MASTER'S THESIS

A case study in compositional model-checking of hierarchical PLC programs

Erps, R. (Roel)

Award date:
2021

A Case Study in Compositional Model-Checking of Hierarchical PLC Programs

Graduation Assignment

by

Roel Erps

Date:	13-10-2021	
Student number:		
Course code:	IM9906	
Degree programme:	Open University of the Netherlands, Faculty of Science, Master's Programme in Software Engineering	
Supervisor:	dr. F. Verbeek,	Open University
Thesis committee:	dr. F. Verbeek (chairman),	Open University
	dr. S. Schivo (supervisor),	Open University

Open Universiteit
de beste! www.ou.nl

Acknowledgements

I would like to start by thanking my supervisor Freek Verbeek. His powerful insights and profound knowledge of formal verification guided me throughout this graduation assignment. Without his help I would never have come this far. I would also like to thank Stefano Schivo for his feedback on this graduation assignment.

For the occasional scientific perspective I could always count on my fellow student Ton. Throughout the Master I have learned a lot from him, improving my writing skills is one of them. Thanks Ton!

Last but not least I would like to use this opportunity to thank my wonderful partner Laura. She made sure I had the time needed for this Master and supported me every step of the way. Thank you for giving me the possibility to successfully complete this Master. I would also like to thank my close relatives for the understanding they showed during the completion of this Master.

"People think of education as something they can finish."
– Isaac Asimov, *In an interview by Bill Moyers (1988)*

CONTENTS

List of Figures

List of Tables

ABSTRACT

In the world today, software is everywhere. Thus, maintaining the assurance that the software a person is using is safe and of high quality is imperative. Software for controlling a plant or production facility is often placed in a programmable logic controller (PLC), a robust and reliable embedded controller. Currently, manual testing, which often turns out to be sensitive to errors, is generally used to ensure the quality and safety of PLC programs. Use of model checking for PLC programs could alleviate this problem of error sensitivity by exploring all possible states of the software. An often-discussed challenge is that of the state space explosion when model checking PLC programs, for example. This research focuses on the idea that the hierarchical structure of Siemens PLC programs can be leveraged to apply compositional model checking. Current studies demonstrate the model checking of small lab test scale case studies, and only a handful of studies reveal the model checking of an industrial-sized PLC program in case studies. None of the studies apply compositional model checking to PLC programs. This research presents a novel approach to model checking PLC programs through the addition of compositional reasoning to the model checking of industrial-sized Siemens PLC programs. The presented approach demonstrates successful compositional verification of an industrial-sized PLC program.

1. Introduction

Currently, people's lives are frequently influenced by software in a large diversity of context. Software is used in smartwatches, cars, houses and many more items that individuals interact with daily. Industries also use software (e.g., to produce food, cars and other consumer items as well as for the supply of fresh drinking water and electricity). For a large part, these industries rely on programmable logic controllers (PLCs)[1] and their corresponding software programs to control the diverse equipment in the field or on production lines [3].

The assurance of quality and safety of industrial software is of utmost importance, and model checking software can contribute to this. Correct implementation of software functionality while ensuring the absence of bugs is important to maintain high quality and guarantee the safety of people and the environment. Model checking methods for computer applications are more common than the formal verification of PLC programs. Currently, no commercial tools are available for model checking of PLC programs. However, many studies have been conducted concerning model checking of PLC programs [18].

1.1. Motivation

Currently, assurance for quality and safety of PLC programs is generally achieved by manual testing, where unlike model checking, certain fault states can remain undiscovered. The manual testing of PLC programs requires ample time and effort to eventually obtain only a partial assurance of quality and safety. Such PLC systems may have hundreds of inputs and outputs. A great advantage to model checking is the formal basis with which all states of the model can be explored. Thus, model checking can provide a complete view regarding the quality and safety of PLC programs.

A frequently recurring challenge with model checking is the state space explosion; this also applies to the model checking of PLC programs. A rapidly expanding state space because of the addition of a small number of extra inputs or outputs is a clear example of the state space explosion. This is also reflected in related work, where case studies are often small lab tests [16, 17, 25, 34] conducted only once with a large industrial real-life PLC program [7, 8].

1.2. Contribution

Clear and structured PLC programs often utilize *hierarchical* and modular programming methods. An example of a hierarchical method is presented in the ISA88 standard, which is also commonly used in Siemens PLC programs. The use of these hierarchical programming methods can accomplish modularity of PLC programs. A hierarchical structure helps in dividing the PLC program into smaller pieces, each with its own responsibility. This in turn enables a PLC program with low coupling and thus high modularity.

This research focuses on the idea that this hierarchical and modular structure of Siemens PLC programs can be leveraged to apply compositional reasoning when model checking PLC programs. By using compositional model checking, one large model is divided into multiple sub-models for verification. This divide-and-conquer principle is used for coun-

[1] According to a Siemens salesman, their PLCs are used in at least the following markets (although this is not an exhaustive list): mechanical engineering industry, chemistry, petro-chemistry, offshore markets, food and beverage, automotive, textile processing, plastic processing industry, logistics centers, agriculture, water and wastewater treatment facilities, feed, photo-voltaic manufacturing, pulp and paper industry, concrete industry and mining.

tering the state space explosion. Together with a hierarchical and highly modular PLC program with low coupling, this was demonstrated to be an effective approach.

For this approach to be precise and valid according to the semantics of a PLC, further research into the formal definition of the semantics of Siemens PLCs and their programming languages is required. This research therefore also attempts to formally define the semantics of the Siemens PLC program due to a lack of concise definitions from the manufacturer.

This research proposes a novel approach to compositional model checking of a hierarchical PLC program. The discussed case study further reveals the effectiveness of this approach. All the code, models and materials necessary for the reproduction of the results discussed in this thesis (e.g., the PLC code and export, generation of the logic model and the `nuXmv` models and properties) are located online at the following GitHub repository: https://github.com/roelerps/TIA-XML-modcheck.

1.3. State of the art

Current related work typically presents small-scale lab tests for the model checking of PLC programs [16, 17, 25, 34]. Soliman et al. present rules for the translation of PLC programs to UPPAAL timed automata models [25]. In their paper, Soliman et al. describe a case study of a laboratory system that consists of 12 inputs and 3 outputs. This can be qualified as a small PLC program. A comparable approach has been presented by Li et al [16]. In their paper, they present an automatic translation of PLC programs in the function block diagram (FBD) programming language to NuSMV models. As NuSMV is a symbolic model checker, some performance gain is to be expected using binary decision diagrams. A case study is described, which consists of 20 inputs and 10 outputs. Although the PLC program here is larger than that from the case study in the paper by Soliman et al., it is still considered a small PLC program.

In contrast to the above, research at the European Council for Neclear Research (CERN) demonstrates that model checking PLC programs can also be done at an industrial-sized scale [1, 7, 8]. The most recent paper by Darvas et al. presents a case study that consists of 120 inputs and 80 outputs [8]. In multiple papers by Darvas et al., abstractions, reductions and symbolic model checking have been described to enable model checking of large industrial-sized PLC programs [7, 8]. Additionally, other approaches for the translation of PLC programs into formal models exist. The tools Arcade.PLC [4] and PLC-NuSMV [22], for example, create the opportunity to automatically translate PLC programs into formal models.

Although compositional model checking is not a new technique [6] and has been studied for many years, it has not yet been applied to PLC programs. The compositional model checking approach has already been applied to, for example, the verification of protocols running on on-chip networks [28] and the verification of train control systems [32]. In both approaches, the compositional approach enabled verification of these systems. No studies currently illustrate the compositional approach being applied to PLC programs.

1.4. Outline

The remainder of this document is structured as follows. In Chapter 2, related work concerning the model checking of PLC programs is discussed. In Chapter 3, the running example of this research is introduced and discussed. This running example is also used as

the case study in this research. Chapter 4 discusses the approach of information extraction and the compositional model checking of PLC programs. In Chapter 5, the case study is discussed in further detail together with the discussed approach from Chapter 4. In Chapter 6, the characteristics of this tool and method are discussed. Chapter 7 ends this paper with a conclusion on the discussed case study and the proposed approach for compositional model checking of hierarchical PLC programs.

2. Related Work

Model checking in industrial automation or industrial control systems is not a new research area; a significant amount of research has already been conducted in this area. Nevertheless, there are still many open challenges for verification of PLC programs, which is why this research area remains active and important. A number of these challenges are indicated by Darvas and Blanco in their study [9] and in the survey by Ovatman et al. [18]. These papers are discussed more extensively in the next sections of this chapter.

The remainder of this chapter is structured as follows: first, three surveys are discussed that observed and summarized many studies in the research area of model checking PLC programs. Subsequently, various model checkers are discussed in separate paragraphs with their associated studies. The discussed model checkers and approaches are UPPAAL, NuSMV, nuXmv, Cadence SMV, counterexample-guided abstraction refinement (CEGAR), PLCverif and some others that are, to a lesser degree, related to model checking PLC programs.

Most of the discussed studies are processed and listed in Tables 1 and 2 for comparison in the second to last paragraph of this chapter:

- **Table 1** presents an overview of the discussed model checking tools or methods and their global characteristics.
- **Table 2** presents an overview of the same model checking tools or methods and their detailed properties. It reveals the possibilities and impossibilities together with other details for each tool or method.

The last paragraph of this chapter describes the contribution and relevance of this study for the related research area.

2.1. Surveys

Many studies have already been conducted regarding model checking PLC programs. First, three surveys are discussed in the next three paragraphs to gain an adequate overview of the research field.

One of the early surveys conducted in this research area is done by Lampérière [15]. Here, the focus is on the verification methods for the sequential function chart (SFC) and ladder (LAD) IEC 61131-3 programming languages. Lampérière et al. note studies that present verification of SFC and LAD as being possible by transforming LAD to SFC or Petri nets. Another possibility is the formal expression of the PLC program and its requirements followed by use of a symbolic model verifier (SMV). Verification methods for Grafcet are noted as potentially easily adapted for verification of SFC. The Grafcet specification language (IEC 60848) is very similar to SFC, as Grafcet was the starting point for the development of SFC. Lampérière et al. conclude that at the time of their survey, no method was proposed that could handle more than one PLC programming language. There are, however, methods proposed for the verification of one specific programming language or for a specific purpose, like liveness or safety.

The survey by Ovatman et al. provides an extensive overview of the research field [18]. Besides providing ample background information about PLC's and PLC programming, Ovatman et al. compare and discuss the studies based on the programming language used in each study. Section 5 of the survey discusses the model checking of textual PLC programs. The PLC programming languages that are discussed here are instruction list (IL) and structured text (ST). Ovatman et al. note that the study by Willems is one of the few that uses

timed models and can only handle a few complex programs [31]. The study by Pavlovic et al. demonstrates that their approach can handle more complex programs with a greater state space [21]. Section 6 discusses model checking graphical PLC programs, including the FBD and LAD languages. With FBD, PLC programs are automatically translated in most studies. Ovatman et al. analyze and conclude that model checking of LAD programs is conducted on systems with a maximum of around 100 variables. Another interesting note by Ovatman et al. is the fact that only half of the studies perform automatic transformation of LAD code into formal models. The most used specification language is computation tree logic (CTL), and only a few studies use timed CTL. For the studies based on the SFC programming language, most are semi-automatic at minimum. Abstractions are completed manually after automatic translation. None of the studies concerning SFC have used real-time properties. Together with SFC, studies that are Petri net based also lack explicit use of real-time properties. The study by Ovatman et al. discusses that Petri net studies also lack evaluation of the performance of the model checking based on Petri nets. Furthermore, Ovatman et al. discuss multiple open problems and research challenges. The common challenges are summarized in the list below:

- **State space explosion:** Because PLC programming languages can be quite detailed and granular, the translation of a PLC program into models for model checking can cause the state space to grow exponentially. Different solutions, include applying abstractions to the generated model, compacting recurring places in a Petri net or using an intermediate model generated from the source code.
- **Consistency of the model:** An often-occurring problem is whether the built model is consistent with the system it represents. For the real-time characteristics of PLC programs, it is challenging to ensure consistency. One solution can be the reuse of manually translated and extensively tested templates for timed aspects of the PLC program.
- **Properties to be checked:** Another difficult but important step in model checking is the specification of the properties that are to be checked by the model checker. Model checkers often expect linear-time temporal logic (LTL) or CTL specifications, which is not common practice for people working with PLCs. Automatic translation from natural language or tabular format information to LTL or CTL specifications can become a solution for this challenge.
- **PLC-cycle execution:** The several components of a PLC cycle also create a challenge for model checking. In addition to the PLC program itself, the execution characteristics of a PLC also need to be modeled. To access a consistent model, the timed aspects of the PLC cycle must be modeled as well.
- **Modeling timers:** A commonly used element in PLC programming is the timer. Because of the timed behavior of timers, they are often modeled as a timed automata and used with the real-time model checker UPPAAL.

A list of open research challenges that Ovatman et al. found unresolved at the time of their study is provided below:

- **Conformance generated model with original program:** When generating a model from PLC source code, it is important that the model is still in conformance with the actual system. Often conformance testing is used, but another possible solution could be the generation of PLC source code and a model, which are both generated from a parent model.

- **Performance of the presented solutions:** In the discussed studies, performance has often been measured in errors found by the proposed solution, but information about the size of the program is missing or vague. More detailed information about the study is necessary to compare the performance of the proposed solutions.
- **Networked or multitasking PLCs:** Research into networked or multitasking PLCs still has challenges, mainly the state space explosion that occurs when model checking these types of PLC programs.

Although the survey by Rösch et al. has a slightly different scope, namely, a review of model-based testing approaches [23], it also notes an interesting subject that can be beneficial for model checking. Failures and faults caused by the hardware can have devastating consequences. Rösch et al. determined that for model-based testing, the system's reaction to faults requires more research. Failures and faults in the PLC system itself trigger an organization block (OB) associated with that type of failure or fault. These fault OBs interrupt the normal PLC cycle. Because of the importance and impact of these interrupts, Table 2 has a column that indicates whether interrupts have been considered in each study.

2.2. UPPAAL

One of the two most used model checkers is UPPAAL, which is also reflected in Tables 1 and 2 through the number of studies discussing UPPAAL. One of the early studies was completed by Willems [31]. Willems presents a set of tools that makes it possible to convert a PLC program in the IL programming language to timed automata for verification. A peculiarity to his research is the method to split the PLC program in a timed and untimed part. By splitting the PLC program, most of the program is located in the untimed part. In this part, time does not influence the execution of the program. Through considering time, the state space grows quickly. Thus, by splitting the PLC program in the manner as proposed by Willems, the state space explosion can be countered.

In their study, Zoubek et al. introduce a framework for automatic verification of LAD PLC programs [34]. They present a case study in which an Allen-Bradley PLC is used. According to Zoubek et al., an automatically converted PLC program into a timed automata cannot be verifiable without automatic or manual abstraction(s). Wardana et al. also present an automatic verification approach but for continuous function chart (CFC) programs [30]. In their research, Wardana et al. propose a method for the transformation of CFC programs to timed automata for verification.

Soliman and Frey propose an approach for the verification of safety applications that are built from PLCopen safety function blocks (SFBs) [24]. These SFBs are safety blocks in the FBD programming language. Soliman et al. continue their previous research [24] by transforming the extensible markup language (XML) exported SFBs to timed automata automatically [25]. Further details of this study and its comparison with other studies are located in Tables 1 and 2. In their paper, Soliman and Frey also describe the formal definition and transformation rules they used for their proposed approach.

In their paper, Enoiu et al. present a method of transforming FBD into timed automata and test generation [11]. Former related work has focused on transforming PLC programs into formal models and/or timed automata [24, 25, 30, 31, 34]. However, for verification, test properties are also needed, which differentiates this research from the former. In contrast to the other UPPAAL papers mentioned here, Enoiu et al. indicate future work in the direction of finding more data about usage in realistic industrial-sized PLC systems.

2.3. NuSMV and nuXmv

The other often-used model checker is NuSMV and (to a lesser degree) its successor nuXmv. Pavlovic and Ehrich present a method for model checking PLC programs written in FBD with NuSMV [20]. The method described in their paper demonstrates a translation via a text-based FBD version (textFBD) to a proposed tFBD language. The paper indicates that textFBD is the IL representation of the PLC program. The translation to tFBD is mainly achieved to reduce the state space of the model.

Pakonen et al. use a different approach in FBD PLC program verification. Pakonen et al. propose a toolset for model checking FBD PLC programs [19]. In their paper, they describe a manual translation of elementary PLC functions to NuSMV models. Subsequently, they use the open-source modelling tool `Simantics` for building models of the FBD PLC program. Additionally, this tool allows for visualization of counter examples when properties fail in a test. Possibilities and impossibilities of this approach are compared with other studies and presented in Table 2.

An example of an automated tool for the translation of FBD PLC programs to NuSMV models is presented in a study by Li et al. [16]. Here, the authors present their tool called PLC-NuSMV compiler, which accomplishes exactly that. A PLCopen XML export from the engineering software together with a truth table in .CSV format can be transformed into a NuSMV model for verification of the PLC program. The used case study, the `Falcon Controller`, consists of a safety program, and it includes 18 inputs and outputs. In Table 1, this case study is compared with others. In his master thesis, Qeriqi, a co-author with Li et al. [16], further describes the details of the PLC-NuSMV tool [22]. Here, Qeriqi offers more information about the PLC-NuSMV compiler, for example, the structure of the compiler. Qeriqi also notes the modular architecture of the PLC-NuSMV compiler, which enables easy extension for the support of more PLC languages. Qeriqi uses the same case study in his master thesis as is used by the Li et al. [16] study.

Another paper using the NuSMV model checker for the verification of PLC programs is by Kottler et al. [14]. Kottler et al. propose a method to show that PLC programs in LAD can be modeled and verified in NuSMV with use of CTL specifications. No automation is proposed in the paper, but with a small case study and manual fault injection, Kottler et al. indicate the possible positive effect of the verification of LAD PLC programs with NuSMV.

2.4. Cadence SMV

Although Cadence SMV seems to have ceased, there are still some informative studies using Cadence SMV. In their paper, De Smet and Rossi present a case study of some laboratory tests with an Allen-Bradley SLC-500 PLC [10]. They use the LAD programming language, and the PLC consists of 15 inputs and 15 outputs, which is a small case study compared to some of the other studies listed in Table 1. De Smet and Rossi conclude their paper through noting that even symbolic model checking is not capable of verifying very large or unbounded state space.

Another study using Cadence SMV is by Jee et al. They propose a tool called the `FBDVerifier` [12]. This tool verifies FBD PLC programs using Verilog models. Then, LDA file exports are generated from the FBD code using the engineering environment. These LDA files are then converted into Verilog models automatically. The user adds properties to be verified, and Cadence SMV verifies these properties on the Verilog models. In case a property fails, the tool visualizes the counter-example with a large matrix filled with values

of variables at a specific time. Jee et al. also describe a large case study of 20,000 function blocks and 9,000 variables, which can be compared with other studies in Table 1.

2.5. Counterexample-Guided Abstraction Refinement (CEGAR)

Counterexample-guided abstraction refinement [5] is a method that is frequently used with model checking. The CEGAR method creates an abstract model that most likely contains counterexamples that are not actual counterexamples of the model. These are called erroneous or spurious counterexamples by Clark et al. [5]. An erroneous counterexample is then used to guide the abstraction refinement. In general, this means that the abstraction level will be lowered, and at that point of the model, more detail is introduced. This process can be iterated to remedy all erroneous counterexamples.

Biallas et al. present the `Arcade.PLC` tool in their paper [4]. This tool is described as a verification platform for PLCs. It can handle the following PLC programming languages: IL, ST and the Siemens dialect of IL, statement list (STL). The exports need to be in EXP, POU, ST or AWL file formats, as presented in Table 2. The tool is built in Java and is also one of few publicly available tools. Biallas et al. describe three case studies, all of which are being lab tests. The case studies consist of the PLCopen safety FBDs, a conveyor system and a robot control system.

The second CEGAR-based study is by Nellen et al. [17]. Nellen et al. present two different approaches for verification of safety features of PLC-controlled plants concerning PLC programs developed in the SFC programming language. The first approach consists of the, broadly seen reach-ability analysis. The second approach uses a combination of bounded model checking for the PLC behavior and reach-ability analysis on the environment model. Nellen et al. implemented the methods and approaches in `SpaceEx with CEGAR`, which is publicly available.

2.6. PLCverif

Over the past years, extensive research for the verification of PLC programs has been conducted at CERN. Their studies mainly contain work towards their PLC verification tool called PLCverif. A small number of studies concerning PLCverif are used in the papers written by Darvas et al. and Adiego et al. [1, 7, 8], which are presented in Tables 1 and 2.

Adiego et al. propose a methodology for the translation of a PLC program into an intermediate model for verification purposes [1]. By using an intermediate model, they apply the separation of concerns, make it easy to add new/other model checkers and allow for abstraction techniques on the intermediate model for more advantageous performance of the verification. In their study Adiego et al. describe and use only the translation from the ST programming language. Adiego et al. also demonstrate the applicability of the transformation methodology by describing a CERN lab test. The `OnOff` object from CERN's UNICOS library is used for the case study. It is one of the simpler objects in the UNICOS library and can represent different types of Boolean process equipment.

Darvas et al. go into greater detail regarding the intermediate model and specifically the language used. They investigate the possibility of using the Siemens ST language, that is the structured control language (SCL), modified by emulating the option of register access (STr) [7]. They presented the possibility to transform LAD, FBD, IL and ST into STr, which is used for creation of the intermediate model. Their case study of a cryogenic safety-critical machine consists of 9,500 lines of IL code and can be seen as a large case study when com-

pared with all case studies in Table 1.

More recently, Darvas et al. released a paper about PLCverif, which states that PLCverif will soon be publicly available [8]. This would make PLCverif one of a sparse number of publicly available tools for PLC verification. PLCverif can be provided with IL or ST PLC code (AWL and SCL files) for verification of that PLC code. Additionally, two case studies are described by Darvas et al. [8], both situated at CERN. The first is a magnet test bench and the second is a so-called `Super Proton Synchrotron accelerator`. The size of the programs, which is presented in Table 1, is around 10,000 lines of LAD PLC code and around 200 IO points.

A significant comparative study is conducted by Helder [27], where Helder compares several model checkers' performance with performance of PLCverif. The model checkers for models used are as follows: Spin, NuSMV and nuXmv. The model checkers for the code used are as follows: CBMC, Kinductor, CBMC Incremental, 2LS, CPA-checker and SATABS. The case study used is the CPC program, which is comparable to the `OnOff` object from Adiego et al. [1]. Although most of tests reveal that PLCverif is slightly faster, there are a few tests that demonstrate a significant improvement in performance by using a model checker on code instead of models. As Helder indicates in the thesis, the tests were performed on different machines.

Another interesting study is conducted by Darvas et al [9], who indicate some of the advantages, difficulties and challenges of model checking PLC programs. The most relevant points are listed below:

- **PLC programming languages:** The variety in PLC programming languages themselves and in the implementation of the same language by different vendors create complication. Furthermore, proprietary languages by vendors often have partial or unprecise syntax definitions, which further complicate the formalization of the PLC program.
- **Semantics PLC languages:** Behavior of PLC languages is not regularly obvious or precisely described. Additionally, the real-time behavior of a PLC creates another challenge in how precise this real-time behavior needs to be. By increasing the precision of the real-time behavior, the needed resources for verification also increase. Semantics also seem to depend on the programming environment and PLC hardware that is used.
- **Environment and faults:** The environment (process) is controlled by the PLC, and the environment's behavior, context and dependencies can also be modeled. This generates a more precise model and thus the possibility of less false negative results of the verification. The same applies for assumptions made about faults that can occur. Some faults can physically never occur, but others will most certainly happen in a possible chain of events. Proper assumptions are therefore important in model checking PLC programs.

2.7. Other Tools

Besides the above-mentioned model checkers, more model checkers exist; a small number of them are also mentioned in research targeting PLC programs. Because most studies seem to focus on UPPAAL and SMVs (e.g., NuSMV, nuXmv and Cadence SMV) and for the sake of readability of Tables 1 and 2, the model checkers mentioned in this section are not appended to these tables.

The first example is the study by Zheng et al., where an efficient method for modeling and the verification of PLC systems is proposed [33]. This study uses the model checker Spin for the formal verification of PLC programs by the proposed PLC Checker tool. Although the exact supported PLC programming languages are not specified in the study, the noted PLC programming languages are LAD, IL and ST.

In the study by Voros et al., Petri nets are elaborately described together with the tool PetriDotNet [29]. This tool can be used for modeling and analysis of industrial applications as noted by the case studies in Section 4 of their paper. Voros et al. conclude with the statement that with the described analysis algorithms, they were able to analyze systems that could not previously be analyzed. Further case studies and papers are located in the survey of Ovatman et al. [18] in Table 1 of that survey.

Another example is the study by Song et al., where the Yices SMT Solver is used [26]. Song et al. used the Yices SMT Solver to extend existing research from Jee et al. [13]. The tool FBDTester 2.0 automates test generation for FBD programs and considers the internal memory states of the block. Although FBDTest 2.0 utilizes Yices SMT Solver, the study by Song et al. focuses on the generation of tests for verification.

2.8. Overview of State of the Art

All discussed related work and a selection of additional studies are summarized in Tables 1 and 2 to provide a clear overview and comparison of the related studies.

Table 1 lists a great part of the relevant research in this area. The discussed or supported model checkers are listed for each study, together with the discussed and supported PLC progamming languages. The Automated column presents which part of the process of model checking is automated. Most studies automate the PLC code to formal model translation together with possible abstractions or simplifications of the formal model. In most of the studies, one or more case studies are mentioned. The context of this case study and the size of the case study are also presented in Table 1. Unfortunately, most of the studies discuss or propose a tool that is not publicly available, which is listed in the last column of Table 1. The studies are presented in two separate tables, Tables 1 and 2, to fit the information on the pages. Table 1 presents a rough overview of the studies, and Table 2 presents all specific details mentioned in each study or case study. The Model Checker and Lang. columns are used in both tables for potential to compare study details based on model checker or PLC programming languages. The program size of the possible case studies is often mentioned in a unique unit. For example, one research study measures the program size by counting the lines of IL, another by counting the FBD networks and another through describing the environment that is controlled and monitored by the PLC program. This makes it harder to compare the actual program size of the case studies.

Table 2 presents a more detailed comparison between the proposed solutions in the discussed related studies. The discussed or supported model checkers are listed for each study, together with the discussed and supported PLC progamming languages. Most of the studies present or propose a tool or method for translating the PLC code into formal models. The supported file formats are listed in the Export format column. In the four columns on the right more information is listed about the presented or proposed tool or method:

- **IO-scan cycle:** This column indicates whether the study considers the IO-scan cycle. This is the cycle of a PLC in which the operating system sequentially, reads the inputs,

	Model Checker	Lang.	Automated	Case Study	Study Size	Public
[11]	UPPAAL	FBD	Model to test properties	Train control subsystem: battery management system	30 FBD's	No
[25]	UPPAAL	FBD	PLC code to model	Safety, precision movement table lab-test	3 Networks, 15 IO	No
[24]	UPPAAL	FBD	No automation described	Safety, em. stop with safe stop and equiv. monit.	4 FBD's	No
[30]	UPPAAL	CFC	PLC code to model	Lab-test, process industry	5 Tanks, 3 control loops	No
[31]	UPPAAL	IL	PLC code to model	Unknown	N/a	No [3]
[34]	UPPAAL	LAD	PLC code to model	Pumping line	10 IO	No
[14]	NuSMV	LAD	No automation described	None	N/a	No
[16]	NuSMV	FBD	PLC code to model	"Falcon" safety controller	30 IO	No
[19]	NuSMV	FBD	No automation described	Unclear	Unknown	No [2]
[20]	NuSMV	FBD	No automation but a method described	Rail automation	165 lines IL, 100 vars	No
[10]	Cadence SMV	LAD	PLC code to model	Lab-tests	Unknown	No
[12]	Cadence SMV	FBD	PLC code to model	Nuclear reactor protection system	20.000 blocks and 9.000 vars	No
[4]	Arcade.PLC (CEGAR)	ST, IL and STL	PLC code to model	Lab-tests: PLCopen safety FBD's, Conveyor, Robot	Unknown	Yes
[17]	CEGAR	SFC	Implementation in SpaceEx	Process industry lab-tests	2 tank approx. 20 IO	Yes
[1]	NuSMV, UPPAAL	ST	PLC code to model	CERN	Lab-test "OnOff" object	No
[7]	NuSMV, nuXmv, UPPAAL, THETA (PLCverif)	IL(STL), ST(SCL)	PLC code to model	CERN, cryogenic safety-critical installation	9.500 lines IL	No
[8]	NuSMV, nuXmv, UPPAAL, THETA (PLCverif)	IL(STL), ST(SCL)	PLC code to model	CERN, Safety: magnet test bench, proton accelerator	10.000 lines LAD, 200 IO	No [1]

1) To be released in the upcoming months
2) Partly available through `Simantics`
3) Provided URL is not reachable (anymore)

Table 1: Comparison related studies

executes the PLC program and writes the outputs.

- **Interrupts:** In addition to the IO-scan cycle, the normal cycle can be interrupted. This column indicates whether the study also considers such possible interrupts.
- **Block-based model checking:** The PLC programs in and of themselves are structured by program blocks in a hierarchy. This hierarchy and call structure can be used for block-based model checking. This can be done, for example, by applying compositional reasoning and thus reducing state space by dividing the big program into smaller blocks for model checking.
- **Timers:** Timers are frequently-used timed components in a PLC program. Depending on how these timers are modeled, a real-time model checker is necessary. This column presents whether the study mentions a method about how timers are handled.
- **Variables:** In PLC programs, global variables can be used. These variables can be used more than once for reading and writing. This column reveals whether the study considers variables.

	Model Checker	**Lang.**	**Export format**	**IO-Scan cycle**	**Interrupts**	**Block-based**	**Timers**	**Variables**
[11]	UPPAAL	FBD	N/a	Yes	No	No	Yes	Unknown
[25]	UPPAAL	FBD	.XML (PLCopen format)	Unknown	No	No	Yes	Yes
[24]	UPPAAL	FBD	N/a	No	No	No	No	No
[30]	UPPAAL	CFC	N/a	Yes	No	No	Yes	Unknown
[31]	UPPAAL	IL	IL text-based	Yes	No	No	Yes	Unknown
[34]	UPPAAL	LAD	N/a	Yes	No	No	Yes	Unknown
[14]	NuSMV	LAD	N/a	Unknown	No	No	Yes	Unknown
[16]	NuSMV	FBD	.XML (PLCopen format)	Yes	No	No	No	Unknown
[19]	NuSMV	FBD	N/a	No	No	No	No	Unknown
[20]	NuSMV	FBD	IL text-based	Yes	No	No	No	Yes
[10]	Cadence SMV	LAD	LAD text-based	Yes	No	No	Unknown	Yes
[12]	Cadence SMV	FBD	LDA format to Verilog model	Yes	No	No	Yes	Unknown
[4]	Arcade.PLC (CEGAR)	ST, IL and STL	.EXP, .POU, .ST, .AWL	Yes	No	No	Unknown	Unknown
[17]	CEGAR	SFC	N/a	Yes	No	No	No	No
[1]	NuSMV, UPPAAL	ST	.ST	Yes	No	No	Yes in [2]	Yes
[7]	NuSMV, nuXmv, UPPAAL, THETA (PLCverif)	IL(STL), ST(SCL)	.AWL	Yes	No	No	No	Yes
[8]	NuSMV, nuXmv, UPPAAL, THETA (PLCverif)	IL(STL), ST(SCL)	.SCL, .AWL	Yes in [7]	No	No	No	Unknown

Table 2: Details related studies

2.9. COMPOSITIONAL MODEL CHECKING

Compositional model checking and compositional reasoning are not a new subject. In 1989, Clarke et al. proposed compositional reasoning for model checking [6]. They describe it as a method to reduce complexity of systems with many parallel processes. However, they also note that a similar method can be used when the program contains a modular or hierarchical structure. This is accomplished by hiding details of lower-level components that are not relevant for the higher-level components. Different states can be merged into one which can subsequently reduce the state space.

Verbeek et al. present a compositional approach for model checking protocols running on-chip networks [28]. In this research, discussed results are positive regarding the compositional approach of model checking, whereas the monolithic approach is considered infeasible.

Xie uses a compositional approach for the formal verification of train control systems. In his research [32], Xie converts train control systems into Petri net models. These models are then verified using a compositional approach.

3. RUNNING EXAMPLE

In this chapter, a running example is discussed, which is used throughout the research as an example for explanation and as the case study for this research. This chapter provides a brief overview of the specifics of this running example in terms of the process as well as more details about the PLC and the PLC program of this running example. The discussed running example also poses as a motive for the proposed research.

3.1. PROCESS

Before presenting more details about the running example, the actual process used for the running example is briefly introduced.

For a large high-tech manufacturer, a chemical cleaning installation is supplied. Stainless steel tools that are used in production are cleaned in this machine. For this, two chemical fluids are used. The tool is first placed in a small container, after which it is soaked with the first fluid. After draining this fluid, the tool is flushed with demineralised water before being soaked in the second fluid. This is done separately because the two fluids react uncontrollably with each other, which can result in an explosion.

The actuators and sensors of the machine are as follows (simplified). See Figure 1 for a schematic overview of the machine. (The two black triangles represent a valve.)

- **Supply valve liquid (x2):** These two valves fill the container with the specified liquid when opened.
- **Drain valve:** This valve drains the container when it needs to be emptied.
- **Container full sensor:** This sensor indicates that the container is entirely filled with a liquid.
- **Container emtpy sensor:** This sensor indicates that the container is completely empty and that no liquid resides in the container.

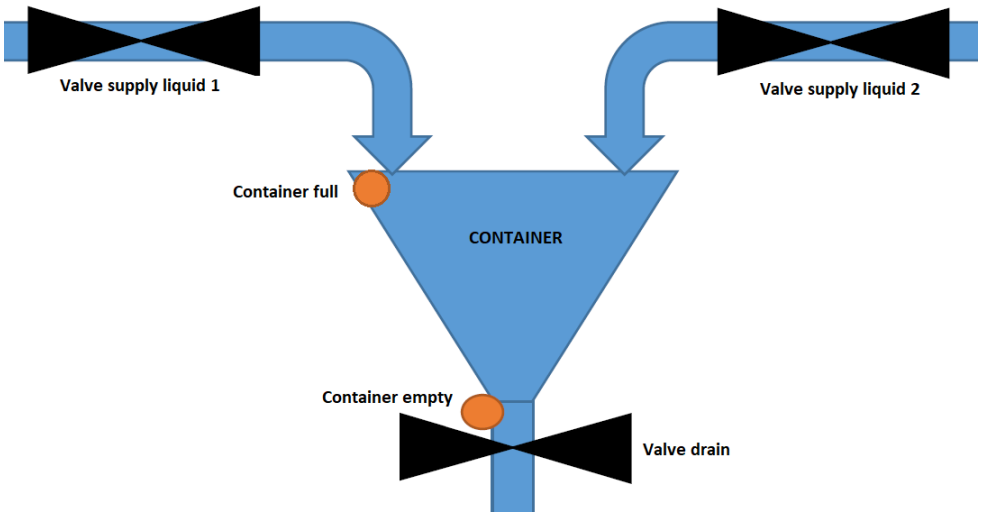


Figure 1: Process (simplified) of the running example

The risk for machine and humans is that the two fluids react dangerously with each other. Thus, it must not be possible to have the two fluids together in the container in any possible event. The goal is to verify this safety property by means of model checking the PLC program.

3.2. Programmable Logic Controller (PLC)

The process as described in the previous section is controlled by a *Siemens S7-1500 PLC*. This PLC is programmed with the Siemens TIA-Portal V16 engineering environment. The PLC blocks are programmed using the following IEC 61131 standard programming languages: FBD, LAD and SCL. For the execution of the PLC blocks, OB1 is used; other OBs are not used in the example.

The *PLC program* for this case study largely revolves around four blocks. This is displayed in Figure 2. Three of the four blocks, called `control modules`, are instances of a block to control a solenoid valve (FB110). An instance of a block means that the block is provided with its own data block for the storage of instance-specific variables. This way, a function block can be reused and used multiple times simultaneously. The fourth block is the program logic (FB5), which controls the valves in a sequential way depending on the sensors and push buttons on the control cabinet. This block is called a `sequencer`.

In short, the hierarchy of the PLC program is as follows: The `sequencer` controls the valves. This is done by global variables that are written by the `sequencer` and read by the `control modules` of the valves. This logical relation between the blocks is illustrated in Figure 2. This reveals the division of responsibilities in the PLC program. The `sequencer` defines when something needs to happen, and the `control modules` define how this will be achieved. For example, the `sequencer` defines when the valves need to open and close, and the `control module` defines how the valve is controlled and monitored. The high level of modularity and re-use of blocks causes this PLC program to have a notably hierarchical structure. This, in turn, is leveraged for compositional model checking as is further discussed in Section 4.2.

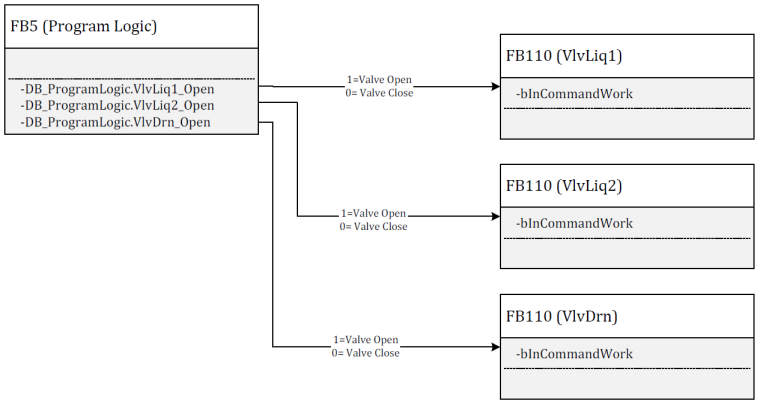
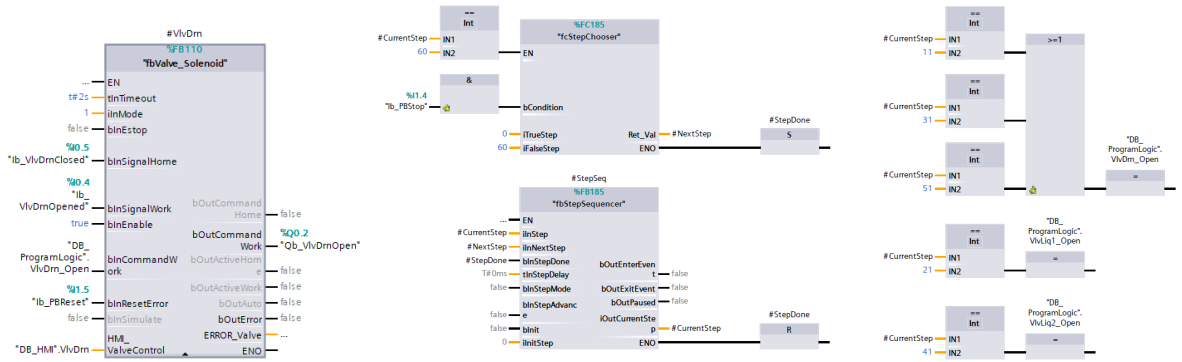


Figure 2: Logic relations PLC program

Even though it is a simple example, the PLC program contains a significant amount of code. This is mainly due to the standardized way of programming, where `control modules` are used to control the valves. These `control modules` consist of broadly applicable functions. An example instance of the `control module` valve solenoid is presented in Figure 3a, representing the instance of the drain valve. In the header of the grey block, it is displayed that this is block FB110 called fbValve_Solenoid. Above the grey block, its instance and thus its data block are displayed. The revealed instance is VlvDrn, which is short for valve drain. On the left-hand side of the block, the inputs of the block are presented, and on the right-hand side, the outputs of the block. This demonstrates that to safely control

a valve, many signals are used. Besides controlling the valve, this block also monitors and alarms when the valve does not respond as expected.

The program logic block (FB5) is a `sequencer` that can move from step to step, depending on various variables. This sequencer in the end controls the valves depending on the step the sequencer is in. A code snippet of the program logic block is presented in Figure 3b. The sequencer consists of numbered steps, which are traversed depending on the conditions of transitions. Two other blocks are used in the program logic block. They act as helpers for the execution of the sequence. These two blocks are now discussed. The upper block, FC185, which is called fcStepChooser, assigns the value of the next step depending on the connected condition and the current step number. This block can be seen as a small helper used to simplify the step choice. Below FC185, the block FB185, which is called fbStepSequencer, is revealed. This can also be seen as a helper. It orchestrates the function that when a transition is made, the current step number is updated with the next step number. On the right-hand side, the generation of three command signals are displayed. These are the signals that are connected to the control modules of the three valves. These signals are the commands for the valve to open. These commands are generated depending on the active step number of the sequence. The valve automatically closes when the open command is not active.



(a) Valve solenoid instance (drain-valve) (b) Program logic block

Figure 3: PLC code snippets

Blocks of the PLC program are called as functions for other blocks in a hierarchical manner. The execution order of the PLC program is as follows. The PLC's OS calls OB1. OB1 then calls FB2, FB3, FB4 and FB5. The blocks FB2, FB3 and FB4 then each call their own instance of FB110. Block FB5 calls FC185 multiple times and FB185 once. This is illustrated in Figure 4 in the call structure figure. A clear list of these blocks and their defined names are in Appendix C. The complete call structure is generated by the developed TIA-XML-modcheck tool and is in Appendix D.

3.3. Comparison

In this section, a comparison of characteristics and features is provided between this running example project and some related studies from Chapter 2 as displayed in Tables 1 and 2. This running example consists of 17 PLC blocks in the programming languages LAD, FBD and SCL. This PLC program uses 17 inputs and outputs to control the equipment of the machine. The machine as subject of this running example serves to chemically clean tools used in the production of high-tech chips.

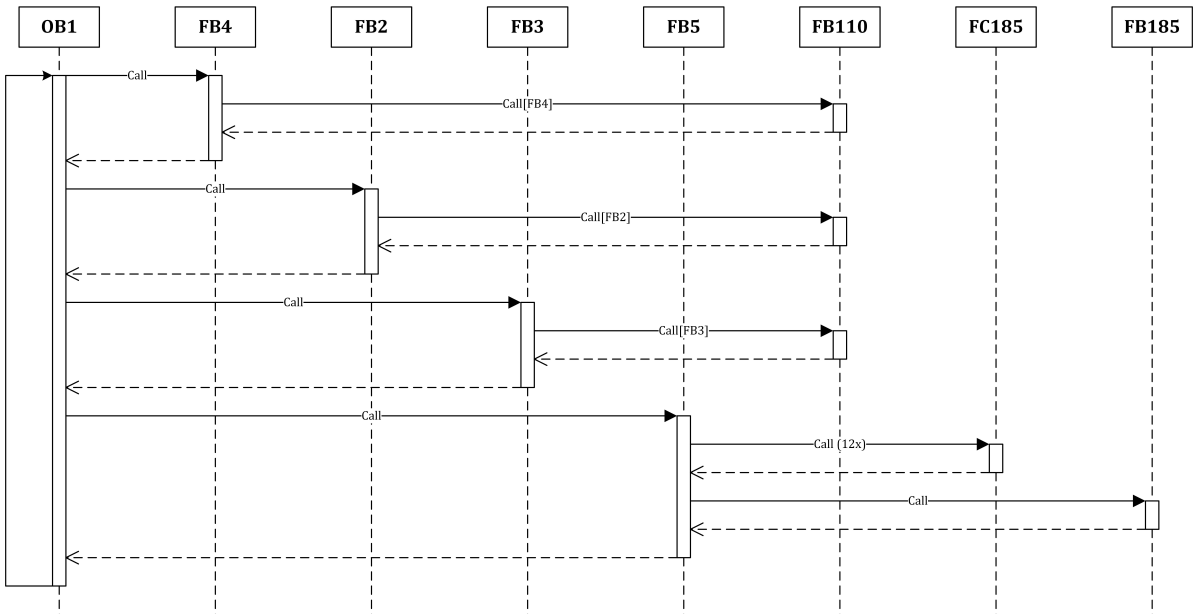


Figure 4: PLC call structure

Observing the details of this running example, more characteristics and features can be found. It can be stated that interrupts are not used, only OB1 —the main cycle— is used. Timers and global variables are also used in this running example. The export format is XML but not the PLCopen format. Siemens TIA Portal uses a proprietary format for their XML export.

4. Overview of Approach

This chapter consists of two sections. The first section describes the process of *information extraction*. Information from the PLC program is exported and processed in several steps into a `nuXmv` model. This `nuXmv` model is then used for formal verification. The second section describes the verification process. *Compositional reasoning* is used for model checking of the `nuXmv` models. Together with *CEGAR*, this enables compositional model checking of PLC programs.

4.1. Information Extraction

This section describes the method of information extraction of the PLC program. The first step of information extraction is the export of the PLC program compile units, which is followed by the graph extraction and then conversion into a logic model of the PLC program. This logic model is then converted into a nuXmv model to enable model checking. These steps are illustrated in Figure 5 accompanied with the section numbers where each step is discussed.

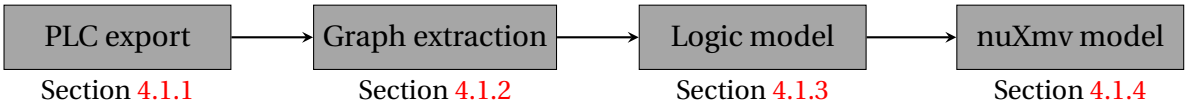


Figure 5: Overview information extraction section

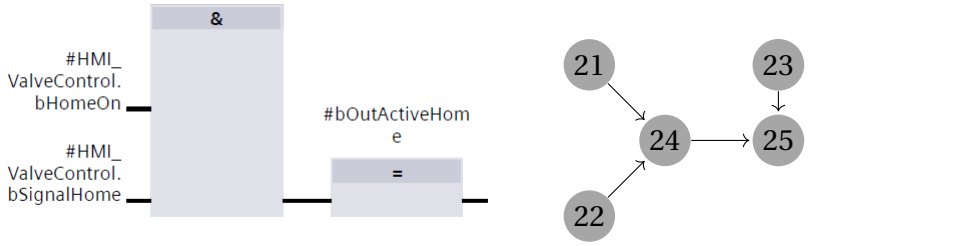
4.1.1 PLC program export

Siemens offers a TIA Portal Openness tool that can export the PLC program independently from which programming language is used. The export functionality uses the XML file format. For each PLC block, an XML file, containing all relevant information about that PLC block and the PLC code of that block, is created. The PLC code is defined for each compile unit. All instructions, variables and wires are provided a unique identifier for each compile unit. The definitions of compile units, instructions, variables and wires are discussed in the next section.

4.1.2 Compile units

The program of a PLC is built with PLC blocks. These PLC blocks divide the PLC program into logic parts and provide the possibility to reuse functionality. A PLC block consists of one or more compile units. Compile units provide a greater overview of the code in a PLC block because the functionality of a PLC block is divided into smaller pieces of PLC code. Each compile unit can consist of instructions, variables and wires.

Example 4.1. *Figure 6a presents a small example of PLC code. It reveals three variables, `#HMI_ValveControl.bHomeOn` and `#HMI_ValveControl.bSignalHome` at the input of the `AND` instruction and `#bOutActiveHome` at the coil assignment. In the same figure, two instructions are found: the `AND` instruction and the coil assignment. The `AND` instruction executes a logical and-function on the connected inputs on the left side of the instruction. The right side outputs the result. The coil assignment assigns the input value on the left side of the instruction to the connected variable. The wires are the lines that connect variables with instructions and instructions with instructions. This is also called the Boolean data flow.*



(a) AND-instruction with coil assignment

(b) Graph representation of example network

Figure 6: Small example PLC code information extraction

Definition 4.1. Let I be a sorted list of instructions, V be a set of variables and W be a set of wires. A *PLC compile unit* is a directed acyclic graph $G = \langle I \times V, W \rangle$ where the set of vertices consists of variables and instructions, and vertices are connected by wires as the edges. It is not possible for a wire to connect two variables. Instructions are sorted because of the sequential execution of instructions as further discussed in Section 4.1.2 in the **Execution order** paragraph. □

Example 4.2. *Through Definition 4.1, the example from Figure 6 can be defined as follows:* $I = \{ \&, = \}$, $V = \{$ *#HMI_ValveControl.bHomeOn, #HMI_ValveControl.bSignalHome, #bOutActiveHome* $\}$ *and* $W = \{$ *<#HMI_ValveControl.bHomeOn, &>, <#HMI_ValveControl.bSignalHome, &>, <&, =>, <=, #bOutActiveHome>* $\}$. *The following execution order of instructions holds here:* $\& \sqsubseteq =$.

Semantics of compile units: To explain the semantics of compile units, Example 4.1 and Figure 6a are used. For each compile unit, unique Ids are assigned to all instructions, variables and wires. Together with Definition 4.1, the graph in Figure 6b is created. The semantics are as follows:

Variables `#HMI_ValveControl.bHomeOn` with Id=21 and `#HMI_ValveControl.bSignalHome` with Id=22 are connected with wires to an `AND` instruction with Id=24. The result of the `AND` instruction is connected with a wire to the input of the coil assignment instruction with Id=25. Variable `#bOutActiveHome` with Id=23 is connected with a wire to the coil assignment instruction with Id=25. The wires in this example are not explicitly numbered but can be found in Figure 6b as the edges between the vertices.

The result is that when the variables with Id=21 and Id=22 are true, the variable with Id=23 is also true. In all other cases, the variable with Id=23 is false.

In general, *instructions* can be simple Boolean instructions (e.g., an `AND` instruction), a complex mathematical calculation or even a call to another PLC block. The common aspect is that an instruction executes a function and uses optional inputs and outputs for its execution. Instructions used in Figure 6b are the `AND` instruction and the coil assignment.

Variables can consist of local or global variables and are used for storing and reading values. Global variables used in Figure 6b are `#HMI_ValveControl.bHomeOn` and `#HMI_ValveControl.bSignalHome`. An example of a local variable is `#bOutActiveHome`. The definition of local or global depends on the place of the declaration of the variable.

Wires connect the two types of building blocks of a compile unit: the instructions and variables. Wires let Boolean data flow between the instructions and variables. To assign variables to the input of an `AND` instruction, wires are used between the variables and the

instruction. To pass the value of the AND instruction to the next instruction, the coil assignment, a wire is used. To assign the value to a variable, a coil assignment is used. A wire between the variable and the coil assignment assures the value is written to the variable. Wires can connect two instructions but not two variables.

Execution order: A very important part of Definition 4.1 is the partial order of the instructions. Instructions of compile units are not all evaluated at the same clock-tick but sequentially so the partial ordering is of importance here. This partial ordering generates an execution order, and this defines the order in which the instructions are executed in the PLC.

In an industry effort to standardize the programming languages used in PLCs the IEC-1131 was created and later renamed the IEC-61131. Unfortunately, there is still room for interpretation for the automation system manufacturers. This means that the Siemens programming language does not fully comply with the IEC-61131; Siemens released several documents over the years stating which parts of the IEC-61131 they do comply with. To analyze the execution order of Siemens PLC programs, the following example is used:

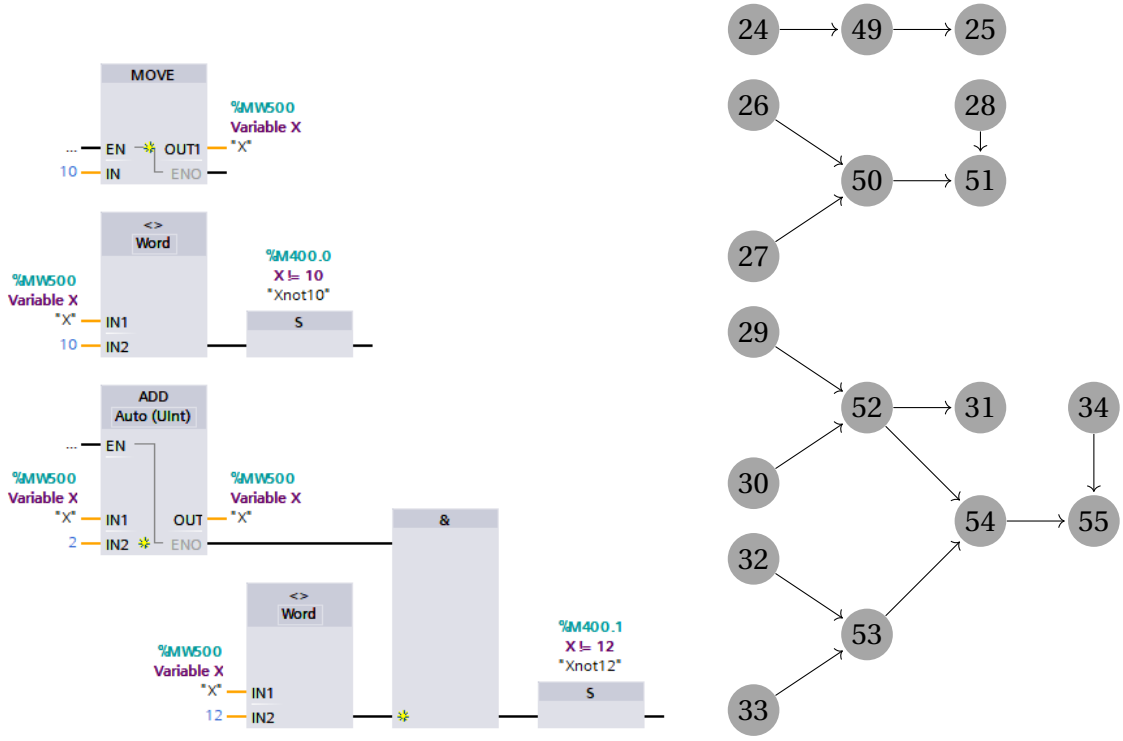
Example 4.3. *Figure 7a presents a small example of PLC code of a compile unit. The wire between the instructions* `add` *and* `&` *visualizes the data flow and order. As revealed on the instruction itself, the* `EN` *input is the trigger to start the instruction, which is always true if it is not connected like in this example. The* `ENO` *of instructions is a signal to indicate that the instruction has been successfully completed. This way, instructions for integers and floating points can be placed in a Boolean data flow.*

Figure 7b illustrates three directed graphs created from the compile unit presented in Figure 7a. The numbers in the vertices are again the Ids used in the export files. An important detail of these graphs is not only the execution order of the graphs but also the instructions inside the graph. Due to the lack of definition in the IEC-61131, the automation system manufacturer is partly free to determine the execution order. Because Siemens documentation about execution ordering was not located, the execution order is defined here in Definition 4.2.

Definition 4.2. A *block* is an ordered list of compile units together with a defined interface of inputs, outputs and local variables. □

The execution order of a block is provided by executing its compile units in the given order. The execution order of a compile units is as follows: If a compile unit consists of more than one graph, the graphs are executed from top to bottom. The execution order of each graph of compile unit C must conform with the following definitions: Let I be a sorted list of the instructions used in C_n. Let i_i be all inputs and i_o all outputs of an instruction I_n in I. Before executing instruction I_n all i_i need to be evaluated first. The execution of instruction I_n is only finished when all i_o are evaluated. The execution of a graph is only completed when all i_o are evaluated of all I_n in the graph.

Example 4.4. *Through Definition 4.1, the example from Figure 7 can be defined for C_n as follows: I = { move, <>, ...}, V = { 10, X, ...} and W = { <10, move>, <move, X>, ...}. The following order holds here: move $\sqsubseteq$ <> $\sqsubseteq$ S $\sqsubseteq$ add $\sqsubseteq$ ....*



(a) Example of execution order

(b) Graph representation of example

Figure 7: Small example PLC code execution order

Semantics of execution order: The main semantics of Figure 7b and Example 4.3 are as follows: The first instruction revealed is the `move` block. This writes the value of 10 to variable `X`. Next, variable `X` is checked if it indeed has a value of 10. If this is not the case, variable `Xnot10` is set to `true`. The following instruction, `add`, adds the value of 2 to the variable `X` and writes the result to variable `X`. This means that variable `X` has the value of 12 at this point. Lastly, whether variable `X` indeed has a value of 12 is verified. If not, the variable `Xnot12` is set to `true`.

To further explain the execution order of a compile unit, the example from Figure 7b and Example 4.3 is used. The graph of Example 4.3 consists of three smaller graphs.

The *first graph* to be executed is the graph with vertices $< 24, 49, 25 >$. This graph consists of one instruction: `move`. First, all its inputs are evaluated, and constant value 10 is read. Then, the outputs are evaluated, and constant value 10 is written to variable `X`.

The *second graph* to be executed is the graph with vertices $< 26, 27, 50, 51, 28 >$. This graph consists of two instructions: <> with id 50 and `S` with id 51. The instruction with which the process is started does not affect the result because all inputs are always first evaluated. Effectively, this means that instruction <> and its inputs are always evaluated first. This means that variable `X` is checked for a value other than the constant value 10. If the value of variable `X` is not 10, then the variable `Xnot10` is set. An evaluation from top to bottom and left to right can be observed here.

The *third graph* to be executed is the graph with vertices $< 29, 30, 51, 31, 32, 33, 53, 54, 55, 34 >$. This graph consists of four instructions: `add` with id 52, <> with id 53, `&` with id 54 and `S` with id 55. To determine the execution order of this graph, it can be started at instruction `S`. Then, by evaluating its inputs, instruction `&` is encountered. This, in turn, leads to

instruction `add` and <>. Conforming with the top-to-bottom approach, `add` is the first instruction executed. This execution is only finished if the output of `add` is evaluated and thus a value of 12 is written to variable `X`. Then instruction <> is evaluated, and together with the completed instruction `add`, the variable `Xnot12` is set to true if the output of instruction `&` is also true.

Generally and informally, the order from top to bottom and left to right provides a solid idea of the execution order. The compile units are executed instruction by instruction, where the next instruction only starts when the previous instruction is finished completely and all outputs of that instruction have been evaluated.

Algorithm graph extraction: This paragraph describes the algorithm that is responsible for the extraction of the graph. The algorithm returns a reversed graph as the result.

Algorithm 1 Graph extraction algorithm per code unit

```
1:  procedure CreateGraph(plcCode)
2:      v ←variables extracted from plcCode
3:      i ←instructions extracted from plcCode
4:      w ← wires extracted from plcCode
5:
6:      g ← create new Graph
7:      for all v ∪ i do
8:          x ← create new vertex
9:          add x to g
10:     end for
11:     for all w do
12:         if w_n has branches then          ▷ Some wires connect one output to multiple inputs
13:             separate w_n into multiple w_2's
14:             replace w_n with set of w_2's
15:         end if
16:     end for
17:     for all w do
18:         e ← create new edge
19:         add e to g
20:     end for
21:
22:     g2 ← create new ReversedGraph of g          ▷ All edges of the graph are reversed
23:     return g2
24: end procedure
```

Definition and semantics algorithm: For each code unit, the algorithm is executed to extract the graph from the PLC code of that code unit. The pseudo-code algorithm is presented in Algorithm 1; this represents the functionality of a part of the developed Java tool.

This algorithm consists of five steps. All steps of the algorithm are described below with the line numbers of the pseudo-code in Algorithm 1:

1. The *first step*, within lines 2 to 4, extracts the variables, instructions and wires from the PLC code export.

2. The *second step* is the creation of a new graph and addition of vertices representing the extracted instructions and variables. Lines 6 to 10 represent this step.
3. In the *third step*, the wires are split when a wire is branched; this is represented in lines 11 to 16. Branching can be used when an instruction is connected with two other instructions. When step three is finished, all wires consist of one source and one destination.
4. The *fourth step* is adding the wires as edges to the graph; this is represented in lines 17 to 20.
5. The *fifth step* creates a reversed version of the graph. This means that the direction of all edges is reversed; this is represented in line 22.

4.1.3 LOGIC MODEL

To universally define and describe the code of a PLC compile unit independently from the model checker used, a logic model is defined. A logic model is an abstraction of PLC code and is a text-based model of the graphical FBD programming language. Other PLC programming languages can also be translated into a logic model. Figure 7 presents the conversion from PLC code to a graph. In the next example, the conversion from graph to logic model is explained, which is based on Example 4.3.

Example 4.5. *Listing 1 presents the logic model for the PLC code discussed in Example 4.3. The graph in Figure 7b is used as input for the creation of the logic model. Line 1 starts with the assignment of a constant value of 10 to variable* `X`*. Line 2 states that variable* `Xnot10` *is set true when variable* `X` *has a value anything but 10. Line 3 reveals the* `add` *instruction, which adds a value of 2 to the variable* `X` *and assigns the variable* `X` *a value of 12. Line 4 states that variable* `Xnot12` *is set true when variable* `X` *has a value anything but 12.*

```
1  X = 10
2  Xnot10 = (if (X <> 10) then TRUE)
3  X = 2 + X
4  Xnot12 = (if (X <> 12) then TRUE)
```

Listing 1: Example logic model

Definition 4.3. A logic model is an ordered list $[(v_0, f_0), (v_1, f_1), \ldots]$ of tuples of type $V \times F$, where each tuple denotes an assignment of formula f_i to variable v_i. □

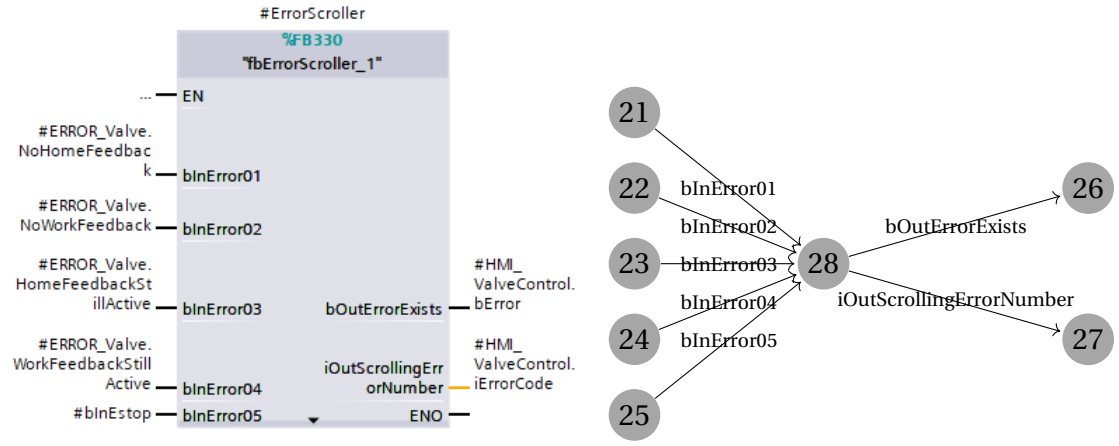
Semantics of logic model: As presented in Listing 1, one compile unit can consist of multiple lines of formulas in the logic model. Furthermore, the logic model must adhere to the execution order as described in Section 4.1.2.

For example, Listing 1 is executed as follows: First, line 1 is executed. This means that first the right-hand side of the assignment is evaluated. After this evaluation is complete for the entire right-hand side, the assignment is executed, and the result is assigned to the variable on the left-hand side. For line 2, the same procedure occurs. The right-hand side here is different; `( if (X <> 10) then TRUE )` means that the left-hand side variable is set to true when `X!=10`. If, for example, `X==10`, the variable on the left-hand side is not

changed. Line 3 is executed by first evaluating the right-hand side 2+X. At this point, before the execution of this assignment, the value of X is 10. The result of the evaluation after execution of the right-hand side is 12. This value is than assigned to variable X. Line 4 evaluates analogous to line 2.

A list with PLC instructions and the corresponding defined symbols and syntax in the logic model is presented in Appendix B.

Example with a block call: Figure 8a presents an example of a function block call, FB330, in this case. In Figure 8b, the graph of the call in Figure 8a is presented.



(a) Example function block call

(b) Graph representation of example call

Figure 8: Small example PLC code information extraction

Figure 8b resembles Figure 6b in that the variables are represented as numbers 21, 22, 23, 24, 25, 26 and 27. The call of FB330 is displayed as an instruction with number 28. The edges here also represent the wires between elements. The difference is that the wires on the side of the call are identified with the corresponding interface variable (e.g., interface variable `bInError01`). This is displayed in Figure 8b with the text near each edge. The graph from Figure 8b is then converted into the logic model presented in Listing 2.

```
1  Call: FB330 (fbErrorScroller), Instance: ErrorScroller (LocalVariable)
2  Inputs:
3    ERROR_Valve.NoHomeFeedback ==> bInError01
4    ERROR_Valve.NoWorkFeedback ==> bInError02
5    ERROR_Valve.HomeFeedbackStillActive ==> bInError03
6    ERROR_Valve.WorkFeedbackStillActive ==> bInError04
7    bInEstop ==> bInError05
8  Outputs:
9    bOutErrorExists ==> HMI_ValveControl.bError
10   iOutScrollingErrorNumber ==> HMI_ValveControl.iErrorCode
```

Listing 2: Example logic model call

In general, *instructions* are represented as conforming to the conversion list in Appendix B. They connect with variables and other instructions. *Variables* are represented by their name and are connected to instructions. *Wires* are not represented in the logic model, but they do determine the placement of variables and instructions in the logic model.

Algorithm logic model: This section describes the algorithm for the creation of the logic model to understand how the logic model of a PLC program is created. First, the algorithm is demonstrated with use of the running example as illustrated in Figure 6 and Listing 1.

Example 4.6. *Figure 9 presents the reversed graph from Figure 6b. The direction of all edges is reversed. In Listing 3, the sequence of the results of the logarithm is displayed. Each line represents the output of a subsequent step of the algorithm. Line 1 reveals the vertex with zero incoming edges:* `cu-25`*. Line 2 reveals the conversion from* `cu-25` *to* `=`*, the vertex with zero outgoing edges* `cu-23` *and the remaining connected vertex* `cu-24`*. Line 3 reveals the conversion from* `cu-24` *to* `&&` *and the connected vertices of* `cu-24`*,* `cu-21` *and* `cu-22`*. Line 4 presents the last step in the algorithm: the conversion of* `cu-xx` *to the corresponding variable names.*

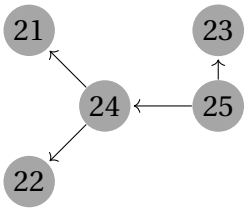


Figure 9: Reversed graph

```
1  cu-25
2  cu-23 = cu-24
3  cu-23 = cu-21 && cu-22
4  bOutActiveHome = (HMI_ValveControl.bHomeOn && HMI_ValveControl.bSignalHome)
```

Listing 3: Algorithm iterations

Definition and semantics algorithm: For each compile unit, the algorithm is executed to extract the logic model from the reversed graph of that compile unit. The pseudo-code algorithm is presented in Algorithm 2; this represents the functionality of a part of the developed Java tool.

The algorithm consists of four steps. All steps of the algorithm are described below with the line numbers of the pseudo-code in Algorithm 2:

1. The *first step* prepares the reversed graph for traversing. Some edges are replaced by new edges to create a continuous Boolean data flow. For example, `move` instructions can interrupt the Boolean data flow. This step ensures that the conditions on the input of the `move` instruction are also used on the input of the instruction connected to the output of the `move` instruction. This step can be found in lines 2 to 7.
2. The *second step* has two possible options: The first option is when the compile unit contains a call to another block. Then, the called block is displayed together with its interface. For both the inputs and outputs, the connected variables and/or instructions are displayed. The second option is when the compile unit does not contain a call to another block. Then, this step traverses the reversed graph to construct the logic model, which can be found in lines 9 to 16. The algorithm starts at the vertex

Algorithm 2 Logic model algorithm for each code unit

```
1:  procedure CreateLogicModel(reversedGraph, plcCode)
2:      if x ∈ reversedGraph contains < Coil, Move, Add > then
3:          x_t ← get vertex connected to ENO output of x
4:          x_s ← get vertex connected to EN or IN input of x
5:          w_o ← get wire < x_t, x >
6:          replace w_o with wire < x_t, x_s >
7:      end if
8:      i ←instructions extracted from plcCode
9:      if i contains call then
10:         output ← call with connected variables/instructions
11:     else
12:         li ← get x where incoming e == 0          ▷ Last instruction of the code unit
13:         lv ← get x where outgoing e == 0          ▷ Last variable of the code unit
14:         output ← lv + li
15:         x_n ← connected vertices of li
16:         add x_n to output          ▷ If x_n ∈ i display instruction, if x_n ∈ v display cu − nn
17:         loop          ▷ Loop recursively until no more connected vertices
18:             x_{n+1} ← connected vertices of x_n
19:             add x_{n+1} to output
20:         end loop
21:     end if
22:
23:     for all cu − nn in output do
24:         replace cu − nn for variable name          ▷ e.g. from cu-23 to bOutActiveHome
25:     end for
26:
27:     return output
28: end procedure
```

with zero incoming edges. The connected output variable is the connected vertex with zero outgoing edges. Instructions are displayed by their logic model representation (see Appendix B). Variables are represented by `cu-nn`, where `nn` represents the id of the variable.

3. The *third step* recursively traverses the reversed graph to find connected vertices until all vertices are traversed and the logic model is completed. This is represented in lines 17 to 20.
4. The *fourth step*, in lines 23 to 25, replaces the `cu-nns` in the logic model for the actual variable names.

4.1.4 NuXmv model

The last step in the information extraction process is the conversion from a logic model to a model suitable for the model checker `nuXmv`. This conversion step has been achieved manually due to time restrictions, but the procedure of the conversion is presented in total detail in this section.

An example is presented here after a short introduction of the global structure of the nuXmv model. The example is followed by definitions for the `nuXmv` model. This section

ends with semantics of the nuXmv model to generate a more advantageous understanding of the model.

Introduction: A particular structure for the nuXmv model is used to represent the execution of PLC code in the PLC. To fully understand the structure of the nuXmv model, this is first briefly introduced here.

A PLC block has an interface: inputs and outputs that are read and written by the PLC block. The inputs of the PLC block are modeled as parameters of a module in the nuXmv model that represents the PLC block. Outputs and local variables are defined as variables in the nuXmv model.

As explained in Section 4.1.2, a PLC block contains multiple compile units. These compile units are sequentially executed in one PLC cycle from the first to the last compile unit. This principle is modeled in the nuXmv mode by using an enumeration type named `loc` which indicates the location of the execution in the block. Possible values of `loc` are `<start, nw1, nw2, ... end>`. The values `start` and `end` indicate the start and end of the PLC block; in these states, a change of value of the input variables of the PLC block is enabled. Due to this construct, the input variables of the PLC block cannot change during the execution of the block. This also conforms with the PLC's I/O cycle. The I/O cycle happens before and after the program execution. This means that physical inputs are read before the program is executed, and the physical outputs gain the updated values after execution of the program. The sequence for `loc` is presented in Listing 4.

In the following example, the conversion of a code unit is illustrated. This example extends to Example 4.5:

```
next(loc) :=
  case
    (loc = start) : nw1;
    (loc = nw1) : nw2;
    (loc = nw2) : nwXX;
    ...
    (loc = nwXX) : end;
    (loc = end) : start;
  esac;
```

Listing 4: Construct for `loc`

Example 4.7. *Listing 5 presents a small part of the nuXmv model converted from the logic model of Listing 1 and Example 4.5. It displays the code for the determination of the next state of the variable `o_bOutActiveHome` (named `bOutActiveHome` in Listing 1). When `loc = nw29`, the AND-evaluation of the two variables `io_HMI_ValceControl_bHomeOn` and `io_HMI_ValveControl_bSignalHome` (named `HMI_ValceControl_bHomeOn` and `HMI_ValveControl_bSignalHome` in Listing 1) occurs. The result of this evaluation becomes the next value for `o_bOutActiveHome`. When `loc` ≠ `nw29`, the value of `o_bOutActiveHome` remains unchanged.*

```
next(o_bOutActiveHome) :=
case
  (loc = nw29) : (io_HMI_ValveControl_bHomeOn & io_HMI_ValveControl_bSignalHome);
```

```
  TRUE: o_bOutActiveHome;
esac;
```

Listing 5: nuXmv model

Definition 4.4. Let B be a `module` in the *nuXmv model* representing a `function block`. The set of parameters P of B represent the inputs of the `function block`. The set of variables V of B represent the outputs and variables of the `function block`. V also consists of a variable for the execution position in B and an instance of a `module` for all used global variables. Let loc be the execution position in B. Start value of $loc = start$ and the `next` value of loc is the next compile unit until end is reached. This cycle is then repeated indefinitely. Let S be the set of `next` statements of the *nuXmv model* representing all variables that are modified by the `function block`. [2] □

Definition 4.5. Let M be a `module` in the *nuXmv model* representing a main block where B is called. The set of variables I of M represent the following: the inputs of B and one instance of `module` B. For each input in I of B, when B has $loc = end$, the input is provided a new non-deterministic value. □

```
MODULE blockname (input1, input2)
VAR
globalvars : globalvar;
loc : {start, nw1, nw2, end};
output1 : boolean;
ASSIGN
init(loc) := start;
init(output1) := FALSE;

next(loc) :=
  case
    (loc = start) : nw1;
    (loc = nw1) : nw2;
    (loc = nw2) : end;
    (loc = end) : start;
  esac;

next(output1) :=
  case
    (loc = nw1) : (input1 & input2);
    TRUE: output1;
  esac;


MODULE main
VAR
input1 : boolean;
input2 : boolean;
Valve : blockname(input1, input2);
ASSIGN
```

[2] Note that in *nuXmv*, a variable can only be used in one `next` statement. When a variable is given a new value more than once in one PLC cycle in the PLC code, these assignments need to be consolidated into one `next` statement.

```
31 init(input1) := FALSE;
32 init(input2) := FALSE;
33
34 next(input1) :=
35   case
36     (Valve.loc = end) : {FALSE, TRUE};
37     TRUE : input1;
38   esac;
39
40 next(input2) :=
41   case
42     (Valve.loc = end) : {FALSE, TRUE};
43     TRUE : input2;
44   esac;
```

Listing 6: Template nuXmv model

Semantics of nuXmv model: In Listing 6, an example template of a nuXmv model is displayed. It presents the overall structure of the nuXmv model of function block named blockname with inputs, input1 and input2, and one output, output1. The model of blockname consists of two *compile units*: nw1 and nw2.

In the next statement starting on line 18 of Listing 6, the next value of output1 is determined. If loc = nw1, then the statement input1 & input2 is evaluated, and the result is assigned to output1. If loc $\neq$ nw1, then the value of output1 remains unchanged.

From line 25 of Listing 6, the main module is presented. The main module instantiates module blockname once and determines a non-deterministic value for the inputs of module blockname for each PLC cycle.

One important note to consider is the granularity of a single state in the proposed nuXmv models. In Section 4.1.3, it is explained that each line with a formula of the logic model is executed sequentially. This means that after every instruction, the state of that logic model changes. The state of the nuXmv model, however, depends on whole compile units. Informally, this means that a whole compile unit is executed at once. This abstraction is done for the sake of state space reduction. If the nuXmv would treat every instruction separately, this would blow up the state space. In the case study, this abstraction causes no problems because in the case study, only single assignments for each variable for each compile unit are made. Informally, no variable is written a (different) value more than once for each compile unit. A possible solution to this limitation can be the modeling of an extra state for the extra assignments of the variable. When, for example, a compile unit with loc = nw2 has two assignments of the same variable, the loc = nw2 can be divided into loc = nw2a and loc = nw2b. This creates an extra state where the second assignment of the variable can be accomplished.

4.2. COMPOSITIONAL MODEL CHECKING

This section describes compositional model checking and how it is applied in this study. In the first part of this section, the applied abstraction of lower-level blocks is discussed. During the verification of the abstracted model a counterexample can be provided by `nuXmv`. This is the indication to apply the CEGAR-method on the model. In the second part of this section, a compositional approach for CEGAR is presented. These steps are visualized in the overview in Figure 10.

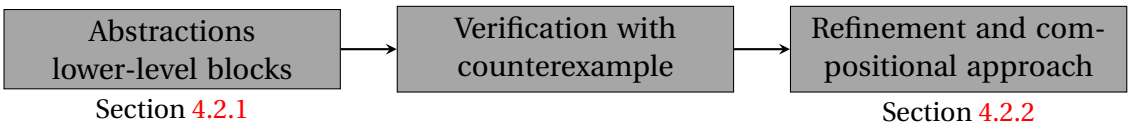


Figure 10: Overview compositional approach

4.2.1 ABSTRACTION OF LOWER-LEVEL BLOCKS

One of the challenges of model checking real systems is the state space explosion. Industrial PLC applications are no exception. For hierarchical programs, a modular technique, as proposed by Clarke et al. [6], can be used. This technique is called compositional reasoning. Combined with model checking, it becomes compositional model checking.

Compositional model checking can be applied to hierarchical PLC programs by hiding details of the lower-level blocks. Only details that are of interest and applicable to the higher-level block are considered, thus merging all states of the lower-level blocks that are not relevant for the model checking of the higher-level block. This reduces the number of states and thus mitigates the state space explosion to an extent.

There are also disadvantages of compositional model checking. It is challenging to determine if the simplified lower-level blocks are still functionally equal to the original lower level blocks. This makes it hard to automate the simplification of the lower-level blocks and still requires human intervention by applying, for example, CEGAR.

Next, an simplified example of compositional model checking is introduced and presented to clarify its applicability to hierarchical PLC programs.

Example 4.8. *For this example two new blocks are introduced and presented. The higher-level block, FB1, is called fbHigh, and the lower-level block, FB10, is called fbLow. These blocks are illustrated in Figure 11.*

To be able to focus on the used approach, blocks FB1 and FB10 consist of minimal functionality. The Block FB10 is called from the block FB1. Functionality of FB10 is as follows: FB10 consists of two inputs, input11 and input 12, and one output, output 11. Let the function of FB10 be an `AND`-function. Only when both input11 and input12 are true, output11 is also true. In all other cases output11 is false. Functionality of FB1 is as follows: FB1 consists of two inputs, input1 and input2, and one output, output 1. Both inputs, input1 and input2, are connected with input11 and input12, of FB10, respectively. The output of FB10, output11, is inverted before it connects with the output of FB1, output1. This results in an overall functionality of FB1 as follows: `output1 = !(input1 & input2)`. This is illustrated in Figure 12, the blue rectangle illustrates the scope of FB1 and the red rectangle illustrates the scope of FB10.

To model check FB1, FB10 must also be considered because `output1` *is controlled by the output* `output11` *of FB10. Here, compositional model checking is used as a solution, modeling FB10 as a black box instead of completely modeling FB10 in the model of FB1. See Definition 4.6 for the definition of modeling a block as a black box. An example of the* `nuXmv` *model containing a black box is in Listing 7.*

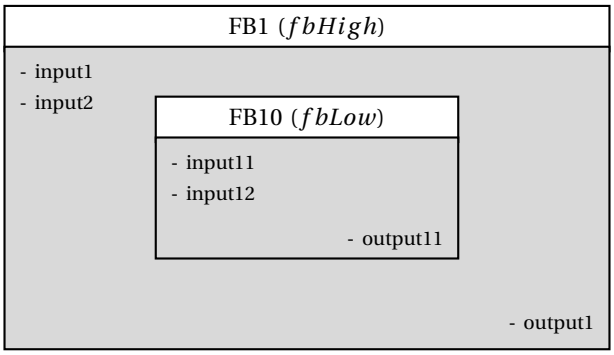


Figure 11: Example hierarchy of higher-level and lower-level blocks

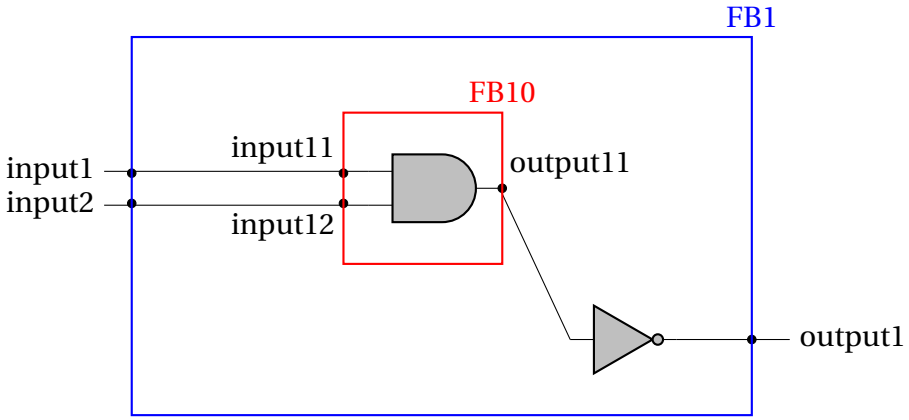


Figure 12: Example functionality of the higher-level and lower-level blocks, FB1 and FB10.

```
next(output11) :=
  case
    (loc = nw1) : {FALSE, TRUE};
    TRUE : output11;
  esac;
```

Listing 7: Example black box output

Definition 4.6. Let B be the block to be modeled as a black box, and let B_{bb} be the modeled black box. All inputs of the interface of B are not relevant for B_{bb}. Let O be a set of all outputs of the interface of B; these are relevant for B_{bb} and are modeled. All outputs in O are modeled so that these are given a new non-deterministic determined value in each PLC cycle. □

Semantics of abstraction of lower-level blocks: In Listing 7, an example of the output of FB10 in FB1 is revealed in `nuXmv` code. FB10 is named the lower-level block, and FB1 is named the higher-level block; this is also illustrated in Figure 11.

The only information needed from the lower-level block is the datatype of the output of the lower-level block. For `Boolean` outputs, a non-deterministic generated value is assigned to the variable connected to the output of the lower-level block. This value can change every PLC cycle. This makes the lower-level block a black box. It is not important what inputs it uses or what the functionality of the block is. Informally stated, it is not necessary to know what the block does, it is only important to assume that the outputs of the lower-level block can have any possible value.

In cases where the output has the datatype `integer`, this becomes more complex. To minimize the state space expansion, it is important to check which values of this variable are used in the higher-level block. If the higher-level block only uses the values of 1, 2 and 3, it suffices to give the output of the lower-level block at least these values and, for example, 0 and 4. The values 6 or 99 do not lead to a different state than 4 and, thus only cause the state space to explode. It is key to observe that values around the overflow value of the datatype are also worth considering.

4.2.2 Compositional approach using CEGAR

A methodology used for further abstraction of an abstract model is CEGAR [5]. This is proposed to keep the state space to a minimum.

In this study, we create the initial abstract model with the black box for lower-level blocks. This model is verified with an initial set of properties. If the model confirms to the properties, no further refinement is necessary. If the verification presents a counterexample, an additional manual check is needed. This check is to verify that the counterexample is not a counterexample in the abstracted model without the black box. This generates the conclusion that this counterexample is introduced by the black box. The abstractions enable the model to reach states that are not reachable in the real system.

The refinement step is used to integrate an assumption of the lower-level block in the model of the higher-level block. This is exemplified in Example 4.9.

Example 4.9. *This example presents a counterexample and an application of Definition 4.7. FB10 from Figure 11 is replaced with a black box. This means that the variables connected to the outputs of FB10 are provided a random value as revealed in the `nuXmv` example code in Listing 7. When model checking this abstracted model, a counterexample is given by `nuXmv` for the following property P_b:*

$$\boldsymbol{AG}\,(\,(\mathit{loc} = \mathit{end}\,\&\,\mathit{input1}\,\&\,\mathit{input2}\,)\,\rightarrow !\,\mathit{output1}\,) \tag{1}$$

This property checks, for example, whether the output (output1) is false when both input1 and input2 are true.

In Listing 7, the variable `output11` is given a random value. The value of this output is normally determined by FB10 as described in Example 4.8. This output is also used in FB1 to propagate the value to the output `output1` variable of FB1. Thus, `output1` is also assigned a random value.

In the counterexample, a state is revealed where `loc = end` and `input1` and `input2` are both `TRUE`, but `output1` is also `TRUE`. This is caused by the abstracted model of the output of FB10. Instead of modeling FB10 as a solution to the counterexample, the model of FB1

is refined. Here the proposed compositional approach is applied as a solution to the counterexample and to eliminate the need to completely model FB10 for the verification of FB1. The assumption is that `output11` *of FB10 is* `true` *when both inputs of FB10 are* `true`*. This property, P_{ll}, reveals this assumption:*

$$\mathbf{AG}\,(\,(\,loc = end\ \&\ input11\ \&\ input12\,)\ \rightarrow\ output11\,)\,) \tag{2}$$

Specification 2 is used for the verification of FB10 to ensure that FB10 actually satisfies the property. Specification 2 is also used for the verification of FB1 by applying it as an assumption. The invariant definition of `nuXmv` *is used to apply this assumption to FB1. Definition 4.7 is used for the compositional approach. This provides the following INVAR specification and verification property:*

$$\mathbf{INVAR}\,(\,(\,loc = end\ \&\ input11\ \&\ input12\,)\ \rightarrow\ output11\,)\,)$$

$$\mathbf{AG}\,(\,(\,loc = end\ \&\ input1\ \&\ input2\,)\ \rightarrow !output1\,) \tag{3}$$

Specification 3 combines the assumption/property for FB10, which is also verified by the verification of FB10, with the property for FB1. Informally, The following is checked with this property: FB1 propagates the signals `input1` *and* `input2` *to the inputs of the lower-level block FB10, and FB10 correctly combines the signals of* `input11` *and* `input12` *to its output* `output11`*, FB1, then, inverts the signal of* `output11` *and outputs this signal with* `output1`*. The assumption in Specification 3 is expressed with the* `INVAR` *statement. This instructs the higher-level model to only consider the states where this assumption/invariant holds. Verifying Specification 3 checks the functionality illustrated in Figure 12 by the blue rectangle but excluding the area of the red rectangle. The functionality in the red rectangle is verified with the verification of the lower-level block, FB10, by the property in Specification 2.*

Definition 4.7. Let B be a block, and let B_{ll} be the lower-level block that is called in B. When for a property of B, an output of B_{ll} is used, the property of that output from B_{ll} is added to the model of B in the following way: Let P_b be the property of B and P_{ll} the property of the output of B_{ll}. Let SD_{ll} be the state definition of P_{ll}. For example, $P_{ll} = AG(Q)$ then $SD_{ll} = Q$. The compositional approach for the `nuXmv` model of B is then to add SD_{ll} to the model of B as an invariant state. The model of B therefore contains INVAR (SD_{ll}) and P_b. □

Semantics of compositional approach using CEGAR: Example 4.9 illustrates the proposed approach of compositional model checking of a hierarchical PLC program. First, it is important to understand the larger picture of this approach. The approach uses the modularity of the PLC program by verifying only the block itself and not all lower-level blocks. Every lower-level block is verified separately. Let the higher-level block consist of one lower-level block. Then, first the lower-level block is verified. Next, the higher-level block is verified. In the model of the higher-level block, the lower-level block is modeled as a black box conforming to Definition 4.6. Because the lower-level block is successfully verified, this property can be used as an assumption over the lower-level block in the model of the higher-level block, also see Example 4.9. Lastly the higher-level block is successfully verified.

To enable compositional and hierachical model checking it is important that the property used for the verification of the block itself is the same as the property that is used for the assumption in a higher-level block. For example, it is not correct to verify the block with $AF(Q)$ and use the assumption of $AG(Q)$ in the higher-level block. The block can reach states during verification that it cannot reach when verifying the higher-level block.

Furthermore, as longs as a compositional property is in the form $AG(...)$, it is possible to model them in `nuXmv`. Using the `INVAR` specification in `nuXmv` causes the model checker to apply this invariant to all states of the model. Informally, the assumption is that the lower-level block always, complies with this specification. That the lower-level block actually conforms to this assumption is checked during the verification of that property for the lower-level block itself.

Thus, as long as properties are in the form $AG(...)$ it is possible to use them as an assumption in a higher-level block and the compositional approach is possible. For example, let $AG(Q)$ be the property of the lower-level block and $AG(P)$ the property of the higher-level block.

First, the lower-level block is successfully verified. Thus, Specification 4 holds. Then, the higher-level block is provided with the assumption of $AG(Q)$ by applying `INVAR` Q and the property $AG(P)$ is successfully verified. This can be combined into one specification, Specification 5. Thus, after successful verification, also Specification 5 holds.

$$\mathbf{AG}\,(\,Q\,) \tag{4}$$

$$\mathbf{AG}\,(\,Q\,) \rightarrow \mathbf{AG}\,(\,P\,) \tag{5}$$

As both Specification 4 and 5 hold, *modus ponens* can be applied to result into Specification 6. This proves that $AG(P)$ holds for the higher-level block.

$$\frac{\begin{array}{l}\mathbf{AG}\,(\,Q\,)\\ \mathbf{AG}\,(\,Q\,) \rightarrow \mathbf{AG}\,(\,P\,)\end{array}}{\mathbf{AG}\,(\,P\,)} \tag{6}$$

The proposed compositional approach is sound as demonstrated above, as long as properties in the form of $AG(...)$ are used. This is because of the use of the `INVAR` and that properties of blocks can be used for assumptions in higher-level blocks, thus to enable a compositional approach. This approach does not apply if AF, EG and EF properties are used. For example, replace $AG(Q)$ with $AF(Q)$ in both Specification 4 and 5. Although *modus ponens* can still be applied, the assumption $AF(Q)$ can not be modeled in `nuXmv` in the higher-level block as an `INVAR`.

5. Case study

In this chapter, the case study of this research is discussed. The machine process and PLC program used for this case study are discussed in Chapter 3.

This chapter focuses on the leadup to the model checking process for the discussed PLC program. The following steps have been implemented: First, the PLC program requires preparation for exporting. Then, information about the PLC program is extracted from the export by means of the developed TIA-XML-modcheck tool. This extracted information is then used for the creation of the `nuXmv` model. After the specification of the properties to verify, the model checking is executed. Lastly, the results and the validation of the models is discussed. This order is also displayed in Figure 13.

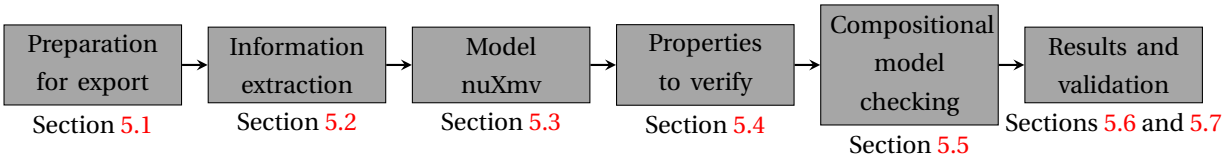


Figure 13: Overview case study section

5.1. Preparation for export

To successfully extract the information from the export, the PLC program needs some preparation before exporting. Although the export function of TIA Portal Openness supports all programming languages, the developed TIA-XML-modcheck tool, for now, only supports the FBD programming language. The LAD Programming language is also a graphical programming language but uses different constructs for compile units. The SCL programming language is text-based, which leads to a different structure of the export file.

As the focus of this research is the compositional model checking aspect of this case study, the choice is made to support only FBD and *convert the LAD blocks into FBD*. This is an integrated function of the TIA Portal engineering environment. Blocks FB110, FB330, FC329 and FC333 are written in LAD but converted to FBD. For the sequencer, two small SCL blocks, FB185 and FC185, are used. These are not exported nor translated into logic models. Their functions are manually added to the `nuXmv` model, as the functions are very trivial, being sequences consisting of mainly steps and transitions.

5.2. Information extraction

The process of information extraction consists of the export of the PLC program and conversion of the PLC program into logic models. This process is illustrated in Figure 5. The steps of this process are discussed in more detail below.

Exporting the PLC program to XML files is done with the help of the Siemens TIA Portal tool: TIA Portal Openness. A list of the XML files created by TIA Portal Openness is in Appendix C. This list is generated by the TIA-XML-modcheck tool.

The next step is for the TIA-XML-modcheck tool to import the exported PLC program and to create the logic model as discussed in Section 4.1.3. The TIA-XML-modcheck tool generates the logic models of the PLC blocks written in the FBD programming language. These logic models are the input to create the `nuXmv` models and are presented in Appendix E.

5.3. MODEL NUXMV

To model check the PLC program, the logic model first needs to be converted into a nuXmv model. The creation of the nuXmv models is done manually with the help of the logic models from the previous section. To further simplify the creation of nuXmv models, a template for the nuXmv models is created. This template is discussed in more detail in Section 4.2.1.

During the manual process of converting the logic models into nuXmv models, three exceptions came to light. The *first exception* is that the biggest block, FB110, needed optimizations to access the verification time within a usable time frame. These optimizations are at the end of this subsection. The *second exception* is for FB5, the program sequence. Because the blocks called in FB5 (FB185 and FC185) are written in the SCL programming language, there is no logic model created for these blocks. The functionality of FB185 and FC185 is very basic, and thus, these blocks are omitted from FB5 and replaced with manually created nuXmv constructs. The last and *third exception* is the assignment of multiple values to a variable. In nuXmv, it is only possible to use one `next` statement per variable. If a variable is assigned a value multiple times in a compile unit, then these assignments need to be divided into multiple states. However, this is not applicable to this case study, as in this PLC program, variables are only written a value once per compile unit. This is discussed in more detail together with a proposed solution in Section 4.1.4.

Optimizations: The first manually constructed nuXmv model of FB110 demonstrated that it would trigger a state space explosion. It was not able to verify the given set of CTL specifications within a reasonable time frame. The verification was stopped manually after a duration of 60 minutes.

After manual inspection, the nuXmv model appears to feature parts of *dead code* and *unbounded integers*. The *dead code* part consists of an integer variable and two Boolean variables that are assigned values but are never used in the PLC program. After manual deletion of the dead code parts, the model is verifiable for all named specifications. However, the verification took a fair amount of time, around 30 minutes.

After further manual inspection, *unbounded integer variables* were found. These integer variables are only used for comparing their value to a few other constant integer values (e.g., the integer values 1, 2 and 10). By manually bounding the integer variable to {0, 1, 2 ,10}, all other values are not considered for the states of the model. This last optimization brings the verification time back to a maximum of 180 seconds for all named specifications.

The nuXmv models with integer constants demonstrated a longer verification time. By increasingly adding these constants, the verification time again exceeded a reasonable time frame. Therefore, the nuXmv models of FB110 and FC333 are extended with a module named `global`. This module consists of global constants, and the use of these nuXmv constants reduced the verification time again within a reasonable time frame of a maximum of 180 seconds.

The created nuXmv models of the PLC program are in Appendix F.

5.4. PROPERTIES TO VERIFY

For model checking the nuXmv models, properties need to be specified to which the models should comply to. The properties to verify cover a broad spectrum of properties of the specified models. The specified properties are discussed in more detail in the following

paragraphs. A complete list of all checked properties per block is in Appendix G.

The first properties specified for all blocks concern *basic reachability* properties of the models. Two example properties are specified in Specifications 7 and 8. Specification 7 specifies that the variable `loc` will necessarily have the value `end`. When this property is successfully verified, this reveals that the model will necessarily reach the location where `loc=end`. It also illustrates that the model does not indefinitely loop in a compile unit before reaching `loc=end`, which thus prevents it from reaching the location where `loc=end`. However, there is an intended loop in the model which enables the cyclic execution of the block. From the location `loc=end`, the model circles back to the location `loc=start` to execute the code for another cycle. Specification 8 is more strict and defines that every path necessarily leads back to the location where `loc=end`.

In addition to Specification 7, Specification 9 checks if no loops are possible from the location `loc=end` to the location `loc=start`. It defines that the next state from `loc=end` will always be `loc=start`. Together with the check for loops before reaching the location `loc=end` from Specification 7, this completes the check for loops.

With a successful verification of these properties, we conclude that the model keeps cycling through its locations and passes through the location `loc=end`. During manual simulation, it was also possible to see the model cycle through all locations as expected.

$$\mathbf{AF}\ (\ loc = end\ ) \tag{7}$$

$$\mathbf{AG}\ \mathbf{AF}\ (\ loc = end\ ) \tag{8}$$

$$\mathbf{AG}\ (\ loc = end\ ->\ \mathbf{AX}\ (\ loc = start\ )\ ) \tag{9}$$

As discussed in Section 4.1.4, the execution of the block is finished when it reaches the end location. After reaching the end location, the newly computed output values can be used for further processing. This fact is used to check the model for *basic liveness* properties. In the end location of the model, whether the outputs can become true is verified. An example is presented in Specification 10 and reveals the verification of the output `output_bOutCommandWork` of the block FB110, which is used to control a solenoid valve. Although the functionality is not checked, whether there is a state in the model where this output can become true is checked. These kind of liveness properties are tested after the construction of the model.

$$\mathbf{EF}\ (\ loc = end\ \&\ output\_bOutCommandWork\ ) \tag{10}$$

Next is the check to verify the functionality of the `nuXmv` model of the PLC block that controls the valves (FB110). This is done with several functional properties. An important property among these functional properties for this case study is to check if there is a possibility that a solenoid valve is given a command to open and a command to close at the same time. This can lead to an undefined position of the actual valve. This property is presented in Specification 11, and it checks that in all states, both commands are never given to the valve. This is checked when `loc = end` and conforms with the PLC's I/O cycle as discussed in Section 4.1.4.

$$\begin{aligned}\mathbf{AG}\ (\ loc = end\ ->\ !\ (\ &output\_bOutCommandHome \\ &\&\ output\_bOutCommandWork\ )\ )\end{aligned} \tag{11}$$

The following two properties also check the functionality of the nuXmv model of FB110. Specification 12 checks whether the right command is output in automatic mode. Automatic mode is defined as `iInMode = 0sd16_1`. Here, it is checked that FB110 does not output the command to close the valve when it receives the command to open the valve from the step program (FB5). Specification 13 checks the situation when the emergency stop is activated. In this case study, the choice was made to give the valve no command at all. This is checked with Specification 13 below.

$$\mathbf{AG}\ (\ (\ loc = end\ \&\ bInCommandWork\ \&\ iInMode\ =\ 0sd16\_1\ )\ ->\ !\ output\_bOutCommandHome\ ) \tag{12}$$

$$\mathbf{AG}\ (\ (\ loc = end\ \&\ bInEstop\ )\ ->\ (\ !\ output\_bOutCommandHome\ \&\ !\ output\_bOutCommandWork\ )\ ) \tag{13}$$

With the properties described above and some additional properties, whether FB110 properly executes the commands it receives from the step program is verified. The valve cannot be controlled in an incorrect manner when it complies with the properties above. The part that remains is the step program FB5. With the following properties, whether FB5 sends the correct commands to FB110 is verified. When FB5 is programmed incorrectly, it could be possible to send the open command to both valves at the same time.

Specification 14 checks if FB5 only sends the command to open one of the two liquid supply valves and never both valves at the same time. Specification 15 checks that if one valve is sent a command to open, the other two valves are not given a command to open. This property is displayed for the supply valve of liquid 1, but the two other remaining combinations are also verified. These are in Appendix G. Specification 16 even checks that when a valve is sent the open command, the other valves are physically in the closed state.

$$\mathbf{AG}\ (\ loc = end\ ->\ !\ (\ bVlvLiq1\_Open\ \&\ bVlvLiq2\_Open\ )\ ) \tag{14}$$

$$\mathbf{AG}\ (\ (\ loc = end\ \&\ bVlvLiq1\_Open\ )\ ->\ !\ (\ bVlvDrn\_Open\ |\ bVlvLiq2\_Open\ )\ ) \tag{15}$$

$$\mathbf{AG}\ (\ (\ loc = end\ \&\ bVlvLiq1\_Open\ )\ ->\ (\ Ib\_VlvDrnClosed\ \&\ Ib\_VlvLiq2Closed\ )\ ) \tag{16}$$

A complete overview of all verified properties is in Appendix G.

5.5. Compositional model checking

In this section, the compositional reasoning and model checking are discussed in more detail. First, the use of black boxes in the nuXmv models is presented. (This is as introduced in Section 4.2.1.) Then, it is illustrated that through using CEGAR, it is possible to create verification properties using compositional reasoning.

In Figure 4 from Section 3 and in Appendix D, it is clear that some blocks execute lower-level blocks. The blocks executing lower-level blocks are as follows: FB2, FB3, FB4, FB5 and FB110. In this case study, however, only FB110 is of interest. FB2, FB3 and FB4 only execute their own instance of FB110; no further logic is added in FB2, FB3 and FB4. The executed lower-level blocks in FB5 are programmed in the SCL programming language, which cannot be directly converted into FBD. The FBD programming language is mainly used in this case study and is the only supported language of the developed TIA-XML-modcheck tool.

FB110 is of interest because of the logic in this block and the execution of three lower-level blocks that contain relatively simple logic. The three executed lower-level blocks are FB330, FC329 and FC333.

The approach of defining the execution of lower-level blocks as black boxes is already described in Section 4.2.1. The application of black boxes can also be found in the `nuXmv` models (see Appendix F). The variables that are connected to outputs of lower-level blocks are modeled as displayed in Listing 7. When, however, one of these variables has an impact on a specified property, it is possible that the used black box definition is specified too loosely. This causes the property to fail, although the system under verification does comply with the property. With this counterexample and the CEGAR method, the properties are redefined using a compositional approach, as explained in Section 4.2.2.

An example of a property that failed verification due to the implementation of a black box is the property in Specification 17. This is the functionality of FB330, which consolidates all errors generated by FB110 into one error signal that is assigned to an output of FB110.

$$\textbf{AG}\ (\ (\ \mathit{loc} = \mathit{end}\ \&\ \mathit{output\_ERROR\_Valve\_NoHomeFeedback} \\ \&\ \mathit{output\_bOutAuto}\ )\ ->\ \mathit{output\_bOutError}\ ) \tag{17}$$

The property in Specification 17 fails because of the following reasons. The variable `output_bOutError` is assigned the value of variable `inout_HMI_ValveControl_bError`. Variable `inout_HMI_ValveControl_bError` is assigned a non-deterministic value by means of a black box because normally this variable is assigned to an output of the lower-level block FB330; this is the output named `output_bOutErrorExists`. Variable `output_bOutError` is thus also assigned a non-deterministic value, and this causes Specification 17 to fail.

The next step is applying the approach discussed in Section 4.2.2 and adding an assumption about the lower-level block (FB330) in this block (FB110). This is done by adding the property of the lower-level block to the model as discussed in Section 4.2.2. The property has to define output `inout_HMI_ValveControl_bError` of FB330. This is then added to the `nuXmv` model as displayed below in Specification 18.

$$\textbf{INVAR}\ (\ (\ \mathit{loc} = \mathit{end}\ \&\ \mathit{output\_ERROR\_Valve\_NoHomeFeedback}\ )\ ->\ \mathit{inout\_HMI\_ValveControl\_bError}\ )$$

$$\textbf{AG}\ (\ (\ \mathit{loc} = \mathit{end}\ \&\ \mathit{output\_ERROR\_Valve\_NoHomeFeedback} \\ \&\ \mathit{output\_bOutAuto}\ )\ ->\ \mathit{output\_bOutError}\ ) \tag{18}$$

Now, the value of variable `inout_HMI_ValveControl_bError` is not a non-deterministic value anymore. Informally, if variable `output_ERROR_Valve_NoHomeFeedback` is true, then variable `inout_HMI_ValveControl_bError` must also be true, which is basically part of the logic of FB330.

All properties for compositional model checking are in Appendix H.

5.6. Results

The results of this case study for the safety of the described process depend on the potential for contact between the two liquids. To analyze this, the PLC program can be divided into

two parts. One part is the control of a solenoid valve, which is done by FB110, called a *driver*. The other part is the sequence of steps controlling the overall process, called a *step program*.

The *driver* is model checked, and it is never possible to, for example, command the valve to open and close at the same time. Furthermore, the valve will only send a command to the valve when the driver receives a command to do so. These situations might sound trivial but in this case are model checked to be absolutely sure no erroneous controls are possible. See the properties of FB110 in Appendix G for all checked properties.

The *step program* sends commands to the drivers of the valves. The step program coordinates when each valve needs to close or open. Here, it is checked whether only one command to open is sent to one valve and that the other two valves must be closed. See the properties of FB5 in Appendix G for all checked properties.

Both the driver and the step program pass the model checking properties. Additionally, the lower-level blocks of the driver pass their model checking properties. This means that for the situations specified in the properties, this PLC program is safe with regards to the dosing of the two liquids.

The specified properties cover the automatic mode of the valve driver in conjunction with the step program. However, if manual operation is enabled, for example an HMI, the case study is currently not safe. Manual operation is not verified and is therefore considered unsafe. Furthermore, manual operations are not secured nor interlocked in the current PLC program.

In reality, there are a multitude of possible unsafe situations caused by, for example, failing hardware. However, these hardware faults are difficult for the PLC to recognize with the current layout of inputs and outputs. Nevertheless, this is part of the hardware design and would be revealed during a proper risk evaluation; it is not within the scope of this study.

5.7. VALIDATION

Validation of the results of the case study is precisely executed. The validation is divided into several elements, corresponding to several intermediate results of the information extraction and model checking.

The first check is completed after generation of the logic models. Each logic model is checked, line for line, against the original PLC program. This ensures that the logic models match the corresponding PLC code. A PLC program can consist of a great number of different instructions; validation is completed only for the instructions that are used in the case study. During this validation step, whether the order of PLC instructions is maintained correctly in the logic model is also verified.

The next check is done after manual conversion from logic models to `nuXmv` models. By means of the specified properties, some sanity checks are executed on the `nuXmv` models. This is achieved to ensure the models cycle through the locations and the outputs can take different values. The properties that check the functionality of the `nuXmv` models are also checked in the PLC program. This is done by means of simulation where the PLC program is loaded into a simulated PLC. The inputs of the block under testing can be manipulated to simulate the property. After the manual conversion from logic models to `nuXmv` models, these properties revealed some typos in the `nuXmv` models after the initial verification run. Together with the logic models and the simulated PLC blocks, these typos were rectified.

6. Discussion

In this chapter, several subjects are discussed. First, the advantages of the formal definitions and the support for the latest TIA Portal engineerin environment are discussed. Then, the subject of compositional model checking is reviewed. This chapter ends with the characteristics and applicability of the proposed approach.

6.1. Formal definitions

One of the valuable results of this study is the *formal definition of compile units and execution order* as discussed in Section 4.1.2. Standardization efforts in the PLC industry have had an effect but do not provide a watertight solution and lack formal definitions. An example is the IEC-61131 standard. This standard leaves room for interpretation and is too extensive to implement in an error-free manner. Formal definitions of the IEC-61131 standard can therefore be valuable. However, formal definitions concerning these topics are hard to find. Manufacturers even leave ambiguity or incompleteness in their informal definitions and explanations or do not describe them at all. Formal definitions on these topics coupled with Siemens PLCs are also absent from published literature. The importance of specific literature concerning Siemens PLCs is that every PLC manufacturer can choose different implementations that can both effect the definitions of compile units and the execution order.

6.2. Support for new engineering environment

Another important result from this study is the conversion from a graphical programming language (i.e., FBD) to a textual *logic model*. With the help of XML export files from TIA Portal with TIA Openness and the developed TIA-XML-modcheck tool, it is possible to quickly convert FBD PLC programs into the discussed logic models. The current version of the TIA-XML-modcheck tool only supports the FBD programming language; other Siemens programming languages are not yet supported. Siemens TIA Portal supports a great number of instructions, but not all instructions are currently supported by the TIA-XML-modcheck tool. The most popular instructions are supported, and at least all instructions used in the PLC program of the case study are supported. The supported instructions are in Appendix B.

The logic model is used as an intermediate model. This provides greater flexibility to the used programming languages and model checkers. Other programming languages can be added and converted to logic models. Other model checkers can be used by converting the logic models to the defined syntax of the model checker. The idea of an intermediate model is also discussed in Adiego et al.'s [1] study, although they used SCL as the input programming language, which is text-based. Darvas et al. [7] discuss the option to convert LAD and FBD to STL by means of the Siemens engineering environment. However, this is not true for TIA Portal, the most recent Siemens engineering environment, which was introduced in 2011. Here FBD and LAD can be converted to each other but not to STL. However, in this case study, a novel method is presented that takes the TIA Portal Openness XML export files as an input.

6.3. COMPOSITIONAL MODEL CHECKING

This case study also delivers valuable and promising results for the *compositional approach* of model checking hierarchical PLC programs. Modular PLC programs are becoming more the standard, and exactly this characteristic is leveraged in this study. This could improve the scalability of model checking PLC programs to a great extent, through using the proposed compositional approach. The actual improvement in scalability is not yet tested in this study, but it enables a performance comparison between this compositional approach and other approaches of model checking for PLC programs found in the literature.

The logic models together with other extracted information using the TIA-XML-modcheck tool are manually converted into `nuXmv` models. This could and should be automated; however, this did not fit the timeframe of this study. A limitation of this proposed approach occurs when a variable is assigned a value more than once in a compile unit. As discussed in Section 4.1.4, the proposed `nuXmv` models consist of a single assignment per variable per compile unit. In this case study, no situation occurred where a variable was assigned a value more than once in one compile unit. In Section 4.1.4, a solution to this limitation is proposed, but it comes at the expense of a larger state space. After the initial verification run, it was obvious that the state space of the `nuXmv` models was too large, and verification could not finish within 60 minutes. Further investigation revealed dead code and unbounded integer variables, which caused an unnecessarily large state space. After removing the dead code and bounding the integer variables, the verification could complete in a reasonable time frame of 180 seconds.

The presented CEGAR method for creating the compositional model check properties is currently accomplished manually. The creation of (compositional) properties still demands significant time and specific knowledge about model checking and CTL. The compositional approach uses the inference rule of modus ponens for CTL, which means that this approach is considered sound. The proposed method involves the use of invariants for the assumptions of the lower-level blocks. These invariants apply to every possible state of the model. This provides the limitation that, only *AG* properties can be correctly transformed into invariants. For the case study this poses no problem as the lower-level properties are all *AG* properties. For the use of properties other than *AG* with this approach, additional research is needed.

In this case study, the lower-level blocks consist of relatively basic logics, and thus the compositional properties are not complex to create. When the lower-level blocks become more complex, the creation of compositional properties becomes more difficult. Generally, a PLC programmer does not possess the competencies to create these CTL properties. This could endanger this approach. Other approaches in the literature, however, acknowledge the same issue.

6.4. CHARACTERISTICS

One characteristic of this proposed approach is the fact that the *modularity* of modern PLC programs is leveraged for compositional model checking. Hierarchical approaches (e.g., the ISA-88 standard) dictate the breakdown of a machine into smaller modules. The use of this standard has become more common over the years. This can work in favor of this novel compositional approach. The possible time savings and increase in scalability can also be an enabling factor for the model checking of real industrial PLC programs. The complexity decreases with a compositional approach. Modular sub parts can be verified without losing

the ability to verify properties of the whole PLC program.

A second characteristic of this study is the use of *Siemens TIA Portal and TIA Portal Openness* for export of the PLC program. This enables direct export of PLC program blocks. Although TIA Portal has existed for 10 years, current literature mainly uses the previous engineering environment from Siemens. Export options of these two engineering environments are not comparable nor fully compatible with one another. Thus, the approach of this study gains an advantage for both now and in the future.

6.5. APPLICABILITY

The degree of applicability depends mainly on two subjects beyond the proposed approach and tool. First, the PLC program that is used as input influences the applicability and effectiveness of the proposed approach. Second, the competencies and knowledge about model checking of the engineer who applies this approach greatly influences the applicability of this approach.

PLC programs that are structured in a hierarchical way gain the full advantage of the proposed approach. Other more old-fashioned PLC programs generally consist of highly coupled blocks and are less structured. Applying the proposed approach on these highly coupled PLC programs would be more difficult. Dividing the PLC program in multiple smaller parts becomes more complex when the PLC program is highly coupled and not hierarchically structured. The support for the TIA Portal engineering environment together with the support for the FBD graphical programming language create a starting point for a novel approach for model checking of modern PLC program, which increases future applicability.

Applicability is also largely impacted by the competencies and knowledge of the designated *engineer*. A typical PLC *engineer* has no knowledge of model checking or CTL. In any case, it is necessary for the engineer to know the basics used for model checking and more detailed knowledge about CTL to specify the properties needed for verification.

The use of the proposed compositional approach does not limit the applicability. The approach is considered sound as demonstrated in Section 4.2.2. The used invariant is applicable to every state of the model and can therefore only be used when the property in the lower-level block is of the form $AG(...)$.

To increase applicability of the *TIA-XML-modcheck* too, the creation of `nuXmv` models could be automated; this is currently done manually. By extending the supported programming languages for the *TIA-XML-modcheck* tool, applicability can be further increased. Now, only FBD blocks are supported, but additional programming languages could easily be added.

7. Conclusion

In this study, numerous steps for compositional model checking of PLC programs are discussed. From the PLC export of the PLC program, a graph, which is used for the creation of logic models, is extracted. These logic models are manually converted into `nuXmv` models and are then used for model checking. With the help of CEGAR, compositional model checking is presented for the verification of PLC programs. To enable these conversions, several formal definitions are proposed in this research. A case study is used to demonstrate and present results of the proposed approach. The results reveal the applicability and several advantages of the proposed approach. In the following paragraphs, the steps are summarized in greater detail.

The first step is *information extraction*, where the PLC program is successfully exported, using a novel approach from the engineering environment TIA Portal with the help of TIA Openness. Several formal definitions are provided and discussed to develop a tool for the conversion from export to a logic model. Definitions are provided for compile units, execution order and the logic model. The algorithm for the conversion from export to logic model is also offered and discussed in more detail. This approach enables a fast and clear conversion from the original PLC program to a logic model. The conversion to the `nuXmv` model, however, is still manually executed.

The second step consists of the *compositional model checking* of the `nuXmv` models. For this step, the `nuXmv` model is used, and lower-level blocks are considered as black boxes. The definition for black boxes is also provided and discussed in this study. Then, using CEGAR, the next abstraction refinement steps are determined. These are then added to the `nuXmv` model. Using this approach has allowed for the creation of compositional verification properties and thus compositional model checking.

This study also discusses the application of the proposed approach to a given *case study*. This case study describes how the proposed approach is applied and how all the steps of the approach are executed. With the case study, a realistic example application for compositional model checking of a PLC program is provided. Because of the hierarchical structure of the PLC program, the results consist of two parts: the part of the step program and the part of the driver. The step program defines when something needs to happen, and the driver controls the actual equipment and defines how the controls need to occur. It also monitors the state of the equipment. For the automatic mode of this case study, it has been revealed that a dangerous situation cannot occur. Thus, the safety of the process is guaranteed from a software standpoint.

Contributions of this study can be grouped into three categories. The first category consists of *formal definitions* stated in this study. Relevant formal definitions about, for example, programming languages and manufacturer specific implementations are not provided by mentioned related studies. Available documentation mainly consists of informal descriptions, and they tend to be ambiguous. The second category consists of *support for modern PLC programs*. Direct exports from TIA Portal with TIA Openness are supported by the developed tool. In addition, modern hierarchical and modular PLC programs can be leveraged through use of its structure and modularity for compositional model checking. The third category consists of *the logic model and compositional model checking of PLC programs*. Use of the logic model enables a free choice in model checker. In this research, `nuXmv` is used due to similarities between the `nuXmv` and logic model syntax. Additionally, CEGAR is used to enable the compositional approach discussed in this study.

7.1. FUTURE WORK

The proposed approach of this case study has some elements that could benefit from further research and work. This further research is mainly needed to enlarge the applicability of the developed tool and further substantiate the proposed compositional approach and its scalability. Further research is also needed in the direction of formal definitions of the PLC and the programming languages to access a solid foundation for formal verification.

The largest gain for applicability would be the possibility of an automated conversion from the logic model to the `nuXmv` model, which is currently achieved manually. Another task that currently demands significant time is the creation of compositional verification properties for the `nuXmv` models. It would be a great feature if this could be (at least partly) automated.

Currently, only the FBD programming language is supported with at least all instructions occurring in the PLC program of the case study. To increase applicability, future work could focus on extending the TIA-XML-modcheck tool with support for more or all programming languages available in TIA Portal. Parts of a PLC program can be programmed in multiple programming languages. Therefore, the support for more programming languages would be a welcome addition. Depending on the PLC program, it is now possible that the PLC program uses an instruction that is not supported by the TIA-XML-modcheck tool. It would be another positive addition to further extend the support of instructions used in PLC programs. Currently, the most common and all basic instructions are supported (see Appendix B).

An interesting follow-up to this study would be to compare the performance of this compositional approach to approaches of other state-of-the-art tools. Mainly scalability is of great interest because this is where the approach of this study should excel.

The lack of formal semantics of PLC standards as mentioned in Section 6.1 instigates a valuable research direction for future work. The lack of formal definitions found throughout this research has resulted in the creation of a few formal definitions used here. These formal definitions, however, are not yet complete, and this surely requires more work. This direction of future work can form the basis for further research in formal verification of PLCs in general. This is considered a very important condition for further research in model checking PLC programs. For formal verification of PLC programs, it is crucial to have solid formal definitions about PLC programs.

REFERENCES

[1] B. F. Adiego, D. Darvas, J.-C. Tournier, E. B. Viñuela, J. O. Blech, and V. M. Suárez. Automated generation of formal models from ST control programs for verification purposes. (CERN-ACC-NOTE-2014-0037), 2014. 2, 8, 9, 11, 12, 41

[2] B. F. Adiego, D. Darvas, E. B. Viñuela, J.-C. Tournier, V. M. G. Suárez, and J. O. Blech. Modelling and formal verification of timing aspects in large PLC programs. *IFAC Proceedings Volumes*, 47(3):3333 – 3339, 2014. 19th IFAC World Congress. 12

[3] E. R. Alphonsus and M. O. Abdullah. A review on the applications of programmable logic controllers (PLCs). *Renewable and Sustainable Energy Reviews*, 60:1185–1205, 2016. 1

[4] S. Biallas, J. Brauer, and S. Kowalewski. Arcade.PLC: A verification platform for programmable logic controllers. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 338–341. ACM, 2012. 2, 8, 11, 12

[5] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. 8, 32

[6] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pages 353–362, 1989. 2, 13, 30

[7] D. Darvas, I. Majzik, and E. B. Viñuela. PLC program translation for verification purposes. *Periodica Polytechnica Electrical Engineering and Computer Science*, 61(2):151–165, 2017. 1, 2, 8, 11, 12, 41

[8] D. Darvas, E. B. Viñuela, and V. Molnár. PLCverif re-engineered: An open platform for the formal analysis of PLC programs. In *Proceedings of the 17th International Conference on Accelerator and Large Experimental Physics Control Systems*. JACoW, 2019. Waiting for review. 1, 2, 8, 9, 11, 12

[9] D. Darvas, E. B. Viñuela, and I. Majzik. What is Special About PLC Software Model Checking? In *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, pages 1781–1786. JACoW, 2018. 4, 9

[10] O. De Smet and O. Rossi. Verification of a controller for a flexible manufacturing line written in ladder diagram via model-checking. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, volume 5, pages 4147–4152, 2002. 7, 11, 12

[11] E. P. Enoiu, D. Sundmark, and P. Pettersson. Model-based test suite generation for function block diagrams using the UPPAAL model checker. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 158–167, 2013. 6, 11, 12

[12] E. Jee, S. Jeon, S. D. Cha, K. Y. Koh, J. Yoo, G.-Y. Park, and P.-H. Seong. FBDVerifier: Interactive and visual analysis of counterexample in formal verification of function block diagram. *J. Res. Pract. Inf. Technol.*, 42:171–188, 2010. 7, 11, 12

[13] E. Jee, D. Shin, S. Cha, J.-S. Lee, and D.-H. Bae. Automated test case generation for FBD programs implementing reactor protection system software. *Software Testing, Verification and Reliability*, 24(8):608–628, 2014. 10

[14] S. Kottler, M. Khayamy, S. R. Hasan, and O. Elkeelany. Formal verification of ladder logic programs using NuSMV. In *SoutheastCon 2017*, pages 1–5, 2017. 7, 11, 12

[15] S. Lampérière-Couffin, O. Rossi, J. Roussel, and J. Lesage. Formal validation of PLC programs: A survey. In *1999 European Control Conference (ECC)*, pages 2170–2175, 1999. 4

[16] J. Li, A. Qeriqi, M. Steffen, and I. C. Yu. Automatic translation from FBD-PLC-programs to NuSMV for model checking safety-critical control systems. In *Norsk Informatikkonferanse (NIK)*, 2016. 1, 2, 7, 11, 12

[17] J. Nellen, K. Driessen, M. Neuhäusser, E. Ábrahám, and B. Wolters. Two CEGAR-based approaches for the safety verification of PLC-controlled plants. *Information Systems Frontiers*, 18(5):927–952, Oct. 2016. 1, 2, 8, 11, 12

[18] T. Ovatman, A. Aral, D. Polat, and A. Ünver. An overview of model checking practices on verification of PLC software. *Software & Systems Modeling*, 15:937–960, 12 2014. 1, 4, 10

[19] A. Pakonen, T. Mätäsniemi, J. Lahtinen, and T. Karhela. A toolset for model checking of PLC software. In *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–6, 2013. 7, 11, 12

[20] O. Pavlovic and H. Ehrich. Model checking plc software written in function block diagram. In *2010 Third International Conference on Software Testing, Verification and Validation*, pages 439–448, 2010. 7, 11, 12

[21] O. Pavlovic, R. Pinger, and M. Kollmann. Automation of formal verification of PLC programs written in IL. In *Proceedings of 4th International Verification Workshop in connection with CADE-21 (VERIFY'07)*, pages 152–163. CEUR, 2007. 5

[22] A. Qeriqi. A PLC-NuSMV compiler for model checking safety-critical control systems. Master's thesis, University of Oslo, Boks 1072 Blindern, NO-0316 OSLO, Norway, May 2016. 2, 7

[23] S. Rösch, S. Ulewicz, J. Provost, and B. Vogel-Heuser. Review of model-based testing approaches in production automation and adjacent domains—current challenges and research gaps. *Journal of Software Engineering and Applications*, 08:499–519, 01 2015. 6

[24] D. Soliman and G. Frey. Verification and validation of safety applications based on plcopen safety function blocks using timed automata in UPPAAL. *IFAC Proceedings Volumes*, 42(5):34 – 39, 2009. 6, 11, 12

[25] D. Soliman, K. Thramboulidis, and G. Frey. Function block diagram to UPPAAL timed automata transformation based on formal models. *IFAC Proceedings Volumes*, 45(6):1653 – 1659, 2012. 1, 2, 6, 11, 12

[26] J. Song, E. Jee, and D.-H. Bae. FBDTester 2.0: Automated test sequence generation for FBD programs with internal memory states. *Sci. Comput. Program.*, 163:115–137, 2018. 10

[27] P. van den Helder. Verifcation of PLC code used at CERN. Master's thesis, Eindhoven University of Technology, Metaforum, Groene Loper 5, 5612 AP Eindhoven, Netherlands, May 2016. 9

[28] F. Verbeek, P. M. Yaghini, A. Eghbal, and N. Bagherzadeh. A compositional approach for verifying protocols running on on-chip networks. *IEEE Transactions on Computers*, 67(7):905–919, July 2018. 2, 13

[29] A. Vörös, D. Darvas, Ákos Hajdu, A. Klenik, K. Marussy, V. Molnár, T. Bartha, and I. Majzik. Industrial applications of the PetriDotNet modelling and analysis tool. *Science of Computer Programming*, 157:17 – 40, 2018. 10

[30] A. N. I. Wardana, J. Folmer, and B. Vogel-Heuser. Automatic program verification of continuous function chart based on model checking. In *2009 35th Annual Conference of IEEE Industrial Electronics*, pages 2422–2427, 2009. 6, 11, 12

[31] H. Willems. Compact Timed Automata for PLC Programs. Technical report, University of Nijmegen, 11 1999. 5, 6, 11, 12

[32] Y. Xie. Formal modeling and verification of train control systems. Master's thesis, École Centrale de Lille, Cité scientifique, CS 20048, 59651 Villeneuve d'Ascq cedex, France, 2019. 2, 13

[33] Y. Zheng, G. Luo, J. Sun, J. Zhang, and Z. Wang. PLC modeling and checking based on formal method. *Journal of Software Engineering and Applications*, 3:1054–1059, 2010. 10

[34] B. Zoubek, J.-M. Roussel, and M. Kwiatkowska. Towards automatic verification of ladder logic programs. In *Proc. IMACS Multiconference on Computational Engineering in Systems Applications (CESA)*, 2003. 1, 2, 6, 11, 12

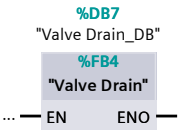
A. APPENDIX: PLC PROGRAM

Main [OB1]

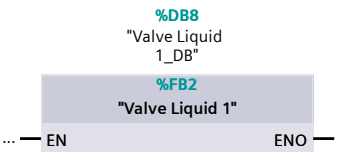
Main Properties					
General					
Name	Main	**Number**	1	**Type**	OB
Language	FBD	**Numbering**	Automatic		
Information					
Title	"Main Program Sweep (Cycle)"	**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value	Comment
Input			
Initial_Call	Bool		Initial call of this OB
Remanence	Bool		=True, if remanent data are available
Temp			
Constant			

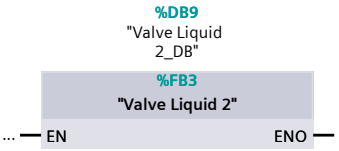
Network 1:



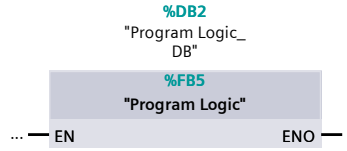
Network 2:



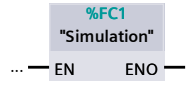
Network 3:



Network 4:



Network 5:



Network 6:

Program Logic [FB5]

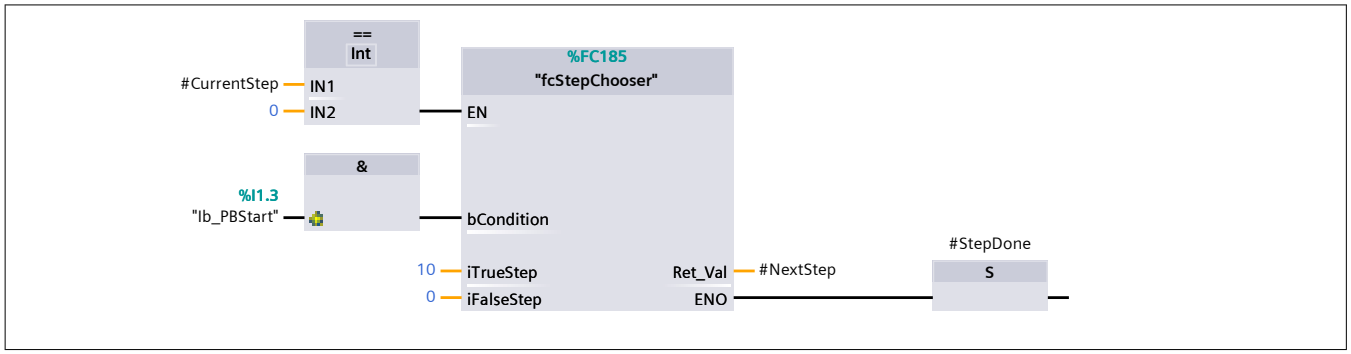
Program Logic Properties					
General					
Name	Program Logic	**Number**	5	**Type**	FB
Language	FBD	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
StepSeq	"fbStepSequencer"			True	True	True	False		Modified by Ken Brey 2010-08-25 KLB. Added Initialization inputs and logic. Modified by Ken Brey 2006-08-08 Removed initialization of inputs. Initializing inputs doesn't do anything. The input value if un-wired will be the last value set for that input in a different call. The initialized value in the declaration section does not set it back to default if unwired. Allways supply values for all inputs. Modified by Tim Jager 2005-12-13 no block Id in error message Modified by Boris: branched -> simplified version, no errors, no timeout; still have to use static memory for TON/CurStep (might be fixed later using in/out structure) Modified by Nick Shea: 2012-03-30 Re-instated step mode
Input									
iInStep	Int	0	Non-retain	False	False	False	False		Constant: current step required for the FB to run
iInNextStep	Int	0	Non-retain	False	False	False	False		Next step to go to if step is done
bInStepDone	Bool	false	Non-retain	False	False	False	False		if step done logic is true, then we move from IN_STEP to NEXT_STEP

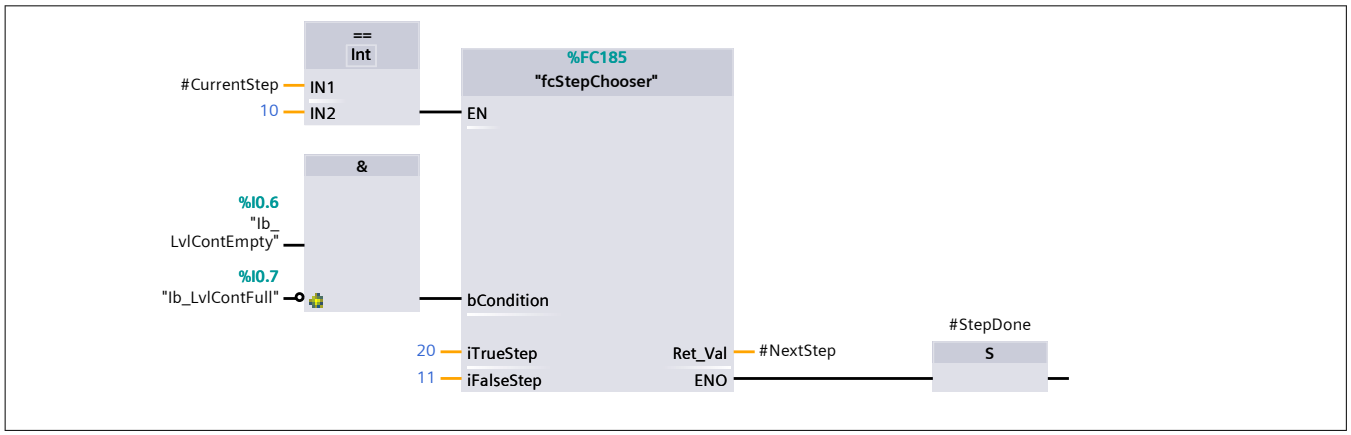
Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
tInStepDelay	Time	T#0ms	Non-retain	False	False	False	False		After the step done input is true, we wait for x seconds before moving to next
bInStepMode	Bool	false	Non-retain	False	False	False	False		Pause sequence at completion of current step
bInStepAdvance	Bool	false	Non-retain	False	False	False	False		If paused, trigger next step
bInit	Bool	false	Non-retain	False	False	False	False		
iInitStep	Int	0	Non-retain	False	False	False	False		
▼ Output									
bOutEnterEvent	Bool	false	Non-retain	False	False	False	False		
bOutExitEvent	Bool	false	Non-retain	False	False	False	False		
bOutPaused	Bool	false	Non-retain	False	False	False	False		Sequence is currently paused.
iOutCurrentStep	Int	0	Non-retain	False	False	False	False		output current step
InOut									
▼ Static									
▼ TON_StepDelay	TON_TIME		Non-retain	False	False	False	False		used to delay moving to the next step.
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
iCurrentStep	Int	0	Non-retain	False	False	False	False		Current step of the sequencer
bAlreadyInThisStep	Bool	false	Non-retain	False	False	False	False		
CurrentStep	Int	0	Non-retain	True	True	True	True		
NextStep	Int	0	Non-retain	True	True	True	False		
StepDone	Bool	false	Non-retain	True	True	True	False		
Temp									
Constant									

Network 1: 0 -- Idle

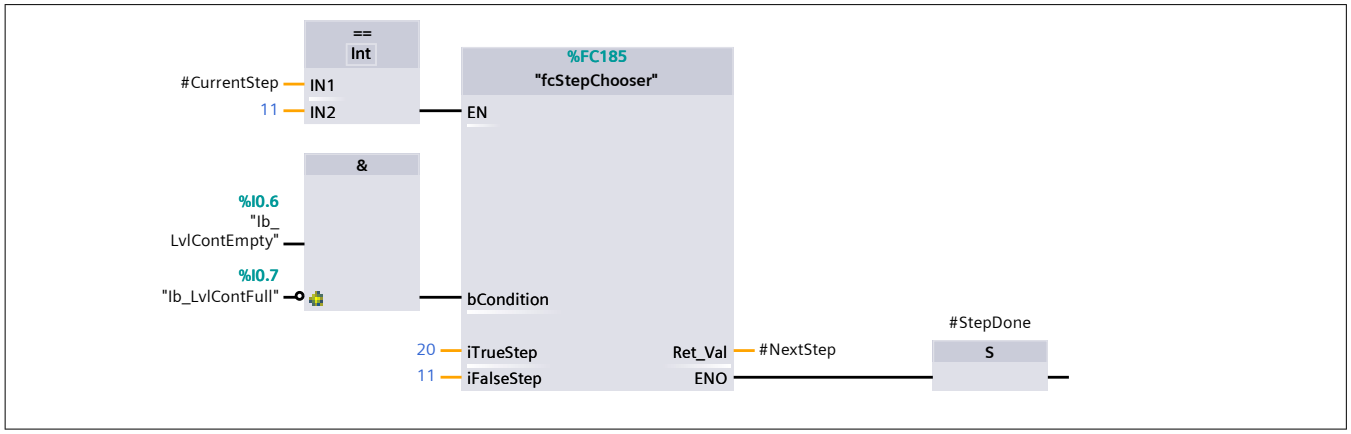


Network 3: 10 -- Controleer bak op aanwezigheid vloeistof

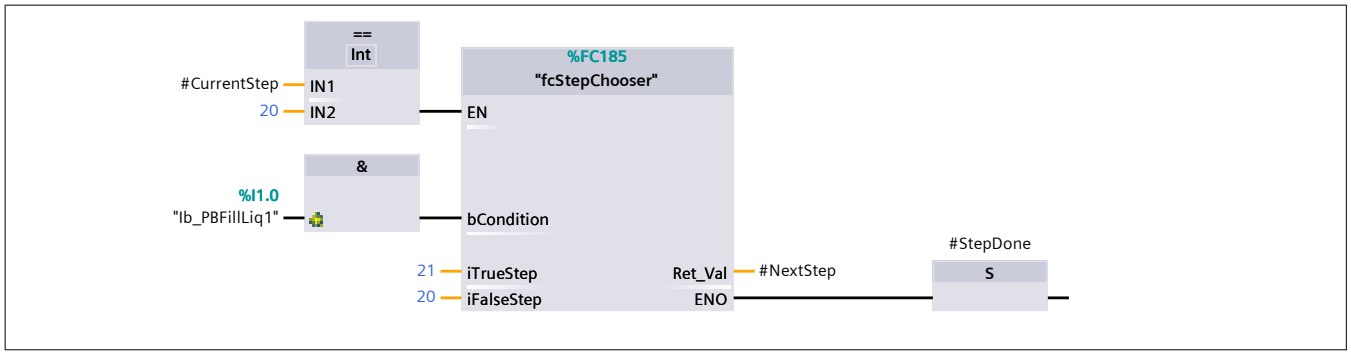
vloeistof aanwezig 10->11
geen vloeistof aanwezig 10->20



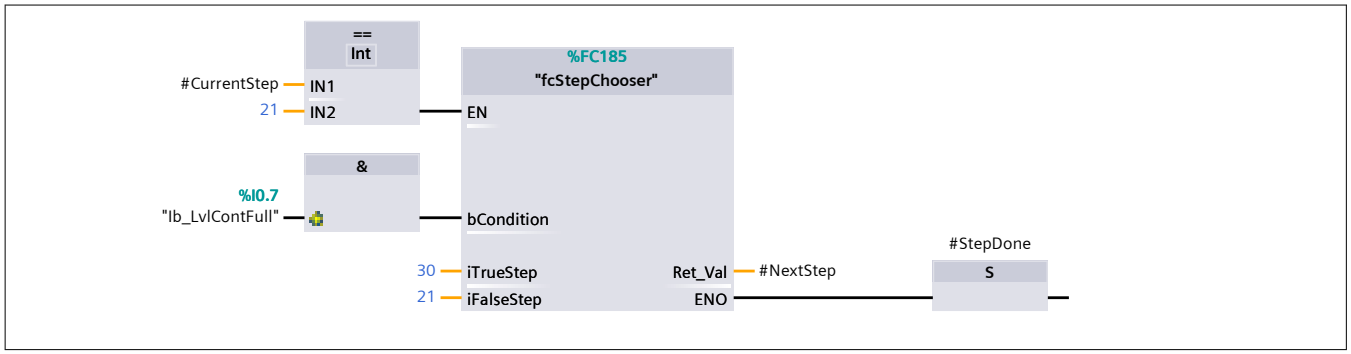
Network 4: 11 -- Vloeistof aanwezig, leeg maken tot bak leeg



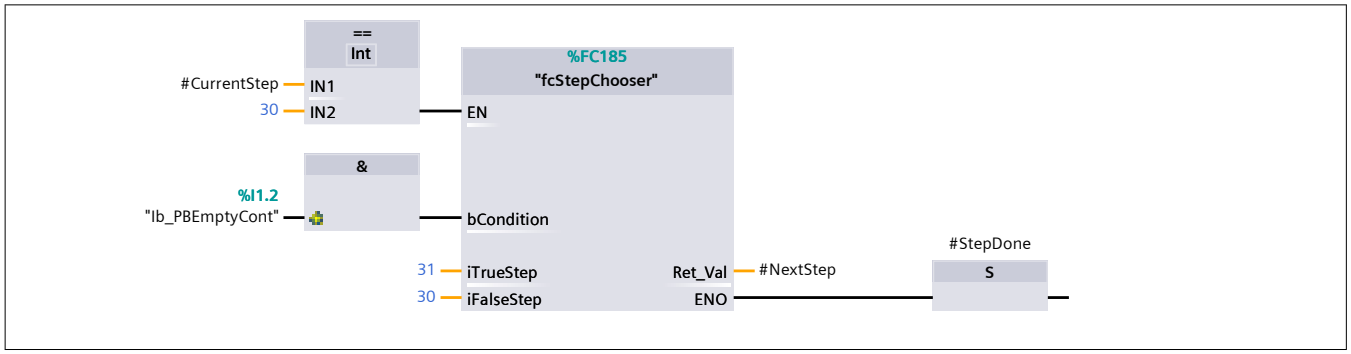
Network 6: 20 -- Aanvraag vloeistof 1



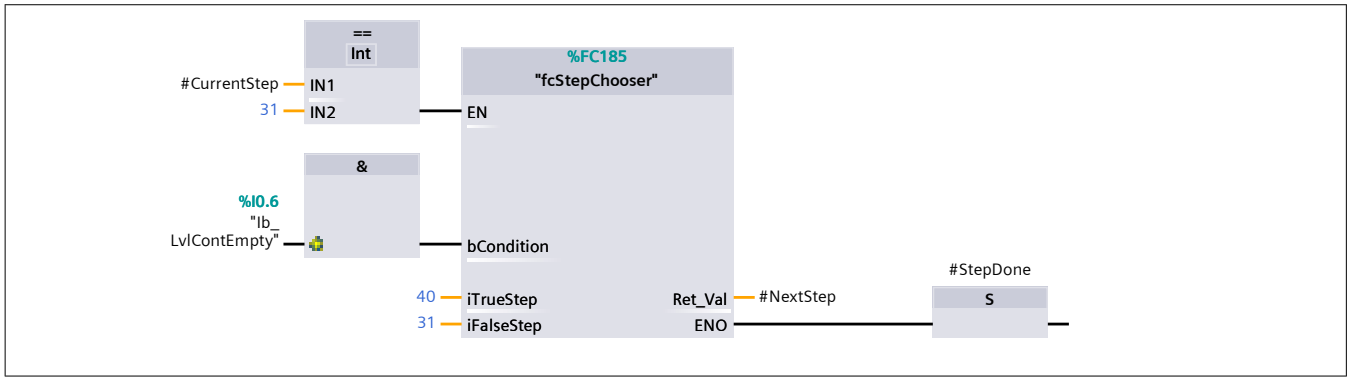
Network 7: 21 -- Vullen bak vloeistof 1 tot bak vol



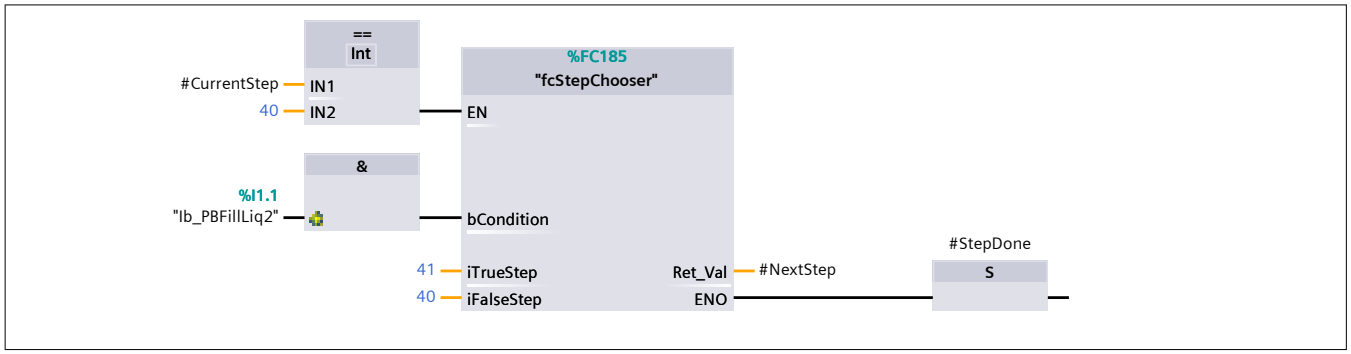
Network 9: 30 -- Aanvraag leegmaken bak vloeistof 1



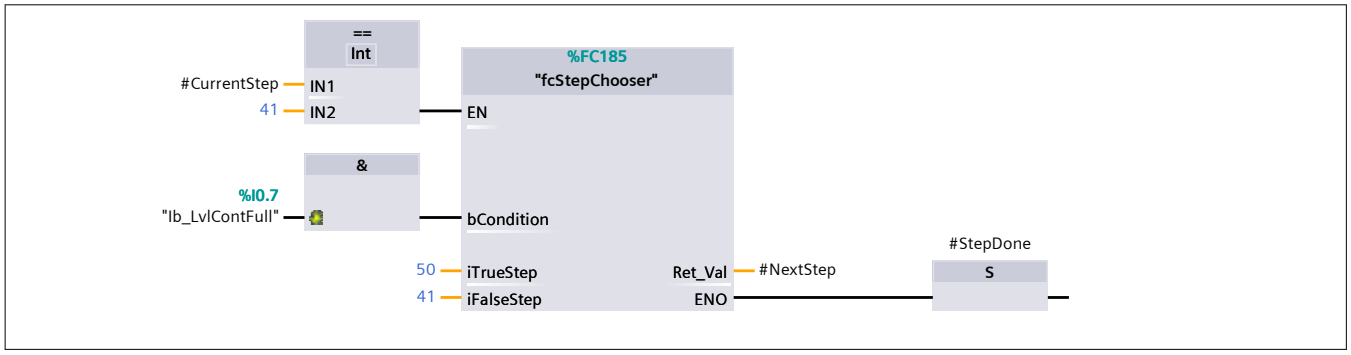
Network 10: 31 -- Leeg maken bak vloeistof 1



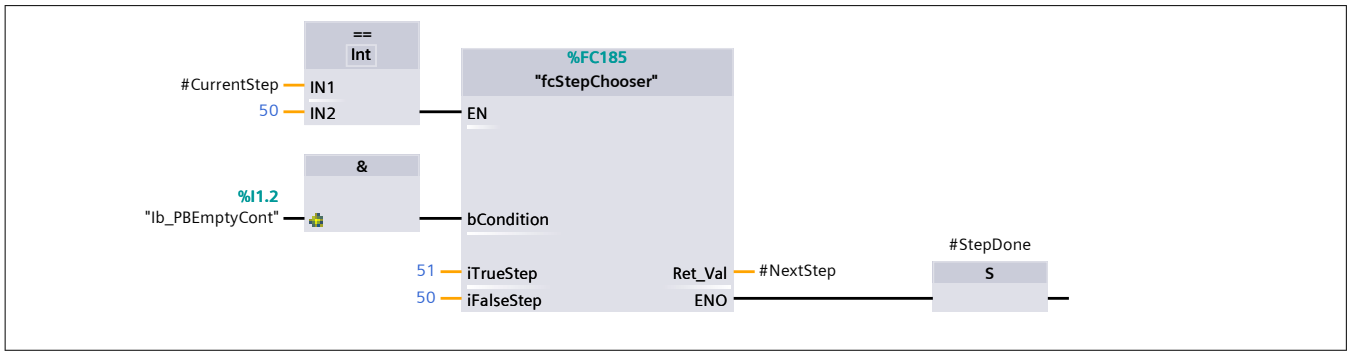
Network 12: 40 -- Aanvraag vloeistof 2



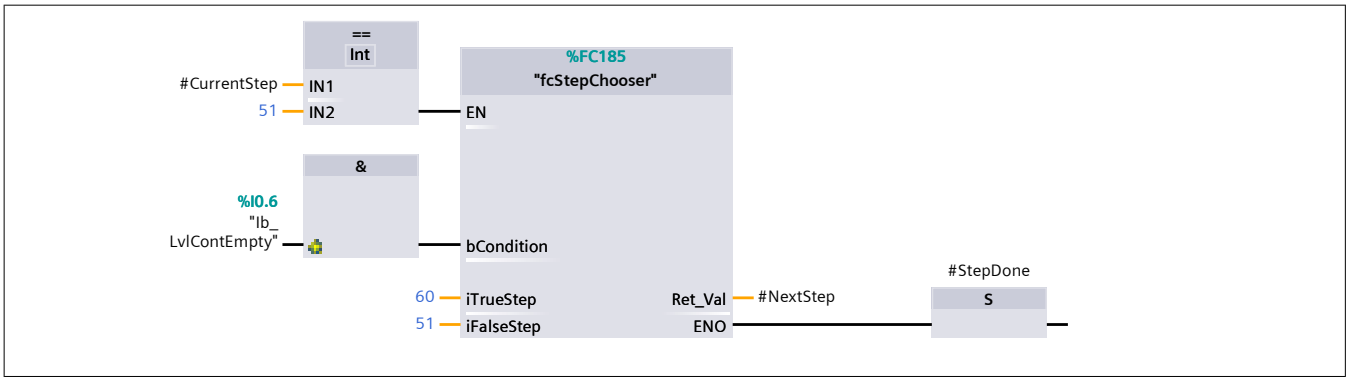
Network 13: 41 -- Vullen bak vloeistof 2



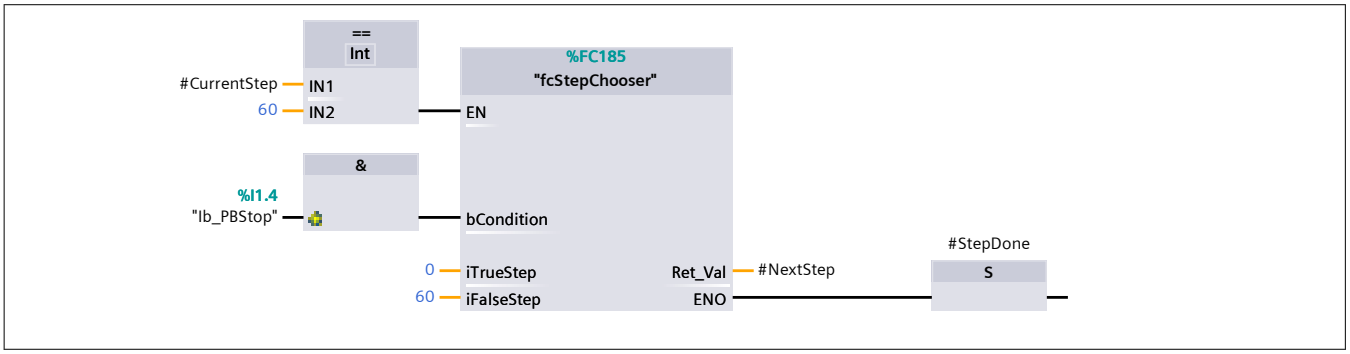
Network 15: 50 -- Aanvraag leegmaken bak vloeistof 2



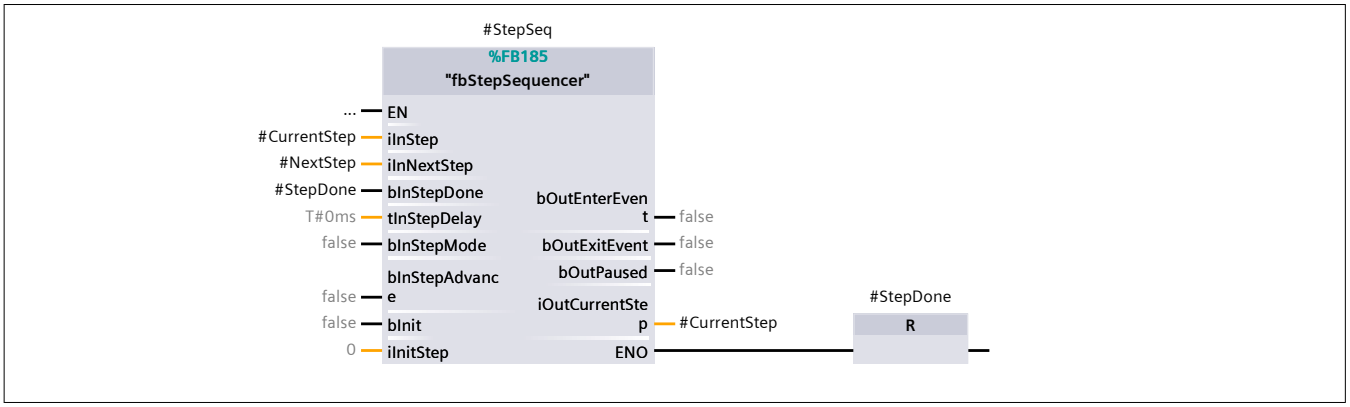
Network 16: 51 -- Leeg maken bak vloeistof 2



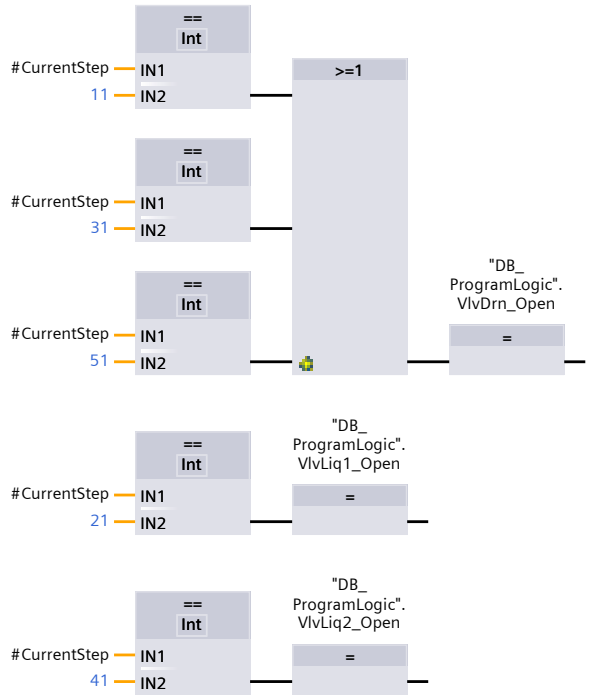
Network 18: 60 -- Cyclus klaar



Network 20: Step Sequencer



Network 22: Outputs


==
Int
#CurrentStep
IN1
11
IN2
>=1
==
Int
#CurrentStep
IN1
31
IN2
==
Int
#CurrentStep
IN1
51
IN2
"DB_
ProgramLogic".
VlvDrn_Open
=
==
Int
#CurrentStep
IN1
21
IN2
"DB_
ProgramLogic".
VlvLiq1_Open
=
==
Int
#CurrentStep
IN1
41
IN2
"DB_
ProgramLogic".
VlvLiq2_Open
=

Valve Drain [FB4]

Valve Drain Properties					
General					
Name	Valve Drain	**Number**	4	**Type**	FB
Language	FBD	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
▼ VlvDrn	"fbValve_Solenoid"			True	True	True	False		
▼ Input									
tInTimeout	Time	T#0ms	Non-retain	False	False	False	False		Timeout time for an actuator's feedback to activate before giving a fault
iInMode	Int	0	Non-retain	False	False	False	False		Mode Selection
bInEstop	Bool	false	Non-retain	False	False	False	False		Estop
bInSignalHome	Bool	false	Non-retain	False	False	False	False		Home position feedback
bInSignalWork	Bool	false	Non-retain	False	False	False	False		Work position feedback
bInEnable	Bool	false	Non-retain	False	False	False	False		Home and Work Enabled
bInCommandWork	Bool	false	Non-retain	False	False	False	False		Move to work position in automatic mode
bInResetError	Bool	false	Non-retain	False	False	False	False		Reset Error
bInSimulate	Bool	false	Non-retain	False	False	False	False		Activate device simulation
▼ Output									
bOutCommandHome	Bool	false	Non-retain	False	False	False	False		Home position command
bOutCommandWork	Bool	false	Non-retain	False	False	False	False		Work position command

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bOutActiveHome	Bool	false	Non-retain	False	False	False	False		Valve is in home position
bOutActiveWork	Bool	false	Non-retain	False	False	False	False		Valve is in work position
bOutAuto	Bool	false	Non-retain	False	False	False	False		Block in auto mode
bOutError	Bool	false	Non-retain	False	False	False	False		Error exists
▼ ERROR_Valve	"udtError_Valve"		Non-retain	False	False	False	False		Valve error structure
NoHomeFeedback	Bool	false	Non-retain	False	False	False	False		Home position feedback not active
NoWorkFeedback	Bool	false	Non-retain	False	False	False	False		Work position feedback not active
HomeFeedbackStillActive	Bool	false	Non-retain	False	False	False	False		Home position feedback still active
WorkFeedbackStillActive	Bool	false	Non-retain	False	False	False	False		Work position feedback still active
▼ InOut									
HMI_ValveControl	"udtHMI_ValveControl"			False	False	False	False		HMI valve control
▼ Static									
▼ ErrorScroller	"fbErrorScroller"			False	False	False	False		Sub-block to handle error scroller
▼ Input									
bInError01	Bool	false	Set in IDB	False	False	False	False		Error 1 In Error Array
bInError02	Bool	false	Set in IDB	False	False	False	False		Error 2 In Error Array
bInError03	Bool	false	Set in IDB	False	False	False	False		Error 3 In Error Array
bInError04	Bool	false	Set in IDB	False	False	False	False		Error 4 In Error Array
bInError05	Bool	false	Set in IDB	False	False	False	False		Error 5 In Error Array
bInError06	Bool	false	Set in IDB	False	False	False	False		Error 6 In Error Array
bInError07	Bool	false	Set in IDB	False	False	False	False		Error 7 In Error Array
bInError08	Bool	false	Set in IDB	False	False	False	False		Error 8 In Error Array

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Set-point	Supervision	Comment
bInError09	Bool	false	Set in IDB	False	False	False	False		Error 9 In Error Array
bInError10	Bool	false	Set in IDB	False	False	False	False		Error 10 In Error Array
bInError11	Bool	false	Set in IDB	False	False	False	False		Error 11 In Error Array
bInError12	Bool	false	Set in IDB	False	False	False	False		Error 12 In Error Array
bInError13	Bool	false	Set in IDB	False	False	False	False		Error 13 In Error Array
bInError14	Bool	false	Set in IDB	False	False	False	False		Error 14 In Error Array
bInError15	Bool	false	Set in IDB	False	False	False	False		Error 15 In Error Array
bInError16	Bool	false	Set in IDB	False	False	False	False		Error 16 In Error Array
bInError17	Bool	false	Set in IDB	False	False	False	False		Error 17 In Error Array
bInError18	Bool	false	Set in IDB	False	False	False	False		Error 18 In Error Array
bInError19	Bool	false	Set in IDB	False	False	False	False		Error 19 In Error Array
bInError20	Bool	false	Set in IDB	False	False	False	False		Error 20 In Error Array
bInError21	Bool	false	Set in IDB	False	False	False	False		Error 21 In Error Array
bInError22	Bool	false	Set in IDB	False	False	False	False		Error 22 In Error Array
bInError23	Bool	false	Set in IDB	False	False	False	False		Error 23 In Error Array
bInError24	Bool	false	Set in IDB	False	False	False	False		Error 24 In Error Array
bInError25	Bool	false	Set in IDB	False	False	False	False		Error 25 In Error Array
bInError26	Bool	false	Set in IDB	False	False	False	False		Error 26 In Error Array
bInError27	Bool	false	Set in IDB	False	False	False	False		Error 27 In Error Array
bInError28	Bool	false	Set in IDB	False	False	False	False		Error 28 In Error Array
bInError29	Bool	false	Set in IDB	False	False	False	False		Error 29 In Error Array
bInError30	Bool	false	Set in IDB	False	False	False	False		Error 30 In Error Array
▼ Output									
bOutErrorExists	Bool	false	Set in IDB	False	False	False	False		An Error Exists

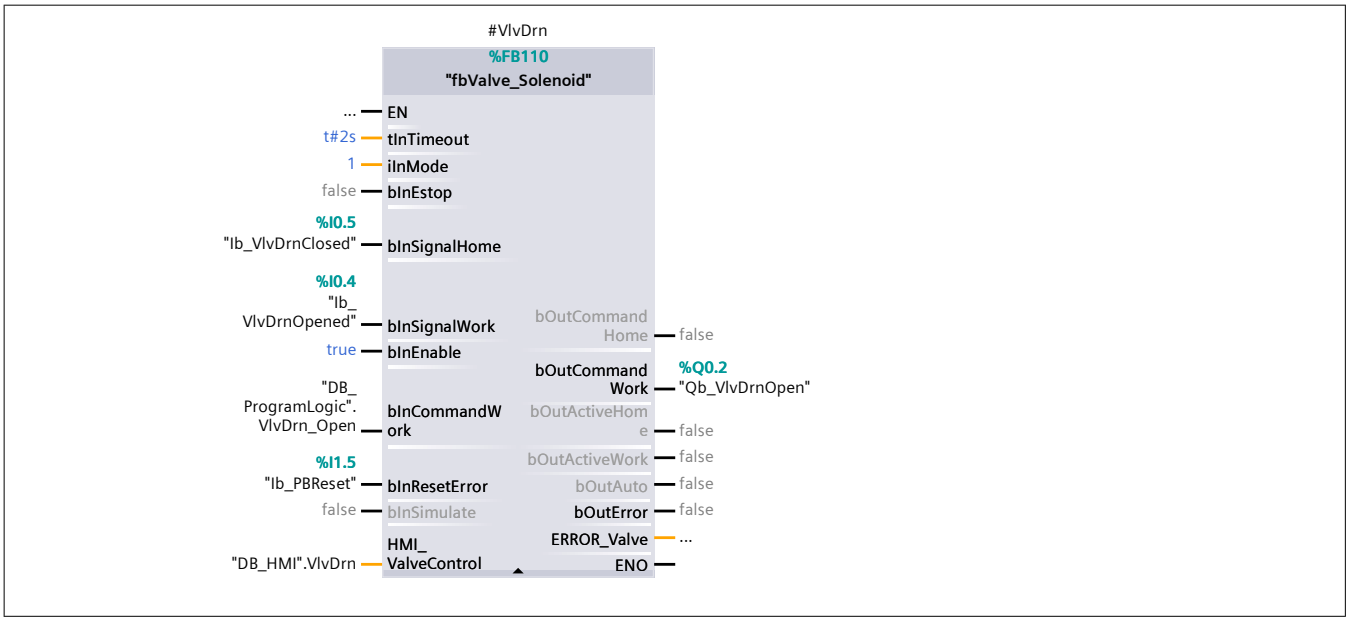
Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
iOutScrollingErrorNumber	Int	0	Set in IDB	False	False	False	False		Current Index when scrolling through errors
InOut									
▼ Static									
▼ TON_ErrorDelay	TON_TIME		Non-retain	False	False	False	False		Delay between each scroll for a new error
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
▼ abErrors	Array[1..30] of Bool		Set in IDB	False	False	False	False		
abErrors[1]	Bool	false	Set in IDB	False	False	False	False		
abErrors[2]	Bool	false	Set in IDB	False	False	False	False		
abErrors[3]	Bool	false	Set in IDB	False	False	False	False		
abErrors[4]	Bool	false	Set in IDB	False	False	False	False		
abErrors[5]	Bool	false	Set in IDB	False	False	False	False		
abErrors[6]	Bool	false	Set in IDB	False	False	False	False		
abErrors[7]	Bool	false	Set in IDB	False	False	False	False		
abErrors[8]	Bool	false	Set in IDB	False	False	False	False		
abErrors[9]	Bool	false	Set in IDB	False	False	False	False		

Name	Data type	Default value	Retain	Acces-sible from HMI/OPC UA/Web API	Wri ta-ble fro m HM I/O PC UA/ We b API	Visible in HMI engi-neer-ing	Set-point	Super-vision	Comment
abErrors[10]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[11]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[12]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[13]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[14]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[15]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[16]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[17]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[18]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[19]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[20]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[21]	Bool	false	Set in IDB	False	Fals e	False	False		

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[22]	Bool	false	Set in IDB	False	False	False	False		
abErrors[23]	Bool	false	Set in IDB	False	False	False	False		
abErrors[24]	Bool	false	Set in IDB	False	False	False	False		
abErrors[25]	Bool	false	Set in IDB	False	False	False	False		
abErrors[26]	Bool	false	Set in IDB	False	False	False	False		
abErrors[27]	Bool	false	Set in IDB	False	False	False	False		
abErrors[28]	Bool	false	Set in IDB	False	False	False	False		
abErrors[29]	Bool	false	Set in IDB	False	False	False	False		
abErrors[30]	Bool	false	Set in IDB	False	False	False	False		
iNextScrollNum	Int	1	Non-retain	False	False	False	False		Index of next error position
iErrorsScrollNum	Int	1	Non-retain	False	False	False	False		Index of current error position
bScrolling	Bool	false	Non-retain	False	False	False	False		Indicates we are scrolling
bTON_ErrorDelay	Bool	false	Non-retain	False	False	False	False		
▼ TON_TimeOut	TON_TIME		Non-retain	False	False	False	False		Timer for Error Timeout

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
iLastMode	Int	0	Non-retain	False	False	False	False		Last mode code
bNewMode	Bool	false	Non-retain	False	False	False	False		New mode selected
bEnableHome	Bool	false	Non-retain	False	False	False	False		Home position enabled
bEnableWork	Bool	false	Non-retain	False	False	False	False		Work position enabled
bAutoMode	Bool	false	Non-retain	False	False	False	False		Auto Mode is active
bManualMode	Bool	false	Non-retain	False	False	False	False		Manual Mode is active
bReset	Bool	false	Non-retain	False	False	False	False		
bTON_TimeOut	Bool	false	Non-retain	False	False	False	False		
bPB_Home	Bool	false	Non-retain	False	False	False	False		
bPB_Work	Bool	false	Non-retain	False	False	False	False		
bPB_ResetError	Bool	false	Non-retain	False	False	False	False		
Temp									
Constant									

Network 1:


#VlvDrn
%FB110
"fbValve_Solenoid"
... EN
t#2s tInTimeout
1 iInMode
false bInEstop
%I0.5
"Ib_VlvDrnClosed" bInSignalHome
%I0.4
"Ib_VlvDrnOpened" bInSignalWork
true bInEnable
"DB_ProgramLogic".VlvDrn_Open bInCommandWork
%I1.5
"Ib_PBReset" bInResetError
false bInSimulate
"DB_HMI".VlvDrn HMI_ValveControl
bOutCommandHome false
bOutCommandWork %Q0.2 "Qb_VlvDrnOpen"
bOutActiveHome false
bOutActiveWork false
bOutAuto false
bOutError false
ERROR_Valve ...
ENO

Valve Liquid 1 [FB2]

Valve Liquid 1 Properties					
General					
Name	Valve Liquid 1	**Number**	2	**Type**	FB
Language	FBD	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
▼ VlvLiq1	"fbValve_Solenoid"			True	True	True	False		
▼ Input									
tInTimeout	Time	T#0ms	Non-retain	False	False	False	False		Timeout time for an actuator's feedback to activate before giving a fault
iInMode	Int	0	Non-retain	False	False	False	False		Mode Selection
bInEstop	Bool	false	Non-retain	False	False	False	False		Estop
bInSignalHome	Bool	false	Non-retain	False	False	False	False		Home position feedback
bInSignalWork	Bool	false	Non-retain	False	False	False	False		Work position feedback
bInEnable	Bool	false	Non-retain	False	False	False	False		Home and Work Enabled
bInCommandWork	Bool	false	Non-retain	False	False	False	False		Move to work position in automatic mode
bInResetError	Bool	false	Non-retain	False	False	False	False		Reset Error
bInSimulate	Bool	false	Non-retain	False	False	False	False		Activate device simulation
▼ Output									
bOutCommandHome	Bool	false	Non-retain	False	False	False	False		Home position command
bOutCommandWork	Bool	false	Non-retain	False	False	False	False		Work position command

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bOutActiveHome	Bool	false	Non-retain	False	False	False	False		Valve is in home position
bOutActiveWork	Bool	false	Non-retain	False	False	False	False		Valve is in work position
bOutAuto	Bool	false	Non-retain	False	False	False	False		Block in auto mode
bOutError	Bool	false	Non-retain	False	False	False	False		Error exists
▼ ERROR_Valve	"udtError_Valve"		Non-retain	False	False	False	False		Valve error structure
NoHomeFeedback	Bool	false	Non-retain	False	False	False	False		Home position feedback not active
NoWorkFeedback	Bool	false	Non-retain	False	False	False	False		Work position feedback not active
HomeFeedbackStillActive	Bool	false	Non-retain	False	False	False	False		Home position feedback still active
WorkFeedbackStillActive	Bool	false	Non-retain	False	False	False	False		Work position feedback still active
▼ InOut									
HMI_ValveControl	"udtHMI_ValveControl"			False	False	False	False		HMI valve control
▼ Static									
▼ ErrorScroller	"fbErrorScroller"			False	False	False	False		Sub-block to handle error scroller
▼ Input									
bInError01	Bool	false	Set in IDB	False	False	False	False		Error 1 In Error Array
bInError02	Bool	false	Set in IDB	False	False	False	False		Error 2 In Error Array
bInError03	Bool	false	Set in IDB	False	False	False	False		Error 3 In Error Array
bInError04	Bool	false	Set in IDB	False	False	False	False		Error 4 In Error Array
bInError05	Bool	false	Set in IDB	False	False	False	False		Error 5 In Error Array
bInError06	Bool	false	Set in IDB	False	False	False	False		Error 6 In Error Array
bInError07	Bool	false	Set in IDB	False	False	False	False		Error 7 In Error Array
bInError08	Bool	false	Set in IDB	False	False	False	False		Error 8 In Error Array

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Set-point	Supervision	Comment
bInError09	Bool	false	Set in IDB	False	False	False	False		Error 9 In Error Array
bInError10	Bool	false	Set in IDB	False	False	False	False		Error 10 In Error Array
bInError11	Bool	false	Set in IDB	False	False	False	False		Error 11 In Error Array
bInError12	Bool	false	Set in IDB	False	False	False	False		Error 12 In Error Array
bInError13	Bool	false	Set in IDB	False	False	False	False		Error 13 In Error Array
bInError14	Bool	false	Set in IDB	False	False	False	False		Error 14 In Error Array
bInError15	Bool	false	Set in IDB	False	False	False	False		Error 15 In Error Array
bInError16	Bool	false	Set in IDB	False	False	False	False		Error 16 In Error Array
bInError17	Bool	false	Set in IDB	False	False	False	False		Error 17 In Error Array
bInError18	Bool	false	Set in IDB	False	False	False	False		Error 18 In Error Array
bInError19	Bool	false	Set in IDB	False	False	False	False		Error 19 In Error Array
bInError20	Bool	false	Set in IDB	False	False	False	False		Error 20 In Error Array
bInError21	Bool	false	Set in IDB	False	False	False	False		Error 21 In Error Array
bInError22	Bool	false	Set in IDB	False	False	False	False		Error 22 In Error Array
bInError23	Bool	false	Set in IDB	False	False	False	False		Error 23 In Error Array
bInError24	Bool	false	Set in IDB	False	False	False	False		Error 24 In Error Array
bInError25	Bool	false	Set in IDB	False	False	False	False		Error 25 In Error Array
bInError26	Bool	false	Set in IDB	False	False	False	False		Error 26 In Error Array
bInError27	Bool	false	Set in IDB	False	False	False	False		Error 27 In Error Array
bInError28	Bool	false	Set in IDB	False	False	False	False		Error 28 In Error Array
bInError29	Bool	false	Set in IDB	False	False	False	False		Error 29 In Error Array
bInError30	Bool	false	Set in IDB	False	False	False	False		Error 30 In Error Array
▼ Output									
bOutErrorExists	Bool	false	Set in IDB	False	False	False	False		An Error Exists

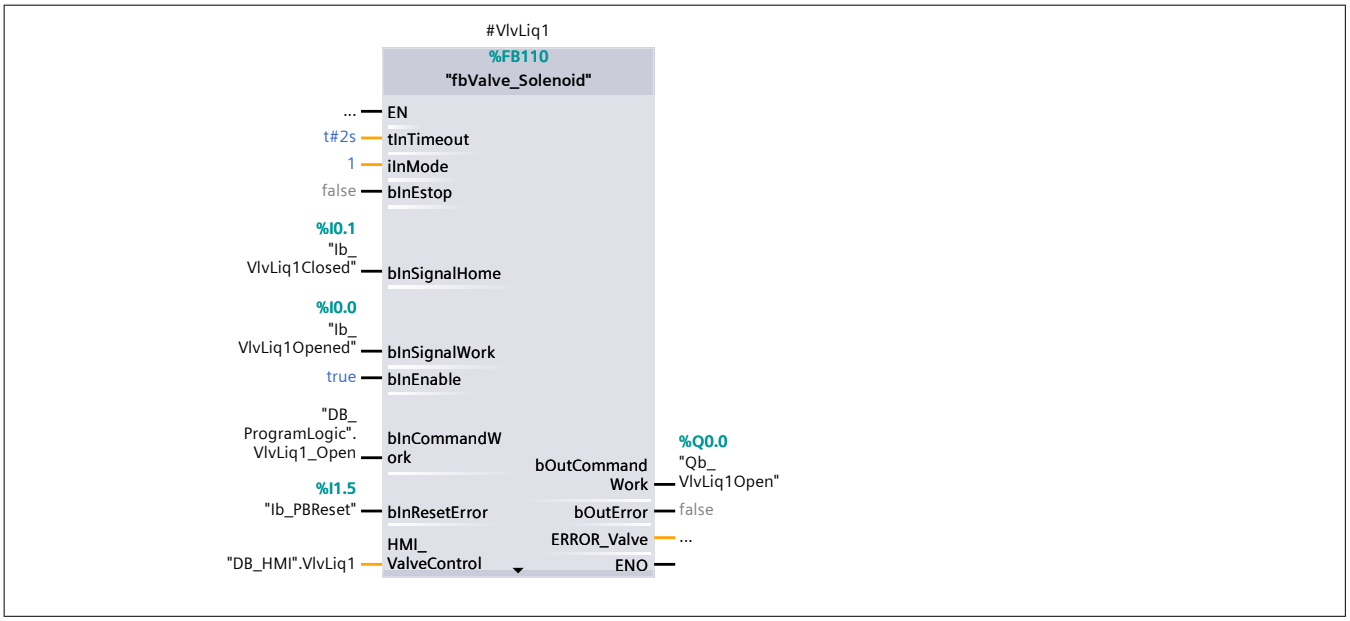
Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
iOutScrollingErrorNumber	Int	0	Set in IDB	False	False	False	False		Current Index when scrolling through errors
InOut									
▼ Static									
▼ TON_ErrorDelay	TON_TIME		Non-retain	False	False	False	False		Delay between each scroll for a new error
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
▼ abErrors	Array[1..30] of Bool		Set in IDB	False	False	False	False		
abErrors[1]	Bool	false	Set in IDB	False	False	False	False		
abErrors[2]	Bool	false	Set in IDB	False	False	False	False		
abErrors[3]	Bool	false	Set in IDB	False	False	False	False		
abErrors[4]	Bool	false	Set in IDB	False	False	False	False		
abErrors[5]	Bool	false	Set in IDB	False	False	False	False		
abErrors[6]	Bool	false	Set in IDB	False	False	False	False		
abErrors[7]	Bool	false	Set in IDB	False	False	False	False		
abErrors[8]	Bool	false	Set in IDB	False	False	False	False		
abErrors[9]	Bool	false	Set in IDB	False	False	False	False		

Name	Data type	Default value	Retain	Acces-sible from HMI/OPC UA/Web API	Wri ta-ble fro m HM I/O PC UA/ We b API	Visible in HMI engi-neer-ing	Set-point	Super-vision	Comment
abErrors[10]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[11]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[12]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[13]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[14]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[15]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[16]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[17]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[18]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[19]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[20]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[21]	Bool	false	Set in IDB	False	Fals e	False	False		

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[22]	Bool	false	Set in IDB	False	False	False	False		
abErrors[23]	Bool	false	Set in IDB	False	False	False	False		
abErrors[24]	Bool	false	Set in IDB	False	False	False	False		
abErrors[25]	Bool	false	Set in IDB	False	False	False	False		
abErrors[26]	Bool	false	Set in IDB	False	False	False	False		
abErrors[27]	Bool	false	Set in IDB	False	False	False	False		
abErrors[28]	Bool	false	Set in IDB	False	False	False	False		
abErrors[29]	Bool	false	Set in IDB	False	False	False	False		
abErrors[30]	Bool	false	Set in IDB	False	False	False	False		
iNextScrollNum	Int	1	Non-retain	False	False	False	False		Index of next error position
iErrorsScrollNum	Int	1	Non-retain	False	False	False	False		Index of current error position
bScrolling	Bool	false	Non-retain	False	False	False	False		Indicates we are scrolling
bTON_ErrorDelay	Bool	false	Non-retain	False	False	False	False		
▼ TON_TimeOut	TON_TIME		Non-retain	False	False	False	False		Timer for Error Timeout

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
iLastMode	Int	0	Non-retain	False	False	False	False		Last mode code
bNewMode	Bool	false	Non-retain	False	False	False	False		New mode selected
bEnableHome	Bool	false	Non-retain	False	False	False	False		Home position enabled
bEnableWork	Bool	false	Non-retain	False	False	False	False		Work position enabled
bAutoMode	Bool	false	Non-retain	False	False	False	False		Auto Mode is active
bManualMode	Bool	false	Non-retain	False	False	False	False		Manual Mode is active
bReset	Bool	false	Non-retain	False	False	False	False		
bTON_TimeOut	Bool	false	Non-retain	False	False	False	False		
bPB_Home	Bool	false	Non-retain	False	False	False	False		
bPB_Work	Bool	false	Non-retain	False	False	False	False		
bPB_ResetError	Bool	false	Non-retain	False	False	False	False		
Temp									
Constant									

Network 1:


#VlvLiq1
%FB110
"fbValve_Solenoid"
...
EN
t#2s
tInTimeout
1
iInMode
false
bInEstop
%I0.1
"Ib_
VlvLiq1Closed"
bInSignalHome
%I0.0
"Ib_
VlvLiq1Opened"
bInSignalWork
true
bInEnable
"DB_
ProgramLogic".
VlvLiq1_Open
bInCommandW
ork
bOutCommand
Work
%Q0.0
"Qb_
VlvLiq1Open"
%I1.5
"Ib_PBReset"
bInResetError
bOutError
false
ERROR_Valve
...
"DB_HMI".VlvLiq1
HMI_
ValveControl
ENO

Valve Liquid 2 [FB3]

Valve Liquid 2 Properties					
General					
Name	Valve Liquid 2	**Number**	3	**Type**	FB
Language	FBD	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
▼ VlvLiq2	"fbValve_Solenoid"			True	True	True	False		
▼ Input									
tInTimeout	Time	T#0ms	Non-retain	False	False	False	False		Timeout time for an actuator's feedback to activate before giving a fault
iInMode	Int	0	Non-retain	False	False	False	False		Mode Selection
bInEstop	Bool	false	Non-retain	False	False	False	False		Estop
bInSignalHome	Bool	false	Non-retain	False	False	False	False		Home position feedback
bInSignalWork	Bool	false	Non-retain	False	False	False	False		Work position feedback
bInEnable	Bool	false	Non-retain	False	False	False	False		Home and Work Enabled
bInCommandWork	Bool	false	Non-retain	False	False	False	False		Move to work position in automatic mode
bInResetError	Bool	false	Non-retain	False	False	False	False		Reset Error
bInSimulate	Bool	false	Non-retain	False	False	False	False		Activate device simulation
▼ Output									
bOutCommandHome	Bool	false	Non-retain	False	False	False	False		Home position command
bOutCommandWork	Bool	false	Non-retain	False	False	False	False		Work position command

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bOutActiveHome	Bool	false	Non-retain	False	False	False	False		Valve is in home position
bOutActiveWork	Bool	false	Non-retain	False	False	False	False		Valve is in work position
bOutAuto	Bool	false	Non-retain	False	False	False	False		Block in auto mode
bOutError	Bool	false	Non-retain	False	False	False	False		Error exists
▼ ERROR_Valve	"udtError_Valve"		Non-retain	False	False	False	False		Valve error structure
NoHomeFeedback	Bool	false	Non-retain	False	False	False	False		Home position feedback not active
NoWorkFeedback	Bool	false	Non-retain	False	False	False	False		Work position feedback not active
HomeFeedbackStillActive	Bool	false	Non-retain	False	False	False	False		Home position feedback still active
WorkFeedbackStillActive	Bool	false	Non-retain	False	False	False	False		Work position feedback still active
▼ InOut									
HMI_ValveControl	"udtHMI_ValveControl"			False	False	False	False		HMI valve control
▼ Static									
▼ ErrorScroller	"fbErrorScroller"			False	False	False	False		Sub-block to handle error scroller
▼ Input									
bInError01	Bool	false	Set in IDB	False	False	False	False		Error 1 In Error Array
bInError02	Bool	false	Set in IDB	False	False	False	False		Error 2 In Error Array
bInError03	Bool	false	Set in IDB	False	False	False	False		Error 3 In Error Array
bInError04	Bool	false	Set in IDB	False	False	False	False		Error 4 In Error Array
bInError05	Bool	false	Set in IDB	False	False	False	False		Error 5 In Error Array
bInError06	Bool	false	Set in IDB	False	False	False	False		Error 6 In Error Array
bInError07	Bool	false	Set in IDB	False	False	False	False		Error 7 In Error Array
bInError08	Bool	false	Set in IDB	False	False	False	False		Error 8 In Error Array

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Set-point	Supervision	Comment
bInError09	Bool	false	Set in IDB	False	False	False	False		Error 9 In Error Array
bInError10	Bool	false	Set in IDB	False	False	False	False		Error 10 In Error Array
bInError11	Bool	false	Set in IDB	False	False	False	False		Error 11 In Error Array
bInError12	Bool	false	Set in IDB	False	False	False	False		Error 12 In Error Array
bInError13	Bool	false	Set in IDB	False	False	False	False		Error 13 In Error Array
bInError14	Bool	false	Set in IDB	False	False	False	False		Error 14 In Error Array
bInError15	Bool	false	Set in IDB	False	False	False	False		Error 15 In Error Array
bInError16	Bool	false	Set in IDB	False	False	False	False		Error 16 In Error Array
bInError17	Bool	false	Set in IDB	False	False	False	False		Error 17 In Error Array
bInError18	Bool	false	Set in IDB	False	False	False	False		Error 18 In Error Array
bInError19	Bool	false	Set in IDB	False	False	False	False		Error 19 In Error Array
bInError20	Bool	false	Set in IDB	False	False	False	False		Error 20 In Error Array
bInError21	Bool	false	Set in IDB	False	False	False	False		Error 21 In Error Array
bInError22	Bool	false	Set in IDB	False	False	False	False		Error 22 In Error Array
bInError23	Bool	false	Set in IDB	False	False	False	False		Error 23 In Error Array
bInError24	Bool	false	Set in IDB	False	False	False	False		Error 24 In Error Array
bInError25	Bool	false	Set in IDB	False	False	False	False		Error 25 In Error Array
bInError26	Bool	false	Set in IDB	False	False	False	False		Error 26 In Error Array
bInError27	Bool	false	Set in IDB	False	False	False	False		Error 27 In Error Array
bInError28	Bool	false	Set in IDB	False	False	False	False		Error 28 In Error Array
bInError29	Bool	false	Set in IDB	False	False	False	False		Error 29 In Error Array
bInError30	Bool	false	Set in IDB	False	False	False	False		Error 30 In Error Array
▼ Output									
bOutErrorExists	Bool	false	Set in IDB	False	False	False	False		An Error Exists

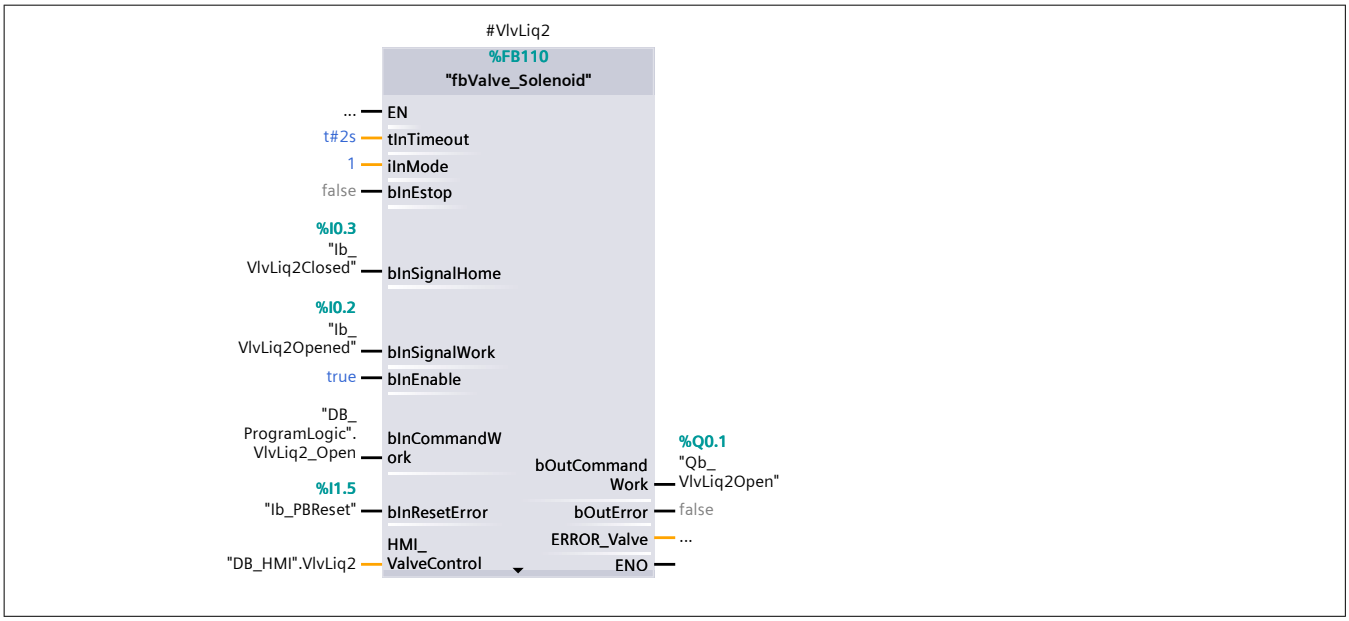
Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
iOutScrollingErrorNumber	Int	0	Set in IDB	False	False	False	False		Current Index when scrolling through errors
InOut									
▼ Static									
▼ TON_ErrorDelay	TON_TIME		Non-retain	False	False	False	False		Delay between each scroll for a new error
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
▼ abErrors	Array[1..30] of Bool		Set in IDB	False	False	False	False		
abErrors[1]	Bool	false	Set in IDB	False	False	False	False		
abErrors[2]	Bool	false	Set in IDB	False	False	False	False		
abErrors[3]	Bool	false	Set in IDB	False	False	False	False		
abErrors[4]	Bool	false	Set in IDB	False	False	False	False		
abErrors[5]	Bool	false	Set in IDB	False	False	False	False		
abErrors[6]	Bool	false	Set in IDB	False	False	False	False		
abErrors[7]	Bool	false	Set in IDB	False	False	False	False		
abErrors[8]	Bool	false	Set in IDB	False	False	False	False		
abErrors[9]	Bool	false	Set in IDB	False	False	False	False		

Name	Data type	Default value	Retain	Acces-sible from HMI/OPC UA/Web API	Wri ta-ble fro m HM I/O PC UA/ We b API	Visible in HMI engi-neer-ing	Set-point	Super-vision	Comment
abErrors[10]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[11]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[12]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[13]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[14]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[15]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[16]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[17]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[18]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[19]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[20]	Bool	false	Set in IDB	False	Fals e	False	False		
abErrors[21]	Bool	false	Set in IDB	False	Fals e	False	False		

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[22]	Bool	false	Set in IDB	False	False	False	False		
abErrors[23]	Bool	false	Set in IDB	False	False	False	False		
abErrors[24]	Bool	false	Set in IDB	False	False	False	False		
abErrors[25]	Bool	false	Set in IDB	False	False	False	False		
abErrors[26]	Bool	false	Set in IDB	False	False	False	False		
abErrors[27]	Bool	false	Set in IDB	False	False	False	False		
abErrors[28]	Bool	false	Set in IDB	False	False	False	False		
abErrors[29]	Bool	false	Set in IDB	False	False	False	False		
abErrors[30]	Bool	false	Set in IDB	False	False	False	False		
iNextScrollNum	Int	1	Non-retain	False	False	False	False		Index of next error position
iErrorsScrollNum	Int	1	Non-retain	False	False	False	False		Index of current error position
bScrolling	Bool	false	Non-retain	False	False	False	False		Indicates we are scrolling
bTON_ErrorDelay	Bool	false	Non-retain	False	False	False	False		
▼ TON_TimeOut	TON_TIME		Non-retain	False	False	False	False		Timer for Error Timeout

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
iLastMode	Int	0	Non-retain	False	False	False	False		Last mode code
bNewMode	Bool	false	Non-retain	False	False	False	False		New mode selected
bEnableHome	Bool	false	Non-retain	False	False	False	False		Home position enabled
bEnableWork	Bool	false	Non-retain	False	False	False	False		Work position enabled
bAutoMode	Bool	false	Non-retain	False	False	False	False		Auto Mode is active
bManualMode	Bool	false	Non-retain	False	False	False	False		Manual Mode is active
bReset	Bool	false	Non-retain	False	False	False	False		
bTON_TimeOut	Bool	false	Non-retain	False	False	False	False		
bPB_Home	Bool	false	Non-retain	False	False	False	False		
bPB_Work	Bool	false	Non-retain	False	False	False	False		
bPB_ResetError	Bool	false	Non-retain	False	False	False	False		
Temp									
Constant									

Network 1:


#VlvLiq2
%FB110
"fbValve_Solenoid"
...
EN
t#2s
tInTimeout
1
iInMode
false
bInEstop
%I0.3
"Ib_
VlvLiq2Closed"
bInSignalHome
%I0.2
"Ib_
VlvLiq2Opened"
bInSignalWork
true
bInEnable
"DB_
ProgramLogic".
VlvLiq2_Open
bInCommandW
ork
bOutCommand
Work
%Q0.1
"Qb_
VlvLiq2Open"
%I1.5
"Ib_PBReset"
bInResetError
bOutError
false
ERROR_Valve
...
"DB_HMI".VlvLiq2
HMI_
ValveControl
ENO

DB_HMI [DB3]

DB_HMI Properties					
General					
Name	DB_HMI	**Number**	3	**Type**	DB
Language	DB	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Static									
VlvLiq1	"udtHMI_ValveControl"		False	True	True	True	False		
iMode	Int	0	False	True	True	True	False		Current mode
iErrorCode	Int	0	False	True	False	True	False		Error code
iStatus	Int	0	False	True	False	True	False		Status for HMI display
bPB_ResetError	Bool	false	False	True	True	True	False		PB Reset block errors
bPB_Home	Bool	false	False	True	True	True	False		PB Move to home in manual mode
bPB_Work	Bool	false	False	True	True	True	False		PB Move to work in manual mode
bPBEN_ResetError	Bool	false	False	True	False	True	False		PB Reset error enabled
bPBEN_Home	Bool	false	False	True	False	True	False		PB Home enabled
bPBEN_Work	Bool	false	False	True	False	True	False		PB Work enabled
bHomeOn	Bool	false	False	True	False	True	False		Home command is on
bWorkOn	Bool	false	False	True	False	True	False		Work command is on
bSignalHome	Bool	false	False	True	False	True	False		Home feedback
bSignalWork	Bool	false	False	True	False	True	False		Work feedback
bError	Bool	false	False	True	False	True	False		Error status
bInterlock	Bool	false	False	True	False	True	False		Valve interlocked

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
VlvLiq2	"udtHMI_ValveControl"		False	True	True	True	True		
iMode	Int	0	False	True	True	True	False		Current mode
iErrorCode	Int	0	False	True	False	True	False		Error code
iStatus	Int	0	False	True	False	True	False		Status for HMI display
bPB_ResetError	Bool	false	False	True	True	True	False		PB Reset block errors
bPB_Home	Bool	false	False	True	True	True	False		PB Move to home in manual mode
bPB_Work	Bool	false	False	True	True	True	False		PB Move to work in manual mode
bPBEN_ResetError	Bool	false	False	True	False	True	False		PB Reset error enabled
bPBEN_Home	Bool	false	False	True	False	True	False		PB Home enabled
bPBEN_Work	Bool	false	False	True	False	True	False		PB Work enabled
bHomeOn	Bool	false	False	True	False	True	False		Home command is on
bWorkOn	Bool	false	False	True	False	True	False		Work command is on
bSignalHome	Bool	false	False	True	False	True	False		Home feedback
bSignalWork	Bool	false	False	True	False	True	False		Work feedback
bError	Bool	false	False	True	False	True	False		Error status
bInterlock	Bool	false	False	True	False	True	False		Valve interlocked
VlvDrn	"udtHMI_ValveControl"		False	True	True	True	True		
iMode	Int	0	False	True	True	True	False		Current mode
iErrorCode	Int	0	False	True	False	True	False		Error code
iStatus	Int	0	False	True	False	True	False		Status for HMI display
bPB_ResetError	Bool	false	False	True	True	True	False		PB Reset block errors
bPB_Home	Bool	false	False	True	True	True	False		PB Move to home in manual mode
bPB_Work	Bool	false	False	True	True	True	False		PB Move to work in manual mode
bPBEN_ResetError	Bool	false	False	True	False	True	False		PB Reset error enabled

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bPBEN_Home	Bool	false	False	True	False	True	False		PB Home enabled
bPBEN_Work	Bool	false	False	True	False	True	False		PB Work enabled
bHomeOn	Bool	false	False	True	False	True	False		Home command is on
bWorkOn	Bool	false	False	True	False	True	False		Work command is on
bSignalHome	Bool	false	False	True	False	True	False		Home feedback
bSignalWork	Bool	false	False	True	False	True	False		Work feedback
bError	Bool	false	False	True	False	True	False		Error status
bInterlock	Bool	false	False	True	False	True	False		Valve interlocked

DB_ProgramLogic [DB4]

DB_ProgramLogic Properties					
General					
Name	DB_ProgramLogic	**Number**	4	**Type**	DB
Language	DB	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Static									
VlvLiq1_Open	Bool	false	False	True	True	True	False		
VlvLiq2_Open	Bool	false	False	True	True	True	False		
VlvDrn_Open	Bool	false	False	True	True	True	False		

Program Logic_DB [DB2]

Program Logic_DB Properties					
General					
Name	Program Logic_DB	**Number**	2	**Type**	DB
Language	DB	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
▼ StepSeq	"fbStepSequencer"		False	True	True	True	False		Modified by Ken Brey 2010-08-25 KLB. Added Initialization inputs and logic. Modified by Ken Brey 2006-08-08 Removed initialization of inputs. Initializing inputs doesn't do anything. The input value if un-wired will be the last value set for that input in a different call. The initialized value in the declaration section does not set it back to default if un-wired. Allways supply values for all inputs. Modified by Tim Jager 2005-12-13 no block Id in error message Modified by Boris: branched -> simplified version, no errors, no timeout; still have to use static memory for TON/CurStep (might be fixed later using in/out structure) Modified by Nick Shea: 2012-03-30 Re-instated step mode
▼ Input									

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
iInStep	Int	0	False	False	False	False	False		Constant: current step required for the FB to run
iInNextStep	Int	0	False	False	False	False	False		Next step to go to if step is done
bInStepDone	Bool	false	False	False	False	False	False		if step done logic is true, then we move from IN_STEP to NEXT_STEP
tInStepDelay	Time	T#0ms	False	False	False	False	False		After the step done input is true, we wait for x seconds before moving to next
bInStepMode	Bool	false	False	False	False	False	False		Pause sequence at completion of current step
bInStepAdvance	Bool	false	False	False	False	False	False		If paused, trigger next step
bInit	Bool	false	False	False	False	False	False		
iInitStep	Int	0	False	False	False	False	False		
▼ Output									
bOutEnterEvent	Bool	false	False	False	False	False	False		
bOutExitEvent	Bool	false	False	False	False	False	False		
bOutPaused	Bool	false	False	False	False	False	False		Sequence is currently paused.
iOutCurrentStep	Int	0	False	False	False	False	False		output current step
InOut									
▼ Static									
▼ TON_StepDelay	TON_TIME		False	False	False	False	False		used to delay moving to the next step.
PT	Time	T#0ms	False	False	False	False	False		
ET	Time	T#0ms	False	False	False	False	False		
IN	Bool	false	False	False	False	False	False		
Q	Bool	false	False	False	False	False	False		
iCurrentStep	Int	0	False	False	False	False	False		Current step of the sequencer
bAlreadyInThisStep	Bool	false	False	False	False	False	False		
CurrentStep	Int	0	False	True	True	True	True		
NextStep	Int	0	False	True	True	True	False		

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
StepDone	Bool	false	False	True	True	True	False		

Valve Drain_DB [DB7]

Valve Drain_DB Properties					
General					
Name	Valve Drain_DB	**Number**	7	**Type**	DB
Language	DB	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
▼ VlvDrn	"fbValve_Solenoid"		False	True	True	True	False		
▼ Input									
tInTimeout	Time	T#0ms	False	False	False	False	False		Timeout time for an actuator's feedback to activate before giving a fault
iInMode	Int	0	False	False	False	False	False		Mode Selection
bInEstop	Bool	false	False	False	False	False	False		Estop
bInSignalHome	Bool	false	False	False	False	False	False		Home position feedback
bInSignalWork	Bool	false	False	False	False	False	False		Work position feedback
bInEnable	Bool	false	False	False	False	False	False		Home and Work Enabled
bInCommandWork	Bool	false	False	False	False	False	False		Move to work position in automatic mode
bInResetError	Bool	false	False	False	False	False	False		Reset Error
bInSimulate	Bool	false	False	False	False	False	False		Activate device simulation
▼ Output									
bOutCommandHome	Bool	false	False	False	False	False	False		Home position command
bOutCommandWork	Bool	false	False	False	False	False	False		Work position command
bOutActiveHome	Bool	false	False	False	False	False	False		Valve is in home position

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bOutActiveWork	Bool	false	False	False	False	False	False		Valve is in work position
bOutAuto	Bool	false	False	False	False	False	False		Block in auto mode
bOutError	Bool	false	False	False	False	False	False		Error exists
ERROR_Valve	"udtError_Valve"		False	False	False	False	False		Valve error structure
NoHomeFeedback	Bool	false	False	False	False	False	False		Home position feedback not active
NoWorkFeedback	Bool	false	False	False	False	False	False		Work position feedback not active
HomeFeedbackStillActive	Bool	false	False	False	False	False	False		Home position feedback still active
WorkFeedbackStillActive	Bool	false	False	False	False	False	False		Work position feedback still active
InOut									
HMI_ValveControl	"udtHMI_ValveControl"		False	False	False	False	False		HMI valve control
Static									
ErrorScroller	"fbErrorScroller"		False	False	False	False	False		Sub-block to handle error scroller
Input									
bInError01	Bool	false	False	False	False	False	False		Error 1 In Error Array
bInError02	Bool	false	False	False	False	False	False		Error 2 In Error Array
bInError03	Bool	false	False	False	False	False	False		Error 3 In Error Array
bInError04	Bool	false	False	False	False	False	False		Error 4 In Error Array
bInError05	Bool	false	False	False	False	False	False		Error 5 In Error Array
bInError06	Bool	false	False	False	False	False	False		Error 6 In Error Array
bInError07	Bool	false	False	False	False	False	False		Error 7 In Error Array
bInError08	Bool	false	False	False	False	False	False		Error 8 In Error Array
bInError09	Bool	false	False	False	False	False	False		Error 9 In Error Array
bInError10	Bool	false	False	False	False	False	False		Error 10 In Error Array

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bInError11	Bool	false	False	False	False	False	False		Error 11 In Error Array
bInError12	Bool	false	False	False	False	False	False		Error 12 In Error Array
bInError13	Bool	false	False	False	False	False	False		Error 13 In Error Array
bInError14	Bool	false	False	False	False	False	False		Error 14 In Error Array
bInError15	Bool	false	False	False	False	False	False		Error 15 In Error Array
bInError16	Bool	false	False	False	False	False	False		Error 16 In Error Array
bInError17	Bool	false	False	False	False	False	False		Error 17 In Error Array
bInError18	Bool	false	False	False	False	False	False		Error 18 In Error Array
bInError19	Bool	false	False	False	False	False	False		Error 19 In Error Array
bInError20	Bool	false	False	False	False	False	False		Error 20 In Error Array
bInError21	Bool	false	False	False	False	False	False		Error 21 In Error Array
bInError22	Bool	false	False	False	False	False	False		Error 22 In Error Array
bInError23	Bool	false	False	False	False	False	False		Error 23 In Error Array
bInError24	Bool	false	False	False	False	False	False		Error 24 In Error Array
bInError25	Bool	false	False	False	False	False	False		Error 25 In Error Array
bInError26	Bool	false	False	False	False	False	False		Error 26 In Error Array
bInError27	Bool	false	False	False	False	False	False		Error 27 In Error Array
bInError28	Bool	false	False	False	False	False	False		Error 28 In Error Array
bInError29	Bool	false	False	False	False	False	False		Error 29 In Error Array
bInError30	Bool	false	False	False	False	False	False		Error 30 In Error Array
▼ Output									
bOutErrorExists	Bool	false	False	False	False	False	False		An Error Exists

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
iOutScrollingErrorNumber	Int	0	False	False	False	False	False		Current Index when scrolling through errors
InOut									
▼ Static									
▼ TON_ErrorDelay	TON_TIME		False	False	False	False	False		Delay between each scroll for a new error
PT	Time	T#0ms	False	False	False	False	False		
ET	Time	T#0ms	False	False	False	False	False		
IN	Bool	false	False	False	False	False	False		
Q	Bool	false	False	False	False	False	False		
▼ abErrors	Array[1..30] of Bool		False	False	False	False	False		
abErrors[1]	Bool	false	False	False	False	False	False		
abErrors[2]	Bool	false	False	False	False	False	False		
abErrors[3]	Bool	false	False	False	False	False	False		
abErrors[4]	Bool	false	False	False	False	False	False		
abErrors[5]	Bool	false	False	False	False	False	False		
abErrors[6]	Bool	false	False	False	False	False	False		
abErrors[7]	Bool	false	False	False	False	False	False		
abErrors[8]	Bool	false	False	False	False	False	False		
abErrors[9]	Bool	false	False	False	False	False	False		

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[10]	Bool	false	False	False	False	False	False		
abErrors[11]	Bool	false	False	False	False	False	False		
abErrors[12]	Bool	false	False	False	False	False	False		
abErrors[13]	Bool	false	False	False	False	False	False		
abErrors[14]	Bool	false	False	False	False	False	False		
abErrors[15]	Bool	false	False	False	False	False	False		
abErrors[16]	Bool	false	False	False	False	False	False		
abErrors[17]	Bool	false	False	False	False	False	False		
abErrors[18]	Bool	false	False	False	False	False	False		
abErrors[19]	Bool	false	False	False	False	False	False		
abErrors[20]	Bool	false	False	False	False	False	False		
abErrors[21]	Bool	false	False	False	False	False	False		

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[22]	Bool	false	False	False	False	False	False		
abErrors[23]	Bool	false	False	False	False	False	False		
abErrors[24]	Bool	false	False	False	False	False	False		
abErrors[25]	Bool	false	False	False	False	False	False		
abErrors[26]	Bool	false	False	False	False	False	False		
abErrors[27]	Bool	false	False	False	False	False	False		
abErrors[28]	Bool	false	False	False	False	False	False		
abErrors[29]	Bool	false	False	False	False	False	False		
abErrors[30]	Bool	false	False	False	False	False	False		
iNextScrollNum	Int	1	False	False	False	False	False		Index of next error position
iErrorsScrollNum	Int	1	False	False	False	False	False		Index of current error position
bScrolling	Bool	false	False	False	False	False	False		Indicates we are scrolling
bTON_ErrorDelay	Bool	false	False	False	False	False	False		
▼ TON_TimeOut	TON_TIME		False	False	False	False	False		Timer for Error Timeout

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
PT	Time	T#0ms	False	False	False	False	False		
ET	Time	T#0ms	False	False	False	False	False		
IN	Bool	false	False	False	False	False	False		
Q	Bool	false	False	False	False	False	False		
iLastMode	Int	0	False	False	False	False	False		Last mode code
bNewMode	Bool	false	False	False	False	False	False		New mode selected
bEnable-Home	Bool	false	False	False	False	False	False		Home position enabled
bEnable-Work	Bool	false	False	False	False	False	False		Work position enabled
bAutoMode	Bool	false	False	False	False	False	False		Auto Mode is active
bManual-Mode	Bool	false	False	False	False	False	False		Manual Mode is active
bReset	Bool	false	False	False	False	False	False		
bTON_Time-Out	Bool	false	False	False	False	False	False		
bPB_Home	Bool	false	False	False	False	False	False		
bPB_Work	Bool	false	False	False	False	False	False		
bPB_Rese-tError	Bool	false	False	False	False	False	False		

Valve Liquid 1_DB [DB8]

Valve Liquid 1_DB Properties					
General					
Name	Valve Liquid 1_DB	**Number**	8	**Type**	DB
Language	DB	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
▼ VlvLiq1	"fbValve_Solenoid"		False	True	True	True	False		
▼ Input									
tInTimeout	Time	T#0ms	False	False	False	False	False		Timeout time for an actuator's feedback to activate before giving a fault
iInMode	Int	0	False	False	False	False	False		Mode Selection
bInEstop	Bool	false	False	False	False	False	False		Estop
bInSignalHome	Bool	false	False	False	False	False	False		Home position feedback
bInSignalWork	Bool	false	False	False	False	False	False		Work position feedback
bInEnable	Bool	false	False	False	False	False	False		Home and Work Enabled
bInCommandWork	Bool	false	False	False	False	False	False		Move to work position in automatic mode
bInResetError	Bool	false	False	False	False	False	False		Reset Error
bInSimulate	Bool	false	False	False	False	False	False		Activate device simulation
▼ Output									
bOutCommandHome	Bool	false	False	False	False	False	False		Home position command
bOutCommandWork	Bool	false	False	False	False	False	False		Work position command
bOutActiveHome	Bool	false	False	False	False	False	False		Valve is in home position

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bOutActiveWork	Bool	false	False	False	False	False	False		Valve is in work position
bOutAuto	Bool	false	False	False	False	False	False		Block in auto mode
bOutError	Bool	false	False	False	False	False	False		Error exists
ERROR_Valve	"udtError_Valve"		False	False	False	False	False		Valve error structure
NoHomeFeedback	Bool	false	False	False	False	False	False		Home position feedback not active
NoWorkFeedback	Bool	false	False	False	False	False	False		Work position feedback not active
HomeFeedbackStillActive	Bool	false	False	False	False	False	False		Home position feedback still active
WorkFeedbackStillActive	Bool	false	False	False	False	False	False		Work position feedback still active
InOut									
HMI_ValveControl	"udtHMI_ValveControl"		False	False	False	False	False		HMI valve control
Static									
ErrorScroller	"fbErrorScroller"		False	False	False	False	False		Sub-block to handle error scroller
Input									
bInError01	Bool	false	False	False	False	False	False		Error 1 In Error Array
bInError02	Bool	false	False	False	False	False	False		Error 2 In Error Array
bInError03	Bool	false	False	False	False	False	False		Error 3 In Error Array
bInError04	Bool	false	False	False	False	False	False		Error 4 In Error Array
bInError05	Bool	false	False	False	False	False	False		Error 5 In Error Array
bInError06	Bool	false	False	False	False	False	False		Error 6 In Error Array
bInError07	Bool	false	False	False	False	False	False		Error 7 In Error Array
bInError08	Bool	false	False	False	False	False	False		Error 8 In Error Array
bInError09	Bool	false	False	False	False	False	False		Error 9 In Error Array
bInError10	Bool	false	False	False	False	False	False		Error 10 In Error Array

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bInError11	Bool	false	False	False	False	False	False		Error 11 In Error Array
bInError12	Bool	false	False	False	False	False	False		Error 12 In Error Array
bInError13	Bool	false	False	False	False	False	False		Error 13 In Error Array
bInError14	Bool	false	False	False	False	False	False		Error 14 In Error Array
bInError15	Bool	false	False	False	False	False	False		Error 15 In Error Array
bInError16	Bool	false	False	False	False	False	False		Error 16 In Error Array
bInError17	Bool	false	False	False	False	False	False		Error 17 In Error Array
bInError18	Bool	false	False	False	False	False	False		Error 18 In Error Array
bInError19	Bool	false	False	False	False	False	False		Error 19 In Error Array
bInError20	Bool	false	False	False	False	False	False		Error 20 In Error Array
bInError21	Bool	false	False	False	False	False	False		Error 21 In Error Array
bInError22	Bool	false	False	False	False	False	False		Error 22 In Error Array
bInError23	Bool	false	False	False	False	False	False		Error 23 In Error Array
bInError24	Bool	false	False	False	False	False	False		Error 24 In Error Array
bInError25	Bool	false	False	False	False	False	False		Error 25 In Error Array
bInError26	Bool	false	False	False	False	False	False		Error 26 In Error Array
bInError27	Bool	false	False	False	False	False	False		Error 27 In Error Array
bInError28	Bool	false	False	False	False	False	False		Error 28 In Error Array
bInError29	Bool	false	False	False	False	False	False		Error 29 In Error Array
bInError30	Bool	false	False	False	False	False	False		Error 30 In Error Array
▼ Output									
bOutErrorExists	Bool	false	False	False	False	False	False		An Error Exists

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
iOutScrollingErrorNumber	Int	0	False	False	False	False	False		Current Index when scrolling through errors
InOut									
▼ Static									
▼ TON_ErrorDelay	TON_TIME		False	False	False	False	False		Delay between each scroll for a new error
PT	Time	T#0ms	False	False	False	False	False		
ET	Time	T#0ms	False	False	False	False	False		
IN	Bool	false	False	False	False	False	False		
Q	Bool	false	False	False	False	False	False		
▼ abErrors	Array[1..30] of Bool		False	False	False	False	False		
abErrors[1]	Bool	false	False	False	False	False	False		
abErrors[2]	Bool	false	False	False	False	False	False		
abErrors[3]	Bool	false	False	False	False	False	False		
abErrors[4]	Bool	false	False	False	False	False	False		
abErrors[5]	Bool	false	False	False	False	False	False		
abErrors[6]	Bool	false	False	False	False	False	False		
abErrors[7]	Bool	false	False	False	False	False	False		
abErrors[8]	Bool	false	False	False	False	False	False		
abErrors[9]	Bool	false	False	False	False	False	False		

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[10]	Bool	false	False	False	False	False	False		
abErrors[11]	Bool	false	False	False	False	False	False		
abErrors[12]	Bool	false	False	False	False	False	False		
abErrors[13]	Bool	false	False	False	False	False	False		
abErrors[14]	Bool	false	False	False	False	False	False		
abErrors[15]	Bool	false	False	False	False	False	False		
abErrors[16]	Bool	false	False	False	False	False	False		
abErrors[17]	Bool	false	False	False	False	False	False		
abErrors[18]	Bool	false	False	False	False	False	False		
abErrors[19]	Bool	false	False	False	False	False	False		
abErrors[20]	Bool	false	False	False	False	False	False		
abErrors[21]	Bool	false	False	False	False	False	False		

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[22]	Bool	false	False	False	False	False	False		
abErrors[23]	Bool	false	False	False	False	False	False		
abErrors[24]	Bool	false	False	False	False	False	False		
abErrors[25]	Bool	false	False	False	False	False	False		
abErrors[26]	Bool	false	False	False	False	False	False		
abErrors[27]	Bool	false	False	False	False	False	False		
abErrors[28]	Bool	false	False	False	False	False	False		
abErrors[29]	Bool	false	False	False	False	False	False		
abErrors[30]	Bool	false	False	False	False	False	False		
iNextScrollNum	Int	1	False	False	False	False	False		Index of next error position
iErrorsScrollNum	Int	1	False	False	False	False	False		Index of current error position
bScrolling	Bool	false	False	False	False	False	False		Indicates we are scrolling
bTON_ErrorDelay	Bool	false	False	False	False	False	False		
▼ TON_TimeOut	TON_TIME		False	False	False	False	False		Timer for Error Timeout

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
PT	Time	T#0ms	False	False	False	False	False		
ET	Time	T#0ms	False	False	False	False	False		
IN	Bool	false	False	False	False	False	False		
Q	Bool	false	False	False	False	False	False		
iLastMode	Int	0	False	False	False	False	False		Last mode code
bNewMode	Bool	false	False	False	False	False	False		New mode selected
bEnable-Home	Bool	false	False	False	False	False	False		Home position enabled
bEnable-Work	Bool	false	False	False	False	False	False		Work position enabled
bAutoMode	Bool	false	False	False	False	False	False		Auto Mode is active
bManual-Mode	Bool	false	False	False	False	False	False		Manual Mode is active
bReset	Bool	false	False	False	False	False	False		
bTON_Time-Out	Bool	false	False	False	False	False	False		
bPB_Home	Bool	false	False	False	False	False	False		
bPB_Work	Bool	false	False	False	False	False	False		
bPB_Rese-tError	Bool	false	False	False	False	False	False		

Valve Liquid 2_DB [DB9]

Valve Liquid 2_DB Properties					
General					
Name	Valve Liquid 2_DB	**Number**	9	**Type**	DB
Language	DB	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
▼ VlvLiq2	"fbValve_Solenoid"		False	True	True	True	False		
▼ Input									
tInTimeout	Time	T#0ms	False	False	False	False	False		Timeout time for an actuator's feedback to activate before giving a fault
iInMode	Int	0	False	False	False	False	False		Mode Selection
bInEstop	Bool	false	False	False	False	False	False		Estop
bInSignalHome	Bool	false	False	False	False	False	False		Home position feedback
bInSignalWork	Bool	false	False	False	False	False	False		Work position feedback
bInEnable	Bool	false	False	False	False	False	False		Home and Work Enabled
bInCommandWork	Bool	false	False	False	False	False	False		Move to work position in automatic mode
bInResetError	Bool	false	False	False	False	False	False		Reset Error
bInSimulate	Bool	false	False	False	False	False	False		Activate device simulation
▼ Output									
bOutCommandHome	Bool	false	False	False	False	False	False		Home position command
bOutCommandWork	Bool	false	False	False	False	False	False		Work position command
bOutActiveHome	Bool	false	False	False	False	False	False		Valve is in home position

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bOutActiveWork	Bool	false	False	False	False	False	False		Valve is in work position
bOutAuto	Bool	false	False	False	False	False	False		Block in auto mode
bOutError	Bool	false	False	False	False	False	False		Error exists
ERROR_Valve	"udtError_Valve"		False	False	False	False	False		Valve error structure
NoHomeFeedback	Bool	false	False	False	False	False	False		Home position feedback not active
NoWorkFeedback	Bool	false	False	False	False	False	False		Work position feedback not active
HomeFeedbackStillActive	Bool	false	False	False	False	False	False		Home position feedback still active
WorkFeedbackStillActive	Bool	false	False	False	False	False	False		Work position feedback still active
InOut									
HMI_ValveControl	"udtHMI_ValveControl"		False	False	False	False	False		HMI valve control
Static									
ErrorScroller	"fbErrorScroller"		False	False	False	False	False		Sub-block to handle error scroller
Input									
bInError01	Bool	false	False	False	False	False	False		Error 1 In Error Array
bInError02	Bool	false	False	False	False	False	False		Error 2 In Error Array
bInError03	Bool	false	False	False	False	False	False		Error 3 In Error Array
bInError04	Bool	false	False	False	False	False	False		Error 4 In Error Array
bInError05	Bool	false	False	False	False	False	False		Error 5 In Error Array
bInError06	Bool	false	False	False	False	False	False		Error 6 In Error Array
bInError07	Bool	false	False	False	False	False	False		Error 7 In Error Array
bInError08	Bool	false	False	False	False	False	False		Error 8 In Error Array
bInError09	Bool	false	False	False	False	False	False		Error 9 In Error Array
bInError10	Bool	false	False	False	False	False	False		Error 10 In Error Array

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bInError11	Bool	false	False	False	False	False	False		Error 11 In Error Array
bInError12	Bool	false	False	False	False	False	False		Error 12 In Error Array
bInError13	Bool	false	False	False	False	False	False		Error 13 In Error Array
bInError14	Bool	false	False	False	False	False	False		Error 14 In Error Array
bInError15	Bool	false	False	False	False	False	False		Error 15 In Error Array
bInError16	Bool	false	False	False	False	False	False		Error 16 In Error Array
bInError17	Bool	false	False	False	False	False	False		Error 17 In Error Array
bInError18	Bool	false	False	False	False	False	False		Error 18 In Error Array
bInError19	Bool	false	False	False	False	False	False		Error 19 In Error Array
bInError20	Bool	false	False	False	False	False	False		Error 20 In Error Array
bInError21	Bool	false	False	False	False	False	False		Error 21 In Error Array
bInError22	Bool	false	False	False	False	False	False		Error 22 In Error Array
bInError23	Bool	false	False	False	False	False	False		Error 23 In Error Array
bInError24	Bool	false	False	False	False	False	False		Error 24 In Error Array
bInError25	Bool	false	False	False	False	False	False		Error 25 In Error Array
bInError26	Bool	false	False	False	False	False	False		Error 26 In Error Array
bInError27	Bool	false	False	False	False	False	False		Error 27 In Error Array
bInError28	Bool	false	False	False	False	False	False		Error 28 In Error Array
bInError29	Bool	false	False	False	False	False	False		Error 29 In Error Array
bInError30	Bool	false	False	False	False	False	False		Error 30 In Error Array
▼ Output									
bOutErrorExists	Bool	false	False	False	False	False	False		An Error Exists

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
iOutScrollingErrorNumber	Int	0	False	False	False	False	False		Current Index when scrolling through errors
InOut									
▼ Static									
▼ TON_ErrorDelay	TON_TIME		False	False	False	False	False		Delay between each scroll for a new error
PT	Time	T#0ms	False	False	False	False	False		
ET	Time	T#0ms	False	False	False	False	False		
IN	Bool	false	False	False	False	False	False		
Q	Bool	false	False	False	False	False	False		
▼ abErrors	Array[1..30] of Bool		False	False	False	False	False		
abErrors[1]	Bool	false	False	False	False	False	False		
abErrors[2]	Bool	false	False	False	False	False	False		
abErrors[3]	Bool	false	False	False	False	False	False		
abErrors[4]	Bool	false	False	False	False	False	False		
abErrors[5]	Bool	false	False	False	False	False	False		
abErrors[6]	Bool	false	False	False	False	False	False		
abErrors[7]	Bool	false	False	False	False	False	False		
abErrors[8]	Bool	false	False	False	False	False	False		
abErrors[9]	Bool	false	False	False	False	False	False		

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[10]	Bool	false	False	False	False	False	False		
abErrors[11]	Bool	false	False	False	False	False	False		
abErrors[12]	Bool	false	False	False	False	False	False		
abErrors[13]	Bool	false	False	False	False	False	False		
abErrors[14]	Bool	false	False	False	False	False	False		
abErrors[15]	Bool	false	False	False	False	False	False		
abErrors[16]	Bool	false	False	False	False	False	False		
abErrors[17]	Bool	false	False	False	False	False	False		
abErrors[18]	Bool	false	False	False	False	False	False		
abErrors[19]	Bool	false	False	False	False	False	False		
abErrors[20]	Bool	false	False	False	False	False	False		
abErrors[21]	Bool	false	False	False	False	False	False		

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[22]	Bool	false	False	False	False	False	False		
abErrors[23]	Bool	false	False	False	False	False	False		
abErrors[24]	Bool	false	False	False	False	False	False		
abErrors[25]	Bool	false	False	False	False	False	False		
abErrors[26]	Bool	false	False	False	False	False	False		
abErrors[27]	Bool	false	False	False	False	False	False		
abErrors[28]	Bool	false	False	False	False	False	False		
abErrors[29]	Bool	false	False	False	False	False	False		
abErrors[30]	Bool	false	False	False	False	False	False		
iNextScrollNum	Int	1	False	False	False	False	False		Index of next error position
iErrorsScrollNum	Int	1	False	False	False	False	False		Index of current error position
bScrolling	Bool	false	False	False	False	False	False		Indicates we are scrolling
bTON_ErrorDelay	Bool	false	False	False	False	False	False		
▼ TON_TimeOut	TON_TIME		False	False	False	False	False		Timer for Error Timeout

Name	Data type	Start value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
PT	Time	T#0ms	False	False	False	False	False		
ET	Time	T#0ms	False	False	False	False	False		
IN	Bool	false	False	False	False	False	False		
Q	Bool	false	False	False	False	False	False		
iLastMode	Int	0	False	False	False	False	False		Last mode code
bNewMode	Bool	false	False	False	False	False	False		New mode selected
bEnable-Home	Bool	false	False	False	False	False	False		Home position enabled
bEnable-Work	Bool	false	False	False	False	False	False		Work position enabled
bAutoMode	Bool	false	False	False	False	False	False		Auto Mode is active
bManual-Mode	Bool	false	False	False	False	False	False		Manual Mode is active
bReset	Bool	false	False	False	False	False	False		
bTON_Time-Out	Bool	false	False	False	False	False	False		
bPB_Home	Bool	false	False	False	False	False	False		
bPB_Work	Bool	false	False	False	False	False	False		
bPB_Rese-tError	Bool	false	False	False	False	False	False		

Open Library V15 / Resources / HMI

fbErrorScroller [FB330] [fbErrorScroller V 3.0.3]

fbErrorScroller Properties					
General					
Name	fbErrorScroller	**Number**	330	**Type**	FB
Language	FBD	**Numbering**	Manual		
Information					
Title	Scrolls through the error codes for display on the HMI	**Author**	DMC	**Comment**	
Family	Error	**Version**	0.1	**User-defined ID**	

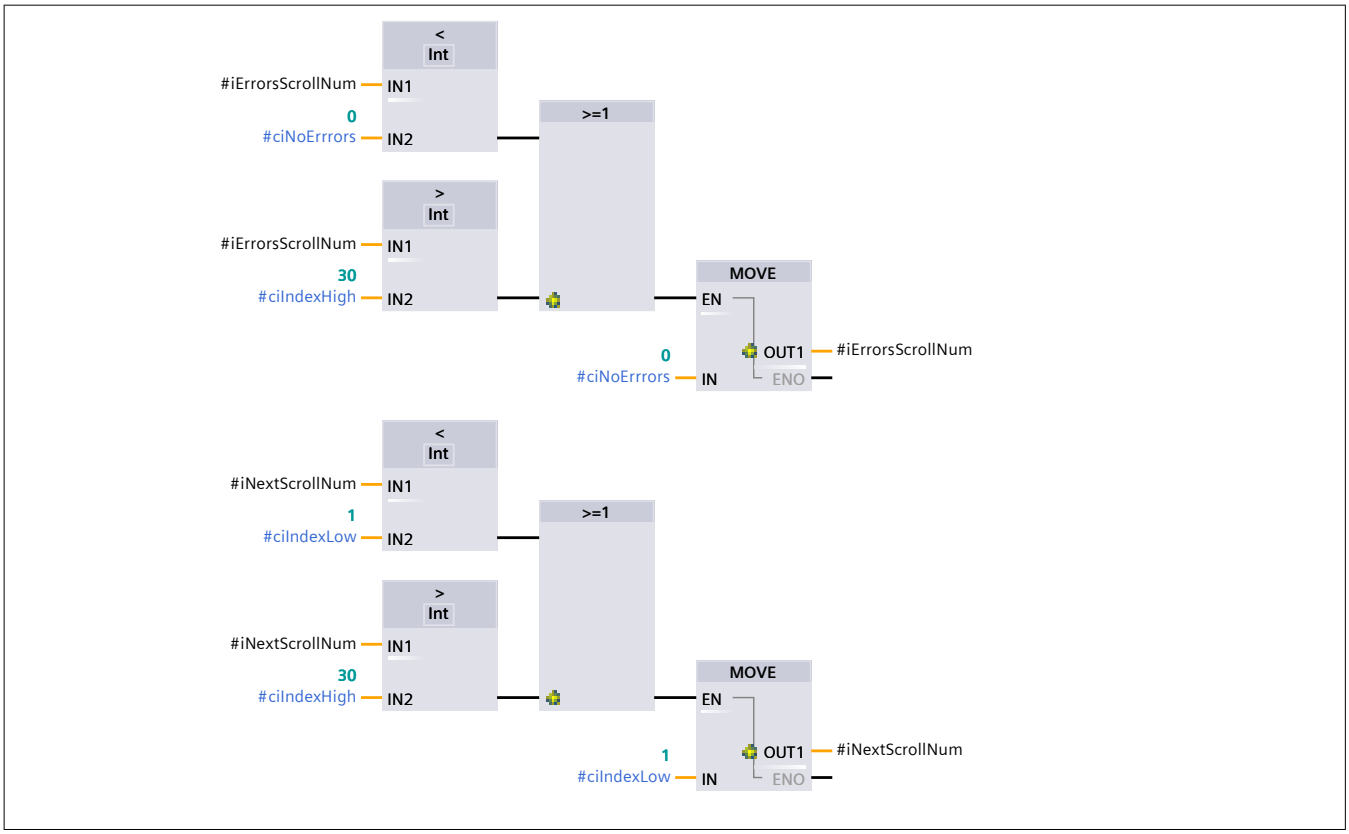
Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Set-point	Supervision	Comment
▼ Input									
bInError01	Bool	false	Set in IDB	False	False	False	False		Error 1 In Error Array
bInError02	Bool	false	Set in IDB	False	False	False	False		Error 2 In Error Array
bInError03	Bool	false	Set in IDB	False	False	False	False		Error 3 In Error Array
bInError04	Bool	false	Set in IDB	False	False	False	False		Error 4 In Error Array
bInError05	Bool	false	Set in IDB	False	False	False	False		Error 5 In Error Array
bInError06	Bool	false	Set in IDB	False	False	False	False		Error 6 In Error Array
bInError07	Bool	false	Set in IDB	False	False	False	False		Error 7 In Error Array
bInError08	Bool	false	Set in IDB	False	False	False	False		Error 8 In Error Array
bInError09	Bool	false	Set in IDB	False	False	False	False		Error 9 In Error Array
bInError10	Bool	false	Set in IDB	False	False	False	False		Error 10 In Error Array
bInError11	Bool	false	Set in IDB	False	False	False	False		Error 11 In Error Array
bInError12	Bool	false	Set in IDB	False	False	False	False		Error 12 In Error Array
bInError13	Bool	false	Set in IDB	False	False	False	False		Error 13 In Error Array
bInError14	Bool	false	Set in IDB	False	False	False	False		Error 14 In Error Array
bInError15	Bool	false	Set in IDB	False	False	False	False		Error 15 In Error Array
bInError16	Bool	false	Set in IDB	False	False	False	False		Error 16 In Error Array

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bInError17	Bool	false	Set in IDB	False	False	False	False		Error 17 In Error Array
bInError18	Bool	false	Set in IDB	False	False	False	False		Error 18 In Error Array
bInError19	Bool	false	Set in IDB	False	False	False	False		Error 19 In Error Array
bInError20	Bool	false	Set in IDB	False	False	False	False		Error 20 In Error Array
bInError21	Bool	false	Set in IDB	False	False	False	False		Error 21 In Error Array
bInError22	Bool	false	Set in IDB	False	False	False	False		Error 22 In Error Array
bInError23	Bool	false	Set in IDB	False	False	False	False		Error 23 In Error Array
bInError24	Bool	false	Set in IDB	False	False	False	False		Error 24 In Error Array
bInError25	Bool	false	Set in IDB	False	False	False	False		Error 25 In Error Array
bInError26	Bool	false	Set in IDB	False	False	False	False		Error 26 In Error Array
bInError27	Bool	false	Set in IDB	False	False	False	False		Error 27 In Error Array
bInError28	Bool	false	Set in IDB	False	False	False	False		Error 28 In Error Array
bInError29	Bool	false	Set in IDB	False	False	False	False		Error 29 In Error Array
bInError30	Bool	false	Set in IDB	False	False	False	False		Error 30 In Error Array
▼ Output									
bOutErrorExists	Bool	false	Set in IDB	False	False	False	False		An Error Exists
iOutScrollingErrorNumber	Int	0	Set in IDB	False	False	False	False		Current Index when scrolling through errors
InOut									
▼ Static									
▼ TON_ErrorDelay	TON_TIME		Non-retain	False	False	False	True		Delay between each scroll for a new error
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors	Array[#ciIndexLow..#ciIndexHigh] of Bool		Set in IDB	False	False	False	False		
abErrors[1]	Bool	false	Set in IDB	False	False	False	False		
abErrors[2]	Bool	false	Set in IDB	False	False	False	False		
abErrors[3]	Bool	false	Set in IDB	False	False	False	False		
abErrors[4]	Bool	false	Set in IDB	False	False	False	False		
abErrors[5]	Bool	false	Set in IDB	False	False	False	False		
abErrors[6]	Bool	false	Set in IDB	False	False	False	False		
abErrors[7]	Bool	false	Set in IDB	False	False	False	False		
abErrors[8]	Bool	false	Set in IDB	False	False	False	False		
abErrors[9]	Bool	false	Set in IDB	False	False	False	False		
abErrors[10]	Bool	false	Set in IDB	False	False	False	False		
abErrors[11]	Bool	false	Set in IDB	False	False	False	False		
abErrors[12]	Bool	false	Set in IDB	False	False	False	False		
abErrors[13]	Bool	false	Set in IDB	False	False	False	False		
abErrors[14]	Bool	false	Set in IDB	False	False	False	False		
abErrors[15]	Bool	false	Set in IDB	False	False	False	False		
abErrors[16]	Bool	false	Set in IDB	False	False	False	False		
abErrors[17]	Bool	false	Set in IDB	False	False	False	False		
abErrors[18]	Bool	false	Set in IDB	False	False	False	False		
abErrors[19]	Bool	false	Set in IDB	False	False	False	False		
abErrors[20]	Bool	false	Set in IDB	False	False	False	False		
abErrors[21]	Bool	false	Set in IDB	False	False	False	False		
abErrors[22]	Bool	false	Set in IDB	False	False	False	False		

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[23]	Bool	false	Set in IDB	False	False	False	False		
abErrors[24]	Bool	false	Set in IDB	False	False	False	False		
abErrors[25]	Bool	false	Set in IDB	False	False	False	False		
abErrors[26]	Bool	false	Set in IDB	False	False	False	False		
abErrors[27]	Bool	false	Set in IDB	False	False	False	False		
abErrors[28]	Bool	false	Set in IDB	False	False	False	False		
abErrors[29]	Bool	false	Set in IDB	False	False	False	False		
abErrors[30]	Bool	false	Set in IDB	False	False	False	False		
iNextScrollNum	Int	1	Non-retain	False	False	False	False		Index of next error position
iErrorsScrollNum	Int	1	Non-retain	False	False	False	False		Index of current error position
bScrolling	Bool	false	Non-retain	False	False	False	False		Indicates we are scrolling
bTON_ErrorDelay	Bool	false	Non-retain	False	False	False	False		
▼ Temp									
bErrorExists	Bool								Error Exists
▼ Constant									
ciNoErrrors	Int	0							
ciIndexLow	Int	1							
ciIndexHigh	Int	30							

Network 1: Check for valid array indexes



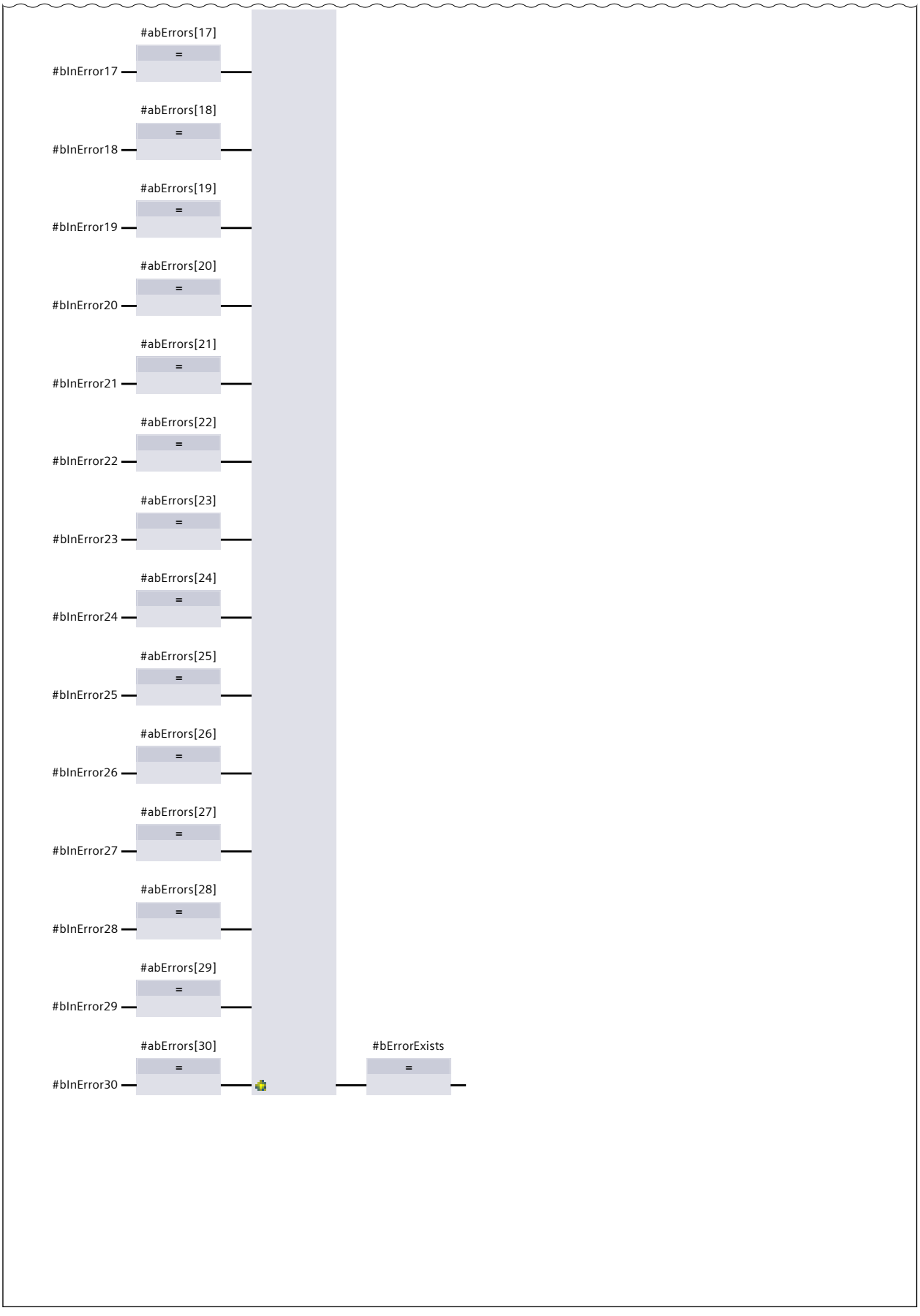
Network 2: Check if error exists

OR all of the possible errors. If any error Exists, SET bErrorExists

Network 2: Check if error exists (1.1 / 2.1)

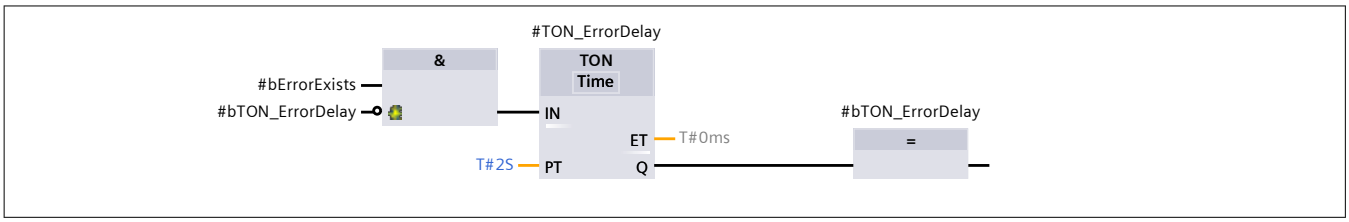
Network 2: Check if error exists (2.1 / 2.1)

1.1 (Page20 - 6)



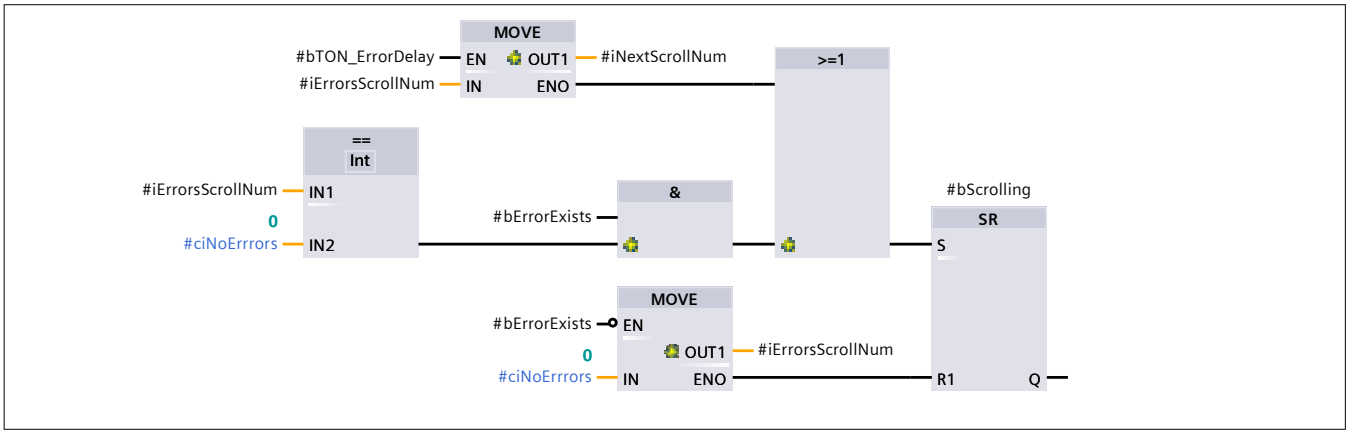
Network 3: Timer to change displayed error

========== Scroll errors on HMI ==========
Timed delay



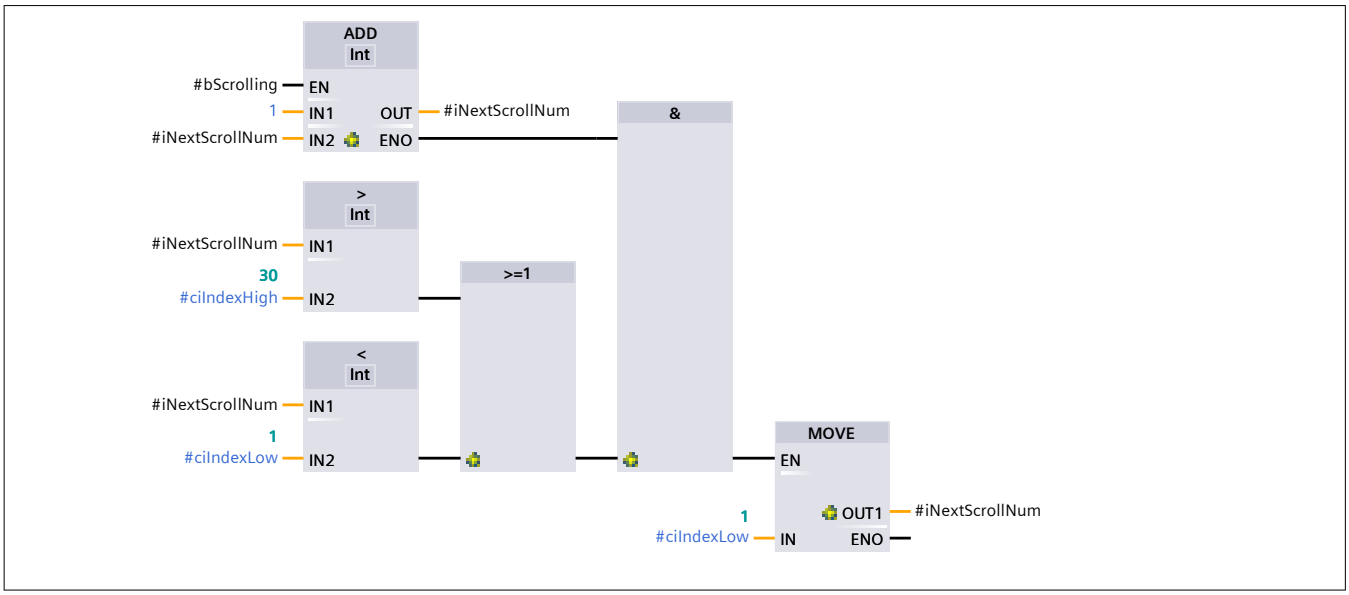
Network 4: Set bit to scroll for the next error

Set a static bit indicating that we are scrolling.
Each pass cycle through the function, we will increment.
If there is no error, set the scroll num to 0 and turn off scrolling.
It scrolls to an error immediately when an error occurs.



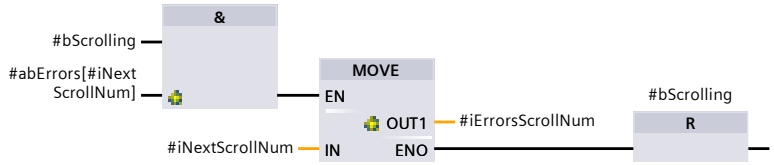
Network 5: Increment the error index checked

The function will only increment the scroll num one value per scan.
It will scroll until it finds the next active error, then stop scrolling.
Roll iNextScrollNum to 0 if it exceeds 30.



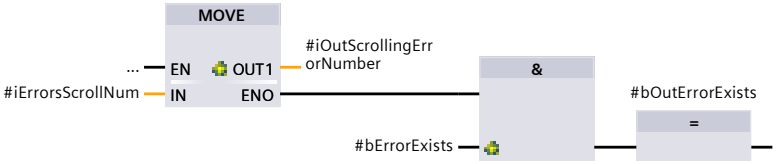
Network 6: Check if the next scroll number has an error

If we found an error, stop scrolling.

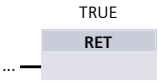


Network 7: Output: Error

Output the index of the error that exists.



Network 8: Set ENO



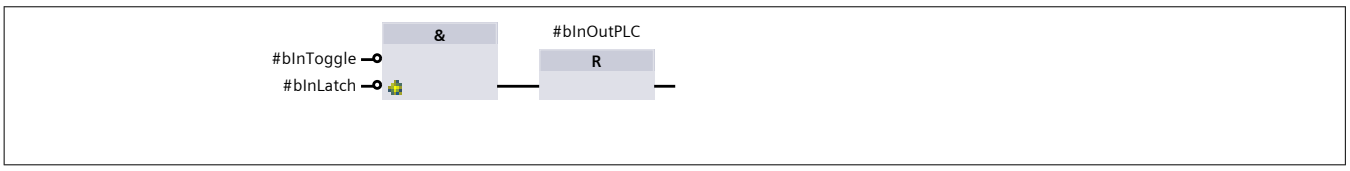
Open Library V15 / Resources / HMI

fcHMIBitEnable [FC329] [fcHMIBitEnable V 3.0.3]

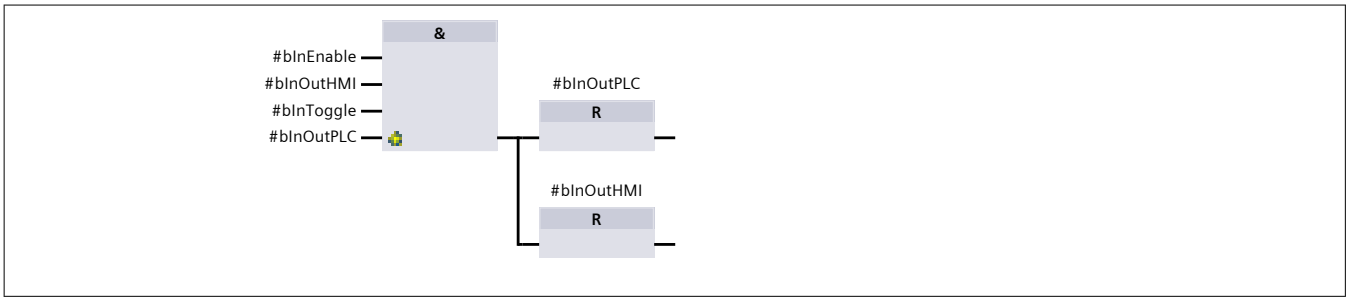
fcHMIBitEnable Properties					
General					
Name	fcHMIBitEnable	**Number**	329	**Type**	FC
Language	FBD	**Numbering**	Automatic		
Information					
Title	Handles an async reading of an HMI bit to avoid race conditions, includes an enable bit	**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value	Comment
▼ Input			
bInToggle	Bool		
bInLatch	Bool		
bInEnable	Bool		
▼ Output			
bOutEnable	Bool		
▼ InOut			
bInOutHMI	Bool		
bInOutPLC	Bool		
Temp			
Constant			
▼ Return			
fcHMIBitEnable	Void		

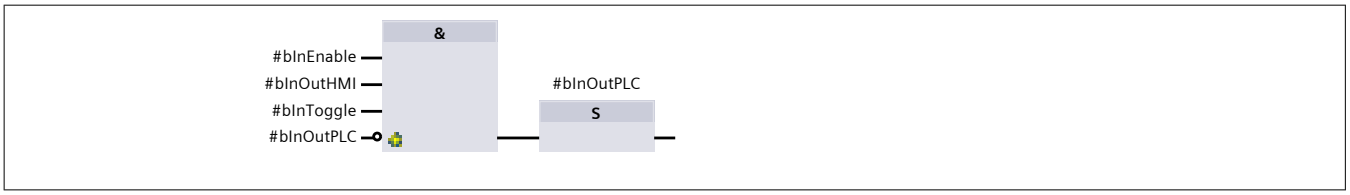
Network 1: If not latching or toggling, reset the PLC bit every scan



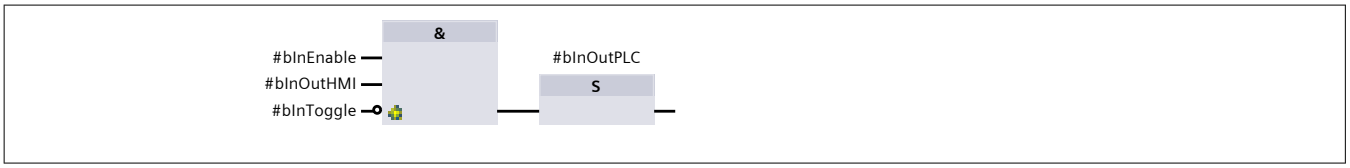
Network 2: Reset PLC bit if on and toggling



Network 3: Set PLC bit if off and toggling



Network 4: If not toggling, set PLC bit



Network 5: Pass through enable

Network 6: Reset HMI Bit

Network 7: Set ENO

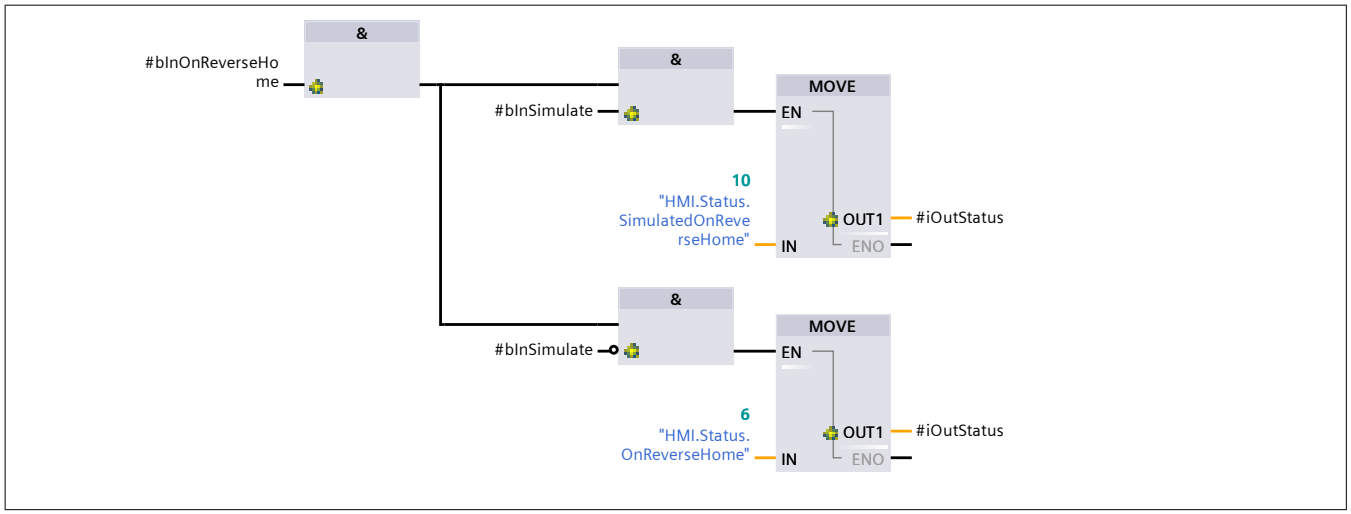
Open Library V15 / Resources / HMI

fcSetHMIStatusSimulation [FC333] [fcSetHMIStatusSimulation V 3.0.4]

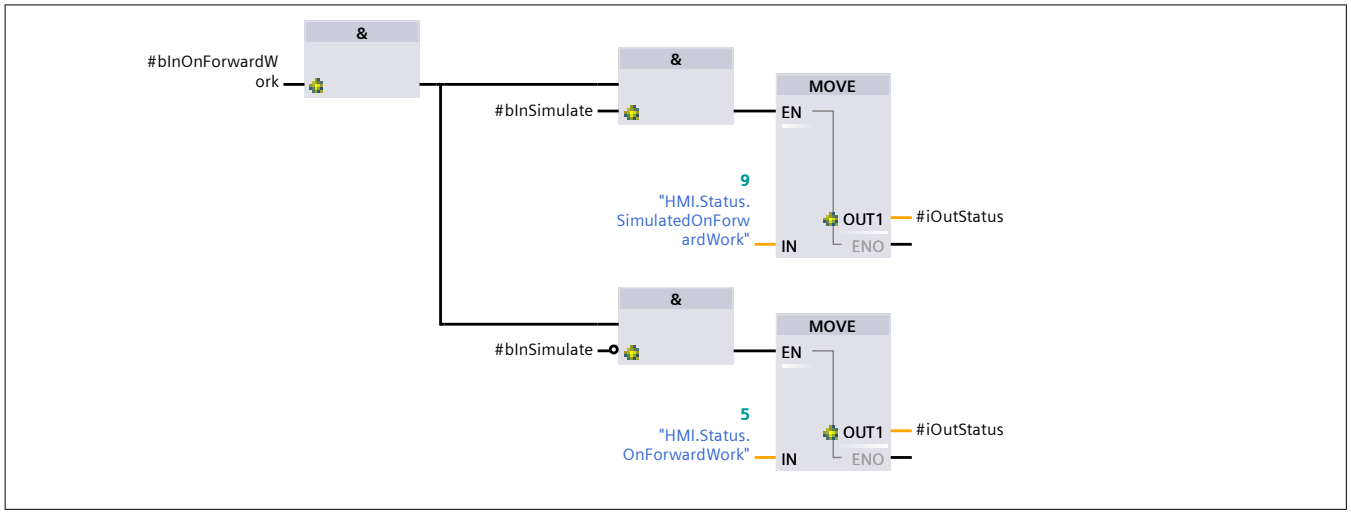
fcSetHMIStatusSimulation Properties					
General					
Name	fcSetHMIStatusSimula-tion	**Number**	333	**Type**	FC
Language	FBD	**Numbering**	Automatic		
Information					
Title	Sets HMI Status	**Author**		**Comment**	Wrapper function for set-ting the HMI status of a device In order of priority: 1. E-stop 2. Error 3. Forward/Work Position 4. Reverse/Home Position 5. Forward/Work On 6. Reverse/Home On 7. None
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value	Comment
▼ Input			
bInEstop	Bool		
bInError	Bool		
bInForwardWork	Bool		
bInReverseHome	Bool		
bInOnForwardWork	Bool		
bInOnReverseHome	Bool		
bInSimulate	Bool		
▼ Output			
iOutStatus	Int		
InOut			
Temp			
Constant			
▼ Return			
fcSetHMIStatusSimulation	Void		

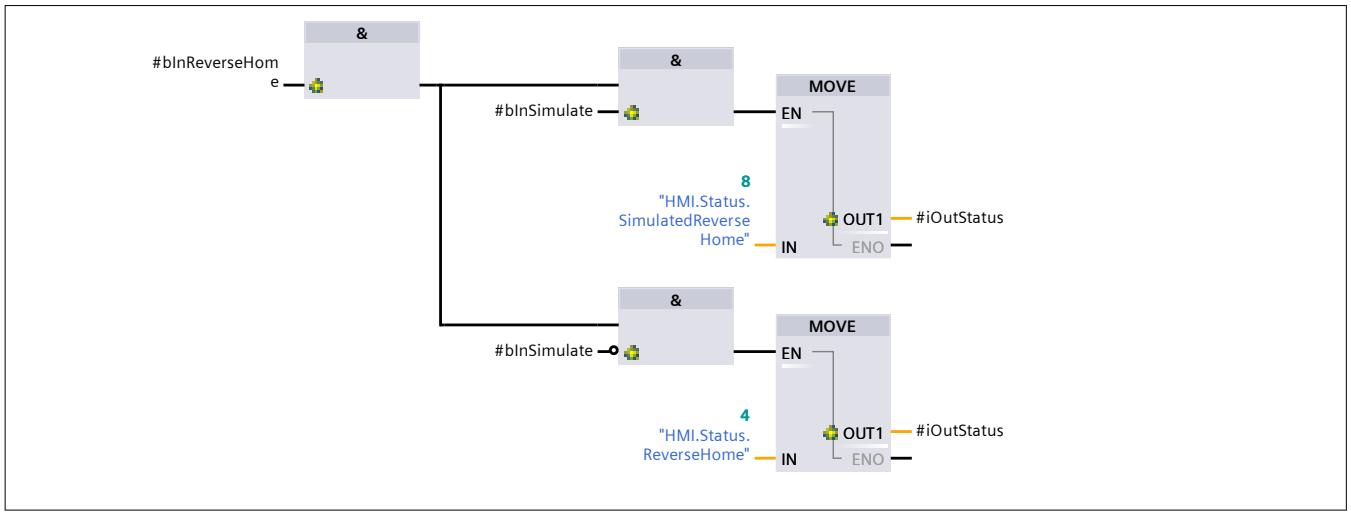
Network 1: Set Reverse/Home On



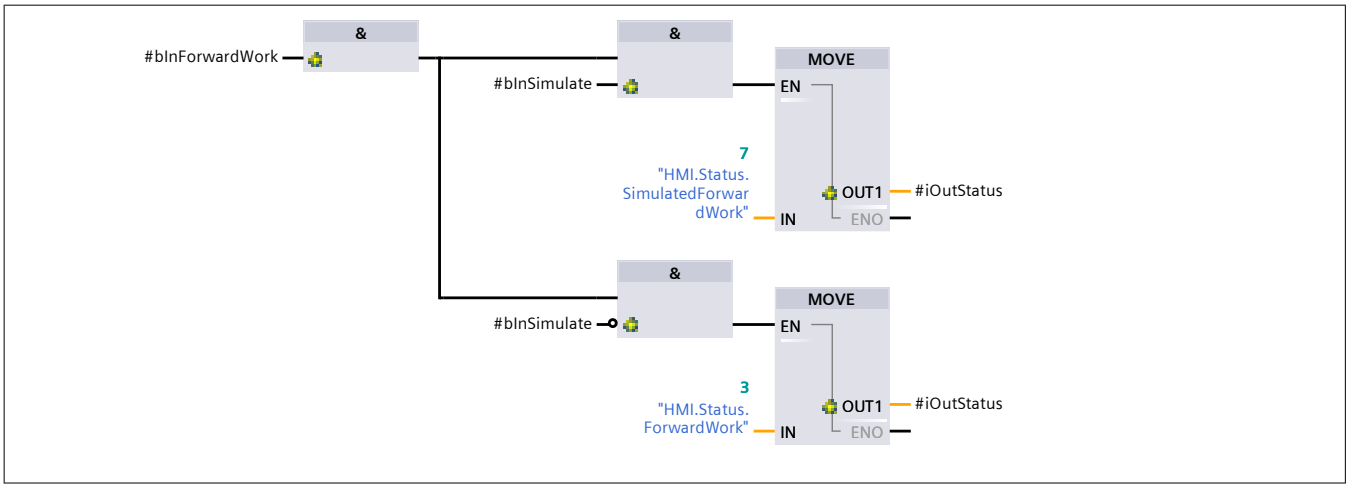
Network 2: Set Forward/Work On



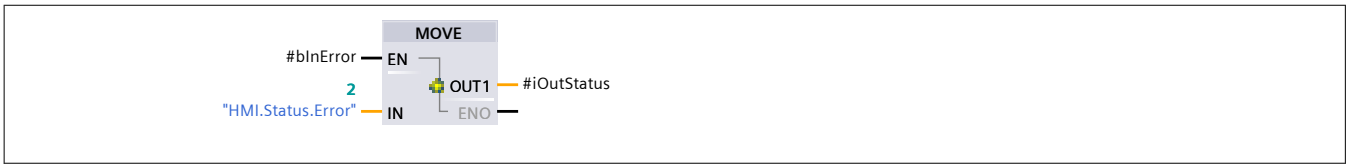
Network 3: Set Reverse/Home Position



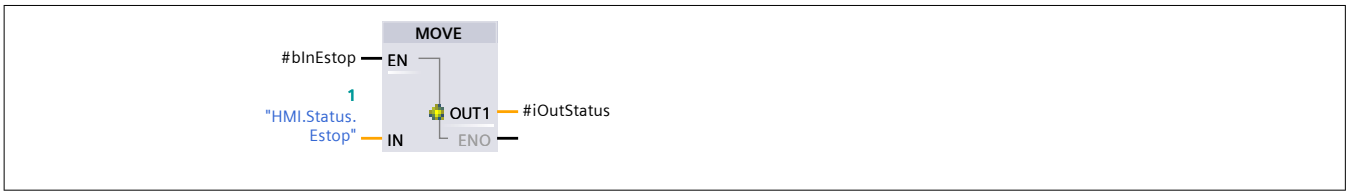
Network 4: Set Forward/Work Position



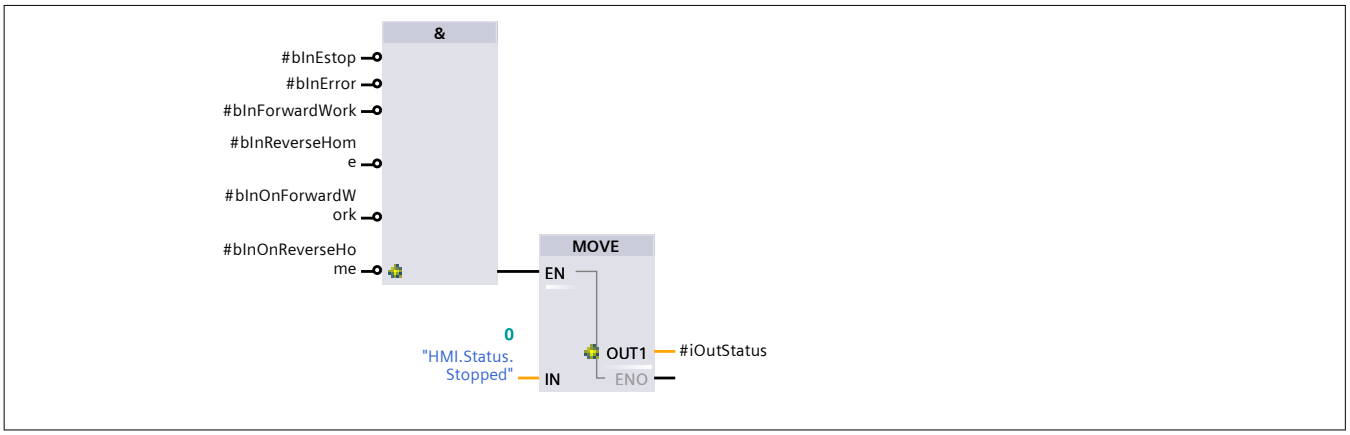
Network 5: Set Error



Network 6: Set Estop



Network 7: Set to none/stopped if nothing is selected



Network 8: Set ENO


TRUE
RET
...

Open Library V15 / Devices

fbValve_Solenoid [FB110] [fbValve_Solenoid V 3.0.6 in test]

fbValve_Solenoid Properties					
General					
Name	fbValve_Solenoid	**Number**	110	**Type**	FB
Language	FBD	**Numbering**	Manual		
Information					
Title	Controls a double- or single-acting valve	**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Set-point	Supervision	Comment
▼ Input									
tInTimeout	Time	T#0ms	Non-retain	False	False	False	False		Timeout time for an actuator's feedback to activate before giving a fault
iInMode	Int	0	Non-retain	False	False	False	False		Mode Selection
bInEstop	Bool	false	Non-retain	False	False	False	False		Estop
bInSignalHome	Bool	false	Non-retain	False	False	False	False		Home position feedback
bInSignalWork	Bool	false	Non-retain	False	False	False	False		Work position feedback
bInEnable	Bool	false	Non-retain	False	False	False	False		Home and Work Enabled
bInCommand-Work	Bool	false	Non-retain	False	False	False	False		Move to work position in automatic mode
bInResetError	Bool	false	Non-retain	False	False	False	False		Reset Error
bInSimulate	Bool	false	Non-retain	False	False	False	False		Activate device simulation
▼ Output									
bOutCommand-Home	Bool	false	Non-retain	False	False	False	False		Home position command
bOutCommand-Work	Bool	false	Non-retain	False	False	False	False		Work position command
bOutActiveHome	Bool	false	Non-retain	False	False	False	False		Valve is in home position
bOutActiveWork	Bool	false	Non-retain	False	False	False	False		Valve is in work position
bOutAuto	Bool	false	Non-retain	False	False	False	False		Block in auto mode

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bOutError	Bool	false	Non-retain	False	False	False	False		Error exists
▼ ERROR_Valve	"udtError_Valve"		Non-retain	False	False	False	False		Valve error structure
NoHomeFeedback	Bool	false	Non-retain	False	False	False	False		Home position feedback not active
NoWorkFeedback	Bool	false	Non-retain	False	False	False	False		Work position feedback not active
HomeFeedbackStillActive	Bool	false	Non-retain	False	False	False	False		Home position feedback still active
WorkFeedbackStillActive	Bool	false	Non-retain	False	False	False	False		Work position feedback still active
▼ InOut									
▼ HMI_ValveControl	"udtHMI_ValveControl"			False	False	False	False		HMI valve control
iMode	Int			False	False	False	False		Current mode
iErrorCode	Int			False	False	False	False		Error code
iStatus	Int			False	False	False	False		Status for HMI display
bPB_ResetError	Bool			False	False	False	False		PB Reset block errors
bPB_Home	Bool			False	False	False	False		PB Move to home in manual mode
bPB_Work	Bool			False	False	False	False		PB Move to work in manual mode
bPBEN_ResetError	Bool			False	False	False	False		PB Reset error enabled
bPBEN_Home	Bool			False	False	False	False		PB Home enabled
bPBEN_Work	Bool			False	False	False	False		PB Work enabled
bHomeOn	Bool			False	False	False	False		Home command is on
bWorkOn	Bool			False	False	False	False		Work command is on
bSignalHome	Bool			False	False	False	False		Home feedback
bSignalWork	Bool			False	False	False	False		Work feedback
bError	Bool			False	False	False	False		Error status
bInterlock	Bool			False	False	False	False		Valve interlocked

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
Static									
ErrorScroller	"fbErrorScroller"			False	False	False	True		Sub-block to handle error scroller
Input									
bInError01	Bool	false	Set in IDB	False	False	False	False		Error 1 In Error Array
bInError02	Bool	false	Set in IDB	False	False	False	False		Error 2 In Error Array
bInError03	Bool	false	Set in IDB	False	False	False	False		Error 3 In Error Array
bInError04	Bool	false	Set in IDB	False	False	False	False		Error 4 In Error Array
bInError05	Bool	false	Set in IDB	False	False	False	False		Error 5 In Error Array
bInError06	Bool	false	Set in IDB	False	False	False	False		Error 6 In Error Array
bInError07	Bool	false	Set in IDB	False	False	False	False		Error 7 In Error Array
bInError08	Bool	false	Set in IDB	False	False	False	False		Error 8 In Error Array
bInError09	Bool	false	Set in IDB	False	False	False	False		Error 9 In Error Array
bInError10	Bool	false	Set in IDB	False	False	False	False		Error 10 In Error Array
bInError11	Bool	false	Set in IDB	False	False	False	False		Error 11 In Error Array
bInError12	Bool	false	Set in IDB	False	False	False	False		Error 12 In Error Array
bInError13	Bool	false	Set in IDB	False	False	False	False		Error 13 In Error Array
bInError14	Bool	false	Set in IDB	False	False	False	False		Error 14 In Error Array
bInError15	Bool	false	Set in IDB	False	False	False	False		Error 15 In Error Array
bInError16	Bool	false	Set in IDB	False	False	False	False		Error 16 In Error Array
bInError17	Bool	false	Set in IDB	False	False	False	False		Error 17 In Error Array
bInError18	Bool	false	Set in IDB	False	False	False	False		Error 18 In Error Array
bInError19	Bool	false	Set in IDB	False	False	False	False		Error 19 In Error Array
bInError20	Bool	false	Set in IDB	False	False	False	False		Error 20 In Error Array
bInError21	Bool	false	Set in IDB	False	False	False	False		Error 21 In Error Array
bInError22	Bool	false	Set in IDB	False	False	False	False		Error 22 In Error Array

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
bInError23	Bool	false	Set in IDB	False	False	False	False		Error 23 In Error Array
bInError24	Bool	false	Set in IDB	False	False	False	False		Error 24 In Error Array
bInError25	Bool	false	Set in IDB	False	False	False	False		Error 25 In Error Array
bInError26	Bool	false	Set in IDB	False	False	False	False		Error 26 In Error Array
bInError27	Bool	false	Set in IDB	False	False	False	False		Error 27 In Error Array
bInError28	Bool	false	Set in IDB	False	False	False	False		Error 28 In Error Array
bInError29	Bool	false	Set in IDB	False	False	False	False		Error 29 In Error Array
bInError30	Bool	false	Set in IDB	False	False	False	False		Error 30 In Error Array
▼ Output									
bOutErrorExists	Bool	false	Set in IDB	False	False	False	False		An Error Exists
iOutScrollingErrorNumber	Int	0	Set in IDB	False	False	False	False		Current Index when scrolling through errors
InOut									
▼ Static									
▼ TON_ErrorDelay	TON_TIME		Non-retain	False	False	False	True		Delay between each scroll for a new error
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
▼ abErrors	Array[1..30] of Bool		Set in IDB	False	False	False	False		
abErrors[1]	Bool	false	Set in IDB	False	False	False	False		
abErrors[2]	Bool	false	Set in IDB	False	False	False	False		
abErrors[3]	Bool	false	Set in IDB	False	False	False	False		
abErrors[4]	Bool	false	Set in IDB	False	False	False	False		
abErrors[5]	Bool	false	Set in IDB	False	False	False	False		

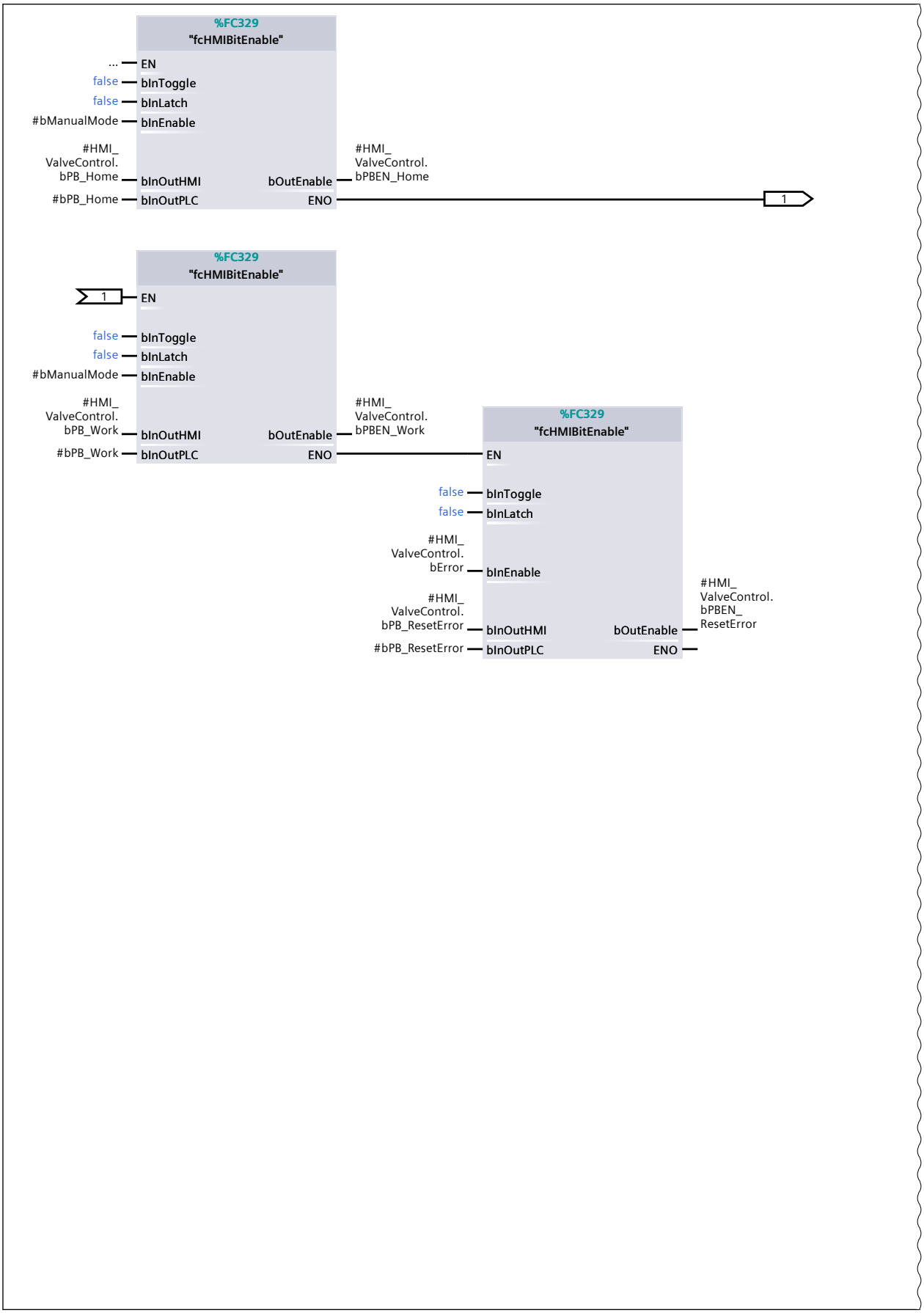
Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[6]	Bool	false	Set in IDB	False	False	False	False		
abErrors[7]	Bool	false	Set in IDB	False	False	False	False		
abErrors[8]	Bool	false	Set in IDB	False	False	False	False		
abErrors[9]	Bool	false	Set in IDB	False	False	False	False		
abErrors[10]	Bool	false	Set in IDB	False	False	False	False		
abErrors[11]	Bool	false	Set in IDB	False	False	False	False		
abErrors[12]	Bool	false	Set in IDB	False	False	False	False		
abErrors[13]	Bool	false	Set in IDB	False	False	False	False		
abErrors[14]	Bool	false	Set in IDB	False	False	False	False		
abErrors[15]	Bool	false	Set in IDB	False	False	False	False		
abErrors[16]	Bool	false	Set in IDB	False	False	False	False		
abErrors[17]	Bool	false	Set in IDB	False	False	False	False		
abErrors[18]	Bool	false	Set in IDB	False	False	False	False		
abErrors[19]	Bool	false	Set in IDB	False	False	False	False		
abErrors[20]	Bool	false	Set in IDB	False	False	False	False		
abErrors[21]	Bool	false	Set in IDB	False	False	False	False		
abErrors[22]	Bool	false	Set in IDB	False	False	False	False		
abErrors[23]	Bool	false	Set in IDB	False	False	False	False		
abErrors[24]	Bool	false	Set in IDB	False	False	False	False		
abErrors[25]	Bool	false	Set in IDB	False	False	False	False		
abErrors[26]	Bool	false	Set in IDB	False	False	False	False		
abErrors[27]	Bool	false	Set in IDB	False	False	False	False		
abErrors[28]	Bool	false	Set in IDB	False	False	False	False		
abErrors[29]	Bool	false	Set in IDB	False	False	False	False		

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
abErrors[30]	Bool	false	Set in IDB	False	False	False	False		
iNextScrollNum	Int	1	Non-retain	False	False	False	False		Index of next error position
iErrorsScrollNum	Int	1	Non-retain	False	False	False	False		Index of current error position
bScrolling	Bool	false	Non-retain	False	False	False	False		Indicates we are scrolling
bTON_ErrorDelay	Bool	false	Non-retain	False	False	False	False		
▼ TON_TimeOut	TON_TIME		Non-retain	False	False	False	True		Timer for Error Timeout
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
iLastMode	Int	0	Non-retain	False	False	False	False		Last mode code
bNewMode	Bool	false	Non-retain	False	False	False	False		New mode selected
bEnableHome	Bool	false	Non-retain	False	False	False	False		Home position enabled
bEnableWork	Bool	false	Non-retain	False	False	False	False		Work position enabled
bAutoMode	Bool	false	Non-retain	False	False	False	False		Auto Mode is active
bManualMode	Bool	false	Non-retain	False	False	False	False		Manual Mode is active
bReset	Bool	false	Non-retain	False	False	False	False		
bTON_TimeOut	Bool	false	Non-retain	False	False	False	False		
bPB_Home	Bool	false	Non-retain	False	False	False	False		
bPB_Work	Bool	false	Non-retain	False	False	False	False		
bPB_ResetError	Bool	false	Non-retain	False	False	False	False		
▼ Temp									
tTimeoutElapsed	Time								Elapsed time for TON_Timeout
bTemp	Bool								
Constant									

Network 1: ---------- Inputs

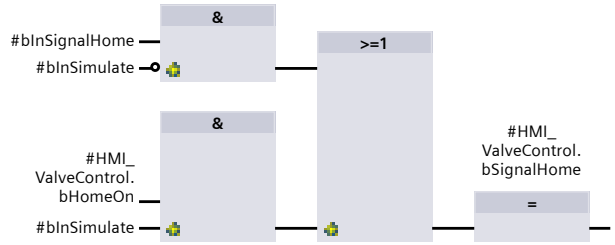
Network 2: Read HMI input buttons

Network 2: Read HMI input buttons (1.1 / 1.2)

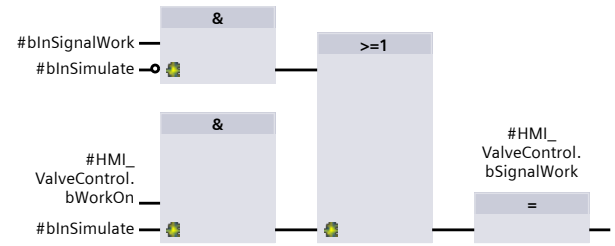


Network 2: Read HMI input buttons (1.2 / 1.2)

Network 3: Update the signal home bit

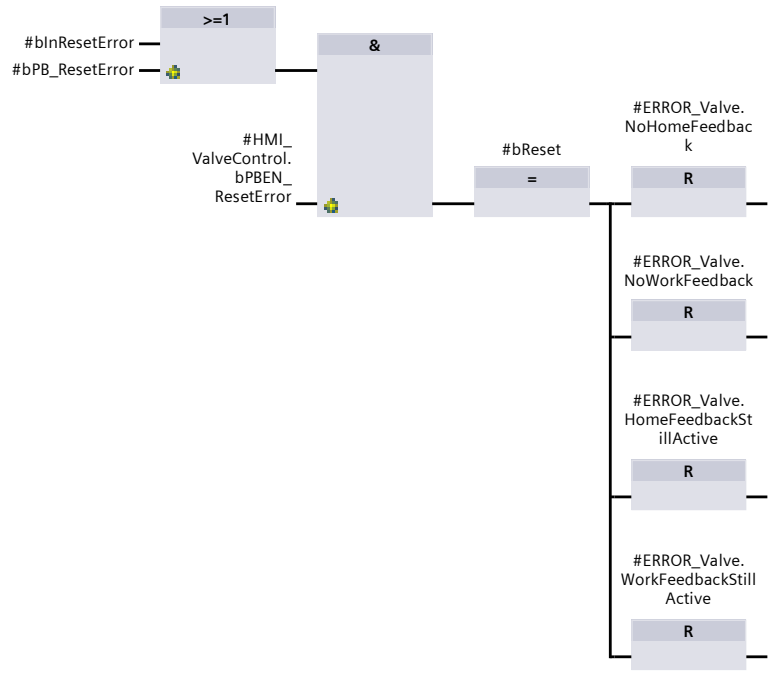


Network 4: Update the signal work bit

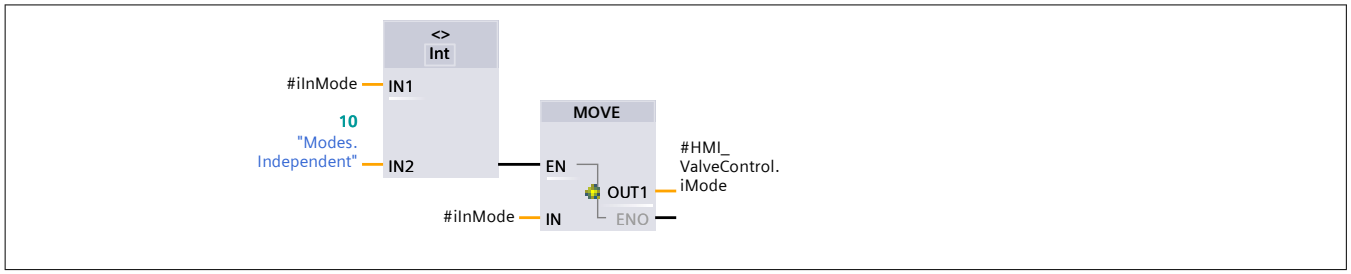


Network 5: ---------- Control Logic ----------

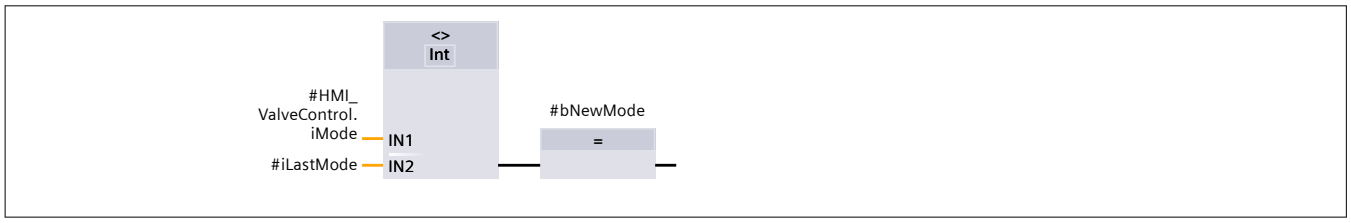
Network 6: Clear all errors



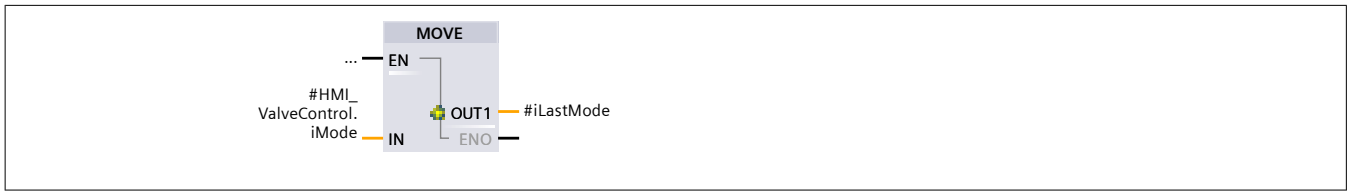
Network 7: Override HMI mode selection



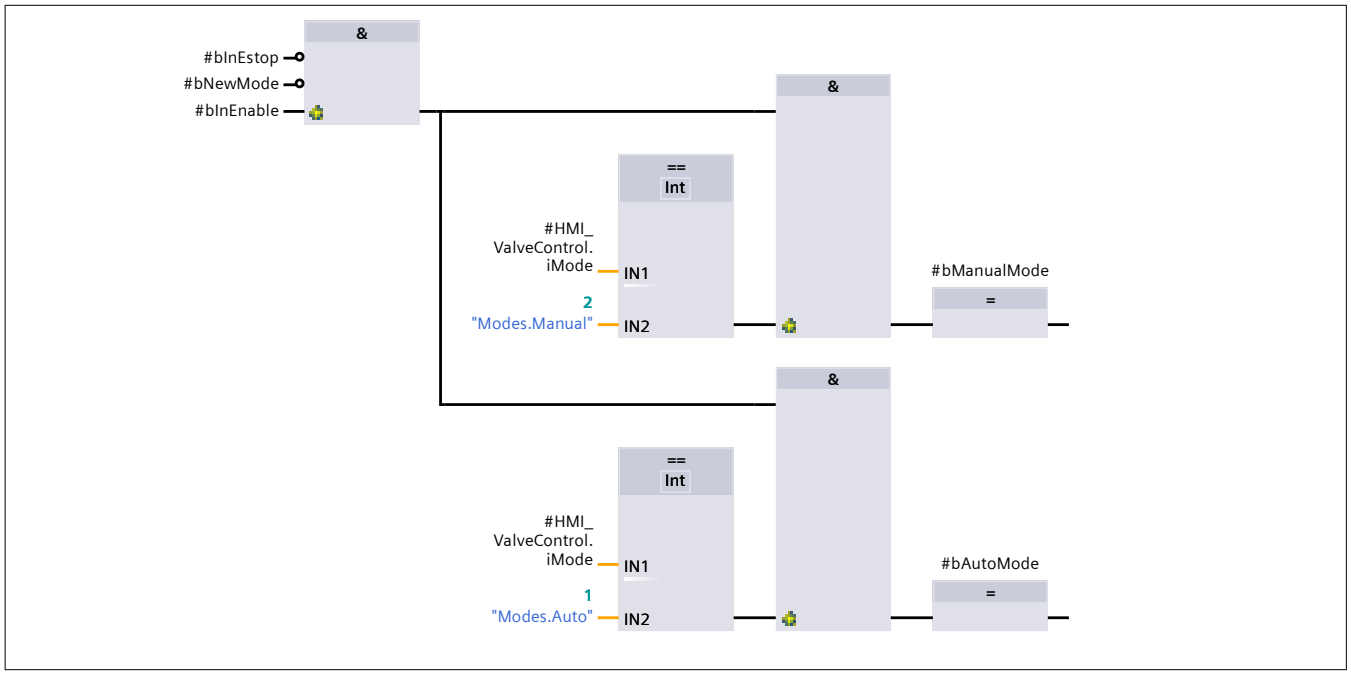
Network 8: Detect change in mode



Network 9: Keep track of last mode

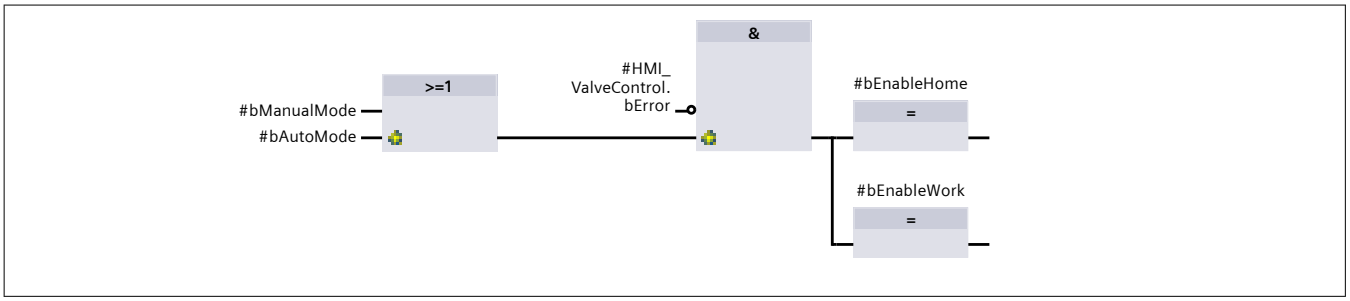


Network 10: Determine if auto or manual mode is active



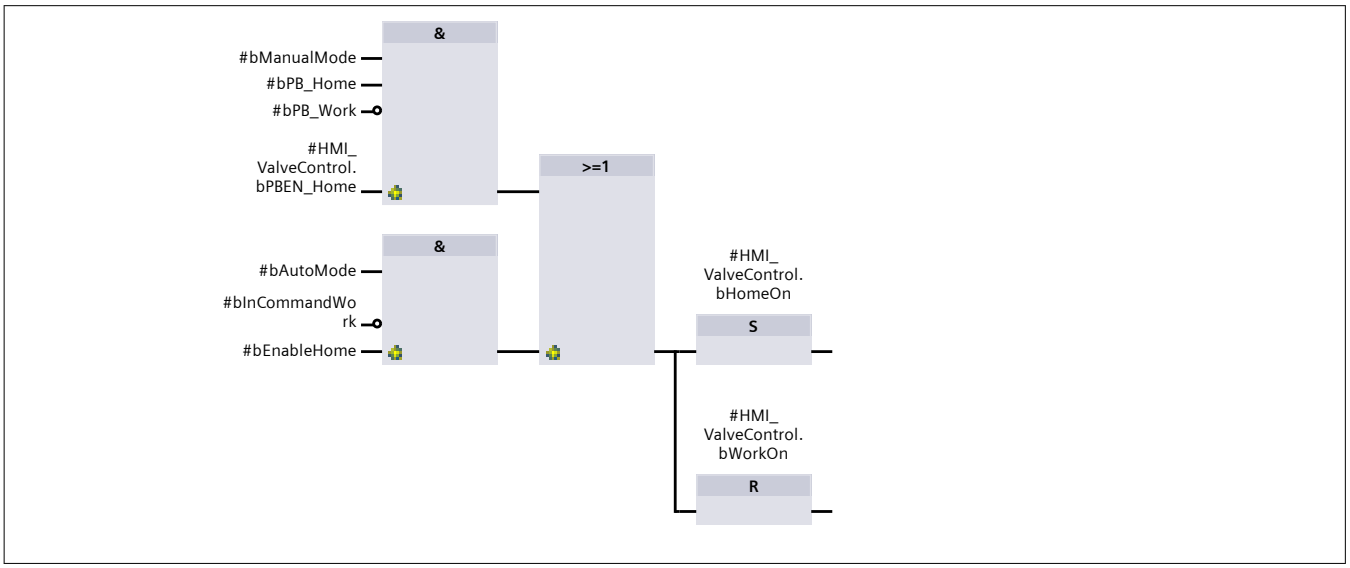
Network 11: Enable Home & Work Positions

IF manual mode AND no estop AND not a new mode AND enabled =>
--if no cylinder errors, enable move to Home and Work
--enable home and work HMI buttons only in manual mode



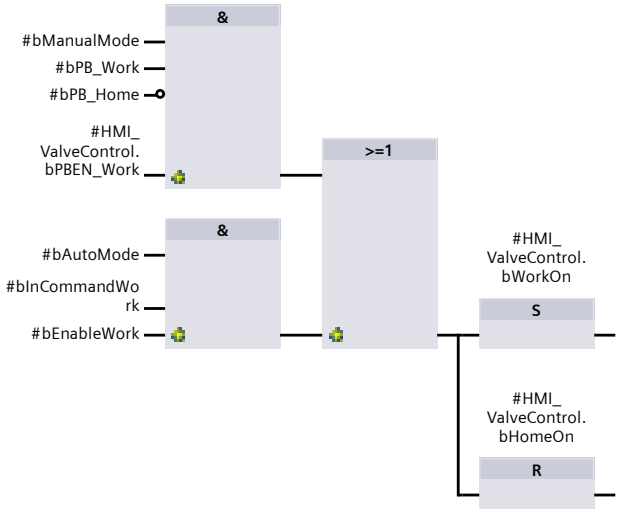
Network 12: Move to Home Position (Retracted)

IF manual mode and home button is pressed =>
--Set close cylinder



Network 13: Move to Work Position (Extended)

IF manual mode and work button button is pressed =>
--Set open cylinder



Network 14: Reset home on if not enabled

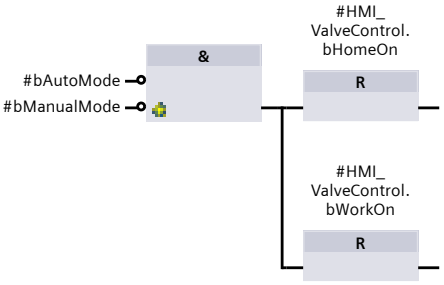
IF in auto AND home is not enabled => RESET home command

Network 15: Reset work on if not enabled

IF in auto AND work is not enabled => RESET work command

Network 16: Reset both home and work if not in auto or manual

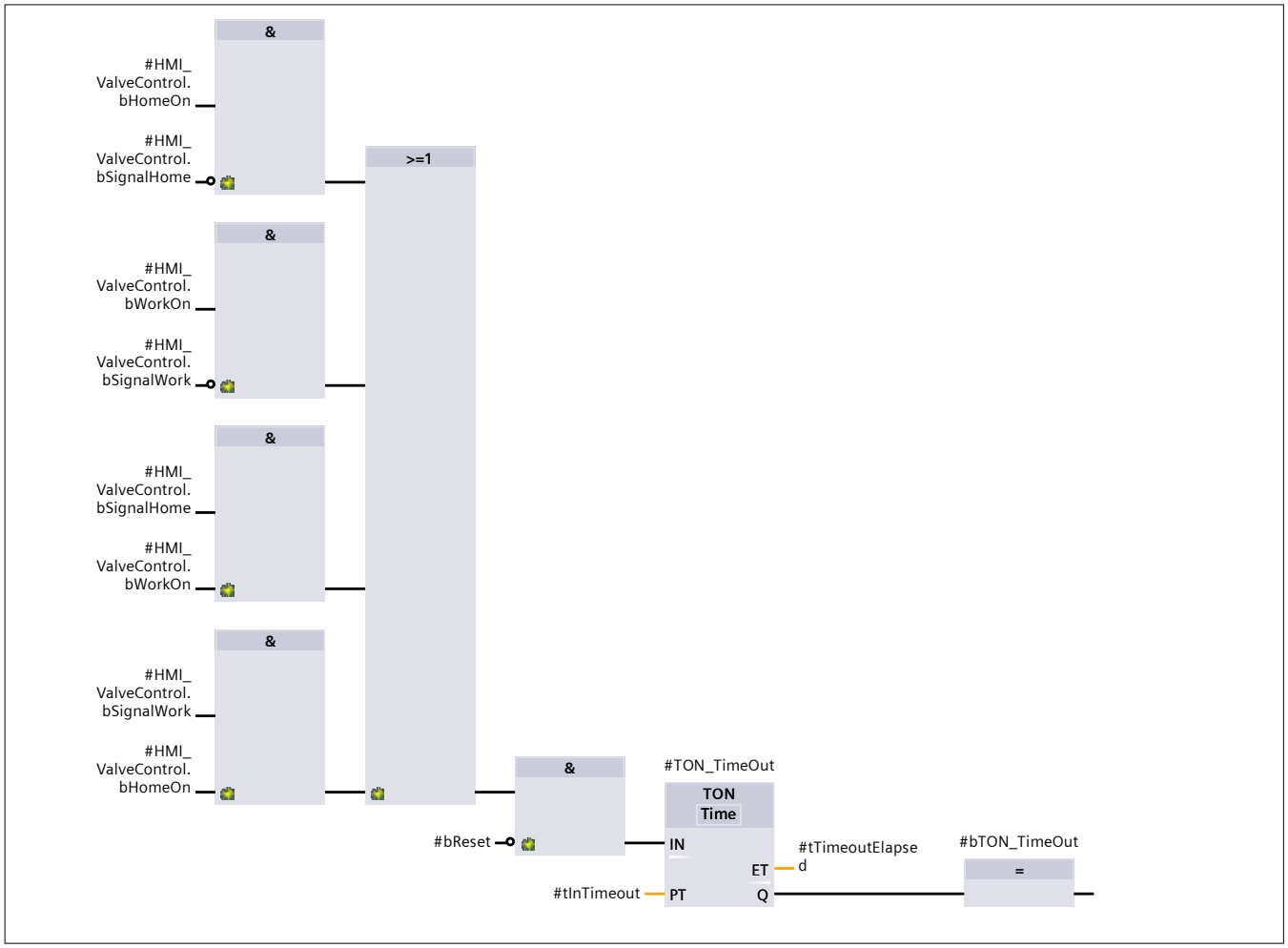
IF not in auto AND not in manual => RESET home and work commands



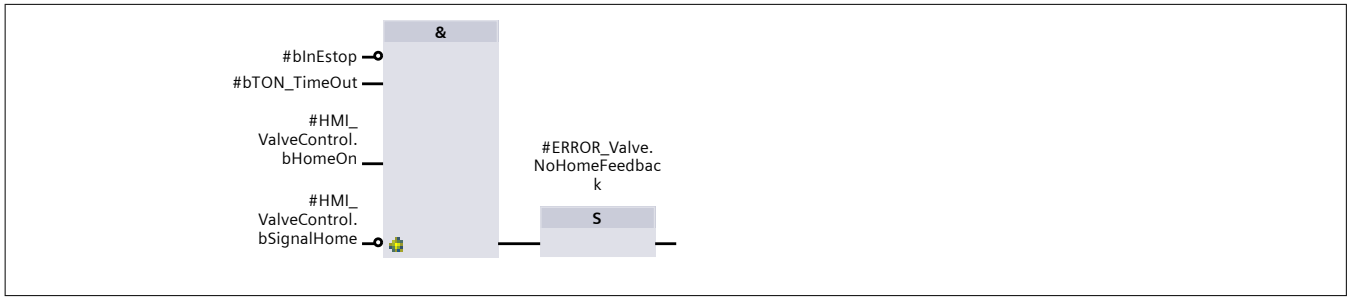
Network 17: ---------- Errors ----------

Network 18: Timeout timer

Time delay for errors

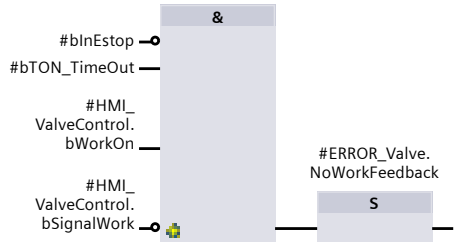


Network 19: Error: Home position feedback not active



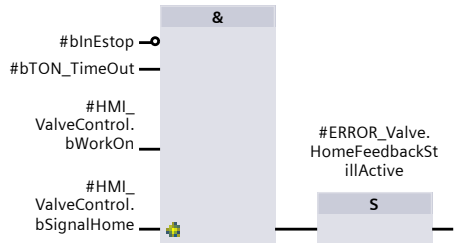
Network 20: Error: Work position feedback not active

Not reach work position



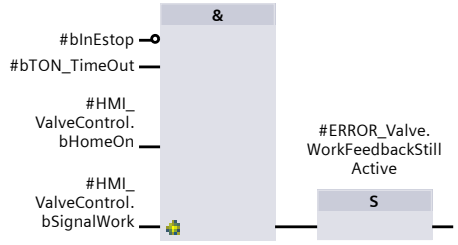
Network 21: Error: Home position feedback still active

Home position still active



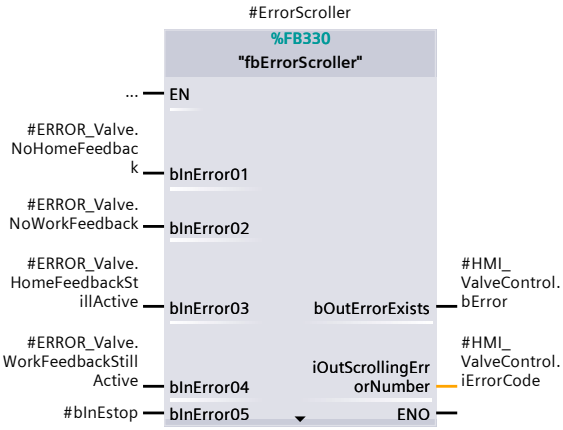
Network 22: Error: Work position feedback still active

Work position still active



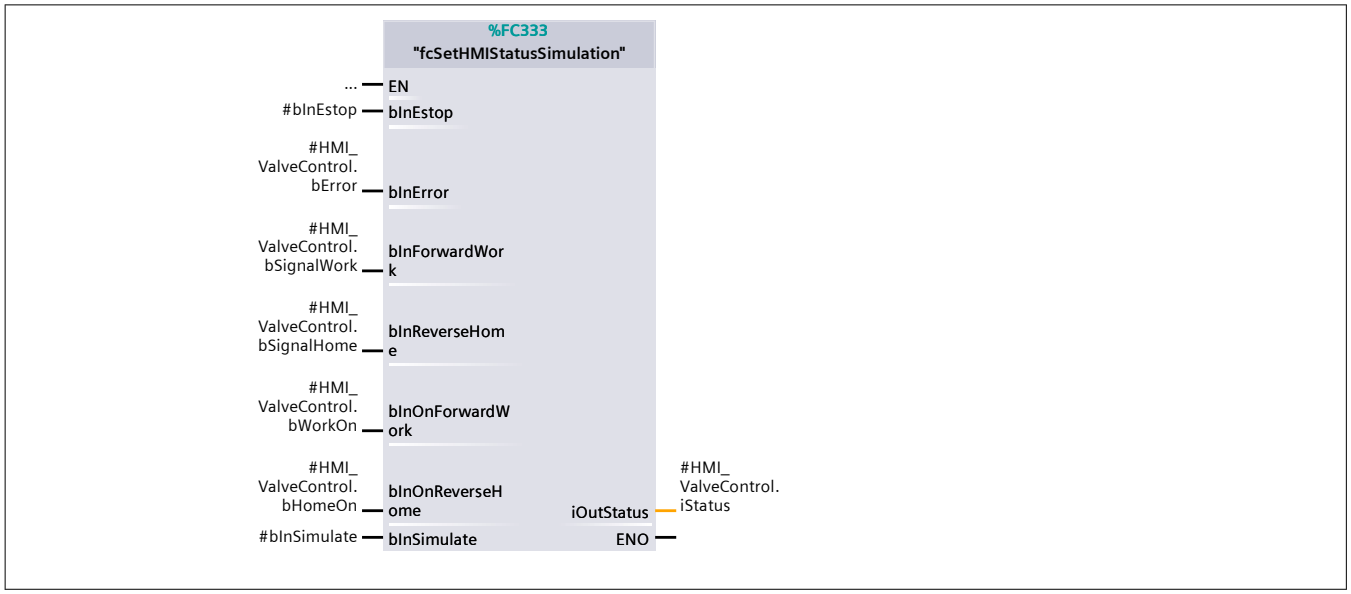
Network 23: Error scroller

Call the Error-Scroller which cyles through all possible errors and checks their state for display on the HMI



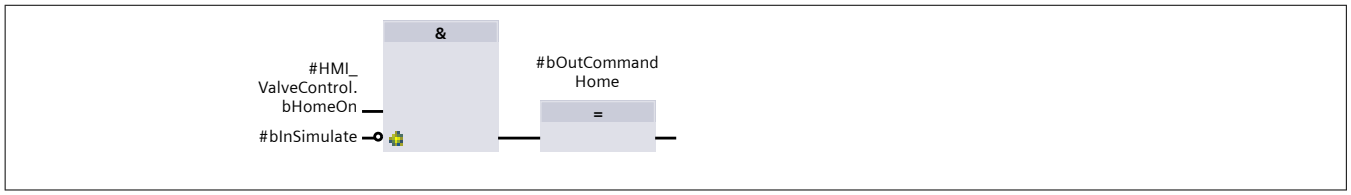
Network 24: Interlock

Network 25: Set HMI status

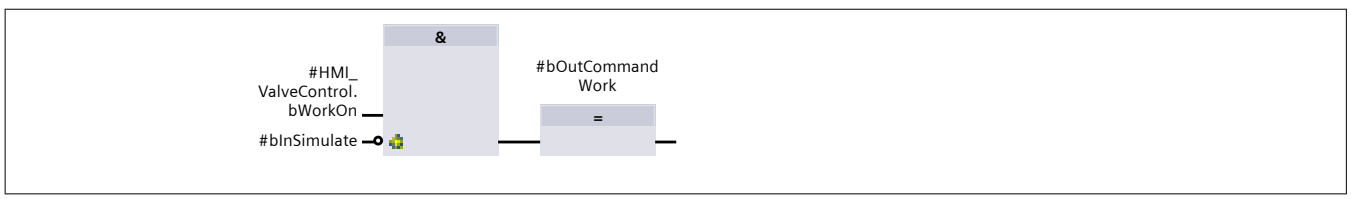


Network 26: ---------- Outputs ----------

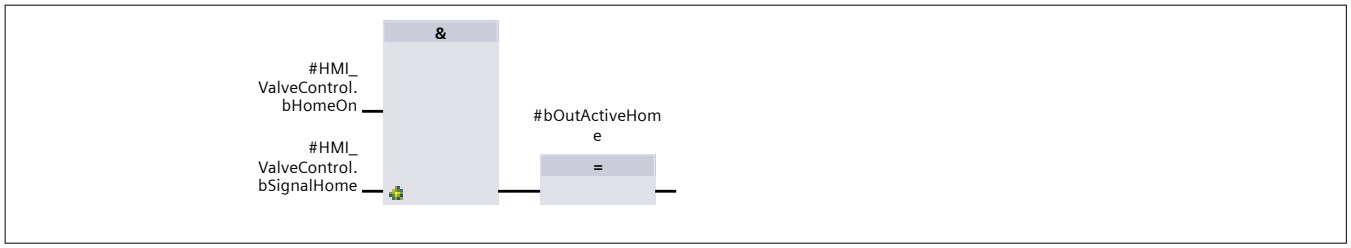
Network 27: Output: Home Command



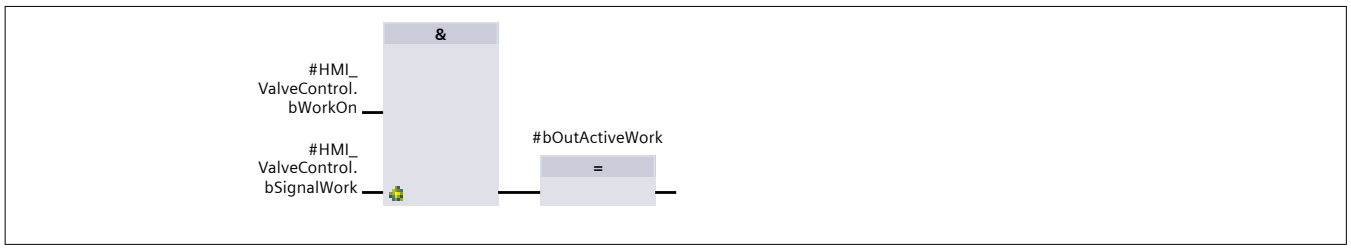
Network 28: Output: Work Command



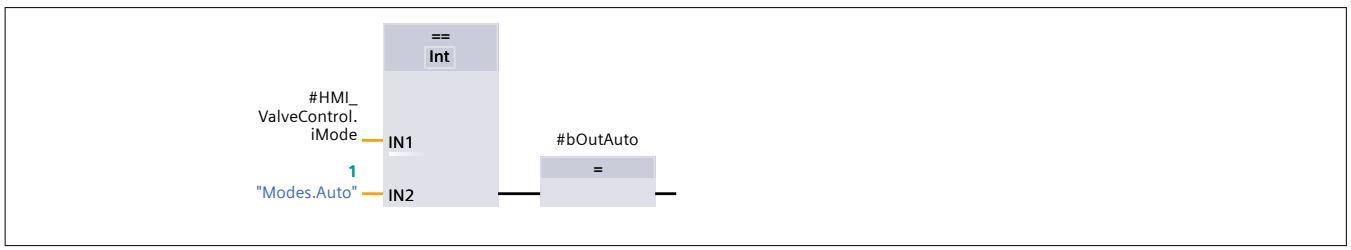
Network 29: Output: Home Active



Network 30: Output: Work Active



Network 31: Output: Auto



Network 32: Output: Error

Network 33: Set ENO

Open Library V15 / Process

fbStepSequencer [FB185] [fbStepSequencer V 3.0.1]

fbStepSequencer Properties					
General					
Name	fbStepSequencer	**Number**	185	**Type**	FB
Language	SCL	**Numbering**	Manual		
Information					
Title	Step sequence controller	**Author**	Boris	**Comment**	Modified by Ken Brey 2010-08-25 KLB. Added Initialization inputs and logic. Modified by Ken Brey 2006-08-08 Removed initialization of inputs. Initializing inputs doesn't do anything. The input value if un-wired will be the last value set for that input in a different call. The initialized value in the declaration section does not set it back to default if unwired. Allways supply values for all inputs. Modified by Tim Jager 2005-12-13 no block Id in error message Modified by Boris: branched -> simplified version, no errors, no timeout; still have to use static memory for TON/CurStep (might be fixed later using in/out structure) Modified by Nick Shea: 2012-03-30 Re-instated step mode
Family		**Version**	4.0	**User-defined ID**	

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input									
iInStep	Int	0	Non-retain	False	False	False	False		Constant: current step required for the FB to run

Name	Data type	Default value	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Setpoint	Supervision	Comment
iInNextStep	Int	0	Non-retain	False	False	False	False		Next step to go to if step is done
bInStepDone	Bool	false	Non-retain	False	False	False	False		if step done logic is true, then we move from IN_STEP to NEXT_STEP
tInStepDelay	Time	T#0ms	Non-retain	False	False	False	False		After the step done input is true, we wait for x seconds before moving to next
bInStepMode	Bool	false	Non-retain	False	False	False	False		Pause sequence at completion of current step
bInStepAdvance	Bool	false	Non-retain	False	False	False	False		If paused, trigger next step
bInit	Bool	false	Non-retain	False	False	False	False		
iInitStep	Int	0	Non-retain	False	False	False	False		
▼ Output									
bOutEnterEvent	Bool	false	Non-retain	False	False	False	False		
bOutExitEvent	Bool	false	Non-retain	False	False	False	False		
bOutPaused	Bool	false	Non-retain	False	False	False	False		Sequence is currently paused.
iOutCurrentStep	Int	0	Non-retain	False	False	False	False		output current step
InOut									
▼ Static									
▼ TON_StepDelay	TON_TIME		Non-retain	False	False	False	True		used to delay moving to the next step.
PT	Time	T#0ms	Non-retain	False	False	False	False		
ET	Time	T#0ms	Non-retain	False	False	False	False		
IN	Bool	false	Non-retain	False	False	False	False		
Q	Bool	false	Non-retain	False	False	False	False		
iCurrentStep	Int	0	Non-retain	False	False	False	False		Current step of the sequencer
bAlreadyInThisStep	Bool	false	Non-retain	False	False	False	False		
▼ Temp									
bNoStepDelay	Bool								step delay disabled

Name	Data type	Default value	Retain	Acces-sible from HMI/OPC UA/Web API	Wri ta-ble fro m HM I/O PC UA/ We b API	Visible in HMI engi-neer-ing	Set-point	Super-vision	Comment
Constant									

```
0001
0002
0003 //
     ***********************************************************************************
     ***********
0004 // INSTRUCTION SECTION
0005 ENO:=false; //init variables
0006 #bOutExitEvent :=false;
0007 #bOutEnterEvent:=false;
0008
0009 IF (#iInStep=#iCurrentStep) THEN
0010   IF NOT #bAlreadyInThisStep THEN
0011     #bOutEnterEvent:=true;
0012     #bAlreadyInThisStep:=true;
0013   END_IF;
0014   IF #bInStepDone THEN
0015     //Run Step done Timer
0016     #TON_StepDelay(IN:=(TRUE),PT:=#tInStepDelay);
0017
0018     //check if we are using a step delay
0019     #bNoStepDelay:= (#tInStepDelay<=T#0MS);
0020
0021     //We can move to the next step if (STEP_DONE and StepDelay timer complete)
0022     // AND
0023     //(we are in step mode and the step advance is true, or if we are not in
     step mode
0024     // AND
0025     //It is not the first scan of this step
0026     IF ((#TON_StepDelay.Q OR #bNoStepDelay) AND (NOT #bInStepMode OR (#bIn-
     StepMode AND #bInStepAdvance)) AND NOT #bOutEnterEvent) THEN
0027       #iCurrentStep   := #iInNextStep;   //move to next step
0028       //Reset Step done Timer
0029       #TON_StepDelay(IN:=(false),PT:=#tInStepDelay);
0030       #bOutExitEvent:=true;
0031       #bAlreadyInThisStep:=false;
0032     END_IF;
0033   END_IF;
0034   #bOutPaused := #bInStepDone AND (#TON_StepDelay.Q OR #bNoStepDelay) AND
     (#bInStepMode AND NOT #bInStepAdvance);
0035   //Only set this if we are ok and in the right step
0036   ENO:= true;
0037 END_IF;
0038 IF (#bInit) THEN
0039   #iCurrentStep := #iInitStep;
0040   #bInit:=false;  //Clear so that if it is unwired on the next call, it is
     not run.
0041   #bAlreadyInThisStep:=false;
```

```
0042 END_IF;
0043
0044 //Set output variables
0045 #iOutCurrentStep := #iCurrentStep;
0046 #tInStepDelay := T#0MS;
0047
```

Symbol	Address	Type	Comment
#bAlreadyInThisStep		Bool	
#bInit		Bool	
#bInStepAdvance		Bool	If paused, trigger next step
#bInStepDone		Bool	if step done logic is true, then we move from IN_STEP to NEXT_STEP
#bInStepMode		Bool	Pause sequence at completion of current step
#bNoStepDelay		Bool	step delay disabled
#bOutEnterEvent		Bool	
#bOutExitEvent		Bool	
#bOutPaused		Bool	Sequence is currently paused.
#iCurrentStep		Int	Current step of the sequencer
#iInitStep		Int	
#iInNextStep		Int	Next step to go to if step is done
#iInStep		Int	Constant: current step required for the FB to run
#iOutCurrentStep		Int	output current step
#tInStepDelay		Time	After the step done input is true, we wait for x seconds before moving to next
#TON_StepDelay		IEC_Timer	used to delay moving to the next step.
#TON_StepDelay.Q		Bool	

Open Library V15 / Process

fcStepChooser [FC185] [fcStepChooser V 3.0.1]

fcStepChooser Properties					
General					
Name	fcStepChooser	**Number**	185	**Type**	FC
Language	SCL	**Numbering**	Manual		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.0	**User-defined ID**	

Name	Data type	Default value	Comment
▼ Input			
bCondition	Bool		
iTrueStep	Int		
iFalseStep	Int		
Output			
InOut			
▼ Temp			
iResult	Int		
Constant			
▼ Return			
fcStepChooser	Int		

```
0001
0002    IF #bCondition THEN
0003      #iResult:=#iTrueStep;
0004    ELSE
0005      #iResult:=#iFalseStep;
0006    END_IF;
0007    ENO:=true;
0008    #fcStepChooser:=#iResult;
0009
```

Symbol	Address	Type	Comment
#bCondition		Bool	
#fcStepChooser		Int	
#iFalseStep		Int	
#iResult		Int	
#iTrueStep		Int	

B. APPENDIX: PLC INSTRUCTIONS

```
Coil:
  output = input

Not:
  output = !input

Set:
  output = (if input then TRUE)

Reset:
  output = (if input then FALSE)

SetReset:
  output = ((if input1 then TRUE) (elseif input2 then FALSE)

And:
  output = input1 && input2

Or:
  output = input1 || input2

Equal:
  output = input1 == input2

LessThen:
  output = input1 < input2

GreaterThen:
  output = input1 > input2

Move:
  output = input
  or
  output = (if input1 then input2)

Add:
  output = input1 + input2

Call block1:
  Call: block1 (name of block1)
  Inputs:
  var1 ==> input1
  var2 ==> input2
  Outputs:
  output1 ==> var3
  output2 ==> var4
```

C. Appendix: Export list of PLC program blocks

```
OrganizationBlock: OB1 (Main)
XMLFileLocation: C:\...\Main.xml
ProgrammingLanguage: FBD
FunctionBlock: FB2 (Valve Liquid 1)
XMLFileLocation: C:\...\Valve Liquid 1.xml
ProgrammingLanguage: FBD
FunctionBlock: FB3 (Valve Liquid 2)
XMLFileLocation: C:\...\Valve Liquid 2.xml
ProgrammingLanguage: FBD
FunctionBlock: FB4 (Valve Drain)
XMLFileLocation: C:\...\Valve Drain.xml
ProgrammingLanguage: FBD
FunctionBlock: FB5 (Program Logic)
XMLFileLocation: C:\...\Program Logic.xml
ProgrammingLanguage: FBD
FunctionBlock: FB110 (fbValve_Solenoid)
XMLFileLocation: C:\...\Open Library V15\Devices\fbValve_SolenoidFBD.xml
ProgrammingLanguage: FBD
FunctionBlock: FB185 (fbStepSequencer)
XMLFileLocation: C:\...\Open Library V15\Process\fbStepSequencer.xml
ProgrammingLanguage: SCL
FunctionBlock: FB330 (fbErrorScroller)
XMLFileLocation: C:\...\Open Library V15\Resources\HMI\fbErrorScrollerFBD.xml
ProgrammingLanguage: FBD
Function: FC185 (fcStepChooser)
XMLFileLocation: C:\...\Open Library V15\Process\fcStepChooser.xml
ProgrammingLanguage: SCL
Function: FC329 (fcHMIBitEnable)
XMLFileLocation: C:\...\Open Library V15\Resources\HMI\fcHMIBitEnableFBD.xml
ProgrammingLanguage: FBD
Function: FC333 (fcSetHMIStatusSimulation)
XMLFileLocation: C:\...\Open Library V15\Resources\HMI\fcSetHMIStatusSimulationFBD.
    xml
ProgrammingLanguage: FBD
GlobalDB: DB3 (DB_HMI)
XMLFileLocation: C:\...\DB_HMI.xml
GlobalDB: DB4 (DB_ProgramLogic)
XMLFileLocation: C:\...\DB_ProgramLogic.xml
InstanceDB: iDB2 (Program Logic_DB)
XMLFileLocation: C:\...\Program Logic_DB.xml
InstanceDB: iDB7 (Valve Drain_DB)
XMLFileLocation: C:\...\Valve Drain_DB.xml
InstanceDB: iDB8 (Valve Liquid 1_DB)
XMLFileLocation: C:\...\Valve Liquid 1_DB.xml
InstanceDB: iDB9 (Valve Liquid 2_DB)
XMLFileLocation: C:\...\Valve Liquid 2_DB.xml
```

D. Appendix: Call structure

```
Main OB1
|--Valve Drain FB4 (Valve Drain_DB DB7)
|--|--fbValve_Solenoid FB110 (VlvDrn)
|--|--|--fcHMIBitEnable FC329
|--|--|--fcHMIBitEnable FC329
|--|--|--fcHMIBitEnable FC329
|--|--|--fbErrorScroller FB330 (ErrorScroller)
|--|--|--fcSetHMIStatusSimulation FC333
|--Valve Liquid 1 FB2 (Valve Liquid 1_DB DB8)
|--|--fbValve_Solenoid FB110 (VlvLiq1)
|--|--|--fcHMIBitEnable FC329
|--|--|--fcHMIBitEnable FC329
|--|--|--fcHMIBitEnable FC329
|--|--|--fbErrorScroller FB330 (ErrorScroller)
|--|--|--fcSetHMIStatusSimulation FC333
|--Valve Liquid 2 FB3 (Valve Liquid 2_DB DB9)
|--|--fbValve_Solenoid FB110 (VlvLiq2)
|--|--|--fcHMIBitEnable FC329
|--|--|--fcHMIBitEnable FC329
|--|--|--fcHMIBitEnable FC329
|--|--|--fbErrorScroller FB330 (ErrorScroller)
|--|--|--fcSetHMIStatusSimulation FC333
|--Program Logic FB5 (Program Logic_DB DB2)
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fcStepChooser FC185
|--|--fbStepSequencer FB185 (StepSeq)
```

E. APPENDIX: LOGIC MODELS PLC PROGRAM

```
OrganizationBlock: OB1 (Main) (FBD)
Network 1:
Call: FB4 (Valve Drain), Instance: Valve Drain_DB (GlobalVariable)
Inputs:
Outputs:
Network 2:
Call: FB2 (Valve Liquid 1), Instance: Valve Liquid 1_DB (GlobalVariable)
Inputs:
Outputs:
Network 3:
Call: FB3 (Valve Liquid 2), Instance: Valve Liquid 2_DB (GlobalVariable)
Inputs:
Outputs:
Network 4:
Call: FB5 (Program Logic), Instance: Program Logic_DB (GlobalVariable)
Inputs:
Outputs:

FunctionBlock: FB110 (fbValve_Solenoid) (FBD)
Network 2:
Call: FC329 (fcHMIBitEnable)
Inputs:
false ==> bInToggle
false ==> bInLatch
bManualMode ==> bInEnable
HMI_ValveControl.bPB_Home ==> bInOutHMI
bPB_Home ==> bInOutPLC
Outputs:
bOutEnable ==> HMI_ValveControl.bPBEN_Home
eno ==> en
Call: FC329 (fcHMIBitEnable)
Inputs:
eno ==> en
false ==> bInToggle
false ==> bInLatch
bManualMode ==> bInEnable
HMI_ValveControl.bPB_Work ==> bInOutHMI
bPB_Work ==> bInOutPLC
Outputs:
bOutEnable ==> HMI_ValveControl.bPBEN_Work
eno ==> en
Call: FC329 (fcHMIBitEnable)
Inputs:
eno ==> en
false ==> bInToggle
false ==> bInLatch
HMI_ValveControl.bError ==> bInEnable
HMI_ValveControl.bPB_ResetError ==> bInOutHMI
bPB_ResetError ==> bInOutPLC
Outputs:
bOutEnable ==> HMI_ValveControl.bPBEN_ResetError
Network 3:
HMI_ValveControl.bSignalHome = ((bInSignalHome && !bInSimulate) || (HMI_ValveControl
```

```
    .bHomeOn && bInSimulate))
Network 4:
HMI_ValveControl.bSignalWork = ((bInSignalWork && !bInSimulate) || (HMI_ValveControl
    .bWorkOn && bInSimulate))
Network 6:
bReset = ((bInResetError || bPB_ResetError) && HMI_ValveControl.bPBEN_ResetError)
ERROR_Valve.NoHomeFeedback = (if ((bInResetError || bPB_ResetError) &&
    HMI_ValveControl.bPBEN_ResetError) then FALSE)
ERROR_Valve.NoWorkFeedback = (if ((bInResetError || bPB_ResetError) &&
    HMI_ValveControl.bPBEN_ResetError) then FALSE)
ERROR_Valve.HomeFeedbackStillActive = (if ((bInResetError || bPB_ResetError) &&
    HMI_ValveControl.bPBEN_ResetError) then FALSE)
ERROR_Valve.WorkFeedbackStillActive = (if ((bInResetError || bPB_ResetError) &&
    HMI_ValveControl.bPBEN_ResetError) then FALSE)
Network 7:
HMI_ValveControl.iMode = (if (iInMode <> Modes.Independent) then iInMode)
Network 8:
bNewMode = (HMI_ValveControl.iMode <> iLastMode)
Network 9:
iLastMode = HMI_ValveControl.iMode
Network 10:
bManualMode = ((HMI_ValveControl.iMode == Modes.Manual) && (!bInEstop && !bNewMode
    && bInEnable))
bAutoMode = ((HMI_ValveControl.iMode == Modes.Auto) && (!bInEstop && !bNewMode &&
    bInEnable))
Network 11:
bEnableHome = (!HMI_ValveControl.bError && (bManualMode || bAutoMode))
bEnableWork = (!HMI_ValveControl.bError && (bManualMode || bAutoMode))
Network 12:
HMI_ValveControl.bHomeOn = (if ((bManualMode && bPB_Home && !bPB_Work &&
    HMI_ValveControl.bPBEN_Home) || (bAutoMode && !bInCommandWork && bEnableHome))
    then TRUE)
HMI_ValveControl.bWorkOn = (if ((bManualMode && bPB_Home && !bPB_Work &&
    HMI_ValveControl.bPBEN_Home) || (bAutoMode && !bInCommandWork && bEnableHome))
    then FALSE)
Network 13:
HMI_ValveControl.bWorkOn = (if ((bManualMode && bPB_Work && !bPB_Home &&
    HMI_ValveControl.bPBEN_Work) || (bAutoMode && bInCommandWork && bEnableWork))
    then TRUE)
HMI_ValveControl.bHomeOn = (if ((bManualMode && bPB_Work && !bPB_Home &&
    HMI_ValveControl.bPBEN_Work) || (bAutoMode && bInCommandWork && bEnableWork))
    then FALSE)
Network 14:
HMI_ValveControl.bHomeOn = (if (bAutoMode && !bEnableHome) then FALSE)
Network 15:
HMI_ValveControl.bWorkOn = (if (bAutoMode && !bEnableWork) then FALSE)
Network 16:
HMI_ValveControl.bHomeOn = (if (!bAutoMode && !bManualMode) then FALSE)
HMI_ValveControl.bWorkOn = (if (!bAutoMode && !bManualMode) then FALSE)
Network 18:
bTON_TimeOut = (TON: TON_TimeOut IN:(((HMI_ValveControl.bHomeOn && !HMI_ValveControl
    .bSignalHome) || (HMI_ValveControl.bWorkOn && !HMI_ValveControl.bSignalWork) || (
    HMI_ValveControl.bSignalHome && HMI_ValveControl.bWorkOn) || (HMI_ValveControl.
    bSignalWork && HMI_ValveControl.bHomeOn)) && !bReset) PT:tInTimeout)
Network 19:
ERROR_Valve.NoHomeFeedback = (if (!bInEstop && bTON_TimeOut && HMI_ValveControl.
```

```
    bHomeOn && !HMI_ValveControl.bSignalHome) then TRUE)
Network 20:
ERROR_Valve.NoWorkFeedback = (if (!bInEstop && bTON_TimeOut && HMI_ValveControl.
    bWorkOn && !HMI_ValveControl.bSignalWork) then TRUE)
Network 21:
ERROR_Valve.HomeFeedbackStillActive = (if (!bInEstop && bTON_TimeOut &&
    HMI_ValveControl.bWorkOn && HMI_ValveControl.bSignalHome) then TRUE)
Network 22:
ERROR_Valve.WorkFeedbackStillActive = (if (!bInEstop && bTON_TimeOut &&
    HMI_ValveControl.bHomeOn && HMI_ValveControl.bSignalWork) then TRUE)
Network 23:
Call: FB330 (fbErrorScroller), Instance: ErrorScroller (LocalVariable)
Inputs:
ERROR_Valve.NoHomeFeedback ==> bInError01
ERROR_Valve.NoWorkFeedback ==> bInError02
ERROR_Valve.HomeFeedbackStillActive ==> bInError03
ERROR_Valve.WorkFeedbackStillActive ==> bInError04
bInEstop ==> bInError05
Outputs:
bOutErrorExists ==> HMI_ValveControl.bError
iOutScrollingErrorNumber ==> HMI_ValveControl.iErrorCode
Network 24:
HMI_ValveControl.bInterlock != bInEnable
Network 25:
Call: FC333 (fcSetHMIStatusSimulation)
Inputs:
bInEstop ==> bInEstop
HMI_ValveControl.bError ==> bInError
HMI_ValveControl.bSignalWork ==> bInForwardWork
HMI_ValveControl.bSignalHome ==> bInReverseHome
HMI_ValveControl.bWorkOn ==> bInOnForwardWork
HMI_ValveControl.bHomeOn ==> bInOnReverseHome
bInSimulate ==> bInSimulate
Outputs:
iOutStatus ==> HMI_ValveControl.iStatus
Network 27:
bOutCommandHome = (HMI_ValveControl.bHomeOn && !bInSimulate)
Network 28:
bOutCommandWork = (HMI_ValveControl.bWorkOn && !bInSimulate)
Network 29:
bOutActiveHome = (HMI_ValveControl.bHomeOn && HMI_ValveControl.bSignalHome)
Network 30:
bOutActiveWork = (HMI_ValveControl.bWorkOn && HMI_ValveControl.bSignalWork)
Network 31:
bOutAuto = (HMI_ValveControl.iMode == Modes.Auto)
Network 32:
bOutError = HMI_ValveControl.bError
Network 33:

FunctionBlock: FB185 (fbStepSequencer) (SCL)

Function: FC185 (fcStepChooser) (SCL)

FunctionBlock: FB330 (fbErrorScroller) (FBD)
Network 1:
iErrorsScrollNum = (if ((iErrorsScrollNum < ciNoErrrors) || (iErrorsScrollNum >
```

```
    ciIndexHigh)) then ciNoErrrors)
143 iNextScrollNum = (if ((iNextScrollNum < ciIndexLow) || (iNextScrollNum > ciIndexHigh
    )) then ciIndexLow)
144 Network 2:
145 abErrors[1] = bInError01
146 abErrors[2] = bInError02
147 abErrors[3] = bInError03
148 abErrors[4] = bInError04
149 abErrors[5] = bInError05
150 abErrors[6] = bInError06
151 abErrors[7] = bInError07
152 abErrors[8] = bInError08
153 abErrors[9] = bInError09
154 abErrors[10] = bInError10
155 abErrors[11] = bInError11
156 abErrors[12] = bInError12
157 abErrors[13] = bInError13
158 abErrors[14] = bInError14
159 abErrors[15] = bInError15
160 abErrors[16] = bInError16
161 abErrors[17] = bInError17
162 abErrors[18] = bInError18
163 abErrors[19] = bInError19
164 abErrors[20] = bInError20
165 abErrors[21] = bInError21
166 abErrors[22] = bInError22
167 abErrors[23] = bInError23
168 abErrors[24] = bInError24
169 abErrors[25] = bInError25
170 abErrors[26] = bInError26
171 abErrors[27] = bInError27
172 abErrors[28] = bInError28
173 abErrors[29] = bInError29
174 abErrors[30] = bInError30
175 bErrorExists = (bInError01 || bInError02 || bInError03 || bInError04 || bInError05
    || bInError06 || bInError07 || bInError08 || bInError09 || bInError10 ||
    bInError11 || bInError12 || bInError13 || bInError14 || bInError15 || bInError16
    || bInError17 || bInError18 || bInError19 || bInError20 || bInError21 ||
    bInError22 || bInError23 || bInError24 || bInError25 || bInError26 || bInError27
    || bInError28 || bInError29 || bInError30)
176 Network 3:
177 bTON_ErrorDelay = (TON: TON_ErrorDelay IN:(bErrorExists && !bTON_ErrorDelay) PT:T#2S
    )
178 Network 4:
179 iNextScrollNum = (if bTON_ErrorDelay then iErrorsScrollNum)
180 iErrorsScrollNum = (if !bErrorExists then ciNoErrrors)
181 bScrolling = ((if ((bErrorExists && (iErrorsScrollNum == ciNoErrrors)) ||
    bTON_ErrorDelay) then TRUE) (elseif !bErrorExists then FALSE))
182 Network 5:
183 (iNextScrollNum = (if bScrolling then 1 + iNextScrollNum))
184 iNextScrollNum = (if (((iNextScrollNum > ciIndexHigh) || (iNextScrollNum <
    ciIndexLow)) && bScrolling) then ciIndexLow)
185 Network 6:
186 iErrorsScrollNum = (if (bScrolling && abErrors[iNextScrollNum]) then iNextScrollNum)
187 bScrolling = (if (bScrolling && abErrors[iNextScrollNum]) then FALSE)
188 Network 7:
```

```
iOutScrollingErrorNumber = iErrorsScrollNum
bOutErrorExists = bErrorExists
Network 8:

Function: FC329 (fcHMIBitEnable) (FBD)
Network 1:
bInOutPLC = (if (!bInToggle && !bInLatch) then FALSE)
Network 2:
bInOutPLC = (if (bInEnable && bInOutHMI && bInToggle && bInOutPLC) then FALSE)
bInOutHMI = (if (bInEnable && bInOutHMI && bInToggle && bInOutPLC) then FALSE)
Network 3:
bInOutPLC = (if (bInEnable && bInOutHMI && bInToggle && !bInOutPLC) then TRUE)
Network 4:
bInOutPLC = (if (bInEnable && bInOutHMI && !bInToggle) then TRUE)
Network 5:
bOutEnable = bInEnable
Network 6:
bInOutHMI = FALSE
Network 7:

Function: FC333 (fcSetHMIStatusSimulation) (FBD)
Network 1:
iOutStatus = (if (bInSimulate && bInOnReverseHome) then HMI.Status.
    SimulatedOnReverseHome)
iOutStatus = (if (!bInSimulate && bInOnReverseHome) then HMI.Status.OnReverseHome)
Network 2:
iOutStatus = (if (bInSimulate && bInOnForwardWork) then HMI.Status.
    SimulatedOnForwardWork)
iOutStatus = (if (!bInSimulate && bInOnForwardWork) then HMI.Status.OnForwardWork)
Network 3:
iOutStatus = (if (bInSimulate && bInReverseHome) then HMI.Status.
    SimulatedReverseHome)
iOutStatus = (if (!bInSimulate && bInReverseHome) then HMI.Status.ReverseHome)
Network 4:
iOutStatus = (if (bInSimulate && bInForwardWork) then HMI.Status.
    SimulatedForwardWork)
iOutStatus = (if (!bInSimulate && bInForwardWork) then HMI.Status.ForwardWork)
Network 5:
iOutStatus = (if bInError then HMI.Status.Error)
Network 6:
iOutStatus = (if bInEstop then HMI.Status.Estop)
Network 7:
iOutStatus = (if (!bInEstop && !bInError && !bInForwardWork && !bInReverseHome && !
    bInOnForwardWork && !bInOnReverseHome) then HMI.Status.Stopped)
Network 8:

FunctionBlock: FB5 (Program Logic) (FBD)
Network 1:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 0) ==> en
Ib_PBStart ==> bCondition
10 ==> iTrueStep
0 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
```

```
eno ==> StepDone = TRUE
Network 3:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 10) ==> en
(Ib_LvlContEmpty && !Ib_LvlContFull) ==> bCondition
20 ==> iTrueStep
11 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 4:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 11) ==> en
(Ib_LvlContEmpty && !Ib_LvlContFull) ==> bCondition
20 ==> iTrueStep
11 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 6:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 20) ==> en
Ib_PBFillLiq1 ==> bCondition
21 ==> iTrueStep
20 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 7:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 21) ==> en
Ib_LvlContFull ==> bCondition
30 ==> iTrueStep
21 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 9:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 30) ==> en
Ib_PBEmptyCont ==> bCondition
31 ==> iTrueStep
30 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 10:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 31) ==> en
Ib_LvlContEmpty ==> bCondition
```

```
40 ==> iTrueStep
31 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 12:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 40) ==> en
Ib_PBFillLiq2 ==> bCondition
41 ==> iTrueStep
40 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 13:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 41) ==> en
Ib_LvlContFull ==> bCondition
50 ==> iTrueStep
41 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 15:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 50) ==> en
Ib_PBEmptyCont ==> bCondition
51 ==> iTrueStep
50 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 16:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 51) ==> en
Ib_LvlContEmpty ==> bCondition
60 ==> iTrueStep
51 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 18:
Call: FC185 (fcStepChooser)
Inputs:
(CurrentStep == 60) ==> en
Ib_PBStop ==> bCondition
0 ==> iTrueStep
60 ==> iFalseStep
Outputs:
Ret_Val ==> NextStep
eno ==> StepDone = TRUE
Network 20:
```

```
Call: FB185 (fbStepSequencer), Instance: StepSeq (LocalVariable)
Inputs:
CurrentStep ==> iInStep
NextStep ==> iInNextStep
StepDone ==> bInStepDone
Outputs:
iOutCurrentStep ==> CurrentStep
eno ==> StepDone = FALSE
Network 22:
DB_ProgramLogic.VlvDrn_Open = ((CurrentStep == 11) || (CurrentStep == 31) || (
    CurrentStep == 51))
DB_ProgramLogic.VlvLiq1_Open = (CurrentStep == 21)
DB_ProgramLogic.VlvLiq2_Open = (CurrentStep == 41)

FunctionBlock: FB4 (Valve Drain) (FBD)
Network 1:
Call: FB110 (fbValve_Solenoid), Instance: VlvDrn (LocalVariable)
Inputs:
t#2s ==> tInTimeout
1 ==> iInMode
Ib_VlvDrnClosed ==> bInSignalHome
Ib_VlvDrnOpened ==> bInSignalWork
true ==> bInEnable
DB_ProgramLogic.VlvDrn_Open ==> bInCommandWork
Ib_PBReset ==> bInResetError
DB_HMI.VlvDrn ==> HMI_ValveControl
Outputs:
bOutCommandWork ==> Qb_VlvDrnOpen

FunctionBlock: FB2 (Valve Liquid 1) (FBD)
Network 1:
Call: FB110 (fbValve_Solenoid), Instance: VlvLiq1 (LocalVariable)
Inputs:
t#2s ==> tInTimeout
1 ==> iInMode
Ib_VlvLiq1Closed ==> bInSignalHome
Ib_VlvLiq1Opened ==> bInSignalWork
true ==> bInEnable
DB_ProgramLogic.VlvLiq1_Open ==> bInCommandWork
Ib_PBReset ==> bInResetError
DB_HMI.VlvLiq1 ==> HMI_ValveControl
Outputs:
bOutCommandWork ==> Qb_VlvLiq1Open

FunctionBlock: FB3 (Valve Liquid 2) (FBD)
Network 1:
Call: FB110 (fbValve_Solenoid), Instance: VlvLiq2 (LocalVariable)
Inputs:
t#2s ==> tInTimeout
1 ==> iInMode
Ib_VlvLiq2Closed ==> bInSignalHome
Ib_VlvLiq2Opened ==> bInSignalWork
true ==> bInEnable
DB_ProgramLogic.VlvLiq2_Open ==> bInCommandWork
Ib_PBReset ==> bInResetError
DB_HMI.VlvLiq2 ==> HMI_ValveControl
```

```
Outputs:
bOutCommandWork ==> Qb_VlvLiq2Open
```

F. Appendix: NuXmv models PLC program

FB110:

```
MODULE fb110(iInMode, bInSignalHome, bInSignalWork, bInEnable, bInCommandWork,
    bInResetError, bInSimulate, bInEstop)
VAR
global : global;
loc : {start, nw1, nw2, nw3, nw4, nw5, nw6, nw7 ,nw8, nw9, nw10, nw11, nw12, nw13,
    nw14, nw15, nw16, nw17, nw18, nw19, nw20, nw21, nw22, nw23, nw24, nw25, nw26,
    nw27, nw28, nw29, nw30, nw31, nw32, nw33, end};

output_bOutCommandWork : boolean;
output_bOutCommandHome : boolean;
output_bOutActiveHome : boolean;
output_bOutActiveWork : boolean;
output_bOutAuto : boolean;
output_bOutError : boolean;

output_ERROR_Valve_NoHomeFeedback : boolean;
output_ERROR_Valve_NoWorkFeedback : boolean;
output_ERROR_Valve_HomeFeedbackStillActive : boolean;
output_ERROR_Valve_WorkFeedbackStillActive : boolean;

inout_HMI_ValveControl_bPB_Home : boolean;
inout_HMI_ValveControl_bPBEN_Home : boolean;
inout_HMI_ValveControl_bPB_Work : boolean;
inout_HMI_ValveControl_bPBEN_Work : boolean;
inout_HMI_ValveControl_bError : boolean;
inout_HMI_ValveControl_bPB_ResetError : boolean;
inout_HMI_ValveControl_bPBEN_ResetError : boolean;
inout_HMI_ValveControl_bSignalHome : boolean;
inout_HMI_ValveControl_bSignalWork : boolean;
inout_HMI_ValveControl_bInterlock : boolean;
inout_HMI_ValveControl_bHomeOn : boolean;
inout_HMI_ValveControl_bWorkOn : boolean;
inout_HMI_ValveControl_iMode : signed word[16];
inout_HMI_ValveControl_iStatus : signed word[16];

static_bManualMode : boolean;
static_bAutoMode : boolean;
static_bPB_Home : boolean;
static_bPB_Work : boolean;
static_bEnableHome : boolean;
static_bEnableWork : boolean;
static_bTON_TimeOut : boolean;
static_bPB_ResetError : boolean;
static_bReset : boolean;
static_bNewMode : boolean;
static_iLastMode : signed word[16];

timer_TON_TimeOut : {0,1,2};

randomModeNr : {0,1,2,3,4,5,6,7,8,9,10};

ASSIGN
```

```
50 init(loc) := start;
51
52 init(output_bOutCommandWork) := FALSE;
53 init(output_bOutCommandHome) := FALSE;
54 init(output_bOutActiveHome) := FALSE;
55 init(output_bOutActiveWork) := FALSE;
56 init(output_bOutAuto) := FALSE;
57 init(output_bOutError) := FALSE;
58
59 init(output_ERROR_Valve_NoHomeFeedback) := FALSE;
60 init(output_ERROR_Valve_NoWorkFeedback) := FALSE;
61 init(output_ERROR_Valve_HomeFeedbackStillActive) := FALSE;
62 init(output_ERROR_Valve_WorkFeedbackStillActive) := FALSE;
63
64 init(inout_HMI_ValveControl_bPB_Home) := FALSE;
65 init(inout_HMI_ValveControl_bPBEN_Home) := FALSE;
66 init(inout_HMI_ValveControl_bPB_Work) := FALSE;
67 init(inout_HMI_ValveControl_bPBEN_Work) := FALSE;
68 init(inout_HMI_ValveControl_bError) := FALSE;
69 init(inout_HMI_ValveControl_bPB_ResetError) := FALSE;
70 init(inout_HMI_ValveControl_bPBEN_ResetError) := FALSE;
71 init(inout_HMI_ValveControl_bSignalHome) := FALSE;
72 init(inout_HMI_ValveControl_bSignalWork) := FALSE;
73 init(inout_HMI_ValveControl_bInterlock) := FALSE;
74 init(inout_HMI_ValveControl_bHomeOn) := FALSE;
75 init(inout_HMI_ValveControl_bWorkOn) := FALSE;
76 init(inout_HMI_ValveControl_iMode) := 0sd16_0;
77 init(inout_HMI_ValveControl_iStatus) := 0sd16_0;
78
79 init(static_bManualMode) := FALSE;
80 init(static_bAutoMode) := FALSE;
81 init(static_bPB_Home) := FALSE;
82 init(static_bPB_Work) := FALSE;
83 init(static_bEnableHome) := FALSE;
84 init(static_bEnableWork) := FALSE;
85 init(static_bTON_TimeOut) := FALSE;
86 init(static_bPB_ResetError) := FALSE;
87 init(static_bReset) := FALSE;
88 init(static_bNewMode) := FALSE;
89 init(static_iLastMode) := 0sd16_0;
90
91 next(loc) :=
92   case
93     (loc = start) : nw1;
94     (loc = nw1) : nw2;
95     (loc = nw2) : nw3;
96     (loc = nw3) : nw4;
97     (loc = nw4) : nw5;
98     (loc = nw5) : nw6;
99     (loc = nw6) : nw7;
100    (loc = nw7) : nw8;
101    (loc = nw8) : nw9;
102    (loc = nw9) : nw10;
103    (loc = nw10) : nw11;
104    (loc = nw11) : nw12;
105    (loc = nw12) : nw13;
```

```
    (loc = nw13) : nw14;
    (loc = nw14) : nw15;
    (loc = nw15) : nw16;
    (loc = nw16) : nw17;
    (loc = nw17) : nw18;
    (loc = nw18) : nw19;
    (loc = nw19) : nw20;
    (loc = nw20) : nw21;
    (loc = nw21) : nw22;
    (loc = nw22) : nw23;
    (loc = nw23) : nw24;
    (loc = nw24) : nw25;
    (loc = nw25) : nw26;
    (loc = nw26) : nw27;
    (loc = nw27) : nw28;
    (loc = nw28) : nw29;
    (loc = nw29) : nw30;
    (loc = nw30) : nw31;
    (loc = nw31) : nw32;
    (loc = nw32) : nw33;
    (loc = nw33) : end;
    (loc = end) : start;
  esac;


next(inout_HMI_ValveControl_bPBEN_Home) :=
  case
    (loc = nw2) : {FALSE, TRUE};
    TRUE : inout_HMI_ValveControl_bPBEN_Home;
  esac;
next(inout_HMI_ValveControl_bPB_Home) :=
  case
    (loc = nw2) : {FALSE, TRUE};
    TRUE : inout_HMI_ValveControl_bPB_Home;
  esac;
next(static_bPB_Home) :=
  case
    (loc = nw2) : {FALSE, TRUE};
    TRUE : static_bPB_Home;
  esac;

next(inout_HMI_ValveControl_bPBEN_Work) :=
  case
    (loc = nw2) : {FALSE, TRUE};
    TRUE : inout_HMI_ValveControl_bPBEN_Work;
  esac;
next(inout_HMI_ValveControl_bPB_Work) :=
  case
    (loc = nw2) : {FALSE, TRUE};
    TRUE : inout_HMI_ValveControl_bPB_Work;
  esac;
next(static_bPB_Work) :=
  case
    (loc = nw2) : {FALSE, TRUE};
    TRUE : static_bPB_Work;
  esac;
```

```
162
163 next(inout_HMI_ValveControl_bPBEN_ResetError) :=
164   case
165     (loc = nw2) : {FALSE, TRUE};
166     TRUE : inout_HMI_ValveControl_bPBEN_ResetError;
167   esac;
168 next(inout_HMI_ValveControl_bPB_ResetError) :=
169   case
170     (loc = nw2) : {FALSE, TRUE};
171     TRUE : inout_HMI_ValveControl_bPB_ResetError;
172   esac;
173 next(static_bPB_ResetError) :=
174   case
175     (loc = nw2) : {FALSE, TRUE};
176     TRUE : static_bPB_ResetError;
177   esac;
178
179
180 next(inout_HMI_ValveControl_bSignalHome) :=
181   case
182     (loc = nw3) : (bInSignalHome & !bInSimulate) | (inout_HMI_ValveControl_bHomeOn &
       bInSimulate);
183     TRUE : inout_HMI_ValveControl_bSignalHome;
184   esac;
185
186 next(inout_HMI_ValveControl_bSignalWork) :=
187   case
188     (loc = nw4) : (bInSignalWork & !bInSimulate) | (inout_HMI_ValveControl_bWorkOn &
       bInSimulate);
189     TRUE : inout_HMI_ValveControl_bSignalWork;
190   esac;
191
192 next(output_ERROR_Valve_NoHomeFeedback) :=
193   case
194     (loc = nw6 & (bInResetError | static_bPB_ResetError) &
      inout_HMI_ValveControl_bPBEN_ResetError) : FALSE;
195     (loc = nw19) & (!bInEstop & static_bTON_TimeOut & inout_HMI_ValveControl_bHomeOn
       & !inout_HMI_ValveControl_bSignalHome) : TRUE;
196     TRUE : output_ERROR_Valve_NoHomeFeedback;
197   esac;
198
199 next(output_ERROR_Valve_NoWorkFeedback) :=
200   case
201     (loc = nw6 & (bInResetError | static_bPB_ResetError) &
      inout_HMI_ValveControl_bPBEN_ResetError) : FALSE;
202     (loc = nw20) & (!bInEstop & static_bTON_TimeOut & inout_HMI_ValveControl_bWorkOn
       & !inout_HMI_ValveControl_bSignalWork) : TRUE;
203     TRUE : output_ERROR_Valve_NoWorkFeedback;
204   esac;
205
206 next(output_ERROR_Valve_HomeFeedbackStillActive) :=
207   case
208     (loc = nw6 & (bInResetError | static_bPB_ResetError) &
      inout_HMI_ValveControl_bPBEN_ResetError) : FALSE;
209     (loc = nw21) & (!bInEstop & static_bTON_TimeOut & inout_HMI_ValveControl_bWorkOn
       & inout_HMI_ValveControl_bSignalHome) : TRUE;
```

```
210     TRUE : output_ERROR_Valve_HomeFeedbackStillActive;
211   esac;
212
213 next(output_ERROR_Valve_WorkFeedbackStillActive) :=
214   case
215     (loc = nw6 & (bInResetError | static_bPB_ResetError) &
      inout_HMI_ValveControl_bPBEN_ResetError) : FALSE;
216     (loc = nw22) & (!bInEstop & static_bTON_TimeOut & inout_HMI_ValveControl_bHomeOn
       & inout_HMI_ValveControl_bSignalWork) : TRUE;
217     TRUE : output_ERROR_Valve_WorkFeedbackStillActive;
218   esac;
219
220 next(static_bReset) :=
221   case
222     (loc = nw6) : ((bInResetError | static_bPB_ResetError) &
      inout_HMI_ValveControl_bPBEN_ResetError);
223     TRUE : static_bReset;
224   esac;
225
226 next(inout_HMI_ValveControl_iMode) :=
227   case
228     (loc = nw7 & iInMode != global.Modes_Independent) : iInMode;
229     TRUE : inout_HMI_ValveControl_iMode;
230   esac;
231
232 next(static_bNewMode) :=
233   case
234     (loc = nw8) : inout_HMI_ValveControl_iMode != static_iLastMode;
235     TRUE : static_bNewMode;
236   esac;
237
238 next(static_iLastMode) :=
239   case
240     (loc = nw9) : inout_HMI_ValveControl_iMode;
241     TRUE : static_iLastMode;
242   esac;
243
244 next(static_bManualMode) :=
245   case
246     (loc = nw10) : inout_HMI_ValveControl_iMode = global.Modes_Manual & (!bInEstop &
       !static_bNewMode & bInEnable);
247     TRUE: static_bManualMode;
248   esac;
249
250 next(static_bAutoMode) :=
251   case
252     (loc = nw10) : inout_HMI_ValveControl_iMode = global.Modes_Auto & (!bInEstop & !
      static_bNewMode & bInEnable);
253     TRUE: static_bAutoMode;
254   esac;
255
256 next(static_bEnableHome) :=
257   case
258     (loc = nw11) : !inout_HMI_ValveControl_bError & (static_bManualMode |
      static_bAutoMode);
259     TRUE: static_bEnableHome;
```

```
260   esac;
261
262 next(static_bEnableWork) :=
263   case
264     (loc = nw11) : !inout_HMI_ValveControl_bError & (static_bManualMode |
      static_bAutoMode);
265     TRUE: static_bEnableWork;
266   esac;
267
268 next(inout_HMI_ValveControl_bHomeOn) :=
269   case
270     (loc = nw12) & (static_bManualMode & static_bPB_Home & !static_bPB_Work &
      inout_HMI_ValveControl_bPBEN_Home) | (static_bAutoMode & !bInCommandWork &
      static_bEnableHome) : TRUE;
271     (loc = nw13) & ((static_bManualMode & static_bPB_Work & !static_bPB_Home &
      inout_HMI_ValveControl_bPBEN_Work) | (static_bAutoMode & bInCommandWork &
      static_bEnableWork)) : FALSE;
272     (loc = nw14) & (static_bAutoMode & !static_bEnableHome): FALSE;
273     (loc = nw16) & (!static_bAutoMode & !static_bManualMode): FALSE;
274     TRUE: inout_HMI_ValveControl_bHomeOn;
275   esac;
276
277 next(inout_HMI_ValveControl_bWorkOn) :=
278   case
279     (loc = nw12) & (static_bManualMode & static_bPB_Home & !static_bPB_Work &
      inout_HMI_ValveControl_bPBEN_Home) | (static_bAutoMode & !bInCommandWork &
      static_bEnableHome) : FALSE;
280     (loc = nw13) & ((static_bManualMode & static_bPB_Work & !static_bPB_Home &
      inout_HMI_ValveControl_bPBEN_Work) | (static_bAutoMode & bInCommandWork &
      static_bEnableWork)) : TRUE;
281     (loc = nw15) & (static_bAutoMode & !static_bEnableWork): FALSE;
282     (loc = nw16) & (!static_bAutoMode & !static_bManualMode): FALSE;
283     TRUE: inout_HMI_ValveControl_bWorkOn;
284   esac;
285
286 next(timer_TON_TimeOut) :=
287   case
288     (loc = nw18) & !(((inout_HMI_ValveControl_bHomeOn & !
      inout_HMI_ValveControl_bSignalHome) | (inout_HMI_ValveControl_bWorkOn & !
      inout_HMI_ValveControl_bSignalWork) | (inout_HMI_ValveControl_bSignalHome &
      inout_HMI_ValveControl_bWorkOn) | (inout_HMI_ValveControl_bSignalWork &
      inout_HMI_ValveControl_bHomeOn)) & !static_bReset) : 0;
289     (loc = nw18) & (((inout_HMI_ValveControl_bHomeOn & !
      inout_HMI_ValveControl_bSignalHome) | (inout_HMI_ValveControl_bWorkOn & !
      inout_HMI_ValveControl_bSignalWork) | (inout_HMI_ValveControl_bSignalHome &
      inout_HMI_ValveControl_bWorkOn) | (inout_HMI_ValveControl_bSignalWork &
      inout_HMI_ValveControl_bHomeOn)) & !static_bReset) & (timer_TON_TimeOut < 2) :
      timer_TON_TimeOut + 1;
290     TRUE : timer_TON_TimeOut;
291   esac;
292
293 next(static_bTON_TimeOut) :=
294   case
295     (loc = nw18) & (timer_TON_TimeOut < 2) : FALSE;
296     (loc = nw18) & (timer_TON_TimeOut >= 2) : TRUE;
297     TRUE : static_bTON_TimeOut;
```

```
298   esac;
299
300 next(inout_HMI_ValveControl_bError) :=
301   case
302     (loc = nw23) : {FALSE, TRUE};
303     TRUE : inout_HMI_ValveControl_bError;
304   esac;
305
306 next(inout_HMI_ValveControl_bInterlock) :=
307   case
308     (loc = nw24) : !bInEnable;
309     TRUE: inout_HMI_ValveControl_bInterlock;
310   esac;
311
312 next(inout_HMI_ValveControl_iStatus) :=
313   case
314     (loc = nw25 & randomModeNr = 0) : swconst(0, 16);
315     (loc = nw25 & randomModeNr = 1) : swconst(1, 16);
316     (loc = nw25 & randomModeNr = 2) : swconst(2, 16);
317     (loc = nw25 & randomModeNr = 3) : swconst(3, 16);
318     (loc = nw25 & randomModeNr = 4) : swconst(4, 16);
319     (loc = nw25 & randomModeNr = 5) : swconst(5, 16);
320     (loc = nw25 & randomModeNr = 6) : swconst(6, 16);
321     (loc = nw25 & randomModeNr = 7) : swconst(7, 16);
322     (loc = nw25 & randomModeNr = 8) : swconst(8, 16);
323     (loc = nw25 & randomModeNr = 9) : swconst(9, 16);
324     (loc = nw25 & randomModeNr = 10) : swconst(10, 16);
325     TRUE : inout_HMI_ValveControl_iStatus;
326   esac;
327
328 next(output_bOutCommandHome) :=
329   case
330     (loc = nw27) : (inout_HMI_ValveControl_bHomeOn & !bInSimulate);
331     TRUE: output_bOutCommandHome;
332   esac;
333
334 next(output_bOutCommandWork) :=
335   case
336     (loc = nw28) : (inout_HMI_ValveControl_bWorkOn & !bInSimulate);
337     TRUE: output_bOutCommandWork;
338   esac;
339
340 next(output_bOutActiveHome) :=
341   case
342     (loc = nw29) : (inout_HMI_ValveControl_bHomeOn &
      inout_HMI_ValveControl_bSignalHome);
343     TRUE: output_bOutActiveHome;
344   esac;
345
346 next(output_bOutActiveWork) :=
347   case
348     (loc = nw30) : (inout_HMI_ValveControl_bWorkOn &
      inout_HMI_ValveControl_bSignalWork);
349     TRUE: output_bOutActiveWork;
350   esac;
351
```

```
next(output_bOutAuto) :=
  case
    (loc = nw31) : (inout_HMI_ValveControl_iMode = global.Modes_Auto);
    TRUE: output_bOutAuto;
  esac;

next(output_bOutError) :=
  case
    (loc = nw32) : (inout_HMI_ValveControl_bError);
    TRUE: output_bOutError;
  esac;


MODULE global
DEFINE
Modes_Independent := 0sd16_10;
Modes_Manual := 0sd16_2;
Modes_Auto := 0sd16_1;


MODULE main
VAR
iInMode : signed word[16];
bInSignalHome : boolean;
bInSignalWork : boolean;
bInEnable : boolean;
bInCommandWork : boolean;
bInResetError : boolean;
bInSimulate : boolean;
bInEstop : boolean;
TestVlv : fb110(iInMode, bInSignalHome, bInSignalWork, bInEnable, bInCommandWork,
    bInResetError, bInSimulate, bInEstop);
randomModeNr : {0,1,2,10};

ASSIGN
init(iInMode) := 0sd16_1;
init(bInSignalHome) := FALSE;
init(bInSignalWork) := FALSE;
init(bInEnable) := FALSE;
init(bInCommandWork) := FALSE;
init(bInResetError) := FALSE;
init(bInSimulate) := FALSE;
init(bInEstop) := FALSE;

next(iInMode) :=
  case
    (TestVlv.loc = end & randomModeNr = 0) : swconst(0, 16);
    (TestVlv.loc = end & randomModeNr = 1) : swconst(1, 16);
    (TestVlv.loc = end & randomModeNr = 2) : swconst(2, 16);
    (TestVlv.loc = end & randomModeNr = 10) : swconst(10, 16);
    TRUE : iInMode;
  esac;

next(bInSignalHome) :=
  case
    (TestVlv.loc = end) : {FALSE, TRUE};
```

```
    TRUE : bInSignalHome;
  esac;

next(bInSignalWork) :=
  case
    (TestVlv.loc = end) : {FALSE, TRUE};
    TRUE : bInSignalWork;
  esac;

next(bInEnable) :=
  case
    (TestVlv.loc = end) : {FALSE, TRUE};
    TRUE : bInEnable;
  esac;

next(bInCommandWork) :=
  case
    (TestVlv.loc = end) : {FALSE, TRUE};
    TRUE : bInCommandWork;
  esac;

next(bInResetError) :=
  case
    (TestVlv.loc = end) : {FALSE, TRUE};
    TRUE : bInResetError;
  esac;

next(bInSimulate) :=
  case
    (TestVlv.loc = end) : {FALSE, TRUE};
    TRUE : bInSimulate;
  esac;

next(bInEstop) :=
  case
    (TestVlv.loc = end) : {FALSE, TRUE};
    TRUE : bInEstop;
  esac;
```

FB330:

```
MODULE fb330(bInError01, bInError02, bInError03, bInError04, bInError05)
VAR
loc : {start, nw1, nw2, nw3, nw4, nw5, nw6, nw7 ,nw8, end};

output_bOutErrorExists : boolean;
output_iOutScrollingErrorNumber : signed word[16];

static_abErrors : array 0..5 of boolean;
static_iNextScrollNum : signed word[16];
static_iErrorsScrollNum : signed word[16];
static_bScrolling : boolean;
static_bTON_ErrorDelay : boolean;

temp_bErrorExists : boolean;

```

```
const_ciNoErrors : signed word[16];
const_ciIndexLow : signed word[16];
const_ciIndexHigh : signed word[16];

timer_TON_ErrorDelay : {0,1,2};


ASSIGN
init(loc) := start;

init(output_bOutErrorExists) := FALSE;
init(output_iOutScrollingErrorNumber) := 0sd16_0;

init(static_iNextScrollNum) := 0sd16_0;
init(static_iErrorsScrollNum) := 0sd16_0;
init(static_bScrolling) := FALSE;
init(static_bTON_ErrorDelay) := FALSE;

init(temp_bErrorExists) := FALSE;

init(const_ciNoErrors) := 0sd16_0;
init(const_ciIndexLow) := 0sd16_1;
init(const_ciIndexHigh) := 0sd16_5;

init(timer_TON_ErrorDelay) := 0;

next(const_ciNoErrors) := 0sd16_0;
next(const_ciIndexLow) := 0sd16_1;
next(const_ciIndexHigh) := 0sd16_5;

next(loc) :=
  case
    (loc = start) : nw1;
    (loc = nw1) : nw2;
    (loc = nw2) : nw3;
    (loc = nw3) : nw4;
    (loc = nw4) : nw5;
    (loc = nw5) : nw6;
    (loc = nw6) : nw7;
    (loc = nw7) : nw8;
    (loc = nw8) : end;
    (loc = end) : start;
  esac;

-- network 1
next(static_iErrorsScrollNum) :=
  case
    (loc = nw1) & ((static_iErrorsScrollNum < const_ciNoErrors) | (
    static_iErrorsScrollNum > const_ciIndexHigh)): const_ciNoErrors;
    (loc = nw4) & !temp_bErrorExists : const_ciNoErrors;
    (loc = nw6) & (static_bScrolling & static_abErrors[1] & static_iNextScrollNum =
    0sd16_1) : 0sd16_1;
    (loc = nw6) & (static_bScrolling & static_abErrors[2] & static_iNextScrollNum =
    0sd16_2) : 0sd16_2;
    (loc = nw6) & (static_bScrolling & static_abErrors[3] & static_iNextScrollNum =
    0sd16_3) : 0sd16_3;
```

```
68      (loc = nw6) & (static_bScrolling & static_abErrors[4] & static_iNextScrollNum =
        0sd16_4) : 0sd16_4;
69      (loc = nw6) & (static_bScrolling & static_abErrors[5] & static_iNextScrollNum =
        0sd16_5) : 0sd16_5;
70      TRUE : static_iErrorsScrollNum;
71    esac;
72  next(static_iNextScrollNum) :=
73    case
74      (loc = nw1) & ((static_iNextScrollNum < const_ciIndexLow) | (
        static_iNextScrollNum > const_ciIndexHigh)): const_ciIndexLow;
75      (loc = nw4) & static_bTON_ErrorDelay : static_iErrorsScrollNum;
76      (loc = nw5) & static_bScrolling : static_iNextScrollNum + 0sd16_1;
77      (loc = nw5) & ((static_iNextScrollNum > const_ciIndexHigh) | (
        static_iNextScrollNum < const_ciIndexLow)) &static_bScrolling : const_ciIndexLow;
78      TRUE : static_iNextScrollNum;
79    esac;
80
81  -- network 2
82  next(static_abErrors[1]) :=
83    case
84      (loc = nw2) : bInError01;
85      TRUE : static_abErrors[1];
86    esac;
87
88  next(static_abErrors[2]) :=
89    case
90      (loc = nw2) : bInError02;
91      TRUE : static_abErrors[2];
92    esac;
93
94  next(static_abErrors[3]) :=
95    case
96      (loc = nw2) : bInError03;
97      TRUE : static_abErrors[3];
98    esac;
99
100 next(static_abErrors[4]) :=
101   case
102     (loc = nw2) : bInError04;
103     TRUE : static_abErrors[4];
104   esac;
105
106 next(static_abErrors[5]) :=
107   case
108     (loc = nw2) : bInError05;
109     TRUE : static_abErrors[5];
110   esac;
111
112 next(temp_bErrorExists) :=
113   case
114     (loc = nw2) : bInError01 | bInError02 | bInError03 | bInError04 | bInError05;
115     TRUE : temp_bErrorExists;
116   esac;
117
118 -- network 3
119 next(timer_TON_ErrorDelay) :=
```

```
  case
    (loc = nw3) & !(temp_bErrorExists & !static_bTON_ErrorDelay) : 0;
    (loc = nw3) & (temp_bErrorExists & !static_bTON_ErrorDelay) & (
    timer_TON_ErrorDelay < 2) : timer_TON_ErrorDelay + 1;
    TRUE : timer_TON_ErrorDelay;
  esac;

next(static_bTON_ErrorDelay) :=
  case
    (loc = nw3) & (timer_TON_ErrorDelay < 2) : FALSE;
    (loc = nw3) & (timer_TON_ErrorDelay >= 2) : TRUE;
    TRUE : static_bTON_ErrorDelay;
  esac;

-- network 4
next(static_bScrolling) :=
  case
    (loc = nw4) & ( static_bTON_ErrorDelay | ( temp_bErrorExists & (
    static_iErrorsScrollNum = const_ciNoErrors))) : TRUE;
    (loc = nw4) & ( !temp_bErrorExists) : FALSE;
    (loc = nw6) & (static_bScrolling & static_abErrors[1] & static_iNextScrollNum =
    0sd16_1) : FALSE;
    (loc = nw6) & (static_bScrolling & static_abErrors[2] & static_iNextScrollNum =
    0sd16_2) : FALSE;
    (loc = nw6) & (static_bScrolling & static_abErrors[3] & static_iNextScrollNum =
    0sd16_3) : FALSE;
    (loc = nw6) & (static_bScrolling & static_abErrors[4] & static_iNextScrollNum =
    0sd16_4) : FALSE;
    (loc = nw6) & (static_bScrolling & static_abErrors[5] & static_iNextScrollNum =
    0sd16_5) : FALSE;
    TRUE : static_bScrolling;
  esac;

-- network 7
next(output_bOutErrorExists) :=
  case
    (loc = nw7) : temp_bErrorExists;
    TRUE : output_bOutErrorExists;
  esac;

next(output_iOutScrollingErrorNumber) :=
  case
    (loc = nw7) : static_iErrorsScrollNum;
    TRUE : output_iOutScrollingErrorNumber;
  esac;


MODULE main
VAR
bInError01 : boolean;
bInError02 : boolean;
bInError03 : boolean;
bInError04 : boolean;
bInError05 : boolean;
Test_FB330 : fb330(bInError01, bInError02, bInError03, bInError04, bInError05);

```

```
ASSIGN
init(bInError01) := FALSE;
init(bInError02) := FALSE;
init(bInError03) := FALSE;
init(bInError04) := FALSE;
init(bInError05) := FALSE;


next(bInError01) :=
  case
    (Test_FB330.loc = end) : {FALSE, TRUE};
    TRUE : bInError01;
  esac;

next(bInError02) :=
  case
    (Test_FB330.loc = end) : {FALSE, TRUE};
    TRUE : bInError02;
  esac;

next(bInError03) :=
  case
    (Test_FB330.loc = end) : {FALSE, TRUE};
    TRUE : bInError03;
  esac;

next(bInError04) :=
  case
    (Test_FB330.loc = end) : {FALSE, TRUE};
    TRUE : bInError04;
  esac;

next(bInError05) :=
  case
    (Test_FB330.loc = end) : {FALSE, TRUE};
    TRUE : bInError05;
  esac;
```

FC329:

```
MODULE fc329(bInToggle, bInLatch, bInEnable)
VAR
loc : {start, nw1, nw2, nw3, nw4, nw5, nw6, nw7 , end};

output_bOutEnable : boolean;

inout_bInOutHMI : boolean;
inout_bInOutPLC : boolean;


ASSIGN
init(loc) := start;

init(output_bOutEnable) := FALSE;

init(inout_bInOutHMI) := FALSE;
```

```
17 init(inout_bInOutPLC) := FALSE;
18
19 next(loc) :=
20   case
21     (loc = start) : nw1;
22     (loc = nw1) : nw2;
23     (loc = nw2) : nw3;
24     (loc = nw3) : nw4;
25     (loc = nw4) : nw5;
26     (loc = nw5) : nw6;
27     (loc = nw6) : nw7;
28     (loc = nw7) : end;
29     (loc = end) : start;
30   esac;
31
32 -- network 1
33 next(inout_bInOutPLC) :=
34   case
35     (loc = nw1) & (!bInToggle & !bInLatch): FALSE;
36     (loc = nw2) & (bInEnable & inout_bInOutHMI & bInToggle & inout_bInOutPLC): FALSE
       ;
37     (loc = nw3) & (bInEnable & inout_bInOutHMI & bInToggle & !inout_bInOutPLC): TRUE
       ;
38     (loc = nw4) & (bInEnable & inout_bInOutHMI & !bInToggle): TRUE;
39     (loc = end) : {FALSE, TRUE};
40     TRUE : inout_bInOutPLC;
41   esac;
42
43 -- network 2
44 next(inout_bInOutHMI) :=
45   case
46     (loc = nw2) & (bInEnable & inout_bInOutHMI & bInToggle & inout_bInOutPLC): FALSE
       ;
47     (loc = nw6) : FALSE;
48     (loc = end) : {FALSE, TRUE};
49     TRUE : inout_bInOutHMI;
50   esac;
51
52 -- network 5
53 next(output_bOutEnable) :=
54   case
55     (loc = nw5) : bInEnable;
56     TRUE : output_bOutEnable;
57   esac;
58
59
60 MODULE main
61 VAR
62 bInToggle : boolean;
63 bInLatch : boolean;
64 bInEnable : boolean;
65 Test_FC329 : fc329(bInToggle, bInLatch, bInEnable);
66
67 ASSIGN
68 init(bInToggle) := FALSE;
69 init(bInLatch) := FALSE;
```

```
init(bInEnable) := FALSE;

next(bInToggle) :=
  case
    (Test_FC329.loc = end) : {FALSE, TRUE};
    TRUE : bInToggle;
  esac;

next(bInLatch) :=
  case
    (Test_FC329.loc = end) : {FALSE, TRUE};
    TRUE : bInLatch;
  esac;

next(bInEnable) :=
  case
    (Test_FC329.loc = end) : {FALSE, TRUE};
    TRUE : bInEnable;
  esac;
```

FC333:

```
MODULE fc333(bInEstop, bInError, bInForwardWork, bInReverseHome, bInOnForwardWork,
    bInOnReverseHome, bInSimulate)
VAR
global : global;
loc : {start, nw1, nw2, nw3, nw4, nw5, nw6, nw7, nw8, end};

output_iOutStatus : signed word[16];

ASSIGN
init(loc) := start;

init(output_iOutStatus) := 0sd16_0;

next(loc) :=
  case
    (loc = start) : nw1;
    (loc = nw1) : nw2;
    (loc = nw2) : nw3;
    (loc = nw3) : nw4;
    (loc = nw4) : nw5;
    (loc = nw5) : nw6;
    (loc = nw6) : nw7;
    (loc = nw7) : nw8;
    (loc = nw8) : end;
    (loc = end) : start;
  esac;

-- network 1
next(output_iOutStatus) :=
  case

    (loc = nw1) & (bInSimulate & bInOnReverseHome): global.
    HMI_Status_SimulatedOnReverseHome;
```

```
32      (loc = nw1) & (!bInSimulate & bInOnReverseHome): global.HMI_Status_OnReverseHome
        ;
33      (loc = nw2) & (bInSimulate & bInOnForwardWork): global.
        HMI_Status_SimulatedOnForwardWork;
34      (loc = nw2) & (!bInSimulate & bInOnForwardWork): global.HMI_Status_OnForwardWork
        ;
35      (loc = nw3) & (bInSimulate & bInReverseHome): global.
        HMI_Status_SimulatedReverseHome;
36      (loc = nw3) & (!bInSimulate & bInReverseHome): global.HMI_Status_ReverseHome;
37      (loc = nw4) & (bInSimulate & bInForwardWork): global.
        HMI_Status_SimulatedForwardWork;
38      (loc = nw4) & (!bInSimulate & bInForwardWork): global.HMI_Status_ForwardWork;
39      (loc = nw5) & (bInError) : global.HMI_Status_Error;
40      (loc = nw6) & (bInEstop) : global.HMI_Status_Estop;
41      (loc = nw7) & (!bInEstop & !bInError & !bInForwardWork & !bInReverseHome & !
        bInOnForwardWork & !bInOnReverseHome) : global.HMI_Status_Stopped;
42      TRUE : output_iOutStatus;
43    esac;
44
45
46  MODULE global
47  DEFINE
48  HMI_Status_SimulatedOnReverseHome := 0sd16_10;
49  HMI_Status_OnReverseHome := 0sd16_6;
50  HMI_Status_SimulatedOnForwardWork := 0sd16_9;
51  HMI_Status_OnForwardWork := 0sd16_5;
52  HMI_Status_SimulatedReverseHome := 0sd16_8;
53  HMI_Status_ReverseHome := 0sd16_4;
54  HMI_Status_SimulatedForwardWork := 0sd16_7;
55  HMI_Status_ForwardWork := 0sd16_3;
56  HMI_Status_Error := 0sd16_2;
57  HMI_Status_Estop := 0sd16_1;
58  HMI_Status_Stopped := 0sd16_0;
59
60
61  MODULE main
62  VAR
63  bInEstop : boolean;
64  bInError : boolean;
65  bInForwardWork : boolean;
66  bInReverseHome : boolean;
67  bInOnForwardWork : boolean;
68  bInOnReverseHome : boolean;
69  bInSimulate : boolean;
70
71  Test_FC333 : fc333(bInEstop, bInError, bInForwardWork, bInReverseHome,
        bInOnForwardWork, bInOnReverseHome, bInSimulate);
72
73  ASSIGN
74  init(bInEstop) := FALSE;
75  init(bInError) := FALSE;
76  init(bInForwardWork) := FALSE;
77  init(bInReverseHome) := FALSE;
78  init(bInOnForwardWork) := FALSE;
79  init(bInOnReverseHome) := FALSE;
80  init(bInSimulate) := FALSE;
```

```
81
82 next(bInEstop) :=
83   case
84     (Test_FC333.loc = end) : {FALSE, TRUE};
85     TRUE : bInEstop;
86   esac;
87
88 next(bInError) :=
89   case
90     (Test_FC333.loc = end) : {FALSE, TRUE};
91     TRUE : bInError;
92   esac;
93
94 next(bInForwardWork) :=
95   case
96     (Test_FC333.loc = end) : {FALSE, TRUE};
97     TRUE : bInForwardWork;
98   esac;
99
100 next(bInReverseHome) :=
101   case
102     (Test_FC333.loc = end) : {FALSE, TRUE};
103     TRUE : bInReverseHome;
104   esac;
105
106 next(bInOnForwardWork) :=
107   case
108     (Test_FC333.loc = end) : {FALSE, TRUE};
109     TRUE : bInOnForwardWork;
110   esac;
111
112 next(bInOnReverseHome) :=
113   case
114     (Test_FC333.loc = end) : {FALSE, TRUE};
115     TRUE : bInOnReverseHome;
116   esac;
117
118 next(bInSimulate) :=
119   case
120     (Test_FC333.loc = end) : {FALSE, TRUE};
121     TRUE : bInSimulate;
122   esac;
```

FB5:

```
1 MODULE fb5()
2 VAR
3 loc : {start, nw1, nw2, nw3, nw4, nw5, nw6, nw7 ,nw8, nw9, nw10, nw11, nw12, nw13,
      nw14, nw15, nw16, nw17, nw18, nw19, nw20, nw21, nw22, end};
4
5 Ib_PBStart : boolean;
6 Ib_LvlContEmpty : boolean;
7 Ib_LvlContFull : boolean;
8 Ib_PBFillLiq1 : boolean;
9 Ib_PBEmptyCont : boolean;
10 Ib_PBFillLiq2 : boolean;
```

```
Ib_PBStop : boolean;
Ib_VlvDrnClosed : boolean;
Ib_VlvDrnOpened : boolean;
Ib_VlvLiq1Closed : boolean;
Ib_VlvLiq1Opened : boolean;
Ib_VlvLiq2Closed : boolean;
Ib_VlvLiq2Opened : boolean;

static_CurrentStep : signed word[16];
static_NextStep : signed word[16];
static_bVlvDrn_Open : boolean;
static_bVlvLiq1_Open : boolean;
static_bVlvLiq2_Open : boolean;

ASSIGN
init(loc) := start;

init(Ib_PBStart) := FALSE;
init(Ib_LvlContEmpty) := FALSE;
init(Ib_LvlContFull) := FALSE;
init(Ib_PBFillLiq1) := FALSE;
init(Ib_PBEmptyCont) := FALSE;
init(Ib_PBFillLiq2) := FALSE;
init(Ib_PBStop) := FALSE;
init(Ib_VlvDrnClosed) := FALSE;
init(Ib_VlvDrnOpened) := FALSE;
init(Ib_VlvLiq1Closed) := FALSE;
init(Ib_VlvLiq1Opened) := FALSE;
init(Ib_VlvLiq2Closed) := FALSE;
init(Ib_VlvLiq2Opened) := FALSE;

init(static_CurrentStep) := 0sd16_0;
init(static_NextStep) := 0sd16_0;
init(static_bVlvDrn_Open) := FALSE;
init(static_bVlvLiq1_Open) := FALSE;
init(static_bVlvLiq2_Open) := FALSE;

next(loc) :=
  case
    (loc = start) : nw1;
    (loc = nw1) : nw2;
    (loc = nw2) : nw3;
    (loc = nw3) : nw4;
    (loc = nw4) : nw5;
    (loc = nw5) : nw6;
    (loc = nw6) : nw7;
    (loc = nw7) : nw8;
    (loc = nw8) : nw9;
    (loc = nw9) : nw10;
    (loc = nw10) : nw11;
    (loc = nw11) : nw12;
    (loc = nw12) : nw13;
    (loc = nw13) : nw14;
    (loc = nw14) : nw15;
    (loc = nw15) : nw16;
    (loc = nw16) : nw17;
```

```
    (loc = nw17) : nw18;
    (loc = nw18) : nw19;
    (loc = nw19) : nw20;
    (loc = nw20) : nw21;
    (loc = nw21) : nw22;
    (loc = nw22) : end;
    (loc = end) : start;
  esac;

next(Ib_PBStart) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_PBStart;
  esac;
next(Ib_LvlContEmpty) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_LvlContEmpty;
  esac;
next(Ib_LvlContFull) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_LvlContFull;
  esac;
next(Ib_PBFillLiq1) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_PBFillLiq1;
  esac;
next(Ib_PBEmptyCont) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_PBEmptyCont;
  esac;
next(Ib_PBFillLiq2) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_PBFillLiq2;
  esac;
next(Ib_PBStop) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_PBStop;
  esac;

next(Ib_VlvDrnClosed) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_VlvDrnClosed;
  esac;
next(Ib_VlvDrnOpened) :=
  case
    (loc = end) : {FALSE, TRUE};
    TRUE : Ib_VlvDrnOpened;
  esac;
next(Ib_VlvLiq1Closed) :=
```

```
123   case
124     (loc = end) : {FALSE, TRUE};
125     TRUE : Ib_VlvLiq1Closed;
126   esac;
127 next(Ib_VlvLiq1Opened) :=
128   case
129     (loc = end) : {FALSE, TRUE};
130     TRUE : Ib_VlvLiq1Opened;
131   esac;
132 next(Ib_VlvLiq2Closed) :=
133   case
134     (loc = end) : {FALSE, TRUE};
135     TRUE : Ib_VlvLiq2Closed;
136   esac;
137 next(Ib_VlvLiq2Opened) :=
138   case
139     (loc = end) : {FALSE, TRUE};
140     TRUE : Ib_VlvLiq2Opened;
141   esac;
142
143 -- network 1-19
144 next(static_NextStep) :=
145   case
146     (loc = nw1) & (Ib_PBStart & (static_CurrentStep = 0sd16_0)): 0sd16_10;
147     (loc = nw3) & (Ib_LvlContEmpty & !Ib_LvlContFull & (static_CurrentStep = 0
      sd16_10)): 0sd16_20;
148     (loc = nw3) & (!(Ib_LvlContEmpty & !Ib_LvlContFull) & (static_CurrentStep = 0
      sd16_10)): 0sd16_11;
149     (loc = nw4) & (Ib_LvlContEmpty & !Ib_LvlContFull & (static_CurrentStep = 0
      sd16_11)): 0sd16_20;
150     (loc = nw6) & (Ib_PBFillLiq1 & (static_CurrentStep = 0sd16_20)): 0sd16_21;
151     (loc = nw7) & (Ib_LvlContFull & (static_CurrentStep = 0sd16_21)): 0sd16_30;
152     (loc = nw9) & (Ib_PBEmptyCont & (static_CurrentStep = 0sd16_30)): 0sd16_31;
153     (loc = nw10) & (Ib_LvlContEmpty & (static_CurrentStep = 0sd16_31)): 0sd16_40;
154     (loc = nw12) & (Ib_PBFillLiq2 & (static_CurrentStep = 0sd16_40)): 0sd16_41;
155     (loc = nw13) & (Ib_LvlContFull & (static_CurrentStep = 0sd16_41)): 0sd16_50;
156     (loc = nw15) & (Ib_PBEmptyCont & (static_CurrentStep = 0sd16_50)): 0sd16_51;
157     (loc = nw16) & (Ib_LvlContEmpty & (static_CurrentStep = 0sd16_51)): 0sd16_60;
158     (loc = nw18) & (Ib_PBStop & (static_CurrentStep = 0sd16_60)): 0sd16_0;
159     TRUE : static_NextStep;
160   esac;
161
162 -- network 20
163 next(static_CurrentStep) :=
164   case
165     (loc = nw20) & !(static_NextStep = static_CurrentStep): static_NextStep;
166     TRUE: static_CurrentStep;
167   esac;
168
169 -- network 22
170 next(static_bVlvDrn_Open) :=
171   case
172     (loc = nw22): ((static_CurrentStep = 0sd16_11) | (static_CurrentStep = 0sd16_31)
      | (static_CurrentStep = 0sd16_51)) & (Ib_VlvLiq1Closed & Ib_VlvLiq2Closed);
173     TRUE : static_bVlvDrn_Open;
174   esac;
```

```
175 next(static_bVlvLiq1_Open) :=
176   case
177     (loc = nw22): ((static_CurrentStep = 0sd16_21) & (Ib_VlvDrnClosed &
      Ib_VlvLiq2Closed));
178     TRUE : static_bVlvLiq1_Open;
179   esac;
180 next(static_bVlvLiq2_Open) :=
181   case
182     (loc = nw22): ((static_CurrentStep = 0sd16_41) & (Ib_VlvDrnClosed &
      Ib_VlvLiq1Closed));
183     TRUE : static_bVlvLiq2_Open;
184   esac;
185
186 MODULE main
187 VAR
188 Test_FB5 : fb5();
189
190 ASSIGN
```

G. APPENDIX: NUXMV PROPERTIES TO VERIFY

FB110:

```
-- Basic reachability
SPEC AG AF (TestVlv.loc = end)
SPEC AF (TestVlv.loc = end)
SPEC AG (TestVlv.loc = end -> AX (TestVlv.loc = start))

-- Basic liveness
SPEC EF (TestVlv.loc = end & TestVlv.output_bOutError)
SPEC EF (TestVlv.loc = end & TestVlv.static_bTON_TimeOut)
SPEC EF (TestVlv.loc = end & TestVlv.output_ERROR_Valve_NoHomeFeedback)
SPEC EF (TestVlv.loc = end & TestVlv.output_bOutCommandWork)
SPEC EF (TestVlv.loc = end & TestVlv.output_bOutCommandHome)

-- Functionality
SPEC AG (TestVlv.loc = end -> !(TestVlv.output_bOutCommandHome & TestVlv.
    output_bOutCommandWork))
SPEC AG ((TestVlv.loc = end & !bInEstop & bInEnable & bInCommandWork & !bInSimulate
    & iInMode = 0sd16_1 & !TestVlv.output_bOutError) -> EF TestVlv.
    output_bOutCommandWork)
SPEC AG ((TestVlv.loc = end & !bInEstop & bInEnable & !bInCommandWork & !bInSimulate
     & iInMode = 0sd16_1 & !TestVlv.output_bOutError) -> EF TestVlv.
    output_bOutCommandHome)
SPEC AG ((TestVlv.loc = end & bInCommandWork & iInMode = 0sd16_1) -> !TestVlv.
    output_bOutCommandHome)
SPEC AG ((TestVlv.loc = end & !bInCommandWork & iInMode = 0sd16_1) -> !TestVlv.
    output_bOutCommandWork)
SPEC AG (TestVlv.loc = end & bInEstop -> (!TestVlv.output_bOutCommandHome & !TestVlv
    .output_bOutCommandWork))

SPEC AG ((TestVlv.loc = end & TestVlv.output_bOutActiveHome & TestVlv.
    timer_TON_TimeOut=0 & !bInSimulate & !TestVlv.static_bReset) -> AF(bInSignalHome)
    )
SPEC AG ((TestVlv.loc = end & TestVlv.output_bOutActiveWork & TestVlv.
    timer_TON_TimeOut=0 & !bInSimulate & !TestVlv.static_bReset) -> AF(bInSignalWork)
    )
SPEC AG (TestVlv.loc = end -> !(TestVlv.output_bOutActiveHome & TestVlv.
    output_bOutActiveWork))
SPEC AG ((TestVlv.loc = end & TestVlv.output_bOutCommandWork & iInMode = 0sd16_1) ->
     bInCommandWork)
SPEC AG ((TestVlv.loc = end & TestVlv.output_bOutCommandHome & iInMode = 0sd16_1) ->
     !bInCommandWork)
```

FB330:

```
-- Basic reachability
SPEC AG AF (Test_FB330.loc = end)
SPEC AF (Test_FB330.loc = end)
SPEC AG (Test_FB330.loc = end -> AX (Test_FB330.loc = start))

-- Basic liveness
SPEC EF (Test_FB330.loc = end & Test_FB330.output_bOutErrorExists)
SPEC EF (Test_FB330.loc = end & Test_FB330.output_iOutScrollingErrorNumber = 0sd16_0
    )
```

```
SPEC EF (Test_FB330.loc = end & Test_FB330.output_iOutScrollingErrorNumber = 0sd16_1)
SPEC EF (Test_FB330.loc = end & Test_FB330.output_iOutScrollingErrorNumber = 0sd16_2)
SPEC EF (Test_FB330.loc = end & Test_FB330.output_iOutScrollingErrorNumber = 0sd16_3)
SPEC EF (Test_FB330.loc = end & Test_FB330.output_iOutScrollingErrorNumber = 0sd16_4)
SPEC EF (Test_FB330.loc = end & Test_FB330.output_iOutScrollingErrorNumber = 0sd16_5)
SPEC EF (Test_FB330.loc = end & !Test_FB330.bInError01)

-- Functionality
SPEC EF (Test_FB330.loc = end & !Test_FB330.bInError01 & !Test_FB330.bInError02 & !Test_FB330.bInError03 & !Test_FB330.bInError04 & !Test_FB330.bInError05)
SPEC AG AF ((Test_FB330.loc = end & Test_FB330.output_bOutErrorExists) -> (!(Test_FB330.output_iOutScrollingErrorNumber = 0sd16_0)))
SPEC AG ((Test_FB330.loc = end & !Test_FB330.output_bOutErrorExists) -> (Test_FB330.output_iOutScrollingErrorNumber = 0sd16_0))
SPEC AG AF ((Test_FB330.loc = end & Test_FB330.bInError01 & !Test_FB330.bInError02 & !Test_FB330.bInError03 & !Test_FB330.bInError04 & !Test_FB330.bInError05 & !(Test_FB330.output_iOutScrollingErrorNumber = 0sd16_0)) -> (Test_FB330.output_iOutScrollingErrorNumber = 0sd16_1))
SPEC AG ((Test_FB330.loc = end & !Test_FB330.bInError01 & !Test_FB330.bInError02 & !Test_FB330.bInError03 & !Test_FB330.bInError04 & !Test_FB330.bInError05) -> (Test_FB330.output_iOutScrollingErrorNumber = 0sd16_0))


-- Test compositional Model Checking (FB110)
SPEC AG ((Test_FB330.loc = end & Test_FB330.bInError01) -> (Test_FB330.output_bOutErrorExists))
SPEC AG ((Test_FB330.loc = end & (Test_FB330.bInError03 | Test_FB330.bInError04)) -> (Test_FB330.output_bOutErrorExists))
```

FC329:

```
-- Basic reachability
SPEC AG AF (Test_FC329.loc = end)
SPEC AF (Test_FC329.loc = end)
SPEC AG (Test_FC329.loc = end -> AX (Test_FC329.loc = start))

-- Basic liveness
SPEC EF (Test_FC329.loc = end & Test_FC329.output_bOutEnable)
SPEC EF (Test_FC329.loc = end & !Test_FC329.output_bOutEnable)
SPEC EF (Test_FC329.loc = end & !Test_FC329.inout_bInOutHMI)
SPEC EF (Test_FC329.loc = end & Test_FC329.inout_bInOutPLC)
SPEC EF (Test_FC329.loc = end & !Test_FC329.inout_bInOutPLC)

-- Functionality
SPEC AG ((Test_FC329.loc = end & bInEnable) -> (Test_FC329.output_bOutEnable))
SPEC AG ((Test_FC329.loc = end & bInEnable & bInToggle & Test_FC329.inout_bInOutHMI & !Test_FC329.inout_bInOutPLC) -> Test_FC329.inout_bInOutPLC)
SPEC AG ((Test_FC329.loc = end & bInEnable & !bInToggle & Test_FC329.inout_bInOutHMI) -> Test_FC329.inout_bInOutPLC)
```

FC333:

```
-- Basic reachability
SPEC AG AF (Test_FC333.loc = end)
SPEC AF (Test_FC333.loc = end)
SPEC AG (Test_FC333.loc = end -> AX (Test_FC333.loc = start))

-- Basic liveness
SPEC EF (Test_FC333.loc = end & (Test_FC333.output_iOutStatus = 0sd16_0))
SPEC EF (Test_FC333.loc = end & !(Test_FC333.output_iOutStatus = 0sd16_0))

-- Functionality
SPEC AG ((Test_FC333.loc = end & bInEstop) -> (Test_FC333.output_iOutStatus =
    Test_FC333.global.HMI_Status_Estop))
SPEC AG ((Test_FC333.loc = end & !bInEstop & bInError) -> (Test_FC333.
    output_iOutStatus = Test_FC333.global.HMI_Status_Error))

SPEC AG ((Test_FC333.loc = end & !bInEstop & !bInError & bInSimulate &
    bInForwardWork & !bInOnForwardWork & !bInOnReverseHome & !bInReverseHome) -> (
    Test_FC333.output_iOutStatus = Test_FC333.global.HMI_Status_SimulatedForwardWork)
    )
SPEC AG ((Test_FC333.loc = end & !bInEstop & !bInError & !bInSimulate &
    bInForwardWork & !bInOnForwardWork & !bInOnReverseHome & !bInReverseHome) -> (
    Test_FC333.output_iOutStatus = Test_FC333.global.HMI_Status_ForwardWork))

SPEC AG ((Test_FC333.loc = end & !bInEstop & !bInError & bInSimulate &
    bInOnForwardWork & !bInForwardWork & !bInOnReverseHome & !bInReverseHome) -> (
    Test_FC333.output_iOutStatus = Test_FC333.global.
    HMI_Status_SimulatedOnForwardWork))
SPEC AG ((Test_FC333.loc = end & !bInEstop & !bInError & !bInSimulate &
    bInOnForwardWork & !bInForwardWork & !bInOnReverseHome & !bInReverseHome) -> (
    Test_FC333.output_iOutStatus = Test_FC333.global.HMI_Status_OnForwardWork))

SPEC AG ((Test_FC333.loc = end & !bInEstop & !bInError & bInSimulate &
    bInReverseHome & !bInOnReverseHome & !bInForwardWork & !bInOnForwardWork) -> (
    Test_FC333.output_iOutStatus = Test_FC333.global.HMI_Status_SimulatedReverseHome)
    )
SPEC AG ((Test_FC333.loc = end & !bInEstop & !bInError & !bInSimulate &
    bInReverseHome & !bInOnReverseHome & !bInForwardWork & !bInOnForwardWork) -> (
    Test_FC333.output_iOutStatus = Test_FC333.global.HMI_Status_ReverseHome))

SPEC AG ((Test_FC333.loc = end & !bInEstop & !bInError & bInSimulate &
    bInOnReverseHome & !bInReverseHome & !bInForwardWork & !bInOnForwardWork) -> (
    Test_FC333.output_iOutStatus = Test_FC333.global.
    HMI_Status_SimulatedOnReverseHome))
SPEC AG ((Test_FC333.loc = end & !bInEstop & !bInError & !bInSimulate &
    bInOnReverseHome & !bInReverseHome & !bInForwardWork & !bInOnForwardWork) -> (
    Test_FC333.output_iOutStatus = Test_FC333.global.HMI_Status_OnReverseHome))
```

FB5:

```
-- Basic reachability
SPEC AG AF (Test_FB5.loc = end)
SPEC AF (Test_FB5.loc = end)
SPEC AG (Test_FB5.loc = end -> AX (Test_FB5.loc = start))

-- Basic liveness
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_0)
```

```
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_11)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_20)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_21)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_30)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_31)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_40)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_41)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_50)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_51)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_CurrentStep = 0sd16_60)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_bVlvDrn_Open)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_bVlvLiq1_Open)
SPEC EF (Test_FB5.loc = end & Test_FB5.static_bVlvLiq2_Open)


-- Functionality
SPEC AG ((Test_FB5.loc = end) -> (!(Test_FB5.static_bVlvLiq1_Open & Test_FB5.
    static_bVlvLiq2_Open)))
SPEC AG (((Test_FB5.loc = end) & Test_FB5.static_bVlvDrn_Open) -> (!(Test_FB5.
    static_bVlvLiq1_Open | Test_FB5.static_bVlvLiq2_Open)))
SPEC AG (((Test_FB5.loc = end) & Test_FB5.static_bVlvLiq1_Open) -> (!(Test_FB5.
    static_bVlvDrn_Open | Test_FB5.static_bVlvLiq2_Open)))
SPEC AG (((Test_FB5.loc = end) & Test_FB5.static_bVlvLiq2_Open) -> (!(Test_FB5.
    static_bVlvDrn_Open | Test_FB5.static_bVlvLiq1_Open)))
SPEC AG ((Test_FB5.loc = end & Test_FB5.static_bVlvDrn_Open) -> (Test_FB5.
    Ib_VlvLiq1Closed & Test_FB5.Ib_VlvLiq2Closed))
SPEC AG ((Test_FB5.loc = end & Test_FB5.static_bVlvLiq1_Open) -> (Test_FB5.
    Ib_VlvDrnClosed & Test_FB5.Ib_VlvLiq2Closed))
SPEC AG ((Test_FB5.loc = end & Test_FB5.static_bVlvLiq2_Open) -> (Test_FB5.
    Ib_VlvDrnClosed & Test_FB5.Ib_VlvLiq1Closed))
```

H. APPENDIX: COMPOSITIONAL NUXMV PROPERTIES TO VERIFY

FB110:

```
-- Test compositional Model Checking (FB330)
INVAR ((TestVlv.loc = end & TestVlv.output_ERROR_Valve_NoHomeFeedback) -> (TestVlv.
    inout_HMI_ValveControl_bError))
SPEC AG ((TestVlv.loc = end & TestVlv.output_ERROR_Valve_NoHomeFeedback & TestVlv.
    output_bOutAuto) -> TestVlv.output_bOutError)

SPEC AG ((TestVlv.loc = end & TestVlv.output_ERROR_Valve_NoHomeFeedback & TestVlv.
    output_bOutAuto) -> AF (!TestVlv.output_bOutCommandHome & !TestVlv.
    output_bOutCommandWork))

INVAR ((TestVlv.loc = end & (TestVlv.output_ERROR_Valve_HomeFeedbackStillActive |
    TestVlv.output_ERROR_Valve_WorkFeedbackStillActive)) -> TestVlv.
    inout_HMI_ValveControl_bError)
SPEC AG((TestVlv.loc = end & TestVlv.output_bOutActiveHome & !TestVlv.
    output_bOutError & !bInSimulate & !TestVlv.static_bReset & TestVlv.
    timer_TON_TimeOut < 1) -> (bInSignalHome & !bInSignalWork))

-- Test compositional Model Checking (FC329)
INVAR ((TestVlv.loc = end & TestVlv.static_bManualMode) -> (TestVlv.
    inout_HMI_ValveControl_bPBEN_Home))
SPEC AG ((TestVlv.loc = end & TestVlv.static_bManualMode & TestVlv.static_bPB_Home &
     !TestVlv.static_bPB_Work & !bInSimulate & !TestVlv.output_bOutError & !
    bInSignalHome) -> TestVlv.output_bOutCommandHome)

-- Test compositional Model Checking (FC333)
INVAR (TestVlv.loc=end & bInEstop) -> (TestVlv.inout_HMI_ValveControl_iStatus = 0
    sd16_1)
SPEC AG((TestVlv.loc = end & bInEstop) -> (TestVlv.inout_HMI_ValveControl_iStatus =
    0sd16_1))
```