This Word layout template for Help & Manual defines the entire layout - front page, table of contents, chapter headings, keyword index.

To **customize this template**, use the original file ("Word-Template-A4.docx") from the H&M installation folder, make a copy of it, then tailor it with MS Word to your requirements. You **can use variables** in every place of the document. To insert a variable, insert a COMMENT field in Word, then type the variable name as the field content. Like the field above this text.

Move on to the next page in this document to customize the table of contents (a placeholder for the real TOC later) and modify text and paragraph format of the chapter headings in the middle of this template. You might also want to change the numbering style.

**Important:** before you use the template for publishing in H&M, make sure that general page format and margins are set correctly. H&M will export images and tables to make them fit in the space available.

Selmo Technology Gmbh and EC Software GmbH

**[Type the company address]**

**[Type the phone number]**

**[Type the fax number]**

Selmo Technology Gmbh

**2023**

**Selmo Standard**

Contents

[Selmo Technology Gmbh and EC Software GmbH 1](file:///C:\Users\GeraldOkorn\Documents\Help+Manual\Export\WORD\Selmo_PlcCode.docx#_Toc132716617)

[1. PLC Code 3](#_Toc132716618)

[1.1. Programmstruktur 4](#_Toc132716619)

[1.2. Plant 5](#_Toc132716620)

[1.2.1. Global 7](#_Toc132716621)

# PLC Code

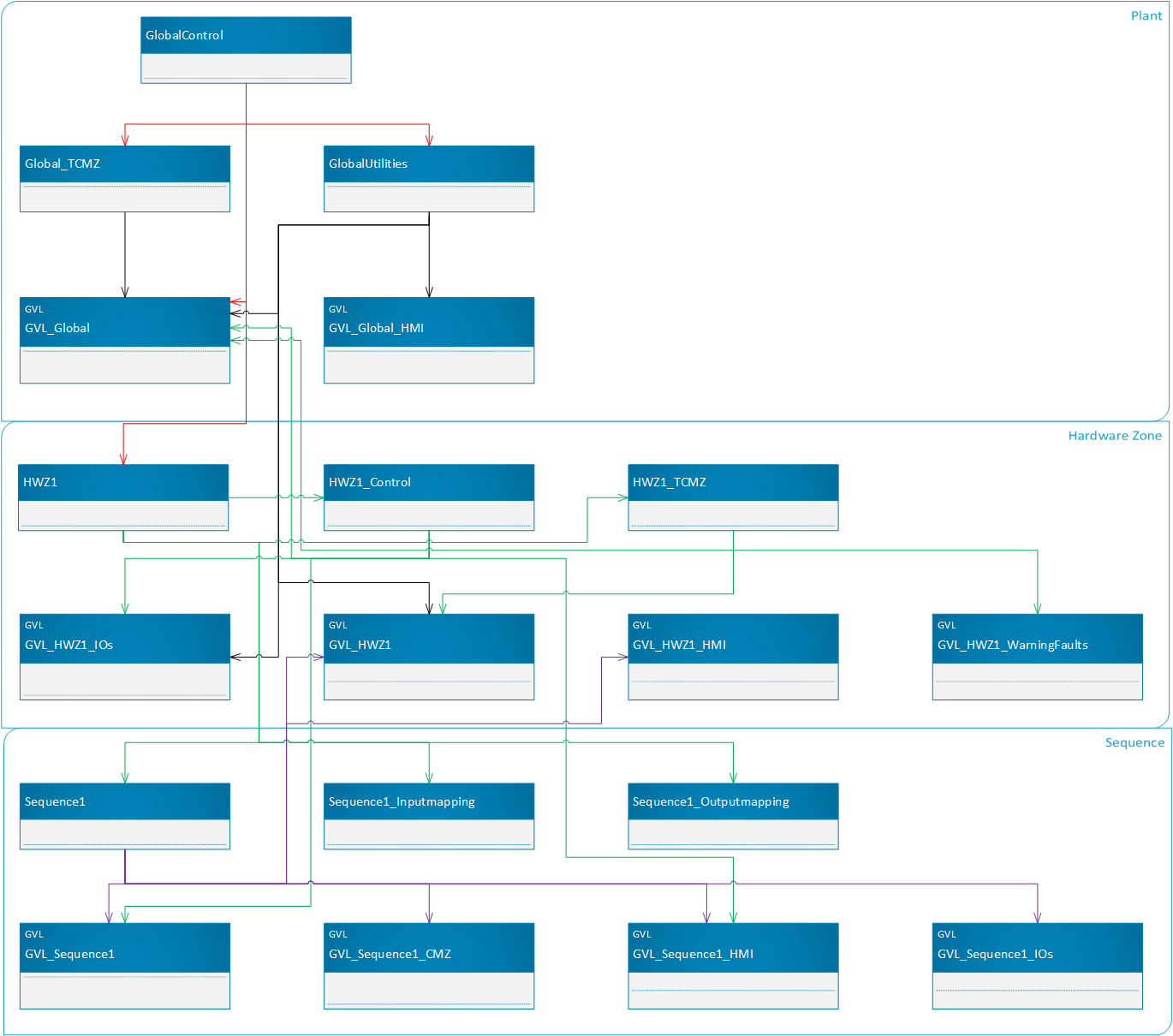
Der PLC-Code wird in der Programmiersprache Structured Text (ST) erzeugt und als PLCopen XML exportiert. Dies kann dann in der ausgewählten Steuerung importiert und geladen werden. Structured Text (ST) ist eine Highlevel-Programmiersprache, die für programmierbare Steuerungen (PLCs) entwickelt wurde. Es verwendet eine ähnliche Syntax wie Hochsprachen wie Pascal oder C und ermöglicht es Benutzern, komplexe Prozesse in einer einfachen und lesbaren Form zu beschreiben. PLCopen ist eine internationale Vereinigung, die sich der Standardisierung von PLC-Programmierung und -Steuerung widmet. Eines ihrer Ziele ist es, eine einheitliche Programmierumgebung für PLCs zu schaffen, um die Interoperabilität zwischen verschiedenen Systemen und Herstellern zu erleichtern. PLCopen unterstützt auch die Verwendung von Structured Text als eine der Programmiersprachen für PLCs. Zusammen ermöglichen Structured Text und PLCopen es Entwicklern, PLC-Systeme einfach und effizient zu programmieren, während gleichzeitig die Interoperabilität und Kompatibilität zwischen Systemen sichergestellt wird.

## Programmstruktur

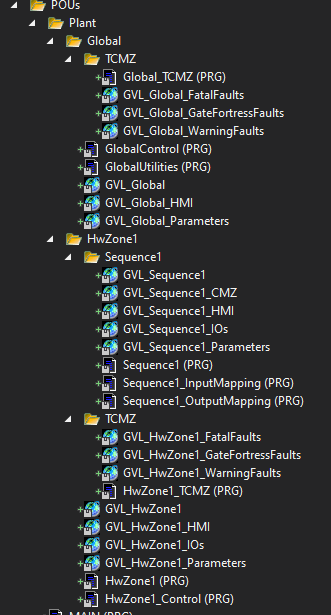
Die Programmstruktur des generierten PLC-Code wird in wie bei der Modellierung in drei Ebenen unterteilt:

* Plant
* Hardware Zone
* Sequence

Jede Ebene setzt sich aus Teilprogrammen und Globalen Variablenlisten zusammen, welche mittels Schnittstellen mit den anderen Ebenen kommunizieren. In der nachfolgenden Graphik sind die Ebenen mit den jeweiligen Teilprogrammen und Globalen Variablenlisten dargestellt. Werden mehre Sequencen oder Hardware Zones modelliert erweitert sich die Struktur generisch horizontal.



## Plant



Selmo Standard ist eine strukturierte Programmiertechnik, bei der das Programm in Einzelteile zerlegt wird, die jeweils einen bestimmten Zweck haben und zusammen die gesamte Steuerungs- und Diagnosefunktionalität bieten. Die Struktur von Selmo Standard kann im Großen und Ganzen aus den folgenden Komponenten bestehen. In den folgenden Abschnitten wird jede Komponente ausführlich beschrieben.

Global Utilities

Das Segment Global Utilities enthält SPS-Logik, die Funktionen bereitstellt, die im gesamten SPS-Programm verwendet werden können. Eine Reihe von Flags und Registern, die jeweils eine eigene Funktion haben, können je nach Bedarf verwendet werden.

Globale Utilities umfassen :

Flash-Ausgang,

Lampentest,

Fehlerquittierung usw.

Hardware-Zonen Control

Dieser Abschnitt bezieht sich auf den Zustand der zugehörigen Sequenzen, d.h. ob sie sich alle im manuellen oder im automatischen Zonenmodus befinden. Ein Zonen-Automatik-Ausgang wird auch verwendet, um alle zugehörigen Sequenzen zu starten, die sich im Automatik-Modus befinden, wenn ein Start-Automatik-Taster für diese Zone gedrückt wird.

Jede Hardware-Zone enthält eine Total Constantly Montiored Zone (T.C.M.Z).

Eine ständig überwachte Zone ist ein Bereich der SPS-Software, der Logik enthält, die ständig überprüft wird.

Not-Aus-Tasten, Leistungsschalter, Schutztore und dergleichen sollten sich in diesem Bereich des Programms befinden.

Die T.C.M.Z. enthält Fehler, die für mehr als eine Sequenz gelten, um zu verhindern, dass diese Fehler in jeder Sequenz doppelt auftreten.

Sequence-Logic

Die Sequenzlogik bezieht sich auf die eigentliche Steuer- und Diagnoselogik für die Schrittfolgen. Das SPS-Segmentlayout gibt an, wie die Sequenzen nacheinander angeordnet sind. Jede Step Sequence Logic ist in vier verschiedene Teile unterteilt.

1. Standard-Beginning

Der "Standard Beginning" bestimmt die Betriebsart der Schrittfolge und die Startbedingungen, er überwacht den Fehlerstatus der Folge, um einen automatischen oder manuellen Betrieb zu ermöglichen.

2. Sequence Logic Control

Der Bereich "Sequence Logic Control" stellt die Logik zur Verfügung, die für die eigentliche Steuerung und Überwachung jeder einzelnen Maschinenbewegung / Aktion verantwortlich ist.

3. Sequence Constantly Monitored Zone

Die "Constantly Monitored Zone" oder C.M.Z. bietet die Logik, die kontinuierlich unsichere Maschinenzustände überwacht, wie z.B. den Ausfall von Sicherungen, Leistungsschaltern usw. Jeder Fehler dieser Art führt zu einen sofortigen Stopp der Step Sequence.

4. Standard End

Der Abschnitt "Standard End" enthält eine Logik, die den Schrittstatus überwacht, um den Schrittzähler zu steuern und Fehlerinformationen zu formatieren, die an das HMI-Diagnosesoftwarepaket "Selmo HMI" gesendet werden.

### Global

Das Global-Segment enthält SPS-Logik, die Funktionen bereitstellt, die im gesamten SPS-Programm verwendet werden können.

#### TCMZ

Eine Total constantly monitored Zone (TCMZ) in Maschinen bezieht sich auf einen Bereich in einer Maschine oder einem System, der ständig überwacht wird, um sicherzustellen, dass die Maschine oder das System ordnungsgemäß funktioniert. Dies kann mithilfe von Sensoren oder anderen Überwachungstechnologien erfolgen, die im Bereich installiert sind und Daten über den Betrieb der Maschine sammeln. Die Überwachung der TCMZ kann dazu beitragen, mögliche Probleme frühzeitig zu erkennen und zu beheben, bevor sie zu Ausfällen oder Schäden führen, und somit die Betriebssicherheit und Zuverlässigkeit der Maschine erhöhen.

Jedes TCMZ ist in 3 definierte Bereiche unterteilt: Fatal Fault, Gate/Fortress Fault und Warning Fault.

##### Global\_TCMZ

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*{attribute 'symbol' := 'readwrite'}*  
**PROGRAM** Global\_TCMZ  
**VAR**  
 *{attribute 'symbol' := 'none'}*  
 fbFatalFault: FB\_CMZFault;  
 aFatalFault: **ARRAY**[0..2] **OF** **BOOL**;  
 *{attribute 'symbol' := 'none'}*  
 fbGateFortressFault: FB\_CMZFault;  
 aGateFortressFault: **ARRAY**[0..1] **OF** **BOOL**;  
 *{attribute 'symbol' := 'none'}*  
 fbWarningFault: FB\_CMZFault;  
 aWarningFault: **ARRAY**[0..0] **OF** **BOOL**;  
**END\_VAR**

Fatal Fault

Das Fatal Fault Flag ist ein SPS-Ausgang, der den Status des Fatal Fault-Bereichs der TCMZ-Matrix der Anlage überwacht, so dass jedes aktivierte Fatal Fault-Bit zum Aktivieren des Fatal Fault Flag Ausgang führt. Dieses Flag wird in allen Hardwarezonen-Fehlermatrizen verwendet, um einen CMZ-Fehler auszulösen und die automatische/manuelle Freigabe in jeder Sequenz innerhalb der Hardwarezone zu verhindern. Eine Standard-Index in jeder Sequenzfehler-Matrix ist für diesen Fehler vorgesehen. Der fatale Fehler muss behoben werden, bevor eine automatische oder manuelle Freigabe möglich ist. Schwerwiegende Fehler können nicht übersteuert werden.

*// ###################### fatal faults ######################*  
 *// TCMZ assigns*  
 aFatalFault[0] := GVL\_Global\_FatalFaults.<xCMZ\_1>;  
 aFatalFault[1] := GVL\_Global\_FatalFaults.<xCMZ\_2>;  
 aFatalFault[2] := GVL\_Global\_FatalFaults.<xCMZ\_3>;  
  
 *// TCMZ call*  
 fbFatalFault(aFaultMatrix := aFatalFault, xFaultActive => GVL\_Global.stGlobalIf.xFatalFault);

<\*\*\*> Variablen Name od. Kommentar wird aus dem Studio automatisch übernommen

Gate/Fortress Fault

Das Gate/Fortress-Fault Flag ist ein SPS-Ausgang, der den Status des Gate/Fortress Fault-Bereichs der TCMZ-Matrix der Anlage überwacht, so dass jedes aktivierte Gate/Fortress Fault-Bit zum Aktivieren des Gate/Fortress Fault Flag Ausgangs führt. Dieses Flag wird in allen Hardwarezonen-Fehlermatrizen verwendet, um einen CMZ-Fehler zu verursachen und die automatische und manuelle Freigabe in jeder Sequenz innerhalb der Hardwarezone zu verhindern.

*// ###################### gate / fortress faults ######################*  
 *// TCMZ assigns*  
 aGateFortressFault[0] := GVL\_Global\_GateFortressFaults.<xCMZ\_1>;  
 aGateFortressFault[1] := GVL\_Global\_GateFortressFaults.<xCMZ\_2>;  
  
 *// TCMZ call*  
 fbGateFortressFault(aFaultMatrix := aGateFortressFault, xFaultActive => GVL\_Global.stGlobalIf.xGateFortressFault);

<\*\*\*> Variablen Name od. Kommentar wird aus dem Studio automatisch übernommen

Warning Fault

Für jede Anlage gibt es eine zugehörige Warning Fault Matrix. Die Warning Fault haben keinen Einfluss auf den Maschinenbetrieb. Der Status der Warning Fault Matrix wird jedoch ständig von der "HMI" überwacht, die alle aktiven Warnmeldungen anzeigt.

*// ###################### warning faults ######################*  
 *// TCMZ assigns*  
 aWarningFault[0] := GVL\_Global\_WarningFaults.<xCMZ\_1>;  
  
 *// TCMZ call*  
 fbWarningFault(aFaultMatrix := aWarningFault, xFaultActive => GVL\_Global.stGlobalIf.xWarningFault);

<\*\*\*> Variablen Name od. Kommentar wird aus dem Studio automatisch übernommen

##### GVL\_Global\_FatalFaults

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*///*  
*/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL**  
 *///<CMZ 1>*  
 <xCMZ\_1>: <**BOOL**>;  
 *///CMZ 2*  
 <xCMZ\_2>: <**BOOL**>;  
 *///CMZ 3*  
 <xCMZ\_3>: <**BOOL**>;  
**END\_VAR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

##### GVL\_Global\_GateFortressFaults

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*///*  
*/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL**  
 *///<CMZ 1>*  
 <xCMZ\_1>: <**BOOL**>;  
 *///<CMZ 2>*  
 <xCMZ\_2>: <**BOOL**>;  
**END\_VAR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

##### GVL\_Global\_WarningFaults

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*///*  
*/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL**  
 *///<CMZ 1>*  
 <xCMZ\_1>: <**BOOL**>;  
**END\_VAR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

#### GlobalControl

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*{attribute 'symbol' := 'readwrite'}*  
**PROGRAM** GlobalControl  
**VAR**  
**END\_VAR**

Globale Calls

Dieser SPS-Code ruft die globalen Objekte und alle Hardware-Zonen auf.

*// ###################### Global Calls #######################*  
 Global\_TCMZ(); *// Global CMZ*   
 GlobalUtilities(); *// Global functions*  
  
*// HWZ calls*  
 <HwZone1>();

Licensing

Dieses Programmmodul dient der ständigen Überprüfung der Lizenz. Falls noch keine Lizenz erworben wurde, kann dies durch eine E-Mail an license@selmo.at oder online über Selmo Studio erfolgen. Für die Lizenzierung wird eine Request-ID "GVL\_Global.strRequestID" benötigt. Diese ID wird vom Programmmodul automatisch ausgelesen. Der erworbene Lizenzschlüssel muss dann unter "GVL\_Global.strLicenseKey" eingetragen werden. Wurde keine Lizenzierung durchgeführt, befindet sich das gesamte SPS-Programm im Demo-Modus. In diesem Modus wird die 'Automatische Betriebsfreigabe' nach 30 Minuten zurückgesetzt. Bitte beachten Sie, dass dieser Programmbaustein nur einmal pro SPS aufgerufen wird! Die Prüfsumme dient zur Konsistenzprüfung und setzt bei Inkonsistenz die 'Automatische Betriebsfreigabe' nach 30 Minuten zurück, auch wenn ein Lizenzschlüssel vorhanden ist. Dies kann bei Bedarf mit den beiden booleschen Variablen "GVL\_Global.xLicenseValid" und "GVL\_Global.xChecksumValid" überprüft werden.

*// ###################### LICENSING ########################*

*{region "Description Licensing"}*

*(\**

*This program module is used to constantly check the license. If a license has not yet been purchased, this can be done by sending an e-mail to support@selmo.at or online via SelmoStudio. A request ID "GVL\_Global.strRequestID" is required for licensing. This ID is read automatically by the program block. The purchased license key must then be entered at "GVL\_Global.strLicenseKey". If no licensing has been carried out, the entire PLC program is in demo mode. In this mode the 'Automatic operation release' is reset after 30 minutes. Please note that this program block is only called once per PLC!*

*\*)*

*{endregion}*   
 SelmoLicenseInfo(  
 strLicenseKey := GVL\_Global.strLicenseKey,  
 dwChecksum := 16#436C185A,  
 strRequestID => GVL\_Global.strRequestID,  
 xLicenseValid => GVL\_Global.xLicenseValid,  
 xChecksumValid => GVL\_Global.xChecksumValid);

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

#### GlobalUtilities

Die Global Utilities sind Teil der SPS, die:

* Ein Ausgang blinkt kontinuierlich, wenn dies erforderlich ist, z. B. um eine Not-Aus-Rückstelllampe zu blinken.
* Aktiviert eine Sirene, wenn eine Zone automatisch startet.
* Stellt einen Ausgang zur Verfügung, der zum Testen aller Lampen verwendet werden kann.

Dieser Abschnitt befindet sich im Segment Global der SPS Prgramms. Da es sich um Logik für globale Bedingungen handelt, wird er nur einmal im gesamten SPS Programm verwendet.

Die Variablen aus diesen Netzwerken werden in den meisten/allen Sequenzen verwendet.

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*{attribute 'symbol' := 'none'}*  
**PROGRAM** GlobalUtilities  
**VAR**  
 iIndex: **INT**;  
 aStartUpSirenMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 aLampTestMatrix: **ARRAY**[0..1] **OF** **BOOL**;  
 aAutoRelMatrix: **ARRAY**[0..1] **OF** **BOOL**;  
 aResetMatrix: **ARRAY**[0..1] **OF** **BOOL**;  
 fbFlashOn: TOF;  
 fbFlashOff: TON;  
**END\_VAR**

Flasher Output

Dieser SPS-Code ist so programmiert, dass er einen Takt-Ausgang zur Verwendung in einer beliebigen Sequenz bereitstellt. Der Timer-Block (fbFlashOn) gibt die Zeit an, in der der Takt EIN ist, der zweite (fbFlashOff) die Zeit, in der der Takt AUS ist. Zusammen wirken sie wie eine Flip-Flop-Schaltung, die einen Ein/Aus-Effekt erzeugt.

//--------------------------------Flasher Output--------------------------------

{region "Description Flasher Output"}

(\*

This PLC code is programmed to provide a Flash Output for use within any sequence.

The Timer block(fbFlashOn) provides the time for which the Flash Output is ON, the second(fbFlashOff) for the time it is OFF.

Together they have the effect of a ‘flip - flop’ circuit to create an on / off effect.

\*)

{endregion}

fbFlashOff(IN:= NOT fbFlashOn.Q, PT:= T#1S);

fbFlashOn(IN:= fbFlashOff.Q, PT:= T#1S , Q=>GVL\_Global.stGlobalIf.xFlashOutput);

Startup Siren

Wenn eine Sequenz aus einer Zone, in der ein Tor geöffnet wurde, startbereit ist, wird der Ausgang der Startsirene aktiviert, wenn der Freigabetaster gedrückt wird.

//--------------------------------Start Up Siren--------------------------------

{region "Description Start Up Siren"}

(\*

When any sequence is ready to start from any zone in which a gate has been opened,

the start - up siren output is activated when the release pushbutton is pressed.

\*)

{endregion}

aStartUpSirenMatrix[0] := GVL\_<HwZone1>.stHwzIf.xStartUpAlarm;

FOR iIndex := 0 TO 0 DO

IF aStartUpSirenMatrix[iIndex] THEN

GVL\_Global.stGlobalIf.xStartUpSiren := TRUE;

EXIT;

ELSE

GVL\_Global.stGlobalIf.xStartUpSiren := FALSE;

END\_IF

END\_FOR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Lamp Test

Die Lampentestfunktion wird über eine Drucktaste auf der HMI realisiert. Durch Drücken dieser Taste wird der Ausgang "Lamp Test" in einer beliebigen Sequenz aktiviert. Dieser Ausgang wird dann zum Testen der Lampenausgänge verwendet, wo immer dies erforderlich ist.

//--------------------------------Lamp Test--------------------------------

{region "Description Lamp Test"}

(\*

The lamp test function is implemented via a pushbutton on the HMI or a hardware Button.

Pressing this button activates the Lamp Test output in any sequence.

This output is then used to test the lamp outputs wherever necessary.

\*)

{endregion}

aLampTestMatrix[0] := GVL\_Global\_HMI.stHmiIf.xHmiLampTest;

aLampTestMatrix[1] := GVL\_<HwZone1>\_IOs.xLampTest;

FOR iIndex := 0 TO 1 DO

IF aLampTestMatrix[iIndex] THEN

GVL\_Global.stGlobalIf.xLampTest := TRUE;

EXIT;

ELSE

GVL\_Global.stGlobalIf.xLampTest := FALSE;

END\_IF

END\_FOR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Global Release

Der SPS-Code summiert die Freigabe-Taster von allen Hardware-Zonen. Dadurch können alle Sequenzen in allen Zonen gleichzeitig automatisch gestartet werden, indem der globale Freigabe-Ausgang in jedem Hardware-Zonen-Automatikfreigabe-Netzwerk verwendet wird.

//--------------------------------Global Automatic Release--------------------------------

{region "Description Global Automatic Release"}

(\*

The PLC code sums together the release Push Button pressed Flag from all Hardware Zone.

This allows all sequences in all zones to be started in automatic simultaneously by employing

the Global Relaise Push Button pressed Output in every Hardware Zone Automatic Release network.

\*)

{endregion}

aAutoRelMatrix[0] := GVL\_Global\_HMI.stHmiIf.xHmiAutomaticRelease;

aAutoRelMatrix[1] := GVL\_<HwZone1>.stHwzIf.xReleasePbPressed;

FOR iIndex := 0 TO 1 DO

IF aAutoRelMatrix[iIndex] THEN

GVL\_Global.stGlobalIf.xRelease := TRUE;

EXIT;

ELSE

GVL\_Global.stGlobalIf.xRelease := FALSE;

END\_IF

END\_FOR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Global Fault Reset

Der folgende SPS-Code summiert die Fehler-Reset-Drucktasten aller Hardware-Zonen. Dies ermöglicht einen gemeinsamen Fehler-Reset für alle Zonen, wenn eine beliebige Fehler-Reset-Drucktaste gedrückt wird.

//--------------------------------Global Fault Reset--------------------------------

{region "Description Global Fault Reset"}

(\*

The PLC code below sums the fault reset flags from all Hardware Zone.

This allows a common fault reset to all zones when any zone fault reset Push Button is pressed.

\*)

{endregion}

aResetMatrix[0] := GVL\_Global\_HMI.stHmiIf.xHmiFaultReset;

aResetMatrix[1] := GVL\_<HwZone1>.stHwzIf.xFaultResetPbPressed;

FOR iIndex := 0 TO 1 DO

IF aResetMatrix[iIndex] THEN

GVL\_Global.stGlobalIf.xFaultReset := TRUE;

EXIT;

ELSE

GVL\_Global.stGlobalIf.xFaultReset := FALSE;

END\_IF

END\_FOR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

#### GVL\_Global

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*///*  
*/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL**  
 */// Globale Schnittstelle*  
 stGlobalIf: stGlobalInterface;  
 */// Output: request id für die Überprüfung des Lizenzschlüssels*  
 strRequestID: **STRING**;  
 */// Output: Lizenz gültig*  
 xLicenseValid: **BOOL**;  
 */// Output: Prüfsumme gültig*  
 xChecksumValid: **BOOL**;  
 */// Input: Lizenzschlüssel für die Nutzung der Sequenz ohne Demomodus*  
 strLicenseKey: **STRING** := '';  
**END\_VAR**

##### stGlobalInterface

**TYPE** stGlobalInterface :  
**STRUCT**  
 */// Output: Startsirene*  
 xStartUpSiren: **BOOL**;  
 */// Output: globales Zurücksetzen von Störungen*  
 xFaultReset: **BOOL**;  
 */// Output: globale Automatik-Freigabe*  
 xRelease: **BOOL**;  
 */// Output: Lampentest*  
 xLampTest: **BOOL**;  
 */// Output: schwerwiegender Fehler*  
 xFatalFault: **BOOL**;  
 */// Output: Tor/Schutzeinrichtung Fehler*  
 xGateFortressFault: **BOOL**;  
 */// Output: Warnmeldung*  
 xWarningFault: **BOOL**;  
 */// Output: Taktsignal 1Hz*  
 xFlashOutput: **BOOL**;  
 */// Input: globale Voreinstellung für Startverzögerung*  
 timGlobalPresetDelay: **TIME** := T#3S;  
**END\_STRUCT**  
**END\_TYPE**

#### GVL\_Global\_HMI

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*///*  
*/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL**  
 */// Globale HMI Schnittstelle*  
 stHmiIf: stHmiGlobalInterface;  
 */// Input: 0: English [1033] | -1: None*  
 iHmiLanguageIndex: **INT** := -1;  
**END\_VAR**

##### stHmiGlobalInterface

**TYPE** stHmiGlobalInterface :  
**STRUCT**  
 */// globaler HMI Button Lampentest*  
 xHmiLampTest: **BOOL**;  
 */// globaler HMI Button Automatik/Handbetriebswahlschalter*   
 xHmiAutoManMode: **BOOL**;  
 */// globaler HMI Button Automatik freigeben*  
 xHmiAutomaticRelease: **BOOL**;  
 */// globaler HMI Button Fehler quittieren*  
 xHmiFaultReset: **BOOL**;  
**END\_STRUCT**  
**END\_TYPE**

#### GVL\_Global\_Parameters

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL** PERSISTENT  
 *///<Parameter 1>*  
 <iParameter1>: <**INT**>;  
 *///<Parameter 2>*  
 <iParameter2>: <**INT**>;  
**END\_VAR**  
  
**VAR\_GLOBAL**  
 */// Empty VAR\_GLOBAL | Workaround for PersistentVars limitations*  
 \_placeholder: **BOOL**;  
**END\_VAR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

#### <HwZone1>

Wie bereits beschrieben, ist eine "Hardware-Zone" ein Bereich einer Maschine/Zelle, der eine unabhängige automatische/manuelle Steuerung ermöglicht. Eine "Hardware Zone" enthält eine oder mehrere Sequenzen.

Die Funktionalität der Hardware-Zone :

* Bestimmt den Auto-/Manuell-Status der Hardware-Zone.
* Bietet einen gemeinsamen Freigabebefehl für alle Sequenzen innerhalb der Hardware-Zone.
* Zeigt den Auto-/Manuell-Status der Hardware-Zone an.
* Überwacht "maschinenunsichere" Zustände im Zusammenhang mit der "Hardware-Zone" und meldet Fehler.
* Überwacht nicht-kritische Alarmmeldungen im Zusammenhang mit der Hardware-Zone und meldet Warnungen.

Die Anzahl der Sequenzen, die mit jeder Hardware-Zone verbunden sind, hängt von der jeweiligen Anwendung ab.

##### GVL\_<HwZone1>

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*///*  
*/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL**  
 */// Hardware Zonen Schnittstelle*  
 stHwzIf: stHardwareZoneInterface;  
**END\_VAR**

###### stHardwareZoneInterface

**TYPE** stHardwareZoneInterface :  
**STRUCT**  
 */// Output: Ein schwerwiegender Fehler liegt vor*  
 xFatalFault: **BOOL**;  
 */// Output: Ein Gate/Fortress Fehler liegt vor*  
 xGateFortressFault: **BOOL**;  
 */// Output: Eine Warnung ist in der Harwarezone aufgetreten*  
 xWarningFault: **BOOL**;  
 */// Output: Der TCMZ-Fehlerausgang wird aktiviert, wenn ein schwerwiegender Fehler oder ein Gate/Fortress-Fehler vorliegt*  
 xTcmzFault: **BOOL**;  
 */// Output: Die Betriebsart Automatik ist aktiviert, wenn alle für die Hardwarezone relevanten Betriebsartenschalter auf Position Automatik eingestellt sind*  
 xAutomaticModeOn: **BOOL**;  
 */// Output: Die Betriebsart Manuell ist aktiviert, wenn alle für die Hardwarezone relevanten Betriebsartenschalter auf Position Manuell eingestellt sind*  
 xManualModeOn: **BOOL**;  
 */// Output: Signal Anfahralarm(Siren)*  
 xStartUpAlarm: **BOOL**;  
 */// Output: Zeigt an, ob irgendeine Sequenz, die mit dieser Hardwarezone verbunden ist, startbereit ist, d.h. keine Verriegelung oder CMZ-Fehler aktiv ist*  
 xAnySeqReadyToStart: **BOOL**;  
 */// Output: Zeigt an, ob alle mit der Zone verbundenen Sequenzen fehlerfrei und damit startbereit sind*  
 xAllSeqReadyToStart: **BOOL**;  
 */// Output: Zeigt an, ob alle mit der Zone verbundenen Sequenzen eine automatische Freigabe haben und somit alle gestartet wurden*  
 xAllSeqAutoRelease: **BOOL**;  
 */// Output: Zeigt an, ob alle mit der Zone verbundenen Sequenzen das EOC Schritt erreicht haben*  
 xAllSeqEocReached: **BOOL**;  
 */// Output: Automatik Modus aktiv*  
 xAutomaticSelected: **BOOL**;  
 */// Output: Manueller Modus aktiv*  
 xManualSelected: **BOOL**;  
 */// Output: Automatik Freigabe wurde gedrückt*  
 xAutoReleaseRequest: **BOOL**;  
 */// Output: Eine beliebige Reset-Taste innerhalb der Zone aktiviert "Reset-Taste gedrückt"*  
 xFaultResetPbPressed: **BOOL**;  
 */// Output: Eine beliebige Release-Taste innerhalb der Zone aktiviert "Release-Taste gedrückt"*  
 xReleasePbPressed: **BOOL**;  
 */// Output: Der Ausgang "Zone bestätigen" wird verwendet, um die manuelle Steuerung bei geöffneten Schutztüren zu ermöglichen*  
 xGateSectionConfirm: **BOOL**;  
 */// Output: Der TCMZ-Fehlerausgang wird aktiviert, wenn ein Gate/Fortress-Fehler vorliegt*  
 xGateFortressFaultTcmz: **BOOL**;  
**END\_STRUCT**  
**END\_TYPE**

##### GVL\_<HwZone1>\_HMI

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*///*  
*/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL**  
 */// Hardware Zonen HMI Schnittstelle*  
 stHmiIf: stHmiHardwareZone;  
 */// Hardware zone automatic release request*  
 xAutomaticReleaseRequest: **BOOL**;  
**END\_VAR**

###### stHmiHardwareZone

**TYPE** stHmiHardwareZone :  
**STRUCT**  
 */// HMI Button aktiviere/deaktiviere EOC Mode*   
 xHmiPbEoc: **BOOL**;   
 */// HMI Anzeige EOC Mode ist ein*  
 xHmiEocModeOn: **BOOL**;  
 */// HMI Anzeige EOC Mode ist aus*  
 xHmiEocModeOff: **BOOL**;  
**END\_STRUCT**  
**END\_TYPE**

##### GVL\_<HwZone1>\_IOs

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*///*  
*/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL**  
 */// Input: Handbetrieb aktiv*  
 xManualModeKeySwitch: **BOOL** := TRUE;  
 */// Input: Automatikbetrieb aktiv*  
 xAutomaticModeKeySwitch: **BOOL**;  
 */// Input: Sicherheitsfunktion aktiv*   
 xSafetyFunctionKeySwitch: **BOOL**;  
 */// Input: Rückmeldekontakt von Sicherheitsrelais*  
 xEmergencyStopRelay: **BOOL**;  
 */// Input: Fehler Reset*  
 xFaultReset: **BOOL**;  
 */// Input: Freigabe Automatik*  
 xRelease: **BOOL**;  
 */// Input: Lampen Test*  
 xLampTest: **BOOL**;  
 */// Output: Lampe Automatikbetrieb ein/aktiv*   
 xAutoLamp: **BOOL**;  
 */// Output: Lampe Sicherheitseinrichtung offen/aktiv*  
 xSafetyGateLamp: **BOOL**;  
 */// Output: Lampe Not-Aus*   
 xEmergencyStopLamp: **BOOL**;  
 */// Output: Lampe End of Cycle aktiv/erreicht*  
 xEocLamp: **BOOL**;  
 */// Output: globaler Fehler Reset*  
 xGlobalFaultReset: **BOOL**;  
**END\_VAR**

##### GVL\_<HwZone1>\_Parameters

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*{attribute 'qualified\_only'}*  
*{attribute 'symbol' := 'readwrite'}*  
**VAR\_GLOBAL** PERSISTENT  
 *///<Parameter 1>*  
 <iParameter1>: <**INT**>;  
**END\_VAR**  
  
**VAR\_GLOBAL**  
 */// Empty VAR\_GLOBAL | Workaround for PersistentVars limitations*  
 \_placeholder: **BOOL**;  
**END\_VAR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

##### <HwZone1>

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*{attribute 'symbol' := 'readwrite'}*  
**PROGRAM** <HwZone1>  
**VAR**  
**END\_VAR**

**Hardware Zone call**

Dieser SPS-Code ruft die Hardware Zonen Objekte auf.

*// Hardware zone object call*  
 <HwZone1>\_TCMZ();  
 <HwZone1>\_Control();

**Step Sequence program call**

Hier werden alle in der Hardware Zone definierten Sequenzen aufgerufen.

*// ----------------------------- <Sequence1> -----------------------------*  
 *// Input mapping*  
 <Sequence1>\_InputMapping();  
   
 *// Sequence program call*  
 <Sequence1>();  
   
 *// Output mapping*  
 <Sequence1>\_OutputMapping();

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

##### <HwZone1>\_Control

*/// copyright SELMO Technology GmbH by SELMOstudio*  
*/// Version 2023.1.0.7650*   
*/// This function has been automatically generated.*  
*{attribute 'symbol' := 'none'}*  
**PROGRAM** <HwZone1>\_Control  
**VAR**  
 iIndex: **INT**;  
 aAutoModeSelectMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 aManualModeSelectMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 aFaultResetMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 aReleaseMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 aAnySeqReady2StartMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 aAllSeqReady2StartMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 aAllSeqAutoRelMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 aAllSeqEOCMatrix: **ARRAY**[0..0] **OF** **BOOL**;  
 fbRTrigPbEoc: R\_TRIG;  
 fbTonGlobalRelease: TON;  
 fbTonStartUpAlarm: TOF;  
**END\_VAR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Mode Selection AUTOMATIC / MANUAL

Automatic Mode

Die Automatik ist nur aktiviert, wenn alle für die Hardware-Zone relevanten Modus-Schlüsselschalter auf Automatik eingestellt sind.

*// Automatic Mode*  
 *{region "Description Mode Selection AUTOMATIC"}*  
 *(\**  
 *Automatic is only enabled when all mode keyswitches relevant to the hardware zone are selected to automatic.*  
 *\*)*  
 *{endregion}*  
 aAutoModeSelectMatrix[0] := GVL\_<HwZone1>\_IOs.xAutomaticModeKeySwitch **AND** **NOT** GVL\_<HwZone1>\_IOs.xManualModeKeySwitch;  
  
 GVL\_<HwZone1>.stHwzIf.xAutomaticModeOn := TRUE;  
 **FOR** iIndex := 0 **TO** 0 **DO**   
 **IF** **NOT** aAutoModeSelectMatrix[iIndex] **THEN**  
 GVL\_<HwZone1>.stHwzIf.xAutomaticModeOn := FALSE;  
 **EXIT**;  
 **END\_IF**  
 **END\_FOR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Manual Mode

Manuell ist aktiviert, wenn alle für die Hardware-Zone relevanten Modus-Schlüsselschalter wie unten dargestellt auf manuell eingestellt sind.

*// Manual Mode*  
 *{region "Description Mode Selection MANUAL"}*  
 *(\**  
 *Manual is enabled when any mode keyswitches relevant to the hardware zone are selected to manual as shown below.*  
 *\*)*  
 *{endregion}*  
 aManualModeSelectMatrix[0] := GVL\_<HwZone1>\_IOs.xManualModeKeySwitch **AND** **NOT** GVL\_<HwZone1>\_IOs.xAutomaticModeKeySwitch ;  
  
 **FOR** iIndex := 0 **TO** 0 **DO**   
 **IF** aManualModeSelectMatrix[iIndex] **THEN**  
 GVL\_<HwZone1>.stHwzIf.xManualModeOn := TRUE;  
 **EXIT**;  
 **ELSE**  
 GVL\_<HwZone1>.stHwzIf.xManualModeOn := FALSE;  
 **END\_IF**  
 **END\_FOR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Auto / Manual Mode Selected

Der folgende SPS-Code bestimmt, ob sich der Status der Hardware-Zone im automatischen oder im manuellen Modus befindet.

*// Auto / Manual Mode Selected*  
 *{region "Description Mode"}*  
 *(\**  
 *The following plc code determines if the status of the Hardware zone is in Automatic or in Manual mode.*  
 *\*)*  
 *{endregion}*  
  
 **IF** GVL\_<HwZone1>.stHwzIf.xManualModeOn **AND** **NOT** GVL\_<HwZone1>.stHwzIf.xAutomaticModeOn **THEN**  
 GVL\_<HwZone1>.stHwzIf.xAutomaticSelected := FALSE;  
 GVL\_<HwZone1>.stHwzIf.xManualSelected := TRUE;  
 **ELSIF** GVL\_<HwZone1>.stHwzIf.xAutomaticModeOn **AND** **NOT** GVL\_<HwZone1>.stHwzIf.xManualModeOn **THEN**  
 GVL\_<HwZone1>.stHwzIf.xAutomaticSelected := TRUE;  
 GVL\_<HwZone1>.stHwzIf.xManualSelected := FALSE;  
 **END\_IF**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Fault Reset

Fault Reset Push Button pressed

Das Drücken eines beliebigen Fehler Reset-Taste innerhalb der Zone, aktiviert den Ausgang Fehler Reset Taste gedrückt.

*// Fault Reset*  
 *{region "Description Fault Reset"}*  
 *(\**  
 *Pressing any reset p/b within the zone activates the reset p/b pressed flag.*  
 *\*)*  
 *{endregion}*  
 aFaultResetMatrix[0] := GVL\_<HwZone1>\_IOs.xFaultReset;  
  
 **FOR** iIndex := 0 **TO** 0 **DO**   
 **IF** aFaultResetMatrix[iIndex] **THEN**  
 GVL\_<HwZone1>.stHwzIf.xFaultResetPbPressed := TRUE;  
 **EXIT**;  
 **ELSE**  
 GVL\_<HwZone1>.stHwzIf.xFaultResetPbPressed := FALSE;  
 **END\_IF**  
 **END\_FOR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

**Zone** **Global Fault Reset**

Der globale Fehler Reset-Ausgang summiert alle Hardwarezonen Fehler Reset-Tasten (Global Utilities) und schaltet den Zonen Fehler Reset-Ausgang ein. Durch die Verwendung des globalen Fehler Reset-Flags auf diese Weise kann der Fehler-Reset einer beliebigen Zone zum Zurücksetzen von Fehlern in allen Zonen verwendet werden.

*// Global Fault Reset Zone*  
 *{region "Description Global Fault Reset"}*  
 *(\**  
 *The global fault reset sums together all zone reset p/b pressed flags (Global Utilities) and energises the reset output.*  
 *By using the global fault reset flag in this manner, the fault reset from any zone can be used to reset faults in all zones.*  
 *\*)*  
 *{endregion}*  
 GVL\_<HwZone1>\_IOs.xGlobalFaultReset := GVL\_Global.stGlobalIf.xFaultReset;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Release Pushbutton Pressed

Das Drücken einer beliebigen Freigabe-Taste innerhalb der Zone aktiviert den Ausgang "Freigabe gedrückt".

*// --------------------------------Release Pushbutton Pressed--------------------------------*  
 *{region "Description Release Pushbutton Pressed"}*  
 *(\**  
 *Pressing any release p/b within the zone activates the release p/b pressed flag.*  
 *\*)*  
 *{endregion}*  
 aReleaseMatrix[0] := GVL\_<HwZone1>\_IOs.xRelease;  
  
 **FOR** iIndex := 0 **TO** 0 **DO**   
 **IF** aReleaseMatrix[iIndex] **THEN**  
 GVL\_<HwZone1>.stHwzIf.xReleasePbPressed := TRUE;  
 **EXIT**;  
 **ELSE**  
 GVL\_<HwZone1>.stHwzIf.xReleasePbPressed := FALSE;  
 **END\_IF**  
 **END\_FOR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Any Step Sequence Ready2Start

Der folgende SPS-Code zeigt an, ob eine Sequenz, die mit dieser Hardwarezone verbunden ist, startbereit ist, d.h. kein Interlock oder C.M.Z.-Fehler aktiv sind.

*// --------------------------------Any Step Sequences Ready2Start--------------------------------*  
 *{region "Description Any Step Sequences Ready2Start"}*  
 *(\**  
 *The following plc code indicates if ANY sequence associated with that hardware zone is ready to start. i.e.*   
 *has no interlock or CMZ fault active.*  
 *\*)*  
 *{endregion}*  
 aAnySeqReady2StartMatrix[0] := GVL\_<Sequence1>.stSeqIf.xNoCMZFault **AND** GVL\_<Sequence1>.stSeqIf.xSeqNoInterlockFault;  
  
 **FOR** iIndex := 0 **TO** 0 **DO**   
 **IF** aAnySeqReady2StartMatrix[iIndex]**THEN**  
 GVL\_<HwZone1>.stHwzIf.xAnySeqReadyToStart := TRUE;  
 **EXIT**;  
 **ELSE**  
 GVL\_<HwZone1>.stHwzIf.xAnySeqReadyToStart := FALSE;  
 **END\_IF**  
 **END\_FOR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

All Step Sequence Ready2Start

Der folgende SPS-Code wird verwendet, um zu prüfen, ob alle mit der Zone verbundenen Sequenzen fehlerfrei und somit startbereit sind. Für diese Funktion wird ein wiederverwendbarer Ausgang zugewiesen. Dieser wird später für die automatische Statuslampenanzeige verwendet.

*// --------------------------------All Step Sequences Ready2Start--------------------------------*  
 *{region "Description All Step Sequences Ready2Start"}*  
 *(\**  
 *The plc code below are used to check if all sequences associated with the zone have no faults and hence are ready to start.*   
 *A reusable output is allocated for this function. This is used later for the automatic status lamp indicator.*  
 *\*)*  
 *{endregion}*  
 aAllSeqReady2StartMatrix[0] := GVL\_<Sequence1>.stSeqIf.xSeqCmzFault **OR** GVL\_<Sequence1>.stSeqIf.xSeqInterlockFault;  
  
 GVL\_<HwZone1>.stHwzIf.xAllSeqReadyToStart := TRUE;  
 **FOR** iIndex := 0 **TO** 0 **DO**   
 **IF** aAllSeqReady2StartMatrix[iIndex] **THEN**  
 GVL\_<HwZone1>.stHwzIf.xAllSeqReadyToStart := FALSE;  
 **EXIT**;  
 **END\_IF**  
 **END\_FOR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Start-up Alarm

Um alle Sequenzen, die mit dieser Zone verbunden sind, zu starten, muss der Freigabetaster gedrückt und alle Tore für die Zone geschlossen sein. Wenn ein Tor geöffnet wurde, muss der Freigabetaster für fünf Sekunden gedrückt gehalten werden, bevor der Ausgang ZONE RELEASE aktiviert wird. Während dieser Verzögerung, wenn ein Tor zuvor geöffnet wurde und alle damit verbundenen Sequenzen bereit sind zu starten, wird während dem Gedrückt halten des Freigabetasters der Startalarm aktiviert und dieser wird dann dazu verwendet eine Sirene zu betätigen. Der Freigabetaster muss die Dauer der Verzögerungszeit des Timers für die Sequenzen, die mit dieser Zone verbunden sind, gedrückt gehalten werden, damit diese starten können. Wenn der Freigabetaster losgelassen wird, wird der Timer zurückgesetzt und der Ausgang ZONE RELEASE wird nicht aktiviert.

*// --------------------------------Start-up Alarm--------------------------------*  
 *{region "Description Start-up Alarm"}*  
 *(\**  
 *To start all sequences associated with this zone, the release pushbutton must be pressed, and all gates for the zone closed.*   
 *If a gate has been opened, the release pushbutton must be held in for five seconds before the ZONE RELEASE output is activated.*   
 *During this delay, if a gate had previously been opened and any associated sequences are ready to start, while the pushbutton is pressed the Start-up alarm is enabled,*   
 *this is then used to operate a siren. The Release pushbutton must be held in for the duration of the delay timer for the sequences associated with this zone to start.*   
 *If the pushbutton is released, the timer is reset and the ZONE RELEASE output will not be activated.*  
 *\*)*  
 *{endregion}*  
   
 fbTonGlobalRelease(IN:= **NOT** GVL\_<HwZone1>.stHwzIf.xGateFortressFault **AND** GVL\_<HwZone1>.stHwzIf.xAutomaticModeOn **AND** GVL\_<HwZone1>.stHwzIf.xAnySeqReadyToStart **AND** GVL\_Global.stGlobalIf.xRelease ,   
 PT:= GVL\_Global.stGlobalIf.timGlobalPresetDelay, Q=> GVL\_<HwZone1>.stHwzIf.xAutoReleaseRequest);  
 GVL\_<HwZone1>.stHwzIf.xStartUpAlarm := **NOT** fbTonGlobalRelease.Q **AND** GVL\_<HwZone1>.stHwzIf.xAutomaticModeOn **AND** GVL\_<HwZone1>.stHwzIf.xAnySeqReadyToStart **AND** GVL\_Global.stGlobalIf.xRelease;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

All Step Sequence Automatic Release

Der nachstehende SPS-Code wird verwendet, um zu prüfen, ob alle mit der Zone verbundenen Sequenzen eine automatische Freigabe haben und somit alle gestartet wurden. Für diese Funktion ist ein wiederverwendbarer Ausgang vorgesehen. Dieser wird unten für die Anzeige der automatischen Statuslampe verwendet.

*// --------------------------------All Step Sequences Automatic Release--------------------------------*  
 *{region "Description All Step Sequences Automatic Release"}*  
 *(\**  
 *The plc code below are used to check if all sequences associated with the zone have automatic release and hence have all started.*   
 *A reusable output is allocated for this function. This is used below for the automatic status lamp indicator.*  
 *\*)*  
 *{endregion}*  
 aAllSeqAutoRelMatrix[0] := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased;  
  
 GVL\_<HwZone1>.stHwzIf.xAllSeqAutoRelease := TRUE;  
 **FOR** iIndex := 0 **TO** 0 **DO**   
 **IF** **NOT** aAllSeqAutoRelMatrix[iIndex] **THEN**  
 GVL\_<HwZone1>.stHwzIf.xAllSeqAutoRelease := FALSE;  
 **EXIT**;  
 **END\_IF**  
 **END\_FOR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Automatic Lamp Indication

Der folgende SPS-Code wird verwendet, um die automatischen Startlampen für diese Zone zu aktivieren. Wenn ALLE zugehörigen Sequenzen startbereit sind, blinken die Lampen. Wenn alle zugehörigen Sequenzen eine automatische Freigabe haben, leuchten die Lampen kontinuierlich, um anzuzeigen, dass alle zugehörigen Sequenzen in Ordnung sind. Der Lampentestkontakt wird einfach verwendet, um zu testen, ob die Lampen funktionieren.

*// --------------------------------Automatic Lamp Indication--------------------------------*  
 *{region "Description Automatic Lamp Indication"}*  
 *(\**  
 *The following PLC code is used to activate the start automatic lamps for that zone.*   
 *If ALL the associated sequences are ready to start then the lamps will flash.*   
 *If All the associated sequences have automatic release then the lamps will be on continuously to indicate that all associated sequences are ok.*   
 *The lamp test contact is simply used to test if the lamps work.*  
 *\*)*  
 *{endregion}*

GVL\_<HwZone1>\_IOs.xAutoLamp := (GVL\_<HwZone1>.stHwzIf.xAllSeqReadyToStart AND GVL\_Global.stGlobalIf.xFlashOutput) OR GVL\_<HwZone1>.stHwzIf.xAllSeqAutoRelease OR GVL\_Global.stGlobalIf.xLampTest;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Voltage Release Section Confirmation

Open Safety Gate Confirmation

Der Zonenbestätigungsausgang wird verwendet, um die manuelle Steuerung bei geöffneten Schutztüren zu ermöglichen. Der Ausgang erregt ein "Confirm PLC"-Relais, das zusammen mit dem Relais für den Schlüsselschalter für die Sicherheitsfunktion die Aktivierung des Spannungsauslösers bei geöffneten Toren ermöglicht.

*// Safety Gate Confirmation*  
 *{region "Description Voltage Release Section Confirmation"}*  
 *(\**  
 *The zone confirm output is used to enable manual control with safety gates open.*  
 *The ouput energises a ‘Confirm PLC’ relay which together with the relay for Safety function keyswitch operated allows the Voltage Release to energise with gates open.*  
 *\*)*  
 *{endregion}*  
  
 GVL\_<HwZone1>.stHwzIf.xGateSectionConfirm := (GVL\_<HwZone1>.stHwzIf.xReleasePbPressed **OR** GVL\_<HwZone1>.stHwzIf.xGateSectionConfirm) **AND** GVL\_<HwZone1>\_IOs.xSafetyFunctionKeySwitch **AND**  
 GVL\_<HwZone1>.stHwzIf.xGateFortressFault **AND** GVL\_<HwZone1>.stHwzIf.xManualModeOn ;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Gate/Fortress Fault detect

Der folgende SPS-Code wird verwendet, um das T.C.M.Z auf Torstörungen zu überprüfen. Um eine manuelle Bewegung bei geöffneten Toren zu ermöglichen, muss der Gate Fortress Fault Ausgang gesperrt werden. Dies wird erreicht, indem die Bestätigung der geöffneten Schutztür (aus dem vorherigen SPS-Code) verwendet wird, um das Gate Fortress Fault Signal zu maskieren, das an alle Sequenzen innerhalb der Zone gesendet wird, in der eine manuelle Funktion ausgeführt wird.

*// Gate/Fortress Fault Detect*  
 *{region "Description Gate/Fortress Fault Detect"}*  
 *(\**  
 *The following plc code is used to check the TCMZ for gate faults.*  
 *In order to allow manual movement with gates open, it is necessary to inhibit the gate fault output.*   
 *This is achieved by using the zone confirm flags (from the previous plc code) to mask the TCMZ gate/fault signal sent to all sequences within the zone in which a manual function is being actioned.*   
 *\*)*  
 *{endregion}*  
   
 GVL\_<HwZone1>.stHwzIf.xGateFortressFaultTcmz := GVL\_<HwZone1>.stHwzIf.xGateFortressFault **AND** **NOT** GVL\_<HwZone1>.stHwzIf.xGateSectionConfirm;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

T.C.M.Z Fault

Der T.C.M.Z.-Fehlerausgang wird aktiviert, wenn ein schwerwiegender Fehler oder ein Gate/Fortress-Fehler vorliegt.

*// TCMZ Fault Detect*  
 *{region "Description TCMZ Fault Detect"}*  
 *(\**  
 *The TCMZ Fault output is activated when a Fatal Fault or Gate/Fortress Fault Exists.*   
 *\*)*  
 *{endregion}*  
   
 GVL\_<HwZone1>.stHwzIf.xTcmzFault := GVL\_<HwZone1>.stHwzIf.xFatalFault **OR** GVL\_<HwZone1>.stHwzIf.xGateFortressFault;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Safety Gate Lamp Indication

Der folgende SPS-Code aktiviert die für die Zone konfigurierten Schutzeinrichtungs-Lampen.

*// --------------------------------Safety Gate Lamp Indication--------------------------------*  
 *{region "Description Safety Gate Lamp Indication"}*  
 *(\**  
 *The following PLC code activates the Safety Gate Lamps which are configured for the zone.*  
 *\*)*  
 *{endregion}*  
  
 GVL\_<HwZone1>\_IOs.xSafetyGateLamp := GVL\_<HwZone1>.stHwzIf.xGateFortressFault **OR** GVL\_Global.stGlobalIf.xLampTest;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Emergency Stop Lamp Indication

Der folgende SPS-Code aktiviert die Not-Aus-Lampen, die für die Zone konfiguriert sind.

*// --------------------------------Emergency Stop Lamp Indication--------------------------------*  
 *{region "Description Emergency Stop Lamp Indication"}*  
 *(\**  
 *The following PLC code activates the Emergency Stop Lamps which are configured for the zone.*  
 *\*)*  
 *{endregion}*  
   
 GVL\_<HwZone1>\_IOs.xEmergencyStopLamp := GVL\_<HwZone1>\_IOs.xEmergencyStopRelay **OR** GVL\_Global.stGlobalIf.xLampTest;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Stop at End of Cycle

Stop at End of Cycle Button Pressed

Dieser SPS-Code erkennt, wenn von einer beliebigen HMI innerhalb der Hardware-Zone eine Anforderung zum Anhalten am End of Cycle Schritt erforderlich ist.

*// Stop at End Of Cycle Button Pressed*   
 *{region "Description Stop at End Of Cycle Button Pressed"}*  
 *(\**  
 *This plc code detects when a Stop at End Of Cycle request has been made from any HMI within the hardware zone.*  
 *\*)*  
 *{endregion}*  
   
 fbRTrigPbEoc(CLK := GVL\_<HwZone1>\_HMI.stHmiIf.xHmiPbEoc);

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Stop at End of Cycle Mode

Bei diesem SPS-Code handelt es sich um eine Standard-Umschaltverriegelung, die es ermöglicht, einen einzelnen HMI-Taster zum Verriegeln/Entriegeln des Stop at End of Cycle Mode zu verwenden.

*// Stop at End Of Cycle Mode*  
 *{region "Description Stop at End Of Cycle Mode"}*  
 *(\**  
 *This plc code is a standard toggle latch which allows a single HMI pushbutton to be used*   
 *to latch /unlatch the stop at End Of Cycle mode.*  
 *\*)*  
 *{endregion}*  
   
 **IF** GVL\_<HwZone1>.stHwzIf.xAutomaticModeOn **AND** fbRTrigPbEoc.Q **AND** **NOT** GVL\_<HwZone1>\_HMI.stHmiIf.xHmiEocModeOn **THEN**   
 GVL\_<HwZone1>\_HMI.stHmiIf.xHmiEocModeOn := TRUE;  
 GVL\_<HwZone1>\_HMI.stHmiIf.xHmiEocModeOff := FALSE;  
 **ELSIF** GVL\_<HwZone1>.stHwzIf.xAutomaticModeOn **AND** fbRTrigPbEoc.Q **AND** **NOT** GVL\_<HwZone1>\_HMI.stHmiIf.xHmiEocModeOff **THEN**   
 GVL\_<HwZone1>\_HMI.stHmiIf.xHmiEocModeOn := FALSE;  
 GVL\_<HwZone1>\_HMI.stHmiIf.xHmiEocModeOff := TRUE;  
 **END\_IF**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

All Step Sequences End of Cycle Reached

Der folgende SPS-Code wird verwendet, um zu prüfen, ob alle mit der Zone verbundenen Sequenzen ihren End of Cycle Schritt erreicht haben. Für diese Funktion wird ein wiederverwendbarer Ausgang zugewiesen. Dieser wird unten für die Anzeige der End of Cycle-Lampe verwendet.

*// All Step Sequences End Of Cycle Reached*  
 *{region "Description Stop at End Of Cycle Mode"}*  
 *(\**  
 *The plc code below are used to check if all sequences associated with the zone have reached their end of cycle.*   
 *A reusable output is allocated for this function. This is used below for the End of cycle lamp indicator.*  
 *\*)*  
 *{endregion}*  
 aAllSeqEOCMatrix[0] := GVL\_<Sequence1>.stSeqIf.xSeqEndOfCycleReached;  
  
   
 GVL\_<HwZone1>.stHwzIf.xAllSeqEocReached := TRUE;  
 **FOR** iIndex := 0 **TO** 0 **DO**   
 **IF** **NOT** aAllSeqEOCMatrix[iIndex] **THEN**  
 GVL\_<HwZone1>.stHwzIf.xAllSeqEocReached := FALSE;  
 **EXIT**;  
 **END\_IF**  
 **END\_FOR**

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Stop At End of Cycle Lamp Indication

Der SPS-Code wird verwendet, um die End of Cycle-Lampen für diese Zone zu aktivieren. Wenn der Modus End of Cycle eingeschaltet ist, blinken die Lampen. Wenn alle zugehörigen Sequenzen ihren End of Cycle Schritt erreicht haben, leuchten die Lampen kontinuierlich. Der Lampentestkontakt wird verwendet, um zu testen, ob die Lampen funktionieren.

*// Stop At End Of Cycle Lamp Indication*  
 *{region "Description Stop at End Of Cycle Mode"}*  
 *(\**  
 *This is used below for the End of cycle lamp indicator.*  
 *It is used to activate the End Of Cycle lamps for that zone. If End Of Cycle mode is ON then the End Of Cycle lamps will flash.*   
 *If All the associated sequences have reached their End Of Cycle, the lamps will be on continuously.*   
 *The lamp test contact is simply used to test if the lamps work.*  
 *\*)*  
 *{endregion}*  
   
 GVL\_<HwZone1>\_IOs.xEocLamp := (GVL\_<HwZone1>\_HMI.stHmiIf.xHmiEocModeOn **AND** GVL\_Global.stGlobalIf.xFlashOutput) **OR** GVL\_<HwZone1>.stHwzIf.xAllSeqEocReached **OR** GVL\_Global.stGlobalIf.xLampTest;

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

##### TCMZ

Eine Total constantly monitored Zone (TCMZ) in Hardware Zonen bezieht sich auf einen bestimmten Bereich in der Industrieanlage, der ständig überwacht wird, um sicherzustellen, dass die Maschine oder das System ordnungsgemäß funktioniert. Dies kann mithilfe von Sensoren oder anderen Überwachungstechnologien erfolgen, die im Bereich installiert sind und Daten über den Betrieb der Maschine sammeln. Die Überwachung der TCMZ kann dazu beitragen, mögliche Probleme frühzeitig zu erkennen und zu beheben, bevor sie zu Ausfällen oder Schäden führen, und somit die Betriebssicherheit und Zuverlässigkeit der Maschine erhöhen.

Jedes TCMZ ist in 3 definierte Bereiche unterteilt: Fatal Fault, Gate/Fortress Fault und Warning Fault.

###### <HwZone1>\_TCMZ

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7650

/// This function has been automatically generated.

{attribute 'symbol' := 'readwrite'}

PROGRAM HwZone1\_TCMZ

VAR

{attribute 'symbol' := 'none'}

fbFatalFault: FB\_CMZFault;

aFatalFault: ARRAY[0..2] OF BOOL;

{attribute 'symbol' := 'none'}

fbGateFortressFault: FB\_CMZFault;

aGateFortressFault: ARRAY[0..1] OF BOOL;

{attribute 'symbol' := 'none'}

fbWarningFault: FB\_CMZFault;

aWarningFault: ARRAY[0..3] OF BOOL;

END\_VAR

Fatal Fault

Der Merker für Fatal Fault ist ein SPS-Ausgang, der den Status des Bereichs für schwerwiegende Fehler in der TCMZ-Matrix der Hardwarezone überwacht, so dass jedes Bit für schwerwiegende Fehler zum Aktivieren des Ausgangs führt. Dieses Flag wird in allen Step Sequence-Fehlermatrizen verwendet, um einen CMZ-Fehler zu verursachen und die automatische/manuelle Freigabe in jeder Sequenz innerhalb der Hardware-Zone zu verhindern. Eine Standard-Index in jeder Sequenzfehler-Matrix ist für diesen Fehler vorgesehen. Der fatale Fehler muss behoben werden, bevor eine automatische oder manuelle Freigabe möglich ist. Schwerwiegende Fehler können nicht übersteuert werden.

// ###################### fatal faults ######################

// TCMZ assigns

aFatalFault[0] := GVL\_Global.stGlobalIf.xFatalFault;

aFatalFault[1] := GVL\_HwZone1\_FatalFaults.<xCMZ\_1>;

aFatalFault[2] := GVL\_HwZone1\_FatalFaults.<xCMZ\_2>;

// TCMZ call

fbFatalFault(aFaultMatrix := aFatalFault, xFaultActive => GVL\_HwZone1.stHwzIf.xFatalFault);

Gate/Fortress Fault

Das Gate/Fortress-Fault Flag ist ein SPS-Ausgang, der den Status des Gate/Fortress Fault-Bereichs der TCMZ-Matrix der Hardwarezone überwacht, so dass jedes aktivierte Gate/Fortress Fault-Bit zum Aktivieren des Gate/Fortress Fault Flag Ausgangs führt. Dieses Flag wird in allen Step Sequence-Fehlermatrizen verwendet, um einen CMZ-Fehler zu verursachen und die automatische/manuelle Freigabe in jeder Sequenz innerhalb der Hardware-Zone zu verhindern.

// ###################### gate / fortress faults ######################

// TCMZ assigns

aGateFortressFault[0] := GVL\_Global.stGlobalIf.xGateFortressFault;

aGateFortressFault[1] := GVL\_HwZone1\_GateFortressFaults.<xCMZ\_1>;

// TCMZ call

fbGateFortressFault(aFaultMatrix := aGateFortressFault, xFaultActive => GVL\_HwZone1.stHwzIf.xGateFortressFault);

Warning Fault

Für jede Hardwarezone gibt es eine zugehörige Warning Fault Matrix. Die Warning Fault haben keinen Einfluss auf den Maschinenbetrieb. Der Status der Warning Fault Matrix wird jedoch ständig von der "HMI" überwacht, die alle aktiven Warnmeldungen anzeigt.

// ###################### warning faults ######################

// TCMZ assigns

aWarningFault[0] := GVL\_Global.stGlobalIf.xWarningFault;

aWarningFault[1] := GVL\_HwZone1\_WarningFaults.<xCMZ\_1>;

aWarningFault[2] := GVL\_HwZone1\_WarningFaults.<xCMZ\_2>;

aWarningFault[3] := GVL\_HwZone1\_WarningFaults.<xCMZ\_3>;

// TCMZ call

fbWarningFault(aFaultMatrix := aWarningFault, xFaultActive => GVL\_HwZone1.stHwzIf.xWarningFault);

###### GVL\_<HwZone1>\_FatalFaults

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7650

/// This function has been automatically generated.

///

/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###

{attribute 'qualified\_only'}

{attribute 'symbol' := 'readwrite'}

VAR\_GLOBAL

///<CMZ 1>

<xCMZ\_1>: <BOOL>;

///<CMZ 2>

<xCMZ\_2>: <BOOL>;

END\_VAR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

###### GVL\_<HwZone1>\_GateFortressFaults

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7650

/// This function has been automatically generated.

///

/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###

{attribute 'qualified\_only'}

{attribute 'symbol' := 'readwrite'}

VAR\_GLOBAL

///<CMZ 1>

<xCMZ\_1>: <BOOL>;

END\_VAR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

###### GVL\_<HwZone1>\_WarningFaults

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7650

/// This function has been automatically generated.

///

/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###

{attribute 'qualified\_only'}

{attribute 'symbol' := 'readwrite'}

VAR\_GLOBAL

///<CMZ 1>

<xCMZ\_1>: <BOOL>;

///<CMZ 2>

<xCMZ\_2>: <BOOL>;

///<CMZ 3>

<xCMZ\_3>: <BOOL>;

END\_VAR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

##### <Sequence1>

Eine Step Sequence ist eine Reihe von Schritten, die in einer bestimmten Reihenfolge ausgeführt werden. Sie kann verwendet werden, um eine Aufgabe oder Prozess zu strukturieren und zu vereinfachen. Jeder Schritt in der Sequenz beschreibt eine spezifische Aktion, die ausgeführt werden muss, um die Aufgabe abzuschließen. Jedes Teil der Maschine, das einen unabhängigen Bewegungsablauf hat, erfordert ein eigenes Step Sequence-Programm mit einem eigenen Step Sequence-Bereich.

###### GVL\_<Sequence1>

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7660

/// This function has been automatically generated.

{attribute 'qualified\_only'}

{attribute 'symbol' := 'readwrite'}

VAR\_GLOBAL

///Sequence Schnittstelle

stSeqIf: stSequenceInterface := (iNumberOfInputs := 10 ,iNumberOfOutputs := 8 ,iSeqEndOfCycleStep := 16);

///Sequence Schrittüberwachung

aStepMonitoringMatrix: ARRAY[1..16] OF stStepMonitoringMatrix := [

(timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>), (timStepTimeoutSet := <T#300000MS>, timStepTimeoutAdd := <T#5000MS>, xDisableTimeout := <FALSE>)];

END\_VAR

{attribute 'qualified\_only'}

VAR\_GLOBAL PERSISTENT

///Output: aktueller Schritt

iStepcounter: INT := 1;

///Output: Schritt voriger Zyklus

iStepcounterLastCycle: INT := 1;

END\_VAR

{attribute 'qualified\_only'}

VAR\_GLOBAL CONSTANT

///Input: letzter Schritt

iEndStep: INT := 16;

END\_VAR

stSequenceInterface

TYPE stSequenceInterface :

STRUCT

///Output: Sequence CMZ Fehler ist aktiv

xSeqCmzFault: BOOL;

///Output: Sequence Automatik Modus ist gewählt

xSeqAutomaticModeSelected: BOOL;

///Output: Sequence Manueller Modus ist gewählt

xSeqManualModeSelected: BOOL;

///Output: Sequence Automatik ist freigegeben

xSeqAutomaticReleased: BOOL;

///Output: Sequence Handbetrieb ist freigegeben

xSeqManualReleased: BOOL;

///Output: Sequence kein Taster wurde gedrückt

xSeqNoPBpressed: BOOL;

///Output: Sequence Einzellschritt Tigger erkannt

xSeqSingleStepOnTrigEvent: BOOL;

///Output: Sequence EOC Schritt erreicht

xSeqEndOfCycleReached: BOOL;

///Output: Sequence Interlock Fehler vorhanden

xSeqInterlockFault: BOOL;

///Output: Sequence Interlock Fehler nicht vorhanden

xSeqNoInterlockFault: BOOL;

///Output: Sequence Matrix ist derzeit nicht null, ein oder mehrere nicht erlaubte Maschinenzustände (Bits) sind vorhanden

xSeqMatrixNoZero: BOOL;

///Output: Sequence kein CMZ Fehler vorhanden

xNoCMZFault: BOOL;

///Output: Sequence Timeout in einem Schritt aufgetreten

xSeqStepTimeout: BOOL;

///Output: Sequence Schritt Monitoring Zeiten einlernen aktiv

xSeqStepTimeTeachModeOn: BOOL;

///Output: Sequence Schritt Monitoring Zeiten einlernen inaktiv

xSeqStepTimeTeachModeOff: BOOL;

///Input: Sequence konfigurierter EOC Schritt

iSeqEndOfCycleStep: INT;

///Output: Sequence Anzahl der Eingangszonen

iNumberOfInputs: INT;

///Output: Sequence Anzahl der Ausgangszonen

iNumberOfOutputs: INT;

END\_STRUCT

END\_TYPE

stStepMonitoringMatrix

TYPE stStepMonitoringMatrix :

STRUCT

///Output: aktuelle Schrittzeit

timStepTimeActual :TIME;

///Output: maximal aufgetretene Schrittzeit

timStepTimeMax :TIME;

///Output: Durchschnittliche Schrittzeit von 5 Schrittzyklen

timStepTimeAvg :TIME;

///Output: minimal aufgetretene Schrittzeit

timStepTimeMin :TIME;

///Output: zuletzt ermittelte Schrittzeit

timStepTimeLast :TIME;

///Output: Zähler für Schrittzeitüberschreitungen

diStepTimeoutCount :DINT;

///Input: berechneter oder manuell eingestellter Sollwert des Timeouts für den Schritt

timStepTimeoutSet :TIME :=T#300S;

///Input: zusätzlicher Wert für die berechnete Zeitüberschreitung

timStepTimeoutAdd :TIME :=T#5S;

///Input: Schrittzeitüberwachung ausschalten

xDisableTimeout :BOOL;

///Output: Zeitüberschreitung aufgetreten

xTimeoutOccurred :BOOL;

END\_STRUCT

END\_TYPE

###### GVL\_<Sequence1>\_CMZ

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7660

/// This function has been automatically generated.

///

/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###

{attribute 'qualified\_only'}

{attribute 'symbol' := 'none'}

VAR\_GLOBAL

///<CMZ 1>

<xCMZ\_1>: <BOOL>;

///<CMZ 2>

<xCMZ\_2>: <BOOL>;

///<CMZ 3>

<xCMZ\_3>: <BOOL>;

///<CMZ 4>

<xCMZ\_4>: <BOOL>;

END\_VAR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

###### GVL\_<Sequence1>\_HMI

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7660

/// This function has been automatically generated.

///

/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###

{attribute 'qualified\_only'}

{attribute 'symbol' := 'readwrite'}

VAR\_GLOBAL

///Sequence HMI Schnittstelle

stHmiIf: stHmiSequence;

///Bit-Array, automatisch generiert von SELMOstudio

aHmi: ARRAY[0..46] OF BOOL;

///Bit-Array, automatisch generiert von SELMOstudio

aHmiStored: ARRAY[0..46] OF BOOL;

///Bit-Array, automatisch generiert von SELMOstudio

aHmiMon: ARRAY[0..0] OF BOOL;

///Bit-Überwachungspuffer-Array, automatisch generiert von SELMOstudio

aHmiMonBuffer: ARRAY[0..0] OF stMonitoring;

///no Manual button text

xManBtn\_Zone\_9: BOOL;

///no Manual button text

xManBtn\_Zone\_10: BOOL;

///no Manual button text

xManBtn\_Zone\_11: BOOL;

///no Manual button text

xManBtn\_Zone\_12: BOOL;

///Diese Variable deaktiviert die Schaltfläche 'Einzelschritt-Modus'

xDisableSingleStepOn: BOOL;

///Diese Variable deaktiviert die Schaltfläche 'Schritt Inkrementieren'

xDisableStepIncrement: BOOL;

///Diese Variable deaktiviert die Schaltfläche 'Schritt zurücksetzen'

xDisableStepReset: BOOL;

///Diese Variable deaktiviert die Schaltfläche 'Sequenz Automatik-Freigabe'

xDisableAutomaticRelease: BOOL;

///Diese Variable deaktiviert die Schaltfläche 'Handebtrieb'

xDisableManualModePanel: BOOL;

///Diese Variable deaktiviert die Schaltfläche 'Parameter'

xDisableParameterPanel: BOOL;

///Diese Variable deaktiviert die Schaltfläche 'Schrittzeit-Überwachung'

xDisableStepTimeMonitoringPanel: BOOL;

///Diese Variable deaktiviert die Schaltfläche 'Diagnose'

xDisableDiagnosePanel: BOOL;

END\_VAR

stHmiSequence

TYPE stHmiSequence :

STRUCT

/// HMI Button Automatik freigeben

xHmiPbAutomaticRelease: BOOL;

/// HMI Anzeige Automatik wurde freigeben

xHmiMsgAutomaticReleased: BOOL;

/// HMI Button Schritt auf 1 stellen

xHmiPbReset: BOOL;

/// HMI Button Schritt um 1 erhöhen

xHmiPbStepIncrement: BOOL;

/// HMI Button Schritt um 1 verringern

xHmiPbStepDecrement: BOOL;

/// HMI Button nächsten zulässigen Schritt suchen

xHmiPbStepSearch: BOOL;

/// HMI Button Einzelschritt Modus ein

xHmiPbSingleStepOn: BOOL;

/// HMI Button Einzelschritt

xHmiPbSingleStep: BOOL;

/// HMI Anzeige Interlock Fehler aktiv

xHmiMsgInterlockFault: BOOL;

/// HMI Button Fehler quittieren

xHmiPbFaultReset: BOOL;

/// HMI Button Schritt Monitoring Zeiten einlernen freigeben

xHmiPbStepTimeTeachMode: BOOL;

/// HMI Anzeige Schritt Monitoring Zeiten einlernen aktiv

xHmiMsgStepTimeTeachModeOn: BOOL;

/// HMI Anzeige Schritt Monitoring Überwachung hat ausgelöst

xHmiMsgStepTimeout: BOOL;

/// HMI Anzeige Schritt Monitoring Überwachung hat ausgelöst und bleibt solange aktiv bis Manuell zurückgesetzt wird

xHmiMsgStepTimeoutStored: BOOL;

/// HMI Anzeige aktuelle Schrittnummer

iHmiMsgStepCounter: INT;

/// HMI Anzeige aktueller Zyklus (wird gelöscht bei jeder Freigabe der Automatik)

uiHmiMsgSequenceCycleCounter: UINT;

END\_STRUCT

END\_TYPE

stMonitoring

TYPE stMonitoring :

STRUCT

///Output: Speicher bereits zugewiesen

xActive :BOOL;

///Output: Gespeicherte Zonennummer, bei der die Überwachungsprüfung ausgelöst wurde

iZoneNo :INT;

///Output: Gespeicherte Schrittnummer, bei der die Überwachungsprüfung ausgelöst wurde

iStepNo :INT;

///Output: Gespeicherter Zeitstempel, zu dem die Überwachungsprüfung ausgelöst wurde

aTimeStamp :stMonitoringTimeStamp;

END\_STRUCT

END\_TYPE

stMonitoringTimeStamp

TYPE stMonitoringTimeStamp :

STRUCT

/// Date Time of timestamp

dtTimeStamp : DT;

/// Time of Day of timestamp

todTimeStamp : TIME\_OF\_DAY;

/// Date of timestamp

datTimeStamp : DATE;

END\_STRUCT

END\_TYPE

###### GVL\_<Sequence1>\_IOs

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7660

/// This function has been automatically generated.

///

/// ### PLEASE DO NOT MAKE ANY CHANGES MANUALLY HERE! Make the changes only in SELMOstudio ###

{attribute 'qualified\_only'}

{attribute 'symbol' := 'readwrite'}

VAR\_GLOBAL

///<Eingang: keine Beschreibung>

<i\_xDecision\_1Path1>: <BOOL>;

///<Eingang: keine Beschreibung>

<i\_xDecision\_1Path2>: <BOOL>;

///<Eingang: keine Beschreibung>

<i\_xZone7>: <BOOL>;

///<Eingang: keine Beschreibung>

<i\_xZone9>: <BOOL>;

///<Eingang: keine Beschreibung>

<i\_xZone10>: <BOOL>;

///<Eingang: keine Beschreibung>

<o\_xZone\_1>: <BOOL>;

///<Eingang: keine Beschreibung>

<o\_xZone9>: <BOOL>;

///<Eingang: keine Beschreibung>

<o\_xZone10>: <BOOL>;

///<Ausgang: keine Beschreibung>

<o\_xZone11>: <BOOL>;

///<Ausgang: keine Beschreibung>

<o\_xZone12>: <BOOL>;

///<Ausgang: keine Beschreibung>

<o\_xZone\_15RepeaterIteration1>: <BOOL>;

///<Ausgang: keine Beschreibung>

<o\_xZone\_17RepeaterIteration2>: <BOOL>;

END\_VAR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

###### GVL\_<Sequence1>\_Parameters

/// copyright SELMO Technology GmbH by SELMOstudio

/// Version 2023.1.0.7660

/// This function has been automatically generated.

{attribute 'qualified\_only'}

{attribute 'symbol' := 'readwrite'}

VAR\_GLOBAL PERSISTENT

///<Parameter 1>

<iParameter1>: <INT>;

///<Parameter 2>

<iParameter2>: <INT>;

END\_VAR

VAR\_GLOBAL

/// Empty VAR\_GLOBAL | Workaround for PersistentVars limitations

\_placeholder: BOOL;

END\_VAR

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

###### <Sequence1>

Die Programmstruktur einer einzelnen Step Sequence besteht aus vier grundlegenden Abschnitten.

1. **Standard Beginning**

2. **Sequence Logic Control**

3. **Sequence C.M.Z (Constantly Monitored Zone)**

4. **Standard End**

Jeder Abschnitt bietet eine bestimmte Funktion.

Das komplette Step Sequence-Programm wurde auf diese Weise unterteilt, um eine logische Organisation der einzelnen Funktionen zu gewährleisten, die zusammen eine komplette Step Sequence ergeben.

Variablen Deklaration

Bei der Definition von Schritten und Zonen werden automatisch die benötigten Variablen deklariert.

 Generierte Deklaration

|  |
| --- |
| /// copyright SELMO Technology GmbH by SELMOstudio  /// Version 2023.1.0.7800  /// This function has been automatically generated.  {attribute 'symbol' := 'readwrite'}  PROGRAM Sequence1  VAR  {attribute 'symbol' := 'none'}  aMatrix: ARRAY[0..48] OF BOOL;  {attribute 'symbol' := 'none'}  aTemp: ARRAY[0..48] OF BOOL;  {attribute 'symbol' := 'none'}  aStep: ARRAY[0..48] OF BOOL;  aPb: ARRAY[0..19] OF BOOL;  aCmz: ARRAY[0..5] OF BOOL;  aZ1: ARRAY[1..16] OF eZoneOperand;  aZ2: ARRAY[1..16] OF eZoneOperand;  aZ3: ARRAY[1..16] OF eZoneOperand;  aZ4: ARRAY[1..16] OF eZoneOperand;  aZ5: ARRAY[1..16] OF eZoneOperand;  aZ6: ARRAY[1..16] OF eZoneOperand;  aZ7: ARRAY[1..16] OF eZoneOperand;  aZ8: ARRAY[1..16] OF eZoneOperand;  aZ9: ARRAY[1..16] OF eZoneOperand;  aZ10: ARRAY[1..16] OF eZoneOperand;  aZ11: ARRAY[1..16] OF eZoneOperand;  aZ12: ARRAY[1..16] OF eZoneOperand;  aZ13: ARRAY[1..16] OF eZoneOperand;  aZ14: ARRAY[1..16] OF eZoneOperand;  aZ15: ARRAY[1..16] OF eZoneOperand;  aZ16: ARRAY[1..16] OF eZoneOperand;  aZ17: ARRAY[1..16] OF eZoneOperand;  aZ18: ARRAY[1..16] OF eZoneOperand;  aZ19: ARRAY[1..16] OF eZoneOperand;  {attribute 'symbol' := 'none'}  fbCmzFault: FB\_CmzFault;  fbStandardBegin: FB\_SeqStandardBegin;  {attribute 'symbol' := 'none'}  fbStandardEnd: FB\_SeqStandardEnd;  {attribute 'symbol' := 'none'}  aStepMonitoringFb: ARRAY[1..16] OF stStepMonitoringFB;  {attribute 'symbol' := 'none'}  fbZone1: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone2: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone3: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone4: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone5: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone6: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone7: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone8: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone9: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone10: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone11: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone12: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone13: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone14: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone15: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone16: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone17: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone18: FB\_Zone;  {attribute 'symbol' := 'none'}  fbZone19: FB\_Zone;  nHmiStoredLoopCounter: INT;  fbTon2: TON;  {attribute 'symbol' := 'none'}  aCrossZone1: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone1: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone1: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone2: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone2: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone2: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone3: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone3: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone3: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone4: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone4: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone4: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone5: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone5: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone5: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone6: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone6: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone6: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone7: ARRAY[0..1] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone7: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone7: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone8: ARRAY[0..1] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone8: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone8: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone9: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone9: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone9: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone10: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone10: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone10: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone11: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aCondZone12: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aCrossZone13: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone13: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone13: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone14: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone14: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone14: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone15: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone15: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone15: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone16: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone16: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone16: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone17: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone17: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone17: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone18: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone18: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone18: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCrossZone19: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  aCondZone19: ARRAY[0..0] OF BOOL := [0(TRUE)];  {attribute 'symbol' := 'none'}  aPaircheckZone19: ARRAY[0..0] OF BOOL;  {attribute 'symbol' := 'none'}  fbErrorSetCtrl0: FB\_ErrorSetCtrl;  {attribute 'symbol' := 'none'}  fbErrorSetCtrl1: FB\_ErrorSetCtrl;  {attribute 'symbol' := 'none'}  fbErrorSetCtrl2: FB\_ErrorSetCtrl;  {attribute 'symbol' := 'none'}  fbErrorSetCtrl3: FB\_ErrorSetCtrl;  {attribute 'symbol' := 'none'}  fbErrorSetCtrl4: FB\_ErrorSetCtrl;  {attribute 'symbol' := 'none'}  fbErrorSetCtrl5: FB\_ErrorSetCtrl;  END\_VAR |

HMI Buttons

In diesem Abschnitt werden die [HMI-Manuell-Buttons](#GVL_Sequence1_HMI) der Zonen der Push-Button-Matrix zugewiesen. Dies ermöglicht die manuelle Bedienung der Zone, wenn sie im Studio als HMI-Button aktiviert wurde.

// --------------------------- HMI buttons ---------------------------

aPb[8] := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_9;

aPb[9] := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_10;

aPb[10] := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_11;

aPb[11] := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_12;

Standard Beginning

Der Funktionsblock fbStandardBegin bestimmt die Startbedingungen für die Schrittfolge und überwacht den sicheren Status, um den automatischen oder manuellen Betrieb zu ermöglichen. Dieser Funktionsblock ist ein Hauptbestandteil jeder Sequence und befindet sich in der Selmo Basic Bibliothek.

**refHwzHmi** ist die Reference(Pointer)zu der Variablenliste "[GVL\_<HwZone1>\_HMI.stHmiIf](#StHmiHardwareZone)"

**refHwzIf** ist die Reference(Pointer)zu der Variablenliste "[GVL\_<HwZone1>.stHwzIf](#StHardwareZoneInterface)"

**refSeqHmi** ist die Reference(Pointer)zu der Variablenliste "[GVL\_<Sequence1>\_HMI.stHmiIf](#StHmiSequence2)"

**refSeqIf** ist die Reference(Pointer)zu der Variablenliste "[GVL\_<Sequence1>.stSeqIf](#StSequenceInterface)"

**xSeqSafetyModeOn** Abfrage des Sicherheitsschlüsselschalter "[GVL\_<HwZone1>\_IOs.xSafetyFunctionKeySwitch](#GVL_HwZone1_IOs)"

**iSeqStepcounter** aktueller Schrittzähler der Sequence "[GVL\_<Sequence1>.iStepcounter](#GVL_Sequence1)"

**aSeqPb** [Array der Handbetrieb Push buttons](#Sequence1_2_HMIButtonsC2DC6D10)wird um einen Index je aktiven HMI - Button je Zone erhöht

**aLampMatrix** [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

**aStepMatrix** [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

**aTempMatrix** [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

// ########################### STANDARD BEGINNING ###########################

{region "Description Standard Beginning"}

(\*

The Standard Beginning determines the step sequence start up conditions and

monitors the safe status of the step sequence to allow Automatic or Manual operation

\*)

{endregion}

fbStandardBegin(

refHwzHmi := GVL\_<HwZone1>\_HMI.stHmiIf,

refHwzIf := GVL\_<HwZone1>.stHwzIf,

refSeqHmi:= GVL\_<Sequence1>\_HMI.stHmiIf,

refSeqIf := GVL\_<Sequence1>.stSeqIf,

xSeqSafetyModeOn := GVL\_<HwZone1>\_IOs.xSafetyFunctionKeySwitch,

iSeqStepcounter := GVL\_<Sequence1>.iStepcounter,

aSeqPb := aPb,

aLampMatrix := aMatrix,

aStepMatrix := aStep,

aTempMatrix := aTemp);

Decisions

In diesem Bereich wird entschieden, welcher Abschnitt der Step Sequence, der zuvor im Studio konfiguriert wurde, abgearbeitet wird. Wenn sich die Sequence im definierten Decision-Schritt befindet, wird erwartet, dass eine Entscheidung getroffen wird. Dies führt dazu, dass der entsprechende Decision-Pfad ausgewählt wird und die weiteren Schrittfolgen festgelegt werden, die dann abgearbeitet werden.

// --------------------------- Decisions ---------------------------

// Decision Decision 1 Path 1 End Jump

IF (GVL\_<Sequence1>.iStepCounterLastCycle = 4 AND GVL\_<Sequence1>.iStepCounter = 5 AND GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased) THEN

GVL\_<Sequence1>.iStepCounter := 6;

END\_IF

// Decision Decision 1 Path 2 End Jump

IF (GVL\_<Sequence1>.iStepCounterLastCycle = 5 AND GVL\_<Sequence1>.iStepCounter = 6 AND GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased) THEN

GVL\_<Sequence1>.iStepCounter := 6;

END\_IF

// Decision Jumps Decision 1

IF (GVL\_<Sequence1>.iStepCounter = 3 AND GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased) THEN

IF (GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path1) AND NOT (GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path2) THEN

GVL\_<Sequence1>.iStepCounter := 4;

ELSIF (GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path2) AND NOT (GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path1) THEN

GVL\_<Sequence1>.iStepCounter := 5;

END\_IF

END\_IF

// Decision Jumps Decision 1 Iteration 2

IF (GVL\_<Sequence1>.iStepCounter = 8 AND GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased) THEN

IF (GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path1) AND NOT (GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path2) THEN

GVL\_<Sequence1>.iStepCounter := 9;

ELSIF (GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path2) AND NOT (GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path1) THEN

GVL\_<Sequence1>.iStepCounter := 10;

END\_IF

END\_IF

Conditinal Jumps

Conditional Jumps sind Anweisungen in einem SPS Programm, die die Ausführung des Programms an eine andere Stelle verzweigen lassen, abhängig von einer bestimmten Bedingung. Sie werden auch als "bedingte Verzweigungen" oder "bedingte Sprünge" bezeichnet.

// --------------------------- Conditional jumps ---------------------------

IF GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased AND GVL\_<Sequence1>.iStepCounter = 7 AND () THEN

GVL\_<Sequence1>.iStepCounter := 2;

END\_IF

IF GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased AND GVL\_<Sequence1>.iStepCounter = 12 AND () THEN

GVL\_<Sequence1>.iStepCounter := 2;

END\_IF

Sequence Logic Control

Der Abschnitt Sequence Logic Control der PLC sorgt für die eigentliche Steuerung und Überwachung jeder einzelnen Maschinenbewegung/Aktion innerhalb der Schrittfolge. Dieser Bereich ist in "Sequenz Zonen" unterteilt. Eine "Sequence Zone" ist für die Steuerung und Überwachung einer einzelnen Aktion (z.B. Motor ein) zuständig. Daher kann jeder Schritt einer Sequenz eine oder mehrere 'Sequence Zones' enthalten. Unter dem Einfluss des Schrittzählers setzt die "Sequenzzone" die relevanten Ausgänge der realen Welt und überwacht die entsprechenden Eingänge der realen Welt, die mit der jeweiligen Bewegung oder Aktion verbunden sind.

Operands

Die Sequence Logic Control ist die Grundlage für die Steuerung und Überwachung des automatischen Betriebs der Maschine.

Für jede Zone gibt es ein Array (aZ1, aZ2, …) welches für jeden Schritt (Index) den Operanden vom System-Layer abbildet. Bei der Zuweisung gibt es folgende Werte:

0=don’t Care (keine Aktion der Zone erfolgt)

1=Interlock Check (I - Überwachung des Eingangssignals der Zone mit Unterbrechung der Automatik)

2=Sequence Check (S - Änderung des Eingangssignal und Ansteuerung des Ausgangssignal der Zone)

3=Monitoring Check (M - Überwachung des Eingangssignals der Zone ohne Unterbrechung der Automatik)

// ########################### SEQUENCE LOGIC CONTROL ###########################

{region "Description Sequence Logic Control"}

(\*

The Sequence Logic Control section of the PLC software provides the actual control and

monitoring of every individual machine movement/action within the step sequence.

This section is divided into ‘Sequence Zones’.

A ‘Sequence zone’ is responsible for the control and monitoring of an individual action(e.g.Clamp Forward).

Therefore each step of a sequence can contain one or more ‘Sequence Zones’.

Under the influence of the step counter, the ‘Sequence Zone’ will set relevant real world outputs and

monitor the corresponding real world inputs associated with the particular movement or operation.

\*)

{endregion}

// Zones Operands Assigns

aZ1[01]:= 0 ; aZ2[01]:= 0 ; aZ3[01]:= 0 ; aZ4[01]:= 0 ; aZ5[01]:= 0 ; aZ6[01]:= 0 ; aZ7[01]:= 0 ; aZ8[01]:= 1 ; aZ9[01]:= 1 ; aZ10[01]:= 0 ; aZ11[01]:= 0 ; aZ12[01]:= 0 ; aZ13[01]:= 0 ; aZ14[01]:= 0 ; aZ15[01]:= 0 ; aZ16[01]:= 0 ; aZ17[01]:= 0 ; aZ18[01]:= 0 ; // Step Name: Step 1

aZ1[02]:= 2 ; aZ2[02]:= 0 ; aZ3[02]:= 0 ; aZ4[02]:= 0 ; aZ5[02]:= 0 ; aZ6[02]:= 0 ; aZ7[02]:= 0 ; aZ8[02]:= 1 ; aZ9[02]:= 0 ; aZ10[02]:= 2 ; aZ11[02]:= 0 ; aZ12[02]:= 2 ; aZ13[02]:= 0 ; aZ14[02]:= 0 ; aZ15[02]:= 0 ; aZ16[02]:= 0 ; aZ17[02]:= 0 ; aZ18[02]:= 0 ; // Step Name: Step 2

aZ1[03]:= 0 ; aZ2[03]:= 0 ; aZ3[03]:= 0 ; aZ4[03]:= 0 ; aZ5[03]:= 0 ; aZ6[03]:= 2 ; aZ7[03]:= 0 ; aZ8[03]:= 0 ; aZ9[03]:= 0 ; aZ10[03]:= 1 ; aZ11[03]:= 2 ; aZ12[03]:= 0 ; aZ13[03]:= 0 ; aZ14[03]:= 0 ; aZ15[03]:= 2 ; aZ16[03]:= 0 ; aZ17[03]:= 0 ; aZ18[03]:= 1 ; // Step Name: Decision 1

aZ1[04]:= 0 ; aZ2[04]:= 1 ; aZ3[04]:= 0 ; aZ4[04]:= 0 ; aZ5[04]:= 1 ; aZ6[04]:= 0 ; aZ7[04]:= 2 ; aZ8[04]:= 0 ; aZ9[04]:= 0 ; aZ10[04]:= 1 ; aZ11[04]:= 0 ; aZ12[04]:= 2 ; aZ13[04]:= 0 ; aZ14[04]:= 0 ; aZ15[04]:= 1 ; aZ16[04]:= 0 ; aZ17[04]:= 0 ; aZ18[04]:= 1 ; // Step Name: Step Path 1

aZ1[05]:= 0 ; aZ2[05]:= 0 ; aZ3[05]:= 1 ; aZ4[05]:= 1 ; aZ5[05]:= 0 ; aZ6[05]:= 0 ; aZ7[05]:= 0 ; aZ8[05]:= 2 ; aZ9[05]:= 0 ; aZ10[05]:= 1 ; aZ11[05]:= 2 ; aZ12[05]:= 0 ; aZ13[05]:= 0 ; aZ14[05]:= 0 ; aZ15[05]:= 1 ; aZ16[05]:= 0 ; aZ17[05]:= 0 ; aZ18[05]:= 1 ; // Step Name: Step Path 2

aZ1[06]:= 0 ; aZ2[06]:= 0 ; aZ3[06]:= 0 ; aZ4[06]:= 0 ; aZ5[06]:= 0 ; aZ6[06]:= 0 ; aZ7[06]:= 2 ; aZ8[06]:= 0 ; aZ9[06]:= 0 ; aZ10[06]:= 1 ; aZ11[06]:= 0 ; aZ12[06]:= 2 ; aZ13[06]:= 0 ; aZ14[06]:= 0 ; aZ15[06]:= 1 ; aZ16[06]:= 0 ; aZ17[06]:= 0 ; aZ18[06]:= 1 ; // Step Name: Step End

aZ1[07]:= 0 ; aZ2[07]:= 0 ; aZ3[07]:= 0 ; aZ4[07]:= 0 ; aZ5[07]:= 0 ; aZ6[07]:= 0 ; aZ7[07]:= 1 ; aZ8[07]:= 0 ; aZ9[07]:= 0 ; aZ10[07]:= 1 ; aZ11[07]:= 2 ; aZ12[07]:= 0 ; aZ13[07]:= 0 ; aZ14[07]:= 0 ; aZ15[07]:= 1 ; aZ16[07]:= 0 ; aZ17[07]:= 0 ; aZ18[07]:= 1 ; // Step Name: Jump 7

aZ1[08]:= 0 ; aZ2[08]:= 0 ; aZ3[08]:= 0 ; aZ4[08]:= 0 ; aZ5[08]:= 0 ; aZ6[08]:= 2 ; aZ7[08]:= 0 ; aZ8[08]:= 0 ; aZ9[08]:= 0 ; aZ10[08]:= 1 ; aZ11[08]:= 2 ; aZ12[08]:= 0 ; aZ13[08]:= 0 ; aZ14[08]:= 0 ; aZ15[08]:= 1 ; aZ16[08]:= 0 ; aZ17[08]:= 2 ; aZ18[08]:= 0 ; // Step Name: Decision 1 Iteration 2

aZ1[09]:= 0 ; aZ2[09]:= 1 ; aZ3[09]:= 0 ; aZ4[09]:= 0 ; aZ5[09]:= 1 ; aZ6[09]:= 0 ; aZ7[09]:= 2 ; aZ8[09]:= 0 ; aZ9[09]:= 0 ; aZ10[09]:= 1 ; aZ11[09]:= 0 ; aZ12[09]:= 2 ; aZ13[09]:= 0 ; aZ14[09]:= 0 ; aZ15[09]:= 1 ; aZ16[09]:= 0 ; aZ17[09]:= 1 ; aZ18[09]:= 0 ; // Step Name: Step Path 1 Iteration 2

aZ1[10]:= 0 ; aZ2[10]:= 0 ; aZ3[10]:= 1 ; aZ4[10]:= 1 ; aZ5[10]:= 0 ; aZ6[10]:= 0 ; aZ7[10]:= 0 ; aZ8[10]:= 2 ; aZ9[10]:= 0 ; aZ10[10]:= 1 ; aZ11[10]:= 2 ; aZ12[10]:= 0 ; aZ13[10]:= 0 ; aZ14[10]:= 0 ; aZ15[10]:= 1 ; aZ16[10]:= 0 ; aZ17[10]:= 1 ; aZ18[10]:= 0 ; // Step Name: Step Path 2 Iteration 2

aZ1[11]:= 0 ; aZ2[11]:= 0 ; aZ3[11]:= 0 ; aZ4[11]:= 0 ; aZ5[11]:= 0 ; aZ6[11]:= 0 ; aZ7[11]:= 2 ; aZ8[11]:= 0 ; aZ9[11]:= 0 ; aZ10[11]:= 1 ; aZ11[11]:= 0 ; aZ12[11]:= 2 ; aZ13[11]:= 0 ; aZ14[11]:= 0 ; aZ15[11]:= 1 ; aZ16[11]:= 0 ; aZ17[11]:= 1 ; aZ18[11]:= 0 ; // Step Name: Step End Iteration 2

aZ1[12]:= 0 ; aZ2[12]:= 0 ; aZ3[12]:= 0 ; aZ4[12]:= 0 ; aZ5[12]:= 0 ; aZ6[12]:= 0 ; aZ7[12]:= 1 ; aZ8[12]:= 0 ; aZ9[12]:= 0 ; aZ10[12]:= 1 ; aZ11[12]:= 2 ; aZ12[12]:= 0 ; aZ13[12]:= 0 ; aZ14[12]:= 0 ; aZ15[12]:= 1 ; aZ16[12]:= 0 ; aZ17[12]:= 1 ; aZ18[12]:= 0 ; // Step Name: Jump 7 Iteration 2

aZ1[13]:= 0 ; aZ2[13]:= 0 ; aZ3[13]:= 0 ; aZ4[13]:= 0 ; aZ5[13]:= 0 ; aZ6[13]:= 0 ; aZ7[13]:= 1 ; aZ8[13]:= 0 ; aZ9[13]:= 0 ; aZ10[13]:= 1 ; aZ11[13]:= 0 ; aZ12[13]:= 2 ; aZ13[13]:= 0 ; aZ14[13]:= 0 ; aZ15[13]:= 0 ; aZ16[13]:= 2 ; aZ17[13]:= 0 ; aZ18[13]:= 2 ; // Step Name: Repeater 8

aZ1[14]:= 0 ; aZ2[14]:= 0 ; aZ3[14]:= 0 ; aZ4[14]:= 0 ; aZ5[14]:= 0 ; aZ6[14]:= 0 ; aZ7[14]:= 0 ; aZ8[14]:= 2 ; aZ9[14]:= 2 ; aZ10[14]:= 0 ; aZ11[14]:= 2 ; aZ12[14]:= 0 ; aZ13[14]:= 0 ; aZ14[14]:= 0 ; aZ15[14]:= 0 ; aZ16[14]:= 0 ; aZ17[14]:= 0 ; aZ18[14]:= 0 ; // Step Name: Step 9

aZ1[15]:= 0 ; aZ2[15]:= 0 ; aZ3[15]:= 0 ; aZ4[15]:= 0 ; aZ5[15]:= 0 ; aZ6[15]:= 0 ; aZ7[15]:= 0 ; aZ8[15]:= 0 ; aZ9[15]:= 0 ; aZ10[15]:= 0 ; aZ11[15]:= 0 ; aZ12[15]:= 0 ; aZ13[15]:= 2 ; aZ14[15]:= 0 ; aZ15[15]:= 0 ; aZ16[15]:= 0 ; aZ17[15]:= 0 ; aZ18[15]:= 0 ; // Step Name: CrossSeq Master Setter 10

aZ1[16]:= 0 ; aZ2[16]:= 0 ; aZ3[16]:= 0 ; aZ4[16]:= 0 ; aZ5[16]:= 0 ; aZ6[16]:= 0 ; aZ7[16]:= 0 ; aZ8[16]:= 0 ; aZ9[16]:= 0 ; aZ10[16]:= 0 ; aZ11[16]:= 0 ; aZ12[16]:= 0 ; aZ13[16]:= 0 ; aZ14[16]:= 2 ; aZ15[16]:= 0 ; aZ16[16]:= 0 ; aZ17[16]:= 0 ; aZ18[16]:= 0 ; // Step Name: CrossSeq Master Getter 10

Zone Call

In diesem Bereich der Sequence werden die Zonen(**fbZone**) die zuvor im Selmo Studio definiert wurden aufgerufen. Jede Zone besitzt eine eigne Instanz mit dem Namen **fbZone1**..\* Dieser ist für jede Zone, unabhängig vom Typ, notwendig. Je nach verwendete Zone im Selmo-Studio, werden die Properties dementsprechend beschrieben.

Folgende Bereiche werden in Abhängigkeit der Zone und Parametrierung wie folgt generiert:

MXIC Parameter

**aCrossZone\*[\*]** Array wird generiert, wenn eine Zonenübergreifende Verriegelung zugewiesen wird. Die Zuweisung erfolgt über die HMI Button "[GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_11](#GVL_Sequence1_HMI)"

**aCondZone\*[\*]** Diese Array dient zur Zonenübergreifende Verriegelung. Die Verriegelung erfolgt über das Feedback der zu überwachenden Zone "[GVL\_<Sequence1>\_IOs.i\_xZone7](#GVL_Sequence1_IOs)"

Paircheck Parmeter

**aPaircheckZone\*[\*]** In diesem Array werden Zonen zugewiesen die für die Pairchecküberwachung herangezogen werden. Der Paircheck erfolgt über das Feedback der zu überwachenden Zonen "[GVL\_<Sequence1>\_IOs.i\_xZone10](#GVL_Sequence1_IOs)"

Allgemeine Parameter der Zone

fbZone\*.**P\_iIndexOfLamp** beschreibt den Index der Zone im Array [aLampMatrix](#Sequence1_2_VariablenDeklaration8B3473F7). Dieser Index ist eindeutig

fbZone\*.**P\_iIndexOfPaircheckLamp**  beschreibt den Index der Zone im Array [aLampMatrix](#Sequence1_2_VariablenDeklaration8B3473F7) falls ein Paircheck aktiviert wurde. Dieser Index ist eindeutig

fbZone\*.**P\_xInputPolarity** legt fest, ob es um eine normale oder eine invertierte Zone handelt. D.h. Das Input Signal der Zone kann entweder True od. auf False abgefragt werden

fbZone\*.**P\_xIsSystemZone** wird für interne Zonen verwendet, welche z.B. bei einer Decision, Timer Step, Repeater usw. benötigt werden.

fbZone\*.**P\_xOutputOpposite** bewirkt eine Verriegelung der Outputs wenn diese in der gleichen Paircheck-Group sind.

fbZone\*(**xNoPBPressed**) Abfrage kein Taster wurde gedrückt "[GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed](#StSequenceInterface)"

fbZone\*(**xAutomaticRelease**) Abfrage Automatik ist freigegeben "[GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased](#StSequenceInterface)"

fbZone\*(**iActualStepCounter**) aktueller Schrittzähler der Sequence "[GVL\_<Sequence1>.iStepcounter](#GVL_Sequence1)"

fbZone\*(**aZone**) Für jede Zone gibt es ein Array ([aZ1, aZ2](#Sequence1_2_Operands17DB7351), …) welches für jeden Schritt (Index) den Operanden vom System-Layer abbildet

fbZone\*(**aLampMatrix**) [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

fbZone\*(**aTempMatrix**) [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

fbZone\*(**aMonMatrix**) [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) werden Daten für die zu Überwachende Zone gespeichert. "[GVL\_<Sequence1>\_HMI.aHmiMon](#GVL_Sequence1_HMI)"

fbZone\*(**aMonBuffer**) In diesem Array werden Daten für die zu Überwachenden Zonen gespeichert. "[GVL\_<Sequence1>\_HMI.aHmiMonBuffer](#GVL_Sequence1_HMI)"

fbZone\*(**aStepMatrix**) [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

Zonen Methoden Call

M\_Input

fbZone\*.M\_Input (**xFeedbackInput**) Das Feedback(Eingangssignal) repräsentiert den Zustand der Zone."[GVL\_<Sequence1>\_IOs.i\_xZone7](#GVL_Sequence1_IOs)",

fbZone\*.M\_Input (**timFeedbackInput**) Hier wird eine Zeitangabe verwendet, um das Feedback der Zone zu verzögern. (Bsp.: Endprellen von Sensorsignale)

fbZone\*.M\_Input(**xManualRelease**) Abfrage Handbetrieb ist freigegeben. "[GVL\_<Sequence1>.stSeqIf.xSeqManualReleased](#StSequenceInterface)"

fbZone\*.M\_Input(**aCross**) Wenn eine Zonenübergreifende Verriegelung zugewiesen wurde, wird der Zustand des Feedbacks durch Betätigung eines Handtaster und der zu überwachenden Zonen ausgegeben."[aCrossZone7](#Sequence1_2_aCrossZoneF93C5E86)"

fbZone\*.M\_Input(**aFeedbackPairCheckInput)** Wenn eine Paircheck Zone zugewiesen wurde, wird der Paircheck(Interlock) durch das Feedback der zu überwachenden Zonen aktiviert."[aPaircheckZone7](#Sequence1_2_aPaircheckZoneBA0D8A60)"

M\_InOutput

fbZone\*.**M\_InOutput** Hier wird der Rückgabewert der Methode dem Ausgang der Zone zugewiesen."[GVL\_<Sequence1>\_IOs.o\_xZone9](#GVL_Sequence1_IOs)"

fbZone\*.M\_InOutput(**xManualRelease**) Abfrage Handbetrieb ist freigegeben. "[GVL\_<Sequence1>.stSeqIf.xSeqManualReleased](#StSequenceInterface)"

fbZone\*.M\_InOutput (**xFeedbackInput**) Das Feedback(Eingangssignal) repräsentiert den Zustand der Zone."[GVL\_<Sequence1>\_IOs.i\_xZone9](#GVL_Sequence1_IOs)",

fbZone\*.M\_InOutput (**xManualPB**)Abfrage Handbetrieb Taste(HMI Button) der Zone. "[GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_9](#GVL_Sequence1_HMI)"

fbZone\*.M\_InOutput (**timFeedbackInput**) Hier wird eine Zeitangabe verwendet, um das Feedback der Zone zu verzögern. (Bsp.: Endprellen von Sensorsignale)

fbZone\*.M\_InOutput(**aCross**) Wenn eine Zonenübergreifende Verriegelung zugewiesen wurde, wird der Zustand des Feedbacks durch Betätigung eines Handtaster und der zu überwachenden Zonen ausgegeben."[aCrossZone9](#Sequence1_2_aCrossZoneF93C5E86)"

fbZone\*.M\_InOutput (**aCond**)Wenn eine Zonenübergreifende Verriegelung zugewiesen wurde, wird der Output der Zone(Rückgabewert der Methode) gesperrt, wenn die zu überwachenden Zonen nicht den erforderlichen Zustand besitzt. "[aCondZone9](#Sequence1_2_aCondZoneA779AADC)"

fbZone\*.M\_InOutput(**aFeedbackPairCheckInput)** Wenn eine Paircheck Zone zugewiesen wurde, wird der Paircheck(Interlock) durch das Feedback der zu überwachenden Zonen aktiviert."[aPaircheckZone9](#Sequence1_2_aPaircheckZoneBA0D8A60)"

M\_Output

fbZone\*.**M\_Output** Hier wird der Rückgabewert der Methode dem Ausgang der Zone zugewiesen."[GVL\_<Sequence1>\_IOs.o\_xZone12](#GVL_Sequence1_IOs)"

fbZone\*.M\_Output(**xManualRelease**) Abfrage Handbetrieb ist freigegeben. "[GVL\_<Sequence1>.stSeqIf.xSeqManualReleased](#StSequenceInterface)"

fbZone\*.M\_Output (**xManualPB**)Abfrage Handbetrieb Taste(HMI Button) der Zone. "[GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_12](#GVL_Sequence1_HMI)"

fbZone\*.M\_Output (**aCond**)Wenn eine Zonenübergreifende Verriegelung zugewiesen wurde, wird der Output der Zone(Rückgabewert der Methode) gesperrt, wenn die zu überwachenden Zonen nicht den erforderlichen Zustand besitzt. "[aCondZone12](#Sequence1_2_aCondZoneA779AADC)"

Timer Zone

In diesem Codeabschnitt wird der Funktionsbaustein fbTon2 für den Timer Step aufgerufen und die dafür konfigurierte Zeit übergeben.

// --------------------------- TIMER CALLS ---------------------------

fbTon2(IN:= GVL\_<Sequence1>.iStepCounter = 2 AND GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased, PT:= T#5000MS);

// --------------------------- ZONE 1 | Step 2 Timer ---------------------------

// Zone Condition Assigns

aCondZone1[0] := TRUE;

aPaircheckZone1[0] := FALSE;

fbZone1.P\_iIndexOfLamp := 11;

fbZone1.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone1.P\_xIsSystemZone := TRUE;

fbZone1( // Step 2 Timer

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ1,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

GVL\_<Sequence1>\_IOs.o\_xZone\_1 := fbZone1.M\_InOutput(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xFeedbackInput := fbTon2.Q,

xManualPB := FALSE,

timFeedbackInput := T#0MS,

aCross := aCrossZone1,

aCond := aCondZone1,

aFeedbackPairCheckInput := aPaircheckZone1);

Desicion Zone

// --------------------------- ZONE 2 | Decision\_1 Path 1 ---------------------------

// Zone Condition Assigns

aCondZone2[0] := TRUE;

aPaircheckZone2[0] := FALSE;

fbZone2.P\_iIndexOfLamp := 13;

fbZone2.P\_iIndexOfPaircheckLamp := 14;

fbZone2.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone2.P\_xIsSystemZone := TRUE;

fbZone2( // Decision\_1 Path 1

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ2,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

fbZone2.M\_Input(

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path1,

timFeedbackInput := T#0MS,

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

aCross := aCrossZone2,

aFeedbackPairCheckInput := aPaircheckZone2);

// --------------------------- ZONE 3 | Decision\_1 Path 1 Inv ---------------------------

// Zone Condition Assigns

aCondZone3[0] := TRUE;

aPaircheckZone3[0] := FALSE;

fbZone3.P\_iIndexOfLamp := 15;

fbZone3.P\_iIndexOfPaircheckLamp := 16;

fbZone3.P\_xInputPolarity := eInputPolarity.Active\_Low;

fbZone3.P\_xIsSystemZone := TRUE;

fbZone3( // Decision\_1 Path 1 Inv

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ3,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

fbZone3.M\_Input(

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path1,

timFeedbackInput := T#0MS,

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

aCross := aCrossZone3,

aFeedbackPairCheckInput := aPaircheckZone3);

// --------------------------- ZONE 4 | Decision\_1 Path 2 ---------------------------

// Zone Condition Assigns

aCondZone4[0] := TRUE;

aPaircheckZone4[0] := FALSE;

fbZone4.P\_iIndexOfLamp := 17;

fbZone4.P\_iIndexOfPaircheckLamp := 18;

fbZone4.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone4.P\_xIsSystemZone := TRUE;

fbZone4( // Decision\_1 Path 2

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ4,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

fbZone4.M\_Input(

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path2,

timFeedbackInput := T#0MS,

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

aCross := aCrossZone4,

aFeedbackPairCheckInput := aPaircheckZone4);

// --------------------------- ZONE 5 | Decision\_1 Path 2 Inv ---------------------------

// Zone Condition Assigns

aCondZone5[0] := TRUE;

aPaircheckZone5[0] := FALSE;

fbZone5.P\_iIndexOfLamp := 19;

fbZone5.P\_iIndexOfPaircheckLamp := 20;

fbZone5.P\_xInputPolarity := eInputPolarity.Active\_Low;

fbZone5.P\_xIsSystemZone := TRUE;

fbZone5( // Decision\_1 Path 2 Inv

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ5,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

fbZone5.M\_Input(

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path2,

timFeedbackInput := T#0MS,

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

aCross := aCrossZone5,

aFeedbackPairCheckInput := aPaircheckZone5);

// --------------------------- ZONE 6 | Decision\_1 XOR ---------------------------

// Zone Condition Assigns

aCondZone6[0] := TRUE;

aPaircheckZone6[0] := FALSE;

fbZone6.P\_iIndexOfLamp := 21;

fbZone6.P\_iIndexOfPaircheckLamp := 22;

fbZone6.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone6.P\_xIsSystemZone := TRUE;

fbZone6( // Decision\_1 XOR

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ6,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

fbZone6.M\_Input(

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path1 XOR GVL\_<Sequence1>\_IOs.i\_xDecision\_1Path2,

timFeedbackInput := T#0MS,

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

aCross := aCrossZone6,

aFeedbackPairCheckInput := aPaircheckZone6);

Input Zone

// --------------------------- ZONE 7 | Zone 7 ---------------------------

// Zone Cross Interlock Assigns

aCrossZone7[0] := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_11;

aCrossZone7[1] := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_12;

// Zone Condition Assigns

aCondZone7[0] := TRUE;

aPaircheckZone7[0] := FALSE;

fbZone7.P\_iIndexOfLamp := 23;

fbZone7.P\_iIndexOfPaircheckLamp := 24;

fbZone7.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone7( // Zone 7

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ7,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

fbZone7.M\_Input(

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xZone7,

timFeedbackInput := T#0MS,

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

aCross := aCrossZone7,

aFeedbackPairCheckInput := aPaircheckZone7);

// --------------------------- ZONE 8 | Zone 7 Inv ---------------------------

// Zone Cross Interlock Assigns

aCrossZone8[0] := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_9;

aCrossZone8[1] := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_10;

// Zone Condition Assigns

aCondZone8[0] := TRUE;

aPaircheckZone8[0] := FALSE;

fbZone8.P\_iIndexOfLamp := 25;

fbZone8.P\_iIndexOfPaircheckLamp := 26;

fbZone8.P\_xInputPolarity := eInputPolarity.Active\_Low;

fbZone8( // Zone 7 Inv

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ8,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

fbZone8.M\_Input(

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xZone7,

timFeedbackInput := T#0MS,

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

aCross := aCrossZone8,

aFeedbackPairCheckInput := aPaircheckZone8);

InOut Zone

// --------------------------- ZONE 9 | Zone 9 ---------------------------

// Zone Condition Assigns

aCondZone9[0] := NOT GVL\_<Sequence1>\_IOs.i\_xZone7;

aPaircheckZone9[0] := GVL\_<Sequence1>\_IOs.i\_xZone10;

fbZone9.P\_iIndexOfLamp := 27;

fbZone9.P\_iIndexOfPaircheckLamp := 28;

fbZone9.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone9.P\_xOutputOpposite := GVL\_<Sequence1>\_IOs.o\_xZone10;

fbZone9( // Zone 9

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ9,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

GVL\_<Sequence1>\_IOs.o\_xZone9 := fbZone9.M\_InOutput(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xZone9,

xManualPB := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_9,

timFeedbackInput := T#0MS,

aCross := aCrossZone9,

aCond := aCondZone9,

aFeedbackPairCheckInput := aPaircheckZone9);

// --------------------------- ZONE 10 | Zone 10 ---------------------------

// Zone Condition Assigns

aCondZone10[0] := NOT GVL\_<Sequence1>\_IOs.i\_xZone7;

aPaircheckZone10[0] := GVL\_<Sequence1>\_IOs.i\_xZone9;

fbZone10.P\_iIndexOfLamp := 29;

fbZone10.P\_iIndexOfPaircheckLamp := 30;

fbZone10.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone10.P\_xOutputOpposite := GVL\_<Sequence1>\_IOs.o\_xZone9;

fbZone10( // Zone 10

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ10,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

GVL\_<Sequence1>\_IOs.o\_xZone10 := fbZone10.M\_InOutput(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xFeedbackInput := GVL\_<Sequence1>\_IOs.i\_xZone10,

xManualPB := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_10,

timFeedbackInput := T#0MS,

aCross := aCrossZone10,

aCond := aCondZone10,

aFeedbackPairCheckInput := aPaircheckZone10);

Output Zone

// --------------------------- ZONE 11 | Zone 11 ---------------------------

// Zone Condition Assigns

aCondZone11[0] := GVL\_<Sequence1>\_IOs.i\_xZone7;

fbZone11.P\_iIndexOfLamp := 31;

fbZone11.P\_xOutputOpposite := GVL\_<Sequence1>\_IOs.o\_xZone12;

fbZone11( // Zone 11

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ11,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

GVL\_<Sequence1>\_IOs.o\_xZone11 := fbZone11.M\_Output(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xManualPB := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_11,

aCond := aCondZone11);

// --------------------------- ZONE 12 | Zone 12 ---------------------------

// Zone Condition Assigns

aCondZone12[0] := GVL\_<Sequence1>\_IOs.i\_xZone7;

fbZone12.P\_iIndexOfLamp := 33;

fbZone12.P\_xOutputOpposite := GVL\_<Sequence1>\_IOs.o\_xZone11;

fbZone12( // Zone 12

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ12,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

GVL\_<Sequence1>\_IOs.o\_xZone12 := fbZone12.M\_Output(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xManualPB := GVL\_<Sequence1>\_HMI.xManBtn\_Zone\_12,

aCond := aCondZone12);

Cross Sequence Zone

// --------------------------- ZONE 13 | CrossSeq\_Master\_Setter\_10 ---------------------------

// Zone Condition Assigns

aCondZone13[0] := TRUE;

aPaircheckZone13[0] := FALSE;

fbZone13.P\_iIndexOfLamp := 35;

fbZone13.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone13.P\_xIsSystemZone := TRUE;

fbZone13( // CrossSeq\_Master\_Setter\_10

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ13,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

IF fbZone13.M\_InOutput(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xFeedbackInput := GVL\_Global.xCS\_Sequence1\_CrossSeq\_Master\_Setter\_10,

xManualPB := FALSE,

timFeedbackInput := T#0MS,

aCross := aCrossZone13,

aCond := aCondZone13,

aFeedbackPairCheckInput := aPaircheckZone13) THEN

GVL\_Global.xCS\_Sequence1\_CrossSeq\_Master\_Setter\_10 := TRUE;

END\_IF

// --------------------------- ZONE 14 | CrossSeq\_Master\_Getter\_10 ---------------------------

// Zone Condition Assigns

aCondZone14[0] := TRUE;

aPaircheckZone14[0] := FALSE;

fbZone14.P\_iIndexOfLamp := 37;

fbZone14.P\_xInputPolarity := eInputPolarity.Active\_Low;

fbZone14.P\_iIndexOfPaircheckLamp := 38;

fbZone14.P\_xIsSystemZone := TRUE;

fbZone14( // CrossSeq\_Master\_Getter\_10

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ14,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

fbZone14.M\_Input(

xFeedbackInput := GVL\_Global.xCS\_Sequence1\_CrossSeq\_Master\_Setter\_10,

timFeedbackInput := T#0MS,

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

aCross := aCrossZone14,

aFeedbackPairCheckInput := aPaircheckZone14);

Repeater Zone

// --------------------------- ZONE 15 | Repeater 8 Iteration 1 ---------------------------

// Zone Condition Assigns

aCondZone15[0] := TRUE;

aPaircheckZone15[0] := FALSE;

fbZone15.P\_iIndexOfLamp := 39;

fbZone15.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone15.P\_xIsSystemZone := TRUE;

fbZone15( // Repeater 8 Iteration 1

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ15,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

IF fbZone15.M\_InOutput(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xFeedbackInput := GVL\_<Sequence1>\_IOs.o\_xZone\_15RepeaterIteration1,

xManualPB := FALSE,

timFeedbackInput := T#0MS,

aCross := aCrossZone15,

aCond := aCondZone15,

aFeedbackPairCheckInput := aPaircheckZone15) THEN

GVL\_<Sequence1>\_IOs.o\_xZone\_15RepeaterIteration1 := TRUE;

END\_IF

// --------------------------- ZONE 16 | Repeater 8 Iteration 1 Inv ---------------------------

// Zone Condition Assigns

aCondZone16[0] := TRUE;

aPaircheckZone16[0] := FALSE;

fbZone16.P\_iIndexOfLamp := 41;

fbZone16.P\_xInputPolarity := eInputPolarity.Active\_Low;

fbZone16.P\_xIsSystemZone := TRUE;

fbZone16( // Repeater 8 Iteration 1 Inv

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ16,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

IF fbZone16.M\_InOutput(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xFeedbackInput := GVL\_<Sequence1>\_IOs.o\_xZone\_15RepeaterIteration1,

xManualPB := FALSE,

timFeedbackInput := T#0MS,

aCross := aCrossZone16,

aCond := aCondZone16,

aFeedbackPairCheckInput := aPaircheckZone16) OR (NOT GVL\_<Sequence1>.stSeqIf.xSeqAutomaticModeSelected AND GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbReset) THEN

GVL\_<Sequence1>\_IOs.o\_xZone\_15RepeaterIteration1 := FALSE;

END\_IF

// --------------------------- ZONE 17 | Repeater 8 Iteration 2 ---------------------------

// Zone Condition Assigns

aCondZone17[0] := TRUE;

aPaircheckZone17[0] := FALSE;

fbZone17.P\_iIndexOfLamp := 43;

fbZone17.P\_xInputPolarity := eInputPolarity.Active\_High;

fbZone17.P\_xIsSystemZone := TRUE;

fbZone17( // Repeater 8 Iteration 2

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ17,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

IF fbZone17.M\_InOutput(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xFeedbackInput := GVL\_<Sequence1>\_IOs.o\_xZone\_17RepeaterIteration2,

xManualPB := FALSE,

timFeedbackInput := T#0MS,

aCross := aCrossZone17,

aCond := aCondZone17,

aFeedbackPairCheckInput := aPaircheckZone17) THEN

GVL\_<Sequence1>\_IOs.o\_xZone\_17RepeaterIteration2 := TRUE;

END\_IF

// --------------------------- ZONE 18 | Repeater 8 Iteration 2 Inv ---------------------------

// Zone Condition Assigns

aCondZone18[0] := TRUE;

aPaircheckZone18[0] := FALSE;

fbZone18.P\_iIndexOfLamp := 45;

fbZone18.P\_xInputPolarity := eInputPolarity.Active\_Low;

fbZone18.P\_xIsSystemZone := TRUE;

fbZone18( // Repeater 8 Iteration 2 Inv

xNoPBPressed := GVL\_<Sequence1>.stSeqIf.xSeqNoPBpressed,

xAutomaticRelease := GVL\_<Sequence1>.stSeqIf.xSeqAutomaticReleased,

iActualStepCounter := GVL\_<Sequence1>.iStepCounter,

aZone := aZ18,

aLampMatrix := aMatrix,

aTempMatrix := aTemp,

aMonMatrix := GVL\_<Sequence1>\_HMI.aHmiMon,

aMonBuffer := GVL\_<Sequence1>\_HMI.aHmiMonBuffer,

aStepMatrix := aStep);

IF fbZone18.M\_InOutput(

xManualRelease := GVL\_<Sequence1>.stSeqIf.xSeqManualReleased,

xFeedbackInput := GVL\_<Sequence1>\_IOs.o\_xZone\_17RepeaterIteration2,

xManualPB := FALSE,

timFeedbackInput := T#0MS,

aCross := aCrossZone18,

aCond := aCondZone18,

aFeedbackPairCheckInput := aPaircheckZone18) OR (NOT GVL\_<Sequence1>.stSeqIf.xSeqAutomaticModeSelected AND GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbReset) THEN

GVL\_<Sequence1>\_IOs.o\_xZone\_17RepeaterIteration2 := FALSE;

END\_IF

Sequence CMZ

Die Constantly Monitored Zone (oder 'C.M.Z') ähnelt der T.C.M.Z., enthält jedoch nur unsichere / schwerwiegende Maschinenfehler, die mit der einzelnen Schrittsequenz zusammenhängen.

**fbErrorSetCtr**\* Der Baustein ermöglich das Setzen und Rücksetzen von CMZ's

fbErrorSetCtrl\*.**P\_timErrorDelay** ermöglicht es einen Fehler zu verzögern, d.h. er muss min. für diese Zeit aktiv sein bevor der Fehler gesetzt wird

fbErrorSetCtrl\*.**P\_xAutoReset** ermöglicht es einen Fehler sobald er behoben wurde automatisch zu quittieren

fbErrorSetCtrl\*.M\_ErrorSet(**xError**)hier wird die Variable, der zuvor im Studio definierte CMZ, zugewiesen "[NOT GVL\_<Sequence1>\_CMZ.xCMZ\_1](#GVL_Sequence1_CMZ)"

fbErrorSetCtrl\*.M\_ErrorSet(**xReset**)Globale oder Sequence bezogener Reset des CMZ "[GVL\_Global.stGlobalIf.xFaultReset](#StGlobalInterface2) ; [GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbFaultReset](#StHmiSequence2)"

**fbCmzFault** Der Baustein überprüft die entsprechenden Arrays ob ein Fehler aufgetreten ist.

fbCmzFault(**aFaultMatrix**)[Array der gesamten CMZ](#Sequence1_2_VariablenDeklaration8B3473F7) zusammengefasst

fbCmzFault(**xFaultActive**)Variable für Sammelfehler des CMZ "[GVL\_<Sequence1>.stSeqIf.xSeqCmzFault](#StSequenceInterface)"

// ########################### SEQUENCE CMZ ###########################

{region "Description Sequence CMZ"}

(\*

The Sequence Constantly Monitored Zone (or ‘CMZ’) is similar to the TCMZ but

will contain only machine unsafe /fatal faults related to the individual step sequence.

\*)

{endregion}

aCmz[0] := fbErrorSetCtrl0.M\_ErrorSet(xError := GVL\_<HwZone1>.stHwzIf.xFatalFault, xReset:= GVL\_Global.stGlobalIf.xFaultReset OR GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbFaultReset);

fbErrorSetCtrl1.P\_xAutoReset := TRUE;

aCmz[1] := fbErrorSetCtrl1.M\_ErrorSet(xError := GVL\_<HwZone1>.stHwzIf.xGateFortressFaultTcmz, xReset:= GVL\_Global.stGlobalIf.xFaultReset OR GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbFaultReset);

fbErrorSetCtrl2.P\_xAutoReset := TRUE;

fbErrorSetCtrl2.P\_timErrorDelay := T#1000MS;

aCmz[2] := fbErrorSetCtrl2.M\_ErrorSet(xError := NOT GVL\_<Sequence1>\_CMZ.xCMZ\_1, xReset:= GVL\_Global.stGlobalIf.xFaultReset OR GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbFaultReset);

aCmz[3] := fbErrorSetCtrl3.M\_ErrorSet(xError := GVL\_<Sequence1>\_CMZ.xCMZ\_2, xReset:= GVL\_Global.stGlobalIf.xFaultReset OR GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbFaultReset);

fbErrorSetCtrl4.P\_xAutoReset := TRUE;

fbErrorSetCtrl4.P\_timErrorDelay := T#500MS;

aCmz[4] := fbErrorSetCtrl4.M\_ErrorSet(xError := NOT GVL\_<Sequence1>\_CMZ.xCMZ\_3, xReset:= GVL\_Global.stGlobalIf.xFaultReset OR GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbFaultReset);

aCmz[5] := fbErrorSetCtrl5.M\_ErrorSet(xError := GVL\_<Sequence1>\_CMZ.xCMZ\_4, xReset:= GVL\_Global.stGlobalIf.xFaultReset OR GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbFaultReset);

fbCmzFault(aFaultMatrix:= aCmz, xFaultActive => GVL\_<Sequence1>.stSeqIf.xSeqCmzFault);

GVL\_<Sequence1>.stSeqIf.xNoCMZFault := NOT GVL\_<Sequence1>.stSeqIf.xSeqCmzFault;

Standard End

Der Funktionsblock **fbStandardEnd** enthält die Logik, der die Information aus dem Maschinenzustand(Schritt) zu Aktorik und Sensorik(Zonen) extrahieren und die Art der nachfolgenden Maßnahmen bestimmt. Dieser überwacht auch die Steuerung des Schrittzählers und liefert die Diagnose an die HMI. Dieser FB ist Hauptbestandteil jeder Sequence und ist in der Selmo Basic Bibliothek zu finden.

**refSeqHmi** ist die Reference(Pointer)zu der Variablenliste "[GVL\_<Sequence1>\_HMI.stHmiIf](#StHmiSequence2)"

**refSeqIf** ist die Reference(Pointer)zu der Variablenliste "[GVL\_<Sequence1>.stSeqIf](#StSequenceInterface)"

**iSeqEndStep** Abfrage des letzten modellierten Schritt "[GVL\_<HwZone1>\_IOs.xSafetyFunctionKeySwitch](#GVL_HwZone1_IOs)"

**iSeqStepcounter** aktueller Schrittzähler der Sequence "[GVL\_<Sequence1>.iStepcounter](#GVL_Sequence1)"

**aLampMatrix** [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

**aStepMatrix** [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

**aTempMatrix** [Array der Bit Matrix](#Sequence1_2_VariablenDeklaration8B3473F7) wird um zwei Indizes je Zone erhöht

**aSeqHmiBits** Array der Bit Matrix wird um zwei Indizes je Zone erhöht und dient zur Information der HMI "[GVL\_<Sequence1>\_HMI.aHmi](#GVL_Sequence1_HMI)"

**aStepMonitoringMatrix** Array zum Speichern der Zeiten je Schritt "[GVL\_<Sequence1>.aStepMonitoringMatrix](#GVL_Sequence1)"

**aStepMonitoringFb** [Array der Speicherbausteine](#Sequence1_2_VariablenDeklaration8B3473F7) je Schritt

// ########################### STANDARD END ###########################

{region "Description Standard End"}

(\*

The Standard End section provides the logic which extracts fault information from

the step sequence fault matrix determining the nature of the fault and subsequent actions.

It also supervises control of the sequence step counter and provides the diagnostic ‘link’ to the ‘HMI’ software.

\*)

{endregion}

GVL\_<Sequence1>.iStepCounterLastCycle := GVL\_<Sequence1>.iStepCounter;

fbStandardEnd(

refSeqHmi:= GVL\_<Sequence1>\_HMI.stHmiIf,

refSeqIf :=GVL\_<Sequence1>.stSeqIf,

iSeqEndStep := GVL\_<Sequence1>.iEndStep,

iSeqStepCounter := GVL\_<Sequence1>.iStepCounter,

aLampMatrix := aMatrix,

aStepMatrix := aStep,

aTempMatrix := aTemp,

aSeqHmiBits := GVL\_<Sequence1>\_HMI.aHmi,

aStepMonitoringMatrix := GVL\_<Sequence1>.aStepMonitoringMatrix,

aStepMonitoringFb := aStepMonitoringFb);

Interlock Stored

// --------------------------- Interlock stored ---------------------------

// Copy HMI Bits to stored array

FOR nHmiStoredLoopCounter := 0 TO (UINT\_TO\_INT(SIZEOF(GVL\_<Sequence1>\_HMI.aHmi)) - 1) BY 1 DO

IF GVL\_<Sequence1>\_HMI.aHmi[nHmiStoredLoopCounter] THEN

GVL\_<Sequence1>\_HMI.aHmiStored[nHmiStoredLoopCounter] := TRUE;

END\_IF

END\_FOR;

// Delete stored array every step counter change

IF GVL\_<Sequence1>.iStepCounterLastCycle <> GVL\_<Sequence1>.iStepCounter OR GVL\_Global.stGlobalIf.xFaultReset OR GVL\_<Sequence1>\_HMI.stHmiIf.xHmiPbFaultReset THEN

FOR nHmiStoredLoopCounter := 0 TO (UINT\_TO\_INT(SIZEOF(GVL\_<Sequence1>\_HMI.aHmi)) - 1) BY 1 DO

GVL\_<Sequence1>\_HMI.aHmiStored[nHmiStoredLoopCounter] := FALSE;

END\_FOR;

END\_IF

###### <Sequence1>\_InputMapping

{region "Description Input Mapping"}

(\*

All Step Sequence Zone Inputs that cannot be directly linked are connected to the real world hardware in this section.

This must be done manually.

\*)

{endregion}

(\*

GVL\_<Sequence1>\_IOs.<i\_xDecision\_1Path1> := ; // Decision: Decision 1

GVL\_<Sequence1>\_IOs.<i\_xDecision\_1Path2> := ; // Decision: Decision 1

GVL\_<Sequence1>\_IOs.<i\_xZone7> := ;

GVL\_<Sequence1>\_IOs.<i\_xZone9> := ;

GVL\_<Sequence1>\_IOs.<i\_xZone10> := ;

\*)

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Im Input-Mapping werden die benötigten Hardware-Inputs auf die Input-Zonen gemappt.

Beispielhaft:

GVL\_Sequence1\_IOs.i\_xDecision\_1Path1 := GVL.I\_Y1; // Decision: Decision 1

GVL\_Sequence1\_IOs.i\_xDecision\_1Path2 := GVL.I\_Y2; // Decision: Decision 1

GVL\_Sequence1\_IOs.i\_xZone7 := GVL.I\_Y3;

GVL\_Sequence1\_IOs.i\_xZone9 := GVL.I\_Y4;

GVL\_Sequence1\_IOs.i\_xZone10 := GVL.I\_Y5;

###### <Sequence1>\_OutputMapping

{region "Description Output Mapping"}

(\*

All Step Sequence Zone Outputs that cannot be directly linked are connected to the real world hardware in this section.

This must be done manually.

\*)

{endregion}

(\*

:= GVL\_<Sequence1>\_IOs.<o\_xZone9>;

:= GVL\_<Sequence1>\_IOs.<o\_xZone10>;

:= GVL\_<Sequence1>\_IOs.<o\_xZone11>;

:= GVL\_<Sequence1>\_IOs.<o\_xZone12>;

\*)

<\*\*\*> Variablen Name, Kommentar und Datentyp werden aus dem Studio automatisch übernommen

Im Output-Mapping werden die Output-Zonen auf die Hardware-Ausgänge wie benötigt gemappt.

Beispielhaft:

GVL.O\_Y1 := GVL\_Sequence1\_IOs.o\_xZone9;

GVL.O\_Y2 := GVL\_Sequence1\_IOs.o\_xZone10;

GVL.O\_Y3 := GVL\_Sequence1\_IOs.o\_xZone11;

GVL.O\_Y4 := GVL\_Sequence1\_IOs.o\_xZone12;

Keyword index

A

Array Condition Zone · 85

Array Cross Zone · 85

Array Paircheck Zone · 85

C

Conditinal Jumps · 80

Cross Sequence Zone · 98

D

Decisions · 79

Desicion Zone · 88

E

End of Cycle Mode · 53

F

Fatal Fault · 57

H

HMI Buttons · 77

I

InOut Zone · 94

Input Zone · 92

Interlock Stored · 106

O

Operands · 82

Output Zone · 96

R

Repeater Zone · 100

S

Sequence CMZ · 103

Sequence Logic Control · 81

Standard Beginning · 78

Standard End · 105

T

Timer Zone · 87

V

Variablen Deklaration der Sequence · 73

Z

Zone Call · 85