SIEMENS

Ingenuity for life

Safety Programming Guideline for SIMATIC S7-1200/1500

SIMATIC Safety Integrated

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

Siemens Industry Online Support

Warranty and Liability

Table of Contents

1 Introduction

The new controller generation SIMATIC S7-1200 and S7-1500 has an up-to-date system architecture and, together with TIA Portal, offers new and efficient programming and configuration options.

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

- CPU stops
- Long compilation processes
- 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 in this document can be universally used on the S7-1200 and S7-1500 controllers.

Advantages

Following the recommendations given in this document provides you with many advantages:

- Reusability of program parts
- Easier acceptance (code review, error detection and correction)
- More flexibility in terms 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 recommendations given in the programming guideline and the programming styleguide always apply to programming safety programs.

Programming Guideline for SIMATIC S7-1200/1500:

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

Programming Styleguide for SIMATIC S7-1200/1500:

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

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

2 Configuring Fail-Safe Controllers

2.1 Selecting the suitable F-CPU

Selecting the F-CPU depends on the following factors:

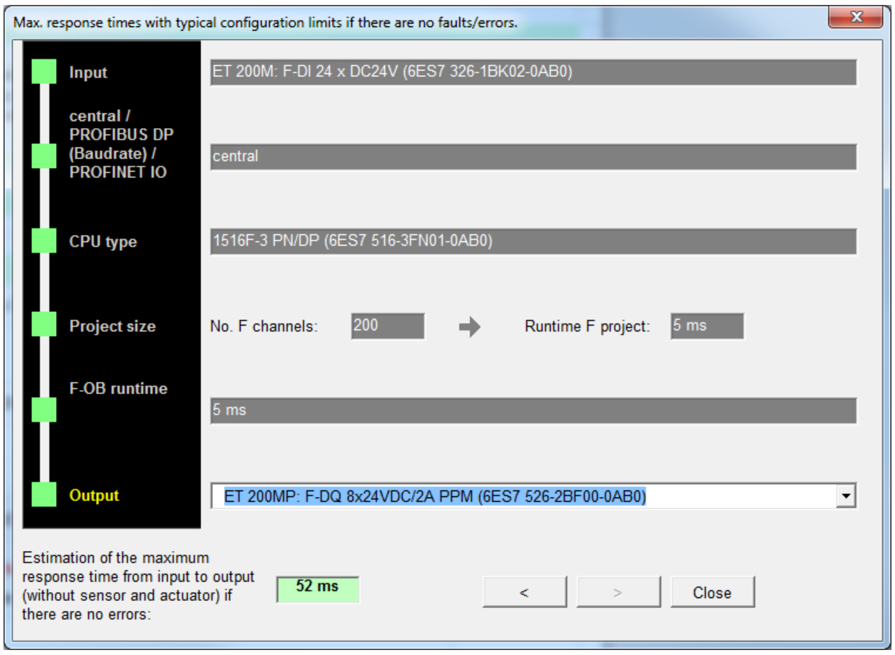
- Runtime of the safety program
- PROFIsafe communication time
- Response time of the safety function
- Number of required inputs and outputs
- Number of connected I/O devices

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

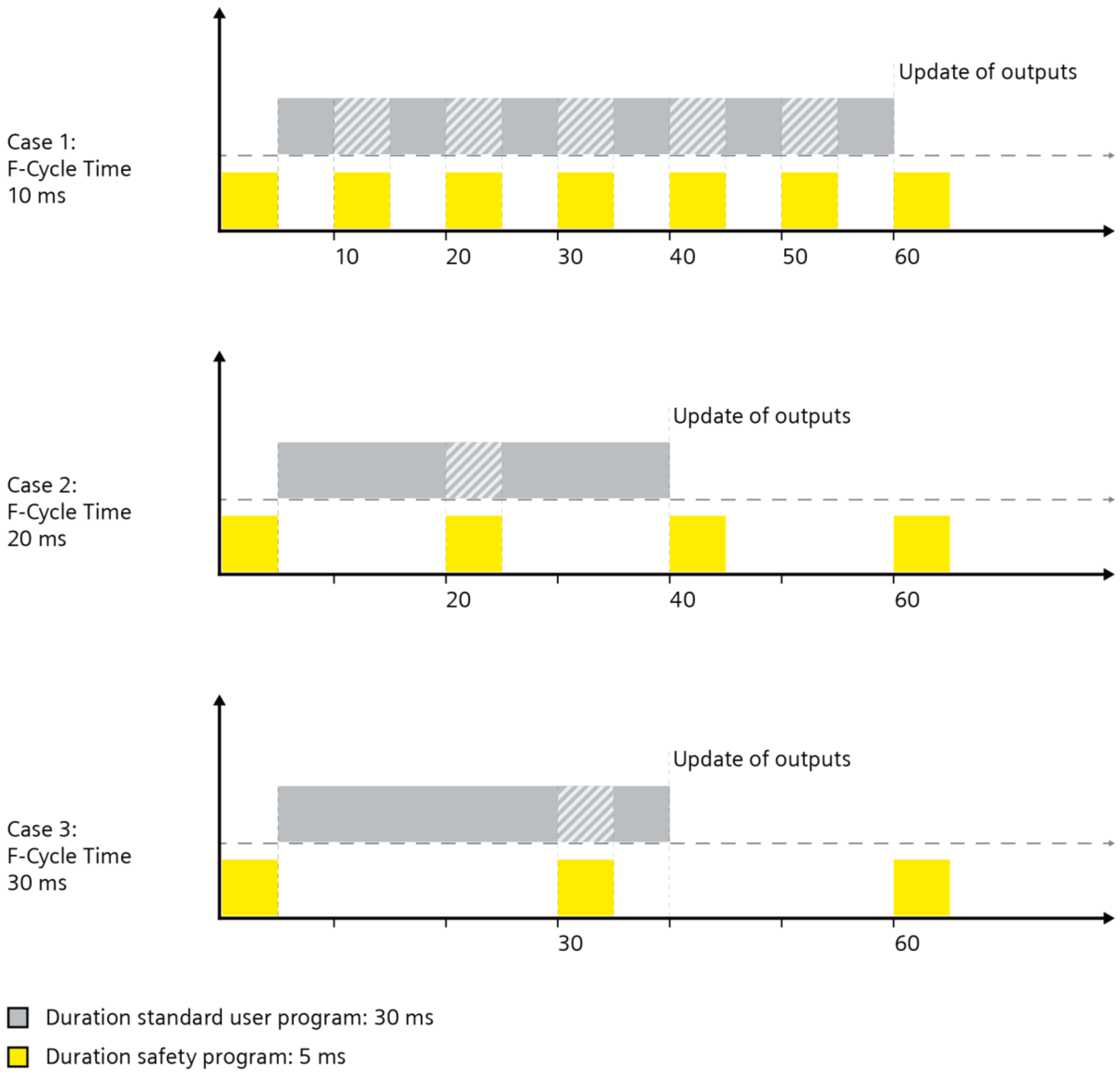
Influence of the safety program's cycle time on the standard user program

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 safety program's cycle time 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 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.

NOTICE	The cycle time must be longer than the execution duration of the safety program.

2.2 PROFIsafe address types

The PROFIsafe address is used to uniquely address F-I/O and protect standard addressing mechanisms such as IP addresses. Uniqueness is defined differently for F-I/O of PROFIsafe address type 1 and F-I/O of PROFIsafe address type 2.

Table 2-1: Differences between the PROFIsafe address types

PROFIsafe address type 1	PROFIsafe address type 2
• The uniqueness of the PROFIsafe address is ensured only by the F-destination address. • The F-destination address must be unique throughout the network and the CPU. • In the safety summary, each F-destination address has to be checked for network- and CPU-wide uniqueness by making sure that the F-destination address ranges of all F-CPUs do not overlap. • The F-destination address and the F-source address are included in the safety program's CRC.	• The uniqueness of the PROFIsafe address is ensured by combining the F-source address and the F-destination address. • The F-destination address must be unique throughout the CPU and differ from all other F-destination addresses of PROFIsafe address type 1 in the same network. • The F-destination address used for the F-I/O of an F-CPU must be unique throughout the network. • The F-destination address and the F-source address are included in the safety program's CRC.

You must ensure that each PROFIsafe address is unique.

The ever increasing networking of plants and plant sections – especially if configured separately – makes accurate planning of the PROFIsafe address assignment all the more necessary.

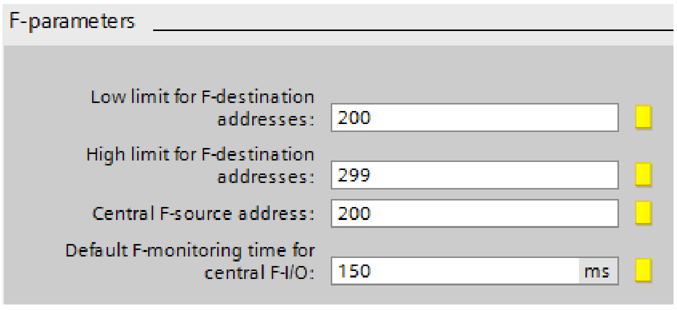
The use of F-I/O of PROFIsafe address type 2 makes handling PROFIsafe addresses easier. With mixed configurations or pure address type 1 configurations, however, one has to be more careful.

Recommendation

- Already at the outset of the project, look at possible communication relationships and network topologies. Consult with the parties involved to derive measures for assigning PROFIsafe addresses.
- Assign separate address ranges to PROFIsafe address types 1 and 2:
  - Assign a low number range to F-I/O of PROFIsafe address type 1$^{1)}$.
  - Assign a high number range to F-I/O of PROFIsafe address type 2.
- Always define unique F-source addresses for all F-CPUs. This makes both working across projects and later extensions easier.

[1] You can define the allowed range for F-destination addresses of PROFIsafe address type 1 in the CPU properties.

Figure 2-3: Defining the address range for F-destination addresses



F-parameters

Low limit for F-destination addresses:	200
High limit for F-destination addresses:	299
Central F-source address:	200
Default F-monitoring time for central F-I/O:	150 ms

Additional information

For more information about PPROFIsafe address types, visit Siemens Industry Online Support:

What is the difference between the PROFIsafe address types 1 and 2 in relation to the uniqueness of the PROFIsafe address?

https://support.industry.siemens.com/cs/ww/en/view/109479905

How do you assign PROFIsafe addresses so that they are unique network-wide and CPU-wide?

https://support.industry.siemens.com/cs/ww/en/view/109740240

2.3 Protecting the F-CPU against unauthorized access

To prevent unauthorized modifications or tampering with a safety program, you must implement appropriate access protection.

This can be done, for example, through organizational measures (such as locking the control cabinet).

However, easier and more efficient access protection can be achieved by assigning passwords.

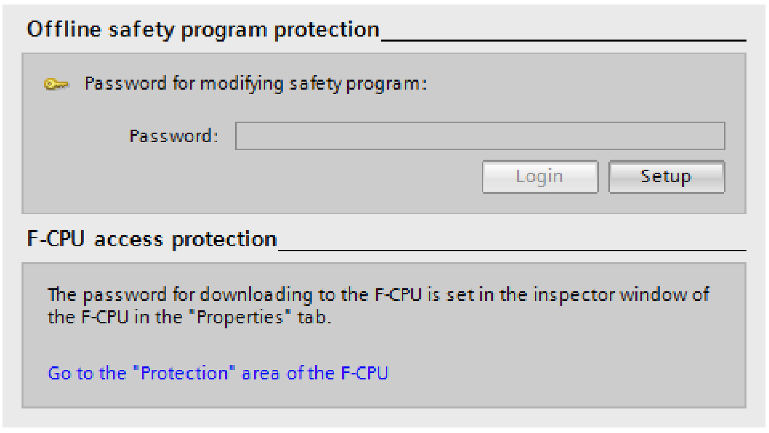
You can set up separate access protection mechanisms for the safety program and the F-CPU.

Access protection for the safety program

Access protection for the safety program makes sure that the F-program is not modified by unauthorized persons.

You define the password for access protection for the safety program in Safety Administration in TIA Portal.

Figure 2-4: Defining the password for the safety program


Offline safety program protection

Password for modifying safety program:

Password:

F-CPU access protection

The password for downloading to the F-CPU is set in the inspector window of the F-CPU in the "Properties" tab.

Go to the "Protection" area of the F-CPU

Once you have logged in with the password for the safety program, you can remove the access rights to the safety program as follows:

- Log out of Safety Administration
- In the menu bar, "Online > Delete access rights"
- Close TIA Portal

Note

Collaboration of programmers with and without rights for the user program

Changes to standard DBs that are read/write accessed by the safety program require a recompilation of the safety program. These standard DBs are not subject to the access permission for the safety program. Therefore, data exchange between the F-program and the standard program requires a defined interface that the programmer of the standard user program does not have to change during his work.

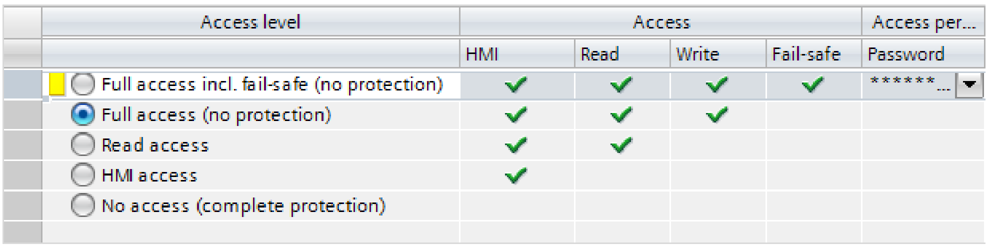
For more information about this data exchange, please refer to Chapter 3.9.

Access protection for the F-CPU

Access protection for the F-CPU ensures that only authorized persons can download a safety program to the device or disable safety mode.

The password for the F-CPU is defined in the CPU properties.

Figure 2-5: Defining the password for the F-CPU



Access level	Access				Access per...
	HMI	Read	Write	Fail-safe	Password
Full access incl. fail-safe (no protection)	✓	✓	✓	✓	******
[✓ Full access (no protection)]	✓	✓	✓		
Read access	✓	✓			
HMI access	✓				
No access (complete protection)					

Access protection applies only to the appropriate F-CPU. This password is also used for identification of the F-CPU and must therefore be unique throughout the network.

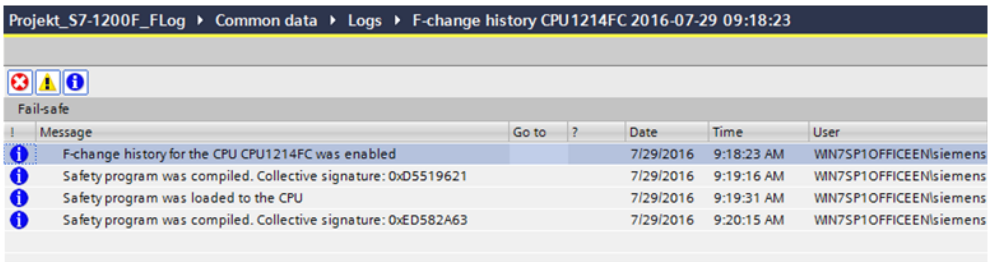
2.4 F-change history

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.

F-change history logs the following:

- F-collective signature
- User name
- Compile time stamp
- Download of the safety program with time stamp
- Compiled F-blocks with signature and time stamp

Figure 2-6: F-change history



Projekt_S7-1200F_FLog ▸ Common data ▸ Logs ▸ F-change history CPU1214FC 2016-07-29 09:18:23

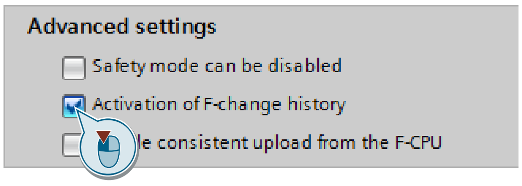
Fail-safe

!	Message	Go to	?	Date	Time	User
●	F-change history for the CPU CPU1214FC was enabled			7/29/2016	9:18:23 AM	WIN7SP1OFFICEEN\siemens
●	Safety program was compiled. Collective signature: 0xD5519621			7/29/2016	9:19:16 AM	WIN7SP1OFFICEEN\siemens
●	Safety program was loaded to the CPU			7/29/2016	9:19:31 AM	WIN7SP1OFFICEEN\siemens
●	Safety program was compiled. Collective signature: 0xED582A63			7/29/2016	9:20:15 AM	WIN7SP1OFFICEEN\siemens

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.

Figure 2-7: Activating F-change history



Advanced settings

- [ ] Safety mode can be disabled
- [x] Activation of F-change history
- [ ] ...le consistent upload from the F-CPU

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.

2.5 Consistently uploading F-CPUs

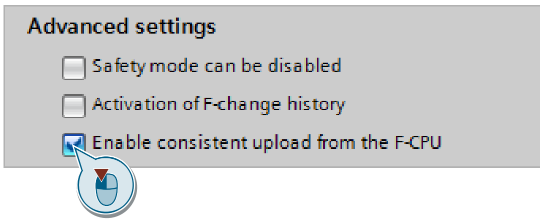
TIA Portal V14 SP1 and higher allows you to consistently upload fail-safe SIMATIC S7-1500 CPUs from the automation system to TIA Portal.

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.

Figure 2-8: Enabling "Consistent upload"



Advanced settings

- [ ] Safety mode can be disabled
- [ ] Activation of F-change history
- [x] Enable consistent upload from the F-CPU

Advantages

As there is no complex "offline" project management, you can avoid errors and further reduce the service effort.

2.6 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

3.1.1 Defining a program structure

Recommendation

- Modularly divide the program code, e.g.,
  - into subparts for detecting, evaluating, reacting or
  - plant sections.
- In the preliminary stages, create a specification for each module (based on the risk assessment requirements).
- Avoid complex signal paths.

Advantages

- Minimizes complexity.
- Reduces programming errors.
- Allows the program code to be analyzed/tested without running the program (e.g., code review or PLCSIM).
- Easily expandable. Simplifies renewed acceptance.
- Reuse of program parts without renewed acceptance.
- Allows advance testing and acceptance of finished program parts.

Example

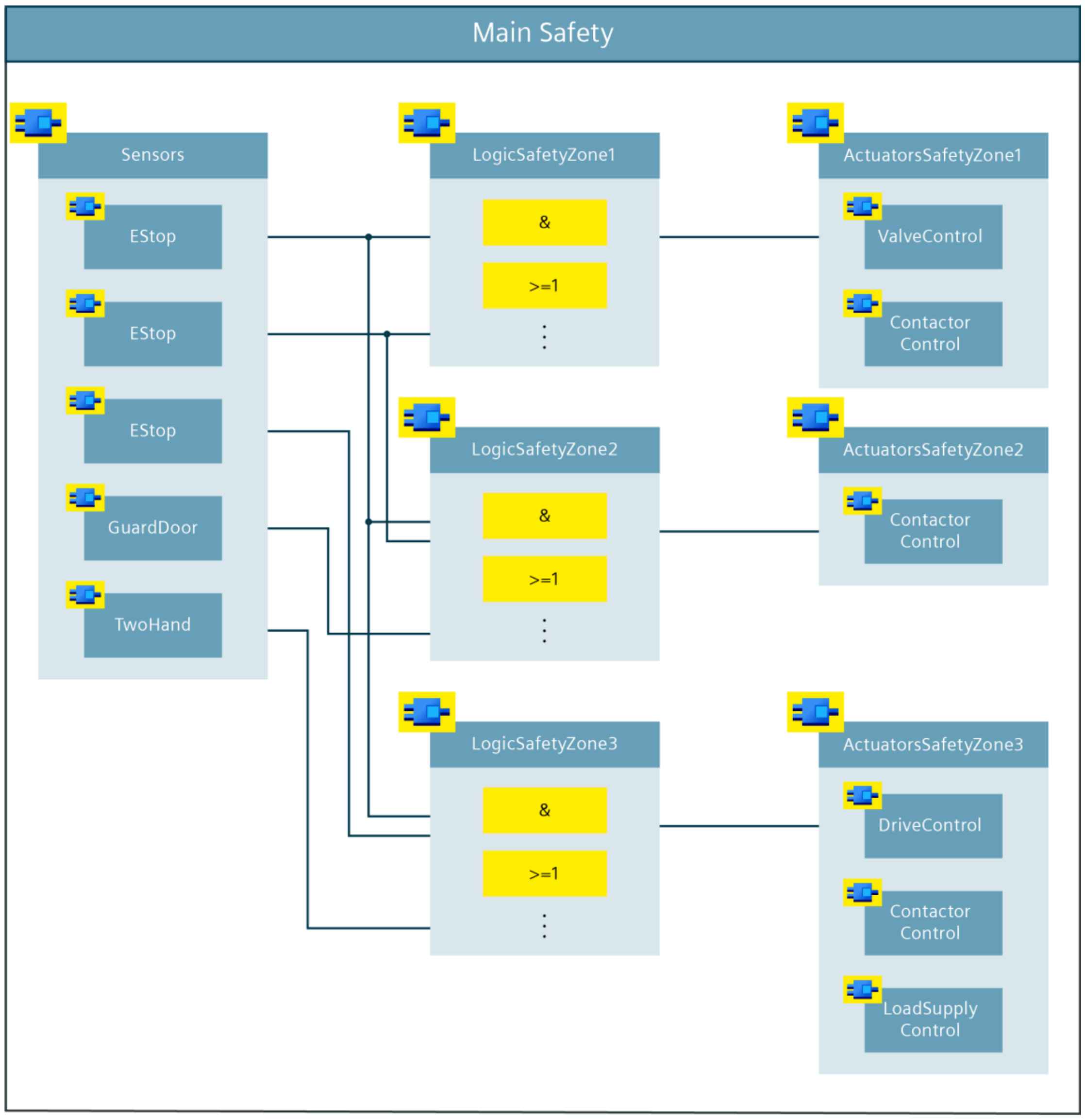
The following figure shows a safety application that is divided into three machine areas (safety zones).

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 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 standard user programs, the number of call levels is limited depending on the CPU. 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.

System instructions ("ESTOP1", "SF_DOOR", etc.) are not included in the number of call levels.

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 limit specified here.

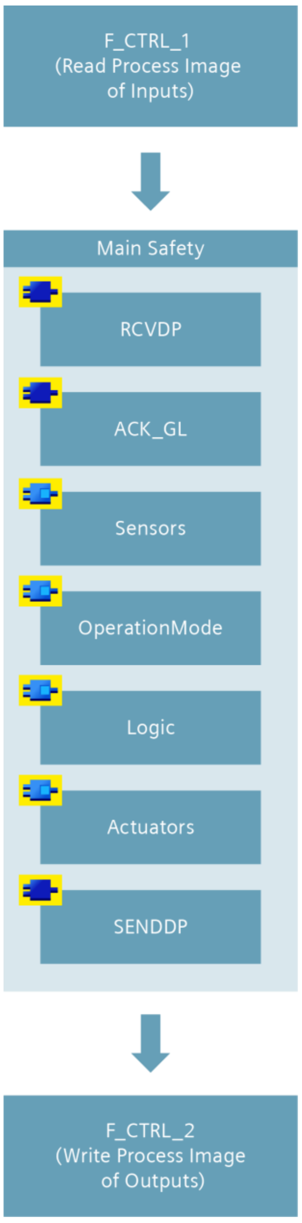
3.1.3 Call sequence of the blocks in the Main Safety

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)

Figure 3-2: Call sequence in the Main Safety



Advantages

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

3.1.4 F-suitable PLC data type

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

F-suitable PLC data types have the following features:

- F-suitable PLC data types are declared and used in the same way as PLC data types.
- All data types that are allowed in the safety program can be used in F-suitable PLC data types.
- Nesting F-suitable PLC data types within other F-suitable PLC data types is not supported.
- F-suitable PLC data types can be used both in the safety program and in the standard user program.

Recommendation

- Create F-suitable PLC data types to structure data also in the safety program.
- Use F-suitable PLC data types to transfer large numbers of variables to blocks.
- Use F-suitable PLC data types for access to I/O ranges.

  In this context, follow the below rules:

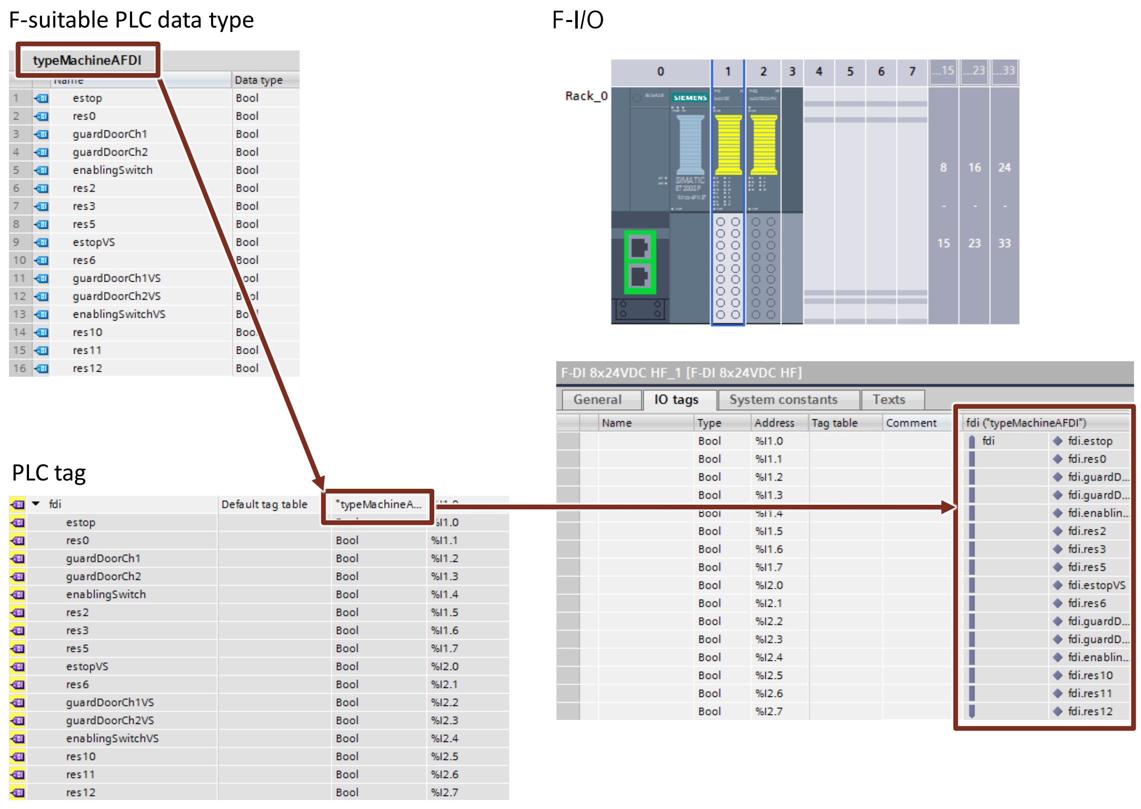
  - The structure of the tags of the F-suitable PLC data type must match the channel structure of the F-I/O.
  - Example of an F-suitable PLC data type for an F-I/O with 8 channels:
    - 8 BOOL variables (channel value) or
    - 16 BOOL variables (channel value + value status)
  - Access to F-I/O is only allowed for activated channels. When configuring a 1oo2 evaluation, the higher-level channel is always deactivated.

Advantages

A change in a PLC data type is automatically updated in all points of use in the user program.

Example

Figure 3-3: Access to I/O ranges with F-suitable PLC data types



typeMachineAFDI

	Name	Data type
1	estop	Bool
2	res0	Bool
3	guardDoorCh1	Bool
4	guardDoorCh2	Bool
5	enablingSwitch	Bool
6	res2	Bool
7	res3	Bool
8	res5	Bool
9	estopVS	Bool
10	res6	Bool
11	guardDoorCh1VS	Bool
12	guardDoorCh2VS	Bool
13	enablingSwitchVS	Bool
14	res10	Bool
15	res11	Bool
16	res12	Bool

fdi	Default tag table	"typeMachineA...	
estop			%I1.0
res0		Bool	%I1.1
guardDoorCh1		Bool	%I1.2
guardDoorCh2		Bool	%I1.3
enablingSwitch		Bool	%I1.4
res2		Bool	%I1.5
res3		Bool	%I1.6
res5		Bool	%I1.7
estopVS		Bool	%I2.0
res6		Bool	%I2.1
guardDoorCh1VS		Bool	%I2.2
guardDoorCh2VS		Bool	%I2.3
enablingSwitchVS		Bool	%I2.4
res10		Bool	%I2.5
res11		Bool	%I2.6
res12		Bool	%I2.7

F-DI 8x24VDC HF_1 [F-DI 8x24VDC HF]

General | IO tags | System constants | Texts

Name	Type	Address	Tag table	Comment	fdi ("typeMachineAFDI")	
	Bool	%I1.0			fdi	fdi.estop
	Bool	%I1.1				fdi.res0
	Bool	%I1.2				fdi.guardD...
	Bool	%I1.3				fdi.guardD...
	Bool	%I1.4				fdi.enablin...
	Bool	%I1.5				fdi.res2
	Bool	%I1.6				fdi.res3
	Bool	%I1.7				fdi.res5
	Bool	%I2.0				fdi.estopVS
	Bool	%I2.1				fdi.res6
	Bool	%I2.2				fdi.guardD...
	Bool	%I2.3				fdi.guardD...
	Bool	%I2.4				fdi.enablin...
	Bool	%I2.5				fdi.res10
	Bool	%I2.6				fdi.res11
	Bool	%I2.7				fdi.res12

3.2 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, include 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 include the signature in the block comment. This makes it easier to track functional changes of the block.

```
//=============================================================================
// Company
//-----------------------------------------------------------------------------
// Library: (that the source is dedicated to)
// Tested with: (test system with FW version)
// Engineering: TIA Portal (SW version)
// Restrictions: (OB types, etc.)
// Requirements: (hardware, technological package, memory needed, etc.)
// Functionality: (that is implemented in the block)
//-----------------------------------------------------------------------------
// Reference to Safety Requirement Specification:
// Safety related information: (SIL/PL (Cat.), DC, methods against CFF for connected
subsystems)
//-----------------------------------------------------------------------------
// Change log table:
// Version   Date          Signature     Expert in charge   Changes applied
// 01.00.00  (dd.mm.yyyy)  (Block CRC)   (Name of expert)   First released version
//=============================================================================
```

3.3 Functional identifiers of variables

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 name of the variables with the appropriate suffixes. The identifier reflects the meaning and purpose of the variables in the source code context.

Choose the variable identifier such that it reflects the logic "1" state ("true").

For example, "maintDoorEnable" or "conveyorSafetyRelease".

Note 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.

3.4 True & False

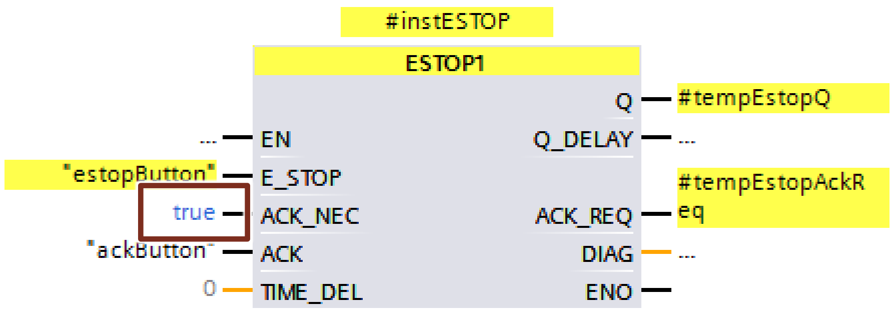
Regarding the use of "TRUE" and "FALSE" signals in safety programs, there are two different use cases:

- Actual parameters on blocks
- Assignments on operations

Actual parameters on blocks

For S7-1200/1500 controllers, you can use the Boolean constants "FALSE" for "0" and "TRUE" for "1" as actual parameters for interconnecting formal parameters during block calls in the safety program. Only the keyword "FALSE" or "TRUE" is written to the formal parameter.

Figure 3-4: "TRUE" / "FALSE" signals as actual parameters



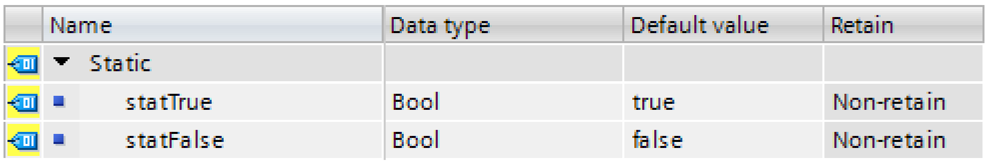
Assignments on operations

To generate "TRUE" / "FALSE" signals for operations, proceed as follows:

8. Create two static tags, "statTrue" and "statFalse", of the BOOL data type.
9. Assign the default "true" to the "statTrue" tag.
10. Assign the default "false" to the "statFalse" tag.

In the complete function block, you can use the tags as reading "TRUE" and "FALSE" signals.

Figure 3-5: Declaration of "TRUE" and "FALSE" signals



Name	Data type	Default value	Retain
Static			
statTrue	Bool	true	Non-retain
statFalse	Bool	false	Non-retain

3.5 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 have to be accepted only once
- Quick programming of additional functions and projects
- Versioning possible with the TIA Portal library concept
- 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.

3.5.1 Standardizing sensor evaluation

Recommendation

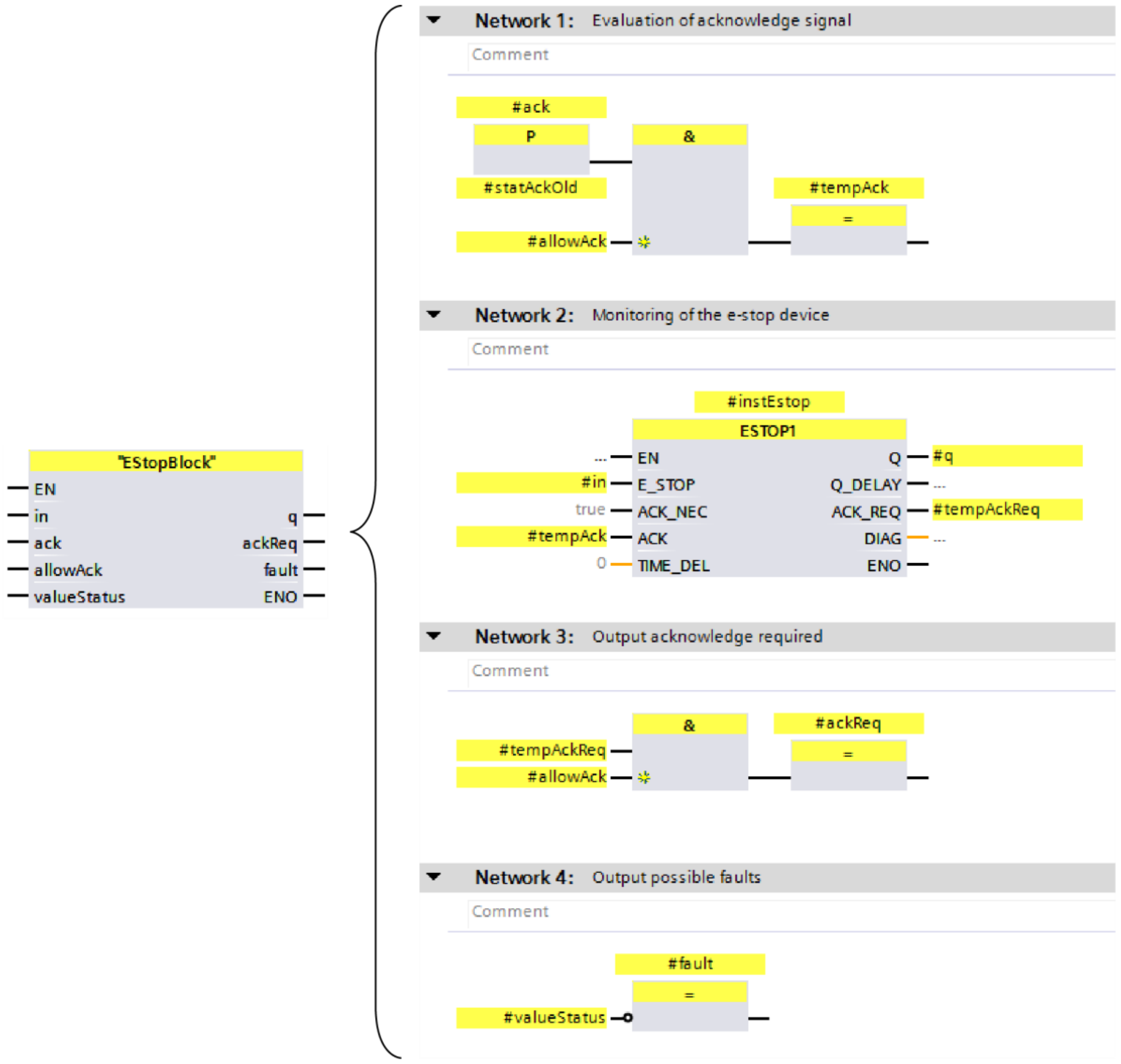
Create a separate function block for each sensor type (e.g., emergency stop command device, safety door, light curtain, etc.) that combines the evaluation of the sensor and the necessary auxiliary functions. Use this sensor block for other sensors of the same type.

Create F-data types for complex sensors.

Auxiliary functions for a safe sensor include, for example:

- Reset
- Restart interlock
- Time functions
- Edge evaluation
- Startup test
- Provision of diagnostic information

Figure 3-6: Standardizing sensor evaluation



3.5.2 Standardizing actuator control

Recommendation

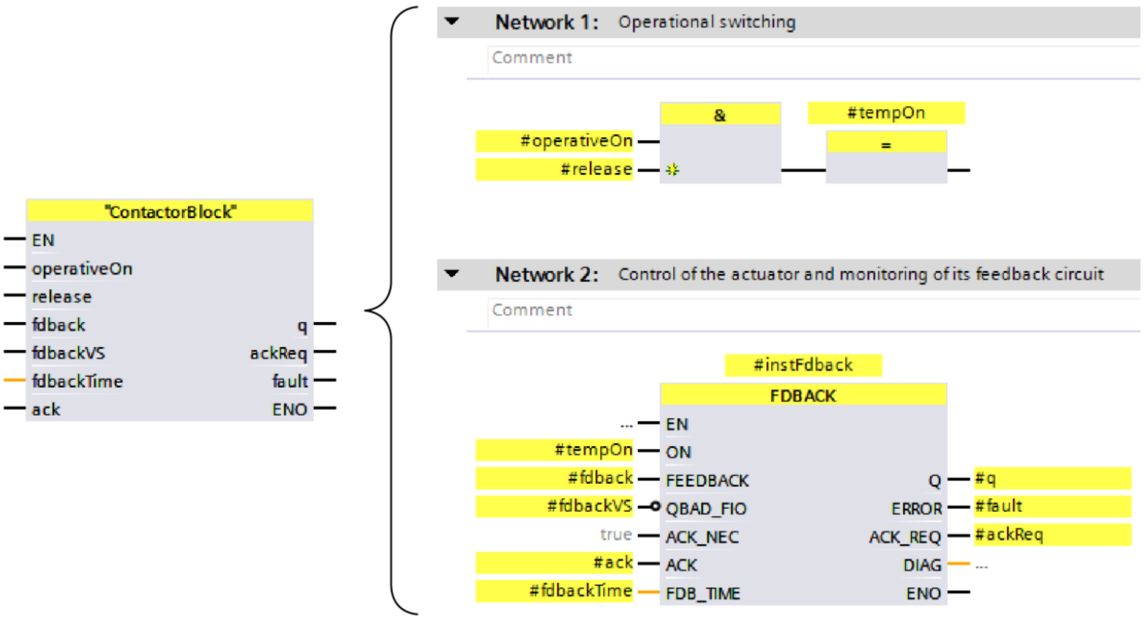
Create a separate function block for each actuator type (e.g., contactors, valves, drives, etc.) that combines actuator control and the necessary auxiliary functions. Use this actuator block for other actuators of the same type.

Create F-data types for complex actuators.

Auxiliary functions for a safe actuator include, for example:

- Feedback circuit monitoring
- Error acknowledgment
- Edge evaluation
- Time functions
- Functional switching
- Provision of diagnostic information

Figure 3-7: Standardizing actuator control



3.6 Programming logic operations

Tasks 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
- Reduce the use of SR blocks to a minimum
- Avoid jumps in binary logic

3.7 Programming mode-dependent safety functions

Recommendation

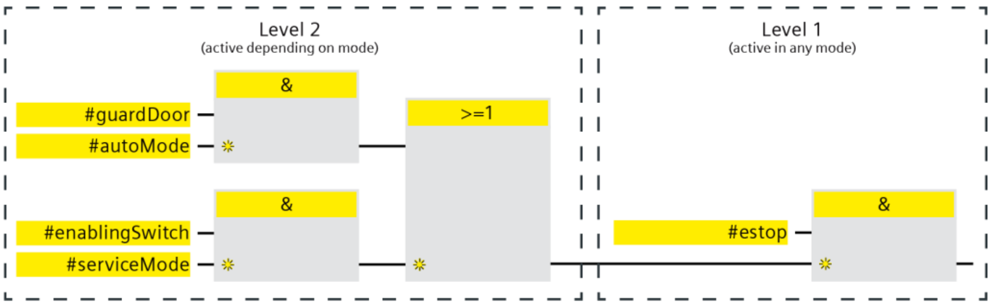
Divide the logic into different levels (see IEC 62061):

- 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 mode-dependent safety functions.
  - 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 3-8: Programming modes



3.8 Accessing global data

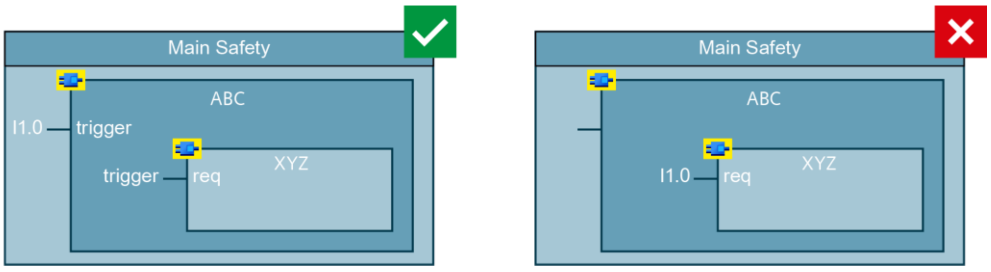
Recommendation

- Connect global data (inputs, outputs, data blocks) at the highest block hierarchy level (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
- Reduces programming errors
- Makes the overall program easier readable as the general function of a block can already be estimated from the interfaces.

Figure 3-9: Connecting global data



3.9 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.

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. As a result, it is not possible to completely separate both program parts.

In order to move non-safety-related functions to the standard user program, you have to define an interface. Global data blocks are best suited for this purpose.

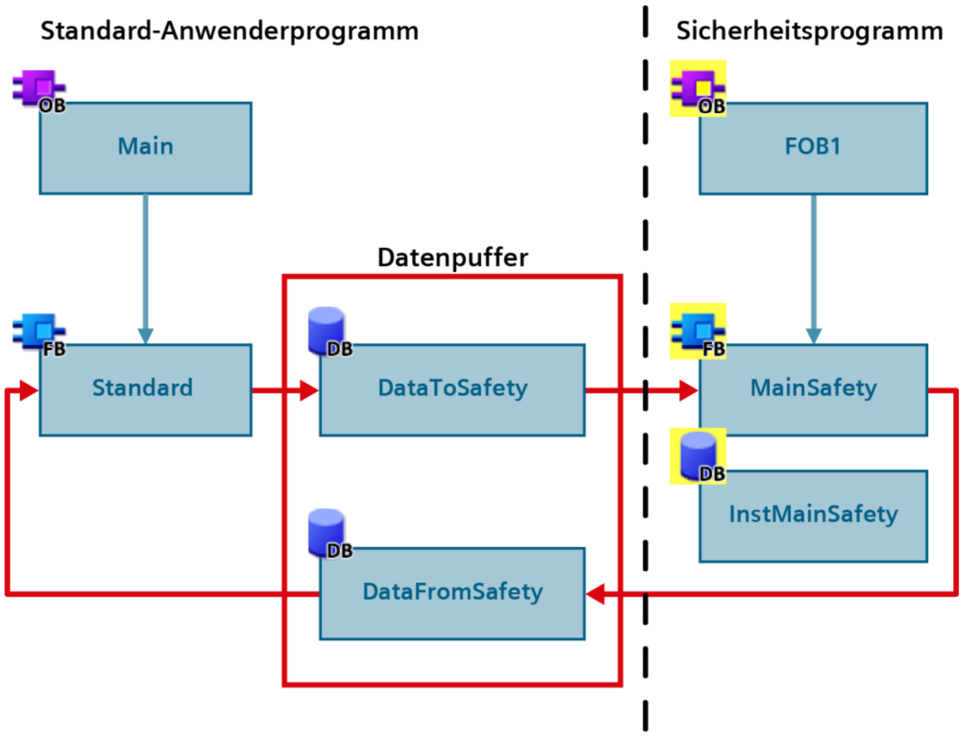
Recommendation

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.

You should not store any other information (e.g., diagnostic data from the standard user program) in the data blocks as each modification of the data block involves a modification of the safety program.

Figure 3-10: Data exchange between standard user program and safety program



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 safety program's signature
- Minimized risk of downtimes caused by data corruption due to write access to the safety program
- Simplified typing 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, provided that interfaces have already been defined

3.9.1 Reading diagnostic and message information from the safety program

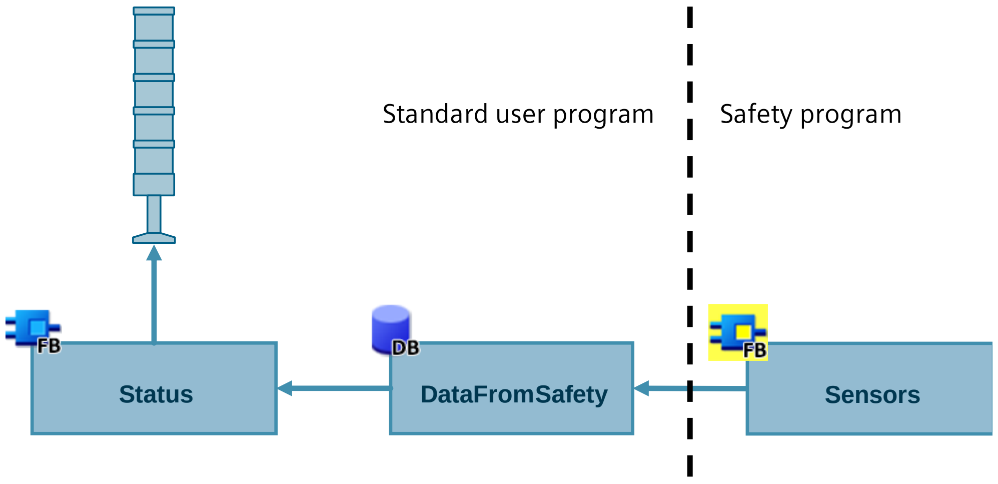
A frequent application for data exchange between the standard user program and the safety program is the visualization of diagnostic and message information such as:

- 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. Smaller 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-data types across all F-function blocks.

Figure 3-11: Reading diagnostic and message information from the safety program



3.9.2 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.

3.9.3 Using non-safe inputs in the safety program

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 acknowledgment / reset buttons or mode selectors. 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.

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.

3.9.4 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 use this convenience for operator control and monitoring of processes and plants even in safety-related applications, additional measures are required.

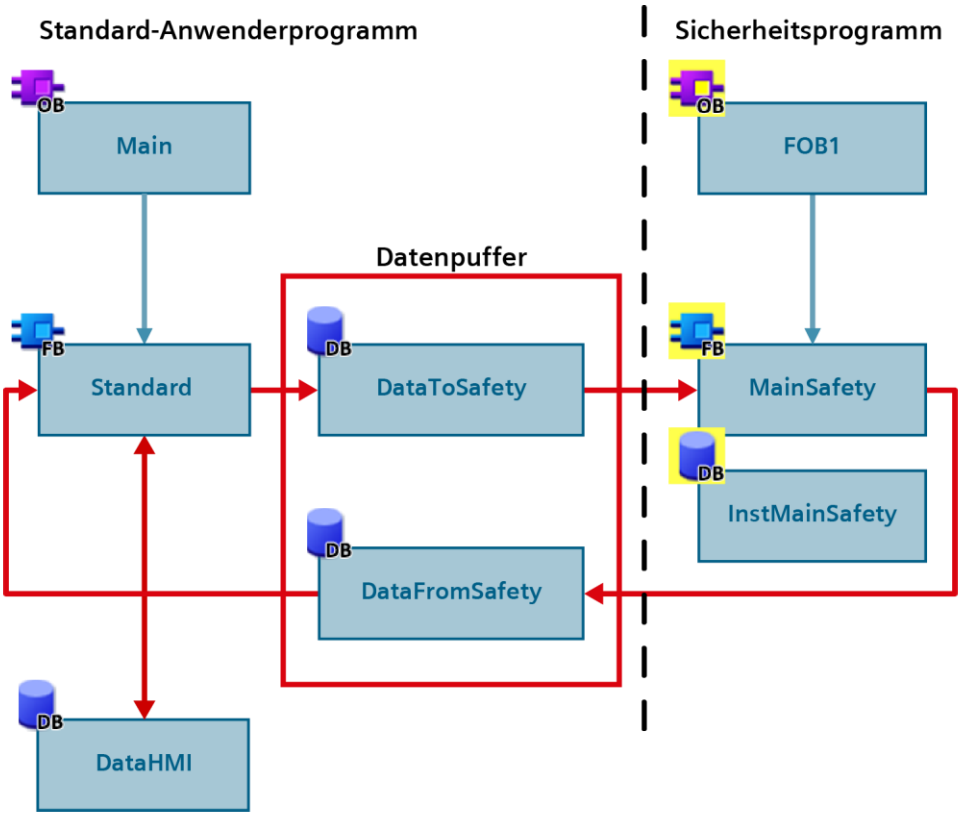
Writing tags from the HMI to the safety program is a problem for the following reasons:

- 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

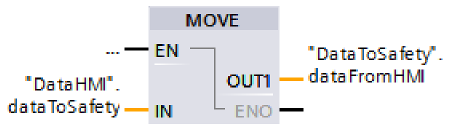
Use another data block for communication with the HMI and copy the safety-related data in the standard user program to the data buffer.

Figure 3-12: 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 3-13: 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/67634251

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:

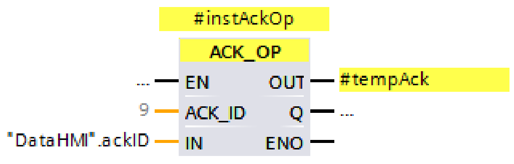
1. Change in/out IN to value "6" for exactly one cycle.
2. Change in/out IN to value at input "ACK_ID" for exactly one cycle within one minute.

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 3-14: "ACK_OP" system block



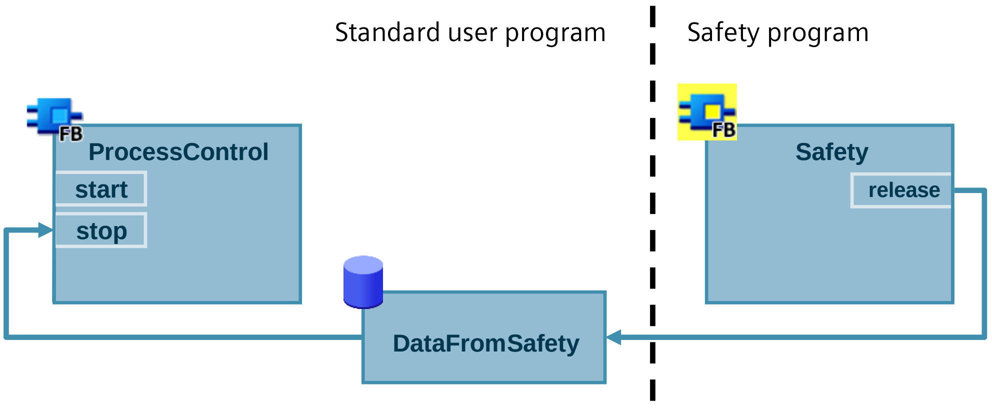
3.10 Resetting functional switching

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

Recommendation

- Lock process control in the standard user program with the release signal from the safety program. As a result, safe shutdown also resets process control.
- Transfer the release signal from the safety program using a global data block (see also Chapter 3.9).

Figure 3-15: Locking process control with the release signal



3.11 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.

3.11.1 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 diagnostic 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:

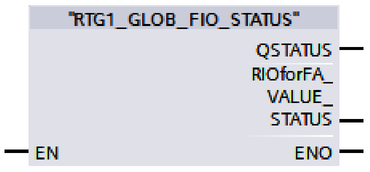
- The "ACK_REQ" tag of the module's F-I/O data block is "true"
- LEDs of channel and module flash alternately between red and green

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.

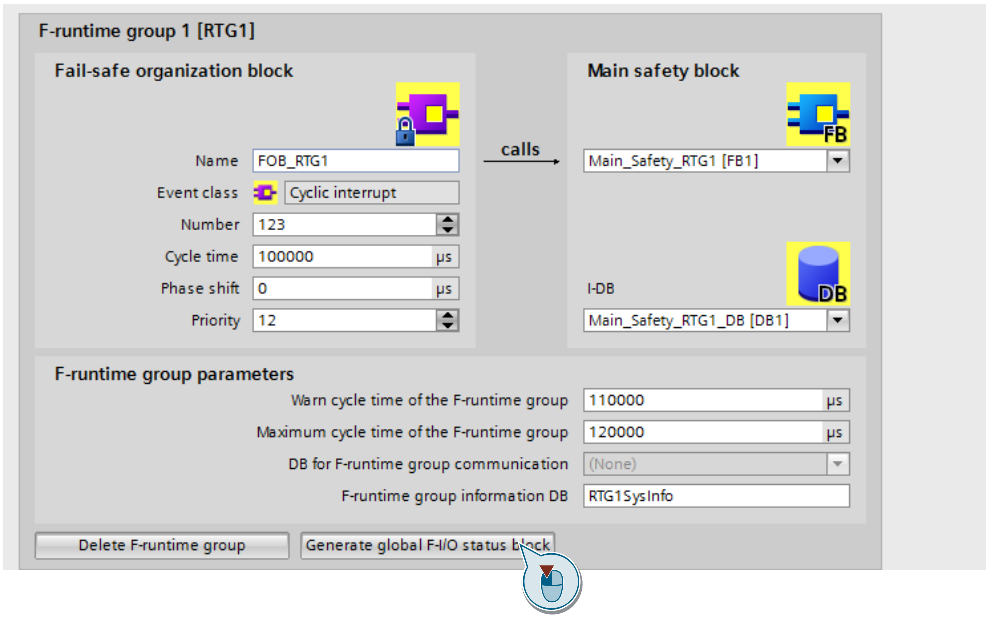
This block evaluates whether instead of the process values, substitute values are output for at least one F-I/O or at least one channel of an F-I/O of an F-runtime group. The "QSTATUS" output shows the result of the evaluation. This process does not consider F-I/Os you disabled using the DISABLE tag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

Figure 3-16: 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 3-17: 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

calls

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

3.11.2 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

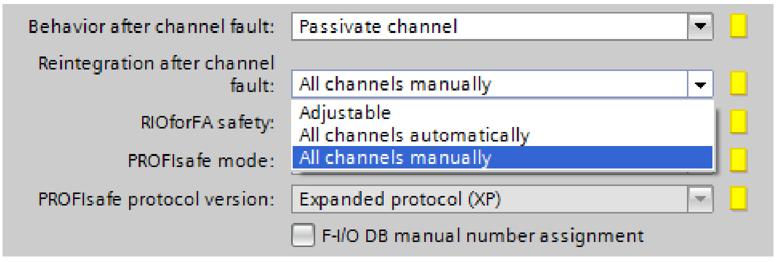
If automatic reintegration is permissible for a certain process depends on the risk assessment.

NOTE Automatic reintegration concerns F periphery/channel faults (e.g., discrepancy faults, short-circuits). Communication faults require manual reintegration (see chapter 3.11.3).

Modules that support "RIOforFA"

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

Figure 3-18: Parameterizing automatic reintegration



Behavior after channel fault: Passivate channel

Reintegration after channel fault: All channels manually

RIOforFA safety:

PROFIsafe mode:

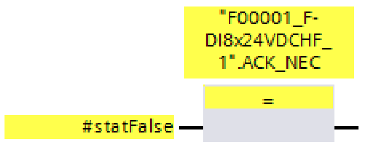
PROFIsafe protocol version: Expanded protocol (XP)

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 3-19: Programming automatic reintegration

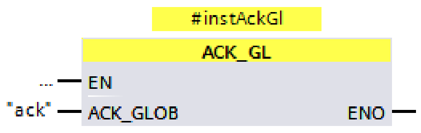


3.11.3 Manual reintegration

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:

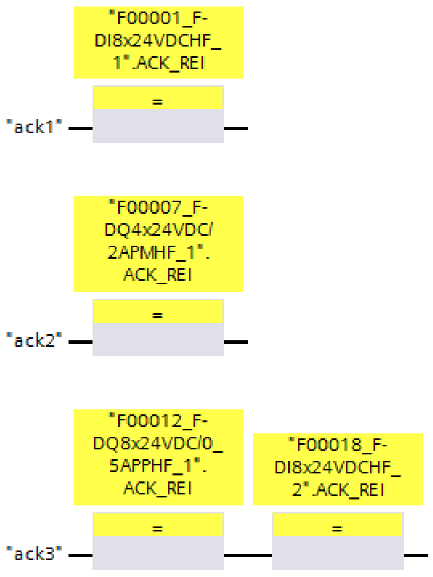
Figure 3-20: "ACK_GL" instruction



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 3-21: 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

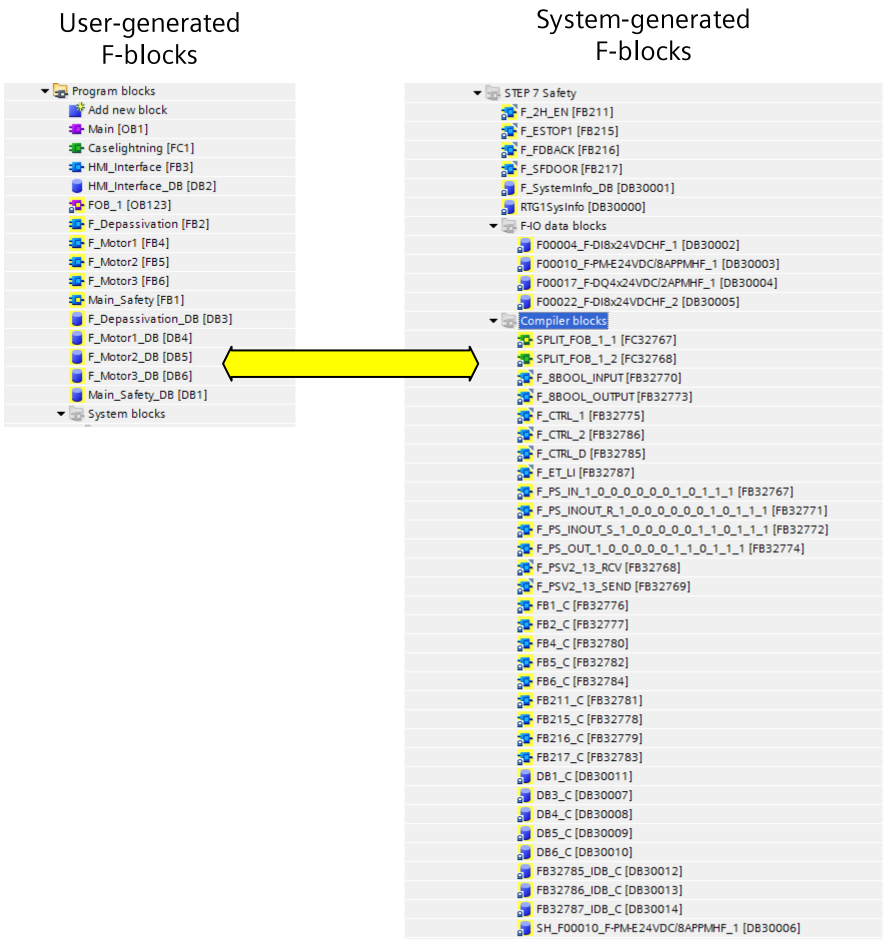
Introduction

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.

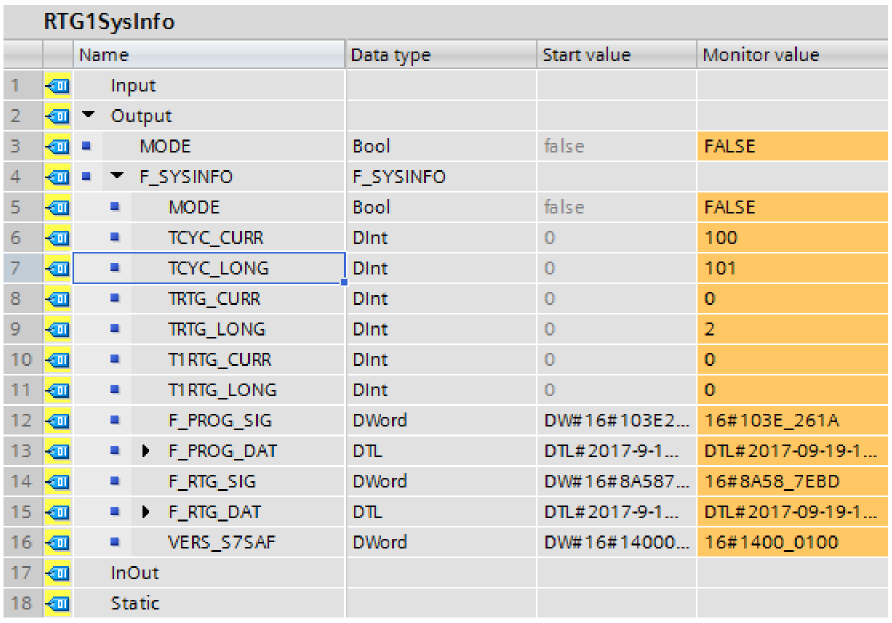
Note Depending on the application, it is not always possible to use all the suggestions. However, they show why certain programming methods cause shorter compilation and program runtimes than a non-optimized program.

Determining the 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.

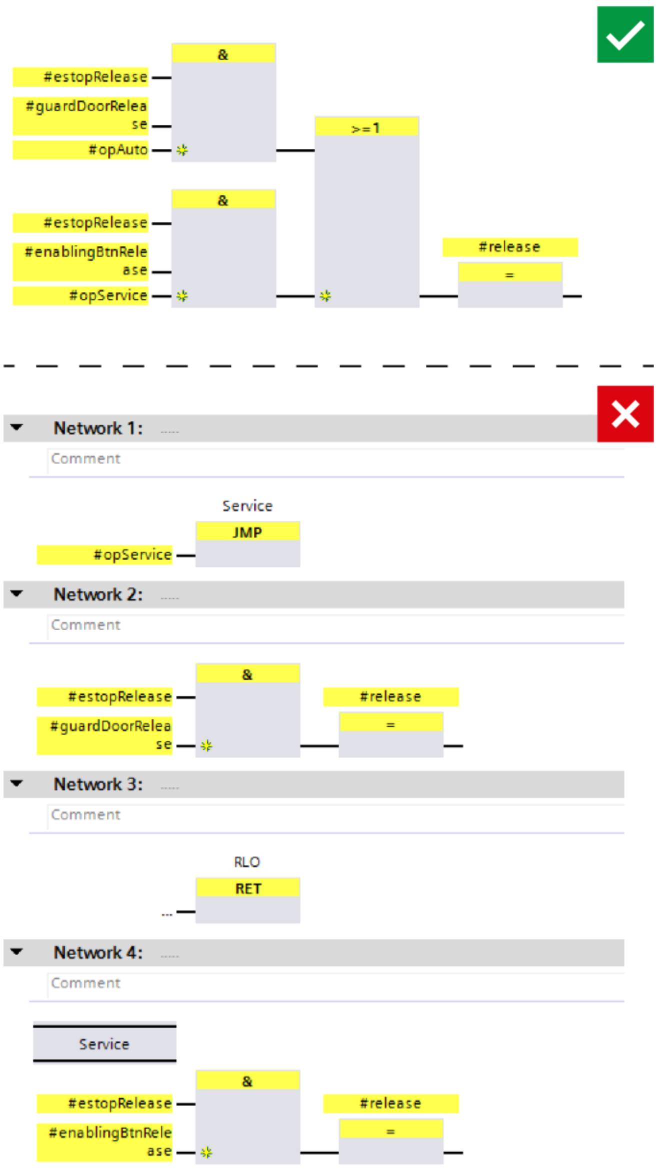
This requires that both alternatives (jump to label is "true" or "false") be calculated in their entirety in the protection program.

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

Recommendation

- Where possible, 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 single timer block.

Recommendation

Reduce the number of timer blocks to a minimum.

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.

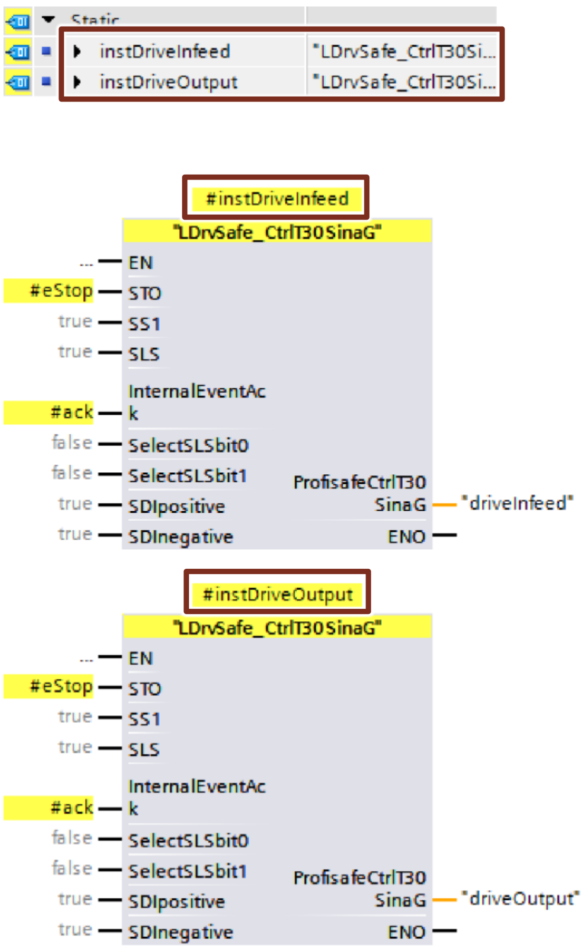
Advantages

- Standardization of safety programs:
  No global data is used for block tags. This allows reuse of the calling block (including the integrated blocks).
- Optimizing the compile performance:
  - Using multi-instances requires less protection code than using instance data blocks.
  - The use of multi-instances reduces the number of F-blocks on the system side. This optimizes the compilation of the safety program.

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, defect 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
- Use of clock memories
- Update of a partial PII by higher-priority alarms

For information about how to correctly program access from the standard user program to the safety program, see Chapter 3.9.

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 arithmetic functions. This must be caught by the user in the program. A library with arithmetic functions for the safety program is available in SIOS: https://support.industry.siemens.com/cs/ww/en/view/109482083	
Division by 0 If a division by 0 occurs in the safety program, the F-CPU goes to STOP. Prior to the division, the divisor must be checked for 0 and the division must be conditionally skipped.	
Access via HMI An HMI is used to write (modify) data (bit memories, DBs) that is read in the safety program. By default, communication has a higher priority than safety. This can result in data corruption. For possible solutions, see Chapter 3.9.	
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; can occur at runtime when there are unfavorable defaults for generating a pointer to F-areas (inputs, outputs, data blocks, etc.).	
Changing start values in F-instance data blocks Start values may only be changed in the interfaces of F-FBs.	
Read / write standard data Read / write access to the same standard data from the safety program is not allowed.	

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

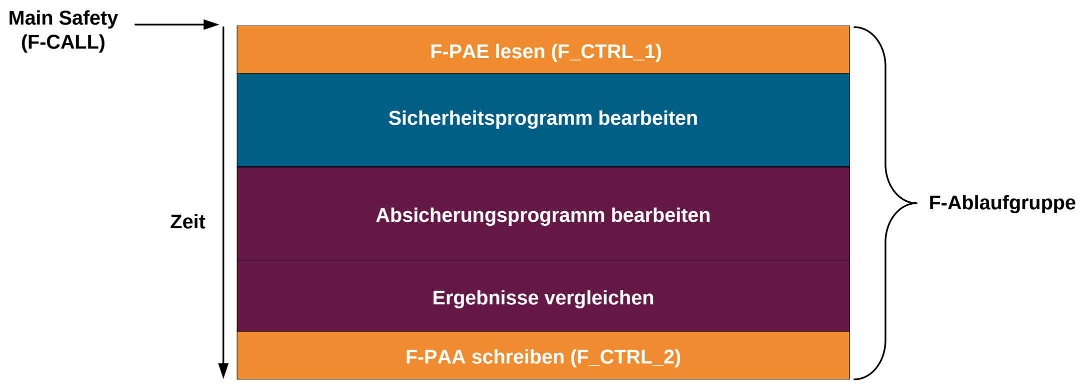
Coded processing

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 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. Then the F-CPU compares the results. If processed correctly, the safe outputs are written. If there are differences between the two program parts (e.g., due to data corruption), the F-CPU goes to stop and generates an entry in the diagnostic buffer.

Figure 5-1: Safety program processing sequence


Main Safety (F-CALL)
Zeit
F-PAE lesen (F_CTRL_1)
Sicherheitsprogramm bearbeiten
Absicherungsprogramm bearbeiten
Ergebnisse vergleichen
F-PAA schreiben (F_CTRL_2)
F-Ablaufgruppe

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.

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 squashed light plastic-sheathed cable. Without cross-circuit detection, this would result in, for example, a two-channel emergency stop circuit not tripping even though only one normally closed contact is faulty (secondary error).

RIOforFA

RIOforFA (Remote IO for Factory Automation) is a standard from the PROFIBUS & PROFINET International organization. Among other features, it describes the following functions:

- 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. For this purpose, they 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 do you need support?

With Industry Online Support, our complete service and support know-how and services are available to you 24/7.

Industry Online Support is the place to go to for information about our products, solutions and services.

Product Information, Manuals, Downloads, FAQs and Application Examples – all the information can be accessed with just a few clicks:
https://support.industry.siemens.com

Technical Support

Siemens Industry's Technical Support offers you fast and competent support for any technical queries you may have, including numerous tailor-made offerings ranging from basic support to custom support contracts.

You can use the web form below to send queries to Technical Support:
www.siemens.com/industry/supportrequest.

Service offer

Our service offer includes the following services:

- Product Training
- Plant Data Services
- Spare Part Services
- Repair Services
- Field & Maintenance Services
- Retrofit & Modernization Services
- Service Programs & Agreements

For detailed information about our service offer, please refer to the Service Catalog:
https://support.industry.siemens.com/cs/sc

Industry Online Support app

The "Siemens Industry Online Support" app provides you with optimum support while on the go. The app is available for Apple iOS, Android and Windows Phone:
https://support.industry.siemens.com/cs/ww/en/sc/2067

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 Styleguide 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 in Factory Automation" https://support.industry.siemens.com/cs/ww/en/view/109747812

6.3 Change documentation

Table 6-2: Change documentation

Version	Date	Modifications
V1.0	10/2017	First version