Programming Guideline Safety for SIMATIC S7-1200/1500

SIMATIC Safety Integrated

https://support.industry.siemens.com/cs/ww/en/view/109750255

Siemens Industry Online Support

Legal information

Use of application examples

Application examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The application examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The application examples merely offer help with typical tasks; they do not constitute customer-specific solutions. You yourself are responsible for the proper and safe operation of the products in accordance with applicable regulations and must also check the function of the respective application example and customize it for your system.
Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the application examples used by technically trained personnel. Any change to the application examples is your responsibility. Sharing the application examples with third parties or copying the application examples or excerpts thereof is permitted only in combination with your own products. The application examples are not required to undergo the customary tests and quality inspections of a chargeable product; they may have functional and performance defects as well as errors. It is your responsibility to use them in such a manner that any malfunctions that may occur do not result in property damage or injury to persons.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness and freedom from defects of the application examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.
By using the application examples you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the application examples at any time without notice. In case of discrepancies between the suggestions in the application examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.
The Siemens terms of use (https://support.industry.siemens.com) shall also apply.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.
In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.
Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.
For additional information on industrial security measures that may be implemented, please visit https://www.siemens.com/industrialsecurity.
Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.
To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under https://www.siemens.com/cert.

© Siemens AG 2024 All rights reserved

Table of contents

© Siemens AG 2024 All rights reserved

1 Introduction

The controller generation SIMATIC S7-1200 and S7-1500 has a modern system architecture and, together with TIA Portal, offers efficient possibilities for programming and configuration.

This programming guideline enables you to:

- Reduce the CPU stops
- Reduce compilation times
- Effect fewer and easier acceptances

This document provides you with many recommendations and information for the optimal configuration and programming of S7-1200/1500 controllers. This helps you create standardized and optimal programming of your automation solutions.

The examples described can be used universally on the S7-1200 and S7-1500 controllers.

Advantages

Following the recommendations given here provides you with many advantages:

- Reusability of program components
- Easier acceptance (code review, error detection and correction)
- Increased flexibility in case of program changes
- Reduction of programming errors
- Increased plant availability by avoiding CPU stops
- Easier readability for third parties
- Reduced runtime of the safety program

Note Not all the recommendations provided in this document can be applied at the same time. In these cases, it is up to you as the user to decide on the prioritization of the recommendations (e.g., standardization or runtime optimization of the safety program).

Programming guideline and style guide

The same recommendations given in the programming guideline and the programming style guide always apply to programming safety programs.

Programming Guideline for SIMATIC S7-1200/1500:

https://support.industry.siemens.com/cs/ww/en/view/90885040

Programming Style Guide for SIMATIC S7-1200/1500:

https://support.industry.siemens.com/cs/ww/en/view/109478084

Guideline on Library Handling in TIA Portal:

https://support.industry.siemens.com/cs/ww/en/view/109747503

This document is a supplement to the documents above and deals with special aspects of programming safety programs with STEP 7.

Note Independent of this document, the statements in the manual "SIMATIC Safety - Configuring and Programming" must be observed – especially warnings contained therein must be strictly observed. Non-compliance means that death or serious injury may occur if proper precautions are not taken.

Warning notice system

This document contains notices that you have to observe in order to ensure your personal safety, as well as to prevent damage to property. Notices relating to your personal safety are highlighted by a warning triangle; notices relating to material damage only do not have a warning triangle. Depending on the hazard level, the warnings are displayed in descending order as follows.

 **DANGER**	Indicates that death or severe personal injury will result if proper precautions are not taken.
 **WARNING**	Indicates that death or severe personal injury may result if proper precautions are not taken.
 **CAUTION**	Indicates that minor personal injury may result if proper precautions are not taken.
NOTICE	Indicates that material damage may result if proper precautions are not taken.

If more than one level of danger exists, the warning notice for the highest level of danger is used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

2 Configuring fail-safe controllers

2.1 Selecting the suitable F-CPU

The selection of the F-CPU depends on the following factors:

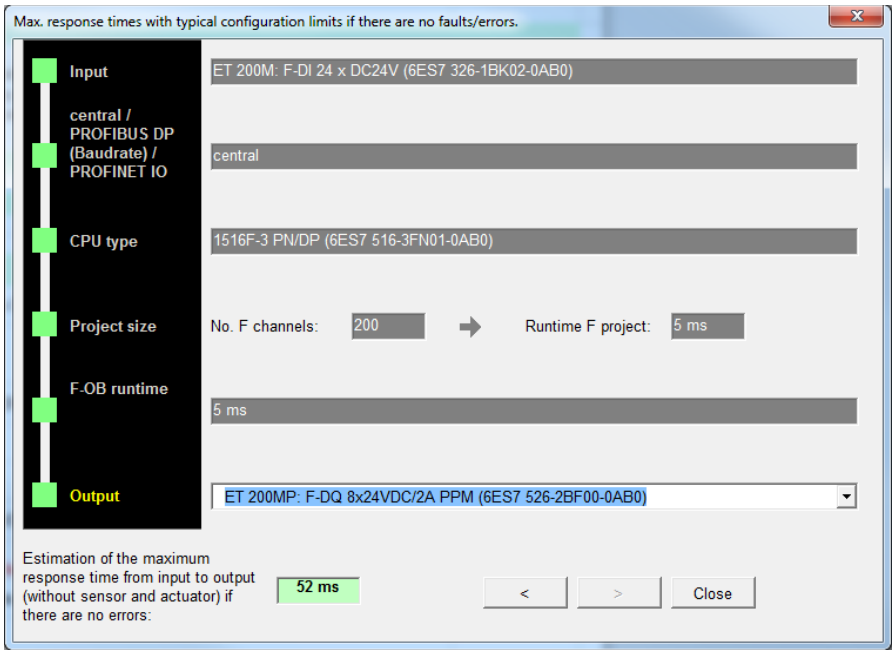
- Runtime of the safety program
- PROFIsafe communication time
- Response time of the safety function
- Number of required inputs and outputs
- Number of connected I/O devices
- Memory requirements of the program

Estimate of the response time

If you already have a rough idea of the automation system you want to use, you can estimate the response time of your safety program using the SIMATIC STEP 7 Response Time Table or go through various scenarios to select the suitable F-CPU:

https://support.industry.siemens.com/cs/ww/en/view/93839056

Figure 2-1: Response time wizard of the SIMATIC STEP 7 Response Time Table



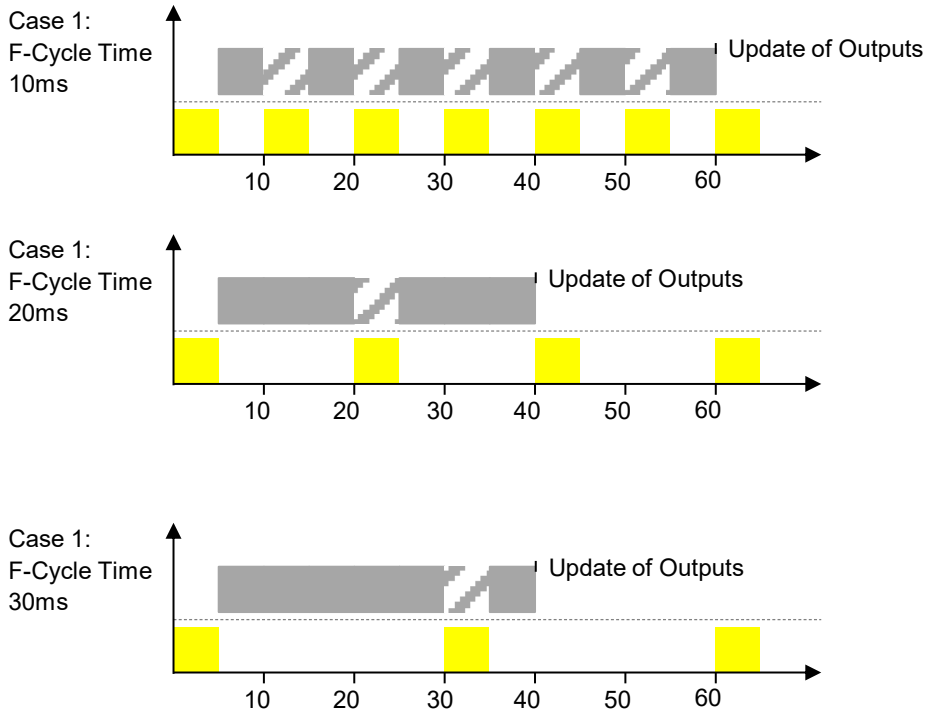
Influence of the safety program's cycle time on the standard user program

A long cycle time of the safety program slows down the response time of your safety functions but allows more time for processing the standard user program.

A short cycle time of the safety program increases the response time of your safety functions but allows less time for processing the standard user program.

The following figure shows the influence of the cycle time of the safety program of the event class "Cyclic interrupt" on the time that is available for processing the standard user program.

Figure 2-2 Influence of the safety program's cycle time on the standard user program



Note

Please note that higher-priority organization blocks (e.g., cyclic interrupt OBs or Motion Control OBs) can also interrupt the safety program in the same way as shown in Figure 2-2.

To make sure that the safety program cannot be interrupted, you can customize the priorities in the properties of the appropriate OBs.

Note

If the cycle time is lower than the operating duration of the safety program, the CPU switches into the STOP state.

Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 5.2 - Defining F-runtime groups.

2.2 F-change history

F-change history acts like the standard user program's change history. In the project tree, "Common data > Logs", one F-change history is created for each F-CPU.

Recommendation

Activate the change history when you start configuring or, at the latest, when you have defined the final project-specific CPU name as the change history is linked to the CPU name.

Advantages

- Ensures that the last change was loaded by comparing the online and offline status of the CRC (Cyclic Redundancy Check).
- Which user changed or downloaded the safety program can be tracked in multi-user projects.
- Synchronization of online and offline status without an online connection between CPU and PG/PC.

NOTICE	F-change history must not be used to detect changes in the safety program or when accepting changes in the F-I/O configuration.

Note Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 10.8 – F-change history

2.3 Consistent uploading of F-CPUs

TIA Portal V14 SP1 and higher allows you to consistently upload fail-safe SIMATIC S7-1500 CPUs from the automation system to TIA Portal.

Recommendation

An upload from the automation system is only possible if the project has been released for it.

When you start configuring, select the "Consistent upload" check box in Safety Administration in TIA Portal.

Advantages

A programmer on the system can load the respective program onto their PG and thus reduce the service effort.

Note The activation of the option for the consistent upload from the F-CPU extends the time for loading the safety-related project data. In addition, more load memory is required on the F-CPU.

2.4 Know-how protection

STEP 7 Safety V14 or higher allows you to activate know-how protection for fail-safe blocks (FCs and FBs).

Know-how protection protects specific program parts against access by unauthorized persons, regardless of the F-CPU's and the safety program's access protection. The contents of an FC or FB cannot be viewed or modified without a password.

Recommendation

During the project phase, determine to what extent it makes sense to protect the blocks of a safety program against third-party access.

Advantages

- Protects your know-how relating to contents of program parts.
- Accepted blocks cannot be modified.

Additional information

The following documentation provides instructions for using know-how protection for different scenarios:

Know-how protection in fail-safe programs:
https://support.industry.siemens.com/cs/ww/en/view/109742314

3 Methods for Safety Programming

3.1 Program structures

Recommendation

When creating a program, make sure that your program is designed to be reusable. Rules and recommendations for programming can be found in the document Programming Style Guide for SIMATIC S7-1200/S7-1500 https://support.industry.siemens.com/cs/ww/en/view/109478084.

3.1.1 Defining the program structure

Recommendation

- Structure the program code modularly, e.g.
  - into sub-areas for sensing, evaluating, responding or
  - By safety functions or
  - By plant units
- Create a specification for each module in advance (based on the requirements of the risk assessment).
- Avoid complex signal paths.

Advantages

- Complexity is minimized.
- Programming errors are reduced.
- Allows the program code to be analyzed/tested without running the program (e.g., code review or PLCSIM).
- Easier expandability and simplification of repeated acceptance.
- Reusability of program sections without repeated acceptance.
- Finished program parts can be tested and accepted in advance.

Example

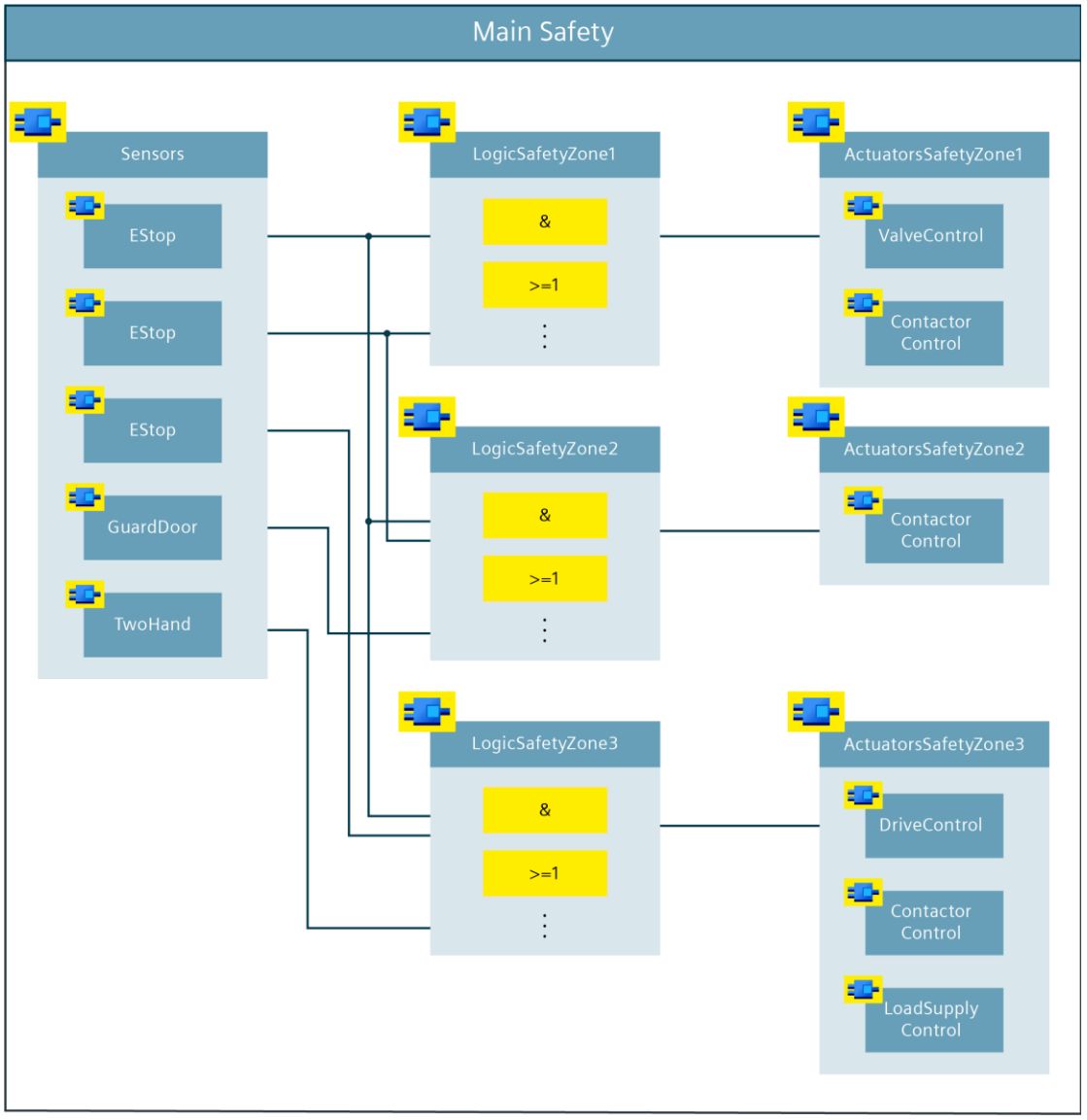
The following figure shows a safety application that is divided into three machine areas (safety zones).

As some of the sensor signals are interconnected across areas (e.g., emergency stop functions that act globally), they are grouped into a "Sensors" FB (they could also be split up into physical or logical areas). The respective sensors are evaluated using standardized function blocks (e.g., "GuardDoor").

The Mobile Panels' blocks are also called here.

Separate logic and actuator FBs are created for each machine area. The respective actuators are controlled using standardized function blocks (e.g., "ContactorControl").

Figure 3-1: Example of a program structure



Note

The structure shown here is an example. Depending on the size and complexity of the safety program, you can also choose a different structure. In smaller applications, it would also be possible to implement the logic and actuator control in a shared function block, for example.

3.1.2 Safety Unit

Software Units offer a convenient way to structure your TIA Portal project. By using the Safety Unit, the safety program can be part of this structure.

The following requirements must be fulfilled in order to use a Safety Unit:

- As of TIA Portal V18/STEP 7 V18
- F-CPU S7-1500 as of FW V2.6
- Under "Extras/Settings/STEP 7 Safety" the option box "Manages safety program in the 'Safety Unit' environment" is selected.
- The F-CPU is recreated.

You receive information on the data exchange between Safety Unit and Standard Units in chapter 3.7.3.

Recommendation

We recommend using the Safety Unit.

Advantages

- Better clarity
- Time reduction thanks to standardization
- Easy assurance of data integrity

3.1.3 Call levels of F-FBs/F-FCs

For safety programs, you can use a maximum of eight call levels. A warning appears when this limit is exceeded and an error message is displayed for pure FC and multi-instance call chains.

Note

On the system side, functions are mapped as FBs with a multi-instance call in the protection program; this is the reason why an error message is also displayed for FC call chains with more than eight call levels.

The program structure in Figure 3-1 shows one way of keeping the call levels flat so that the safety program remains within the limits specified here.

3.1.4 Call sequence of the blocks in the Main Safety

Recommendation

Within the Main Safety, call blocks in the following sequence:

1. Receive blocks from other CPUs (F-CPU-to-F-CPU communication)
2. Error acknowledgment/reintegration of F-modules/F-channels
3. Evaluation block of the sensors
4. Operating mode evaluation
5. Logic operations, calculations, evaluations, etc.
6. Control blocks for safe actuators
7. Send blocks to other CPUs (F-CPU-to-F-CPU communication)

Advantages

- The CPU always uses the latest values
- Facilitates orientation in the Main Safety

Note

Additionally, with pre-processing/post-processing, you have the option of calling standard blocks (FCs) directly before or after an F-runtime group, e.g. for data transfer during fail-safe communication using Flexible F-Link.

3.1.5 F-compliant PLC data types

For safety programs, too, it is possible to optimally structure data using PLC data types.

Recommendation

- Create F-compliant PLC data types (F-UDTs) to also structure data in the safety program.
- Use F-compliant PLC data types to transfer large numbers of tags to blocks.
- Make use of the possibility of nesting F-compliant PLC data types.

Advantages

- A change in a PLC data type is automatically updated in all points of use in the user program.
- Greater transparency through structuring of the data.

Note

Try to design the F-compliant PLC data types as modularly as possible to achieve reusability of the data types as well as of the blocks.

Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 5.1.5 – F-compliant PLC data types

Example

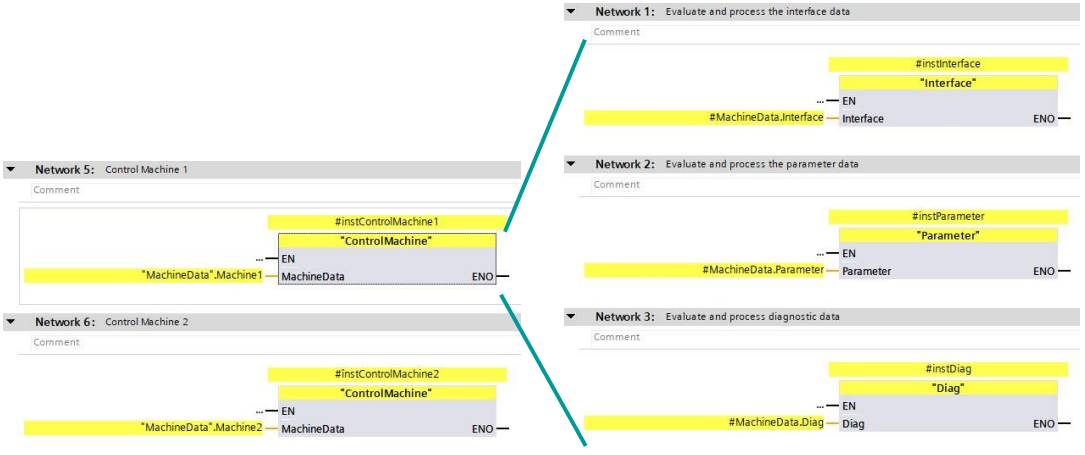
The example below shows the use of F-compliant PLC data types. The F-UDT "typeMachine" (Figure 3-2) contains machine-related data. The data is structured through the use of the further F-UDTs "typeInterface", "typeParameter" as well as "typeDiag" and nesting. Figure 3-3 shows how to access the respective data.

Figure 3-2 Nested F-compliant PLC data type

typeMachine

	Name	Data type
1	Interface	"typeInterface"
2	Parameter	"typeParameter"
3	Diag	"typeDiag"

Figure 3-3 Use of nested F-UDTs



3.1.6 Block information and comments

General

In SIMATIC Safety, the Function Block Diagram (FBD) and Ladder Diagram (LAD) programming languages are available to you. Both languages provide the option to store block and network comments.

Comments have no influence on the signature of F-FBs/F-FCs and can therefore also be edited after acceptance.

Recommendation

In the block comment of your block, enter formal information about the block with the aid of the following template.

```
  (*Add your company*) / (c)Copyright 2022
-------------------------------------------------------------------------------
Title:
Comment/Function:
Library/family:
Author:
Tested with:
Engineering:
Restrictions:
Requirements:
DC: |
Category:
-------------------------------------------------------------------------------
Change log table:
Version | Date | Expert in charge | Changes applied
---------|------------|------------------------|-----------------------------
01.00.00 | 10.10.2022 | *add your Signature here* | *add the expert's name*
|First released
```

If you implement diagnostic functions relevant to the PL or SILCL of another subsystem (Detect or Evaluate) in an F-FB, also enter normative parameters such as PL or SILCL and category (according to ISO 13849-1), DC measures, CCF measures, etc. in the block comment.

After successful acceptance of the block, also enter the signature in the block comment. This makes it easier to track functional changes of the block.

3.2 Functional identifiers of tags

Safety often uses the terms 'shutdown' or 'shutdown signals'. In practice, a safety function is described using this terminology:

"When a safety door is opened, drive XY must be safely shut down."

However, release signals are generally programmed in the technical implementation as a safety program. This is due to the fact that safety interconnections are designed based on the closed-circuit principle.

If, for example, a safety door is closed, it gives the enable to switch on a safe actuator.

Recommendation

Before the start of the project, define a uniform identifier for the tags with the appropriate suffixes. The identifier reflects the meaning and purpose of the tags in the context of source code.

Choose the tag identifier so that it reflects the logic state "1" ("true").

For example, "maintDoorEnable" or "conveyorSafetyRelease".

Note Please note that the standardized names of the drive functions (e.g., STO and SLS) according to IEC 61800-5-2 do not comply with the above recommendation.

3.3 Standardizing blocks

Aside from the actual evaluation of a sensor / control of an actuator, the same conditioning of input and output parameters is often necessary (e.g., edge evaluation, time functions, acknowledgment, etc.).

To this end, it is useful to create and reuse modular blocks.

Block libraries

Siemens Industry Online Support provides block libraries you can use in your project

- LSafe, TÜV-checked library for basic safety functions.
  https://support.industry.siemens.com/cs/ww/en/view/109793462
- LDrvSafe offers fail-safe blocks in interaction between CPU, SINUMERIK ONE; SINAMICS via PROFIsafe and SIMATIC Micro-Drive.
  https://support.industry.siemens.com/cs/ww/en/view/109485794

Recommendation

Create modular blocks that you can reuse:

- Blocks for typical fail-safe sensors
- Blocks for typical fail-safe actuators
- Blocks for frequently used functions (e.g., reintegration, operating mode)

Advantages

- Reused blocks only have to be accepted once
- Faster programming of further functions and projects
- Versioning with the TIA Portal library concept possible
- Standardization of formal parameters across projects and programmers, resulting in easy readability and testability

Note The following block programming shows examples. The actual function depends on the application's risk assessment or the project requirements.

3.4 Programming logic operations

Task of the blocks

- Generate release signals to control the safety-related actuators based on the relevant safety functions
- Link the sensor enables, operating mode enables, etc. to the control signals of the actuators

Recommendation

- Use mainly AND and OR logic elements
- Avoid jumps to binary logic

3.5 Programming operating mode-dependent safety functions

Recommendation

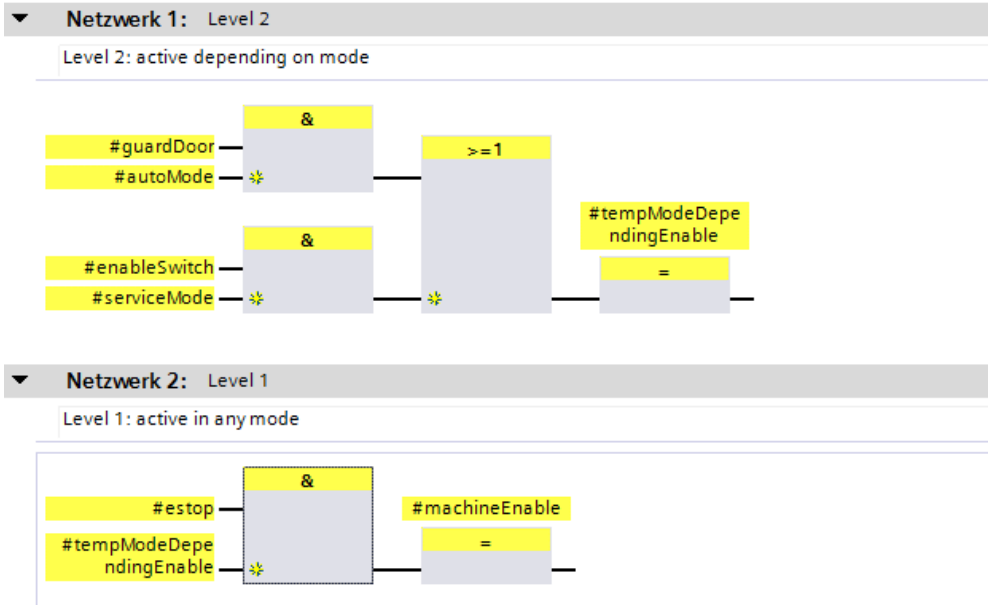
Divide the logic into different levels:

- Level 1: All safety functions that are independent of operating modes and plant statuses.
  - Logic ANDing of all safety functions that are always active.
  - These are typically emergency stop facilities.
- Level 2: All safety functions that are dependent on operating modes.
  - Logic ORing of safety functions that are only active in certain operating modes.
  - For example, safety doors in automatic mode, alternating with enabling buttons in service mode.

Example

Three safety functions are implemented on a machine: The "estop" emergency stop function is active in each mode. The "guardDoor" safety door monitoring and the "enablingSwitch" enabling function are only active in one operating mode.

Figure 3-4: Programming operating modes



3.6 Connection of global data

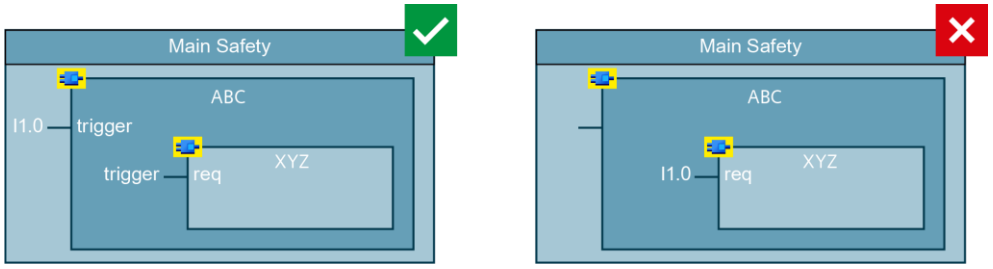
Recommendation

- Connect global data (inputs, outputs, data blocks) in the highest level of the block hierarchy (Main Safety).
- Use the block interfaces to pass signals to lower levels.

Advantages

- Modular block concept
- Reuse of program parts in other projects without modifications
- Programming errors are reduced
- The overall program is easier to read because the general function of a block can already be deduced based on the interfaces.

Figure 3-5: Connection of global data



3.7 Data exchange between standard user program and safety program

In principle, the safety program has the task of executing all functions that represent a risk-reducing measure. All other operational functions, as well as functions for operation and maintenance, belong to the standard user program.

Since in practice information for the diagnostic and reporting concept is also generated in the safety program and operational information is also relevant for the safety program, the two parts of the program cannot be completely separated.

To outsource non-safety-relevant functions to the standard user program, a clearly defined interface is recommended.

Note Please also refer to the information in the SIMATIC Safety - Configuration and Programming manual - Chapter 8.1 - Data exchange between standard user program and safety program

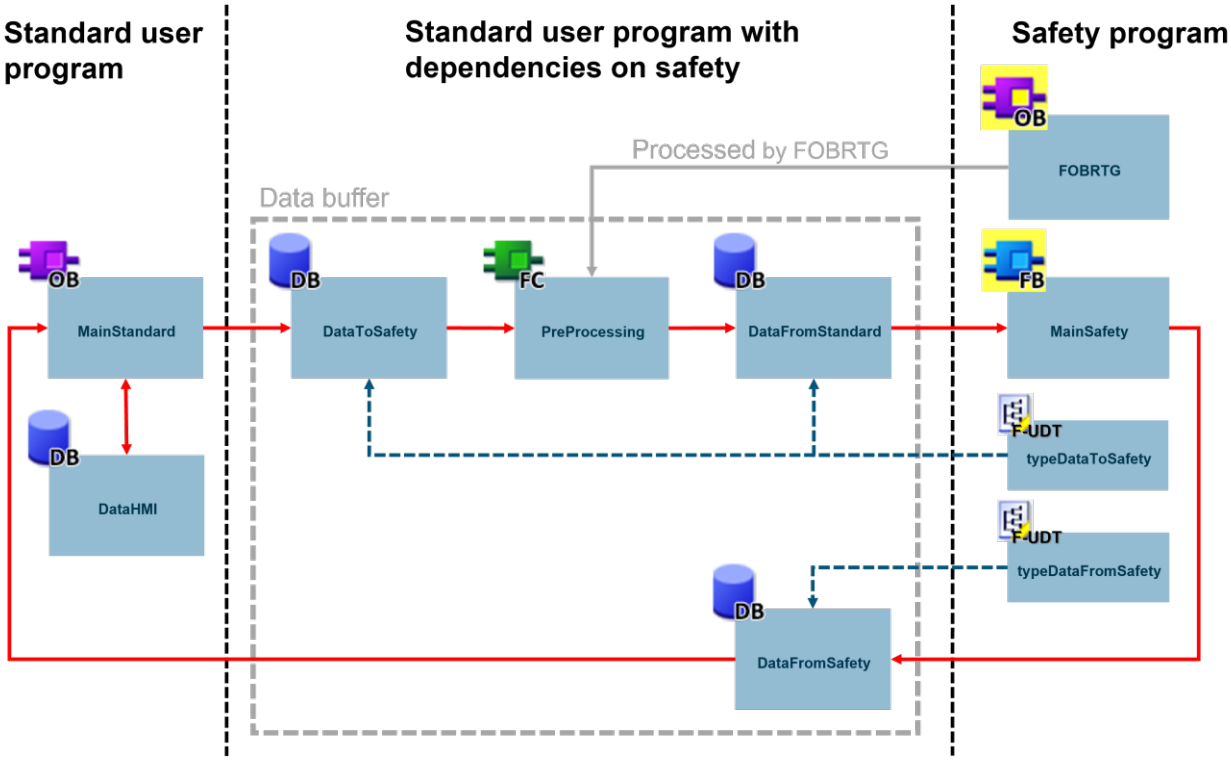
Note Each time the safety program accesses standard variables, the compiler encodes them separately. For performance reasons, it may therefore make sense to follow a slightly modified procedure.

More information can be found in chapter 4.1.4.

3.7.1 Basics of Transfer

Use three standard data blocks (also coupling DBs) for transfer to exchange data between the user program and the safety program. Separate the information transfer into a first block, which transfers information from the standard program to the safety program, a second block, into which the data can be copied, and a third, which transfers data in the opposite direction. An overview can be found in Figure 3-6.

Figure 3-6 Process overview for data exchange between Standard and Safety



F-UDT

Use data blocks of type F-UDT. The data block contains exactly the variables defined in the F-UDT. This ensures that changes to the interfaces can only be made by users with a safety password.

Note

Changes to standard blocks that are read or written by the safety program cause the F program to lose its consistency. A recompile of the safety program is necessary, a download to the CPU is only possible via CPU STOP.

When you make changes to such standard blocks, you are prompted to enter the F password since TIA Portal V16. With the help of F-compliant PLC data types (F-UDTs), you can protect the interface from changes without a safety password.

Preprocessor

Copy the data from the standard program into another data block in the preprocessing of the F runtime group. This ensures that no data is changed during the processing of the safety program.

Note

Changes to safety-related data during the execution of the runtime group will cause the CPU to stop.

Data processing

Make sure that the respective further processing of the data, for example for a diagnostic and reporting concept, is carried out within the standard program to keep the safety program compact.

Runtime group

Use separate coupling DBs for each runtime group.

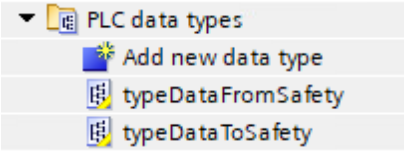
Safety Units

The data exchange between the Standard Unit and the Safety Unit works in the same way as the exchange between the standard program and the safety program.

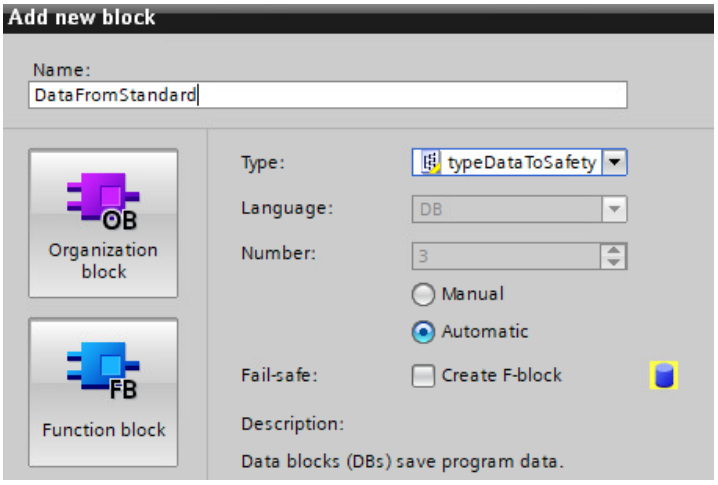
Accordingly, the same programming properties apply when dealing with units as in the standard. Note that the data blocks must be published. This is displayed in the TIA Portal with a green box.

3.7.2 Step-by-step instruction

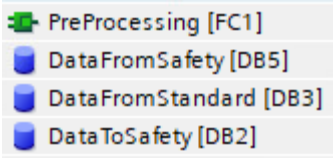
1. Create an F-UDT for transmission in each direction and fill it with the data to be transmitted.



2. Create three data blocks, two of type "typeDataToSafety" and one of type "typeDataFromSafety".

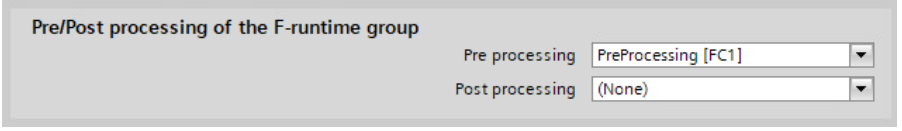


3. Additionally, create an FC "PreProcessing". Use this to copy the data from the "DataFromSafety" to the " DataToSafety " data block.




```
1  "DataFromStandard" := "DataToSafety";
```

4. Open the Safety Administration, choose appropriate runtime group and call the FB "PreProcessing" in the preprocessing of the respective runtime group:

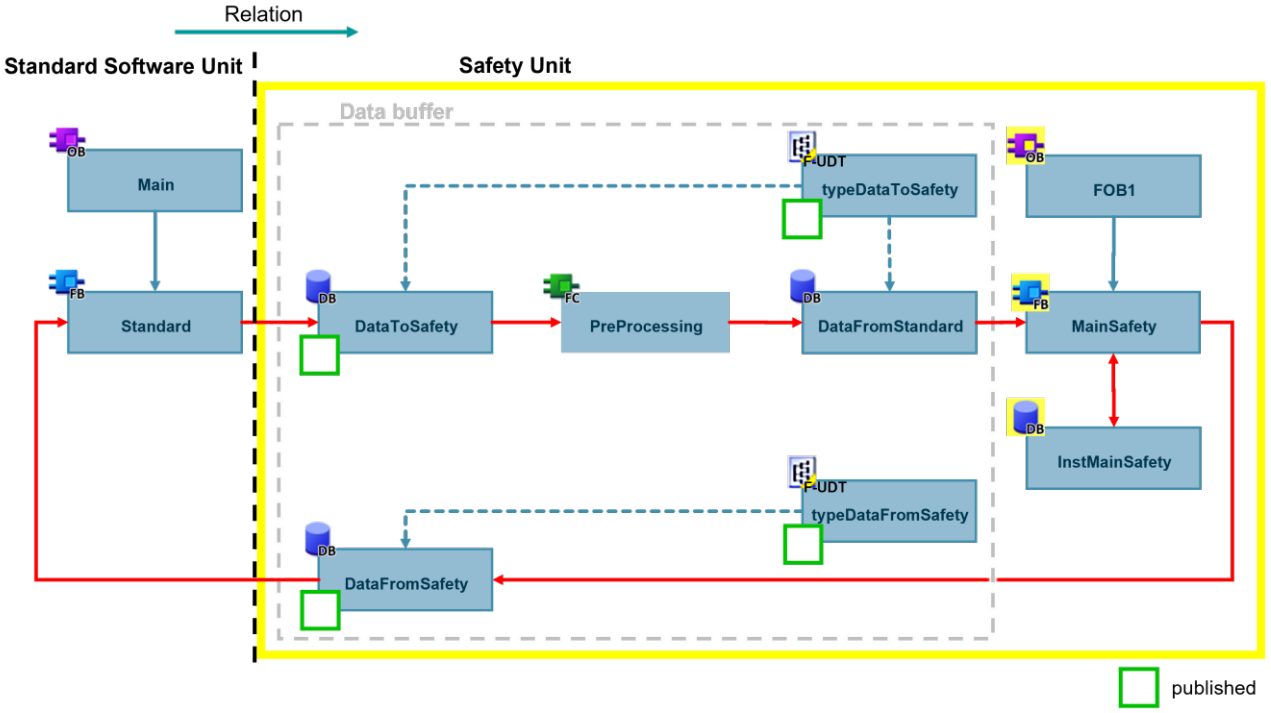


5. Now you can write data that you need in the standard program to the "DataToSafety" and read it from the "DataFromStandard" in the safety program. You can also read data from the safety program from the "DataFromSafety".

3.7.3 Data Exchange between units

The data exchange between units works similarly to the exchange between standard and safety without units.

Figure 3-7 Process overview for data exchange between Standard and Safety between units



Process

1. Create two F-UDTs ("typeDataToSafety" and "typeDataFromSafety") and then create three standard data blocks of the type of the corresponding F-UDTs, in the Safety Unit ("DataToSafety", "DataFromStandard" and "DataFromSafety").
2. Publish the F-UDTs and the data modules in the Safety Unit.
3. Create a relation of the Standard Software Units to the Safety Unit. Publish the necessary data modules and F-UDTs, these are marked with a green box in Figure 3-7 as well as in the TIA Portal.

3.7.4 Transferring HMI signals to the safety program

The data transfer from the HMI to the safety program is to be realized via the standard program. Copy the data from the HMI to the coupling DB in the standard program (Figure 3-6).

Safe signal transmission

Communication between the HMI and the CPU is not safe. Transferring safety-related data requires measures that ensure the safe transfer. This application example shows a suitable safety concept.

Fail-safe transmission of safety-related parameters via web server / HMI:

https://support.industry.siemens.com/cs/ww/en/view/109780314

Resetting safety functions

For resetting safety functions or acknowledging errors using an HMI, TIA Portal provides the "ACK_OP" system block.

An acknowledgment consists of two steps:

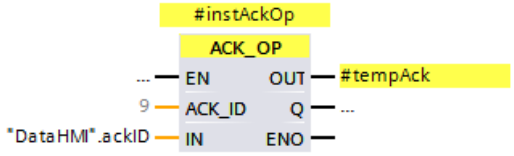
1. Change of IN input/output to the value "6" for exactly one cycle.
2. Change of IN input/output to the value at the "ACK_ID" input within one minute for exactly one cycle.

This system block is an exception to the recommended data exchange.

In each cycle, the system block resets the "IN" InOut parameter to "0". If the data from the HMI is copied in the standard user program, "0" is overwritten with the value from the HMI in each cycle and the condition that the values are present for exactly one cycle is not met.

Therefore, write the tag at the "IN" input directly from the HMI and set the safety program priority higher than that of communication to avoid potential data corruption.

Figure 3-8: System block "ACK_OP"



3.7.5 Using non-safe inputs in the safety program

Recommendation

Standard inputs that are required directly in the safety program should be read directly in the safety program. A "detour" via the standard user program should be avoided.

The background to this is that non-safety-related signals are also included in the application's systematic integrity. Typical examples are the acknowledgment/reset button and operating mode selector switch. Which button switch is allowed to reset which safety function is a direct result of the risk assessment. A change of the command devices must therefore influence the signature and must be made only accompanied by a reassessment and an acceptance test for changes. Furthermore, this is the only way to detect possible data corruption in the standard signal.

NOTICE	The assessment of the specific signals that influence an application's systematic integrity and, depending on this, are evaluated in the standard user program or in the safety program depends on the risk assessment of an application.

Recommendation

Under certain conditions, contrary to the previous recommendation, it may make sense to read standard inputs required in the safety program in the standard program and transfer them to the safety program via a standard data block (as described in chapter 3.7.1). The aim is to achieve a higher level of independence between hardware and software. This is required in particular for standard machines and modular machine concepts.

Advantages

- Better modularization and reusability
- Decoupling of hardware and software

WARNING

In general, because all tags from the standard program are not protected, only fail-safe data or fail-safe signals of F-I/O and of other safety programs (in other F-CPUs) are permitted to be processed in the safety program.

Due to the decoupling of hardware and software, interconnection errors cannot be detected by changes to the signature.

In addition, the information in the respective manuals applies.

Note	Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 8.2 - Data transfer from standard user program to safety program

3.8 F-signatures

The F-signatures are used for unambiguous identification of the safety-oriented program information. They are displayed in the Safety Administration Editor (SAE) and are part of the safety printout. When the signature is changed, the safety program has to be validated again and accepted.

The following signatures are provided in the safety part:

Table 3-1: Overview of F-signatures

Designation	Meaning
Collective F-signature	Identifies a unique state of the fail-safe project data
Collective F-HW signature	In case of changes to fail-safe HW configuration
Collective F-SW signature	In case of changes in the safety program
F-communication address signature	When changes are made to the name or F-communication UUID of communication connections with Flexible F-Link

Recommendation

In contrast to the F-change history, you can reliably detect changes in the safety program through the F signatures. Use the signatures to document block states safely.

Advantages

- Unambiguous identification of fail-safe hardware, software and communication
- More reliable tracking of possible changes and clear documentation
- After a hardware replacement, you can easily prove that the software is unchanged via the collective F-SW signature

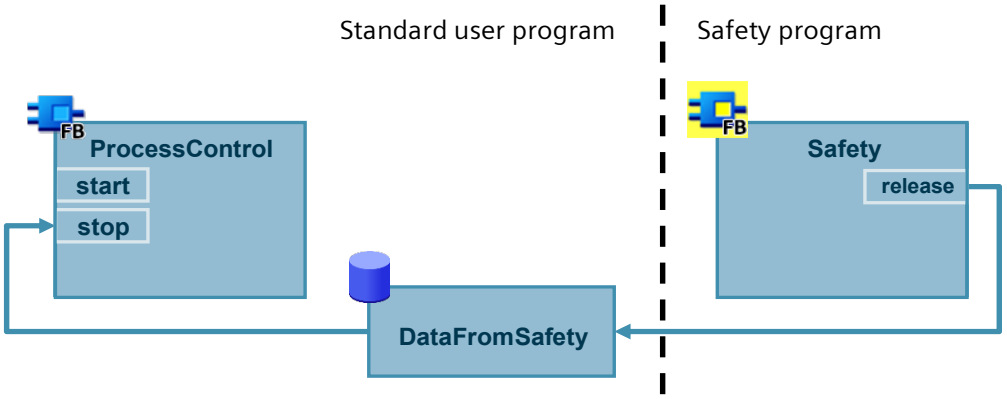
3.9 Resetting operational switching

Safe actuators are often also used for operational switching. The relevant safety standards require that a reset of the safety function does not trigger a restart of the machine. When the safety function is triggered, the operational switching must therefore be reset and a new switch-on signal must be required.

Recommendation

- Interlock the process control in the standard user program with the enable signal from the safety program. As a result of this, a safe shutdown also resets the process control.
- Transfer the enable signal from the safety program using a global data block (see also chapter 3.7).

Figure 3-9: Locking process control with the release signal



3.10 Reintegrating fail-safe I/O modules/channels

If the F-CPU detects an error relevant to safety, it passivates the relevant fail-safe channel or the entire module. Once the error has been corrected, the passivated channel must be reintegrated (depassivated).

As long as a channel is passivated, it uses substitute values. An input provides the process image with the substitute value "0". The substitute value "0" is assigned to an output, regardless of whether or not the program controls the output.

Note

Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 6.5 - Passivation and reintegration of F-I/O

3.10.1 Evaluating passivated modules/channels

General

Whether a channel is passivated can be evaluated as follows:

- The channel's value status is "false"
- The "QBAD" tag of the module's F-I/O data block is "true"
- LEDs of the channel and module light up red
- Entry in the diagnostics buffer

Reintegration can be either manual or automatic. Define the acknowledgment behavior depending on the risk assessment.

Once an error has been corrected, 'ready for acknowledgment' is indicated as follows:

- The "ACK_REQ" tag of the module's F-I/O data block is "true"
- LEDs of channel and module flash alternately between red and green

Globally evaluating the status of F-I/Os / F-channels

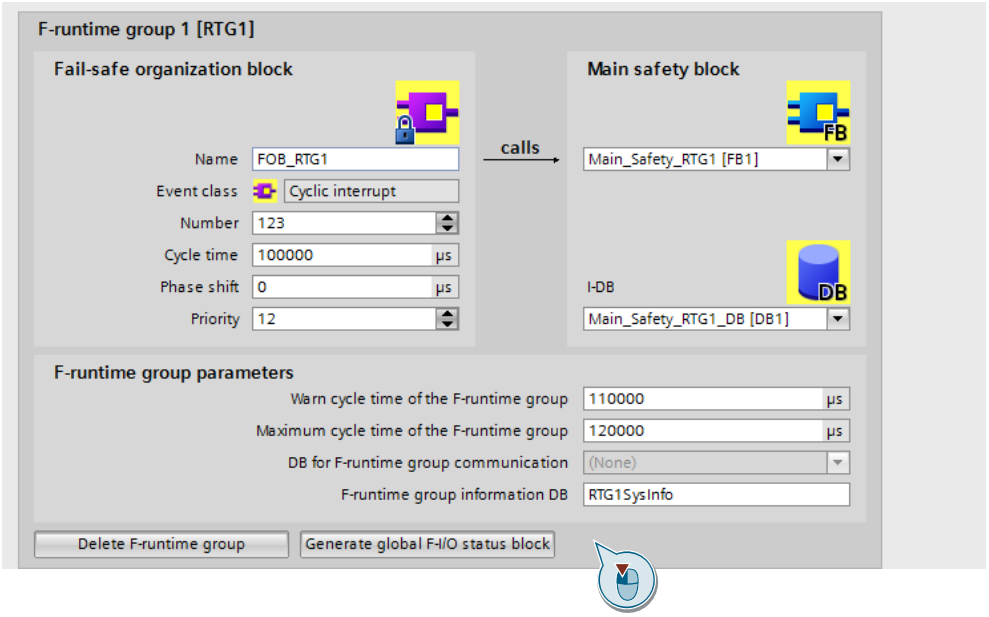
STEP 7 V14 SP1 or higher allows you to have a block generated by the system to globally evaluate the status of all F-I/Os / F-channels of an F-runtime group.

This block evaluates whether substitute values instead of the process values are output for at least one F-I/O or at least one channel of an F-I/O of an F-runtime group. The "QSTATUS" output shows the result of the evaluation. This process does not consider F-I/Os you disabled using the DISABLE tag in the F-I/O DB.

Figure 3-10: System-generated block for global evaluation of F-I/Os

Generate the block in Safety Administration in the settings of the appropriate F-runtime group.

Figure 3-11: Generating the block for global evaluation of F-I/Os



Note

Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 3.3.1 – "F-runtime group" area

3.10.2 Automatic reintegration

Depending on whether the respective module supports the "RIOforFA" standard (see chapter 5), you can implement automatic reintegration in different ways.

WARNING

Automatic reintegration can lead to dangerous situations

Whether automatic reintegration is permissible for a certain process with respect to safety depends on the risk assessment.

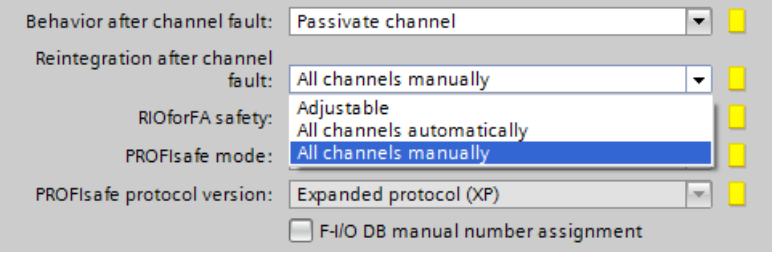
Note

Automatic reintegration applies to F-I/O / channel faults (e.g., discrepancy faults, short-circuits). Communication faults require manual reintegration (see chapter 3.10.3).

Modules that support "RIOforFA"

For modules that support "RIOforFA", you can parameterize automatic reintegration either for the entire module or for single channels.

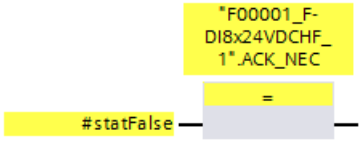
Figure 3-12: Parameterizing automatic reintegration



Modules that do not support "RIOforFA"

For modules that do not support "RIOforFA", program automatic reintegration in the safety program. To do this, set the "ACK_REQ" tag of the respective F-I/O data block to "false":

Figure 3-13: Programming automatic reintegration

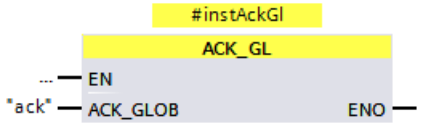


3.10.3 Manual reintegration

Global reintegration of all passivated F-modules

To reintegrate all passivated F-modules / F-channels of an F-runtime group, use the "ACK_GL" instruction:

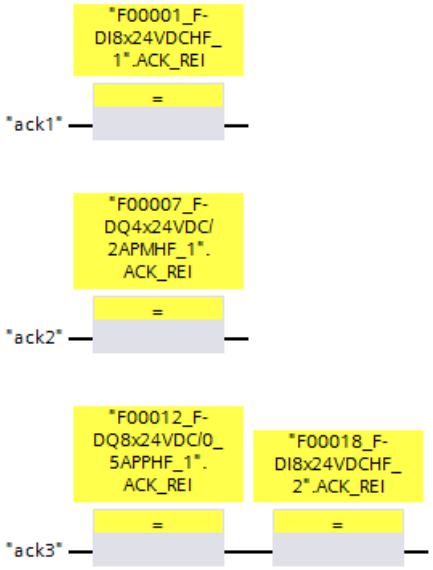
Figure 3-14: "ACK_GL" instruction



Separate reintegration of modules (or of a group of modules)

In distributed plants, it may be required that only local reintegration is allowed (e.g., separate command devices on the control cabinet). To do this, interconnect the "ACK_REI" tags of the respective F-I/O data blocks:

Figure 3-15: Separate reintegration of modules



4 Optimizing Safety Programs

4.1 Optimizing the compilation duration and runtime

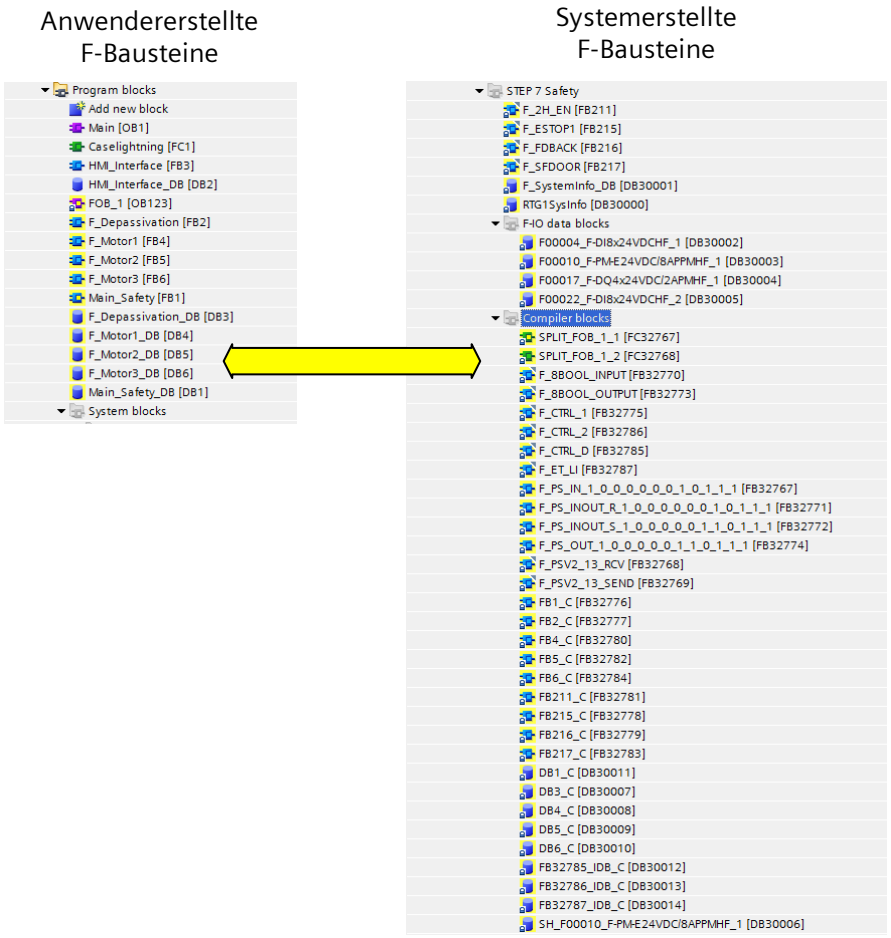
Introduction

User programming protection by means of Coded Processing is an important part of a safety program (see chapter 5). The objective is to detect any data corruption in the safety program and thus prevent non-safe states.

This protection program is generated during the compilation, which extends the compilation duration. The protection program also extends the F-CPU's runtime as the F-CPU additionally processes this program and compares the results with the user program.

You will find the protection program that is automatically generated by the system in the system block folder of your F-CPU.

Figure 4-1: Protection program



Some of the instructions that can be used in the safety program influence a fail-safe controller's performance to a greater extent than others.

This chapter shows different options for reducing the compilation and program runtime.

Note

Depending on the application, it is not always possible to use all the suggestions. However, they show why certain programming methods cause shorter compilation and program runtimes than a non-optimized program.

Determining runtime

TIA Portal automatically creates a data block, "RTGxSysInfo", for each F-runtime group. Among other things, this block contains the current and the longest runtime of this F-runtime group.

You will find this system-generated block in the project tree ("Program blocks > System blocks > STEP 7 Safety").

Figure 4-2: System-generated DB "RTGxSysInfo"

RTG1SysInfo

	Name	Data type	Start value	Monitor value
1	Input			
2	Output			
3	MODE	Bool	false	FALSE
4	F_SYSINFO	F_SYSINFO		
5	MODE	Bool	false	FALSE
6	TCYC_CURR	DInt	0	100
7	TCYC_LONG	DInt	0	101
8	TRTG_CURR	DInt	0	0
9	TRTG_LONG	DInt	0	2
10	T1RTG_CURR	DInt	0	0
11	T1RTG_LONG	DInt	0	0
12	F_PROG_SIG	DWord	DW#16#103E2...	16#103E_261A
13	F_PROG_DAT	DTL	DTL#2017-9-1...	DTL#2017-09-19-1...
14	F_RTG_SIG	DWord	DW#16#8A587...	16#8A58_7EBD
15	F_RTG_DAT	DTL	DTL#2017-9-1...	DTL#2017-09-19-1...
16	VERS_S7SAF	DWord	DW#16#14000...	16#1400_0100
17	InOut			
18	Static			

4.1.1 Jumps in the safety program

In a standard user program, a jump from one network to another (jump to label) or from the block (return) is a simple program branch that is recalculated for each cycle but not additionally protected. This means there is no check as to whether a jump takes place despite the "false" condition, for example due to a memory error caused by EMC.

This is not allowed in a fail-safe program as it must be ensured at all times that the program is in the correct branch.

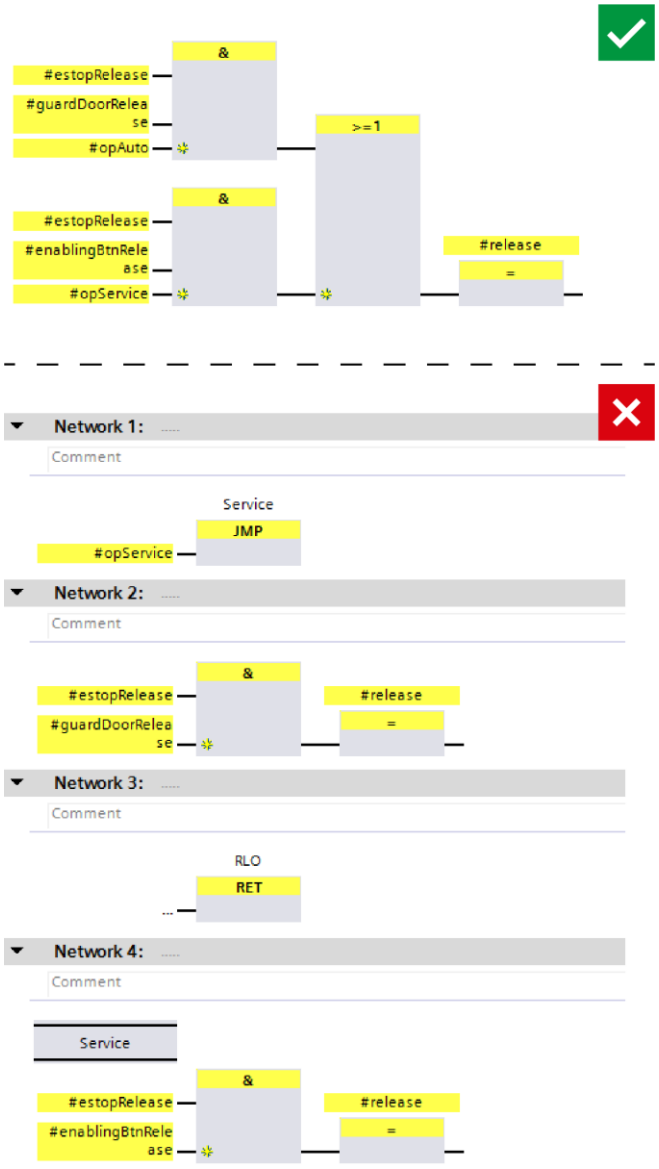
This requires that both alternatives (jump to label is "true" or "false") be calculated in their entirety in the protection program.

The more jumps you use in a safety program, the greater the influence on the controller's performance.

Recommendation

- Avoid jumps in the safety program.
- Use state machines instead of jumps in FBs with binary logic.

Figure 4-3: Avoiding jumps



4.1.2 Timer blocks

Timers are an integral part of a safety program as many of the system functions such as "ESTOP1" internally use these timers. Despite this fact, generating a fail-safe time value requires considerable effort and regeneration for each individual timer block.

Note Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 5.2 - Defining F-runtime groups

Recommendation

Reduce the number of timer blocks to a minimum.

The following blocks use a timer:

- EV1oo2DI
- TWO_H_EN
- ACK_OP
- ESTOP1
- FDBACK
- MUT_P
- TOF
- TON
- TP

4.1.3 Multi-instances

Recommendation

Use multi-instances for fail-safe function blocks. This means that the block-internal tags are integrated into the block interface of the calling block.

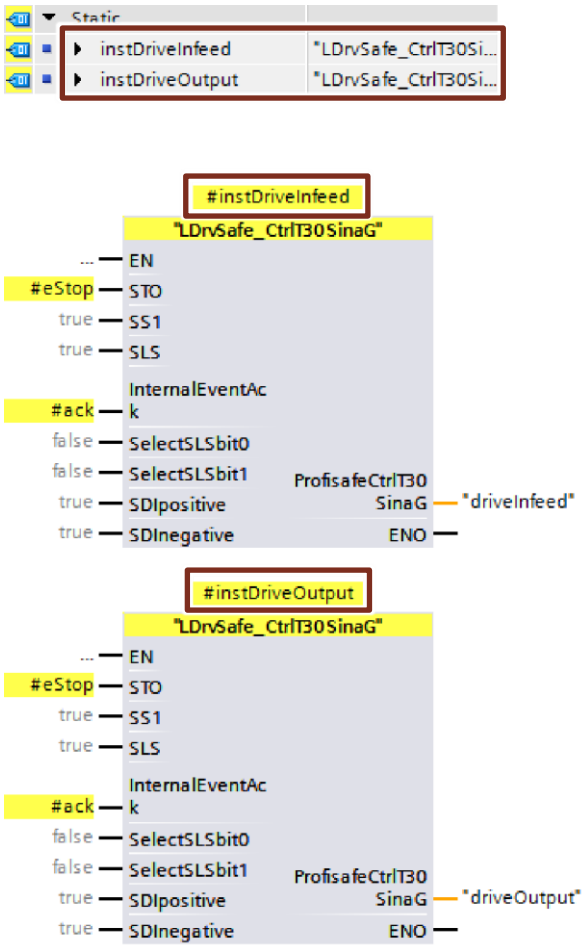
Advantages

- Standardization of safety programs:
  No global data is used for block tags. This allows reuse of the calling block (including the integrated blocks).

Example

Two drives are safely controlled with the same "LDrvSafe_CtrlT30SinaS" function block. The data is stored in multi-instances with unique names.

Figure 4-4: Multi-instances



The "LDrvSafe" library for controlling the safety functions of SINAMICS drives is available in Industry Online Support:

SIMATIC - Fail-safe library LDrvSafe to control the Safety Integrated Functions of the SINAMICS drive family:

https://support.industry.siemens.com/cs/ww/en/view/109485794

4.1.4 Data Access for Standard Variables in the Safety Program

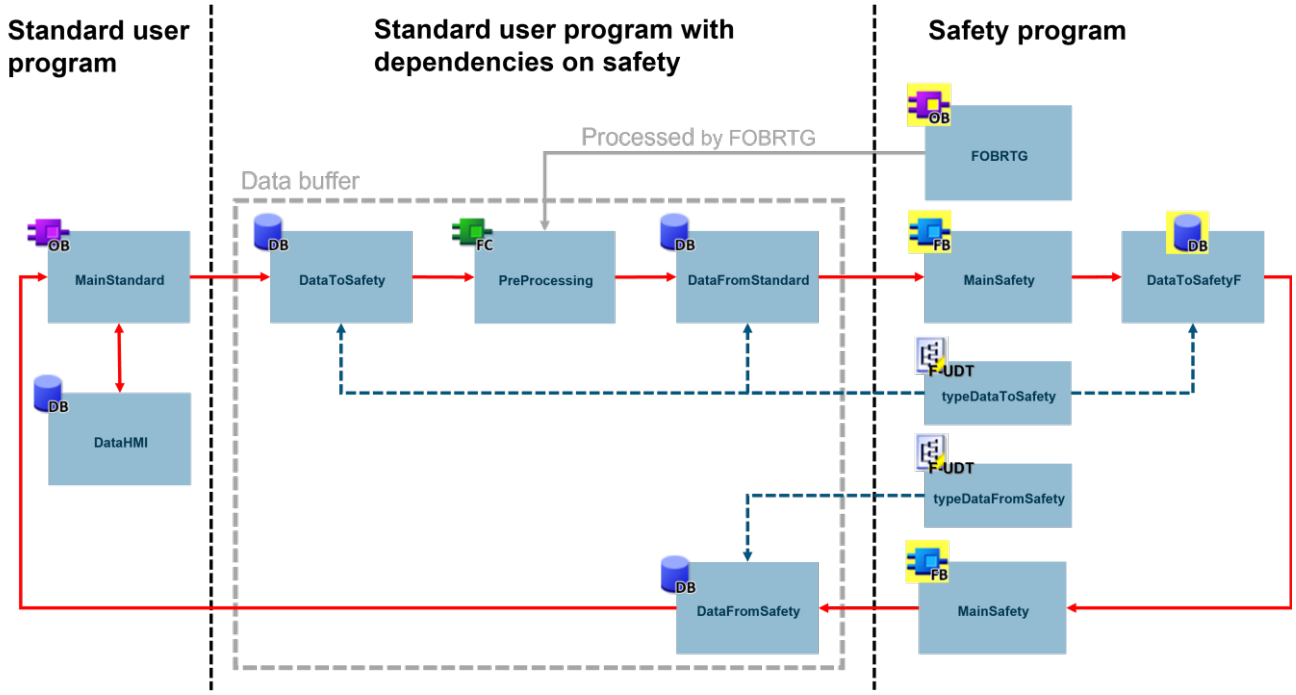
Each time the safety program accesses standard variables, the compiler encodes them separately. This also applies if the same standard variable is accessed multiple times. For performance reasons, it may therefore make sense to copy the standard data once into a fail-safe data area at the beginning of the safety user program and then access these copied variables in the further course.

Note For information on the simplified procedure, see chapter 3.7.

To achieve the desired optimization, use the following resources. Figure 4-5 shows an overview of the process. In the following instructions for changes, the deviations from the simplified procedure are explained.

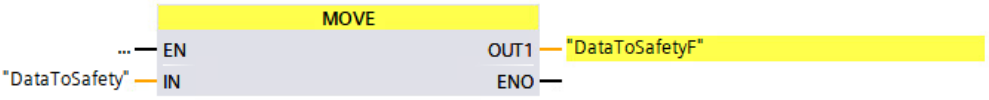
The necessity of using the pre-processing "PreProcessing" remains unaffected by this.

Figure 4-5 Process overview of runtime optimization for the data exchange



Modification instructions

1. Insert an additional, fail-safe data block "DataToSafety**F**" between "DataToSafety" and the processing in the safety program.
2. At the beginning of the safety program, copy the data from "DataToSafety" to "DataToSafetyF". Use a MOVE statement.



3. Then work in the safety program with the fail-safe variables from the data block: "DataToSafetyF".

4.2 Avoiding data corruption

The protection mechanisms within the scope of coded processing (see chapter 5) cyclically analyze the program's execution for data corruption. In case of data corruption, a special system function block triggers an F-STOP of the CPU.

The purpose of this mechanism is to detect influences such as EMC, defective components, etc. and bring the system to a safe state before the machine becomes a risk for humans and the environment.

Aside from external influences, data corruption can also be caused by incorrect programming. The most frequent cause of data corruption is that the standard user program or an external device (e.g., HMI) writes data while the safety program reads that data.

This can occur in the following situations:

- Write access by higher-priority alarms
- Write access by HMI/communication
- Using clock memories

Update of a partial PPI (process image input) by higher-priority alarms

For information about how to correctly program access from the standard user program to the safety program, see chapter 3.7.

Arithmetic functions can cause an overflow or an underflow of the used data type. You then must use a suitable substitute value to finish your calculation. The error-free calculation is displayed at the output ENO for the following functions:

- ADD
- SUB
- MUL
- DIV
- NEG
- ABS
- DWORD_TO_WORD

OPC UA

Deactivate the option "Writable from HMI/OPC UA" for all fail-safe tags in all organization blocks, function blocks, data blocks and functions to prevent data corruption.

Checklist

The following checklist allows you to identify and correct user-generated STOP causes.

Table 4-1: Checklist

Possible causes	Checked
Overflow Underflow or overflow can occur in mathematical functions. This must be intercepted by the user in the program. Therefore, interconnect the ENO output of the mathematical functions	
Division by 0 If a division by 0 occurs in the safety program, the F-CPU goes to STOP. Therefore, interconnect the ENO output of the mathematical functions	
Access via HMI An HMI is used to write (modify) data (bit memories, DBs) that is read in the safety program. As communication has a higher priority than safety by default, this can result in data corruption. Possible solutions can be found in chapter 3.7.	
Standard access to F-data The standard user program modifies data of fail-safe tags or parts of their protection. Write access to F-data is only allowed in the safety program.	
Pointer access to F-data Like standard access, access can occur in runtime when there are unfavorable defaults for generating a pointer to F-areas (inputs, outputs, data blocks, etc.).	

Additional information

For additional information and causes of data corruption, visit Siemens Industry Online Support:

How do you proceed if the F-CPU goes into STOP and the message "Data corruption in the safety program ..." is displayed in the diagnostics buffer?

https://support.industry.siemens.com/cs/ww/en/view/19183712

5 Glossary

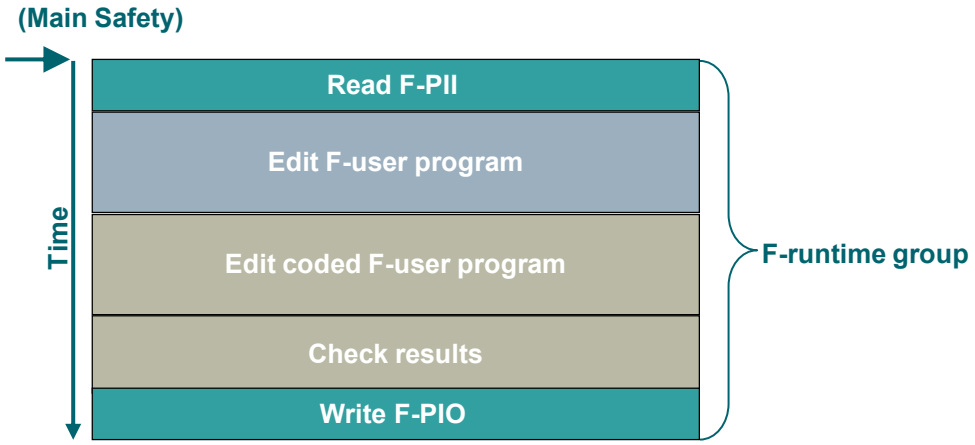
Coded Processing

To meet the normative requirements in terms of redundancy and diversity, all SIMATIC F-CPUs use the "coded processing" principle. In coded processing, the safety program is processed twice by a single processor.

To this end, the compiler generates during a diverse (encoded) safety program during compilation, which is referred to as the protection program.

The first program run processes the unmodified safety program of the user. After that, the protection program is processed. The F-CPU then compares the results. If processed correctly, the safe outputs are written. If the test fails, (e.g., due to data corruption), the F-CPU goes to stop and generates an entry in the diagnostics buffer.

Figure 5-1 Safety program processing sequence



Data corruption

Data corruption means that data of the safety program has been corrupted by external influences (e.g., EMC influences) or illegal write access.

F-CPU

An F-CPU is a controller suitable for safety-related tasks.

PROFIsafe

PROFIsafe is a protocol for fail-safe communication via PROFINET or PROFIBUS.

Cross-circuit

Cross-circuit detection is a diagnostic function of an evaluation unit that detects short-circuits or cross-circuits between two input channels (sensor circuits).

A cross-circuit can be caused, for example, by a sheathed cable being pinched. In devices without cross-circuit detection, this can mean that a two-channel emergency stop circuit does not trip even though only one NC contact is faulty (secondary error).

RIOforFA

RIOforFA (Remote IO for Factory Automation) is a standard from the PROFIBUS & PROFINET International Organization and describes the following functions, among others:

- Synchronous provision of channel-specific diagnostics of remote IOs for high performance
- Channel-specific passivation and reintegration of PROFIsafe remote IOs

Feedback circuit

A feedback circuit is used for monitoring controlled actuators (e.g., relays or load contactors) with positive-action contacts or mirror contacts. The outputs can only be activated when the feedback circuit is closed. When using a redundant shutdown path, the feedback circuit of both actuators must be evaluated. These may also be connected in series.

Reset function/resetting

When a safety function has been triggered, the system must remain in stop until it returns to a safe state for restarting.

Restoring the safety function and clearing the stop command is referred to as the reset function / resetting.

In this context, "acknowledging the safety function" is another frequently used phrase.

Safety program

Part of the safety program that processes safety-related tasks.

STEP 7 Safety Basic/Advanced

STEP 7 Safety Basic and Advanced are STEP 7 option packages that allow you to configure F-CPUs and create a safety program.

- STEP 7 Safety Basic allows you to configure the fail-safe SIMATIC S7-1200 controllers.
- STEP 7 Safety Advanced allows you to configure all fail-safe SIMATIC controllers.

6 Appendix

6.1 Service and support

SiePortal

The integrated platform for product selection, purchasing and support - and connection of Industry Mall and Online support. The SiePortal home page replaces the previous home pages of the Industry Mall and the Online Support Portal (SIOS) and combines them.

- Products & Services
  In Products & Services, you can find all our offerings as previously available in Mall Catalog.
- Support
  In Support, you can find all information helpful for resolving technical issues with our products.
- mySieportal
  mySiePortal collects all your personal data and processes, from your account to current orders, service requests and more. You can only see the full range of functions here after you have logged in.

You can access SiePortal via this address: sieportal.siemens.com

Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers – ranging from basic support to individual support contracts.

Please send queries to Technical Support via Web form:
support.industry.siemens.com/cs/my/src

SITRAIN – Digital Industry Academy

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page:

siemens.com/sitrain

Industry Online Support app

You will receive optimum support wherever you are with the "Industry Online Support" app. The app is available for iOS and Android:

6.2 Links and literature

Table 6-1: Links and literature

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to the entry page of the application example https://support.industry.siemens.com/cs/ww/en/view/109750255
\3\	Programming Guideline for SIMATIC S7-1200/1500 https://support.industry.siemens.com/cs/ww/en/view/90885040
\4\	Programming Style Guide for SIMATIC S7-1200/1500 https://support.industry.siemens.com/cs/ww/en/view/109478084
\5\	SIMATIC Industrial Software SIMATIC Safety - Configuring and Programming https://support.industry.siemens.com/cs/ww/en/view/54110126
\6\	Topic page "Safety Integrated - Safety technology in factory automation" https://support.industry.siemens.com/cs/ww/en/view/109747812

6.3 Change documentation

Table 6-2: Change documentation

Version	Date	Change
V1.0.1	10/2017	First edition
V1.1.0	09/2020	Adaptations and corrections
V1.2.0	09/2021	Adaptations and corrections
V1.3.0	03/2023	Addition of Safety Unit and adaptations
V1.4.0	06/2023	Revision data exchange standard – safety
V1.5.0	07/2023	Simplification of Data Exchange
V1.6.0	07/2024	Extension of Data exchange between units Correction of wording security/ safety in EN version