

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Design of virtual machinery plants and PLCopen compliant safety related functional blocks for bridging PLC programming and machinery safety classes

Pedro Miguel Freitas Sargo



Master's Degree in Mechanical Engineering Automation Branch

Supervisor: Prof. António José Pessoa de Magalhães

July 14, 2023

**Design of virtual machinery plants and PLCopen
compliant safety related functional blocks for bridging
PLC programming and machinery safety classes**

Pedro Miguel Freitas Sargo

Master's Degree in Mechanical Engineering Automation Branch

July 14, 2023

Abstract

Functional safety is a critical aspect of machinery design and operation in the rapidly advancing era of automation. Answering this requirement, FEUP's Mechanical Engineering "Machinery Safety" (MS) curricular unit aims to provide to master course students advanced concepts and methodologies in machinery safety. Yet, to effectively teach functional safety, practical application of concepts is crucial for students to receive feedback and gauge their understanding.

The growing adoption of programmable logic controllers (PLCs) in safety applications emphasizes the importance of integrating PLC programming with safety of machinery subjects. Appearing in parallel with "Machinery Safety" is "Industrial Computing" (IC), which equips students with the necessary skills for computer systems integration and advanced PLC programming. Bridging the gap between these two curricular units would enable students to develop a deeper understanding of functional safety design and gain practical experience in PLC programming. To make this connection possible, a platform is needed for developing and testing functional safety systems. Simulation environments are crucial for hands-on training in industrial automation and safety-related applications, as they provide a safe space to learn and make errors without endangering individuals or equipment.

Factory I/O, developed by Real Games, has emerged as an optimal solution for visualizing and testing logic design problems on a complete set of logic controllers. However, the need for a dedicated simulation platform designed specifically for safety applications is highlighted. And so the present dissertation is framed on the context of expanding Factory I/O's scope to incorporate safety-related scenarios that would leverage its educational potential, creating a powerful resource that bridges PLC programming and machinery safety curricula.

To start, the motivation and objectives of the dissertation, along with the anticipated practical benefits are introduced. In the light of the state of the art the significant choices and approaches adopted during the development of the dissertation are justified. Using Codesys, the tool familiar to "Industrial Computing" students for programming industrial control systems, is the natural solution for the development requirements. Then the focus shifts to the design and efficient interaction of crucial hardwired safety components. It establishes the groundwork for a thorough comprehension of functional safety fundamentals. Delving into the simulation of a functional safety application centered around a safety relay, serves as a logical starting point for the project. Then the process starts of developing safety-related function blocks for a library used in safety PLC application programs. This meticulous process is simplified and is sure to follow relevant safety regulations by following the safety-related specifications established by PLCopen. After testing the function blocks the final chapter provides a set of demonstrations of the practical applications of the safety function blocks and simulated safety devices through realistic case scenarios. These scenarios serve as tangible examples to illustrate the effectiveness of the implemented safety function blocks and their compatibility with different industrial environments.

Keywords: Factory I/O, Function Blocks, Functional safety, IEC 61131-3, Machinery Safety, PLCopen Safety, Programmable Logic Controllers.

Resumo

Na atual era de rápida evolução da automação, a segurança funcional é um aspeto crítico do design e funcionamento de máquinas e instalações automáticas. A unidade curricular "Segurança de Máquinas" tem como objetivo dotar os alunos de metodologias avançadas em segurança de máquinas. Para um ensino eficaz da segurança funcional, a aplicação prática dos conceitos é crucial para que os alunos recebam feedback e avaliem a sua compreensão.

A crescente adoção de controladores lógicos programáveis (CLPs) em aplicações de segurança enfatiza a importância de integrar a programação de PLCs com as matérias de segurança de máquinas. Esta integração é possível com a unidade curricular de "Computação Industrial", que dota os alunos das competências necessárias à integração de sistemas informáticos e à programação avançada de autómatos. A aproximação entre estas duas unidades curriculares permitirá aos alunos aprofundar o conhecimento do projeto de segurança funcional e adquirir experiência prática na programação de autómatos. Para facilitar atividades práticas comuns a estas disciplinas, é necessária uma plataforma para desenvolver e testar sistemas de segurança funcional. Os ambientes de simulação são cruciais para a formação prática em automação industrial e aplicações relacionadas com a segurança, uma vez que proporcionam um espaço seguro para aprender e cometer erros sem pôr em perigo pessoas ou equipamento.

O Factory I/O, desenvolvido pela Real Games, surgiu como uma solução adequada para visualizar e testar problemas de conceção lógica num conjunto completo de controladores lógicos. No entanto, destaca-se a necessidade de uma plataforma de simulação dedicada especificamente para aplicações de segurança. Assim, a presente dissertação enquadra-se no contexto da expansão da aplicação do Factory I/O para incorporar cenários relacionados com segurança de máquinas que potenciem o seu potencial educativo, criando um recurso poderoso que faça a ponte entre a programação de PLCs e segurança de máquinas.

Para começar, é apresentada a motivação e os objectivos da dissertação, bem como os benefícios práticos previstos. À luz do estado da arte, são justificadas as escolhas e abordagens significativas adotadas durante o desenvolvimento da dissertação. A utilização do Codesys, a ferramenta familiar aos estudantes de "Computação Industrial" para a programação de sistemas de controlo industrial, é a solução natural para as necessidades de desenvolvimento. Em seguida, o foco passa a ser a conceção e a interação eficiente de componentes de segurança cablados cruciais. Estabelece-se assim as bases para uma compreensão aprofundada dos fundamentos de segurança funcional. A simulação de uma aplicação de segurança funcional centrada num relé de segurança serve como ponto de partida lógico para o projeto. Em seguida, inicia-se o processo de desenvolvimento de blocos funcionais de segurança para uma biblioteca a utilizar em programas de aplicação de PLC de segurança. Este processo meticuloso é simplificado e respeita as normas de segurança pertinentes, ao seguir as especificações de segurança estabelecidas pela PLCopen. Depois de testar exaustivamente os blocos funcionais, o capítulo final apresenta uma demonstração da aplicação prática dos blocos funcionais de segurança e dos dispositivos de segurança simulados através de cenários realistas. Estes cenários servem como exemplos tangíveis para ilustrar a eficácia dos blocos funcionais implementados e a sua compatibilidade com diferentes ambientes industriais.

Palavras-chave: Blocos de Funções, Controladores Lógicos Programáveis, Factory I/O, IEC 61131-3, PLCopen Safety, Segurança Funcional, Segurança de Máquinas.

Acknowledgements

First and foremost I would like to thank my supervisor, Professor António Pessoa de Magalhães for his guidance, constant availability and help in the past months. Furthermore, I would also like to thank him for introducing me to the world of PLC programming that attracted my interest. Additionally, a word of appreciation to Professor Filipa Lima for her professional advice in this project.

A special thanks to my parents and my siblings for always being there for me, and for making it possible for me to come study in Porto. In particular, thanks to my mother for all the love, hard work and daily support. And to my brother André, and Marta for sharing their experience and words of advice.

Thank you to my friends for all the fun and for keeping me (in)sane over the years. A special thanks to Áurea for the immeasurable encouragement, unwavering support and for making the last years a lot easier.

Pedro Miguel Freitas Sargo

À minha família.

Contents

1	Introduction	1
1.1	Objectives	2
1.2	Dissertation Structure	3
2	Context of the dissertation	4
2.1	Bridging safety of machinery and PLC programming	4
2.2	FACTORY I/O: the promising, yet embryonic, bridging link	5
2.3	Finding an early solution for functional safety development and testing	6
2.3.1	Current functional safety solutions and practices	6
2.3.2	Adopted solution	8
2.4	Standards and practices to consider and include in Safety PLC programming . . .	9
2.5	Summary and steps to complete	10
3	Simulation of a safety relay application	12
3.1	Functional safety based on safety relays	13
3.1.1	Functionality of the simulated safety relay	13
3.2	Software simulation of PNOZ s5 relay	18
3.2.1	Safety relay FB	21
3.2.2	Mode Selection FB	22
3.2.3	Input device monitoring - Equivalent_FB	23
3.2.4	Start operation	25
3.2.5	Output enabling – E_Stop	25
3.3	Peripheral devices simulation	26
3.4	Simulating a safety relay based application	31
3.5	Concluding remarks	34
4	PLCopen compliant safety function blocks library development	36
4.1	Generic aspects of safety function blocks	36
4.2	Function block programming	39
4.2.1	Programming for optimized performance	39
4.2.2	Code optimized programming	43
4.3	Programmed safety function blocks and supporting devices	47
4.4	Testing the function blocks	50
4.5	Concluding remarks	51
5	Case Demonstrations	52
5.1	Palletizing station	52
5.2	Machining station	56
5.3	Multiple zone application	59

5.4	Concluding remarks	61
6	Conclusions and Future Work	62
6.1	Future Work	63

List of Figures

2.1	Factory I/O environment (1)	5
2.2	Hardwired vs programmable logic	7
2.3	Integration of safety logic and overall control system (2)	8
2.4	Safety FB definitions in PLCopen Technical Specification (3)	10
3.1	Emergency stop functional safety solution example. Adapted from: Siemens support website (4)	13
3.2	Block diagram and front view of the PNOZ s5 safety relay. Adapted from, Pilz PNOZ s5 Operating Manual(5)	14
3.3	Wiring diagram with rotary switches and LED lights representation. Adapted from Pilz PNOZ s5 Operating Manual(5)	14
3.4	Timing diagram of relay activation. Adapted from Pilz PNOZ s5 Operating Manual(5)	15
3.5	Safety relay input circuits	16
3.6	a) Single-channel input circuit b) Dual-channel input circuit. Adapted from: Pilz PNOZ s5 Operating Manual(5)	16
3.7	Example of T1 and T2 test signals	17
3.8	Representation of a short circuit across contacts	17
3.9	a) Rising edge start b) Falling edge start signal	18
3.10	Safety relay FB and encapsulated features	19
3.11	SRelay_Fault_Modes data type	20
3.12	SRelay_Model_A FB	21
3.13	CASE statement of Input1_Fault to control the LED lights	21
3.14	SRelay_Model_A FB and its methods	22
3.15	Model_A_Op_Modes data type	22
3.16	Mode_FB, inputs and outputs	23
3.17	Mode_FB function block programming	23
3.18	Equivalent_FB, input and outputs	23
3.19	Equivalent_FB and its methods	24
3.20	Equivalent_FB code	24
3.21	Start_FB, inputs and outputs	25
3.22	E_Stop FB, inputs and outputs	25
3.23	E_Stop Function Block	26
3.24	E_Stop associated Method	26
3.25	Interface display and view selector element of the emergency button interface	27
3.26	Fault injector element	28
3.27	Em_Button_2Contacts FB	28
3.28	Emergency button function representation	28
3.29	Em_Button_2Contacts FB with no fault	29
3.30	Outputs of the Em_Button_2Contacts FB enumeration data type	29
3.31	Em_Button_2Contacts FB outputs with "stuck on" fault	30

3.32	Em_Button_2Contacts FB outputs with "short circuit" fault	30
3.33	Emergency Button visual interface	30
3.34	Factory scenario for safety relay based solution	31
3.35	Diagram of the communication between programs and Factory I/O	32
3.36	Task configuration in Codesys	32
3.37	Safety solution components connection	33
3.38	Safety solution visual interface	33
3.39	OPC UA connection between Codesys and Factory I/O	34
4.1	PLCopen generic safety FB	37
4.2	Generic state diagram. Adapted from PLCopen Safety Software Technical Specification "Part 1: Concepts and Function Blocks" (3)	38
4.3	Example of a transient state, State 1	39
4.4	SF_EmergencyStop input and output variables. Adapted from: Adapted from PLCopen Safety Software Technical Specification "Part 1: Concepts and Function Blocks" (3)	40
4.5	SF_EmergencyStop. Adapted from: Figure 23 of the PLCopen Safety Software Technical Specification "Part 1: Concepts and Function Blocks": "State diagram for SF_EmergencyStop" - Page 64 (3)	41
4.6	Initial part of SF_EmergencyStop starting in "Idle" state	42
4.7	SF_EmergencyStop Outputs	43
4.8	SF_MutingSeq input and output variables. Adapted from: Adapted from PLCopen Safety Software Technical Specification "Part 1: Concepts and Function Blocks" (3)	43
4.9	SF_MutingSeq state diagram. Adapted from: Adapted from Figure 43 of the PLCopen Safety Software Technical Specification "Part 1: Concepts and Function Blocks" State diagram for SF_MutingSeq" - Page 104" (3)	45
4.10	Initial part of the SF_MutingSeq FB	46
4.11	Edge signal in SF_MutingSeq	47
4.12	Symbolic views of the safety guard with and without guard locking	49
4.13	Example of SF_MutingSeq FB testing method	50
4.14	Block diagram of the emergency stop function testing	51
5.1	Factory I/O palletizer access areas	53
5.2	Standard and safe control of the palletizing station	54
5.3	Stacked boxes approach light curtain. Muting enable command activates.	55
5.4	Stages after muting is activated.	55
5.5	Simulation of muting error in light curtain and override action	56
5.6	Standard and safe control structure of the CNC machine	57
5.7	Machining station simulation	58
5.8	Industrial plant with two zone controlled separately	59
5.9	Structure of the standard and safe control of the multiple zone machine	60
5.10	Zone 1 trespassing stops zone actuators.	60
5.11	Trespassing between zones stops all actuators	61

List of Tables

3.1	Front view of safety relay legend	15
3.2	Common variables in a safety relay class	19
3.3	Configuration variables for safety switch devices	27
4.1	Generic input parameters of safety FBs. Adapted from PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” (3)	37
4.2	Ouput setting of a generic "Init" state. Adapted from PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” (3)	38
4.3	Input and outputs parameters of the SF_MutingSeq FB. Adapted from PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” (3)	44
4.4	List of programmed safety function blocks and simulated devices	49

Acronyms

FB	Function Block
IC	Industrial Computing
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
MS	Machine Safety
PLC	Programmable Logic Controller
SCADA	Supervisory Control and Data Acquisition
SIL	Safety Integrity Level
ST	Structured Text

Chapter 1

Introduction

In the current fast-developing era of automation, functional safety plays a major role in the design and operation of machinery plants. Thus, a major challenge in engineering design is to guarantee machine safe operation.

The "Machinery Safety" (MS) curricular unit is designed to teach students in the Mechanical Engineering master's course with advanced methodologies in safety of machinery. To effectively teach functional safety in this field, practical application of the concepts is essential, allowing students to receive feedback and gauge their understanding of the subject.

The increasing use of programmable logic controllers (PLCs) in safety applications highlights the need to establish a strong connection between PLC programming and safety of machinery subjects. This connection is now possible as "Industrial Computing" appears in parallel with "Machinery Safety" for most students. "Industrial Computing" (IC) prepares students in industrial automation for computer systems integration, including advanced PLC programming. Bridging the gap between these two curricular units, will allow its students to gain a deeper understanding of functional safety design and practical experience in PLC programming within the field. Consequently, the need arises to identify a platform that enables the development and practical testing of functional safety systems to apply in practical activities common to MS and IC classes.

Practical training in industrial automation and safety-related applications presents certain challenges. Hands-on training is not possible without the occurrence of errors, which can potentially endanger both individuals and equipment. This emphasizes the importance of simulation environments.

The numerous advantages of utilizing virtual systems in teaching and learning PLC programming have been consistently demonstrated for several years through laboratory activities conducted in "Industrial Computing." In particular, the introduction of Factory I/O, developed by Real Games, has proven to be the optimal virtual solution for visualizing and testing logic design problems ranging from simple to complex.

However Factory I/O does not primarily focus on machinery safety features nor include significant safety components. And so it is highlighted that there is currently a lack of a dedicated

simulation platform specifically designed for safety applications. Therefore, it is logical to leverage the educational potential of Factory I/O by expanding its scope to incorporate safety-related scenarios. By doing so, Factory I/O will evolve into an even more powerful resource that effectively bridges the realms of PLC programming and machinery safety curricula. It is within this framework that the dissertation was proposed and will unfold.

1.1 Objectives

The objective of this dissertation is to create a virtual platform to enable the simulation and control of machinery safety scenarios, expanding the current possibilities of Factory I/O, serving as a bridge between "Machinery Safety" classes and "Industrial Computing" classes.

A major initial goal is the analysis of the currently available safety components and of the simulation possibilities of Factory IO for supporting the simulation of safety scenarios that are typically referred to and analyzed in safety of machinery classes.

Once the available options have been explored, the objective is to develop a strategy for implementing and programming safety functions. Safety PLCs are not only costly to acquire but also expensive to maintain. Additionally, they lack the widely used communication protocols that standard PLCs possess and that Factory I/O supports, including Modbus and OPC UA. To address these challenges, the optimal solution is to explore Codesys, a modern and free integrated development environment (IDE). Codesys seamlessly aligns with the IEC 61131-3 standard, which provides a common framework for programming industrial control systems. Moreover, "Industrial Computing" students are already familiar with Codesys, its usage, and functionalities. Codesys offers the flexibility to incorporate additional development requirements such as visualization tools and testing capabilities. From here, the development of function blocks specifically designed for functional safety purposes begins. These function blocks will adhere to the safety standards recommended by PLCopen. Following the safety-related specifications established by PLCopen simplifies the meticulous process of developing safety function blocks and ensures compliance with relevant safety regulations. To ensure their effectiveness, the function blocks will undergo thorough testing using Codesys in conjunction with Factory I/O, through the OPC UA communication protocol. To facilitate this collaboration, it will be necessary to simulate safety devices that are not currently available in Factory I/O. The simulated safety devices must meet specific requirements, including the capability for manual fault injection, manual command execution when needed, and seamless integration with Factory I/O. The envisioned integration of this development as an expansion to Factory I/O will further enhance the simulation capabilities, fostering a comprehensive and realistic simulation of safe machinery plants.

In summary and informally, the main objective of this project is to develop a platform that seamlessly connects the subjects of machinery safety and PLC programming. This will enable students with experience in PLC programming to easily transition to programmable safety solutions and gain practical knowledge in this field.

1.2 Dissertation Structure

Given the defined objectives chapter 2 begins by uncovering the motivation behind the dissertation, providing insight into the driving factors. The major objectives of the dissertation and the anticipated practical benefits are then clearly stated. The chapter concludes by providing a justification for the significant choices and approaches taken in the development of the dissertation, taking into account the current state of the art in the field.

With a contextualizing view of the development transmitted, chapter 3 focuses on the design and effective interaction of essential hardwired safety components. This chapter lays the foundation for a comprehensive understanding of the basics of functional safety and provides insights into the optimal approach for simulating safety devices. It delves into the simulation of a functional safety application that revolves around a safety relay, which serves as a logical starting point for the project. A demonstration showcases the outcomes of the developed functional safety system within a Factory I/O scenario, setting the stage for more sophisticated safety solutions in the future.

Chapter 4 discusses the development process of safety-related function blocks, which will form an integral part of a library to be utilized in safety PLC application programs. The chapter emphasizes the method employed to ensure the adherence of these function blocks to the PLCopen safety standard. Solutions are devised to address the challenges encountered during the programming of the function blocks. This is followed by a testing phase of the developed function blocks, leading to a demonstration that highlights their potential applications.

The practical application of the safety function blocks and simulated safety devices through realistic case scenarios is demonstrated in chapter 5. These scenarios serve as tangible examples to showcase the effectiveness of the implemented safety function blocks and their compatibility with various industrial environments.

The final chapter presents the conclusions derived from the development of this dissertation and suggests future directions to further enhance the work presented in this document.

Chapter 2

Context of the dissertation

This contextualizing chapter starts by revealing the motivation for the dissertation; departing from here, the major aims of the dissertation and its expected practical benefits are stated; lastly, it justifies the major options and orientation lines adopted in its development at the light of the related state of the art. By following this approach, it is expected that by the end of the chapter a contextualizing view of the development has been transmitted, as well as of the developments to be conducted in subsequent chapters.

2.1 Bridging safety of machinery and PLC programming

The half-year curricular unit “Machinery Safety” (MS) appears in the Mechanical Engineering master’s course in the first semester of the second year. This unit aims to convey advanced core safety of machinery methodologies to students who most likely will have future responsibilities in the area. Teaching these subjects efficiently is not possible without some practical lab activities to be carried out so the students can obtain feedback on their grasp of the subject. In such a manner the unit needs a platform capable of designing safety systems and test them.

Safety programmable logic controllers are now replacing the classical safety relays and thus being increasingly used for safety in industrial applications (6). For this reason, PLC programming is definitely a valuable and even central skill for MS students. In this regard, “Industrial Computing” (IC) is taught simultaneously for most MS students. IC aims to prepare industrial automation students in computer systems integration, including advanced PLC programming. IC is a classic unit in the Mechanical Engineering course and thus much older than MS. Yet, up to now, safety PLCs programming hasn’t been considered a major topic of interest of IC. As a result student’s familiarity with practical solutions for safety of machinery has been very reduced – just a few basic concepts in the bachelor course.

With these two units appearing now in parallel for most students, the past situation has changed, and subjects taught in PLC programming classes can be applied in safety of machinery classes and vice versa. Therefore, it is in the best interest of the teachers and students to complement these two subjects as much as possible. Correlating problems from both curricular units will help educators

to explore the subjects deeper, allowing the students to have a more complete understanding of machinery safety systems and a broader practice of PLC programming, ultimately training future engineers in the safety of machinery sector. A solution would be practical lab activities common to both curricular units where students can practice designing safety systems for machines and program the relevant programmable automats. The present dissertation intends to contribute for that.

2.2 FACTORY I/O: the promising, yet embryonic, bridging link

Practical training is essential for engineers in automation and safety of machinery fields, as it provides the necessary knowledge and skills to design and implement safety-critical systems found in modern manufacturing. Passive (hands-off) training is a common form of learning safety of machinery, such as videos or lectures with no feedback (7; 8). To better train engineers in the design of safety systems a really deep practical approach is crucial. Yet, this type of training is challenging and a hands-on approach can be very risky; for instance, allowing learners to interact with hazardous equipment can cause serious injuries and there is also an investment risk of damaging the equipment itself (9; 10). Training advanced solutions in industrial automation also faces these challenges. However, “Industrial Computing” have solved them with the use of Factory I/O in classes and practical activities concerning distributed PLC programming, visualization, Supervisory Control and Data Acquisition (SCADA) systems design and industrial networking. Factory I/O is developed by Real Games and is a state of the art application for developing simulation environment. Through a user-friendly interface, Factory I/O allows for simulating industrial scenarios, machine components, and injection of faults, provides students with a safe, controlled environment to actively test industrial automation solutions. Figure 2.1 shows this environment with some components (11).



Figure 2.1: Factory I/O environment (1)

Factory I/O users can connect the simulation to an external controller, through built-in I/O drivers, used with certain communication technologies to map sensor and actuator tags to the driver I/O points (12). This is a cheap and very practical solution when used with external PC-based controllers and SoftPLCs such as Codesys, widely used in IC classes, bypassing this way the use of expensive and wiring and space demanding physical PLCs.

Due to all these features, Factory I/O is undoubtedly a cheap, safe, and promising tool in the industrial safety context and nullifies the risk involved in training safety-related tasks. Yet, currently, Factory I/O is not particularly focused in machinery safety features nor includes major safety components. However, it is far from being completely useless in this matter, parts like emergency buttons, safety doors with safety switches, and simple light arrays used as safety light curtains, are sufficient for introducing, developing and testing simple safety solutions (13). Certainly, Factory I/O has the possibility of thriving in other important areas besides standard industrial process control.

2.3 Finding an early solution for functional safety development and testing

One of Real Games' priorities is, in fact, making Factory I/O a valuable tool for safety-related training. However, being this a very challenging and time consuming project, a Factory I/O version embracing *de facto* safety of machinery issues and components will not be available in the short time. Therefore, other solutions for teaching, developing and testing safety systems in early stages have to be adopted. These solutions are, essentially, collections of software objects simulating safety devices and safety logical components to be developed over the best softPLC technology currently available.

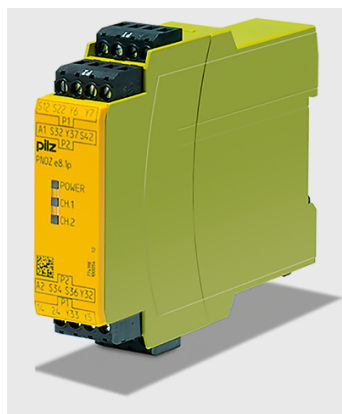
2.3.1 Current functional safety solutions and practices

Machines have specific safety components, such as safety sensors and switches, interlocks, and emergency stop buttons. Depending on the machine and application, these components are designed to monitor various conditions, such as presence, position, speed, temperature, pressure, or voltage.

Safety devices in a machinery safety system are specialized and distinct from standard process control devices due to their primary focus on personnel safety, compliance with safety standards, redundancy, functional independence, specific safety functions, and assigned safety integrity levels. These features collectively ensure a higher level of protection and risk reduction in industrial environments (14).

Functional safety devices are wired to a safety control system, often called safety logic or safety controller. The safety logic receives input signals from the safety components and applies predefined safety functions to determine the appropriate response in a hazardous situation.

The safety controller can be hardwired, like safety relays or safety contactors and, in this case, has a certain limited and configurable functionality (15). The proper safety reaction is determined by internal circuits as a safety relay checks the state of safety devices. When necessary, it triggers output signals to stop the machinery or start precautionary measures (15). To increase reliability, safety relays frequently include redundancy and self-monitoring functions (15). On the other hand, for complex solutions the safety controllers can be programmable, like safety PLCs (15; 16). These controllers combine the functionality of traditional PLCs with safety-specific features. These features include meeting the stringent safety requirements needed for safety-critical tasks, a higher level of reliability, fault tolerance, and compliance with safety standards and regulations allowing for the flexible configuration and integration of safety functions (15; 16). Figure 2.2 displays a safety relay and a safety PLC, and at first glance, certain distinctions are already evident.



(a) Safety Relay by Pilz (17)



(b) Safety PLC by Eaton (18)

Figure 2.2: Hardwired vs programmable logic

The safety logic continuously monitors the inputs from the safety components to detect any deviations or abnormalities that could pose a risk. For example, if a safety sensor detects an operator's hand entering a hazardous area, the safety logic can initiate a safety response to stop the machine.

In many cases, the safety logic is integrated with the overall control system of the machine as shown in Figure 2.3. This integration allows the safety logic to communicate with other operational functions and coordinate their activities and ensures that the machine operates safely while maintaining productivity and efficiency.

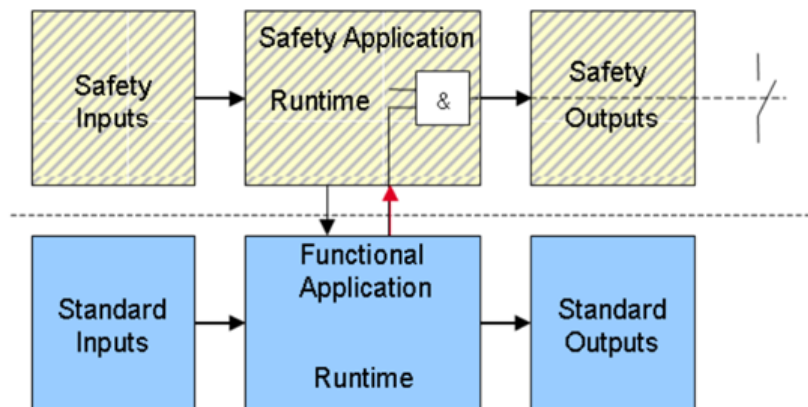


Figure 2.3: Integration of safety logic and overall control system (2)

2.3.2 Adopted solution

A solution found for developing and testing safety systems is to adopt an integrated development environment such as Codesys for simulating the functionality of the system virtually (19).

CODESYS is a modern, free software development platform for industrial automation. It is compatible with the IEC 61131-3. The IEC 61131-3 standard provides a common framework for programming industrial control systems, ensuring consistency and interoperability across different devices and manufacturers. It offers a range of programming languages, promoting modular and standardized development practices in the automation industry. Additionally, CODESYS offers visualization features to enhance the user interface and facilitate interaction with programs. Moreover, it provides advanced communication capabilities, enabling seamless integration and communication with other software systems, devices, and protocols.

CODESYS is an appealing solution due to its capability to seamlessly interact with Factory I/O. By establishing connections between input and output variables from the program and mapped sensor and actuator tags in Factory I/O, CODESYS enables the exchange of data through the machine-to-machine communication protocol OPC UA. This integration allows for the visual testing of developed systems in a secure and controlled environment, providing an added layer of assurance.

Using Codesys makes possible to simulate the safety components through function blocks (FB). The FBs should include I/O variables to simulate electrical signals and contacts. A functional model can be created to simulate the functionality of the components realistically considering inputs and outputs, and a fault model can be implemented to simulate specific faults that affect the functionality of the components and their outputs. The result of this development will be a library of FBs that realistically simulates the functionality of safety devices.

To be more user-friendly and be able to observe the designed system functioning and test it, visual representations of the components can be modelled using image pools of the physical or symbolic structure, furthermore, these interfaces will include manual fault injection and manual command of the devices if necessary.

For the development of the safety logic, standards and requirements must be followed to ensure safety and to enable the logic to be used in real-life situations. The result of this development will be a library of safety function blocks with easy and reliable applications for safety PLCs.

2.4 Standards and practices to consider and include in Safety PLC programming

The rapid advancement of technology has led to a growing number of PLC manufacturers to offer a wide range of equipment brands and models (20). However, users of PLCs often faced the challenge of incompatible communication among PLCs and various programming languages (20). To address this issue, the IEC 61131 standard was established with the aim of transforming this landscape (20). PLCopen is an internationally recognized independent organization that actively contributes to the standardization of PLCs (21). Its valuable contributions include certification, library development, and creation of function blocks (21). By actively engaging with manufacturers and users, PLCopen plays a pivotal role in advancing the standardization of PLCs.

Developing safety functions FBs is a meticulous process due to the numerous safety standards and government requirements available (2). In this regard, PLCopen established safety-related features inside the IEC 61131-3 development environments in collaboration with its members and outside safety-related organizations (2). According to PLCopen (22), these features help to integrate safety-related functionality into developed systems from the early stages of the development cycle and include:

- Representation of the software architecture
- Definition of programming languages
- Definition of safety-related data types
- Definition of user levels for easy programming and error prevention
- Concept of error handling and diagnostic
- Definition of a generic safety-related function block and a set of safety-related function blocks
- Definition of a procedure for PLCopen compliance procedure.

Figure 2.4 shows for each Safety FB in the “PLCopen Safety Software Technical Specification” the definitions provided (3):

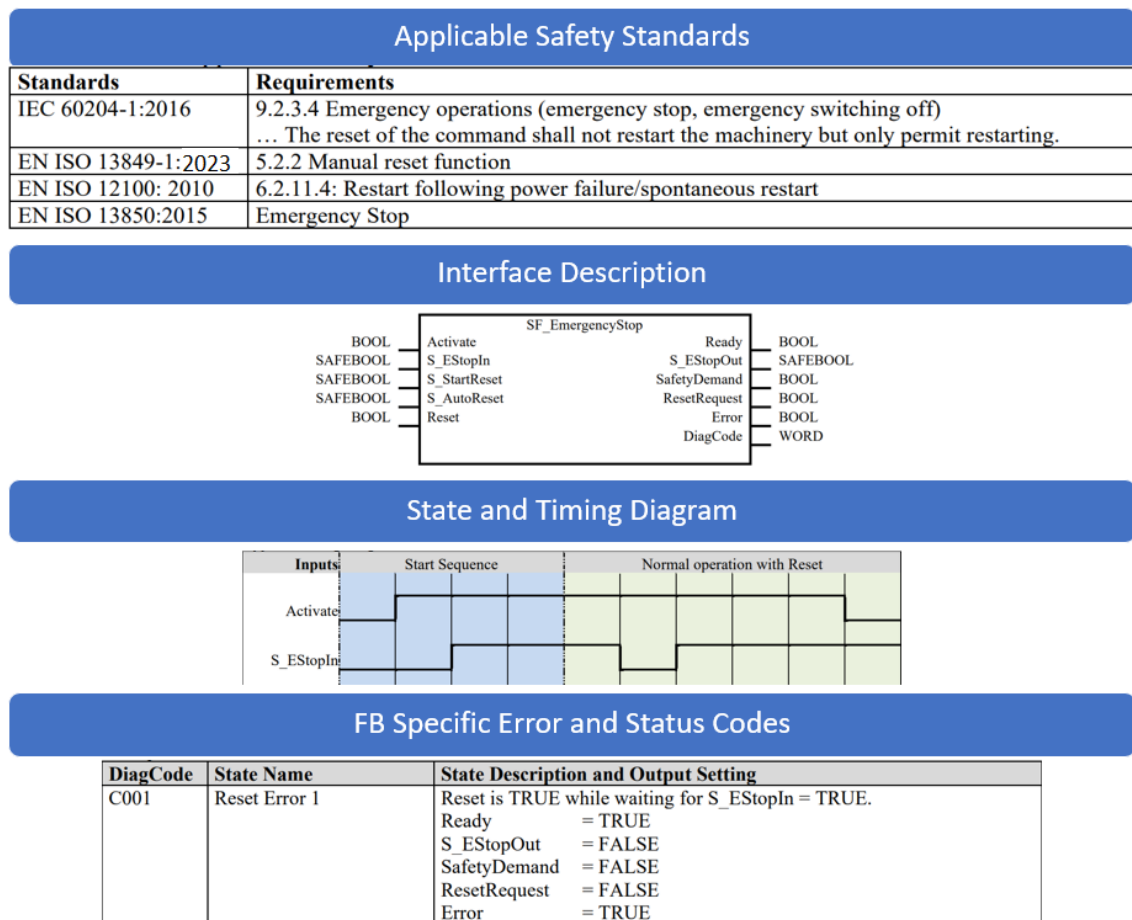


Figure 2.4: Safety FB definitions in PLCopen Technical Specification (3)

This is an attractive approach and seems to be logical to adopt because it eases the process of respecting the relevant safety standards. Furthermore, it is an advantage that more automation brands are increasingly adhering to it.

2.5 Summary and steps to complete

In this chapter, the context of the dissertation was conveyed. The benefits were presented, and it is concluded that the bridging of industrial computing classes and safety of machinery classes will better prepare automation students in future for future roles in the area.

Having an easily controllable virtual environment to develop and test safety of machinery solutions is a safe, cost-effective solution for training in these subjects. Factory I/O is on the right track to becoming a viable solution, but it still has a long way to go. For this reason, and for now, simulating safety solutions in an integrated development environment, with the aid of some visual interfaces, is sufficient.

Adopting the PLCopen safety specifications will help ensure compliance with the necessary standards and requirements to guarantee safety in the control of safety systems. Additionally, the

development of a library of safety function blocks will enable the creation and testing of various safety scenarios.

The next chapter presents the approach adopted to simulate a simple safety solution focused on a safety relay that later gives way to more advanced, standardized solutions.

Chapter 3

Simulation of a safety relay application

As mentioned in the previous chapter, safety relays are being used for logic in safety systems for many years now. A safety relay is selected for a given safety application based on its functionality and its Safety Integrity Level (SIL) (23). For example, for a higher SIL, a safety relay would need to incorporate certain characteristics, such as (23):

- Redundant elements to enhance reliability;
- Augmented diagnostic coverage to detect faults;
- Fault tolerance to make sure that it maintains a safe operation;
- Advanced comparison logic improving the system’s ability to detect faults.

This means that even a small safety relay encapsulates a lot of logic (functionally), and thus is far from being a “trivial” component. Moreover, to perform a safety function, the relay needs input signals from safety devices to detect dangerous situations and drive output devices to act after decision-making, while input and output device failures have to be detected and cannot compromise systems safety.

Departing from this, the idea of starting this project by simulating a functional safety application based on a safety relay came naturally. Simulating a flexible safety relay that includes a set of advanced features is probably the best way to get familiar with the software and design aspects of functional blocks to be included in a safety PLC. On the other hand, modelling input and output safety devices that can fail in some ways is also central to understanding those components and their role in the application.

The design and proper interaction of basic hardwired safety components is thus the subject of the present chapter. The employed PLC programming language is structured text (ST). ST is beneficial and seems only obvious for programming function blocks because it allows for the implementation of complex logic, facilitates code modularity and reusability, and provides strong data typing and readability, resulting in more robust and maintainable function block code. Here, and not surprisingly, the proper use of modern PLC programming features - like object-orientation and integrated visualization as found in Codesys – are central to the development.

3.1 Functional safety based on safety relays

One of the most common safety solutions is using a safety relay to stop a machine when a dangerous situation is detected. The safety relay is responsible for monitoring the emergency stop button. In the event of the emergency stop button being activated, the safety relay performs a safety-oriented action by opening the enabling circuits and deactivating the power safety contactor. After the emergency stop command device is disengaged and the feedback circuit is closed, the Start button can be utilized to reactivate the system. This is precisely the functional safety solution simulated in this early phase of the project: a safe conveying application that includes a safety relay, an emergency stop button, and a safety contactor. Figure 3.1 shows a schematic representation of this safety solution to be modelled.

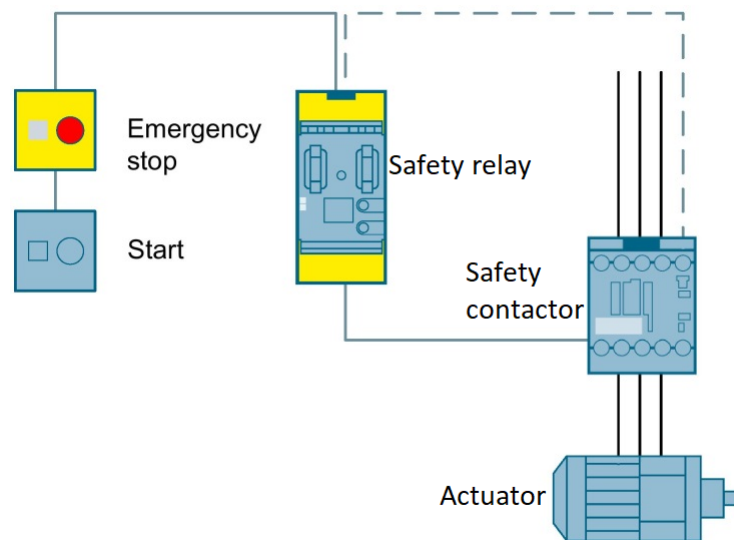


Figure 3.1: Emergency stop functional safety solution example. Adapted from: Siemens support website (4)

3.1.1 Functionality of the simulated safety relay

The simulated safety relay is based on the PNOZ s5 relay by Pilz and has an instant (non-delayed) enabling output circuit and a delayed output enabling circuit (5). Two input circuits are integrated and operated separately for redundancy. The relay has different operation modes and can detect faults that could cause dangerous situations if not handled correctly.

The block diagram of the PNOZ s5 safety relay and its front view can be seen in Figure 3.2. Figure 3.3 displays an example of how the wiring can be done for an emergency stop application along with a representation of the rotary switches and LED lights in the front of the safety relay.

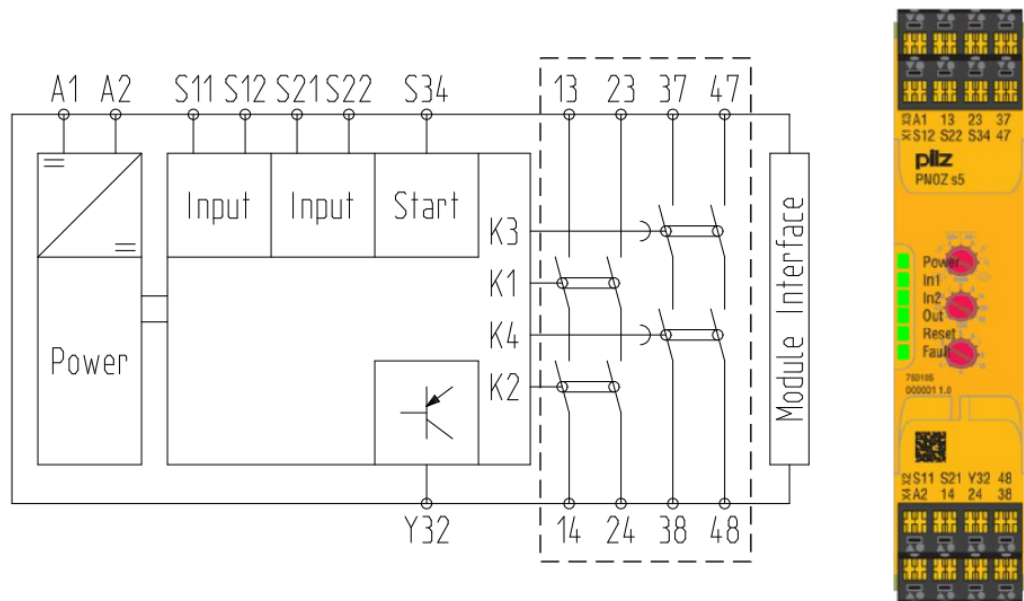


Figure 3.2: Block diagram and front view of the PNOZ s5 safety relay. Adapted from, Pilz PNOZ s5 Operating Manual(5)

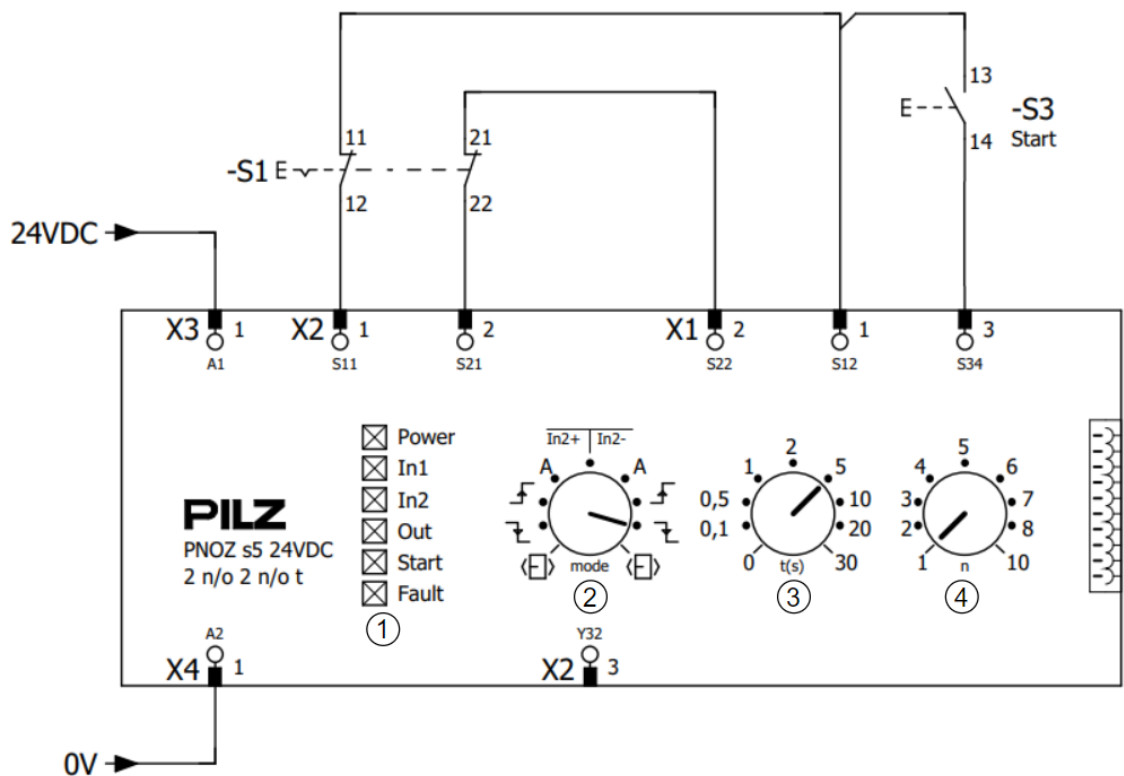


Figure 3.3: Wiring diagram with rotary switches and LED lights representation. Adapted from Pilz PNOZ s5 Operating Manual(5)

Table 3.1 is a legend of the safety relay front view selectors - Figure 3.3.

Table 3.1: Front view of safety relay legend

N°	Description
1	LED lights, signal the status of the safety relay.
2	Mode selector. In2+ = Single-channel input. In2- = Dual-channel input
3	Defines timed output delay, 0-30 s
4	Selector of integer number to multiply the time delay, for delay time higher than 30s.

The relay is powered at 24V DC in terminal A1 and 0V DC in A2, turning on the Power LED. Terminal Y32 is used to transmit a signal to a standard controller, reporting the safety relay status, if required. When the device is active, it indicates that the K contacts are closed. This allows the flow of power from terminals 13, 23, 37, and 47 to terminals 14, 24, 38, and 48 respectively (refer to Figure 3.2). The activation of the device requires both input circuits to be active, meaning the S1 emergency button contacts must be closed, and a Start signal must be present at terminal S34, as illustrated in the timing diagram depicted in Figure 3.4. In the event of a hazardous situation where any input signal is lost while the device is active, contacts K1 and K2 immediately open, followed by contacts K3 and K4 opening after a certain duration.

Referring to Figure 3.3, the "In1" and "In2" LEDs indicate the status of the input circuits. The "Out" LED reflects the state of the K contacts. The "Start" LED illuminates when an input is present at terminal S34 for the start circuit. Lastly, the "Fault" LED activates when a fault is detected.

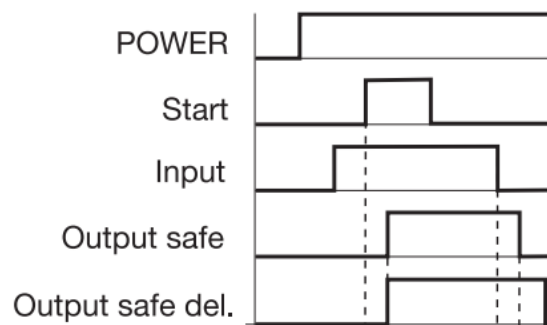


Figure 3.4: Timing diagram of relay activation. Adapted from Pilz PNOZ s5 Operating Manual(5)

In the input circuits, signals are emitted from terminals S11 and S21 which are expected to be received back in terminals S12 and S22, respectively. Figure 3.5 shows a representation of the input circuits.

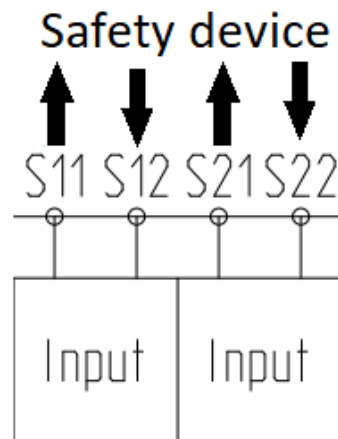


Figure 3.5: Safety relay input circuits

It is in these terminals that the safety input switch devices are wired. A safety switch device basic functionality is receiving an input signal and through internal circuit let the signal pass or not to the outputs. In this case the S1 emergency button is a switch device that receives a signal from the safety relay and pressing the button opens internal contacts and stops the signal from returning to the safety relay input.

Figure 3.6 shows two ways the wiring can be done depending on the type of input device used, single-channel (if the emergency button has just one contact) and dual-channel (if the emergency button has two contacts).

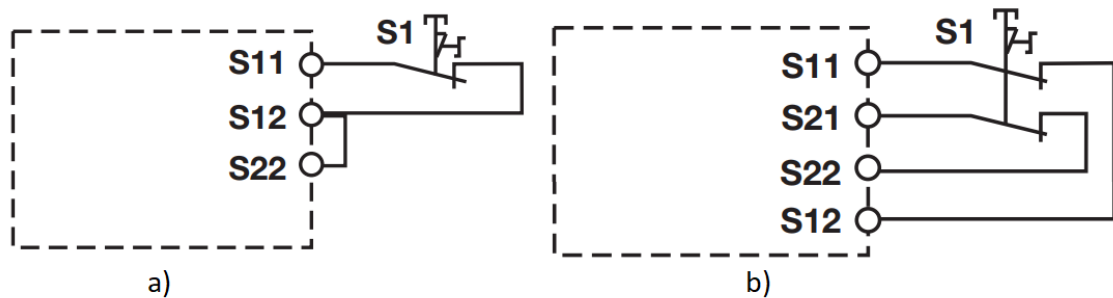


Figure 3.6: a) Single-channel input circuit b) Dual-channel input circuit. Adapted from: Pilz PNOZ s5 Operating Manual(5)

Safety switch devices can have faults like contacts welding or getting stuck in one position. The safety relay can detect this by comparing the state of the two input circuits. To distinguish between each input circuit, test signals (T1 and T2) can be utilized. These signals are generated from terminals S11 and S21 respectively. They are pulsating signals that are inverted to each other, allowing the safety relay to differentiate between the signals it receives. Figure 3.7 shows a representation of the test signals.

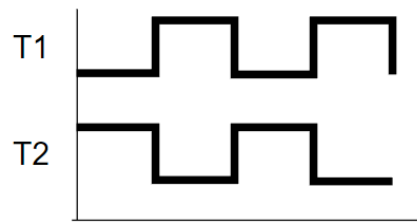


Figure 3.7: Example of T1 and T2 test signals

If the input circuits are not operated simultaneously an error is detected. Another possible fault is a short circuit between the switch device channels. If the wrong signals are received in the input terminals an error is detected. For example, the signal in the S11-S12 circuit "leaks" to the S21-S22 circuit, as shown in Figure 3.8.

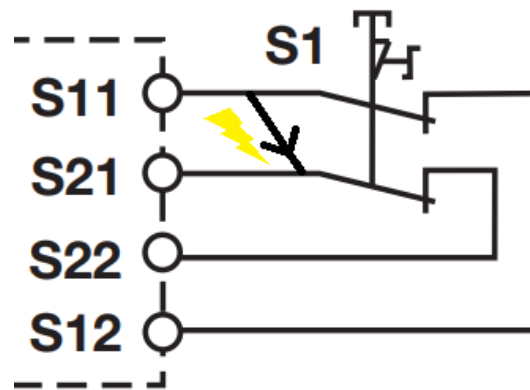


Figure 3.8: Representation of a short circuit across contacts

The delay time for the contacts in the output enabling circuit to open is selected on the relay itself with rotary switches, and there is also a switch to select the operation mode, shown in Figure 3.3. The operation mode changes the expected input circuit, "In2+" modes are for single-channel circuit and "In2-" modes are for dual-channel circuit. In these two groups of modes there are three modes that change the input Start signal to activate the device. It can be a rising edge signal, a falling edge signal or an automatic signal. Figure 3.9 shows the rising edge signal start and a falling edge signal start.

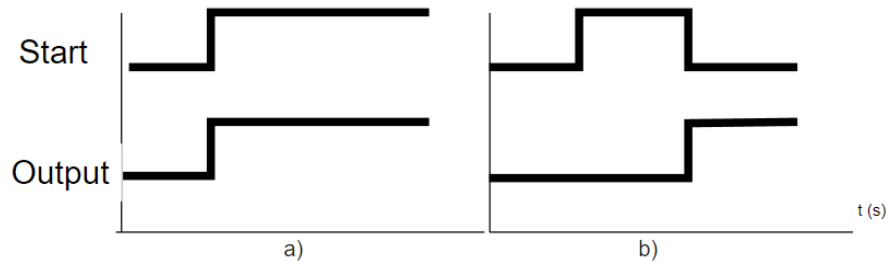


Figure 3.9: a) Rising edge start b) Falling edge start signal

3.2 Software simulation of PNOZ s5 relay

By decomposing the functionality of the PNOZ s5 safety relay one finds that it includes four major features that are worth programming in separate function blocks (FBs). These functional parts are:

- Operating mode selection
- Input monitoring
- Start operation
- Output enabling

The "Mode_FB" function block is responsible for setting the operating mode of the safety relay upon power-up. It detects the "Power On" input and adjusts the internal variable to reflect the desired operation mode.

The "Equivalent_FB" function block monitors the operation of the input circuits and generates an "InputOK" signal, which is used by the output enabling function block.

The "Start_FB" function block initiates the start operation and generates a "Start" signal that activates the output enabling function block, allowing it to close the output circuits.

Lastly, the "EStop_FB" function block is responsible for enabling the output circuits of the safety relay. It requires both an "InputOK" signal and a "Start" signal to activate and allow the output circuits to function.

So, the safety relay was designed as a FB encapsulating all the FBs mentioned above as shown in Figure 3.10.

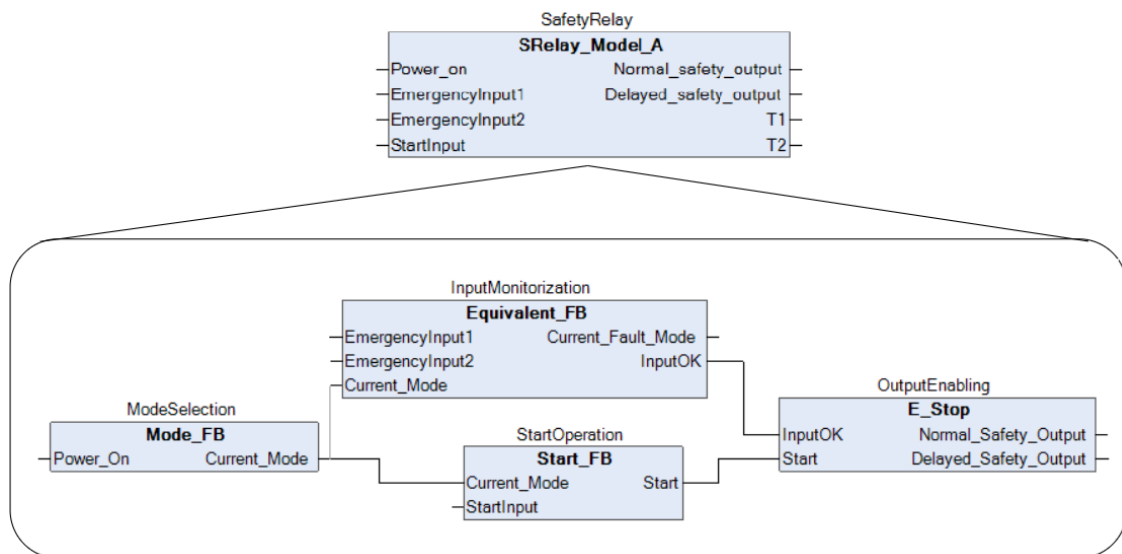


Figure 3.10: Safety relay FB and encapsulated features

The development starts by forming a base class, `SRelay_Base_Class`. So the class for the simulation of the PNOZ, and any other possible future safety relays, extend from this class to incorporate common variables and data types preventing repeated code. Table 3.2 presents the generic variables and data types of any safety relay to be developed.

Table 3.2: Common variables in a safety relay class

Name	Data type and description
Configuration Parameters	
Max_Fail_Mode	SRelay_Fault_Modes. Identifies the possible faults that can be detected. Should be defined when configuring a new safety relay.
Input Variables	
Power_On	BOOL. Signal to activate the safety relay. When TRUE relay is active.
EmergencyInput1	BOOL. Input signal, circuit 1.
EmergencyInput2	BOOL. Input signal, circuit 2.
StartInput	BOOL. Input signal in the reset circuit.
Output Variables	
Led_1, Led_2, Led_3, Led_4, Led_5, Led_6	BOOL. State of interface LED lights.
Instant_Safety_Output	State of a instant output circuit. When TRUE circuit is closed.
Delayed_Safety_Output	State of a timed output circuit. When TRUE circuit is closed.
Internal Variables	
Current_Fault_Mode	SRelay_Fault_Modes. Detected fault in the safety relay.
Start	BOOL. Start signal to enable the output circuits.
InputOK	BOOL. Signal that indicates the input circuits are closed.

Enumeration data types are used in the safety relay simulation programming as they provide a structured and readable representation of specific sets of options or states within the code. By using enumeration data types, the program's data options are clear and consistent, preventing errors due to invalid values. Enumerations enhance code clarity and maintainability by providing a straightforward and standardized way of representing and handling different options or states. For example a USINT data type enumeration, `SRelay_Fault_Modes`, is used for any safety relay fault modes, i.e. the faults that can be detected. This way the system can easily handle and track the fault detected through a list of valid options, allowing for efficient control and management of system behavior, Figure 3.15 show this data type.

```
TYPE SRelay_Fault_Modes :
(
    None := 0,
    Internal_Fault,    //Random fault in the relay, no specific cause
    Input1_Fault,      //Fault in input circuit 1
    Input2_Fault,      //Fault in input circuit 2
    InputCircuit_Fault,//Fault in both input circuits
    Mode_Fault         //Fault related to mode selection
) USINT := None;      //Default data value
END_TYPE
```

Figure 3.11: `SRelay_Fault_Modes` data type

Methods are used throughout the safety relay simulation programming; namely for executing secondary sets of instructions such as LED control and test signal generation. Firstly, methods allow for code modularity, enabling complex tasks to be broken down into smaller, manageable units. This modular approach not only enhances code reusability and readability but also reduces code duplication and promotes efficient code maintenance. Secondly, methods encapsulate a set of instructions within a defined scope, facilitating better organization. This encapsulation ensures that the internal implementation details are concealed, and only the necessary inputs and outputs are exposed, promoting clean and clear code design.

For the complete functionality of the simulated devices some internal variables are commanded through the visual interfaces and for this, properties are used. Properties have a “Get” method that reads the value selected in the element in the interface and a “Set” method changes the value of a variable in the program. This is useful because these methods can do extraneous operations with the value from the visual element before returning it to the program.

When decision making is involved the `CASE OF` instruction is employed for being useful for handling multiple conditional branches based on a single variable or expression. This promotes code maintainability as it provides a clear structure to handle various conditions and facilitates future modifications or additions to the decision logic, such as data type conversion.

3.2.1 Safety relay FB

The safety relay FB encapsulates the functionality of the safety relay, represented in Figure 3.10, and executes the control of the interface display. The function block is then connected to input and output devices and can be seen as a representation of Figure 3.2. Figure 3.12 shows the FB with its inputs and outputs. The `Power_On` variable activates the rest of the functionality and represents the A1-A2 circuit from Figure 3.2. The `T1` and `T2` outputs represent the test signals from the input circuits to be connected to a safety switch device and the `EmergencyInput1` and `EmergencyInput2` variables are the returning signals which are used in the input monitoring FB.

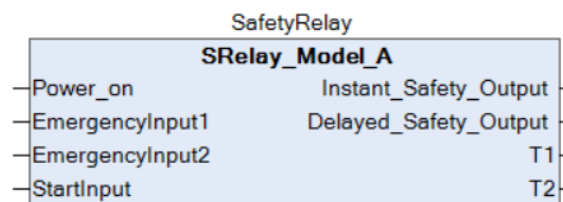


Figure 3.12: SRelay_Model_A FB

A method is used to encapsulate the instructions that generate the output test signals `T1` and `T2`, this way by calling the method the block of code is executed concealed from the main code. In practical applications the signal pulsates very quickly, in very few milliseconds. But in Codesys the signal is limited by the cycle time of the PLC scans. So to simulate the test signals a simple procedure changes the signals value every PLC scan. The result is a signal that pulsates every 20 ms.

The safety relay FB is also responsible for controlling the LED lights of its visual representation using a set of instructions encapsulated in a method `Control_LED` that sets the LED variables to off, on or blinking. The method takes an input which is the intended LED mode and an output has to be associated which is the LED variable. The LED operation modes are listed in a `USINT` data type enumeration, `Enum_Led_mode`. Depending on the fault modes that the safety relay can be in, a `CASE OF` instruction sets the LED lights variables. Figure 3.13 shows one of the `CASE` statements, in this case an `Input1_Fault` is detected, that is to say the input circuits didn't operate simultaneously and the first input circuit is the one that was stuck. `Led_2` controls the "In1" LED and `Led_6` controls the "Fault" LED, as seen in Figure 3.3. When this fault occurs the "Fault" LED lights up and the "In1" blinks.

```

CASE Current_Fault_Mode OF
    Srelay_fault_modes.Input1_Fault:
        Control_led(mode := Enum_Led_Mode.Blink, state=>Led_2);
        Control_led(mode := Enum_Led_Mode.ON, state=>Led_6);
END_CASE

```

Figure 3.13: CASE statement of `Input1_Fault` to control the LED lights

Two properties are used to change the safety relay operation mode and the output delay time internal variables through the visual interface. A "Get" method reads an INT value from the delay time selector in the visual interface and writes it in the `Delay_Time_Set` property and the "Set" method converts the value to TIME data type and writes in the delay time variable. Through a property `Mode_Change` a "Set" method writes the value of the mode selector in the selected mode variable. Figure 3.14 shows the `SRelay_Model_A` FB and its methods. `FB_INIT` is a function block initialization method used to set the initial values and configurations of a function block before it starts executing its main logic.

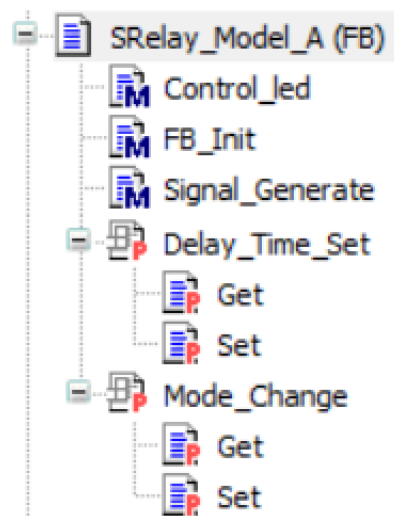


Figure 3.14: `SRelay_Model_A` FB and its methods

3.2.2 Mode Selection FB

For the PNOZ simulation, a `USINT` data type enumeration, `Model_A_Op_Modes`, is used for the possible operating modes. The use of the enumeration ensures that only one valid operation mode can be assigned, eliminating the possibility of errors due to incorrect or undefined modes. Figure 3.15 show this data type.

```

TYPE Model_A_Op_Modes :
(
    Mode_1 := 0,    //Automatic start single-channel
    Mode_2,        //Rising edge start single-channel
    Mode_3,        //Falling edge start single-channel
    Mode_4,        //Automatic start dual-channel
    Mode_5,        //Rising edge start dual-channel
    Mode_6         //Falling edge start dual-channel
) USINT;
END_TYPE

```

Figure 3.15: `Model_A_Op_Modes` data type

The operating mode of the safety relay is meant to be selected through a visual interface. The functionality of the mode selection is done through the `Mode_FB` function block, the input and output variables can be seen in Figure 3.16.

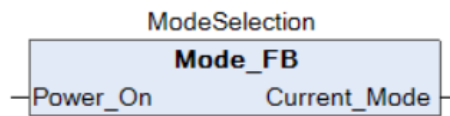


Figure 3.16: Mode_FB, inputs and outputs

The internal variable `Selected_Mode` is adjusted through the visual interface, and the variable `Current_Mode` is the locked operating mode of the safety relay, these are variables of `Model_A_Op_Modes` data type.

For this functionality in the initial moment when power is provided to the relay this FB makes the `Current_Mode` take the value of the `Selected_Mode`. The `Current_Mode` variable is then passed to the other FBs. Changing the selected mode mid-operation will immediately result in a `Mode_Fault` error because this action is only possible when power is off. This FBs programming is presented, as explained, in Figure 3.17.

```

Power_On_Trig(CLK := Power_On); //Rising edge of Power_On
IF Power_On_Trig.Q THEN //When power changes from Off to On
    Current_Mode:= Selected_Mode;
END_IF
//Change mode mid-operation
IF Power_On AND (Current_Mode <> Selected_Mode) THEN
    Current_Fault_Mode:=SRelay_Fault_Modes.Mode_Fault;
END_IF

```

Figure 3.17: Mode_FB function block programming

3.2.3 Input device monitoring - Equivalent_FB

In the input circuits signals pass through safety devices and return to the safety relay already shown in Figure 3.5. This FB simulates the input circuits functionality which is to monitor the signals received from the safety switch devices. The input and output variables used to simulate the input monitoring are shown in Figure 3.18.

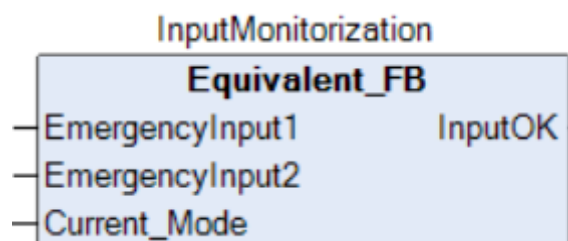


Figure 3.18: Equivalent_FB, input and outputs

The operating mode defined in mode selection FB will dictate what type of wiring is done in the input circuit. In this FB `EmergencyInput1` and `EmergencyInput2` are the outputs of the safety switch device connected to the safety relay.

To detect faults in the input circuit a method, `FaultDetect`, runs a set of instructions. The functionality is commonly present in safety programming. This method uses comparison logic to detect errors with the inputs and change the `Current_Fault_Mode`. Both inputs must change simultaneously, within a margin of 100 milliseconds in this case, if one input circuit is active and the other is inactive an error occurs resulting in a `Input1_Fault` or a `Input2_Fault`. If any input receives the wrong test signal another error occurs resulting in a `InputCircuit_Fault`. Figure 3.19 shows the input monitoring FB and associated methods.



Figure 3.19: Equivalent_FB and its methods

In addition to detecting faults the functionality of this FB is generating a `Input` signal to pass to the output enabling FB signaling that the input circuits are closed. When the safety relay is powered all faults are reset and will be detected again if they persist. In the first three operation modes, which are associated with a single-channel input circuit, both inputs are compared with the T1 test signal, and in the other modes `EmergencyInput1` is compared with the T1 test signal and `EmergencyInput2` is compared with the T2 test signal representing a dual-channel circuit. If the comparisons are true the output `Start` signal is enabled. `Current_Fault_Mode`. The programming was made according with the explained functionality and can be seen in Figure 3.20.

```

Fault_Detect(); //Compares input with T1 and T2
CASE Current_mode OF
model_a_op_modes.Mode_1, model_a_op_modes.Mode_2,
model_a_op_modes.Mode_3:

    Start := EmergencyInput1 = T1 AND EmergencyInput1 = T1;

model_a_op_modes.Mode_4, model_a_op_modes.Mode_5,
model_a_op_modes.Mode_6:

    Start := EmergencyInput1 = T1 AND EmergencyInput1 = T2;

END_CASE

```

Figure 3.20: Equivalent_FB code

3.2.4 Start operation

After monitoring the input channels the safety relay needs a start signal to enable its output circuits. Figure 3.21 presents the inputs and outputs for the Start action.

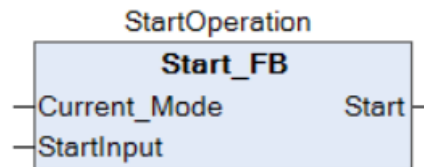


Figure 3.21: Start_FB, inputs and outputs

To generate the `Start` signal for the Output enabling FB the necessary input signal in the `StartInput` depends on the operation mode. There are three possibilities, one is simply a static `TRUE` signal in the `StartInput` input which allows for the automatic start of the relay. The other two options are the start command to happen when there is a rising edge signal or a falling edge signal in the `StartInput` terminal as already seen in Figure 3.9.

A `CASE OF` instruction is used to run a statement that sets the start signal according to the operating mode variable, `Model_A_Op_Modes`, as listed in Figure 3.15.

3.2.5 Output enabling – E_Stop

The output enabling FB is dependent on the other functionality FBs. The function block `E_Stop` can be defined as the action of enabling or disabling the output circuits of the safety relay depending on the `Input` signal and the `Start` signal, as shown in Figure 3.4. Figure 3.22 presents the inputs and outputs of `E_Stop`.

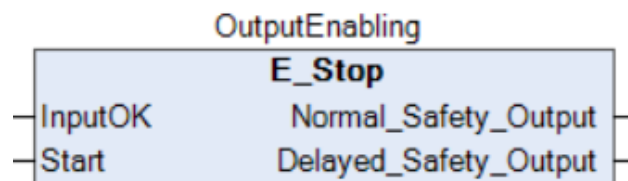


Figure 3.22: E_Stop FB, inputs and outputs

A latch mechanism is utilized to control the outputs of the safety relay. When the relay is powered and both an `Input` signal and a `Start` signal are detected, the latch output switches to true and remains in that state until the `Input` signal is lost. The true latch output enables both safety relay enabling circuits. Conversely, when the latch output is false, a `TurnOff_delayed` method is triggered. This method immediately disables one of the outputs and initiates an off-delay timer to disable the other output after a certain time period.

Figure 3.23 shows the programming of the FB as described, while Figure 3.24 provides a visual representation of the `E_Stop` FB and its associated method.

```

Output_Actuate
(Set := Start AND Input, Reset1 := NOT Input);

Instant_Safety_Output := Output_Actuate.Q1 AND Power_On;
Delayed_Safety_Output := Output_Actuate.Q1 AND Power_On;

IF NOT Output_Actuate.Q1 THEN
    TurnOff_Delayed();
END_IF

```

Figure 3.23: E_Stop Function Block

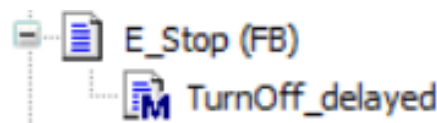


Figure 3.24: E_Stop associated Method

3.3 Peripheral devices simulation

To perform the safety function, the relay needs input signals from safety devices. It was already discussed how the safety relay detects faults in the input switch devices. The faults have to be manually injected in the simulated safety devices. Another requirement is the possibility of manually commanding the devices in a visual interface when necessary.

The process of simulating the peripheral safety devices starts with a base class for which all the devices will be based on, `Switch_Base_Class`. Every new device class developed extends from this base class, this way variables common to all switch devices don't have to be defined again. Table 3.3 shows the configuration variables in the base class used to initialize every new switch device on a program and the internal variables common to all switch devices.

Table 3.3: Configuration variables for safety switch devices

Name	Data type and description
Configuration Parameters	
Switch_Class	STRING. Identifies the type of switch device.
Switch_Suffix	STRING. Identifies the particular device.
Has_Multiple_Views	BOOL. Configures the number of views of the visual display of the device. When TRUE has a physical and a symbolic view.
Is_HMI_Commandable	BOOL. Configures if the device can be actuated in the visual interface. When TRUE allows operation through the interface.
Max_Fail_Mode	Switch_Fault_Modes. Selects the possible fault modes for the device among the listed modes in the enumeration.
Internal Variables	
Bitmap_Image	STRING. Name of the displayed image in the image pool.
Switch_image_pool	STRING. Name of the displayed image pool.
Current_Bitmap_Image	STRING. Name of the displayed image pool and image combination.
Current_view_is_physical	BOOL. Displayed type of view in the moment. When TRUE displays the physical view
Current_Fault_Mode	Switch_Fault_Modes. Current fault mode.

The switch device offers different operational modes, which are defined in an enumeration data type called `Switch_Fault_Modes`. These modes correspond to various faults that the device can experience.

Each switch device is equipped with a visualization interface that depicts the device's current state. The visual representation is based on two image pools: one for physical views and another for symbolic views. The interface also includes a component for selecting the desired type of view to display. To handle this functionality, there are two methods associated with property changes.

The "Get" method reads the selected display mode and writes the value in the "Display_Mode" property. The "Set" method writes the property value to the "Current_view_is_physical" variable. This variable determines whether the physical view or symbolic view should be displayed.

Figure 3.25 provides an illustration of the display element featuring both types of views and the view selector element.

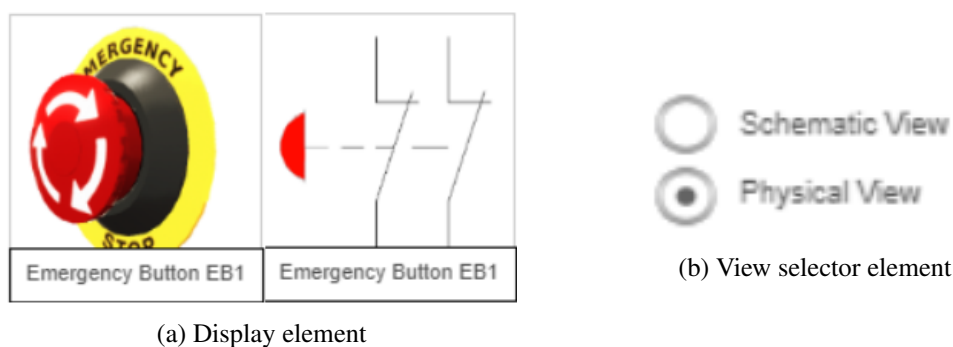


Figure 3.25: Interface display and view selector element of the emergency button interface

To select faults to inject in the system, a combo box element in the interface lists the possible faults allowing to select one of them. A "Get" method reads the selected fault mode and writes the value in a property "Injected_Fault" and a "Set" method writes the property value on the Current_Fault_Mode variable.

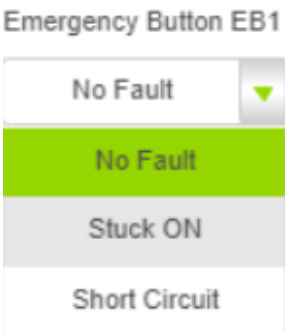


Figure 3.26: Fault injector element

To implement the safety relay solution, the simulation of switch devices began with the essential emergency stop button. In order to enhance safety through redundancy, most safety devices feature multiple circuit channels. Consequently, the simulated emergency button was designed with two channels, each equipped with its own input, output, and contact status variable.

Figure 3.27 presents the input and output variables associated with the simulated device FB, while Figure 3.28 provides a graphical representation of the emergency button FB.

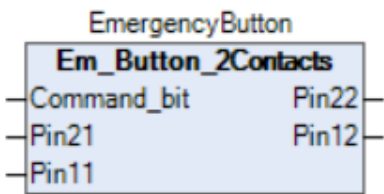


Figure 3.27: Em_Button_2Contacts FB

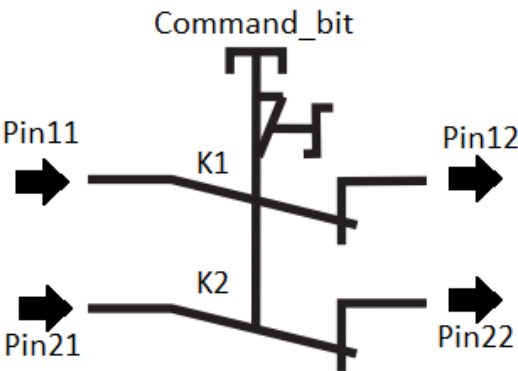


Figure 3.28: Emergency button function representation

The variable `Command_bit` represents the engagement of the button, with a `TRUE` value indicating a pressed button. This variable can be easily linked to a visual interface or a button in Factory I/O. The `Current_State` variable signifies the contact state of each channel. For instance, when the `Current_State` is `TRUE`, it simulates a closed "K1" contact, as depicted in Figure 3.28.

To simulate the normal functioning of the button, a `FALSE` value for the `Command_bit` variable signifies an unpressed button. In this case, the internal contacts are closed, and the `Current_state` variables for each channel are set to `TRUE`.

The functionality of the device is implemented using a `CASE OF` statement, with the condition variable being `Current_Fault_Mode`. Figure 3.29 illustrates a `CASE OF` statement in the programming of the emergency button FB for the event of no injected fault. The associated visual interface for this event is depicted in Figure 3.33a.

```
CASE Current_Fault_Mode OF
Switch_Fault_Modes.None:
  IF Command_bit THEN           //Button actuated
    Current_State := Off;       //Open contact 1
    Current_State_2 := Off;     //Open contact 2
    Bitmap_Image := 'Off';      //Display "Off" image
  ELSE
    Current_State := On;        //Closed contact 1
    Current_State_2 := On;      //Closed contact 2
    Bitmap_Image := 'On';       //Display "On" image
  END_IF
END_CASE
```

Figure 3.29: Em_Button_2Contacts FB with no fault

The outputs `Pin12` and `Pin22` will mirror the signals from the inputs based on the state of the "contacts". To achieve this, an internal variable called `Status_bit` is created, representing the converted Boolean value of `Current_state`. By using an `AND` operator, the output will adopt the value of the input only if the `Status_bit` is `TRUE`. The configuration of the outputs is demonstrated in Figure 3.30.

```
Status_bit := Current_State = On;      //Contact 1 closed
Status_bit_2 := Current_State_2 = On;  //Contact 2 closed
Pin12 := Pin11 AND Status_bit;         //Signal from Pin11 to Pin 12
Pin22 := Pin21 AND Status_bit_2;       //Signal from Pin21 to Pin 22
```

Figure 3.30: Outputs of the Em_Button_2Contacts FB enumeration data type

As mentioned in section 3.1.1, the simulated switch devices can experience two main types of faults: a channel contact stuck in a particular position or a short circuit between the channels. Simulating the first type of fault is straightforward, where one of the channel's status remains stuck at a specific value regardless of the command variable. Figure 3.31 illustrates the `CASE`

OF statement when this fault is injected. The corresponding visual interface for this situation is displayed in Figure 3.33b.

```

CASE Current_Fault_Mode OF
Switch_Fault_Modes.Stuck_On:
  IF Command_bit THEN           //Button actuated
    Current_State := On;        //Closed contact 1 - Stuck
    Current_State_2 := Off;     //Open contact 2
    //Image with one contact stuck closed and another open
    Bitmap_Image := 'Contact Stuck Close Off';
  END_IF
END_CASE

```

Figure 3.31: Em_Button_2Contacts FB outputs with "stuck on" fault

The simulation of a short circuit between channels is achieved by combining both input signals for the outputs using an OR operator. This ensures that the functionality remains the same, but the output setting is altered. Figure 3.32 shows the CASE OF statement when this fault is injected. It is important to note that this fault is only apparent and detectable when the channels are closed.

```

IF Current_Fault_Mode = Switch_Fault_Modes.Short_Circuit THEN
Status_bit := Current_State = On;           //Contact 1 closed
Status_bit_2 := Current_State_2 = On;       //Contact 2 closed
Pin12 := (Pin11 OR Pin21) AND Status_bit; //Signal from both inputs
Pin22 := (Pin11 OR Pin21) AND Status_bit_2;

```

Figure 3.32: Em_Button_2Contacts FB outputs with "short circuit" fault

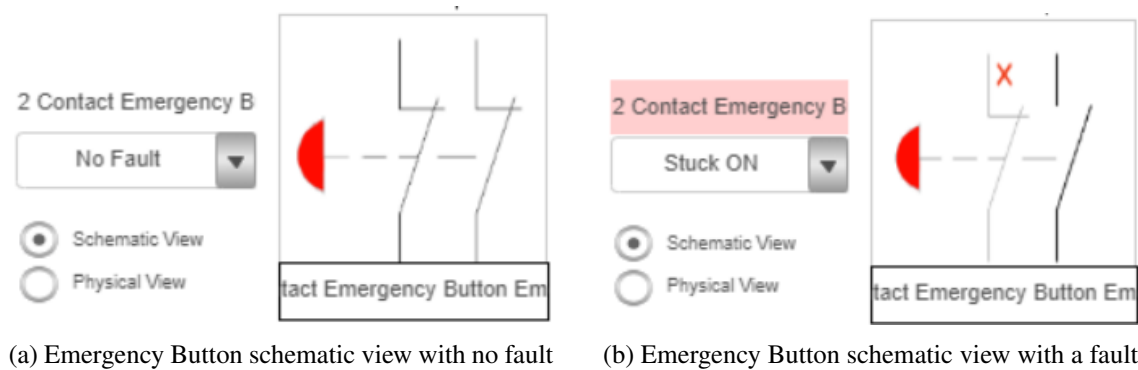


Figure 3.33: Emergency Button visual interface

Other safety switch devices, such as the reset button and the safety contactor, were simulated using the same method as described. A command variable changes the status of the internal circuit and faults can be injected to influence the same circuits. The main differences lie in the number of channels, which may vary depending on the device, and the images used for the display interface. Each device has its own set of images stored in separate image pools to represent its specific state.

3.4 Simulating a safety relay based application

A scenario was created in Factory I/O that includes a conveyor that moves boxes. The conveyor is controlled by a standard program in Codesys. The program is simple, when a button is pressed in Factory I/O the conveyor starts or stops.

The machine must incorporate an emergency stop safety function, which can be achieved through the utilization of the simulated safety relay and safety switch devices. The safety application entails stopping the machine in the event of a hazardous situation. However, since the machine's movement does not cease instantaneously, a warning light remains active for a few seconds to indicate that the machine may not have come to a complete stop. Figure 3.34 shows the Factory I/O scenario.



Figure 3.34: Factory scenario for safety relay based solution

The functional safety system comprises a safety relay that receives inputs from an emergency button and a reset button, and it manages the outputs to two contactors that control the actuators. One safety contactor is controlled by the instant output enabling circuit of the relay, while the other is governed by the delayed output enabling circuit. Figure 3.35 shows a diagram of the connection between the programs and Factory I/O.

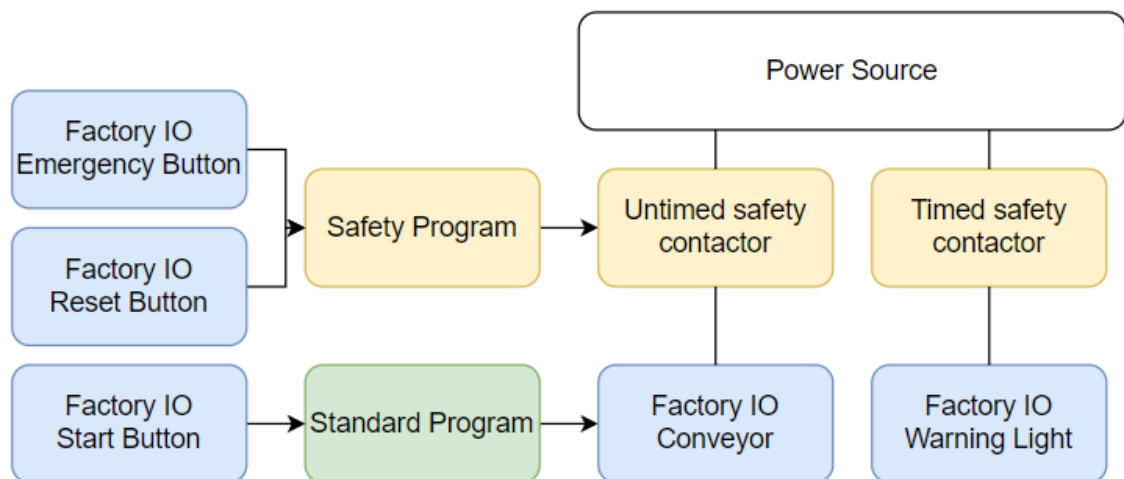


Figure 3.35: Diagram of the communication between programs and Factory I/O

The machine's standard control is managed by a standard program, while the functional safety system is governed by a safety-focused program. Both programs are allocated to different tasks, with the safety-related task given higher priority as presented in the task configuration in Figure 3.36.

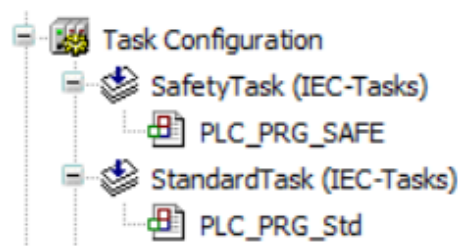


Figure 3.36: Task configuration in Codesys

The reset button is connected in series with normally closed contacts of the contactors so the relay can only be activated with the reset button if both contactors are disabled, this is called a monitored start for the relay. If a contactor has a fault, even if the safety relay can't detect it, the normally closed contact will not close and a start signal will not reach the safety relay to restart the machine.

To program the safety system is as easy as defining the function blocks to be used and then connecting them. Figure 3.37 shows the connection between the components FBs in a safety program. Configuring the safety devices used with the configuration parameters in Table 3.3 prepares the functionality of the devices visual interface in Codesys, shown in Figure 3.38.

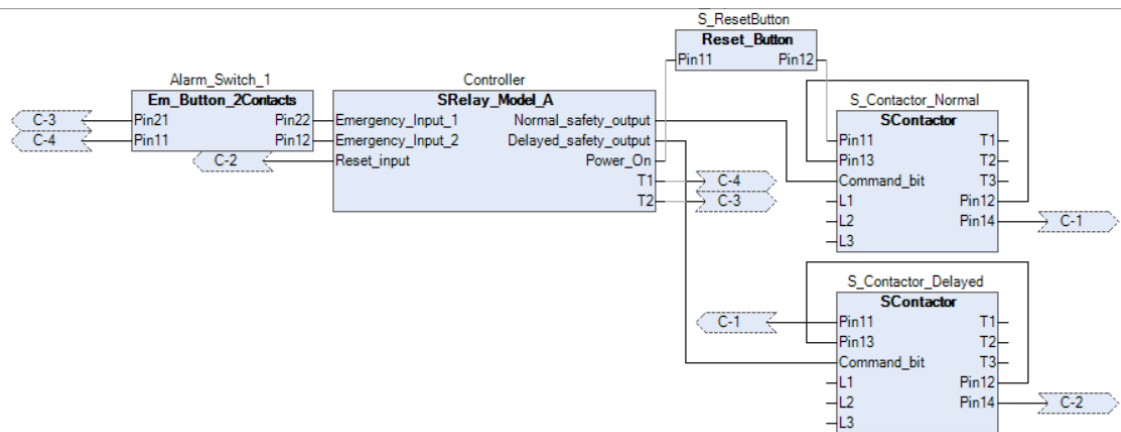


Figure 3.37: Safety solution components connection

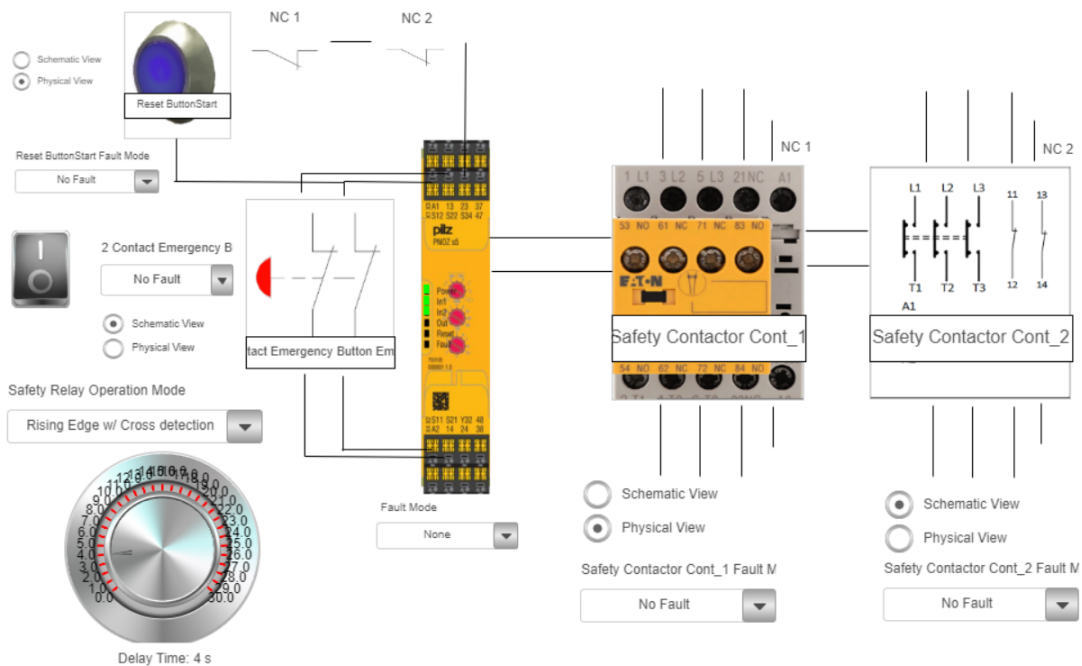


Figure 3.38: Safety solution visual interface

In the safety program, presented in Figure 3.37, pressing an emergency button in Factory I/O engages the `Command_Bit` variable of the emergency button FB that simulates two output channels. Even though the button is not commanded through the interface in Codesys, any faults can still be injected on it and the display will mirror the state of the button in Factory I/O. The safety relay then monitors the emergency button FB and takes an action of disabling the contactors which stops the actuators in Factory I/O. When the emergency button is disengaged the reset button can then be actuated in Factory I/O to change the `Command_Bit` in the reset button FB to enable the safety contactors and allow the standard program to control the actuators. At any time the

visual interface in Codesys can be interacted with to change device views and inject faults while observing the system reaction in Factory I/O.

The connection to Factory I/O is simple on account of the OPC UA communication protocol. Factory I/O, through the protocol connects to the Codesys SoftPLC and browses for the variables that can then be linked to the sensor and actuator tags from the scenario. Figure 3.39 shows this connection.

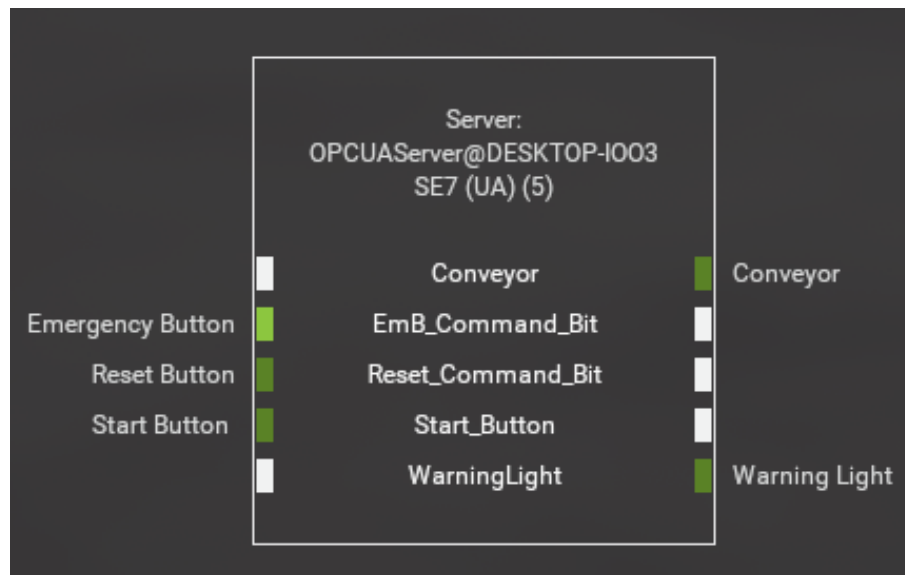


Figure 3.39: OPC UA connection between Codesys and Factory I/O

Results of the simulation show that, when faults are injected in the safety devices through the Codesys interface the safety relay maintains the safety function, stopping the actuators in Factory I/O. Normal operation of the machine is made impossible by the safety system until all faults are fixed and a safe state is achieved. Therefore, this scenario demonstrates the appropriate application of the developed simulations of safety devices.

3.5 Concluding remarks

This chapter presented the adopted approach for simulating a simple safety solution using a safety relay. The developed solution incorporates redundant elements to enhance reliability, as well as robust diagnostic coverage and fault tolerance to ensure safe operation. These attributes align with the requirements of a higher SIL safety component. By separating the functionality of the safety relay into distinct function blocks, the overall process is simplified and organized.

Additionally, specific devices were developed to serve as inputs and outputs for the safety solution. These devices feature visual representations and allow faults to be injected via a visual interface, which are then detected by the safety logic to trigger appropriate actions.

The combined implementation of the relay and safety devices forms a safety system responsible for executing the safety function of a scenario in Factory I/O. The results of the demonstration

show that the developed functional safety system maintains safety of the Factory I/O scenario and has great fault tolerance.

The success of this solution paves the way for the development of more intricate safety solutions for safety PLCs. The subsequent chapter will present the development of a PLCopen compliant library of function blocks.

Chapter 4

PLCopen compliant safety function blocks library development

This chapter describes the development of the safety-related function blocks that will be part of a library to be used in safety PLC application programs. The specification adopted is provided by PLCopen and is applicable for operations of SIL level 1, 2 and 3, being this the highest required for safety of machinery applications (3).

Compliance with PLCopen Safety Software Technical Specification in developing safe function blocks is of utmost importance. This specification offers recommendations, rules and procedures that should be respected.

4.1 Generic aspects of safety function blocks

For programming safety-related functions a distinction is made between standard and safety-related signals with the use of the definition "SAFE" on the latter (3). For example, a SAFEBOOL variable is essentially the same as a BOOL variable except the programmer admits it should be handled with care and can contain more attributes that can be used for SIL calculation (3). Safety-related FBs should have a SF_ prefix and all the SAFE variables the S_ prefix (3). The default value for SAFEBOOL variables should be set to FALSE, and this default value should be maintained during initialization and in the event of any errors (3). For more generic recommendations consult "Part 1: Concepts and Function Blocks" of PLCopen Safety Software Technical Specification (3).

A generic function block can be considered in which other function blocks will be based on. Each FB includes additional inputs and outputs related to its specific functionality. Figure 4.1 shows a representation of this FB. Table 4.1 explains the meaning of each generic input and output.

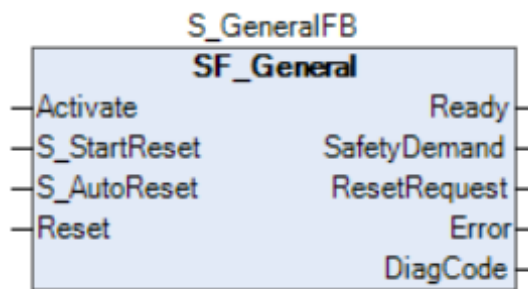


Figure 4.1: PLCopen generic safety FB

Table 4.1: Generic input parameters of safety FBs. Adapted from PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” (3)

Name	Description
Input Parameters	
Activate	BOOL. Activation of FB. Can be connected to a device to represent the status of said device. If no device is connected, a static TRUE signal should be assigned.
S_StartReset	SAFEBOOL. When SAFETRUE allows for automatic reset when the Programmable Electronic System (PES) is started.
S_AutoReset	SAFEBOOL. When SAFETRUE allows automatic reset after dangerous situation is released.
Reset	BOOL. Executes the reset of an action of the FB in which it is employed
Output Parameters	
Ready	BOOL. Indicates that a FB is activated
SafetyDemand	BOOL. Indicates that a safe state is demanded (related to the safety functionality). DiagCode=16#88XX.
ResetRequest	BOOL. Indicates that a Reset signal is expected to continue. Diag-Code=16#84XX
Error	BOOL. Indicates that an error has occurred. DiagCode=16#CXXX
DiagCode	WORD. Diagnostic register in hexadecimal format.

Each safety-related FB has a state diagram that should be respected. Each state has a name and a diagnostic code. The transitions between states have a numerical priority order, being 0 the highest priority. The state diagrams have two horizontal lines, the first line separates the state where the FB is not active from the other states, and the second line separates the safe states from the unsafe states. Safe states are the ones where the "SAFE" outputs are FALSE. The transition to inactivate a FB always has the highest priority. In the event of any error a correspondent state provides the error diagnostic code. Figure 4.2 presents a generic state diagram. Each state has an output setting. Table 4.2 presents the output setting of "Init" state, note that some outputs might be dependent on more than the activated state. The configuration of every FB output settings adheres to the specifications outlined in the PLCopen technical documentation for each respective function block.

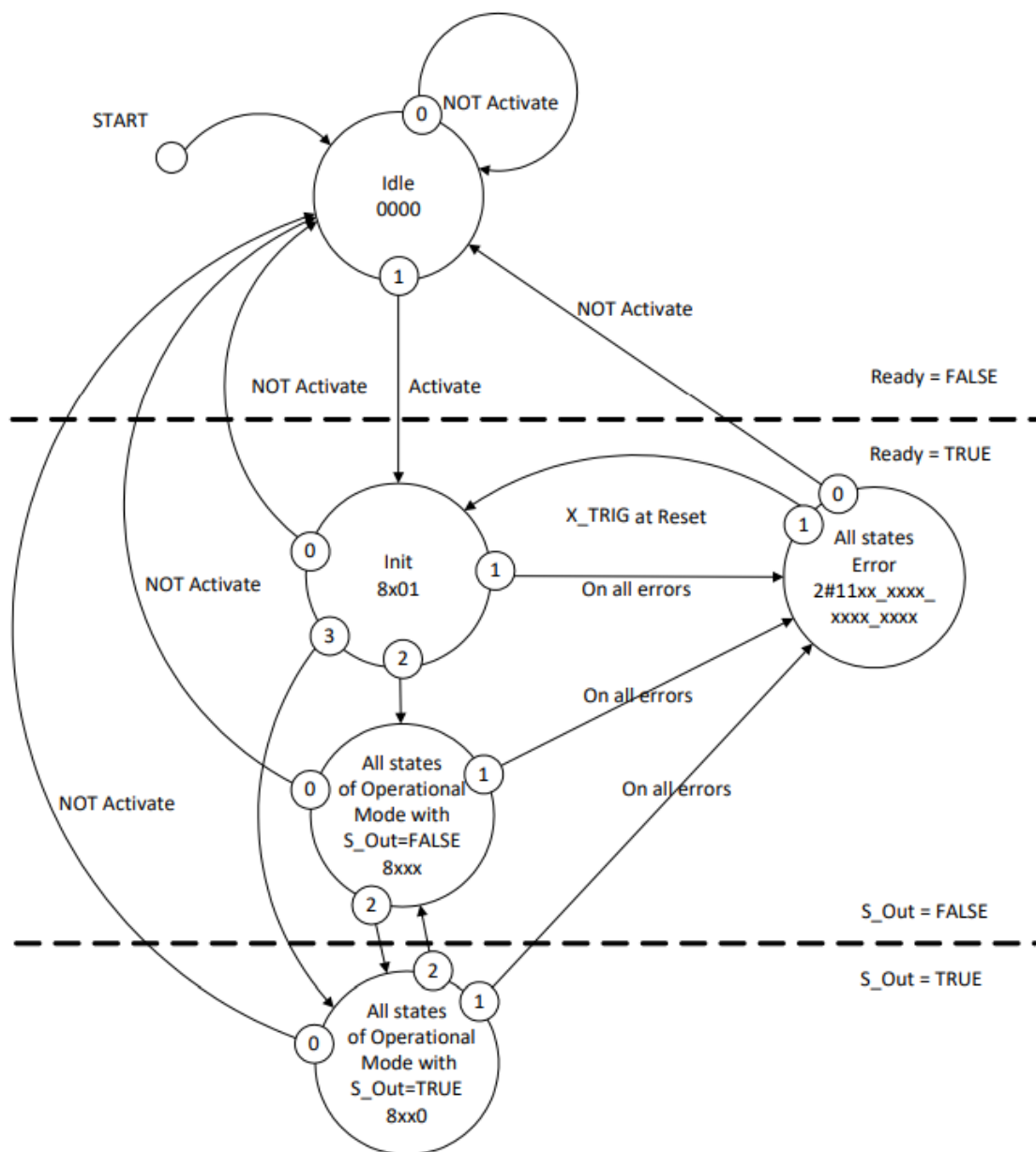


Figure 4.2: Generic state diagram. Adapted from PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” (3)

Table 4.2: Output setting of a generic "Init" state. Adapted from PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” (3)

DiagCode	State Name	Output Setting
8x01	Init state of operation mode	Ready =TRUE S_Out =FALSE SafetyDemand =Depending ResetRequest =Depending Error =FALSE

4.2 Function block programming

To program the function blocks a state machine was implemented to each, where all the input and output variables were declared like in the PLCOpen safety specification (3). The states are listed in an enumeration data type `State_FbName_FB` of `WORD` base data type, where each state has the diagnostic code associated. A `CASE OF` instruction is used to run a set of conditional instructions depending on the current state. Each `CASE` statement checks for conditions defining state transitions, which are based in decision trees programmed in the form of `IF . . . ELSIF . . . ELSE` structures.

An important aspect of programming the state diagrams is making the FB functionality as fast as possible when it is related to safety. And for this reason transient, or temporary, states have to be considered. A transient state exists when a transition to that state and from that state to another are both true in the same PLC scan, otherwise it is called a stable state. Figure 4.3 shows an example in which Input 0 and Input 1 are true, so the transition should be made from State 0 to State 2 in the same controller scan.

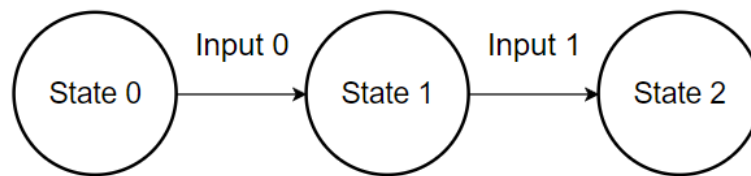


Figure 4.3: Example of a transient state, State 1

However, transitions with rising edge or falling edge signals are only accepted if these signals happen when the appropriate state is active. So even if the transition condition is true it is not valid in a multiple state transition. So the states that have exiting transitions with edge signals are always recognized as a stable states.

Some FB state diagrams are small and simple to analyze, but others have a lot of possible paths and states which makes for extensive programming. For this reason, the safety FBs were programmed with two methods, one optimized for speed and a code-optimized method.

4.2.1 Programming for optimized performance

The way of programming a state machine so it can execute as fast as possible is checking for the conditions that enable all the possible state transition paths that start in any (initial) state and end in a (final) stable state. Programming is thus optimized for speed, since all the allowable initial to final state transitions occur in a row within a single `CASE` or `IF . . . ELSE` execution. A FB that was programmed this way was the `SF_EmergencyStop` FB. This function block is a safety-related function block for monitoring an emergency stop button (3). Figure 4.4 shows the input and output variables of this FB.

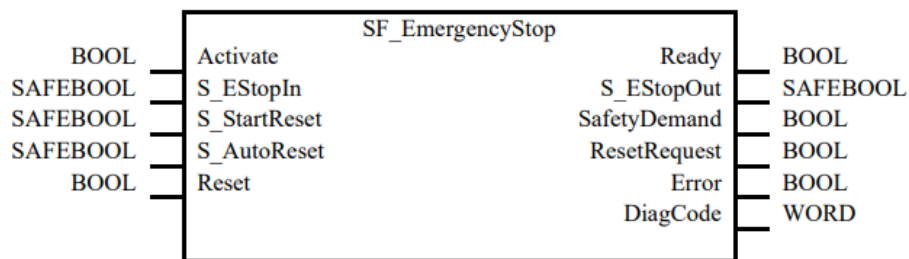


Figure 4.4: SF_EmergencyStop input and output variables. Adapted from: Adapted from PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” (3)

In addition to some generic variables already mentioned the SF_EmergencyStop FB includes the S_EStopIn SAFEBOOL input variable which is the safety demand input (e.g. the engagement of the emergency button), a FALSE value represents an engaged button, i.e. a positive safety demand. It also includes the S_EStopOut SAFEBOOL output variable which is the safety-related response, a TRUE value represents the safety output enabled, in other words, no demand for safety-related response (e.g., emergency stop button not engaged, no internal errors active). Figure 4.5 illustrates the state diagram of the SF_EmergencyStop function block, showcasing specific transitions that are highlighted for future reference.

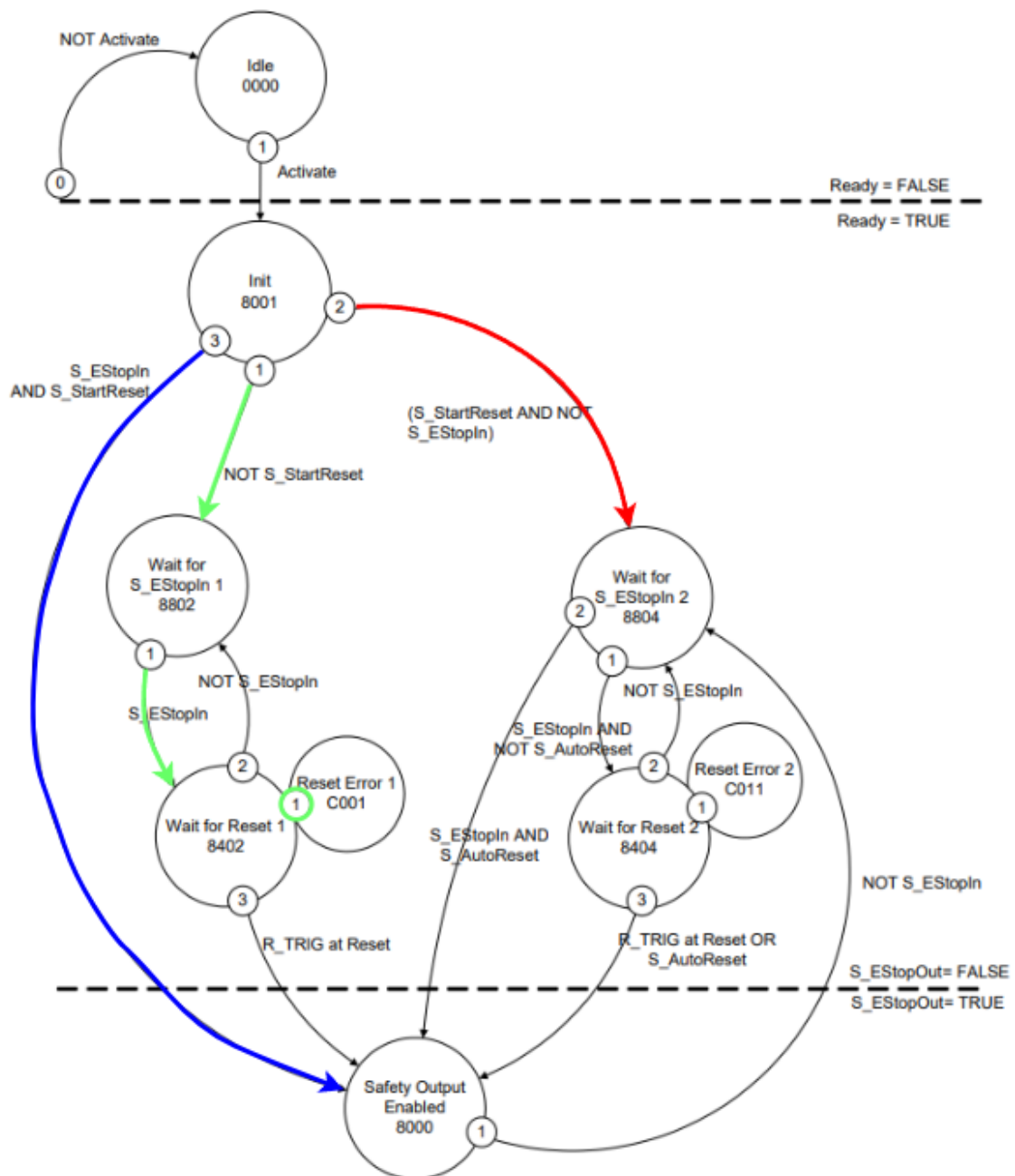


Figure 4.5: SF_EmergencyStop. Adapted from: Figure 23 of the PLCopen Safety Software Technical Specification "Part 1: Concepts and Function Blocks": "State diagram for SF_EmergencyStop" - Page 64 (3)

It is not marked in the state diagram but the transition from any state to the "Idle" state has the highest priority. This is why to start, an IF . . . THEN instruction activates the "Idle" state if Activate is FALSE, if not, the ELSE instruction executes the CASE statement.

In this FB the "Init" state is always a transient state so starting in the "Idle" state after the FB is active the transition can be made in three main paths which are the "Init" exiting paths.

If S_StartReset is FALSE the transition can be made through the path represented in green.

First is the transition 1 from "Init" to "Wait for S_EStopIn 1". The transition can end here but if S_EStopIn is TRUE it goes further to "Wait for Reset 1". The transition can end in this state or if Reset is TRUE the last possible state is "Reset Error 1".

If instead S_StartReset is TRUE and S_EStopIn is FALSE the transition is made through the red path from "Init" to "Wait for S_EStopIn 2" which is the only possible state in this transition path.

The last path option is if S_StartReset is TRUE and S_EStopIn is TRUE the transition is made through the path represented in blue from "Init" to "Safety Output Enabled" which is the only possible state in this transition path.

Figure 4.6 shows the explained initial part of the FB when the active state is "Idle". The numbers in parenthesis are the priority numbers of the all the single transitions made until a stable state is reached.

```

IF NOT Activate THEN
  State_EmergencyStop_FB := Idle;
ELSE
  CASE State_EmergencyStop_FB OF
    Idle:
      IF NOT S_StartReset THEN //Trans 1 from Init to WaitForS_EStopIn1}
        (*Green path*)
        IF S_EStopIn THEN //from WaitForS_EStopIn1 to WaitForReset1
          IF Reset THEN //end in ResetError1
            State_EmergencyStop_FB := ResetError1; //(1111)
          ELSE //or stay in WaitForReset1
            State_EmergencyStop_FB := WaitForReset1; //(111)
          END_IF
        ELSE //or stay in WaitForS_EStopIn1
          State_EmergencyStop_FB := WaitForS_EStopIn1; //(11)
        END_IF
      ELSE
        IF NOT S_EStopIn THEN //Trans 2 to WaitForS_EStopIn2
          (*Red path*)
          State_EmergencyStop_FB := WaitForS_EStopIn2; //(12)
        ELSE //Path 3 from Init to SafetyOutputEnabled1
          (*Blue path*)
          State_EmergencyStop_FB := SafetyOutputEnabled; //(13)
        END_IF
      END_IF
    END_CASE
  END_IF

```

Figure 4.6: Initial part of SF_EmergencyStop starting in "Idle" state

The same method of analysing and programming the possible transitions was used for all the other states. After the state transitions, the outputs are set. The diagnostic code is a WORD variable and is associated with the state, so a data type conversion of the State_EmergencyStop_FB variable sets the DiagCode. The Ready variable is only FALSE when the state "Idle" is active. The second horizontal line in the state diagram states that the safe variable S_EStopOut is only

TRUE when the state "Safety Output Enabled" is active. The `SafetyDemand`, `ResetRequest` and `Error` variables are set according to the diagnostic code. `SafetyDemand` is TRUE when the code starts with "88", `ResetRequest` is TRUE when the code starts with "84" and `Error` is TRUE when the code starts with "C". Figure 4.7 shows the FB outputs.

```

DiagCode := TO_WORD (State_EmergencyStop_FB);
Ready := State_EmergencyStop_FB <> Idle;
S_EStopOut := State_EmergencyStop_FB = SafetyOutputEnabled;
SafetyDemand := (DiagCode AND 16#FF00) = 16#8800;
ResetRequest := (DiagCode AND 16#FF00) = 16#8400;
Error := (DiagCode AND 16#F000) = 16#C000;

```

Figure 4.7: SF_EmergencyStop Outputs

4.2.2 Code optimized programming

Applying the optimized for speed method to more complex FBs would lead to very long, complex and error-prone code because of the numerous possible transition paths, so an alternative is implementing recursion. One solution is executing the transitions CASE OF instruction without featuring transient states in the IF... ELSE... decision tree and execute this inside a REPEAT... UNTIL... loop where the loop only stops when a stable state is reached.

One function block that was programmed this way is the `SF_MutingSeq` FB. Muting is the intended suppression of the safety function (e.g., light barriers). In this FB, sequential muting with four muting sensors is specified (3). Figure 4.8 shows the input and output variables of this FB. In addition to some generic variables already mentioned this FB includes the input and output variables in Table 4.3. Figure 4.9 displays the state diagram of the `SF_MutingSeq` function block (FB), with a specific transition path highlighted for future reference.

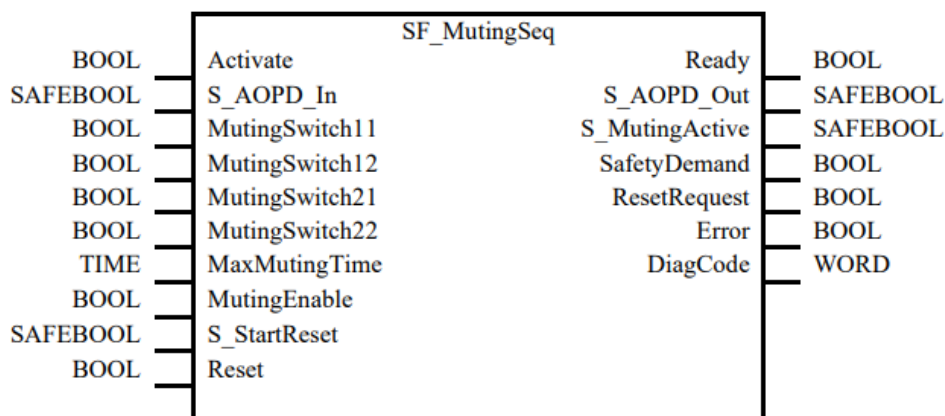


Figure 4.8: `SF_MutingSeq` input and output variables. Adapted from: Adapted from PLCopen Safety Software Technical Specification "Part 1: Concepts and Function Blocks" (3)

Table 4.3: Input and outputs parameters of the SF_MutingSeq FB. Adapted from PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” (3)

Name	Description
Input Parameters	
S_AOPD_In	SAFEBOOL. Signal from Active opto-electronic protective device (AOPD), or safety light barrier. TRUE: Protection field not interrupted
MutingSwitch11	BOOL. Status of Muting sensor 11. TRUE: Workpiece actuates muting sensor 11.
MutingSwitch12	BOOL. Status of Muting sensor 12. TRUE: Workpiece actuates muting sensor 12.
MutingSwitch21	BOOL. Status of Muting sensor 21. TRUE: Workpiece actuates muting sensor 21.
MutingSwitch22	BOOL. Status of Muting sensor 22. TRUE: Workpiece actuates muting sensor 22.
MaxMutingTime	TIME. Maximum time for complete muting sequence. 0-120 min
MutingEnable	BOOL. Command by the control system that enables the start of the muting function when needed by the machine cycle. TRUE: Start of Muting function enabled
Output Parameters	
S_AOPD_Out	SAFEBOOL. Safety related output, indicates status of the muted guard. TRUE: AOPD protection field not interrupted or muting active
S_MutingActive	SAFEBOOL. Indicates status of Muting process. TRUE: Muting active.

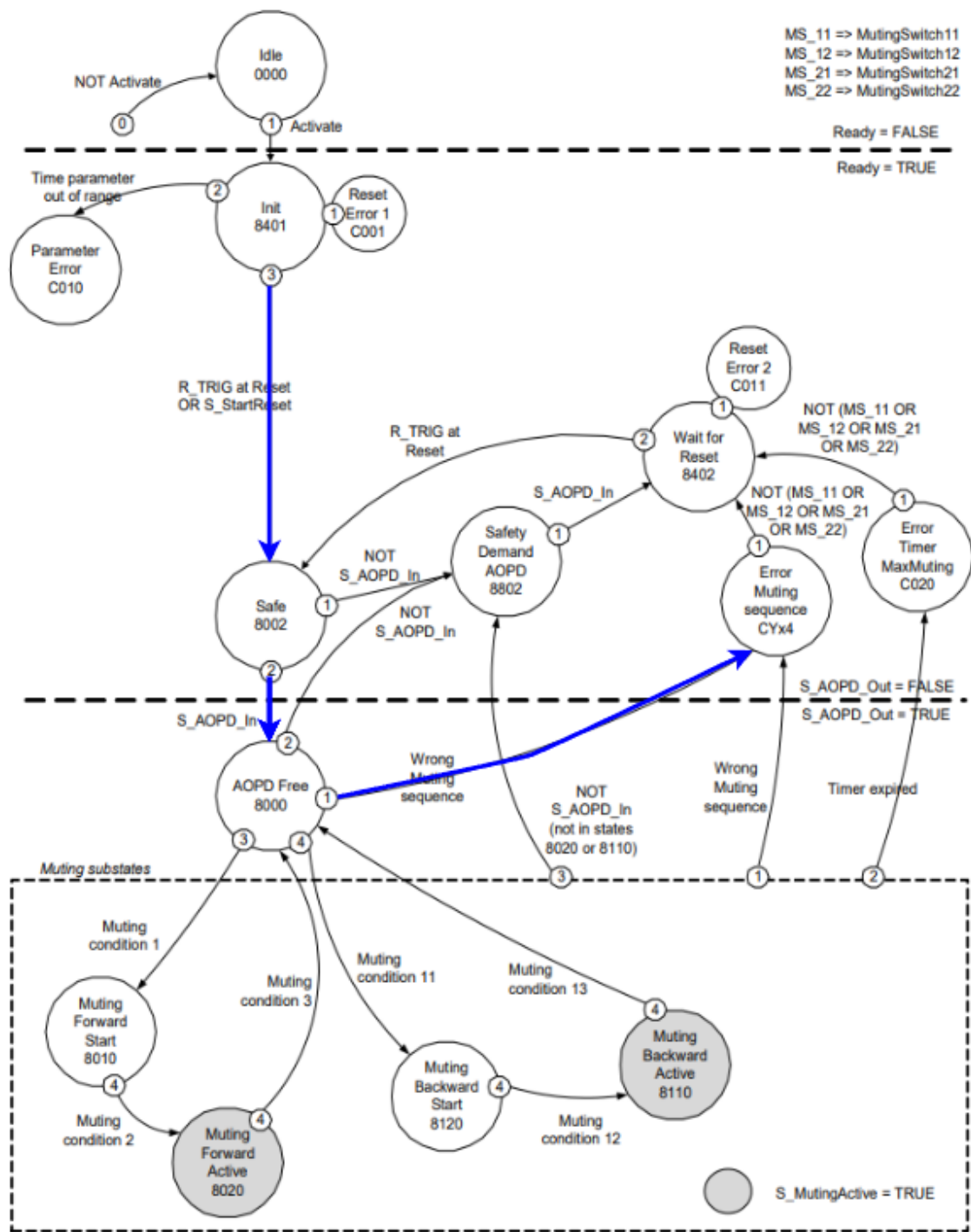


Figure 4.9: SF_MutingSeq state diagram. Adapted from: Adapted from Figure 43 of the PLCopen Safety Software Technical Specification “Part 1: Concepts and Function Blocks” State diagram for SF_MutingSeq” - Page 104” (3)

To program complex function blocks such as this one in addition to the state variable of an enumeration of FB states data type another of the same type is needed. This variable is set in the beginning of each REPEAT cycle scan to represent the initial state of a single transition, assigned

to Starting_State_SF_MutingSeq.

To explain this, Figure 4.10 presents a part of the code that does the multiple transitions in the path represented in blue in Figure 4.9 from "Init" to "Error Muting Sequence", assuming that S_StartReset and the S_AOPD_In variables are TRUE and that there is a wrong muting sequence in this PLC scan. Analyzing the state diagram after executing the "Init" state CASE statement the state variable changes to the "AOPD Free" state. Then the REPEAT loop checks for the ending condition State_SF_MutingSeq = Starting_State_SF_MutingSeq. As it is not true the loop repeats setting the starting state as "AOPD Free" and executes the respective CASE statement. As there is wrong muting sequence the transition from "AOPD Free" to "Error Muting Sequence" happens. The condition to end the loop is checked again and as it is not true the loop repeats only to not realize any transition because this error state is stable and needs a specific action to exit. And so, all the possible transitions are done and the loop ends. Even if multiple transitions to a stable state are done in a single PLC scan this method is slower because when a stable state is reached an unnecessary last loop cycle is done to confirm, much like this example where the CASE statement for the state "Error Muting Sequence" executes but is not needed and does not take any effect.

```

IF NOT Activate THEN
    State_SF_MutingSeq := Idle;
ELSE
    REPEAT
        //Set transition starting state
        Starting_State_SF_MutingSeq := State_SF_MutingSeq;
        CASE State_SF_MutingSeq OF
            Init:
                IF ...
                    (*From Init to ResetError1*)
                    (*From Init to ParameterError*)
                    IF NOT S_AOPD_In THEN
                        State_MutingSeq_FB :=
                            MutingFB_States.SafetyDemandAOPD;
                    ELSE
                        State_MutingSeq_FB := MutingFB_States.AOPDFree;
                    END_IF
                END_IF
            AOPDFree:
                IF *Wrong muting sequence* THEN
                    State_MutingSeq_FB :=
                        MutingFB_States.ErrorMutingSequence;
                    (...)
                END_IF
            END_CASE
        //Check if starting state and final state are the same
        UNTIL State_SF_MutingSeq = Starting_State_SF_MutingSeq
        END_REPEAT
    END_IF

```

Figure 4.10: Initial part of the SF_MutingSeq FB

With this method, conditions for transitions made with edge signals have to be defined in each PLC scan before the loop starts. Like the transition from "Init" to "Safe", in the transition decision tree the variable used is `R_Reset_Init` which is only true when the rising edge `Reset` signal happens in the PLC scan where "Init" is active, as shown in Figure 4.11.

```
Reset_R_Trig(CLK := Reset);
R_Reset_Init := Reset_R_Trig.Q AND State_SF_MutingSeq = Init;
```

Figure 4.11: Edge signal in `SF_MutingSeq`

Muting function blocks differ from other function blocks in that they encompass muting sub-states within a larger state. Transitions between these substates are based on combinations of the sequence of muting sensor inputs. It is crucial that all substates remain stable, so the corresponding `CASE` statements are placed outside the `REPEAT` loop. Not doing so would result in a transition from "AOPDFree" to one of the muting start states, followed immediately by a transition to the "Error Muting Sequence" state within the same PLC scan.

Certain states have variable diagnostic codes that depend on other elements. For example, in the case of the `SF_MutingSeq` FB, the "Error Muting Sequence" state has a variable diagnostic code. The output setting for this state follows the guidelines specified in "6.11.6 Function Block-Specific Error and Status Codes" (pages 108-110) of the PLCopen Safety Software Technical Specification, "Part 1: Concepts and Function Blocks" (3). The diagnostic code relies on the active state preceding the transition and the values of the muting input variables. To achieve this, the previous state is stored in a variable called `Previous_State_SF_MutingSeq` at the moment the transition occurs. This previous state is then utilized when setting the `DiagCode` output. Additionally, the binary value of the muting switch input variables is used to update the appropriate bit in the `DiagCode` hexadecimal code.

4.3 Programmed safety function blocks and supporting devices

The choice of safety function blocks to be programmed was based on several factors, including the prevalence and significance of the safety functions they represent, as well as their compatibility with scenarios that can be simulated in Factory I/O. For example, function blocks related to two-hand control were not implemented because such devices do not exist in Factory I/O, and it is difficult to simulate and engage them on an interface in Codesys. Similarly, pressure-sensitive equipment-related function blocks were not implemented for the same reason.

Certain function blocks are responsible for monitoring the functionality of safety switch devices. Consequently, it was necessary to simulate these devices to accurately assess their performance.

Some implemented function blocks were already mentioned in the present chapter. The FB that monitors an emergency stop button, "`SF_EmergencyStop`". And the "`SF_MutingSeq`" FB that performs a muting function, i.e. suppression of a light barrier safety function, using four muting

switches. Muting enables temporary access to protected areas while maintaining an acceptable level of safety, useful for material feeding or maintenance activities.

Another muting function was implemented, which operation is similar to the "SF_MutingSeq" function block. The "SF_MutingPar_2Sensor" function block performs the muting function using only two input switch sensors. In order to support the functionality of the muting function blocks, a safety switch device simulating a safety light barrier was developed. This simulated device can be activated by a light barrier within Factory I/O, and faults can be injected to simulate a short circuit in the device. Its purpose is to provide a realistic representation of a safety light barrier for testing and validating the muting function blocks. A muting switch sensor FB was also developed to connect with the muting function blocks that can also be activated by default sensors in Factory I/O.

In situations where an error occurs in a muting function, it is necessary to remove the object obstructing the light barrier in order to reset the muting function. To address this, the "SF_Override" function block was implemented. This function block enables the movement of the production line to remove the object blocking the light barrier, thereby resolving the error condition.

An electro-sensitive protective equipment (ESPE) monitoring function block, "SF_ESPE" was also developed. The functionality of the "SF_ESPE" function block is similar to that of the "SF_EmergencyStop" function block, with the distinction that it is specifically designed to monitor a light barrier. The output of the simulated safety light barrier can be connected to the input of the "SF_ESPE" FB.

A "SF_Guard" FB function block was implemented to monitor the position of a safety guard. This function block has two independent input parameters for two switches, which are coupled with a time difference. The safe output of the "SF_Guard" FB is utilized as an input for another programmed function block, "SF_GuardLocking_2". This function block is responsible for controlling the safety guard with guard locking functionality. In the case of the "SF_Guard" function block, a simulation of a safety guard was created. This safety guard comprises two switches that monitor the position of the guard. Faults can be injected to manipulate the behavior of these switches, allowing for comprehensive testing of the "SF_Guard" function block. Similarly, for the "SF_GuardLocking" function block, a simulation of a safety guard with guard locking was implemented. This simulation is analogous to the guard device, but it includes two input command variables. One variable is responsible for opening and closing the guard, while the other controls the locking feature, enabling or disabling the guard locking mechanism. Figure 4.12 shows the symbolic views of the safety guard with and without guard locking.

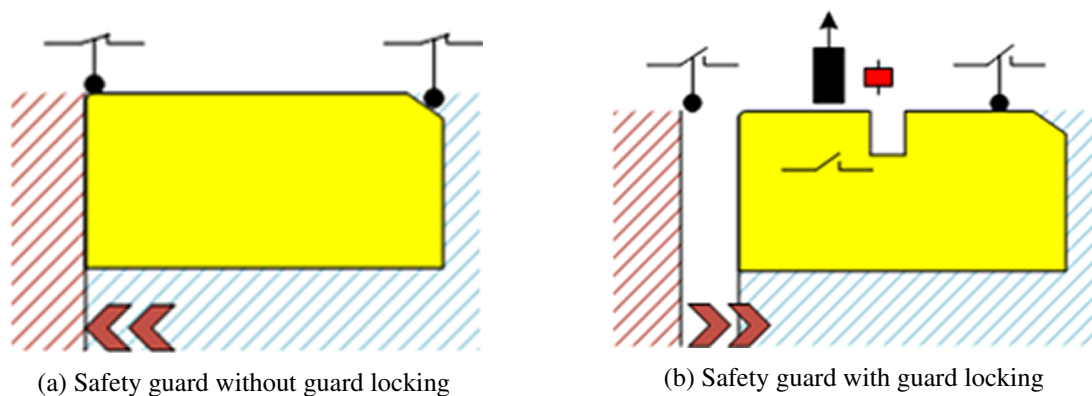


Figure 4.12: Symbolic views of the safety guard with and without guard locking

The "SF_Equivalent" and "SF_Antivalent" function blocks were also implemented. These function blocks are designed to monitor two inputs and convert them into a SAFE output, while also incorporating discrepancy time monitoring. The distinction between the two lies in their input monitoring behavior: the "SF_Equivalent" function block monitors two identical signals, whereas the "SF_Antivalent" function block monitors two opposing signals. These FBs are useful for monitoring safety devices with two output channels.

A "SF_ResetButton" was programmed. This function block incorporates trailing edge functionality to all function blocks with a reset input. Whenever a function block with a reset input generates a reset request signal, the "SF_ResetButton" becomes active and sends a reset signal accordingly.

Lastly, the implemented "SF_EDM" function block is responsible for controlling a safety output and monitoring the subsequent controlled actuators to ensure their correct activation. One example of its usage is to enable a safety contactor and then monitor if the contactor has been successfully enabled as intended.

To conclude Table 4.4 shows the list of developed FBs and simulated devices. The role and importance of these developments will be shown in the next chapter where some “safe virtual plants” will be demonstrated.

Table 4.4: List of programmed safety function blocks and simulated devices

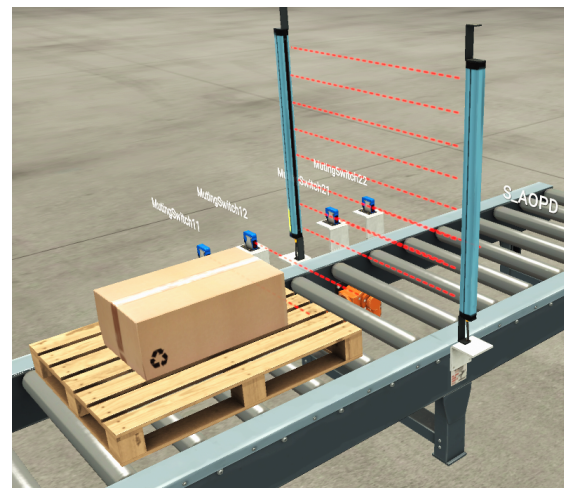
Safety function blocks	Simulated safety devices
SF_EmergencyStop, SF_ResetButton.	Emergency stop button, reset button.
SF_MutingSeq, SF_MutinPar_2Sensor, SF_Override.	Safety light barrier, safety muting sensor.
SF_Guard, SF_GuardLocking_2	Safety guard with and without interlocking
SF_ESPE	Safety contactor
SF_Equivalent, SF_Antivalent	
SF_EDM	

4.4 Testing the function blocks

After programming the FBs comes a testing phase to check for any mistakes. To make the testing process easier, a Codesys feature, “FB Visu Creator”, was used to automatically create simple interfaces of the FBs, including the inputs as buttons and outputs as lights. In some FBs some simple Factory I/O scenarios were used jointly with these interfaces to get a glimpse of the possible solutions to come. This way, the inputs can easily be changed by forcing variables to simulate faults, see the output reaction, and test each state of the state diagram. For example, to test the `SF_MutingSeq` FB a scenario was done in Factory I/O with a light barrier and muting sensors where a conveyor moves a box. In the FB interface, the muting sensors and light barrier bits can be seen changing as the box moves through. Situations can be forced to test the error states, for example, moving the box backward after the muting has started to have a muting switch sequence error or stopping the box to check if the maximum muting time error happens. The diagnostic code changes and can be compared to the state diagram to check if it corresponds. Figure 4.13 shows a testing interface and Factory I/O scenario for testing the `SF_MutingSeq` FB.



(a) Interface for testing the FB



(b) Factory I/O scenario for testing the FB

Figure 4.13: Example of `SF_MutingSeq` FB testing method

Some function blocks are meant to be used together so they were also tested as such. The most basic, but important, safety function is a safety stop by engaging an emergency button, all industrial machinery plants incorporate this safety function.

The functionality of the emergency stop FB was already discussed earlier in this chapter. To test this FB it monitors the simulated emergency button, whose development was already elucidated on the previous chapter, through the "S_Equivalent" FB which converts two equivalent inputs to a safe output, including discrepancy time monitoring. It also receives a reset input from the reset FB which is connected to a simulated reset button, after sending a reset request to the reset FB. This way in one scenario three safety function blocks were tested, namely the "SF_Equivalent", "SF_EmergencyStop" and "SF_ResetButton", while using simulated devices.

The result was a positive functioning of the individual safety FBs as well as the collective solutions. Visual interfaces like the one in Figure 4.13a were used to monitor the safety FBs while faults were injected in the simulated safety devices. Figure 4.14 shows the block diagram of the emergency stop function testing.

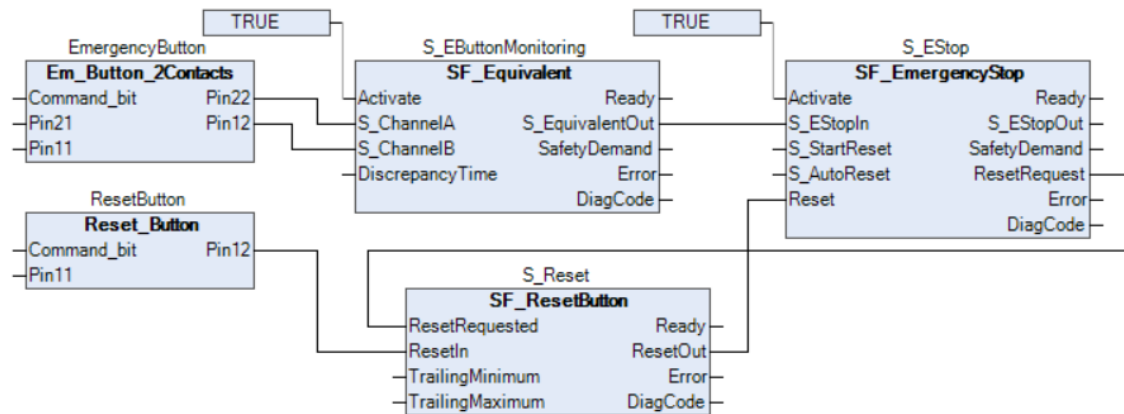


Figure 4.14: Block diagram of the emergency stop function testing

4.5 Concluding remarks

This chapter followed the process of programming PLCopen-compliant function blocks. The significance of adhering to the PLCopen safety software technical specification recommendations and rules was discussed, emphasizing the importance of faithfully following the function blocks state diagrams.

Additionally, the challenges related to programming from simple state diagrams to complex ones was discussed, along with the solutions found and associated implications.

The testing phase of the function blocks was described. In addition to uncovering eventual programming mistakes, the advantages of this stage were revealed.

Looking ahead, the next chapter will delve into the development of realistic safety scenarios as a proof of concept. Building upon the foundations established in past chapters, practical examples of safety solutions will be showcased, addressing specific industrial safety concerns.

Chapter 5

Case Demonstrations

The major goal of this chapter is to show how common but “unsafe” industrial automation scenarios – like those typically developed and discussed in IC classes can benefit a lot from the integration of programmable safety solutions based on the safety function blocks developed in the previous chapter. These practical safety solutions seek to offer verifiable proof of the efficiency and viability of the proposed approach.

A standardized framework for safety implementation is created by integrating the safety function blocks, which follow the PLCopen requirements. This framework makes a guarantee that industrial systems are consistent and reliable. The safety function blocks serve as modular building blocks that make it possible to design complex safety systems that can solve a variety of industrial safety issues.

Moreover, the use of safety device simulations enables the testing and verification of the effectiveness of these safety solutions in a controlled setting. The efficacy of the solutions can be evaluated by simulating various hazardous events and examining how the safety system reacts.

Throughout this chapter, the application of safety function blocks and simulations of safety devices will be demonstrated through realistic case scenarios. These practical examples, spanning across different applications, highlight the flexibility and adaptability of the developed safety solutions. They serve to showcase the effectiveness of the implemented safety function blocks and their compatibility with diverse industrial settings.

5.1 Palletizing station

The first case demonstration scenario is a palletizing station where a machine is responsible for stacking boxes onto a pallet. The machine is enclosed by fixed safety guards placed at appropriate distances. However, there are certain areas that are not enclosed by the safety barriers, including the entrance area for pallets and boxes, the exit area for the pallet and stacked boxes, and a safety guard for maintenance access.

In this scenario, the required safety function is an emergency stop mechanism that activates in case of trespassing into the hazard zone or upon engagement of an emergency button. The

emergency stop function is achieved through the utilization of a safety contactor, which cuts off the power supply to all actuators within the hazard zone.

The machine cycle is controlled by a standard control program and the safety functions are controlled by a safety-specific program. In the station there is a conveyor moving pallets. The pallet stops under a pick-and-place machine, the machine stacks on the pallet three boxes picked up from another conveyor and then it repeats. Figure 5.1 shows the Factory I/O scenario with the access areas highlighted.

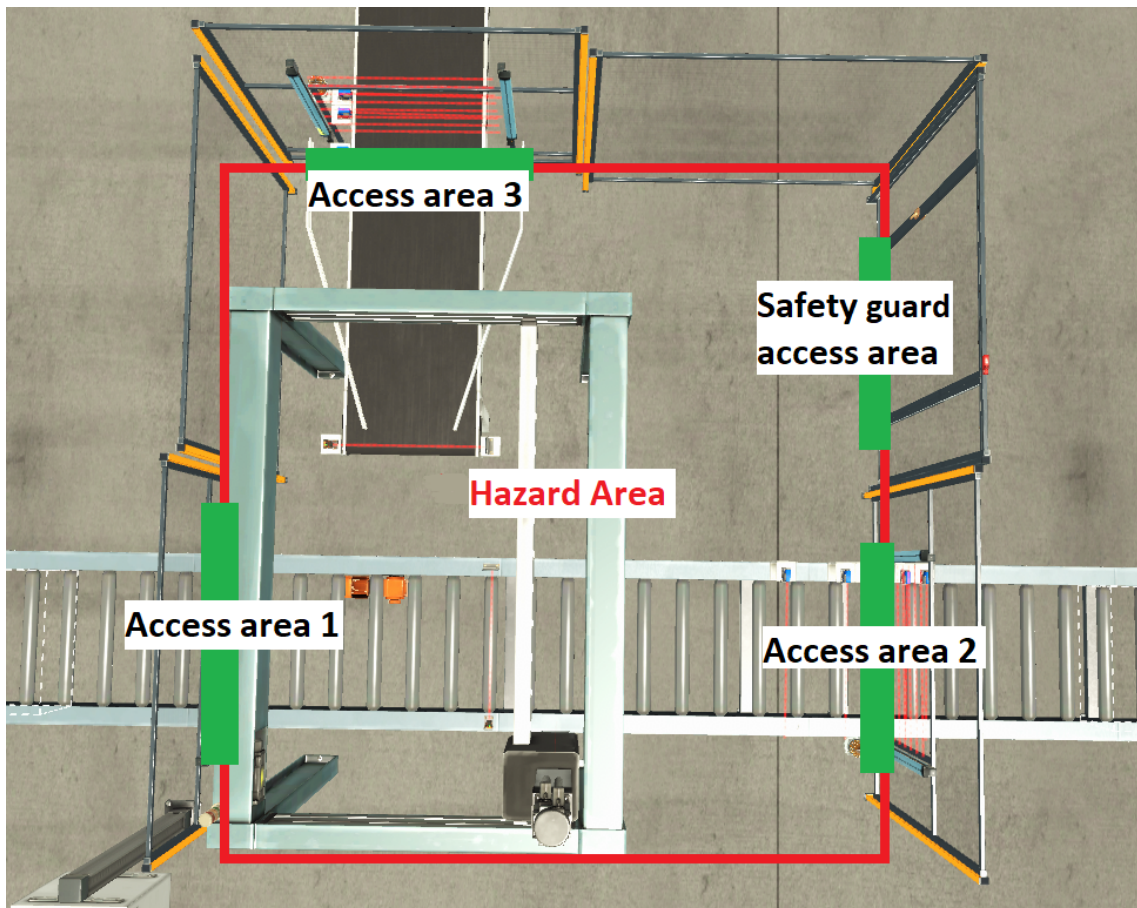


Figure 5.1: Factory I/O palletizer access areas

The safety contactor that cuts the power to the actuators is controlled by three emergency stop FBs, one for each light curtain, and a safety guard monitoring FB, the contactor is only enabled when all FBs safe outputs are enabled. The safety system will be analyzed in parts. Figure 5.2 shows a representation of the structure of the standard and safe control of the machine. The structure aligns with the one depicted in Figure 3.35.

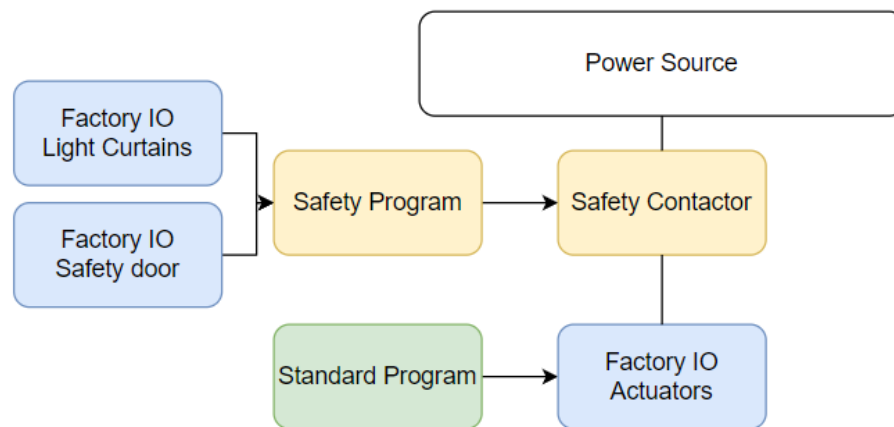


Figure 5.2: Standard and safe control of the palletizing station

The "Access area 1" only has a simple light curtain with no muting because only pallets pass this area and they don't touch the light beams, so the system will stop if anything interrupts this light curtain. The light barrier in Factory I/O controls a simulated light curtain FB in Codesys that produces two output signals, much like the emergency button, that are then connected to the "SF_Equivalent" FB to monitor the discrepancy time of the signals and convert them to a safe output. This monitoring FB is connected to the input of the emergency stop FB.

The "Access area 2" has a muting function with an override function that allows the pallets with the stacked boxes to pass. Every light curtain with a muting function must have a warning light that indicates when the function is active. The functionality is the same as with the "Light Curtain 1" with the addition of the "SF_MutingSeq" FB and an "SF_Override" FB. The override function block allows the movement of the actuators in case of an error in the muting function to remove the objects from the monitoring zone with a start override command.

The "Access area 3" has the same functions as "Light Curtain 2" except it doesn't have an override option. In case of a muting error, the box in the monitoring zone has to be removed by hand.

Finally, the "Safety guard access area" has the "SF_Guard" FB monitoring the position of the safety guard that is simulated by a switch device FB and monitored with the "SF_Equivalent" FB, if it opens the safe output variable of the emergency stop FB is disabled.

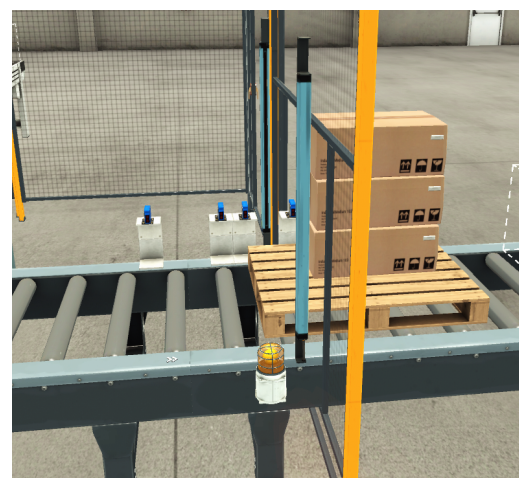
In the "Access area 2," as the stacked boxes approach the exiting conveyor, a standard control sensor activates the conveyor and the muting enable command is linked to this sensor as per PLCopen recommendations. Once the objects pass the standard sensor, the muting enable signal is interrupted to prevent a new muting cycle in the opposite direction, ensuring that muting is only possible when objects are leaving the station and preventing intentional activation of muting from outside the hazard zone. Figure 5.3 illustrates the moment when the stacked boxes are approaching the light curtain, enabling the muting function to begin. Figure 5.4a depicts the boxes exiting the hazard zone with muting active, while the muting enable command is already disabled. Finally, Figure 5.4b shows the boxes past the light curtain, where muting is disabled and not possible in any direction.



Figure 5.3: Stacked boxes approach light curtain. Muting enable command activates.



(a) Muting active and muting enable command inactive



(b) Muting and muting enable command inactive.

Figure 5.4: Stages after muting is activated.

Simulating a dangerous situation where the boxes collapse during the muting phase will trigger an error, resulting in the machine being stopped and remaining disabled until all errors are resolved, such as clearing the objects. Figure 5.5a illustrates the boxes interrupting the light curtain when the error is detected. In such cases, the error signal is transmitted to the override function block (FB), which generates an "override possible" signal, activating the override button. Pressing the button initiates the override function until the objects have exited the hazard zone, as depicted in Figure 5.5b.

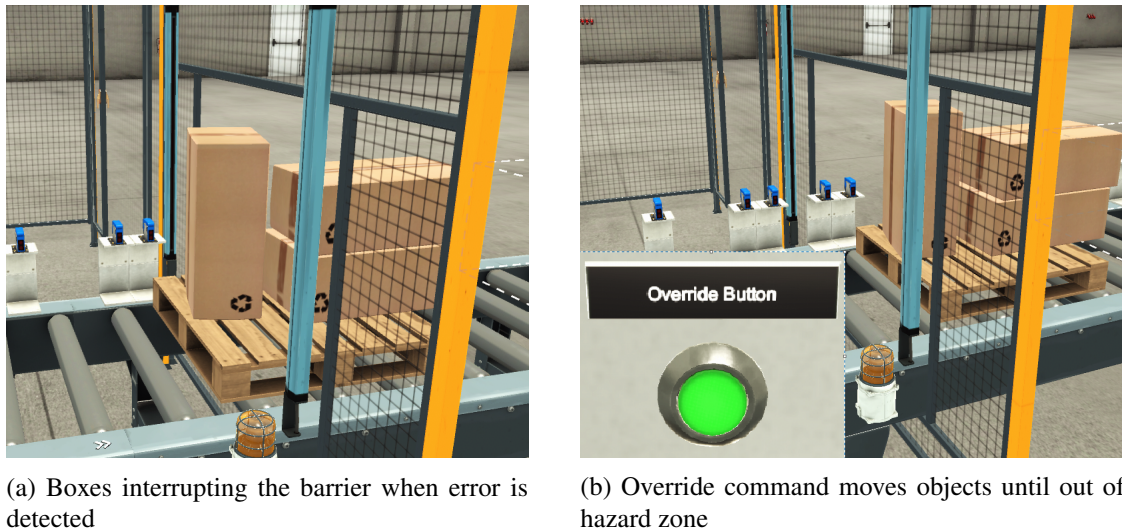


Figure 5.5: Simulation of muting error in light curtain and override action

After the hazard zones are cleared the reset command resets the safety functions and enables the safety contactor output circuits allowing the standard control program to control the machine.

This scenario exemplifies a realistic industrial situation where the safety system effectively detects hazard zone breaching and executes an emergency stop. The safety is ensured through the use of simulated safety devices and safety logic implemented with standardized function blocks. All the safety devices are controlled via Factory I/O, and faults can be injected through an interface in Codesys for testing and validation purposes.

5.2 Machining station

To simulate a computer numerical control (CNC) machining station a scenario was made in Factory I/O with the most similar objects available. Fixed guards simulate the structure of the CNC machine and a safety guard simulates the window of the machine. Inside there is a pusher object that when pushing on a "material" object simulates the machining process.

A standard program controls the machine cycle and a safety specific program executes the safety function. The safety function consists on an emergency stop when an emergency button is pressed and hazard zone access prevention through a guard lock. When the machine is safe, i.e. door closed and locked, pressing a start button starts the machining process. After the machining process is done or if an emergency occurs and a safety contactor output circuit opens, the machine stops. Figure 5.6 presents a simplified structure of the control of the machine that aligns with the one depicted in Figure 3.35. The standard program receives information from the safety program about the status of the safety function. So when an emergency occurs not only the safety function cuts the power to the actuator but the standard controller detects it and enters a safe state where the unlock request can be made after the machine has stopped.

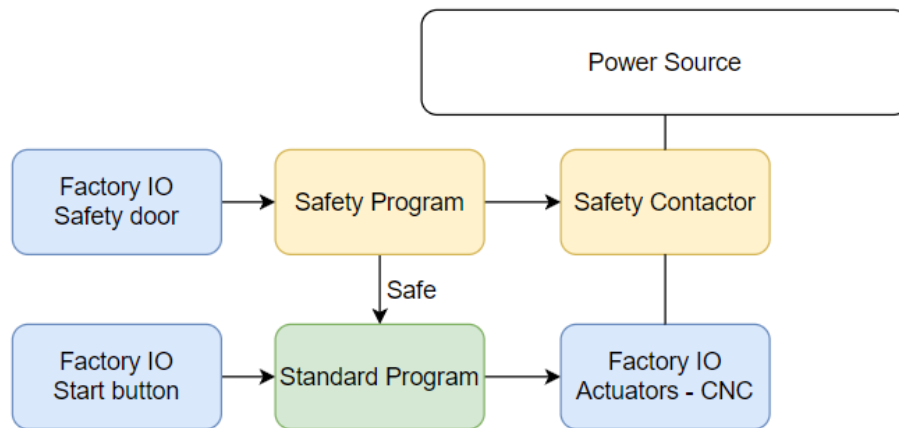
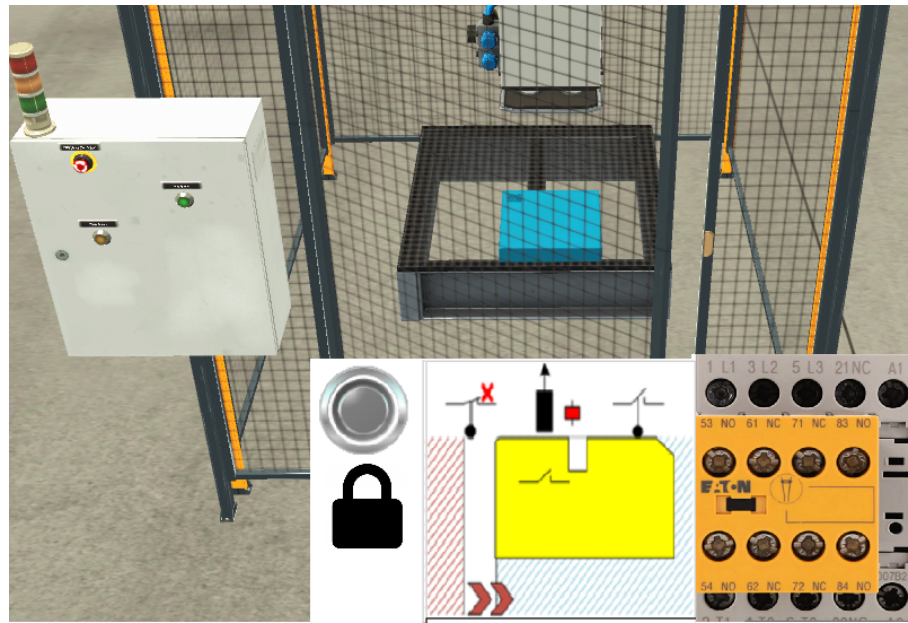
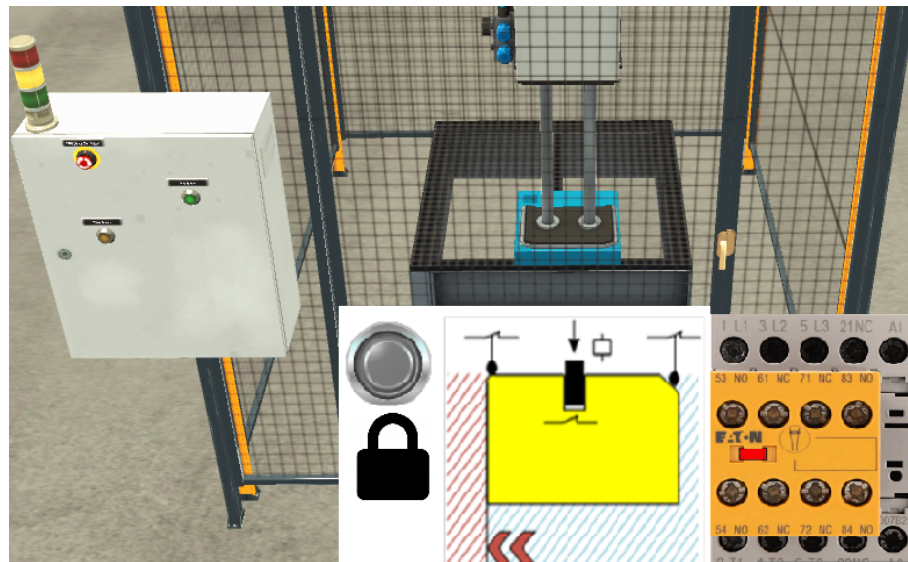


Figure 5.6: Standard and safe control structure of the CNC machine

The safety program effectively ensures that the machine operates only in a safe state by continuously monitoring the position of the safety guard and the status of the guard lock. The position of the door is monitored using the "SF_Guard" function block, which receives signals from a simulated device in Codesys linked to the door in Factory I/O. The guard lock is monitored by the "SF_GuardLocking_2" function block, which controls the locking of the safety guard and receives input from the guard monitoring function block regarding the door position. An interface in Codesys allows for observation of the guard lock state, initiation of the unlock request using a button, and injection of faults in the guard system. Figure 5.7a illustrates an unsafe situation where the door is open and unlocked, and a fault exists where one of the guard switches is stuck in a closed position, triggering the safety contactor to cut power to the machine. Figure 5.7b demonstrates a safe situation where the door is closed, the unlock request is disabled, and a reset signal closes the safety contactor output circuit, enabling the machine to operate safely.



(a) Unsafe state, machine unable to start



(b) Safe state, machine able to work

Figure 5.7: Machining station simulation

This scenario provides a simulation of a machining station, ensuring safety through the utilization of simulated safety devices and guard locking logic that adheres to PLCopen specifications. By leveraging the available features within Factory I/O, the program replicates a realistic environment where safety measures are of utmost importance. Through the integration of simulated safety devices and the implementation of guard locking logic, the program effectively safeguards the machining station, preventing hazardous situations.

5.3 Multiple zone application

Some industrial plant and production lines have "zones" that are controlled separately. Figure 5.8 shows a plant with two zones, where different machining is done in each. A work object enters Zone 1 to be machined then passes through to zone 2 to do a different operation. Safety has to be ensured in each zone. A safe state is maintained by monitoring a light curtain in the material entrance of zone 1, a light curtain between zone 1 and 2 and a safety guard in each zone. In case of a hazardous situation a safety contactor cuts the power to the corresponding zone.

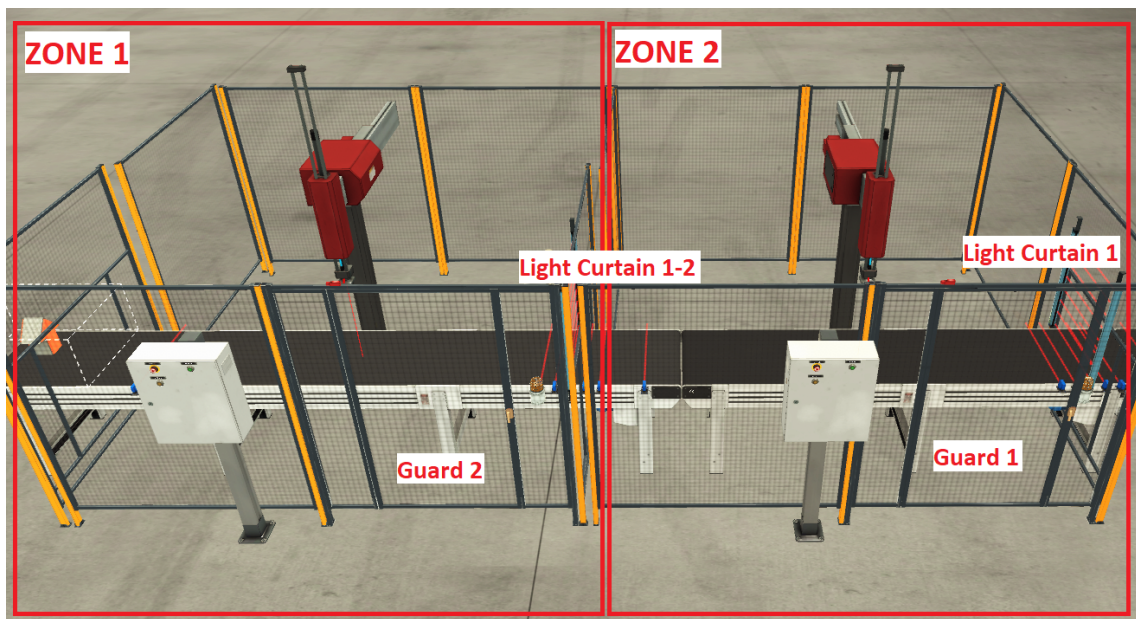


Figure 5.8: Industrial plant with two zone controlled separately

A single standard program is capable of controlling the separate zones, while the safety program ensures safety in both of them as well. The structure that corresponds to the one depicted in Figure 3.35 is presented in Figure 5.9. The safety contactor that cuts the power to zone one in case of a dangerous situation is controlled by the muting FB monitoring "Light Curtain 1" and the guard FB that monitors the safety "Guard 1". The safety contactor that cuts the power to zone 2 in case of a dangerous situation is controlled by the guard FB that monitors the safety "Guard 2". The muting FB that monitors the "Light Curtain 1-2" is an input to both the safety contactors, so an error here stops both zones.

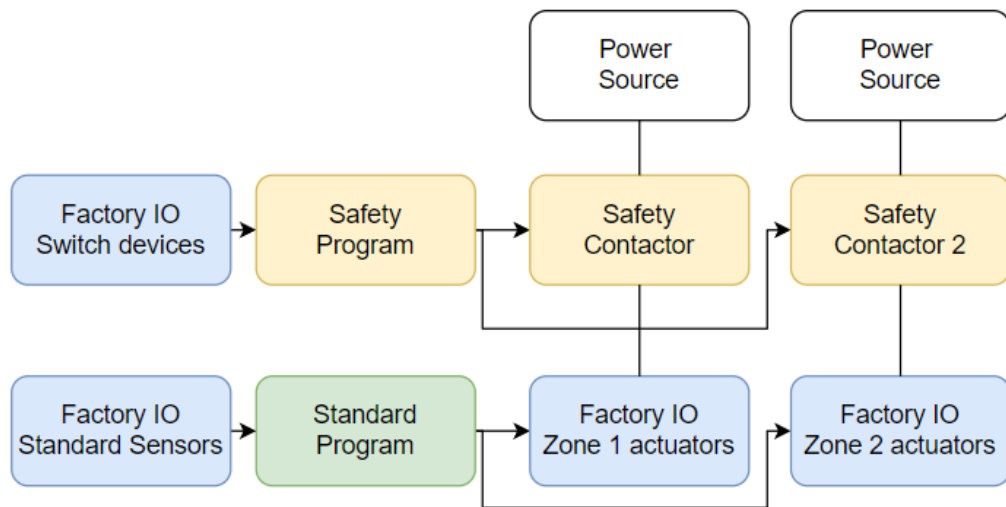


Figure 5.9: Structure of the standard and safe control of the multiple zone machine

Each zone can only work on one object at a time and standard process control sensors are used to monitor the number of objects in each zone. These standard control sensors are linked to the enable command of the light curtains muting functions, as recommended by PLCopen. This way objects can only enable the muting and pass through the light curtains to the hazard zone when needed by the standard process.

To showcase some dangerous situations and the system reaction, errors were intentionally caused. By opening the safety "Guard 1", breaching the first zone, the "SF_Guard" FB safe output is lost and the safety contactor opens its output circuits stopping the actuators of that zone. Zone 2 continues its operation as normal. This situation is shown in Figure 5.10, note that a red warning light warns of a problem with zone 1 and a green light signals normal operation in zone 2.

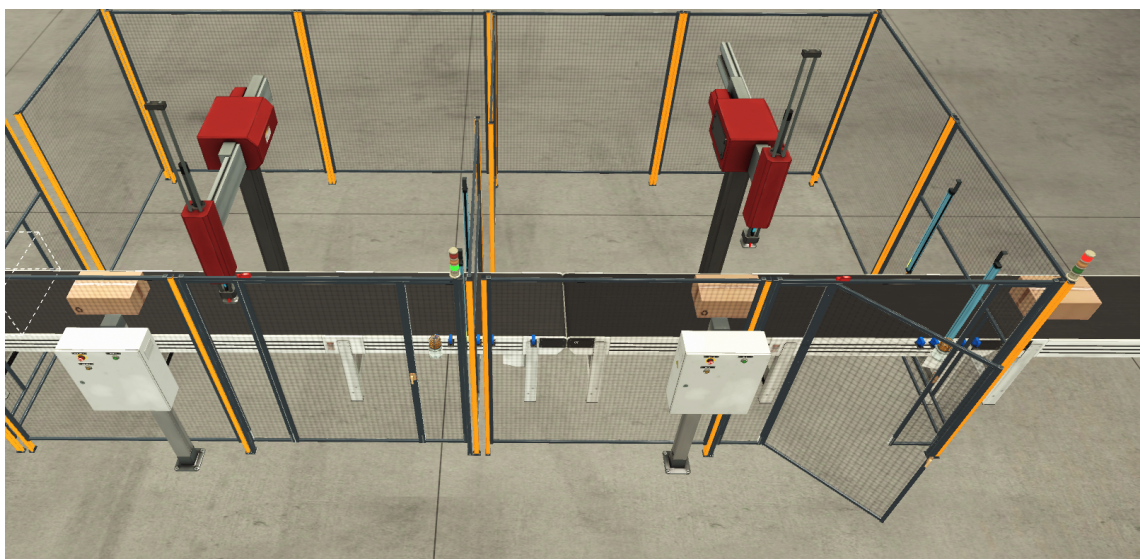


Figure 5.10: Zone 1 trespassing stops zone actuators.

Moving a random object into the light curtain between the two zones causes an error and both safety contactors cut the power to all the actuators since this means that either zone can be compromised. This situation is shown in Figure 5.11.

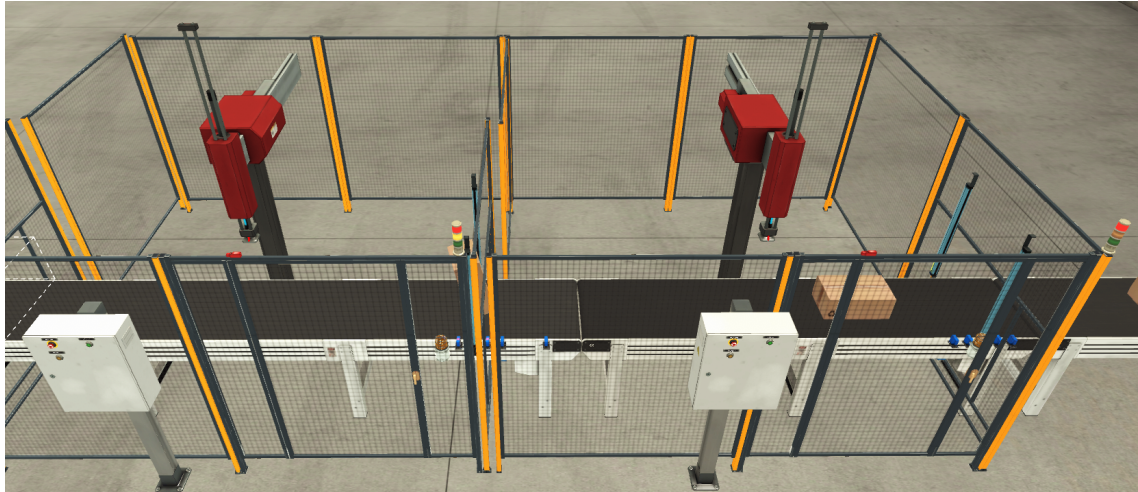


Figure 5.11: Trespassing between zones stops all actuators

This safety application demonstrates the capability of effectively controlling and ensuring safety in multiple zones within a machine. Through seamless communication between various safety functions and programs, the system enables the simultaneous monitoring of distinct hazard zones, providing robust safety measures without compromising productivity.

5.4 Concluding remarks

This chapter has discussed a variety of safety solutions that have been created for realistic circumstances, guaranteeing that safety is maintained through the use of simulated safety devices and programmed safety function blocks.

The integration of simulated safety devices has allowed for the replication of real-world scenarios, enabling accurate evaluation of safety measures. The dangers of implementing safety measures in the real applications are reduced by testing and fine-tuning them in a controlled setting.

Additionally, using programmed safety function blocks has shown to be essential for creating reliable safety systems. By encapsulating preset safety features, these function blocks provide consistent and dependable safety behavior across a range of applications. Because of their modular design, they may be easily integrated into current systems, enabling effective safety solutions.

In summary, the demonstrated advancements underscore the effectiveness and potential of simulated safety devices and programmed safety function blocks in guaranteeing safety within realistic industrial settings.

Chapter 6

Conclusions and Future Work

This dissertation focused on addressing the necessity for a dedicated simulation platform that can effectively connect the "Machinery Safety" and "Industrial Computing" curricula. By integrating the concepts of safety of machinery and PLC programming, students can acquire a comprehensive understanding of functional safety principles and practical experience in the field. The training scenarios encompass potential errors that may pose risks to both individuals and equipment, emphasizing the importance of virtual simulation environments. It was highlighted that a specialized platform tailored specifically for safety applications was required. Therefore, this dissertation proposed the creation of a platform capable of developing and testing functional safety solutions expanding the simulation ability of Factory I/O to safety-related scenarios.

The proposal entailed the development of simulations for safety devices that can introduce faults and offer visual representations of the devices. To program functional safety solutions effectively, a library of PLCopen compliant safety function blocks was necessary, guaranteeing compatibility and adherence to industry standards.

After justifying the choices and approaches taken in the development at the light of the state of the art the focus shifted towards simulating and ensuring proper interaction of basic hardwired safety components throughout chapter 3. This provided a solid foundation for the project by simulating a functional safety application centered around a safety relay. A simulation of a safety relay was developed and specific safety devices featuring visual interfaces that allow faults to be injected. The elements were combined to form a safety system responsible for executing the safety function of a scenario in Factory I/O. The results of the demonstration proved that the developed functional safety system reduces risk and has great fault tolerance. This chapter played a vital role in comprehending the fundamental aspects of functional safety and establishing the groundwork for future developments.

Moving towards more intricate solutions in chapter 4, the development procedure of PLCopen-compliant safety-related function blocks was outlined. Emphasizing the importance of adhering to the PLCopen technical specifications, this ensured conformity with standard and governmental requirements. The challenges associated with programming the function blocks were thoroughly examined, and effective solutions were presented. The testing phase of the function blocks not

only aided in detecting programming errors but also unveiled additional benefits, such as providing insights into potential future applications. As a result, a comprehensive collection of safety function blocks representing the essential and frequently used functional safety functions was developed. In addition to the safety function blocks, a list of simulated safety devices was created to complement the formation of realistic and complete safety systems.

In order to provide tangible evidence of the effectiveness and feasibility of the approach taken, practical safety solutions were implemented. The significant advantages of integrating programmable safety solutions based on the developed safety function blocks were demonstrated in typical "unsafe" industrial automation scenarios in Factory I/O, often discussed in "Industrial Computing" contexts. By incorporating safety device simulations, controlled testing and verification of these safety solutions were carried out. The efficacy of the solutions was assessed by simulating hazardous events and analyzing the response of the safety system.

Although the presented solution is in an embryonic phase, it is possible for students to use their PLC programming and machinery safety knowledge from classes to develop and test machinery safety systems for scenarios discussed in class. The modularity of the developed library of safety function blocks makes the programming of safety functions easy and compliant to relevant safety standards. The structure created in Codesys is easily accessible to students who already have practice in the environment from "Industrial Computing". Thus, it is possible to conclude that the present dissertation achieved the objectives outlined at the beginning. However, individuals without prior knowledge of Codesys may find the developed platform lacking in user-friendliness. Furthermore, the potential of the presented platform to generate safety solutions is limited by the simulated safety devices and the available features within Factory I/O.

The author's personal experience in developing this dissertation can be described as a rewarding challenge. Through this project, the author gained valuable and detailed insights into machine functional safety, which is highly relevant for the possibility of seeking a career in an industrial setting. This work facilitated a deeper understanding of the concepts previously learned in the "Machinery Safety" and "Industrial Computing" curricular units, while also enhancing the author's programming skills. Overall, the project provided an opportunity for knowledge expansion and skill development in the field of machine functional safety and PLC programming.

6.1 Future Work

Considering the didactic objective of this dissertation, it is advisable to create additional materials with a similar aim, which involve the creation of platforms capable of assessing students at a practical level.

Another proposal for future endeavors is to apply the same methodology from this project to extend Factory I/O for motion control scenarios, in accordance with the PLCopen motion standard. This initiative can allow further bridging of PLC programming classes with motion control classes.

Due to the challenges in simulating representative safety devices and their corresponding functions, certain safety function blocks, such as two-hand control or pressure-sensitive equipment,

were not implemented. As a potential avenue for future research, it is recommended to explore methods for incorporating more complex safety functions and devices within Factory I/O.

References

- [1] Real Games, “Next-gen PLC training.” Available: <https://factoryio.com/>, 2023. Accessed on May 10, 2023.
- [2] PLCopen, “PLCopen safety - integrating safety functionality into the development system.” Available: https://plcopen.org/sites/default/files/downloads/plcopen_safety_overview.pdf, 2021. Accessed on May 3, 2023.
- [3] PLCopen, “Part 1: Concepts and function blocks.” Available: https://plcopen.org/system/files/downloads/plcopen_safety_part_1_version_2.01.pdf, 2020.
- [4] Siemens AG, “Emergency stop shutdown to SIL 1 or PL c with a 3SK2 safety relay.” Available: <https://support.industry.siemens.com/cs/document/109485642/emergency-stop-shutdown-to-sil-1-or-pl-c-with-a-3sk2-safety-relay?dti=0&lc=en-AR>, 2009-2023. Accessed on June 8, 2023.
- [5] Pilz GmbH & Co., “PNOZ s5.” Available: https://www.pilz.com/download/open/PNOZ_s5_Operat_Man_21397-EN-13.pdf, 2019.
- [6] K. Rástočný, J. Ždánky, and J. Hrbček, “The problems related to realization of safety function with SIL4 using PLC,” pp. 1–5, 01 2020.
- [7] M. Kassem, “The importance of engaging engineering and construction learners in virtual worlds and serious games,” 2013.
- [8] M. Dado, L. Kotek, R. Hnilica, and Z. Tůma, “The application of virtual reality for hazard identification training in the context of machinery safety: A preliminary study,” *Manufacturing Technology*, vol. 18, no. 5, pp. 732–736, 2018.
- [9] M. Kassem, L. Benomran, and J. Teizer, “Virtual environments for safety learning in construction and engineering: seeking evidence and identifying gaps for future research,” *Visualization in Engineering*, vol. 5, no. 1, p. 16, 2017.
- [10] B. Z. Pérez, M. M. Marín, and E. I. Pérez, “Developing a virtual environment for safety training,” in *Electronics, Robotics and Automotive Mechanics Conference (CERMA 2007)*, pp. 545–550, 2007.

- [11] Real Games, “FACTORY I/O User Manual - Parts.” Available: <https://docs.factoryio.com/manual/parts/>, 2023. Accessed on May 10, 2023.
- [12] Real Games, “FACTORY I/O User Manual - I/O Drivers.” Available: <https://docs.factoryio.com/manual/drivers/#io-drivers>, 2023. Accessed on May 10, 2023.
- [13] Al Douglas Robotics, “ERT405 Safety PLC simulation overview.” Available: <https://youtu.be/-nLQT6CBep4>, 2020. Accessed on May 10, 2023.
- [14] T. Bömer and D. M. Schaefer, “Differences between using standard components or safety components to implement safety functions of machinery.” Available: https://www.dguv.de/medien/ifa/en/prg/en13849/standard_components.pdf, 2011. Accessed on June 3, 2023.
- [15] Siemens Industry, Inc., “Evolving from safety relays to safety PLCs.” Available: <https://www.goies.com/S7-1200F-Evolving-from-safety-relays-to-safety-pls-wp-1118.pdf>, 2018. Accessed on June 3, 2023.
- [16] H. Kanamaru, T. Mogi, and N. Aoyama, “Functional safety application using safety PLC,” in *SICE Annual Conference 2007*, pp. 2489–2492, 2007.
- [17] Pilz GmbH & Co., “PNOZelog safety relay.” Available: <https://www.pilz.com/en-INT/products/relay-modules/safety-relays-protection-relays/pnozelog-safety-relays>, 2018. Accessed on June 3, 2023.
- [18] Eaton, “Safety PLC easysafety.” Available: <https://www.eaton.com/br/pt-br/catalog/industrial-control--drives--automation---sensors/easysafety-programmable-safety-relays.html>, 2023. Accessed on June 3, 2023.
- [19] CODESYS, “Why CODESYS?.” Available: <https://www.codesys.com/the-system/why-codesys.html>, 2023. Accessed on May 10, 2023.
- [20] K.-H. John and M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems*. Springer Berlin, Heidelberg, 1 ed., 2010.
- [21] PLCopen, “IEC 61131-3: a standard programming resource.” Available: https://plcopen.org/sites/default/files/downloads/intro_iec_oct2016.pdf. Accessed on June 6, 2023.
- [22] PLCopen, “PLCopen safety introduction.” Available: <https://youtu.be/es2WI8W3LgY>, 2021. Accessed on May 3, 2023.
- [23] IEC, “Functional safety of electrical/electronic/programmable electronic safety-related systems.” Standard, 2010. IEC 61508:2010.