SIEMENS
Ingenuity for life

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

Table of contents

Legal information **2**

1 Introduction **4**

Warning notice system 5

2 Configuring Fail-Safe Controllers **6**

2.1 Selecting the suitable F-CPU 6
2.2 F-change history 8
Advantages 8
2.3 Consistently uploading F-CPUs 8
2.4 Know-how protection 9

3 Methods for Safety Programming **10**

3.1 Program structures 10
Recommendation 10
3.1.1 Defining the program structure 10
3.1.2 Call levels of F-FBs/F-FCs 12
3.1.3 Call sequence of the blocks in the Main Safety 12
3.1.4 F-compliant PLC data types 13
3.1.5 Block information and comments 14
3.2 Functional identifiers of tags 15
3.3 Standardizing blocks 16
3.4 Programming logic operations 17
3.5 Programming operating mode-dependent safety functions 17
3.6 Connection of global data 18
3.7 Data exchange between standard user program and safety program 19
3.7.1 Data exchange with the help of F-UDTs 19
3.7.2 Reading tags from the standard user program which can change during the runtime of an F-runtime group 22
3.7.3 Reading diagnostic and message information from the safety program 22
3.7.4 Transferring operational information to the safety program 23
3.7.5 Using non-safe inputs in the safety program 24
3.7.6 Transferring HMI signals to the safety program 25
3.8 Resetting operational switching 27
3.9 Reintegrating fail-safe I/O modules/channels 28
3.9.1 Evaluating passivated modules/channels 28
3.9.2 Automatic re-integration 30
3.9.3 Manual reintegration 31

4 Optimizing Safety Programs **32**

4.1 Optimizing the compilation duration and runtime 32
4.1.1 Jumps in the safety program 33
4.1.2 Timer blocks 35
4.1.3 Multi-instances 35
4.2 Avoiding data corruption 37

5 Glossary **39**

6 Appendix **41**

6.1 Service and support 41
6.2 Industry Mall 42
6.3 Links and literature 42
6.4 Change documentation 42

1 Introduction

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

If the programming is sloppy, the many options provided by STEP 7 can also produce negative results:

- CPU stop
- Long compilation process
- Additional, comprehensive acceptance testing

This document provides you with many recommendations and notes 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 universally used 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 styleguide

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 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

Selecting the F-CPU depends on the following factors:

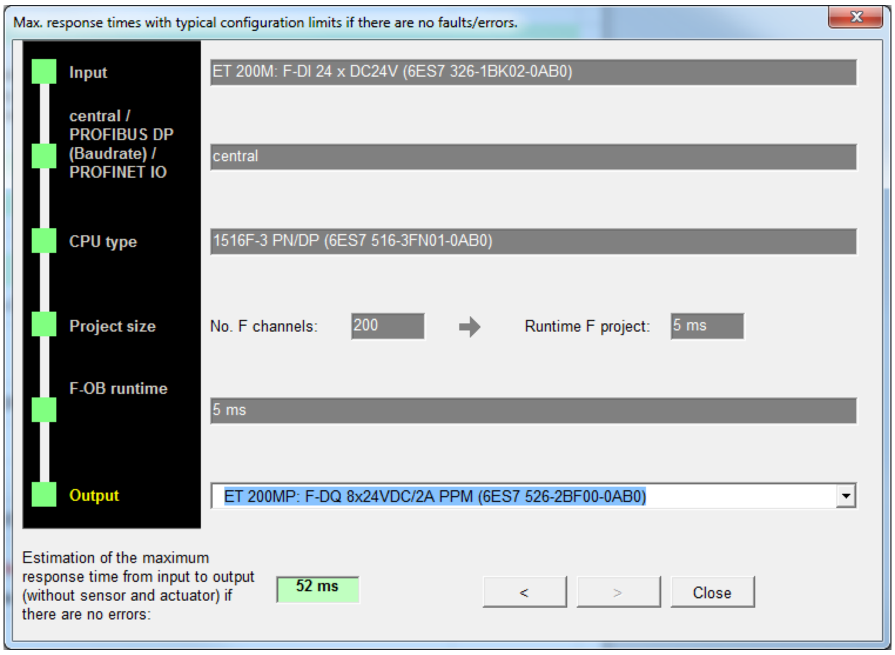
- Runtime of the safety program
- PROFIsafe communication time
- Response time of the safety function
- Number of required inputs and outputs
- Number of connected I/O devices
- Memory requirements of the program

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 Reaction 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: Reaction time wizard of the SIMATIC STEP 7 Reaction Time Table



Max. response times with typical configuration limits if there are no faults/errors.

Input: ET 200M: F-DI 24 x DC24V (6ES7 326-1BK02-0AB0)

central / PROFIBUS DP (Baudrate) / PROFINET IO: central

CPU type: 1516F-3 PN/DP (6ES7 516-3FN01-0AB0)

Project size: No. F channels: 200 → Runtime F project: 5 ms

F-OB runtime: 5 ms

Output: ET 200MP: F-DQ 8x24VDC/2A PPM (6ES7 526-2BF00-0AB0)

Estimation of the maximum response time from input to output (without sensor and actuator) if there are no errors: 52 ms

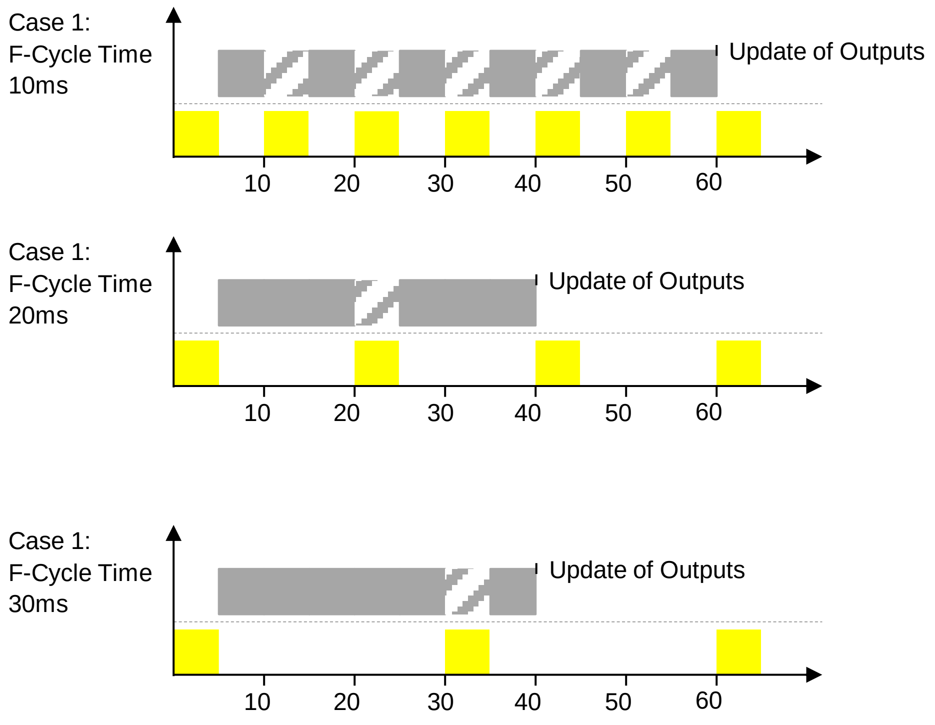
Influence of the safety program's cycle time on the standard user program

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



⬛ Duration of standard user program: 30ms
Priority: ≤11

🟨 Duration of safety program: 5ms
Priority: 12

Note

Please note that higher-priority organization blocks (e.g., cyclic interrupt OBs or motion control OBs) can 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

The cycle time must be longer than the execution duration of the safety program as otherwise the CPU would change to the STOP state.

Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – 5.2

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 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.
- Which user changed or downloaded the safety program can be tracked in multi-user projects.
- Matching 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 11

2.3 Consistently uploading 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, check the "Consistent upload" check box in Safety Administration in TIA Portal.

Advantages

A programmer on the system can load the respective program onto his 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. Additionally,
more load memory is needed 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 across contents of program parts.
- Accepted blocks cannot be modified.

Additional information

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

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

- Divide the program code into modules, e.g.
  - into sub-areas for sensing, evaluating, reacting 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

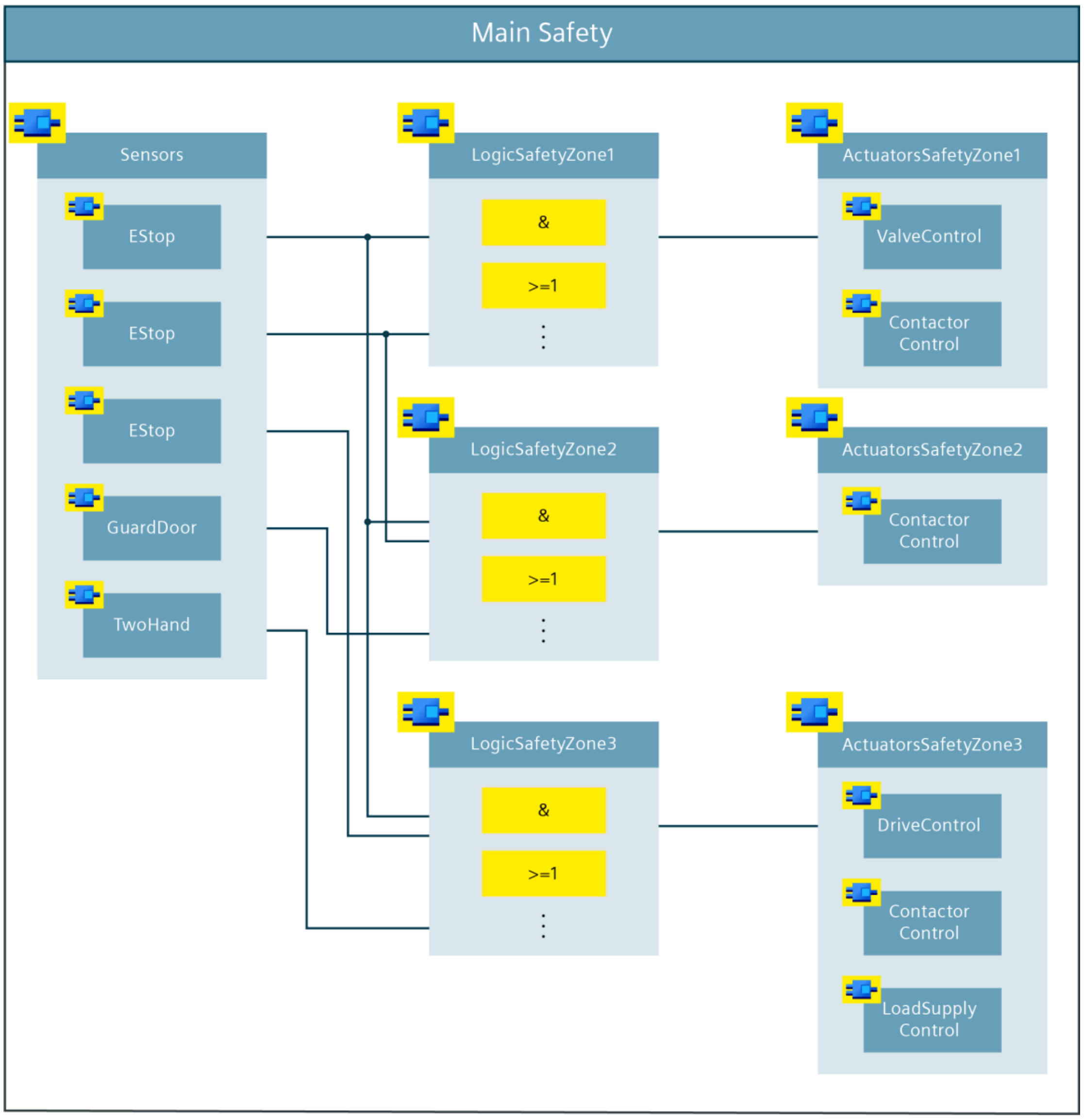
The following figure shows a safety application that is divided into three machine areas (safety zones).

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, for example, also be possible to implement the logic and actuator control in a shared function block.

3.1.2 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 relatively flat so that the safety program remains within the limits specified here.

3.1.3 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-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-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.

3.1.4 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 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 to nest 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 by structuring of the data.

Note

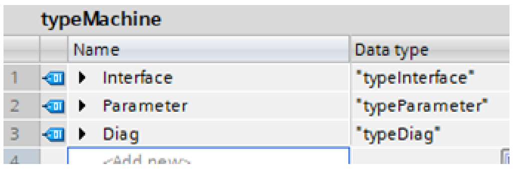
Try to design the F-compliant PLC data types as modular as possible to achieve reusability of the data types as well as of the blocks.

Please also observe the information iin the manual SIMATIC Safety - Configuring and Programming – chapter 5.1.5

Example

The example below shows the use of two F-compliant PLC data types. The F-UDT "typeMachine" (Figure 2-1) contains machine-related data. The data is structured by the use of the other F-UDTs "typeInterface", "typeParameter" as well as "typeDiag" and nesting. Figure 3-3 shows how to access the respective data.

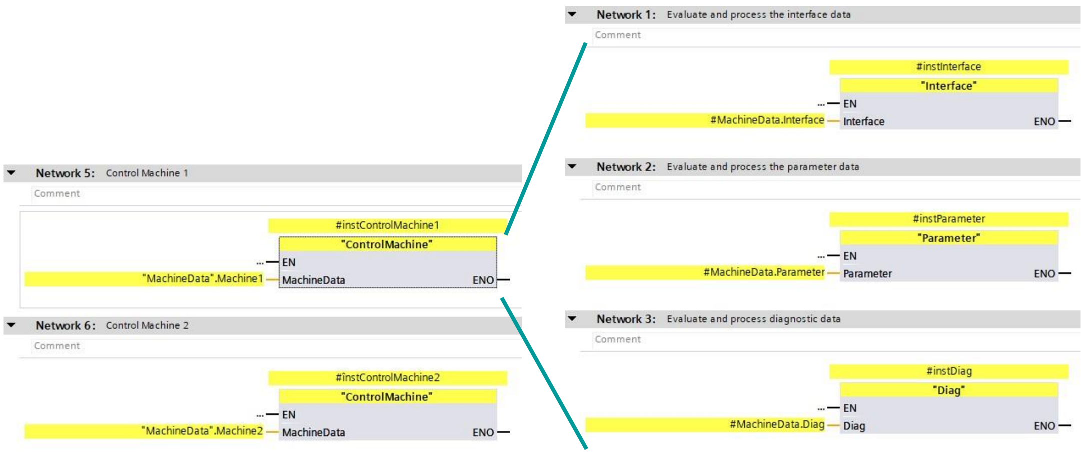
Figure 3-2 Nested F-compliant PLC data type



typeMachine

	Name	Data type
1	Interface	"typeInterface"
2	Parameter	"typeParameter"
3	Diag	"typeDiag"
4	<Add new>	

Figure 3-3 Use of nested F-UDTs



Network 5: Control Machine 1
Comment
#instControlMachine1
"ControlMachine"
... — EN
"MachineData".Machine1 — MachineData ENO —

Network 6: Control Machine 2
Comment
#instControlMachine2
"ControlMachine"
... — EN
"MachineData".Machine2 — MachineData ENO —

Network 1: Evaluate and process the interface data
Comment
#instInterface
"Interface"
... — EN
#MachineData.Interface — Interface ENO —

Network 2: Evaluate and process the parameter data
Comment
#instParameter
"Parameter"
... — EN
#MachineData.Parameter — Parameter ENO —

Network 3: Evaluate and process diagnostic data
Comment
#instDiag
"Diag"
... — EN
#MachineData.Diag — Diag ENO —

3.1.5 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.

If you implement diagnostic functions relevant to the PL / SILCL of another subsystem (Detect or Evaluate) in an F-FB, also enter normative parameters such as PL / 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 source code context.

Choose the tag identifier such that it reflects the logic "1" state ("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.

Siemens Industry Online Support provides block libraries you can use in your project, for example "LDrvSafe":

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

Recommendation

Create modular blocks 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 ADD and OR logic elements
- Avoid jumps to binary logic

3.5 Programming operating mode-dependent safety functions

Recommendation

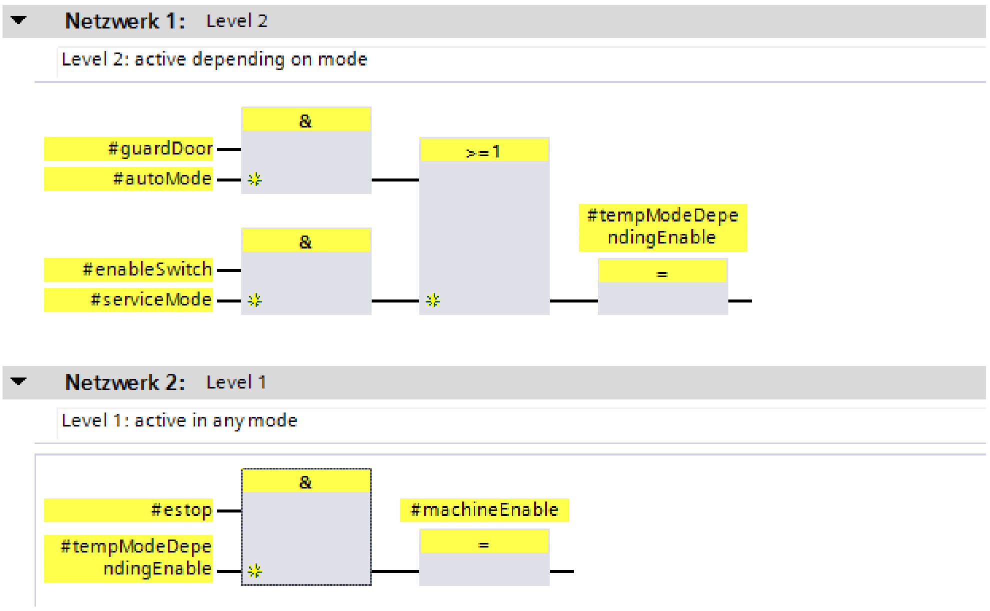
Divide the logic into different levels:

- Level 1: All safety functions that are independent of 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 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 mode.

Figure 3-4: Programming modes



3.6 Connection of global data

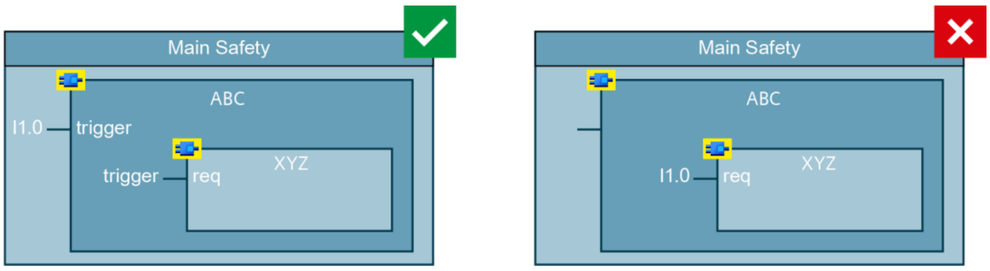
Recommendation

- 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

The safety program's task is to execute all the functions that represent a risk-reducing action. All other operational functions and functions for operation and maintenance are part of the standard user program.

As in practice, information for the diagnostic and signaling concept is also generated in the safety program and operational information is also relevant to the safety program, it is not possible to completely separate both program parts.

In order to move non-safety-related functions to the standard user program, a clearly defined interface is recommended. Global data blocks are best suited for this purpose.

3.7.1 Data exchange with the help of F-UDTs

Recommendation

- Use global standard data blocks to exchange data between the standard user program and the safety program.
- To ensure a good overview of which program part reads and which one writes, it is recommended to create two data blocks for the two directions.
- For better handling, the use of F-compliant PLC data types (F-UDT) is recommended (see chapter 3.1.4). Create an F-UDT for each direction and instantiate them as global data blocks (see Figure 3-6 and Figure 3-8). Through the use of F-UDTs, changing the interface between the standard and safety program is reserved for users with the Safety password. Changes to the interface by the standard user is therefore not possible. Unintentional changes which would necessitate loading of the standard user program via the system state STOP can be prevented by this.
- The compiler carries out separate coding whenever the safety program accesses standard tags. This also applies if there are multiple accesses to the same standard tag. For performance reasons it may therefore make sense to copy the standard data at the beginning of the safety user program once to a fail-safe data area and then to access this copied data (see Figure 3-7).
- Use separate data blocks if there are several runtime groups.

Advantages

- Lean F-runtime group
- Better overview of the exchanged data
- Changes of the diagnostic and signaling concept in the standard user program do not affect the signature of the safety program
- Simplified typification of F-blocks
- Changes to the standard user program can be loaded without stopping the CPU
- Standard user program and safety program can be created independently of each other and changed if interfaces have already been defined
- Protection of the interface

Restrictions

- The use of F-UDTs for data exchange between standard and safety program with arrays is not supported.

Figure 3-6: Data exchange between standard user program and safety program

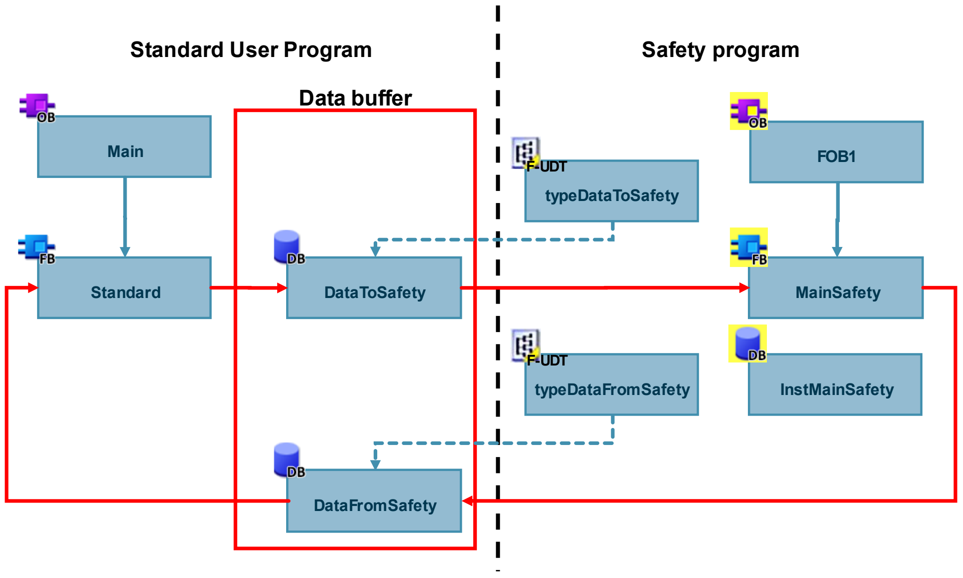
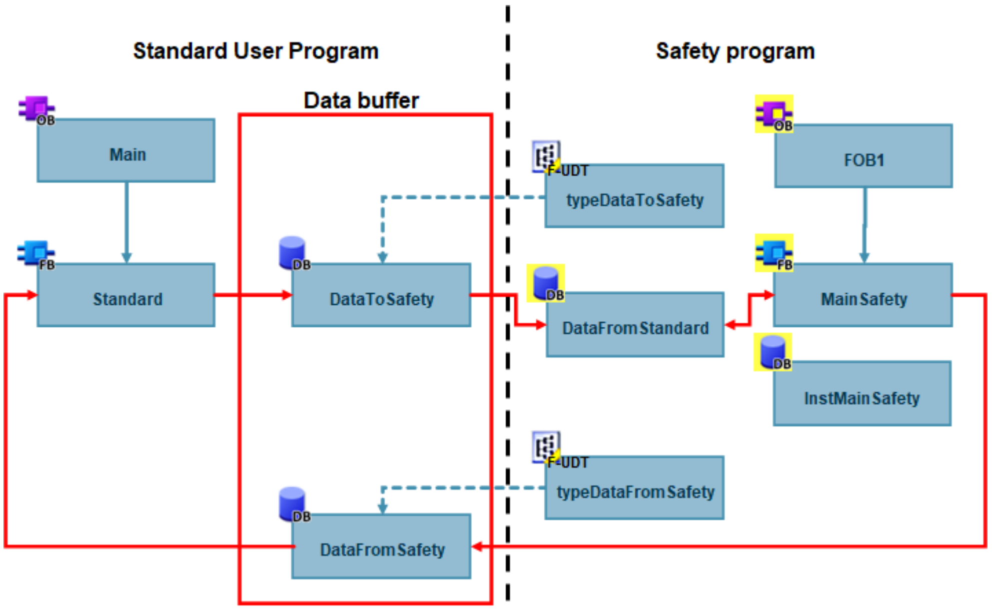


Figure 3-7 Data exchange with intermediate buffer "DataFromStandard"



Note

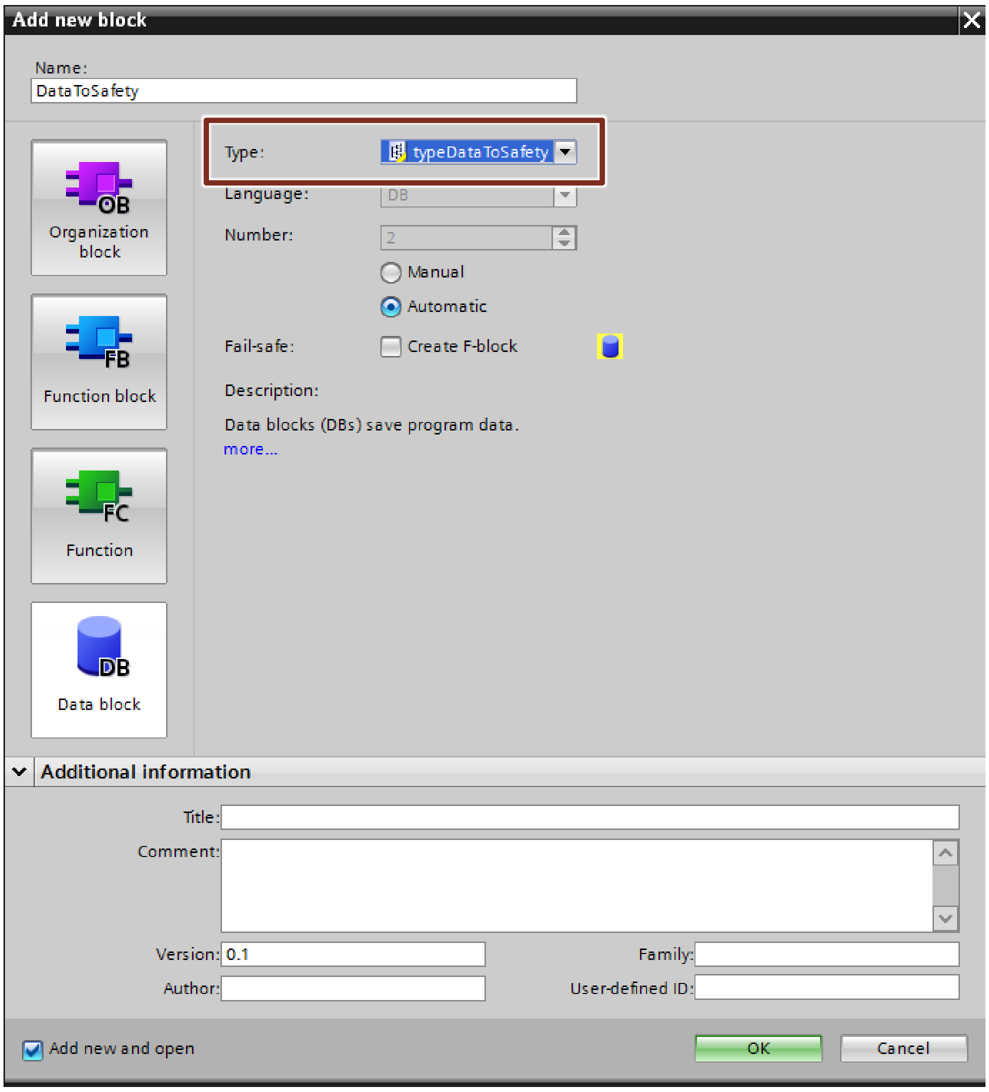
Interoperation of programmers with and without authorization for the safety program

Changes in standard blocks that are read- or write-accessed by the safety program cause the F-program to lose its consistency. An additional compile of the safety program is needed and the next download to the CPU is only possible via system state STOP.

If such standard blocks are changed, you will be requested to enter the F-password as of TIA Portal V16.

With the help of F-compliant PLC data types (F-UDTs) you can protect the interface against changes without a Safety password.

Figure 3-8 Creating a global data block as an instance of an F-compliant PLC data type



Name: DataToSafety

Type: typeDataToSafety

Language: DB

Number: 2

Manual

[✓ Automatic]

Fail-safe: Create F-block

Description:

Data blocks (DBs) save program data.

Title:

Comment:

Version: 0.1

Family:

Author:

User-defined ID:

[✓ Add new and open]

3.7.2 Reading tags from the standard user program which can change during the runtime of an F-runtime group

If you want to read tags from the standard user program in the safety program, and these tags can be changed by the standard user program during runtime of the F-runtime group in which they are read, for example, because your standard user program is being processed by a higher-priority cyclic interrupt, you must use own tags of a standard DB for this purpose.

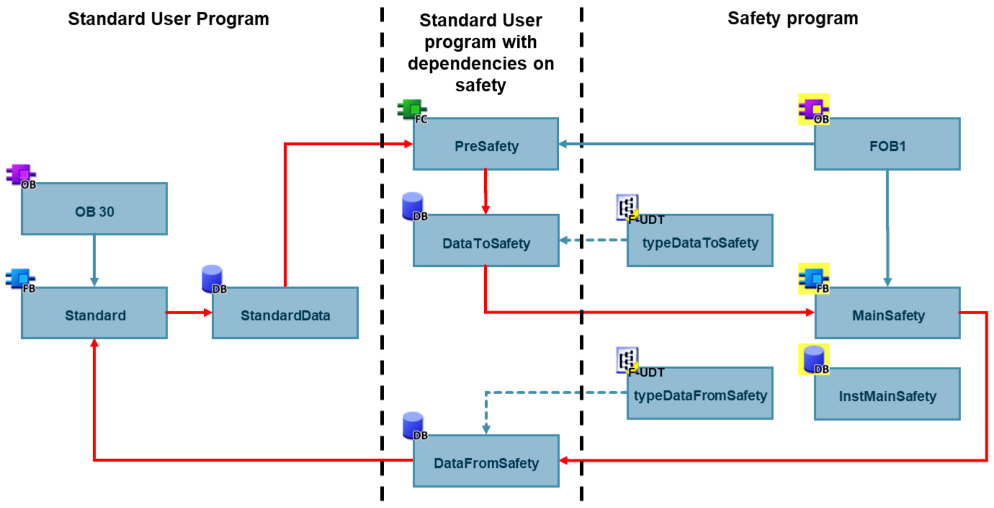
Recommendation

Use a standard DC for preprocessing in the F-runtime group to copy the data from the standard user program to a data area accessed by the safety program.

Advantages

- Data consistency is retained.
- Prevention of data corruption in the safety program and the CPU stop it causes

Figure 3-9 Reading standard data which can change during runtime of the safety program



3.7.3 Reading diagnostic and message information from the safety program

A frequent application for data exchange between the standard user program and the safety program is the visualization of diagnostic and message information such as:

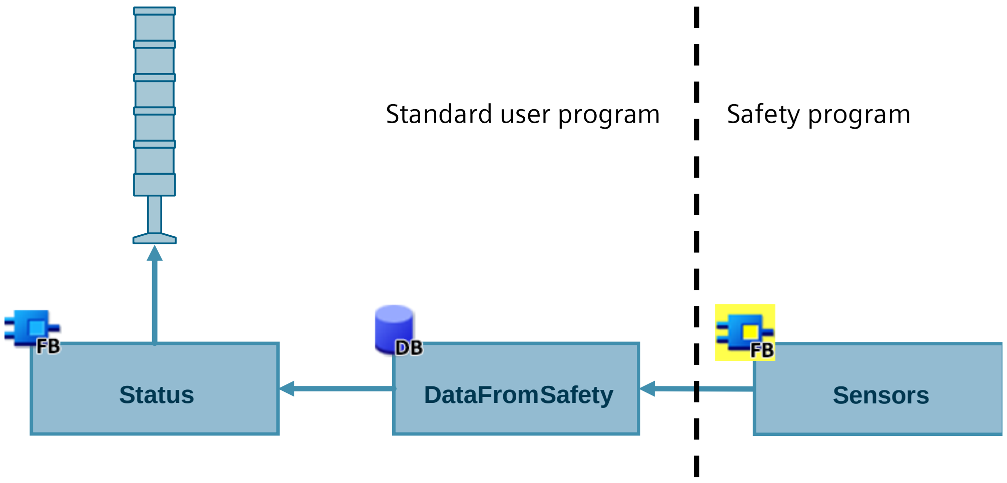
- Acknowledgment requests of errors
- Reset requests of safety functions
- Error messages
- States of safety functions

Transfer the "raw data" from the safety program. The logic operation then takes place in the standard user program. This has the advantage that the safety program is kept lean and is independent of changes in the standard user program.

Small changes at a later stage (e.g., changes to the control of an indicator light) are made in the standard user program. This does not change accepted F-blocks.

If you transfer a large amount of diagnostic data from the safety program, create an F-data type for this purpose. A tag with a self-defined data type keeps the block interface compact and clear. For data always to be transferred in a similar way, it is recommended to standardize these F-compliant PLC data types across all F-function blocks.

Figure 3-10: Reading diagnostic and message information from the safety program



Note

Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 8.1

3.7.4 Transferring operational information to the safety program

In many applications, it is essential that specific non-safety-related results of logic operations are transferred from the standard user program to the safety program. These are typically operational switch-on conditions (e.g., operational and fail-safe switching of a motor starter) or machine states for mode preselection.

Prepare the data in the standard user program to the greatest possible extent. The more non-safety-related logic is implemented in the standard user program, the easier it is to implement changes to the logic relevant to the process.

Note

Please also observe the information in the manual SIMATIC Safety - Configuring and Programming – chapter 8.2

3.7.5 Using non-safe inputs in the safety program

Recommendation

Standard inputs that are required directly in the safety program must 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.

Advantage

- 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

3.7.6 Transferring HMI signals to the safety program

Human-machine interfaces (HMIs) are convenient, essential components in a machine operator's daily work. In order to make use of this convenience for operator control and monitoring of processes and plants even in safety-related applications, additional measures are required.

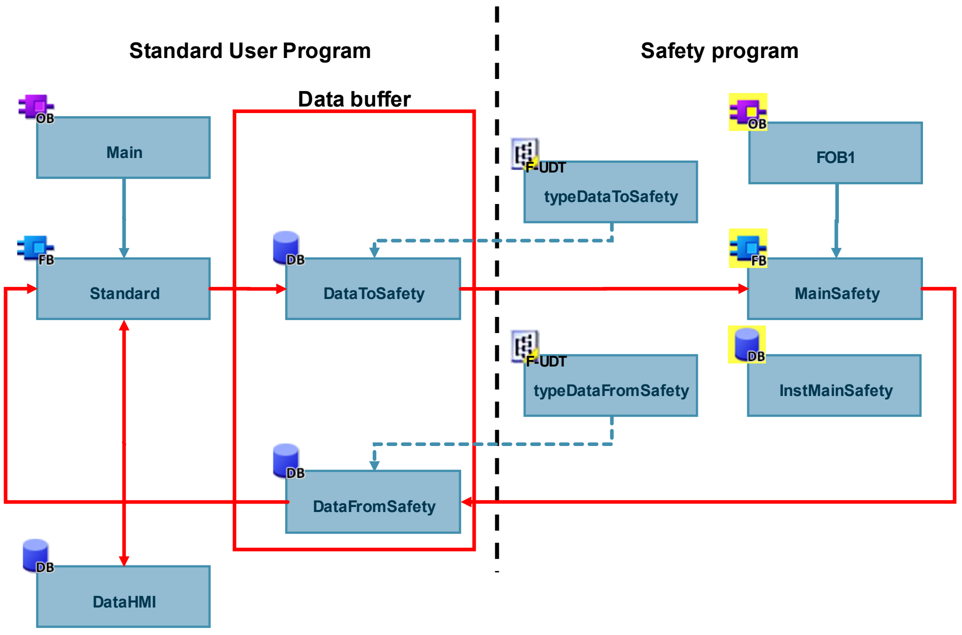
Writing tags from the HMI to the safety program is a problem for the following reasons:

- Signals from the HMI panel are not safety-related and not monitored. An error can result in forbidden changes of safety-related values, which increases the risk.
- Communication between the HMI and the CPU is acyclic. As a result, the HMI's write access may take place while processing the safety program. The first program run then still uses the original value. The encoded user program uses the value that has been updated in the meantime. This causes data corruption in the safety program and therefore a stop of the CPU (see chapter 5).

Recommendation

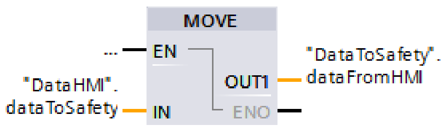
Use another data block for communication with the HMI and copy the safety-related data in the standard user program to the data buffer.

Figure 3-11: Data exchange between HMI and safety program



Create a data type for the data from the HMI to the safety program. Use this data type in the HMI tags, in the data buffer for the safety program and in the standard user program where the data is copied.
To add more tags to be written from the HMI to the safety program, merely modify this data type.

Figure 3-12: Copying data from the HMI to the safety program in the standard user program



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:

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:

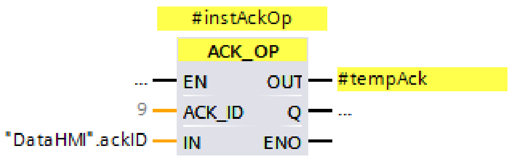
1. Change of IN input/output to the value "6" for exactly one cycle.
2. Change of IN input/output to the value at the ACK_ID input within one minute for exactly one cycle.

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 3-13: System block "ACK_OP"



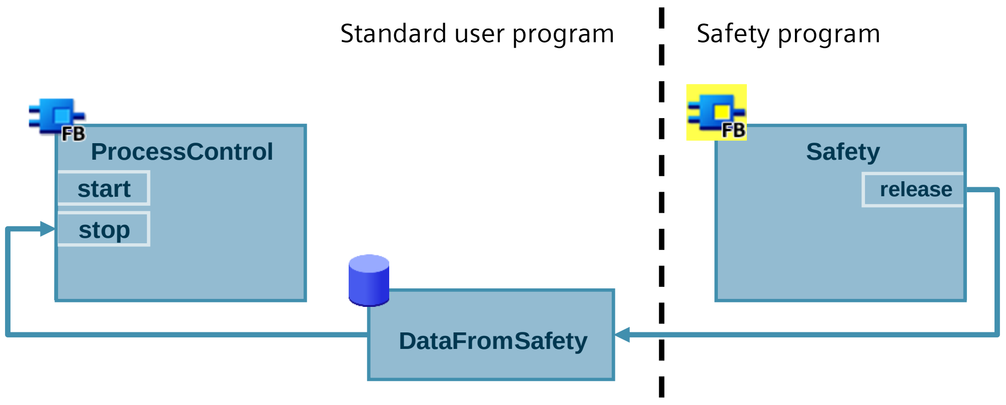
3.8 Resetting operational switching

Safe actuators are often also used for operational switching. The relevant safety standards require that a reset of the safety function does not trigger 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 3-14: Locking process control with the release signal



3.9 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

3.9.1 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 channel and module light up red
- Entry in 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:

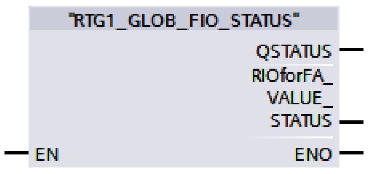
- The "ACK_REQ" tag of the module's F-I/O data block is "true"
- LEDs of channel and module flash alternately between red and green

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.

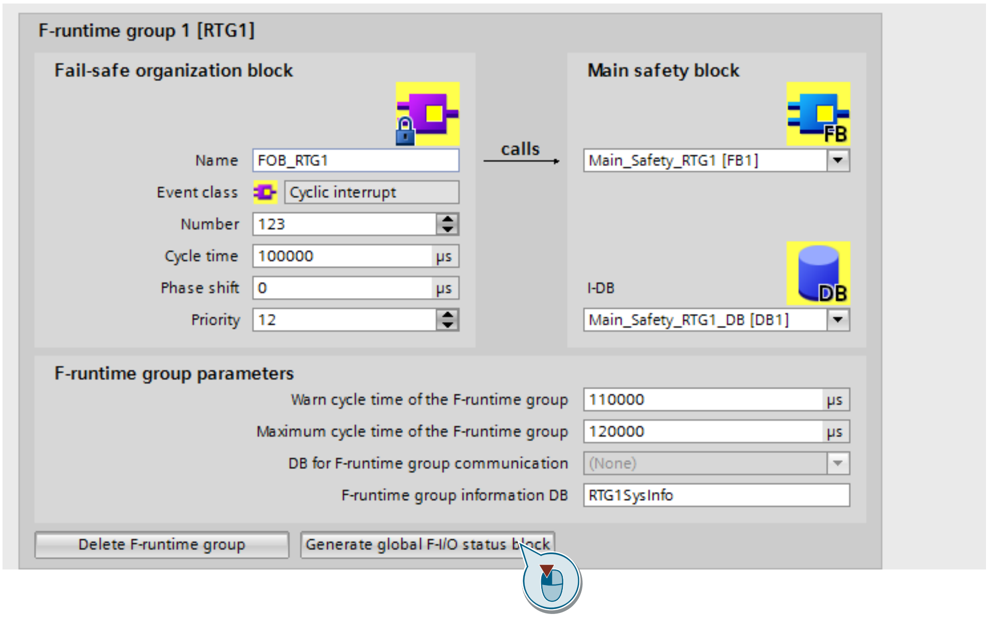
This block evaluates whether instead of the process values, substitute values are output for at least one F-I/O or at least one channel of an F-I/O of an F-runtime group. The "QSTATUS" output shows the result of the evaluation. This process does not consider F-I/Os you disabled using the DISABLE tag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

Figure 3-15: 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 3-16: Generating the block for global evaluation of F-I/Os



F-runtime group 1 [RTG1]

Fail-safe organization block

Name	FOB_RTG1
Event class	Cyclic interrupt
Number	123
Cycle time	100000 µs
Phase shift	0 µs
Priority	12

Main safety block

Main_Safety_RTG1 [FB1]

I-DB

Main_Safety_RTG1_DB [DB1]

F-runtime group parameters

Warn cycle time of the F-runtime group	110000 µs
Maximum cycle time of the F-runtime group	120000 µs
DB for F-runtime group communication	(None)
F-runtime group information DB	RTG1SysInfo

Note

Please also observe the information iin the manual SIMATIC Safety - Configuring and Programming – chapter 3.3.1

3.9.2 Automatic re-integration

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 depends in terms of safety depends on the risk assessment.

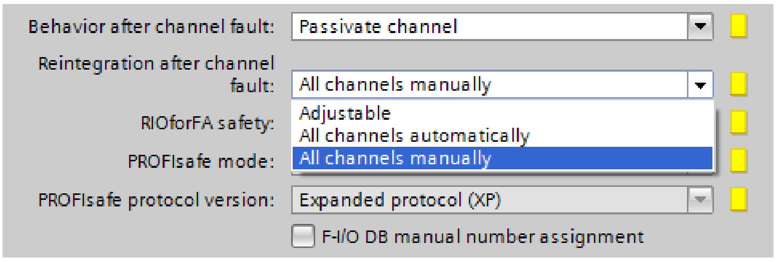
Note

Automatic reintegration concerns F-I/O / channel faults (e.g., discrepancy faults, short-circuits). Communication faults require manual reintegration (see chapter 0).

Modules that support "RIOforFA"

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

Figure 3-17: Parameterizing automatic reintegration

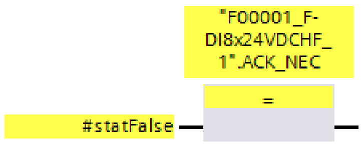


Behavior after channel fault:	Passivate channel
Reintegration after channel fault:	All channels manually
RIOforFA safety:	Adjustable / All channels automatically / All channels manually
PROFIsafe mode:	
PROFIsafe protocol version:	Expanded protocol (XP)
	F-I/O DB manual number assignment

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 3-18: Programming automatic reintegration

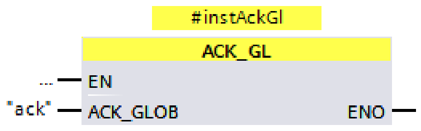


3.9.3 Manual reintegration

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:

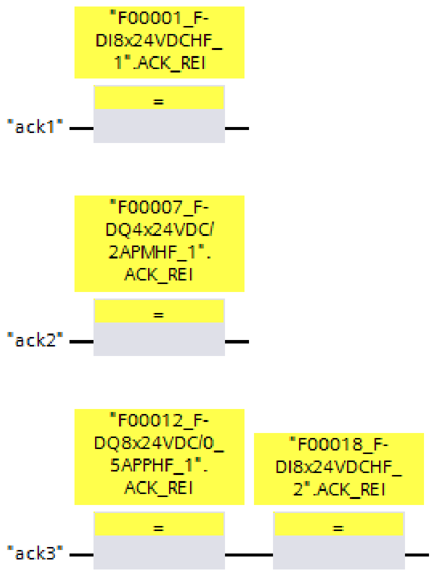
Figure 3-19: "ACK_GL" instruction



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 3-20: Separate reintegration of modules


"F00001_F-DI8x24VDCHF_1".ACK_REI
=
"ack1"
"F00007_F-DQ4x24VDC/2APMHF_1".ACK_REI
=
"ack2"
"F00012_F-DQ8x24VDC/0_5APPHF_1".ACK_REI
=
"F00018_F-DI8x24VDCHF_2".ACK_REI
=
"ack3"

4 Optimizing Safety Programs

4.1 Optimizing the compilation duration and runtime

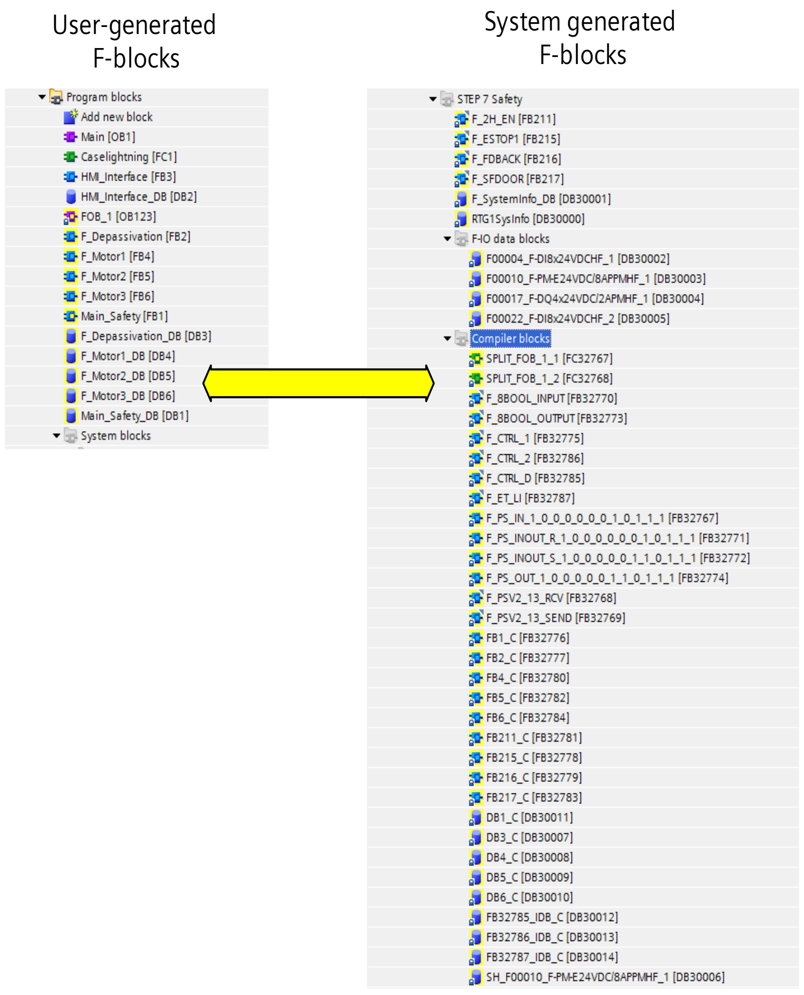
Introduction

User programming protection by 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



```
Program blocks
    Add new block
    Main [OB1]
    Caselightning [FC1]
    HMI_Interface [FB3]
    HMI_Interface_DB [DB2]
    FOB_1 [OB123]
    F_Depassivation [FB2]
    F_Motor1 [FB4]
    F_Motor2 [FB5]
    F_Motor3 [FB6]
    Main_Safety [FB1]
    F_Depassivation_DB [DB3]
    F_Motor1_DB [DB4]
    F_Motor2_DB [DB5]
    F_Motor3_DB [DB6]
    Main_Safety_DB [DB1]
  System blocks
```

```
STEP 7 Safety
  F_2H_EN [FB211]
  F_ESTOP1 [FB215]
  F_FDBACK [FB216]
  F_SFDOOR [FB217]
  F_SystemInfo_DB [DB30001]
  RTG1SysInfo [DB30000]
  F-IO data blocks
    F00004_F-DI8x24VDCHF_1 [DB30002]
    F00010_F-PM-E24VDC/8APPMHF_1 [DB30003]
    F00017_F-DQ4x24VDC/2APMHF_1 [DB30004]
    F00022_F-DI8x24VDCHF_2 [DB30005]
  Compiler blocks
    SPLIT_FOB_1_1 [FC32767]
    SPLIT_FOB_1_2 [FC32768]
    F_8BOOL_INPUT [FB32770]
    F_8BOOL_OUTPUT [FB32773]
    F_CTRL_1 [FB32775]
    F_CTRL_2 [FB32786]
    F_CTRL_D [FB32785]
    F_ET_LI [FB32787]
    F_PS_IN_1_0_0_0_0_0_0_1_0_1_1_1 [FB32767]
    F_PS_INOUT_R_1_0_0_0_0_0_0_1_0_1_1_1 [FB32771]
    F_PS_INOUT_S_1_0_0_0_0_0_1_1_0_1_1_1 [FB32772]
    F_PS_OUT_1_0_0_0_0_0_1_1_0_1_1_1 [FB32774]
    F_PSV2_13_RCV [FB32768]
    F_PSV2_13_SEND [FB32769]
    FB1_C [FB32776]
    FB2_C [FB32777]
    FB4_C [FB32780]
    FB5_C [FB32782]
    FB6_C [FB32784]
    FB211_C [FB32781]
    FB215_C [FB32778]
    FB216_C [FB32779]
    FB217_C [FB32783]
    DB1_C [DB30011]
    DB3_C [DB30007]
    DB4_C [DB30008]
    DB5_C [DB30009]
    DB6_C [DB30010]
    FB32785_IDB_C [DB30012]
    FB32786_IDB_C [DB30013]
    FB32787_IDB_C [DB30014]
    SH_F00010_F-PM-E24VDC/8APPMHF_1 [DB30006]
```

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

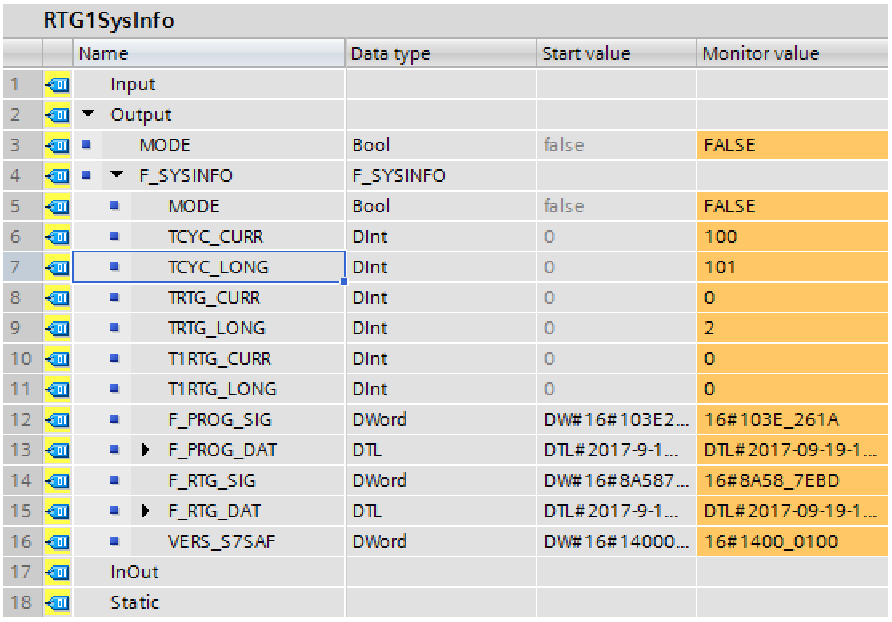
Depending on the application, it is not always possible to use all the suggestions. However, they show why certain programming methods cause shorter compilation and program runtimes than a non-optimized program.

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 whether or not, for example due to a memory error caused by EMC, a jump takes place despite the "false" condition.

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.

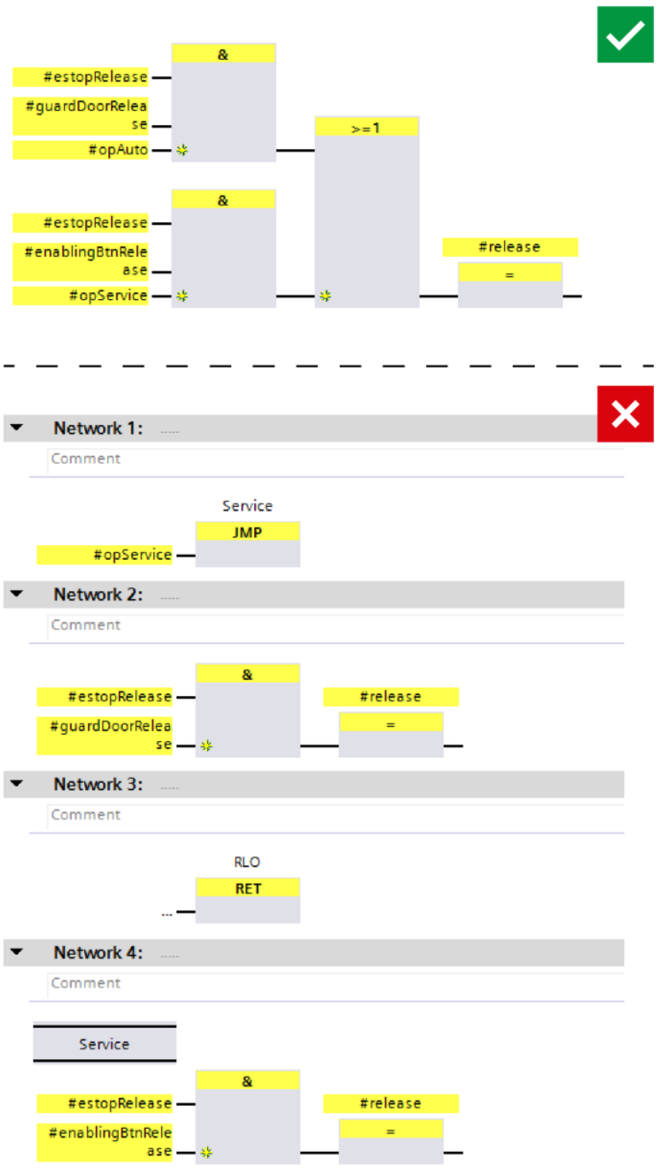
This requires that both alternatives (jump to label is "true" or "false") be calculated in their entirety in the protection program.

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 failsafe time value requires considerable effort and regeneration for each single timer block.

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

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.

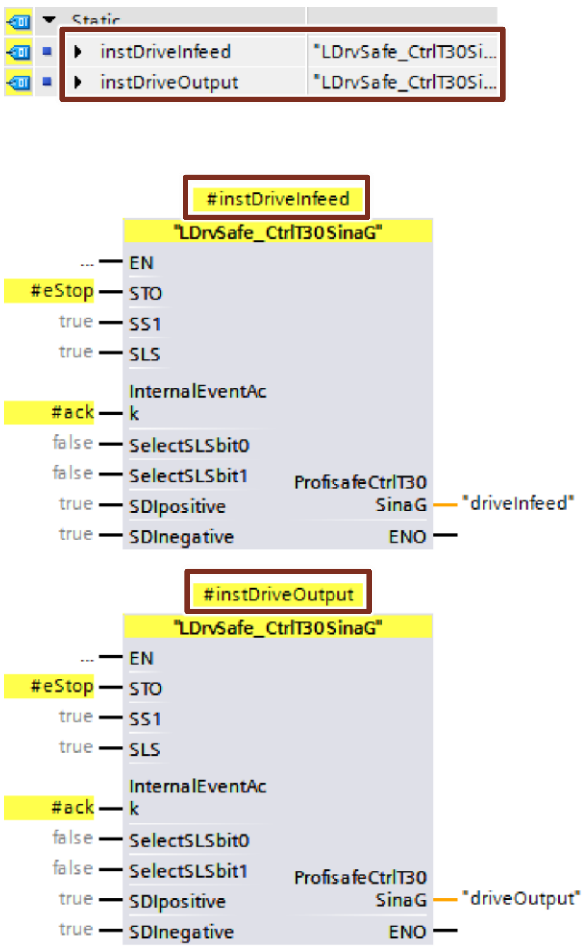
Advantages

- 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:

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

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 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 a overflow or a 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

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 math functions. This must be caught by the user in the program. Therefore, interconnect the ENO output of the arithmetic 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 arithmetic functions	
Access via HMI An HMI is used to write (modify) data (bit memories, DBs) that is read in the safety program. As by default, communication has a higher priority than Safety, this can result in data corruption. Information on this 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 Identical to standard access; access can occur at runtime when there are unfavorable defaults for generating a pointer to F-areas (inputs, outputs, data blocks, etc.).	

Additional information

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

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 the compile a diverse (encoded) safety program that 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.

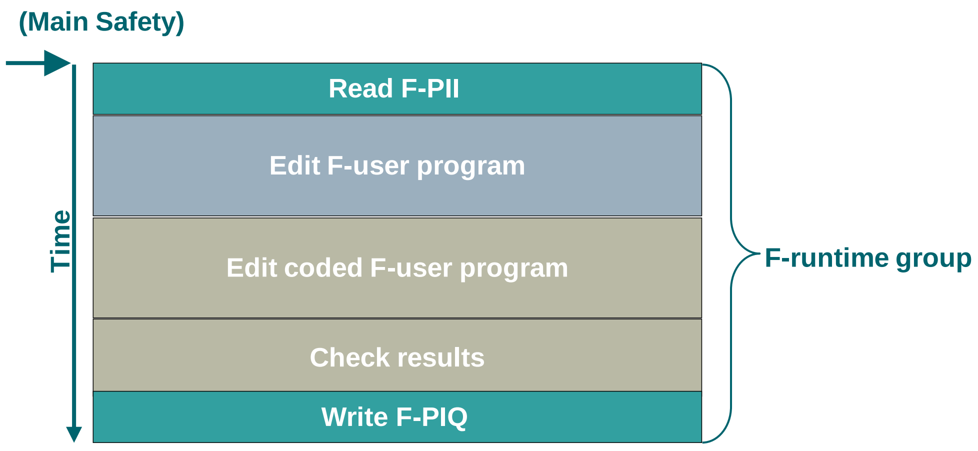


Figure 5-1 Safety program processing sequence

Data corruption

Data corruption means that data of the safety program has been tampered with 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 cable casing being squashed. 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

Industry Online Support

Do you have any questions or need assistance?

Siemens Industry Online Support offers round the clock access to our entire service and support know-how and portfolio.

The Industry Online Support is the central address for information about our products, solutions and services.

Product information, manuals, downloads, FAQs, application examples and videos – all information is accessible with just a few mouse clicks:

support.industry.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:

siemens.com/SupportRequest

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.de/sitrain

Service offer

Our range of services includes the following:

- Plant data services
- Spare parts services
- Repair services
- On-site and maintenance services
- Retrofitting and modernization services
- Service programs and contracts

You can find detailed information on our range of services in the service catalog web page:

support.industry.siemens.com/cs/sc

Industry Online Support app

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

support.industry.siemens.com/cs/ww/de/sc/2067

6.2 Industry Mall

The Siemens Industry Mall is the platform on which the entire siemens Industry product portfolio is accessible. From the selection of products to the order and the delivery tracking, the Industry Mall enables the complete purchasing processing – directly and independently of time and location:
mall.industry.siemens.com

6.3 Links and literature

Table 6-1

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to this entry page of this 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 Styleguide for SIMATIC S7-1200/1500 https://support.industry.siemens.com/cs/ww/en/view/109478084
\5\	SIMATIC Industrie Software SIMATIC Safety – Configuring and Programming https://support.industry.siemens.com/cs/ww/en/view/54110126
\6\	Topic page "Safety Integrated – Safety in Factory Automation" https://support.industry.siemens.com/cs/ww/en/view/109747812

6.4 Change documentation

Table 6-2

Version	Date	Modifications
V1.0.1	10/2017	First edition
V1.1.0	09/2020	Adaptations and corrections
V1.2.0	09/2021	Adaptations and corrections