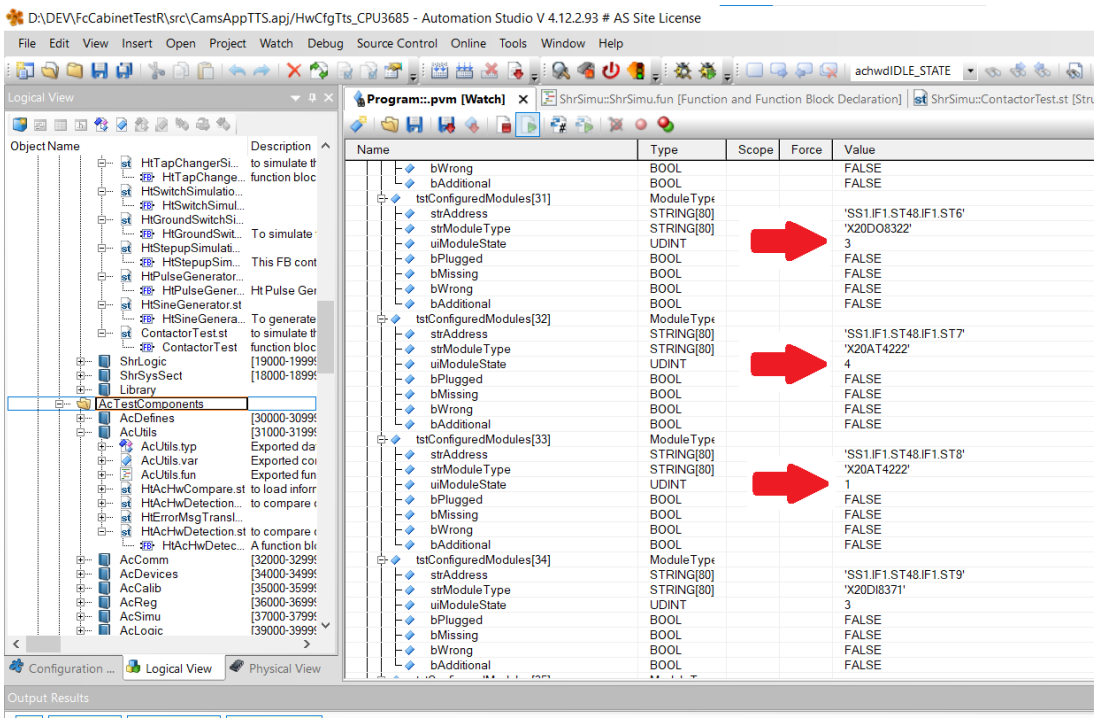


Hilfeinformation zu "DiagGetNumInfo" (Diese Funktion zeigt den aktuellen Zustand eines Moduls an)

Die Funktion "DiagGetNumInfo" war essenziell für den Modultest, da sie aufzeigt, ob alle in der Software konfigurierten Module korrekt platziert sind. Sie unterscheidet zwischen vier Zuständen:

1. **Module configured only:** Dieser Zustand signalisiert, dass ein in der Software konfiguriertes Modul physisch nicht eingefügt wurde.
2. **Module inserted only:** Dieser Status zeigt an, dass ein Modul physisch vorhanden ist, welches in der Software jedoch nicht konfiguriert wurde.
3. **Module configured and inserted:** Dieser Zustand bestätigt, dass das Modul sowohl in der Software konfiguriert als auch physisch korrekt eingefügt wurde.
4. **Inserted module different from configured module:** Dieser Status weist darauf hin, dass ein Modul an einer Stelle eingefügt wurde, an der in der Software ein anderes Modul konfiguriert ist.

Haefely AG stellte bereits einen Funktionsblock zur Verfügung, der diese Funktion integrierte. Dies ermöglichte im Beispielprojekt Experimente mit dem Modultest. Insbesondere gab es einen Bereich (siehe Bild "Beispielprojekt Hardware"), in dem Module entfernt, falsche Module an deren Stelle eingesetzt werden konnten und überprüft wurde, ob die Watch-Funktion in "Automation Studio" die verschiedenen Zustände korrekt anzeigte:



Watch von konfigurierten Modulen (der zeigt beim welchen State ein Modul ist)

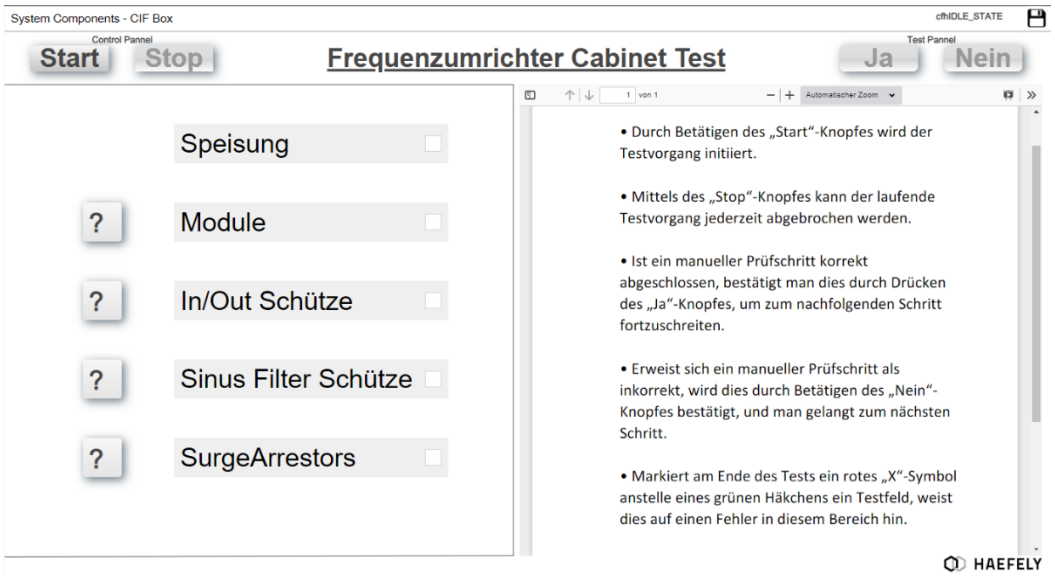
Wie man erkennen kann, gibt die Funktion in der "Watch"-Ansicht für jedes Modul den aktuellen Zustand zurück. Hierbei wird der Typ des konfigurierten Moduls, beispielsweise "X20AT4222", zusammen mit seinem Status (in diesem Fall 4) angezeigt. In dem vorliegenden Beispiel zeigt der Status 4 bei dem konfigurierten Modul "X20AT4222" an, dass an dieser Stelle ein anderes Modul eingesteckt wurde, was nicht korrekt ist.

Mit Hilfe des Beispielprojekts und dieser speziellen Funktion konnte effektiv experimentiert werden. Anschließend wurde die Funktion mit geringfügigen Anpassungen in das Hauptprojekt integriert.

6.2. Projekt

6.2.1. GUI-Entwicklung

Wie bereits erläutert, erstreckte sich die Entwicklung des GUI von Beginn bis zum Ende der Programmierphase. Das Grundgerüst wurde relativ frühzeitig erstellt, wobei kleinere Implementierungen häufig zu einem späteren Zeitpunkt hinzugefügt wurden. Dies setzte sich fort, bis das GUI seine endgültige Form erlangte. Es war von entscheidender Bedeutung, dass die endgültige Gestaltung des GUI so intuitiv wie möglich war, sodass es von jedem Benutzer problemlos genutzt werden konnte. Die finale Darstellung des GUI präsentiert sich wie folgt:



GUI-System (entworfen für die Benutzeroberfläche der Kunden)

Gemäß der Bedienungsanleitung funktioniert das GUI-System wie folgt:

- Durch Betätigen des „Start"-Knopfes wird der Testvorgang initiiert.
- Mittels des „Stop"-Knopfes kann der laufende Testvorgang jederzeit abgebrochen werden.
- Ist ein manueller Prüfschritt korrekt abgeschlossen, bestätigt man dies durch Drücken des „Ja"-Knopfes, um zum nachfolgenden Schritt fortzuschreiten.
- Erweist sich ein manueller Prüfschritt als inkorrekt, wird dies durch Betätigen des „Nein"-Knopfes bestätigt, und man gelangt zum nächsten Schritt.
- Markiert am Ende des Tests ein rotes „X"-Symbol anstelle eines grünen Häkchens ein Testfeld, weist dies auf einen Fehler in diesem Bereich hin.
- Durch Anklicken des Fragezeichen-Symbols, positioniert links neben dem Testfeld, erhält man detaillierte Informationen über die Ursache des Fehlers.

Die Bezeichnung „Prüfschritte“ bezieht sich auf die fünf spezifizierten Tests: den Speisungstest, den Moduletest, den Test der Ein- und Ausgangsschütze, den Sinusfilterschützetest sowie den Surge Arrestors Test.

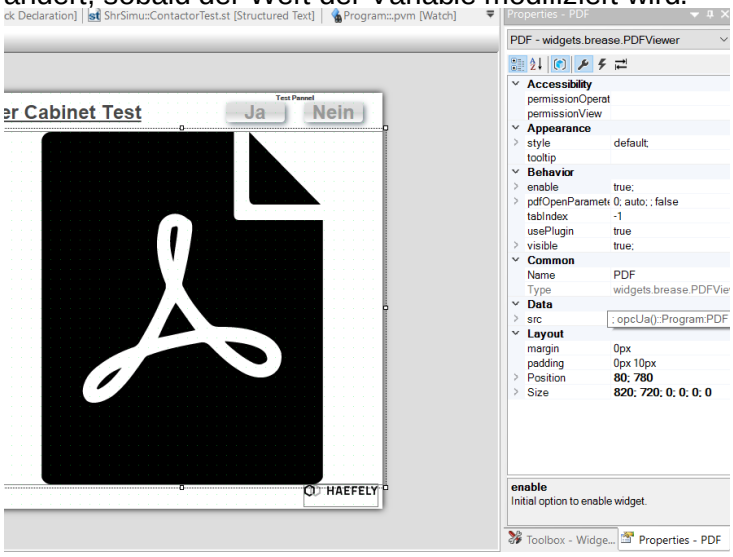
Unter „manuellen Prüfschritten“ versteht man jene Tests, die aufgrund fehlender Anbindung an ein B&R-Modul nicht automatisiert überprüft werden konnten. Der Frequenzumrichter wurde gemäß festgelegten Standards konzipiert, was das Hinzufügen neuer Module und deren Verknüpfung mit den Komponenten des Frequenzumrichters ausschließt. Deshalb war es notwendig, einen alternativen Ansatz zu finden, um diese Komponenten manuell zu überwachen.

So war die Entwicklung des GUI-Systems in Automation Studio:

- Zuerst war es notwendig, eine Seite (Page) zu generieren. Diese Seite stellt die Benutzeroberfläche dar, die dem Kunden präsentiert wird, sobald er das GUI verwendet.

GUI-System Entwicklung von Hauptpage

Die Seite ist mit der PDF-Variable verknüpft, sodass sich das angezeigte PDF entsprechend ändert, sobald der Wert der Variable modifiziert wird.

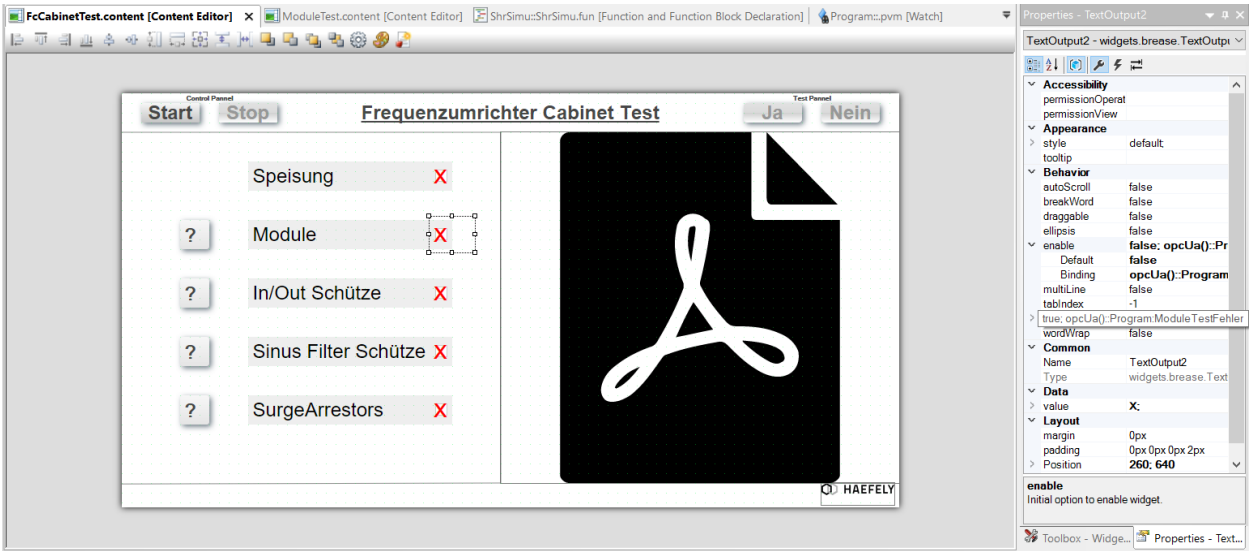


PDF Verbindung mit der PDF-Variable

```
1:
    HwDetect.ModuleTestEndeC:=TRUE;
    ContactorBausteinSinus.ContactorTestEndeC:=TRUE;
    ContactorBausteinSchutze.ContactorTestEndeC:=TRUE;
    SurgeArrestorsBaustein.SurgeArrestorsTestEndeC:=TRUE;
    PDF := 'Media\PDF\SpeisungPage.pdf';
    IF TestNP THEN
        Speisung:= TRUE;
        TestN:=2;
    END_IF

    IF FehlerInfo THEN
        SpeisungFehler:= TRUE;
        TestN:=2;
    END_IF
2:
    TestNP:= FALSE;
    FehlerInfo:=FALSE;

    PDF := 'Media\PDF\LaufendesTest\ModuleTestLauft.pdf';

    HwDetect.BeginModuleTest:=TRUE;

    IF HwDetect.ModuleTestEnde THEN
        TestN:=3;
    END_IF
3:
    PDF := 'Media\PDF\LaufendesTest\INOUTlauft.pdf';

    ContactorBausteinSchutze.bStart:=TRUE;
```

Beispiel von verschiedenen PDF String-Werte (rosa Buchstabe)

Die Rechtecke, welche entweder grüne Häkchen oder rote X-Symbole darstellen, sind mit BOOL-Variablen verknüpft. Diese geben an, ob ein Test erfolgreich war oder nicht, repräsentiert durch die Zustände TRUE oder FALSE.



Verbindung der erster roter X mit der BOOL Variable ModuleTestFehler

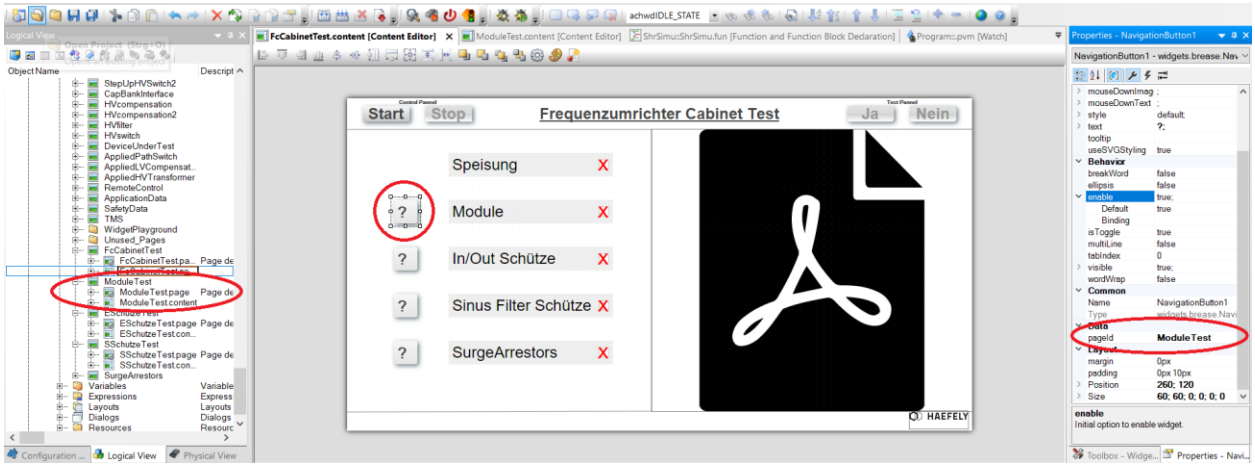
Falls die Variable "ModuleTestFehler" den Wert TRUE annimmt, weist dies auf einen Fehler im Module Test hin. In einem solchen Fall wird das rote X-Symbol aktiviert und sichtbar gemacht.

```
IF TestN > 2 AND HwDetect.MTOF_ALL THEN
    ModuleTest := TRUE;
ELSE    ModuleTest := FALSE;
END_IF
IF TestN > 2 AND NOT HwDetect.MTOF_ALL THEN
    ModuleTestFehler := TRUE;
ELSE    ModuleTestFehler := FALSE;
END_IF
```

Variablen ModuleTest und ModuleTestFehler

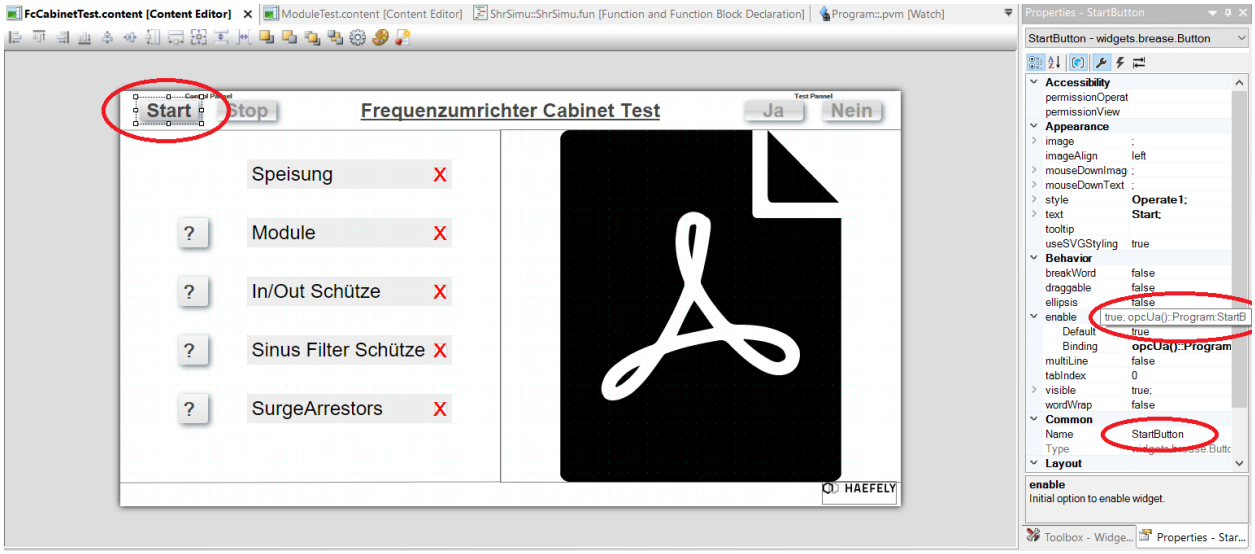
Die Variable "ModuleTest" steuert die Aktivierung und Deaktivierung des grünen Häkchens, während die Variable "ModuleTestFehler" das rote X-Symbol aktiviert oder deaktiviert.

Die Schaltflächen mit den Fragezeichen-Symbolen fungieren als Navigationsbuttons. Bei Betätigung leiten sie den Nutzer zu einem anderen Bereich weiter. Beispielsweise führt die erste dieser Schaltflächen zur Seite "ModuleTestPage":



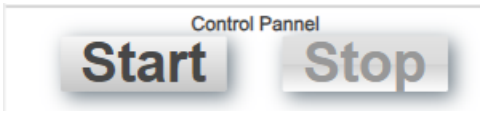
Funktion der Navigationsknöpfe (Öffnung von ModuleTestPage)

Die Schaltflächen "Start", "Stop", "Ja" und "Nein" agieren als Momentary Push Buttons. Sie sind mit bestimmten Events verknüpft, sodass sie diverse Aktionen auslösen oder beenden können. Nachfolgend ein Beispiel bezogen auf den "Start"-Button:

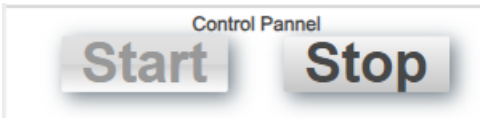


Start Knopf Verbindungen

Der "Start"-Knopf trägt die Bezeichnung "StartButton", und seine Aktivierung bzw. Deaktivierung wird durch die Variable "StartB" gesteuert. Ist der Knopf aktiviert, lässt er sich betätigen. Ist er hingegen deaktiviert, kann er nicht gedrückt werden. Wenn beispielsweise bereits ein Test läuft, wäre es unlogisch, den "Start"-Knopf erneut zu betätigen. In solch einem Szenario wäre es ratsam, den laufenden Test über den "Stop"-Knopf zu beenden und dann den Test erneut zu starten:

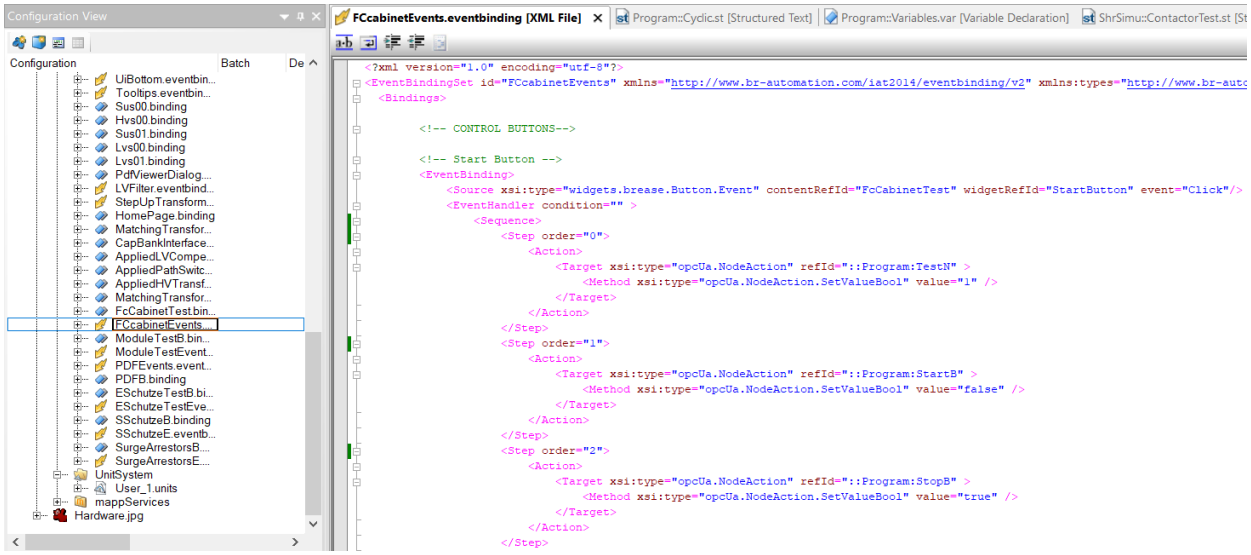


Start Knopf aktiviert



Start Knopf deaktiviert

Um bestimmte Aktionen auszulösen oder zu unterbinden, sind Eventbindings erforderlich. Im Folgenden wird ein Beispiel für die Eventverknüpfungen des "Start"-Knopfes präsentiert:



Start Knopf Events 1.

Start Knopf Events 2.

Folgende Aktionen werden ausgelöst, sobald der "Start"-Knopf betätigt wird:

i. Die Variable TestN nimmt den Wert 1 an. Sie ist eine UDINT-Variable, die Werte zwischen 0 und 6 annehmen kann. Jeder dieser Werte entspricht einem bestimmten Prozessschritt: Bei Wert 0 befindet man sich auf der Hauptseite, bei Wert 1 wird der Speisungstest ausgeführt, bei Wert 2 der Module Test und so weiter. Durch Betätigen des "Start"-Knopfes wechselt TestN auf den Wert 1 und der erste Prüfschritt, der Speisungstest, wird eingeleitet.

ii. Die Variable StartB wechselt ihren Zustand auf FALSE, wodurch der "Start"-Knopf deaktiviert wird. Der Hintergrund dieser Aktion wurde bereits erläutert.

iii. StopB nimmt den Wert TRUE an, wodurch der "Stop"-Knopf aktiviert wird. Der Benutzer hat nun die Möglichkeit, den Prozess durch Betätigen des "Stop"-Knopfes zu beenden.

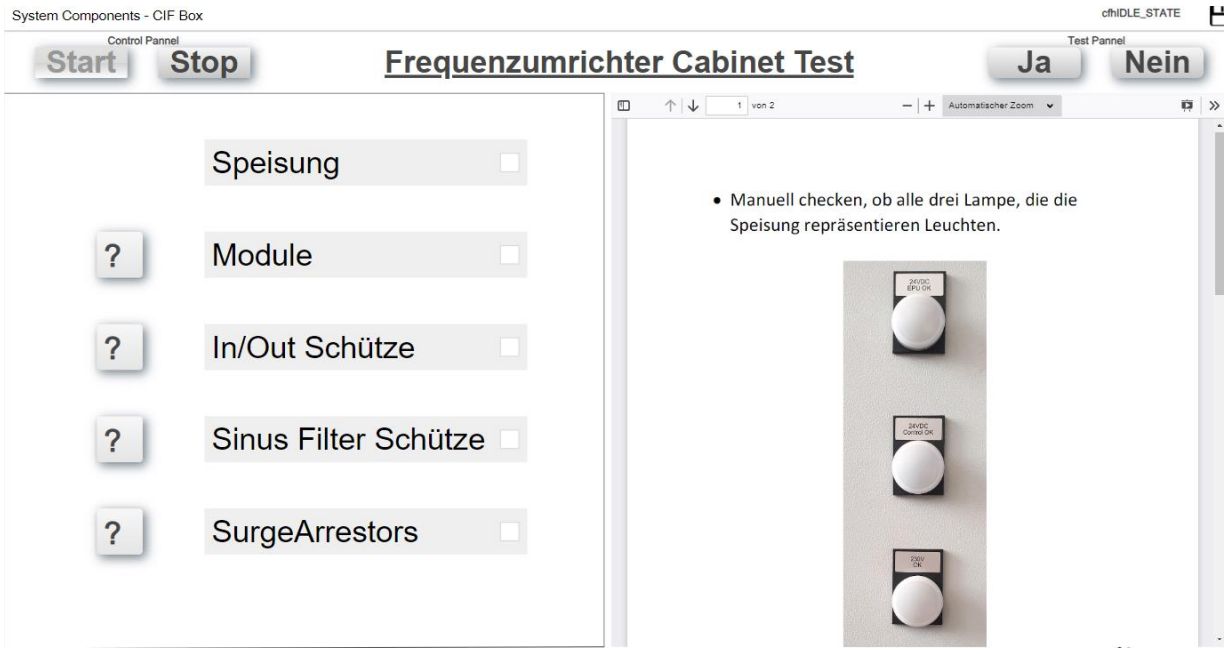
iv. Durch die Einstellung von JaB auf TRUE wird der "Ja"-Knopf aktiviert. Dies ermöglicht dem Benutzer, durch dessen Betätigung die Richtigkeit eines Prozessschritts zu bestätigen.

v. Die Variable NeinB wird auf TRUE gesetzt und aktiviert damit den "Nein"-Knopf. Der Benutzer kann nun, falls erforderlich, einen Prozessschritt als fehlerhaft markieren.

6.2.2. Speisungstest

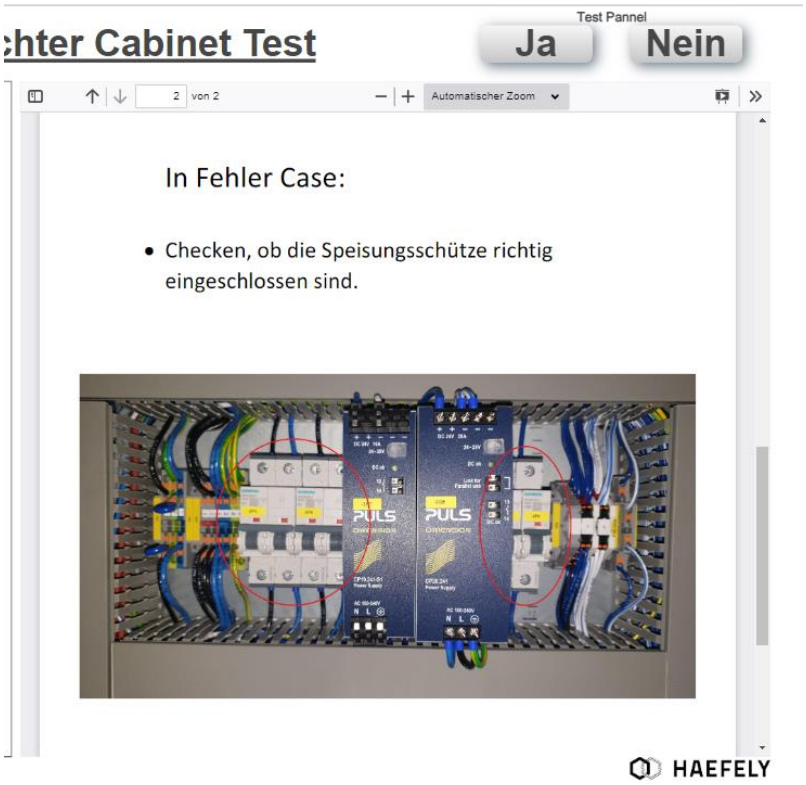
Es wurde bereits erläutert, weshalb einige Tests lediglich manuell durchgeführt werden können. Von den fünf vorhandenen Tests ist der Speisungstest ein manueller Test. Die vorgeschlagene Lösung besteht darin, manuell zu überprüfen, ob die Speisungslampen leuchten. Wenn die Lampen leuchten, ist alles in Ordnung. Leuchten sie nicht, liegt ein Fehler vor.

Während der Speisungstest durchgeführt wird, präsentiert das GUI dem Nutzer ein PDF. Dieses Dokument gibt Anweisungen, die Lampen der Speisung zu überprüfen:



Speisungstest Anleitung

Im gleichen PDF gibt es auch einen Tipp, was man machen kann, um den Fehler zu korrigieren fast es einen Fehler gibt:



Lösungsmöglichkeit in Fall, dass es einen Fehler beim Speisungstest gibt

Implementierung des Speisungstests im Hauptprogramm:

```
// (Anfang) Laufen Test

    CASE TestN OF
        0:
            PDF := 'Media\PDF\TestInfoPDF\FcCabinetTest.pdf';
            StartB := TRUE;
            StopB := FALSE;
            JaB := FALSE;
            NeinB := FALSE;
            Speisung:= FALSE;
            SpeisungFehler:= FALSE;
            HwDetect.ModuleTestEndeC:=FALSE;
            ContactorBausteinSinus.ContactorTestEndeC:=FALSE;
            ContactorBausteinSchutze.ContactorTestEndeC:=FALSE;
            SurgeArrestorsBaustein.SurgeArrestorsTestEndeC:=FALSE;
        1:
            HwDetect.ModuleTestEndeC:=TRUE;
            ContactorBausteinSinus.ContactorTestEndeC:=TRUE;
            ContactorBausteinSchutze.ContactorTestEndeC:=TRUE;
            SurgeArrestorsBaustein.SurgeArrestorsTestEndeC:=TRUE;
            PDF := 'Media\PDF\SpeisungPage.pdf';
            IF TestNP THEN
                Speisung:= TRUE;
                TestN:=2;
            END_IF

            IF FehlerInfo THEN
                SpeisungFehler:= TRUE;
                TestN:=2;
            END IF
```

Speisungstest im Hauptprogramm

Wie bereits erwähnt, wird der Speisungstest im Case 1 des Systems durchgeführt, sobald die Variable TestN den Wert 1 annimmt. Die Bezeichnung in rosa Buchstaben gibt an, welches PDF aktuell angezeigt wird. In diesem Fall handelt es sich um die Anleitung für den Speisungstest, wie im Bild „Speisungstest Anleitung“ dargestellt.

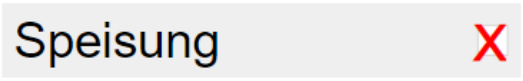
Es gibt zwei Hauptvariablen, die im Zusammenhang mit dem Nutzerfeedback stehen:

1. TestNP (no Problem): Diese wird auf TRUE gesetzt, wenn der Benutzer den „Ja“-Knopf betätigt. Dies signalisiert, dass der Test erfolgreich war.
2. FehlerInfo: Wird auf TRUE gesetzt, sobald der „Nein“-Knopf gedrückt wird, was auf einen Fehler während des Tests hindeutet.

Wenn TestNP den Wert TRUE annimmt, wird die Variable Speisung aktiviert. Ist dies der Fall, wird das grüne Häkchen im Bereich des Speisungstests sichtbar, was den erfolgreichen Abschluss dieses Testabschnitts signalisiert.

Speisungstest richtig

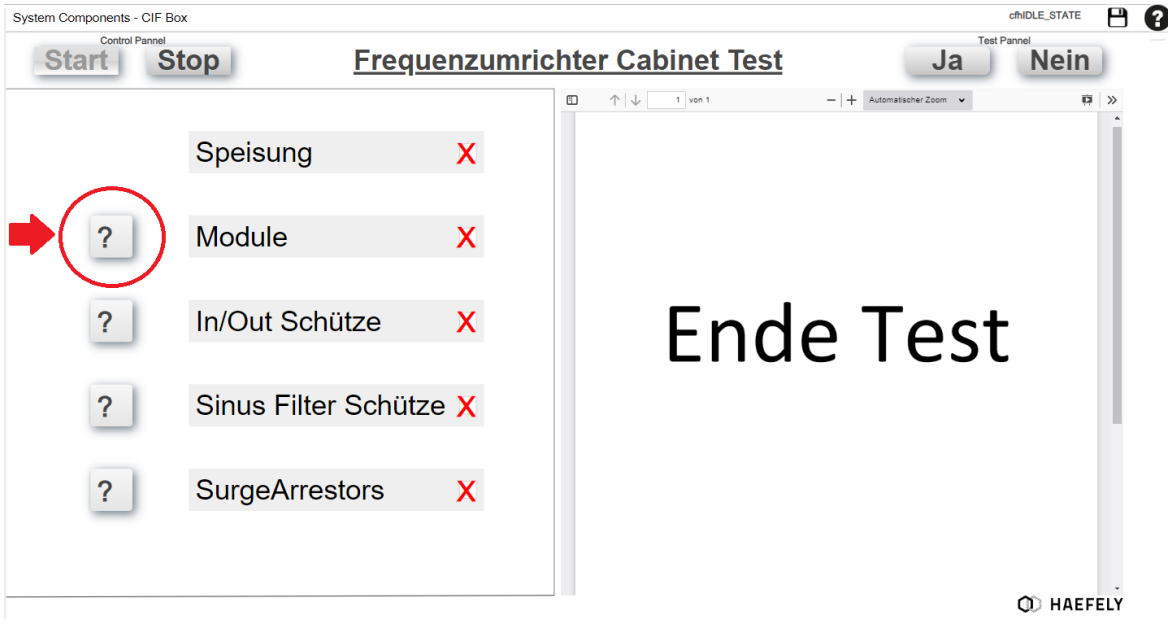
Die Variable FehlerInfo steht im Zusammenhang mit der Fehlererkennung. Wenn sie den Wert TRUE annimmt, was bedeutet, dass ein Fehler während des Speisungstests festgestellt wurde, wird die Variable SpeisungFehler aktiviert. In der Folge wird im GUI der rote X beim Speisungstest sichtbar, um auf den festgestellten Fehler hinzuweisen. Bei einem Wert von FALSE für FehlerInfo bleibt der rote X unsichtbar, da kein Fehler vorliegt:



Speisungstest falsch

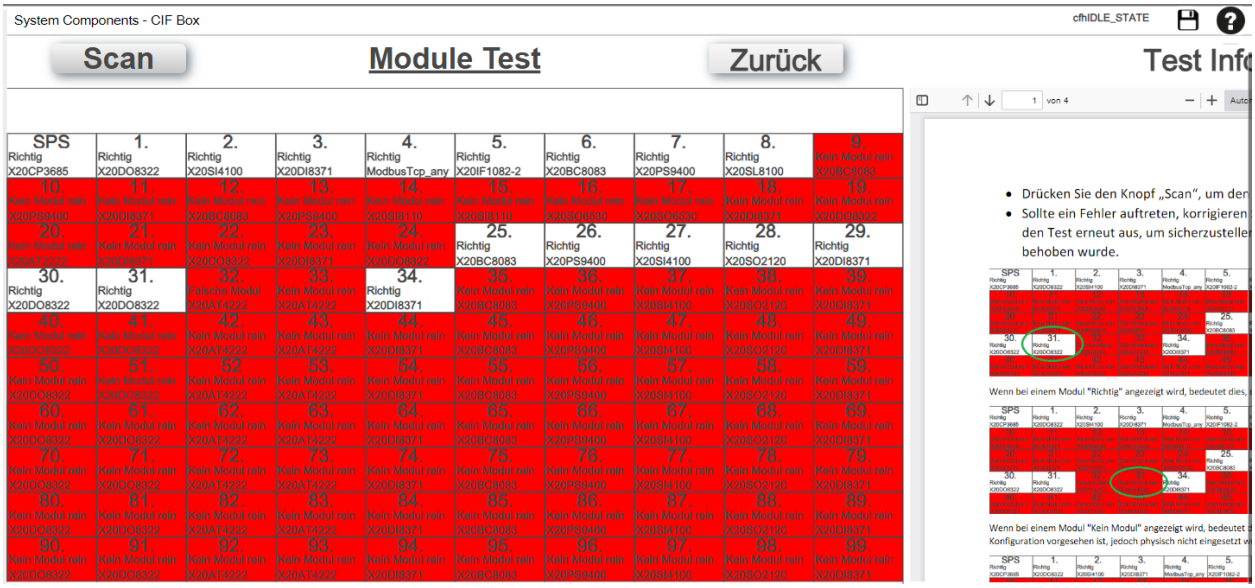
6.2.3. Module Test

Während des Module Tests, wenn ein Fehler auftritt, kann der Nutzer die ModuleTestPage öffnen. Dort wird detailliert angezeigt, wo genau das Problem lokalisiert ist:



ModuleTestPage öffnen

So sieht der ModuleTestPage aus:



ModuleTestPage

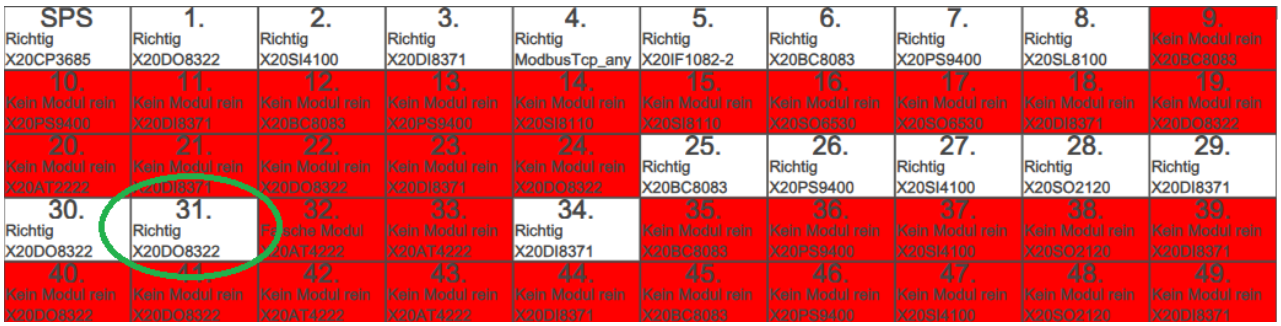
Im Modul-Test gibt es einen „Scan“-Knopf, den der Nutzer betätigen kann, um zu ermitteln, wo das Problem liegt. Nach einer Korrektur kann dieser Knopf erneut betätigt werden, um zu überprüfen, ob das Problem behoben wurde.

Ein weiterer Knopf, beschriftet mit „Zurück“, dient als Navigationselement und führt den Nutzer zurück zur Hauptseite.

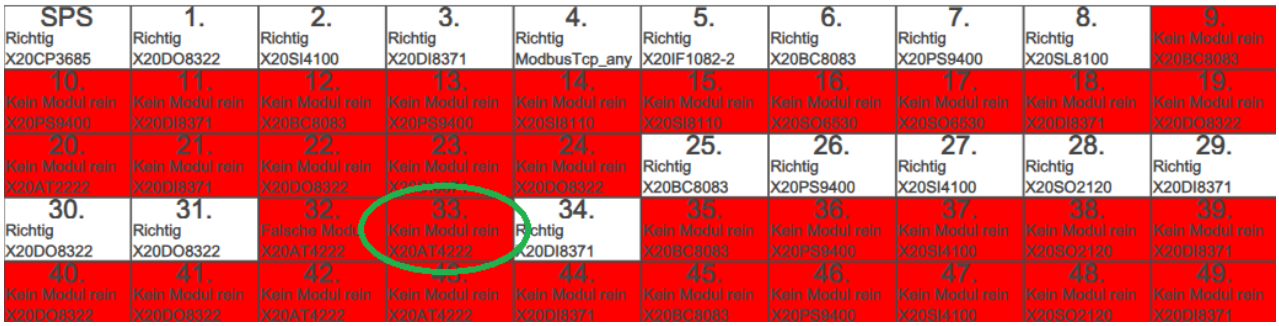
Die dargestellten Rechtecke repräsentieren die in der Software konfigurierten B&R-Module. Die auf den Rechtecken angezeigte Zahl gibt an, an welcher Stelle das jeweilige Modul konfiguriert ist. Die Bezeichnung wie „X20...“ zeigt den Modultyp an, der an dieser Position eingesetzt werden soll. Ist das Rechteck weiß und zeigt den Text „Richtig“, so ist alles korrekt konfiguriert. Ein rotes Rechteck mit dem Hinweis „Fehler“ signalisiert, dass hier ein Problem besteht und korrigierende Maßnahmen erforderlich sind.

Das PDF auf der rechten Seite dient als Anleitung für den Nutzer. Es erläutert die einzelnen Schritte und gibt Hinweise zur Problemlösung, es sieht so aus:

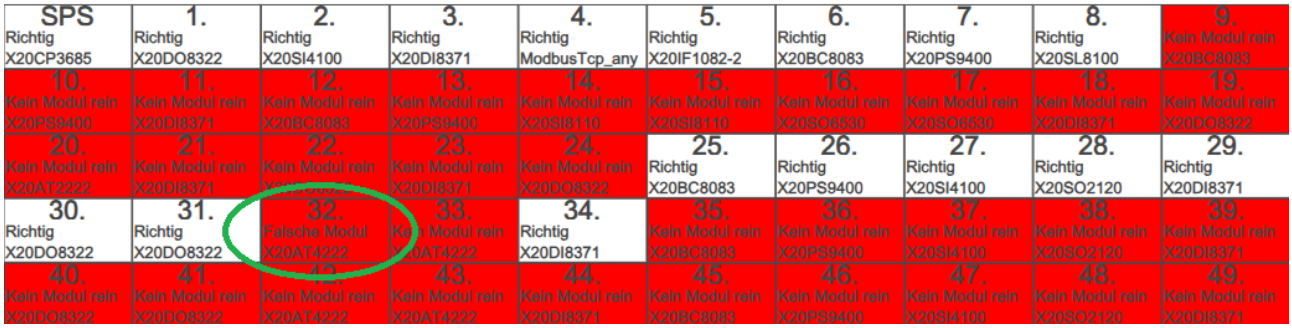
- Drücken Sie den Knopf „Scan“, um den Test auszuführen.
- Sollte ein Fehler auftreten, korrigieren Sie diesen und führen Sie den Test erneut aus, um sicherzustellen, dass das Problem behoben wurde.



Wenn bei einem Modul "Richtig" angezeigt wird, bedeutet dies, dass es korrekt platziert wurde.



Wenn bei einem Modul "Kein Modul" angezeigt wird, bedeutet dies, dass das Modul zwar in der Konfiguration vorgesehen ist, jedoch physisch nicht eingesetzt wurde.

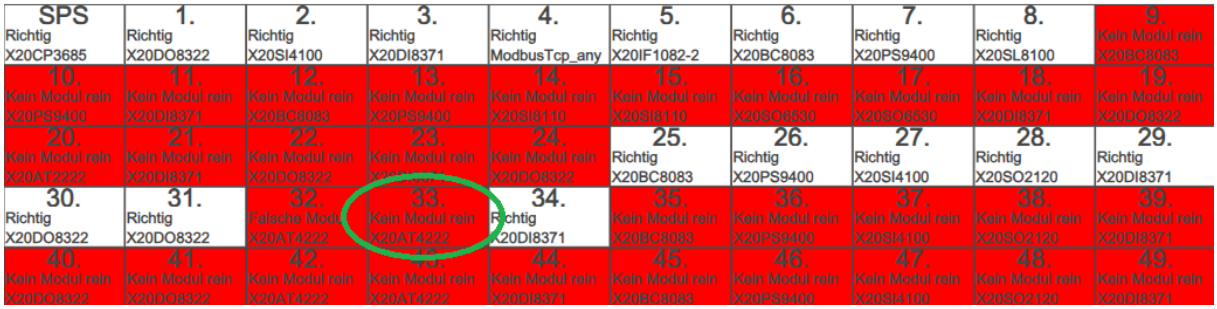


Wenn bei einem Modul "Falsches Modul" angezeigt wird, bedeutet dies, dass an dieser Stelle ein anderes Modul eingesetzt wurde als in der Konfiguration vorgesehen.

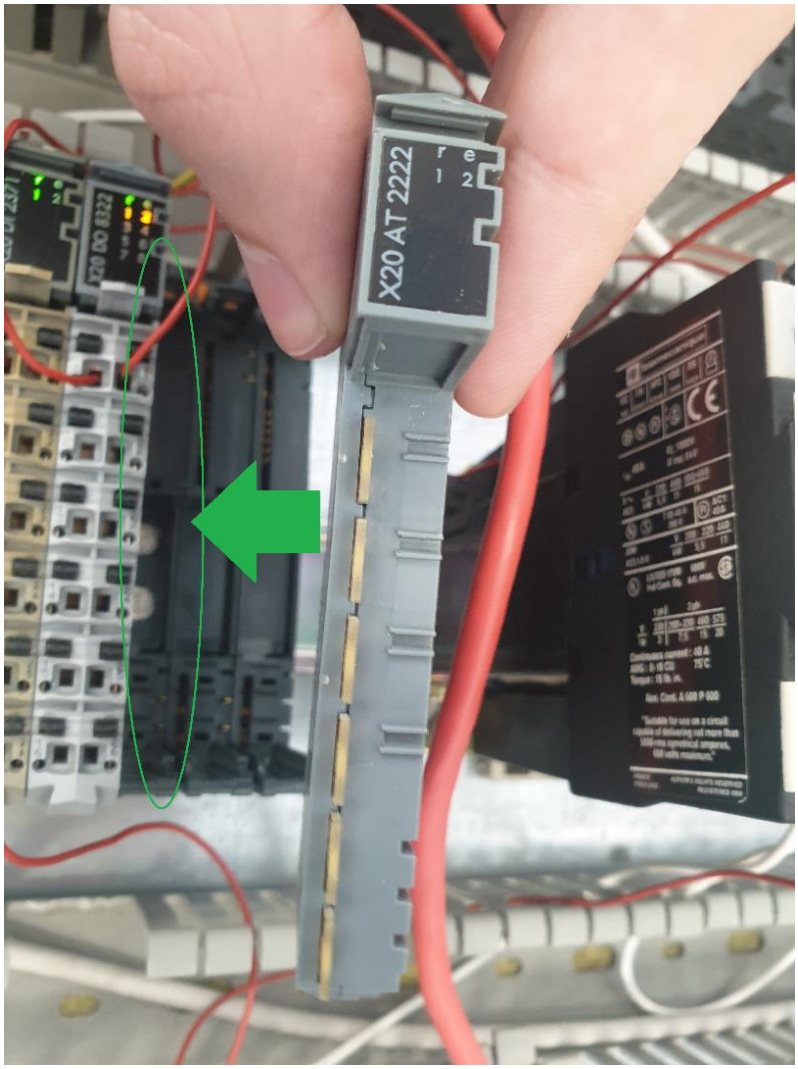
Wenn ein Modulfeld leer ist, bedeutet dies, dass es in der Konfiguration nicht vorgesehen ist und daher ignoriert werden kann.

Im Fehlerfall:

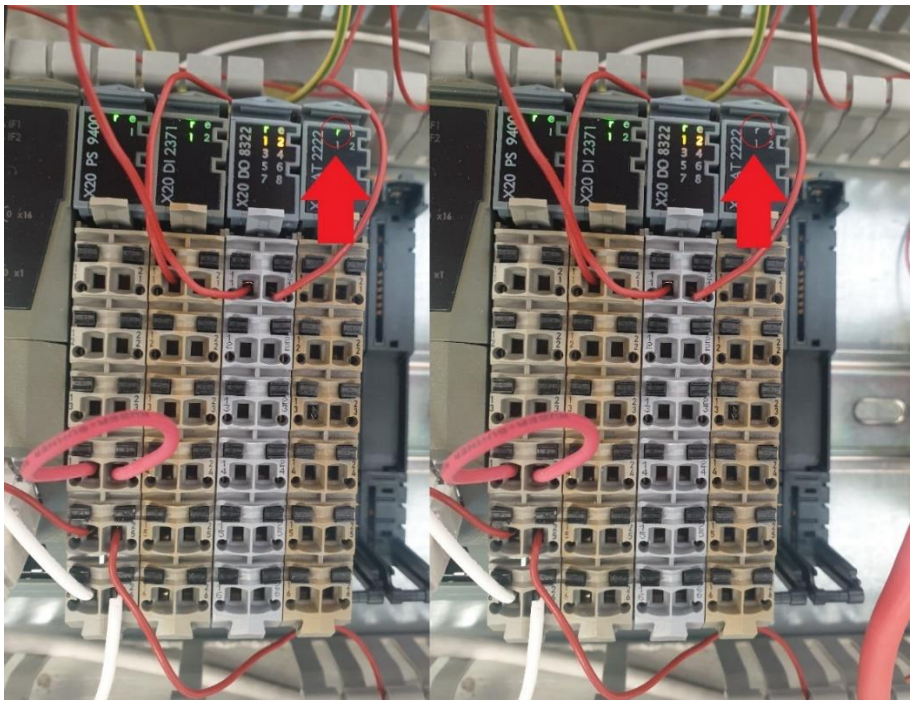
- Bei der Meldung «Kein Modul rein»: Überprüfen Sie, 1) ob das entsprechend konfigurierte Modul korrekt eingesetzt wurde und 2) ob das Zeichen «r» auf dem Modul stabil grün leuchtet.



Wenn bei einem Modul die Anzeige «Kein Modul rein» erscheint, bedeutet dies, dass das Modul zwar in der Konfiguration vorhanden ist, jedoch physisch nicht eingesetzt wurde.



1) Sollte das Modul in der Konfiguration aufgeführt sein, aber physisch nicht eingesetzt sein, setzen Sie es an der entsprechenden Stelle korrekt ein.



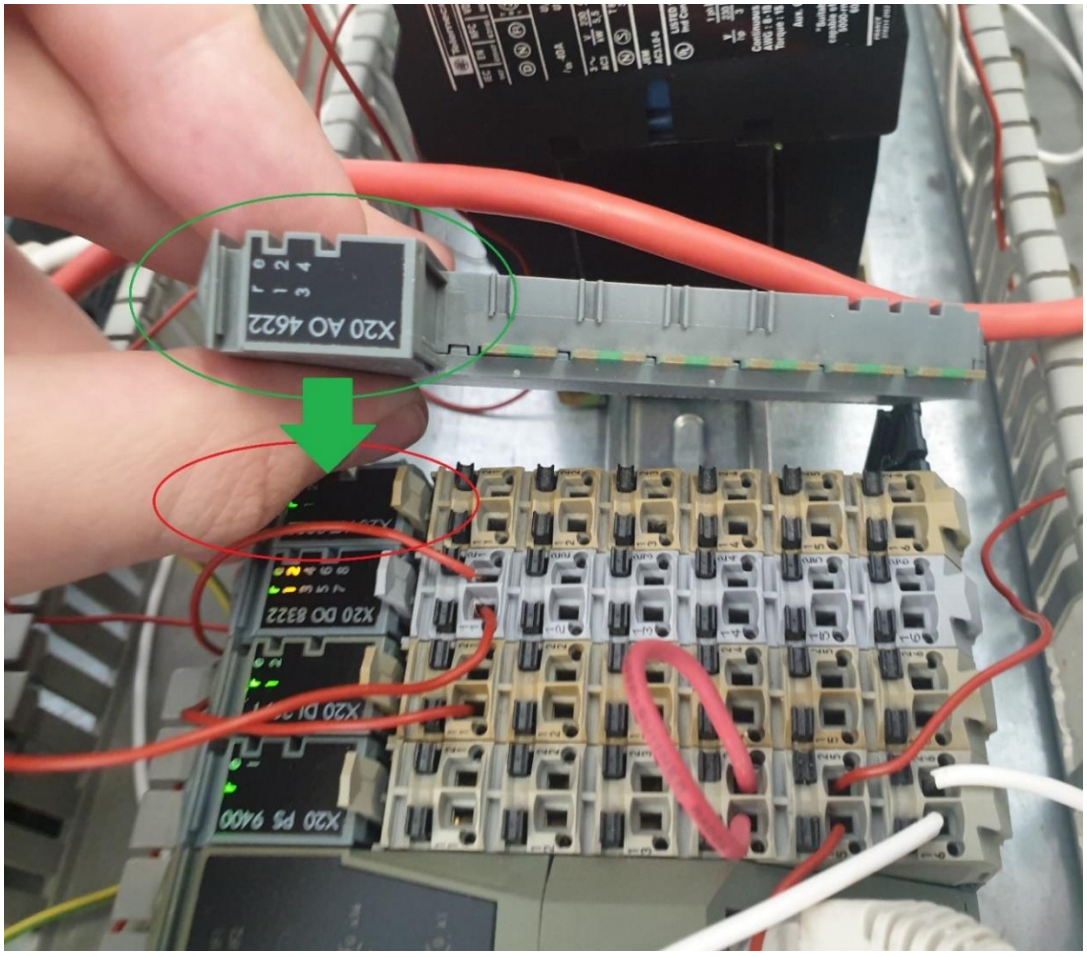
2) Wenn der Buchstabe „r“ bei diesem Modul, anstatt konstant zu leuchten, blinkt, rot aufleuchtet oder gar nicht sichtbar ist, könnte dies darauf hinweisen, dass mit dem Modul etwas nicht in Ordnung ist. In diesem Fall wäre es ratsam, das betreffende Modul durch ein neues zu ersetzen.

*Es könnte vorkommen, dass durch eine Verwechslung das Modul an einer gänzlich falschen Position eingesetzt wurde. Überprüfen Sie daher doppelt, ob es an der vorgesehenen Stelle platziert ist, entsprechend seiner Konfiguration. Ein Beispiel dafür könnte sein, dass ein anderes Modul, welches in der Konfiguration festgelegt wurde, tatsächlich in der Praxis fehlt und vor dem aktuellen Modul platziert werden sollte.

Im Fall von «Falsches Modul»: Überprüfen Sie, welches Modul eingesetzt wurde, entfernen Sie es und setzen Sie das korrekte Modul ein.

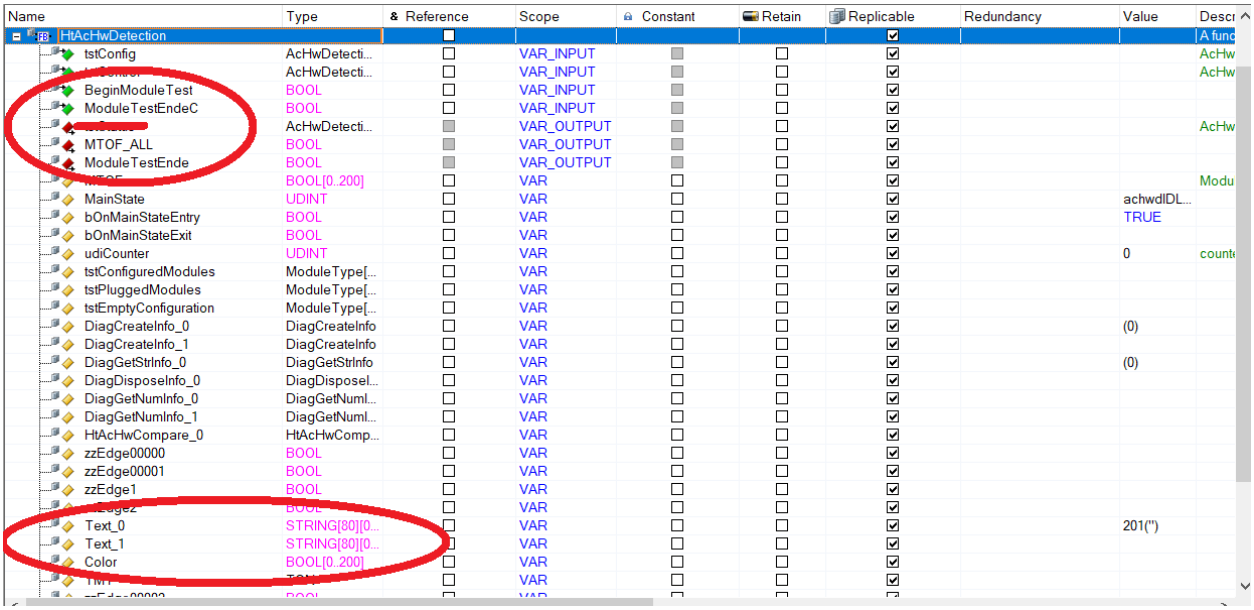
SPS Richtig X20CP3685	1. Richtig X20DO8322	2. Richtig X20SI4100	3. Richtig X20DI8371	4. Richtig ModbusTcp_any	5. Richtig X20IF1082-2	6. Richtig X20BC8083	7. Richtig X20PS9400	8. Richtig X20SL8100	9. Kein Modul rein X20BC8083
10. Kein Modul rein X20PS9400	11. Kein Modul rein X20DI8371	12. Kein Modul rein X20BC8083	13. Kein Modul rein X20PS9400	14. Kein Modul rein X20SI8110	15. Kein Modul rein X20SI8110	16. Kein Modul rein X20SO6530	17. Kein Modul rein X20SO6530	18. Kein Modul rein X20DI8371	19. Kein Modul rein X20DO8322
20. Kein Modul rein X20AT2222	21. Kein Modul rein X20DI8371	22. Kein Modul rein X20[illegible]	23. Kein Modul rein X20DI8371	24. Kein Modul rein X20DO8322	25. Richtig X20BC8083	26. Richtig X20PS9400	27. Richtig X20SI4100	28. Richtig X20SO2120	29. Richtig X20DI8371
30. Richtig X20DO8322	31. Richtig X20DO8322	32. Falsche Modul X20AT4222	33. Kein Modul rein X20AT4222	34. Richtig X20DI8371	35. Kein Modul rein X20BC8083	36. Kein Modul rein X20PS9400	37. Kein Modul rein X20SI4100	38. Kein Modul rein X20SO2120	39. Kein Modul rein X20DI8371
40. Kein Modul rein X20DO8322	41. Kein Modul rein X20DO8322	42. Kein Modul rein X20AT4222	43. Kein Modul rein X20AT4222	44. Kein Modul rein X20DI8371	45. Kein Modul rein X20BC8083	46. Kein Modul rein X20PS9400	47. Kein Modul rein X20SI4100	48. Kein Modul rein X20SO2120	49. Kein Modul rein X20DI8371

Wenn ein Modul den Status «Falsches Modul» anzeigt, bedeutet dies, dass an dieser Position ein anderes Modul eingesetzt wurde als vorgesehen.



Wie im Kapitel zum Beispielprojekt beschrieben, gab es bereits einen Funktionsblock von Haefely AG, der überprüft, ob ein konfiguriertes Modul auch an seinem vorgesehenen Platz eingesteckt ist. Und wie bereits erwähnt, wurden kleine Anpassungen vorgenommen.:

Wie Variablen und Programmtext beim Funktionsblock zu addieren:



Module Detect Variablen

Die addierten Variablen sind:

- BeginModuleTest: Diese Variable aktiviert den Module Test, sobald das System im Zustand „TestN = 2“ ist.

```
IF BeginModuleTest THEN
    tstControl.bReqHwDetection:=FALSE;
    TMT.IN:=TRUE;
    TMT.PT:=UINT_TO_TIME(1000);
END_IF
```

- ModuleTestEnde: Dies ist eine Variable, die dem Hauptprogramm signalisiert, dass der Module Test abgeschlossen ist. Das System kann dann zum nächsten Zustand übergehen, wobei "TestN" den Wert 3 annimmt.

```
FOR udiCounter := DiagCreateInfo_0.nrEntries TO DiagCreateInfo_0.nrEntries DO
    ModuleTestEnde :=TRUE;
END_FOR
```

- ModuleTestEndeC: Diese Variable wird verwendet, um die Variable „ModuleTestEnde“ zurückzusetzen, jedes Mal wenn der "Stop"-Knopf gedrückt wird. Dies geschieht, damit bei einem erneuten Start des Tests, „ModuleTestEnde“ den Test normal durchlaufen lässt, anstatt sofort zum nächsten Prüfschritt zu wechseln.

```
IF ModuleTestEndeC=FALSE THEN
    ModuleTestEnde:=FALSE;
END_IF
```

- MTOF_ALL: Wenn diese Variable den Wert TRUE hat, bedeutet dies, dass alle konfigurierten B&R Module korrekt eingesteckt sind.

```
MTOF_ALL := TRUE;

IF MTOF[udiCounter] = FALSE THEN
    MTOF_ALL := FALSE;
END_IF
```

- Text_0, Text_1 und Color: Diese Variablen beeinflussen das GUI visuell. „Text_0“ zeigt an, ob ein konfiguriertes Modul korrekt eingesteckt ist. „Text_1“ gibt den Modultyp an. Die Variable „Color“ aktiviert die rote Farbe für ein Rechteck, falls bei dem entsprechenden Modul ein Fehler vorliegt. „udi Counter“ gibt an, welches konfigurierte Modul gemeint ist.

```
CASE tstConfiguredModules[udiCounter].uiModuleState OF
    0:
        Text_0[udiCounter] := '';
        Text_1[udiCounter] := '';
        MTOF[udiCounter] := TRUE;
    1:
        Text_0[udiCounter] := 'Kein Modul rein';
        Text_1[udiCounter] := tstConfiguredModules[udiCounter].strModuleType;
        Color[udiCounter] := TRUE;
        MTOF[udiCounter] := FALSE;
    2:
        Text_0[udiCounter] := 'Extra Modul';
        Text_1[udiCounter] := tstConfiguredModules[udiCounter].strModuleType;
        Color[udiCounter] := TRUE;
        MTOF[udiCounter] := FALSE;
    3:
        Text_0[udiCounter] := 'Richtig';
        Text_1[udiCounter] := tstConfiguredModules[udiCounter].strModuleType;
        Color[udiCounter] := FALSE;
        MTOF[udiCounter] := TRUE;
    4:
        Text_0[udiCounter] := 'Falsche Modul';
        Text_1[udiCounter] := tstConfiguredModules[udiCounter].strModuleType;
        Color[udiCounter] := TRUE;
        MTOF[udiCounter] := FALSE;
END_CASE
```

Die im Hauptprogramm vorgenommenen Ergänzungen stehen in direktem Zusammenhang mit den Variablen des "Module Detect"-Funktionsblocks:

- Den Funktionsblock in Hauptprogramm anrufen.

```
PROGRAM _CYCLIC

    HwDetect();
```

- Wenn das System den Case TestN=2 erreicht, aktiviert die Variable BeginModuleTest den Module Test. Sobald der Test abgeschlossen ist, führt die Variable ModuleTestEnde das System zum nächsten Case (TestN=3).

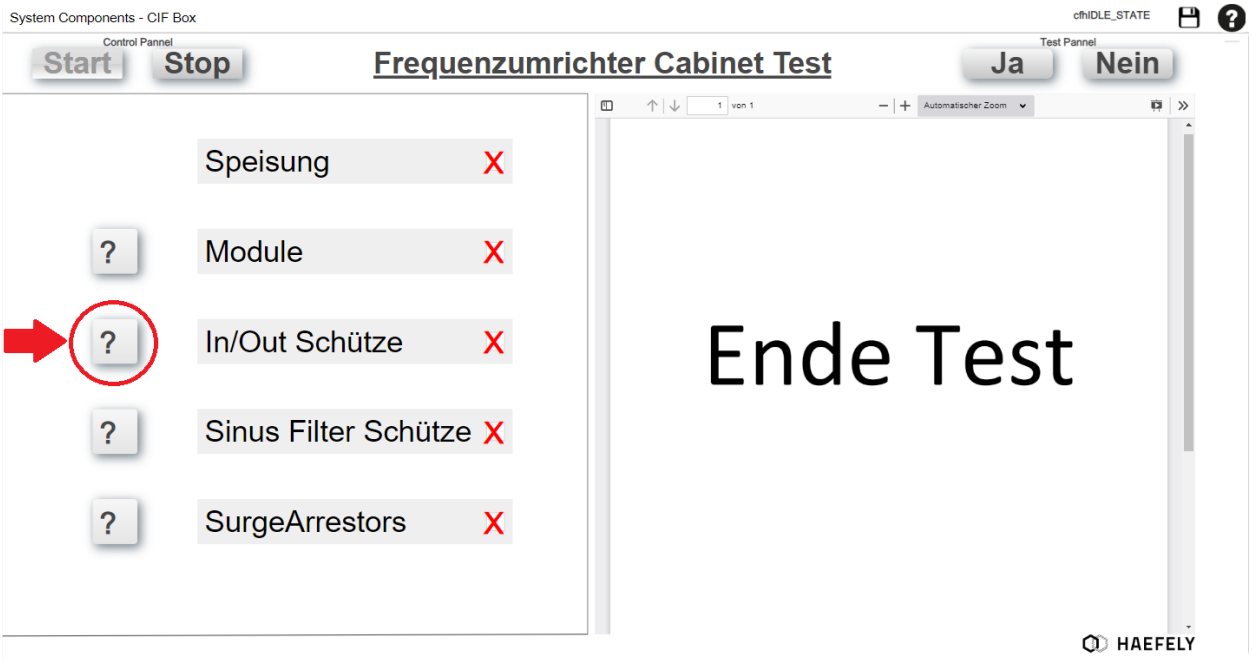
```
2:
    TestNP:= FALSE;
    FehlerInfo:=FALSE;

    PDF := 'Media\PDF\LaufendesTest\ModuleTestLauft.pdf';

    HwDetect.BeginModuleTest:=TRUE;

    IF HwDetect.ModuleTestEnde THEN
        TestN:=3;
    END_IF
```

Wenn alle konfigurierten Module korrekt eingesteckt sind, wird das grüne Häkchen aktiviert, ansonsten nicht (Variable ModuleTest). Wenn eines oder mehrere konfigurierte Module falsch eingesteckt sind, wird das rote X aktiviert, ansonsten nicht (Variable ModuleTestFehler).

```
//(Anfang)Geprüfte Tests

    IF TestN > 2 AND HwDetect.MTOF_ALL THEN
        ModuleTest := TRUE;
    ELSE    ModuleTest := FALSE;
    END_IF
    IF TestN > 2 AND NOT HwDetect.MTOF_ALL THEN
        ModuleTestFehler := TRUE;
    ELSE    ModuleTestFehler := FALSE;
    END_IF
```

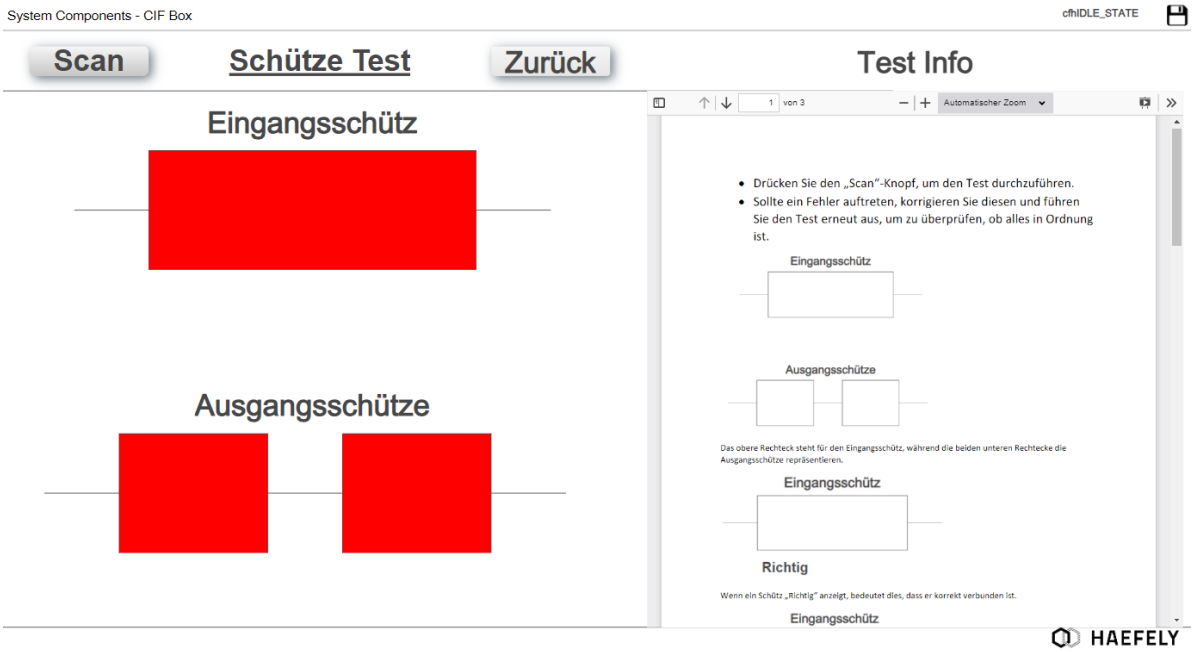
6.2.4. Ein-Ausgangsschütze Test (IN/OUT Schütze)

Bei Fehlern im Ein-/Ausgangsschütze-Test kann der Benutzer die Seite "INOUTSchützePage" öffnen, um zu ermitteln, wo das Problem liegt.:



INOUTSchützePage öffnen

So sieht der INOUTSchützePage aus:



INOUTSchützePage

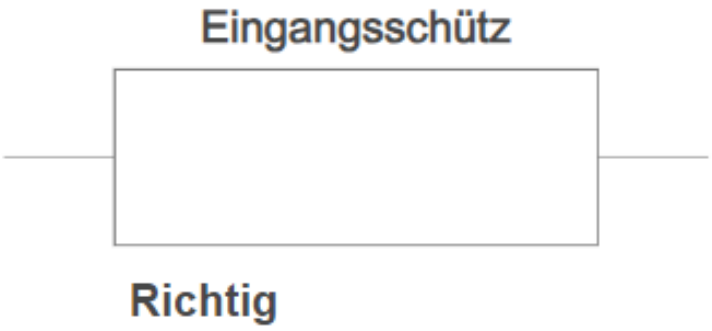
Der "Scan"-Knopf kann gedrückt werden, um festzustellen, wo das Problem liegt. Nach einer Korrektur kann er erneut gedrückt werden, um zu überprüfen, ob das Problem behoben wurde.

Mit dem "Zurück"-Knopf kann der Benutzer zur Hauptseite zurückkehren.

Das obere Rechteck repräsentiert den Eingangsschütz und die beiden unteren die Ausgangsschütze. Ein weißes Rechteck, das "Richtig" anzeigt, bedeutet, dass alles in Ordnung ist. Ein rotes Rechteck mit der Aufschrift "Fehler" weist auf ein Problem hin, das korrigiert werden muss. Das PDF auf der rechten Seite dient als Anleitung für den Benutzer. Es erläutert die verschiedenen Elemente und bietet Lösungsvorschläge, es sieht so aus:

- Drücken Sie den „Scan"-Knopf, um den Test durchzuführen.
- Sollte ein Fehler auftreten, korrigieren Sie diesen und führen Sie den Test erneut aus, um zu überprüfen, ob alles in Ordnung ist.

Das obere Rechteck steht für den Eingangsschütz, während die beiden unteren Rechtecke die Ausgangsschütze repräsentieren.



Wenn ein Schütz „Richtig“ anzeigt, bedeutet dies, dass er korrekt verbunden ist.

Wenn ein Schütz „Fehler“ anzeigt, bedeutet dies, dass er nicht korrekt verbunden ist.

Im Fehlerfall:

- Wenn der Fehler beim Eingangsschütz liegt, überprüfen Sie, ob der Eingangsschütz 49K6 korrekt verbunden ist.

- Wenn der Fehler bei den Ausgangsschützen liegt, überprüfen Sie, ob die Ausgangsschützen 50K0 und 51K0 korrekt verbunden sind.

Der für den IN/OUT Schütze Test verwendete Funktionsblock war derselbe wie für den Sinus Filter Schütze, da beide Funktionen identisch waren (Contactor Funktionsblock). Dieser Funktionsblock wurde von Haefely AG bereitgestellt. Die zusätzlich implementierten Erweiterungen sind wie folgt:

Variablen und Programmtext, die dem Funktionsblock hinzugefügt wurden:

ContactorTest		☐					☑
tstConfig	ContactorTest...	☐	VAR_INPUT		☐		☑
tstSensor	ContactorTest...	☐	VAR_INPUT		☐		☑
bStart	BOOL	☐	VAR_INPUT		☐		☑
ContactorTestEndeC	BOOL	☐	VAR_INPUT		☐		☑
tstActor	ContactorTest...	☐	VAR_OUTPUT		☐		☑
tstStatus	ContactorTest...	☐	VAR_OUTPUT		☐		☑
ContactorTestEnde	BOOL	☐	VAR_OUTPUT		☐		☑
TON_Contactor01	TON	☐	VAR		☐		☑
MainState	ContactorTest...	☐	VAR		☐		☑
TON_Contactor02	TON	☐	VAR		☐		☑
TD	TON	☐	VAR		☐		☑
bOnMainStateEntry	BOOL	☐	VAR		☐		☑
bOnMainStateExit	BOOL	☐	VAR		☐		☑
Color1	BOOL	☐	VAR		☐		☑
Color2	BOOL	☐	VAR		☐		☑
Color3	BOOL	☐	VAR		☐		☑
zzEdge00000	BOOL	☐	VAR		☐		☑

Contactor Variablen

Die addierten Variablen sind:

- ContactorTestEnde: Diese Variable informiert das Hauptprogramm darüber, dass der Contactor Test abgeschlossen ist und das System zum nächsten Case übergehen kann.

```
ELSIF tstConfig.bSecContactorExist THEN
    tstStatus.bContactor02aIsOk := FALSE;
    tstStatus.strContactor02aError := 'Fehler';
    tstStatus.bContactor02bIsOk := FALSE;
    tstStatus.strContactor02bError := 'Fehler';
    bOnMainStateExit := TRUE;
    MainState := sftIDLE_STATE;
    ContactorTestEnde:=TRUE;
    bStart:=FALSE;
END_IF
```

- ContactorTestEndeC: Diese Variable wird verwendet, um die Variable ContactorTestEnde jedes Mal zurückzusetzen, wenn der "Stop"-Knopf gedrückt wird.

```
IF ContactorTestEndeC=FALSE THEN
    ContactorTestEnde:=FALSE;
END_IF
```

- Color1, Color2 und Color3: die sind Variablen, die das GUI visuell beeinflussen, die Colors aktivieren die rote Farbe bei den Rechtecken, wenn bei diesem Schütz einen Fehler gibt.

```
Color1:= NOT tstStatus.bContactor01IsOk;
Color2:= NOT tstStatus.bContactor02aIsOk;
Color3:= NOT tstStatus.bContactor02bIsOk;
```

Die Ergänzungen, die im Hauptprogramm vorgenommen wurden, stehen in direktem Zusammenhang mit den Variablen des Contactor-Funktionsblocks:

- Den Funktionsblock in Hauptprogramm anrufen.

```
ContactorBausteinSchutze();
```

- Die Funktion deaktiviert vorübergehend andere Funktionen, damit sie die Pins der Digital Outputs kontrollieren kann, ohne kritische Fehlermeldungen zu erhalten.

```
IF ContactorBausteinSchutze.bStart OR ContactorBausteinSinus.bStart THEN
    ContactorStartAktiviert:=TRUE;
ELSE
    ContactorStartAktiviert:=FALSE;
END_IF
```

```
IF NOT ContactorStartAktiviert THEN
    pvActor.tstComponent.tstSpttsPowerCtlLogic00 := PwCtl01.tstActor;
    TESTEN:=FALSE;
END_IF

IF  ContactorStartAktiviert THEN
    TESTEN:=TRUE;
END_IF
```

```
        IF ContactorBausteinSchutze.tstStatus.enmContactorState <> sftIDLE_STATE  THEN
            pvConfig.tstComponent.tstLvf01.bDefined := FALSE;
            pvConfig.tstComponent.tstSpttsPowerCtlLogic00.bDefined := FALSE;
            TestenContactor:=FALSE;
        ELSE
            pvConfig.tstComponent.tstLvf01.bDefined := TRUE;
            pvConfig.tstComponent.tstSpttsPowerCtlLogic00.bDefined := TRUE;
            TestenContactor:=TRUE;
        END_IF
```

```
IF ContactorBausteinSchutze.bStart THEN
    ContactorBausteinSchutze.tstSensor.bContactor01IsOn := pvSensor.tstComponent.tstSpttsPowerCtlLogic00.tstPowerCtlSensor.tstCabinet[0].bPowerSwitch01IsOk;
    ContactorBausteinSchutze.tstSensor.bContactor02aIsOn := pvSensor.tstComponent.tstSpttsPowerCtlLogic00.tstPowerCtlSensor.tstCabinet[0].bContactor01IsOn;
    ContactorBausteinSchutze.tstSensor.bContactor02bIsOn :=pvSensor.tstComponent.tstSpttsPowerCtlLogic00.tstPowerCtlSensor.tstCabinet[0].bContactor02IsOn;

    pvActor.tstComponent.tstSpttsPowerCtlLogic00.tstPowerCtlActor.bPowerSwitch01Enable := ContactorBausteinSchutze.tstActor.bSetContactor01;
    pvActor.tstComponent.tstSpttsPowerCtlLogic00.tstPowerCtlActor.bContactor01Enable := ContactorBausteinSchutze.tstActor.bSetContactor02a;
    pvActor.tstComponent.tstSpttsPowerCtlLogic00.tstPowerCtlActor.bContactor02Enable := ContactorBausteinSchutze.tstActor.bSetContactor02b;
END_IF
```

- Wenn sich das System in Case TestN = 3 befindet, aktiviert die Variable ContactorBausteinSchutze.bStart den IN/OUT Schütze Test. Sobald der Test abgeschlossen ist, bewegt die Variable ContactorBausteinSchutze.ContactorTestEnde das System zum nächsten Case (TestN=4).

```
3:
    PDF := 'Media\PDF\LaufendesTest\INOUTlauft.pdf';

    ContactorBausteinSchutze.bStart:=TRUE;

    IF ContactorBausteinSchutze.ContactorTestEnde THEN
        TestN:=4;
    END_IF
```

- Wenn alle drei Schütze korrekt arbeiten, wird die Variable ResultSchutze auf TRUE gesetzt.

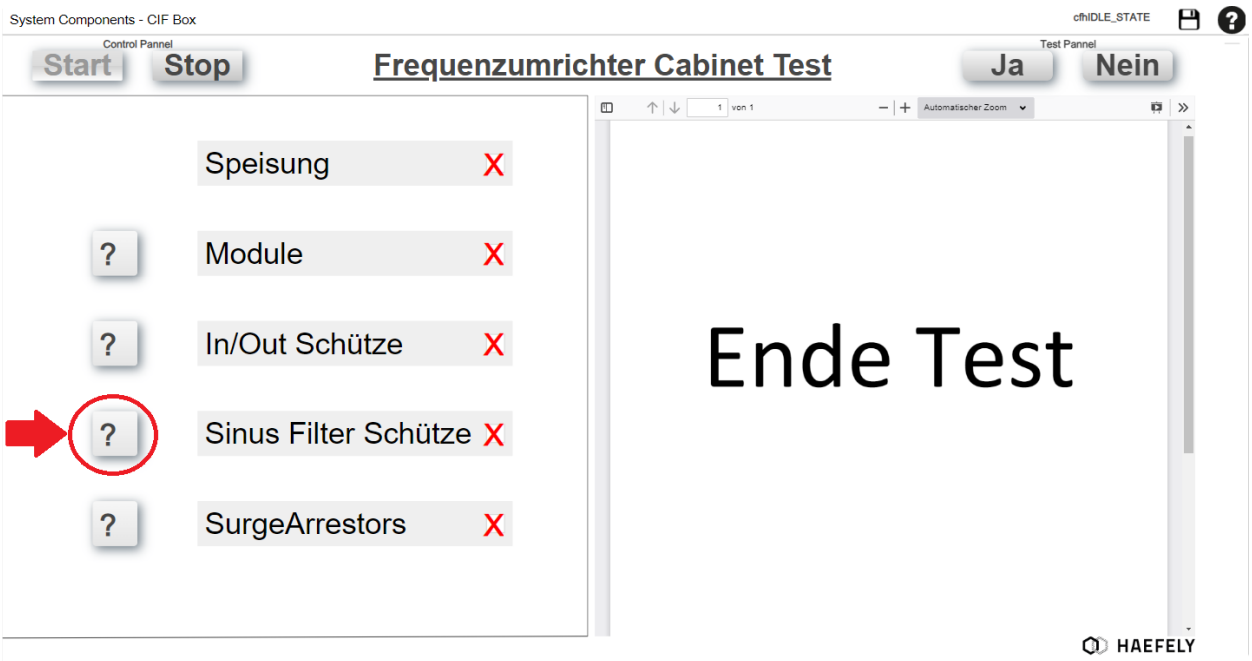
```
ResultSchutze:= ContactorBausteinSchutze.tstStatus.bContactor01IsOk AND
ContactorBausteinSchutze.tstStatus.bContactor02aIsOk AND
ContactorBausteinSchutze.tstStatus.bContactor02bIsOk;
```

- Wenn alle Schütze korrekt arbeiten, wird der grüne Haken aktiviert, andernfalls nicht (Variable InOutSchutze). Wenn einer oder mehrere Schütze nicht ordnungsgemäß funktionieren, wird das rote X aktiviert, ansonsten nicht (Variable InOutSchutzeFehler).

```
IF TestN > 3 AND ResultSchutze THEN
    InOutSchutze := TRUE;
ELSE    InOutSchutze := FALSE;
END_IF
IF TestN > 3 AND NOT ResultSchutze THEN
    InOutSchutzeFehler := TRUE;
ELSE    InOutSchutzeFehler := FALSE;
END_IF
```

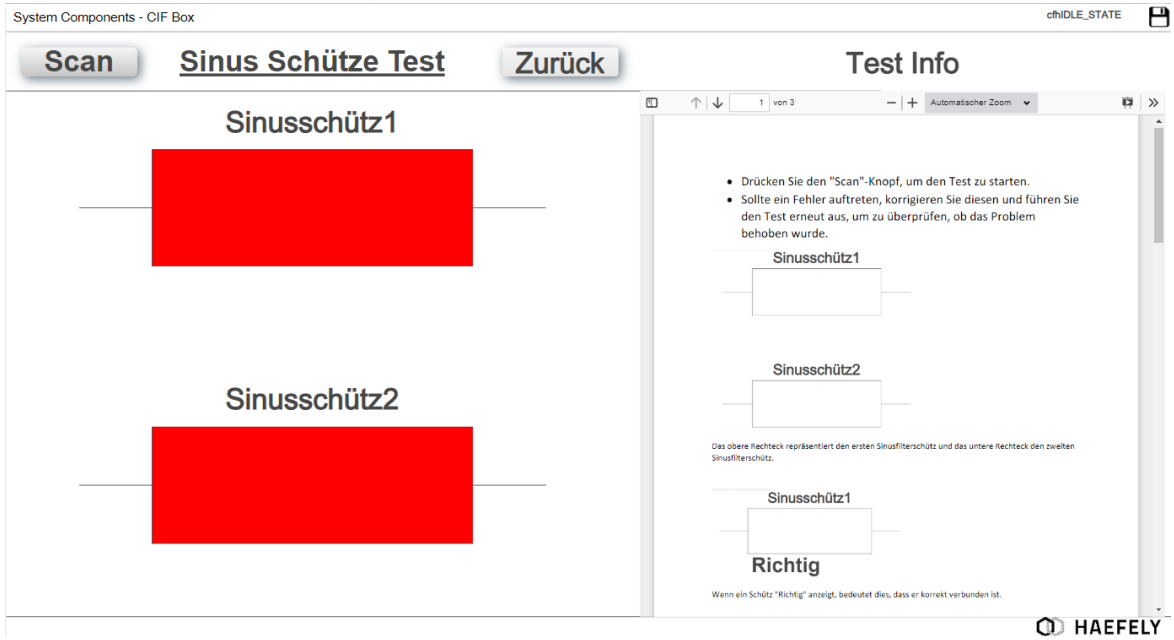
6.2.5. Sinusschütze Test

Beim Sinusfilterschütze-Test kann man bei einem Fehler die Seite "SinusSchützePage" öffnen, um den Fehlerort zu identifizieren:



SinusSchützePage öffnen

So sieht der SinusSchützePage aus:



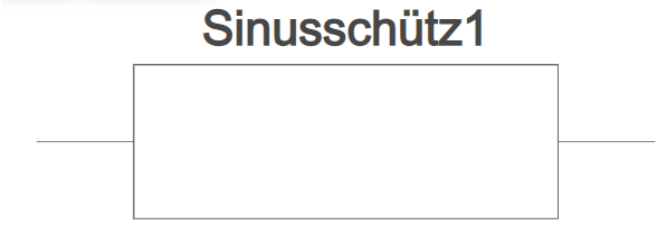
SinusSchützePage

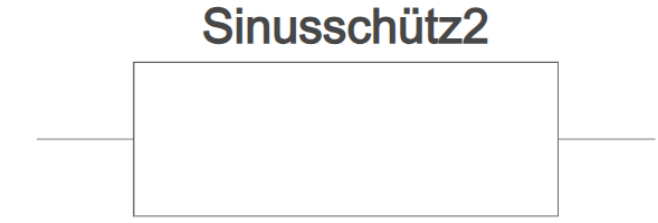
Der "Scan"-Knopf kann betätigt werden, um festzustellen, wo das Problem liegt. Nach einer Korrektur kann er erneut gedrückt werden, um zu überprüfen, ob das Problem behoben wurde.

Der "Zurück"-Knopf ist ein Navigationsknopf, der den Nutzer zurück zur Hauptseite führt. Das obere Rechteck symbolisiert den ersten Sinusschütz und das untere den zweiten.

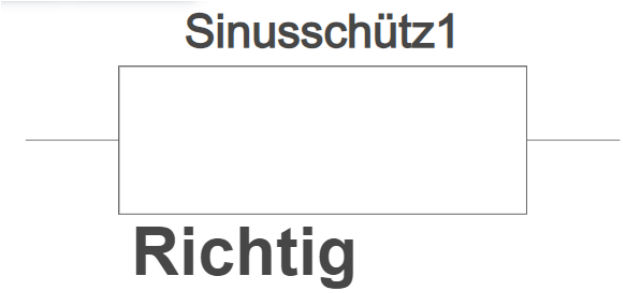
Wenn das Rechteck weiß gefärbt ist und „Richtig“ anzeigt, ist alles in Ordnung. Erscheint das Rechteck jedoch in roter Farbe und zeigt „Fehler“, gibt es ein Problem, das behoben werden muss. Das PDF rechts ist eine Anleitung für den Nutzer. Es erläutert die verschiedenen Funktionen und bietet Lösungsvorschläge. Es sieht so aus:

- Drücken Sie den "Scan"-Knopf, um den Test zu starten.
- Sollte ein Fehler auftreten, korrigieren Sie diesen und führen Sie den Test erneut aus, um zu überprüfen, ob das Problem behoben wurde.

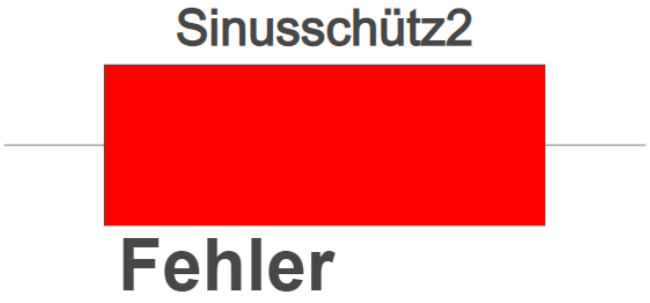




Das obere Rechteck repräsentiert den ersten Sinusfilterschütz und das untere Rechteck den zweiten Sinusfilterschütz.



Wenn ein Schütz "Richtig" anzeigt, bedeutet dies, dass er korrekt verbunden ist.

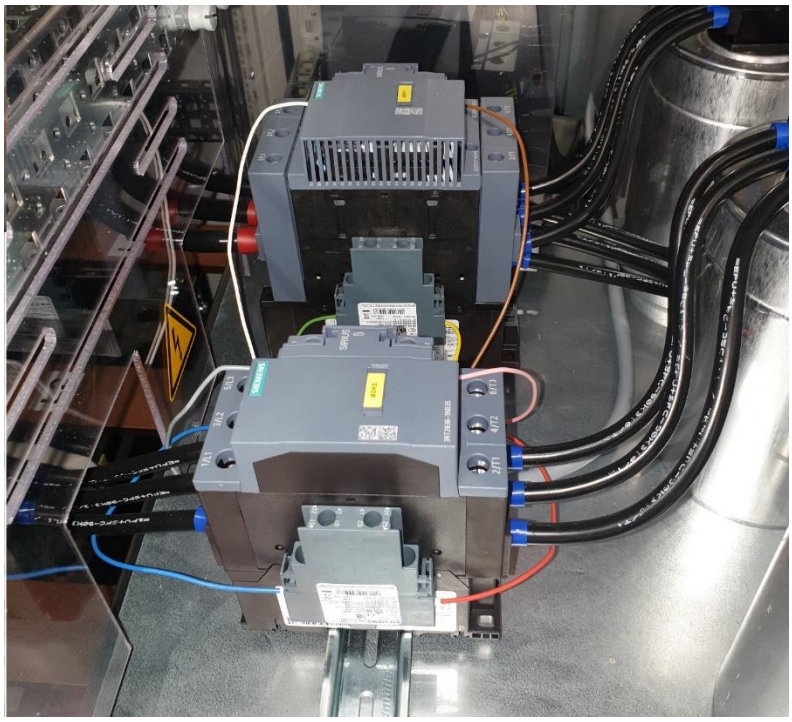


Wenn ein Schütz "Fehler" anzeigt, bedeutet dies, dass er nicht korrekt verbunden ist.

Im Fehlerfall:

- Überprüfen Sie, ob die Lichter der Koppelrelais (26K1) und (26K2) normal leuchten. Wenn sie nicht normal leuchten, liegt möglicherweise ein Problem mit ihrer Verbindung vor.

- Wenn beide Koppelrelais ordnungsgemäß funktionieren und dennoch ein Fehler auftritt, überprüfen Sie, ob die Verbindungen bei den Hauptrelais (90K1) und (90K3) korrekt sind.



Wie bereits erwähnt, wurde für die Sinusschütze der gleiche Funktionsblock verwendet wie beim Ein-Ausgangsschütze Test (IN/OUT Schütze). Dies bedeutet, dass die hinzugefügten Variablen identisch sind.

Die im Hauptprogramm vorgenommenen Ergänzungen stehen in direktem Zusammenhang mit den Variablen des Contactor-Funktionsblocks:

- Den Funktionsblock in Hauptprogramm anrufen.

```
ContactorBausteinSinus();
```

- Die Funktion deaktiviert zeitweise andere Funktionen, damit sie die Pins der Digital Outputs manipulieren kann, ohne kritische Fehler zu verursachen.

```
IF ContactorBausteinSchutze.bStart OR ContactorBausteinSinus.bStart THEN
    ContactorStartAktiviert:=TRUE;
ELSE
    ContactorStartAktiviert:=FALSE;
END_IF


  IF NOT ContactorStartAktiviert THEN
      pvActor.tstComponent.tstSpttsPowerCtlLogic00 := PwCtl01.tstActor;
      TESTEN:=FALSE;
  END_IF

  IF  ContactorStartAktiviert THEN
      TESTEN:=TRUE;
  END_IF
```

```
IF ContactorBausteinSinus.tstStatus.enmContactorState <> sftIDLE_STATE  THEN
    pvConfig.tstComponent.tstLvf01.bDefined := FALSE;
    pvConfig.tstComponent.tstSpttsPowerCtlLogic00.bDefined := FALSE;
    TestenContactor:=FALSE;
ELSE
    pvConfig.tstComponent.tstLvf01.bDefined := TRUE;
    pvConfig.tstComponent.tstSpttsPowerCtlLogic00.bDefined := TRUE;
    TestenContactor:=TRUE;
END_IF
IF ContactorBausteinSinus.bStart THEN
   ContactorBausteinSinus.tstSensor.bContactor01IsOn := pvSensor.tstComponent.tstLvf01.tstSinFilter00.tstThreePhSwitch.tstSwitch_3Ph.bIsClosed;
   ContactorBausteinSinus.tstSensor.bContactor02aIsOn := pvSensor.tstComponent.tstLvf01.tstSinFilter01.tstThreePhSwitch.tstSwitch_3Ph.bIsClosed;

   pvActor.tstComponent.tstLvf01.tstSinFilter00.tstThreePhSwitch.tstSwitch_3Ph.bCmdClose := ContactorBausteinSinus.tstActor.bSetContactor01;
   pvActor.tstComponent.tstLvf01.tstSinFilter01.tstThreePhSwitch.tstSwitch_3Ph.bCmdClose := ContactorBausteinSinus.tstActor.bSetContactor02a;
END_IF
```

- Wenn das System den Case TestN = 4 erreicht, wird durch die Variable ContactorBausteinSchutze.bStart der Sinusschütze-Test aktiviert. Nach Beendigung des Tests führt die Variable ContactorBausteinSinus.ContactorTestEnde das System zum nächsten Case (TestN=5) weiter.

```
4:
    PDF := 'Media\PDF\LaufendesTest\SinusTestLauft.pdf';

    ContactorBausteinSinus.bStart:=TRUE;

    IF ContactorBausteinSinus.ContactorTestEnde THEN
        TestN:=5;
    END_IF
```

- Wenn beide Sinusschütze korrekt arbeiten, wird die Variable ResultSinus auf TRUE gesetzt.

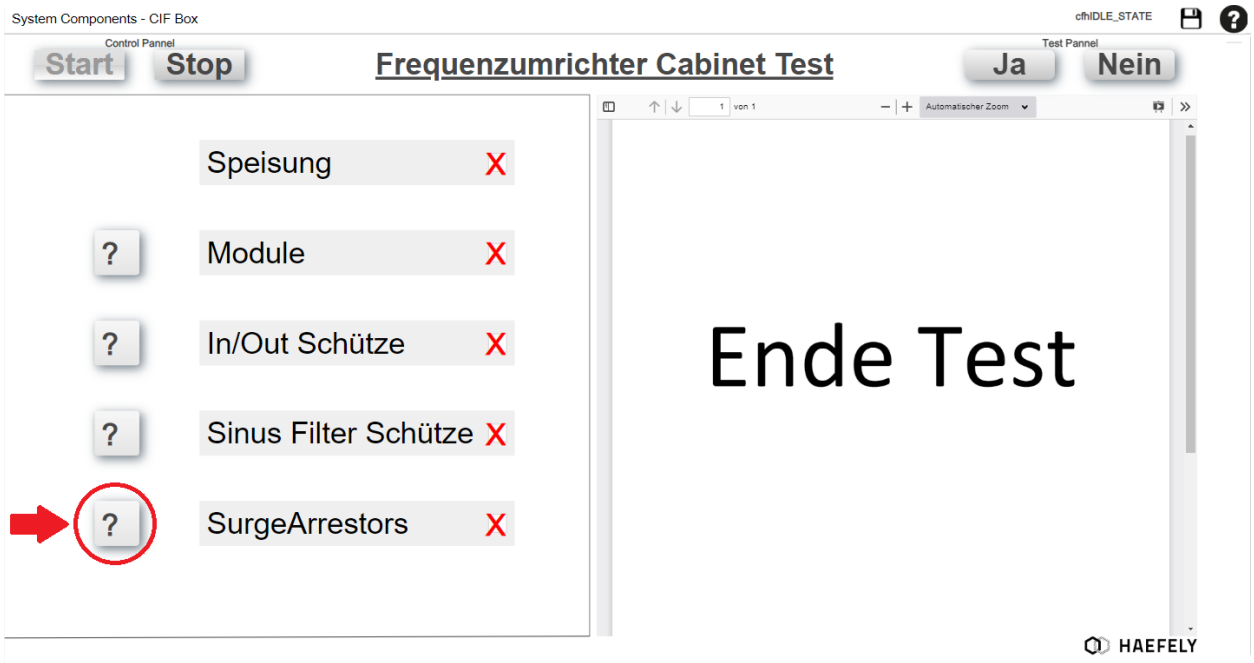
```
ResultSinus:= ContactorBausteinSinus.tstStatus.bContactor01IsOk AND
ContactorBausteinSinus.tstStatus.bContactor02aIsOk;
```

Wenn alle Sinus Filter Schütze korrekt arbeiten, wird der grüne Haken aktiviert, andernfalls nicht (Variable SinusFilterSchutze). Wenn einer oder mehrere Sinus Filter Schütze nicht korrekt arbeiten, wird das rote "X" aktiviert; andernfalls nicht (Variable SinusFilterSchutzeFehler).

```
IF TestN > 4 AND ResultSinus THEN
    SinusFilterSchutze := TRUE;
ELSE    SinusFilterSchutze := FALSE;
END_IF
IF TestN > 4 AND NOT ResultSinus THEN
    SinusFilterSchutzeFehler := TRUE;
ELSE    SinusFilterSchutzeFehler := FALSE;
END_IF
```

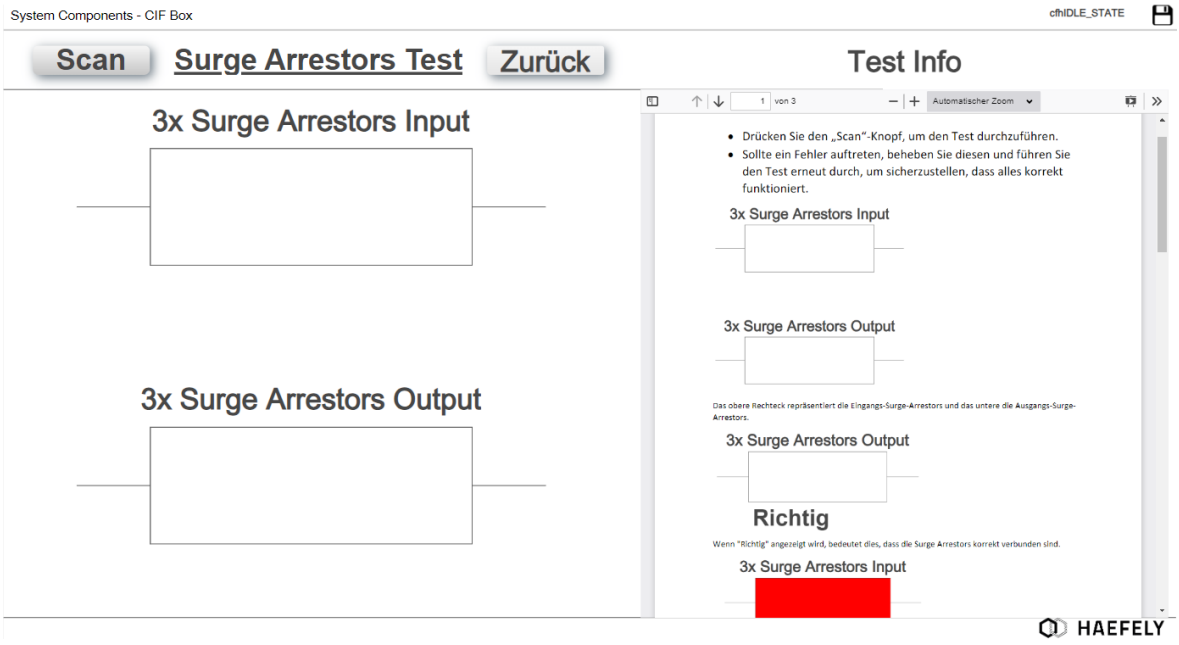
6.2.6. Surge Arrestors Test

Bei einem Fehler im "Surge Arrestors Test" kann man die Seite "SurgeArrestorsPage" öffnen, um die genaue Fehlerursache zu identifizieren:



SurgeArrestorsPage öffnen

So sieht der SurgeArrestorsPage aus:



SurgeArrestorsPage

Mit dem "Scan"-Knopf kann der Nutzer überprüfen, wo ein Problem vorliegt. Nach Behebung des Problems kann er den Scan erneut durchführen, um zu überprüfen, ob der Fehler behoben wurde.

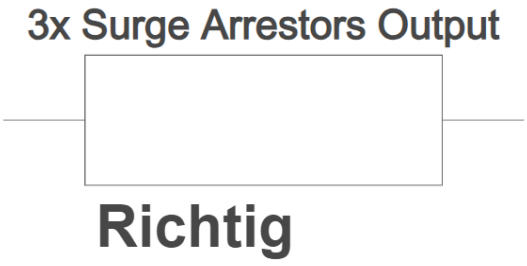
Der "Zurück"-Knopf dient als Navigationshilfe, um den Benutzer zur Hauptseite zurückzuführen. Das obere Rechteck symbolisiert das erste Set von drei Überspannungsableitern (Surge Arrestors) und das untere das zweite Set.

Ein weißes Rechteck mit der Beschriftung „Richtig“ signalisiert, dass alles in Ordnung ist. Ein rotes Rechteck mit der Beschriftung „Fehler“ weist darauf hin, dass eine Korrekturmaßnahme erforderlich ist.

Das PDF auf der rechten Seite bietet dem Benutzer eine Anleitung und Hinweise zur Fehlerbehebung, es sieht so aus:

- Drücken Sie den „Scan“-Knopf, um den Test durchzuführen.
- Sollte ein Fehler auftreten, beheben Sie diesen und führen Sie den Test erneut durch, um sicherzustellen, dass alles korrekt funktioniert.

Das obere Rechteck repräsentiert die Eingangs-Surge-Arrestors und das untere die Ausgangs-Surge-Arrestors.

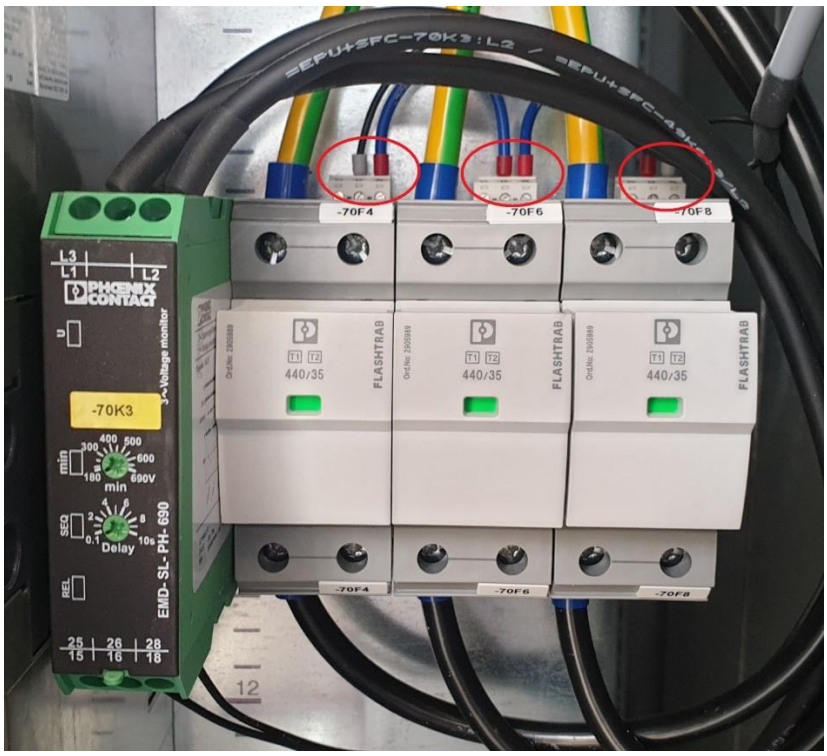


Wenn "Richtig" angezeigt wird, bedeutet dies, dass die Surge Arrestors korrekt verbunden sind.

Wenn "Fehler" angezeigt wird, bedeutet dies, dass sie nicht korrekt verbunden sind.

Im Fehlerfall:

- Sollte ein Fehler bei den Eingangs-Surge Arrestors (70F4, 70F6, 70F8) auftreten, können Sie die Pins 11 und 12 eines oder mehrerer Surge Arrestors miteinander verbinden (kurzschließen), um festzustellen, ob der Fehler bei diesem oder diesen Surge Arrestors liegt.



- Sollte ein Fehler bei den Ausgangs-Surge Arrestors (99F3, 99F7, 99F8) auftreten, können Sie die Pins 11 und 12 eines oder mehrerer Surge Arrestors miteinander verbinden (kurzschließen), um festzustellen, ob der Fehler bei diesem oder diesen Surge Arrestors liegt.

Das sind die Variablen des Funktionsblocks für die Surge Arrestors:

Name	Type	Reference	Scope	Constant	Retain	Replicable
SurgeArrestorsFB		☐				☑
BeginSurgeArrestors	BOOL	☐	VAR_INPUT	☐	☐	☑
SA1	BOOL	☐	VAR_INPUT	☐	☐	☑
SA2	BOOL	☐	VAR_INPUT	☐	☐	☑
SurgeArrestors	SurgeArrester...	☐	VAR_INPUT	☐	☐	☑
SurgeArrestorsTestEndeC	BOOL	☐	VAR_INPUT	☐	☐	☑
SurgeArrestorsResultat	BOOL	☐	VAR_OUTPUT	☐	☐	☑
SurgeArrestorsTestEnde	BOOL	☐	VAR_OUTPUT	☐	☐	☑
TMSA	TON	☐	VAR	☐	☐	☑
SurgeArrestor01	STRING[80]	☐	VAR	☐	☐	☑
SurgeArrestor02	STRING[80]	☐	VAR	☐	☐	☑
ColorSurgeArrestor01	BOOL	☐	VAR	☐	☐	☑
ColorSurgeArrestor02	BOOL	☐	VAR	☐	☐	☑
zzEdge00000	BOOL	☐	VAR	☐	☐	☑

Surge Arrestors Variablen

Das ist der Programmtext des Funktionsblocks für die Surge Arrestors:

```
FUNCTION_BLOCK SurgeArrestorsFB

    IF SurgeArrestorsTestEndeC=FALSE THEN
        SurgeArrestorsTestEnde:=FALSE;
    END_IF

    TMSA();

    IF BeginSurgeArrestors  THEN

        TMSA.IN:=TRUE;
        TMSA.PT:=UINT_TO_TIME(1000);

        IF SA1 THEN
            SurgeArrestor01 := 'Richtig';
            ColorSurgeArrestor01:=FALSE;
        ELSE
            SurgeArrestor01 := 'Fehler';
            ColorSurgeArrestor01:=TRUE;
        END_IF

        IF SA2 THEN
            SurgeArrestor02 := 'Richtig';
            ColorSurgeArrestor02:=FALSE;
        ELSE
            SurgeArrestor02 := 'Fehler';
            ColorSurgeArrestor02:=TRUE;
        END_IF

        IF SA1 AND SA2 THEN
            SurgeArrestorsResultat:=TRUE;
        ELSE
            SurgeArrestorsResultat:=FALSE;
        END_IF

        IF TMSA.Q THEN
            SurgeArrestorsTestEnde:= TRUE;
            TMSA.IN:=FALSE;
            BeginSurgeArrestors:=FALSE;
        END_IF

    END_IF

END_FUNCTION_BLOCK
```

Die Variablen sind:

- BeginSurgeArrestors: Diese Variable startet den Test der Surge Arrestors, wenn das System im Zustand TestN = 5 ist.
- SurgeArrestorsTestEnde: Diese Variable informiert das Hauptprogramm darüber, dass der Surge Arrestors Test abgeschlossen ist, und ermöglicht es dem System, zum nächsten Zustand zu wechseln, bei dem TestN den Wert 6 annimmt.
- SurgeArrestorsTestEndeC: Dies ist eine Variable, die dazu dient, die Variable SurgeArrestorsTestEnde jedes Mal zurückzusetzen, wenn man den "Stop"-Knopf betätigt.

- SurgeArrestorsResultat: Wenn diese Variable den Wert TRUE annimmt, bedeutet das, dass alle Surge Arrestors korrekt funktionieren.

- SA1 und SA2: Diese beiden Variablen stellen den Kontakt zu den Pins des Digitalen Eingangs her, um zu prüfen, ob sie mit Spannung versorgt werden.

- ColorSurgeArrestor01 und ColorSurgeArrestor02: Dies sind Variablen, die das GUI visuell beeinflussen. Diese Color-Variablen aktivieren die rote Farbe bei den zugehörigen Rechtecken, falls bei einem der Surge Arrestors Sets ein Fehler vorliegt.

Die Ergänzungen, die im Hauptprogramm vorgenommen wurden, standen in direktem Zusammenhang mit den Variablen des Contactor Funktionsblocks:

- Den Funktionsblock in Hauptprogramm anrufen.

```
SurgeArrestorsBaustein();
```

- Wenn das System den Case TestN = 5 erreicht, aktiviert die Variable BeginSurgeArrestors den Surge Arrestors Test. Sobald der Test beendet ist, informiert die Variable SurgeArrestorsTestEnde das Hauptprogramm, und das System wechselt zum nächsten Case (TestN=6).

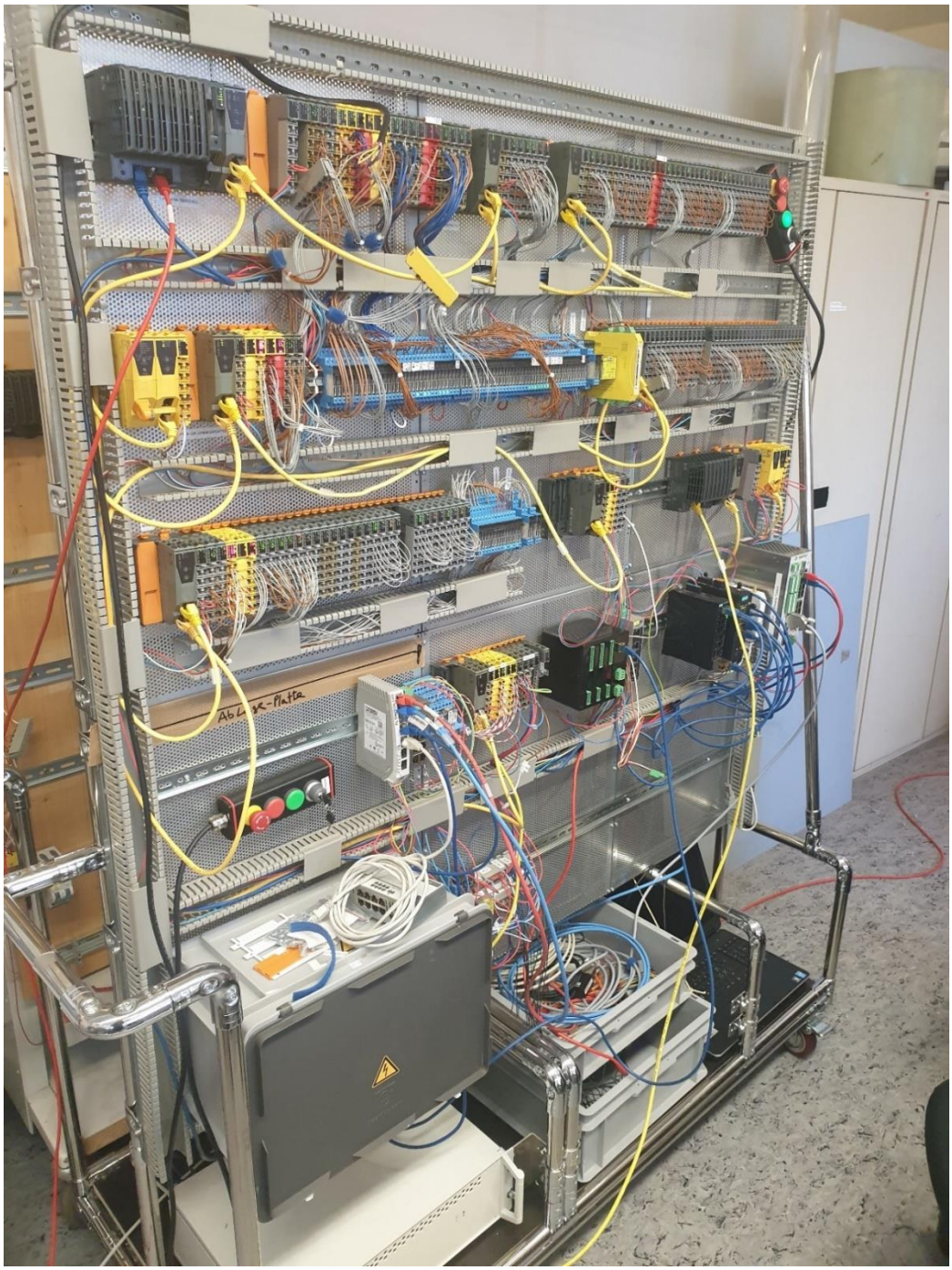
```
5:
    PDF := 'Media\PDF\LaufendesTest\SurgeTestLauft.pdf';
    SurgeArrestorsBaustein.BeginSurgeArrestors:=TRUE;

    IF SurgeArrestorsBaustein.SurgeArrestorsTestEnde THEN
        TestN:=6;
    END IF
```

- Wenn beide Surge Arrestors Sets korrekt funktionieren, wird das Symbol des grünen Häkchens aktiviert. Funktioniert jedoch eines oder beide Sets nicht ordnungsgemäß, wird das rote X-Symbol aktiviert..

```
IF TestN > 5 AND SurgeArrestorsBaustein.SurgeArrestorsResultat THEN
    SurgeArrestors := TRUE;
ELSE    SurgeArrestors := FALSE;
END_IF
IF TestN > 5 AND NOT SurgeArrestorsBaustein.SurgeArrestorsResultat THEN
    SurgeArrestorsFehler := TRUE;
ELSE    SurgeArrestorsFehler := FALSE;
END_IF
```

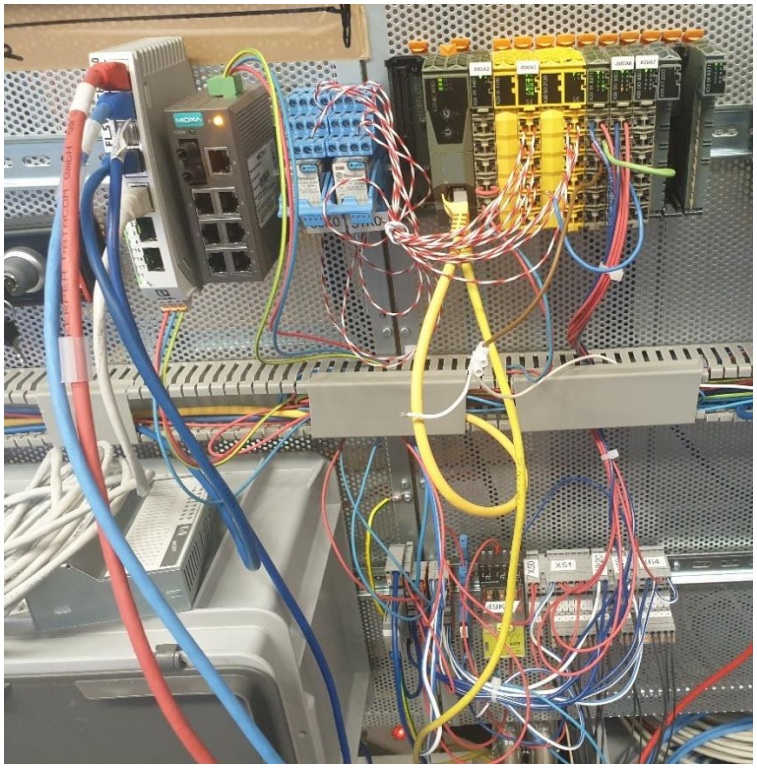
7. Durchführung der Tests

Um die Tests durchzuführen, musste zunächst die Software auf die SPS hochgeladen werden. Bei Haefely AG gibt es eine spezielle SPS, die mehrere Anlagen mit Modulen und anderen Komponenten simuliert. Unter diesen simulierten Anlagen befindet sich auch der Frequenzumrichter. Hier eine Beschreibung des Aussehens:

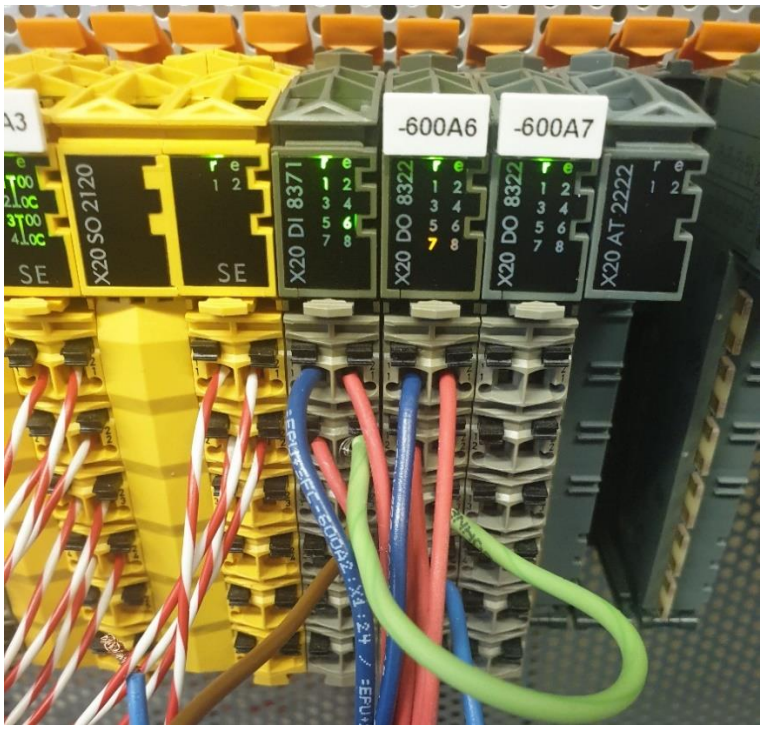


Testenanlage

Die Simulationsumgebung bei Haefely AG stellt den Frequenzumrichter realitätsnah dar und ermöglicht es, verschiedene Testszenarien und Bedingungen zu emulieren. Dies gewährleistet, dass die Software korrekt mit der physischen Hardware des Frequenzumrichters interagiert, bevor sie in realen Anwendungen eingesetzt wird. Die Simulation berücksichtigt alle wichtigen Komponenten und deren Interaktionen, um ein umfassendes Testen der Software zu ermöglichen.



Simulation des Frequenzumrichters



B&R Module die für die Tests benutzt waren

Die Tests wurden wie folgt durchgeführt: Durch das Ein- und Ausstecken der Kabel wurde überprüft, ob das GUI korrekt auf diese Änderungen reagiert. Dabei wurde beobachtet, ob im GUI das Wort "Richtig" innerhalb des zugehörigen Rechtecks angezeigt wurde, wenn die Verbindung korrekt hergestellt war. Ebenso wurde kontrolliert, ob "Fehler" angezeigt wurde, wenn die Verbindungen nicht korrekt waren. Dies entsprach den Beschreibungen in den zugehörigen Test-PDFs.

8. Zielerreichung

Nach eingehender Überprüfung und Durchführung der Tests wurden folgende Ziele erreicht:

- **Benutzerfreundlichkeit des GUI-Systems:** Das entwickelte GUI-System gewährleistet eine intuitive und benutzerfreundliche Handhabung, was eine effiziente Bedienung und Verständlichkeit für den Anwender sicherstellt.
- **Speisung Test:** Das System bietet dem Kunden eine klare visuelle Rückmeldung darüber, ob die Speisung korrekt konfiguriert ist. Dies erleichtert die Fehlererkennung und -behebung erheblich.
- **Module Test:** Durch die Implementierung des Tests kann der Kunde sowohl die korrekte Installation aller B&R-Module gemäß der Softwarekonfiguration überprüfen als auch sicherstellen, dass sie an den vorgesehenen Positionen platziert sind und nicht fehlerhaft vertauscht wurden.
- **Ein-Ausgangsschütze Test:** Mit dem entwickelten Testverfahren kann der Kunde eindeutig erkennen, ob die B&R DI/Os, die mit den Eingangs- und Ausgangsschützen verbunden sind, korrekt konfiguriert und in einwandfreiem Zustand sind.
- **Sinusschütze Test:** Der Kunde hat die Möglichkeit, zu überprüfen, ob die B&R DI/Os, die mit den Sinusschützen verbunden sind, korrekt konfiguriert sind und ordnungsgemäß funktionieren.
- **Surge Arrestors Test:** Das System ermöglicht es dem Kunden, sicherzustellen, dass die B&R DI/Os, die mit den Surge Arrestors verbunden sind, korrekt positioniert und voll funktionsfähig sind.

Insgesamt erfüllt das System somit alle vorgegebenen Anforderungen und gewährleistet eine zuverlässige und effiziente Überwachung und Steuerung der entsprechenden Komponenten.

9. Abschluss

9.1. Reflektion

Während ich auf die vergangenen sechs Wochen zurückblicke, empfinde ich eine tiefgehende Dankbarkeit und Zufriedenheit. Die Chance, an diesem Projekt teilzunehmen, war zweifellos eine wertvolle Erfahrung, die ich nicht so schnell vergessen werde.

Die Komplexität und Tiefe, die mir die Bereiche der Automation, B&R SPS und des strukturierten Textes boten, waren nicht nur herausfordernd, sondern auch unglaublich bereichernd. Jeder Tag brachte neue Erkenntnisse und Perspektiven mit sich und half mir dabei, nicht nur meine technischen Fähigkeiten zu schärfen, sondern auch mein Verständnis für die Anwendungspraxis in der realen Industriewelt zu vertiefen.

Das erste Eintauchen in die Welt von B&R SPS und strukturiertem Text stellte mich vor viele neue Herausforderungen. Doch trotz der gelegentlichen Schwierigkeiten fand ich mich immer wieder fasziniert von den Möglichkeiten und dem Potential, das diese Technologien in der modernen Automation bieten.

Diese Erfahrung hat nicht nur mein berufliches Profil bereichert, sondern auch meine Neugier und mein Verlangen nach Weiterbildung in diesen Bereichen geweckt. Das Projekt hat mir klargemacht, dass es immer Raum für Weiterentwicklung und Neues Lernen gibt und dass die Welt der Technik eine ständig sich weiterentwickelnde Landschaft ist. Es hat mir auch gezeigt, dass ich in diesem Bereich eine langfristige und erfüllende Karriere sehe.

Abschließend bin ich Haefely AG unendlich dankbar für diese wertvolle Gelegenheit. Es war nicht nur ein Sprungbrett für meine berufliche Entwicklung, sondern auch eine Zeit intensiven Lernens, Entdeckens und Wachsens. Ich freue mich darauf, diese neu gewonnenen Kenntnisse in künftigen Projekten und Herausforderungen anzuwenden und weiterzuentwickeln.

9.2. Verdankungen

Inmitten der täglichen Herausforderungen und Entdeckungen, die dieses Projekt mit sich brachte, kann ich nicht übersehen, wie entscheidend die Unterstützung und Anleitung meiner Kollegen war. An vorderster Front möchte ich meinem Vorgesetzten, Herrn Adrian John, meine tiefe Dankbarkeit ausdrücken. Seine Führung und Geduld haben den Weg für meine Lernerfahrungen und Fortschritte geebnet.

Meine Dankbarkeit erstreckt sich ebenfalls auf meine engagierten Kolleginnen, Anna Koziec und Laura Wollschläger. Ihre ständige Bereitschaft, mir zu helfen, insbesondere in technischen Angelegenheiten, war ein unschätzbares Sicherheitsnetz während meiner Arbeit.

Ein besonderer Dank geht an Anna Koziec. Durch ihre Expertise und methodische Anleitung konnte ich nicht nur die Nuancen von Automation Studio schneller verstehen, sondern auch effizient in mapp View arbeiten. Ihre Lehrmethode und ihre Geduld haben es mir ermöglicht, diese Werkzeuge effektiv zu nutzen und meinen Arbeitsablauf zu optimieren.

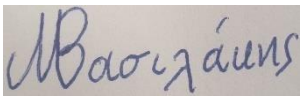
Die Zusammenarbeit mit solch kompetenten und unterstützenden Kollegen war zweifellos einer der Höhepunkte meiner Zeit hier. Ihr Einfluss hat nicht nur die Qualität meiner Arbeit verbessert, sondern auch meinen persönlichen und beruflichen Wachstumspfad bereichert.

9.3. Eigenständigkeitserklärung

Ich bestätige, dass ich die vorliegende Diplomarbeit selbstständig verfasst und alle benutzten Quellen gekennzeichnet habe.

Vorname / Name:
Mario-Otto Vasilakis

Ort / Datum / Unterschrift:
Basel 26.09.2023



M.Vasilakis

9.4. Quellenangaben

https://www.br-automation.com/de-ch