

Reference Application for CODESYS Controller Redundancy

Abstract:

This application note describes a reference application that extends a CODESYS controller redundancy system with additional mechanisms for monitoring and managing communication availability. It demonstrates how the communication path to Modbus TCP fieldbus devices can be monitored to trigger a controller switchover when required, and how a common IP address can be provided for higher-level systems such as SCADA or OPC UA clients. The reference application is intended as a basis for understanding, testing, and adapting these mechanisms to application-specific requirements.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UC20-M4000	2839160000	HW 01.00.00
2	IE-SW-AL05LM-5TX	2682250000	HW 01.00 FW 01.19
3	UR20-FBC-MOD-TCP-V2	2476450000	HW 01.05.00 FW 02.16.00

Software reference

No.	Software name	Article No.	Software version
4	CODESYS Development System		3.5 SP21 Patch 4 - 3.5.21.40
	CODESYS Control Runtime		4.21.0.0-1
	CODESYS Redundancy		4.3.0.0
	IoDrvModbusTCP		3.5.17.0
	RedundancyManager		3.5.22.20
	Reference Application		V2

Contact

Weidmüller Interface GmbH & Co. KG
Klingenbergstraße 26
32758 Detmold, Germany
www.weidmueller.com

For any further support please contact your
local sales representative:
<https://www.weidmueller.com/countries>

Content

1	Warning and Disclaimer.....	4
2	Introduction.....	5
3	System Architecture.....	6
3.1	System Overview.....	6
3.2	Network Topology.....	7
3.3	CODESYS Redundancy Principle (System Redundancy).....	8
3.4	Media Redundancy.....	9
3.5	Reference Application Functions	10
4	Reference Application Architecture.....	12
4.1	Application Structure.....	12
5	Local Modbus TCP Link Monitoring	13
5.1	Purpose and Monitored Failure Scenario.....	13
5.2	Functional Principle	15
5.3	Link Monitor State Machine	25
5.4	Function Blocks and Interfaces.....	35
5.5	Configuration and Parameters	42
5.6	Error Handling and Recovery.....	47
6	Common IP Address Management.....	52
6.1	Purpose and Monitored Failure Scenario.....	52
6.2	Functional Principle	56
6.3	IP Alias State Machine.....	58
6.4	Function Blocks and Configuration	66
6.5	Error Handling and Recovery.....	71

1 Warning and Disclaimer

Warning

Controls may fail in unsafe operating conditions, causing uncontrolled operation of the controlled devices. Such hazardous events can result in death and / or serious injury and / or property damage. Therefore, there must be safety equipment provided / electrical safety design or other redundant safety features that are independent from the automation system.

Disclaimer

This Application Note / Quick Start Guide / Example Program does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Each user is responsible for the correct operation of his control system. By using this Application Note / Quick Start Guide / Example Program prepared by Weidmüller, you accept that Weidmüller cannot be held liable for any damage to property and / or personal injury that may occur because of the use. The names *CODESYS Development System* is a registered trademark of CODESYS.

Note

The given descriptions and examples do not represent any customer-specific solutions, they are simply intended to help for typical tasks. The user is responsible for the proper operation of the described products. Application notes / Quick Start Guides / Example Programs are not binding and do not claim to be complete in terms of configuration as well as any contingencies. By using this Application Note / Quick Start Guide / Example Program, you acknowledge that we cannot be held liable for any damages beyond the described liability regime. We reserve the right to make changes to this application note / quick start guide / example at any time without notice. In case of discrepancies between the proposals Application Notes / Quick Start Guides / Program Examples and other Weidmüller publications, like manuals, such contents have always more priority to the examples. We assume no liability for the information contained in this document. Our liability, for whatever legal reason, for damages caused using the examples, instructions, programs, project planning and performance data, etc. described in this Application Note / Quick Start Guide / Example is excluded.

Security notes

In order to protect equipment, systems, machines and networks against cyber threats, it is necessary to implement (and maintain) a complete state-of-the-art industrial security concept. The customer is responsible for preventing unauthorized access to his equipment, systems, machines and networks. Systems, machines and components should only be connected to the corporate network or the Internet if necessary and appropriate safeguards (such as firewalls and network segmentation) have been taken.

2 Introduction

This application note describes the individual steps required to implement a redundant Ethernet network using **Rapid Spanning Tree Protocol (RSTP)** with Weidmüller managed Ethernet switches and **UR20-FBC-MOD-TCP-V2** Modbus TCP fieldbus couplers.

The redundant network presented in this application note is designed to increase the availability of Industrial Ethernet communication by automatically restoring the communication path in the event of a cable or network link failure. The document explains the required hardware topology and the configuration of both the managed Ethernet switches and the Modbus TCP fieldbus couplers to establish a fully operational RSTP network.

This application note focuses exclusively on the implementation of **media redundancy**. The configuration of **CODESYS Controller Redundancy** and **Modbus TCP client redundancy** is described in a separate application note. When combined, both concepts provide a highly available automation system that tolerates both controller failures and network failures.



Note:

The setup, configuration steps and test results described in this application note have been validated exclusively using the hardware and software components listed in the **Hardware Reference** and **Software Reference** sections of this document.

The use of different hardware, firmware versions, software versions, network infrastructure components, or additional third-party devices may influence the behavior of the system and could lead to results that differ from those described herein. Therefore, any deviations from the documented setup must be evaluated and validated individually.

3 System Architecture

3.1 System Overview

The system presented in this application note demonstrates a redundant controller setup using two UC20 controllers and distributed Modbus TCP I/O. The objective is to maintain application execution and communication to the fieldbus level in defined failure scenarios.

The controller redundancy is provided by the CODESYS Redundancy Configuration functionality. In normal operation, one controller operates as an active PLC, while the second controller operates as the standby PLC. Both controllers execute the same application and are synchronized via a dedicated redundancy link. If the active controller becomes unavailable, the standby controller can take over application execution at runtime.

The distributed I/O is connected via two UR20-FBC-MOD-TCP-V2 Modbus TCP fieldbus couplers. For Modbus TCP communication in a redundant controller setup, the CODESYS Modbus TCP Client redundancy mechanism is used. This allows the Modbus TCP communication to be managed according to the current redundancy state of the controllers.

The fieldbus network is built as a redundant Ethernet topology using two managed Ethernet switches. Rapid Spanning Tree Protocol (RSTP) is used to prevent network loops during normal operation and to provide an alternative communication path in case of a cable interruption.

The overall setup combines controller redundancy and network redundancy. Controller redundancy ensures that the application can continue running on the second controller if the active controller fails. Network redundancy ensures that communication to the Modbus TCP fieldbus couplers can continue if one Ethernet connection in the fieldbus network is interrupted.

3.2 Network Topology

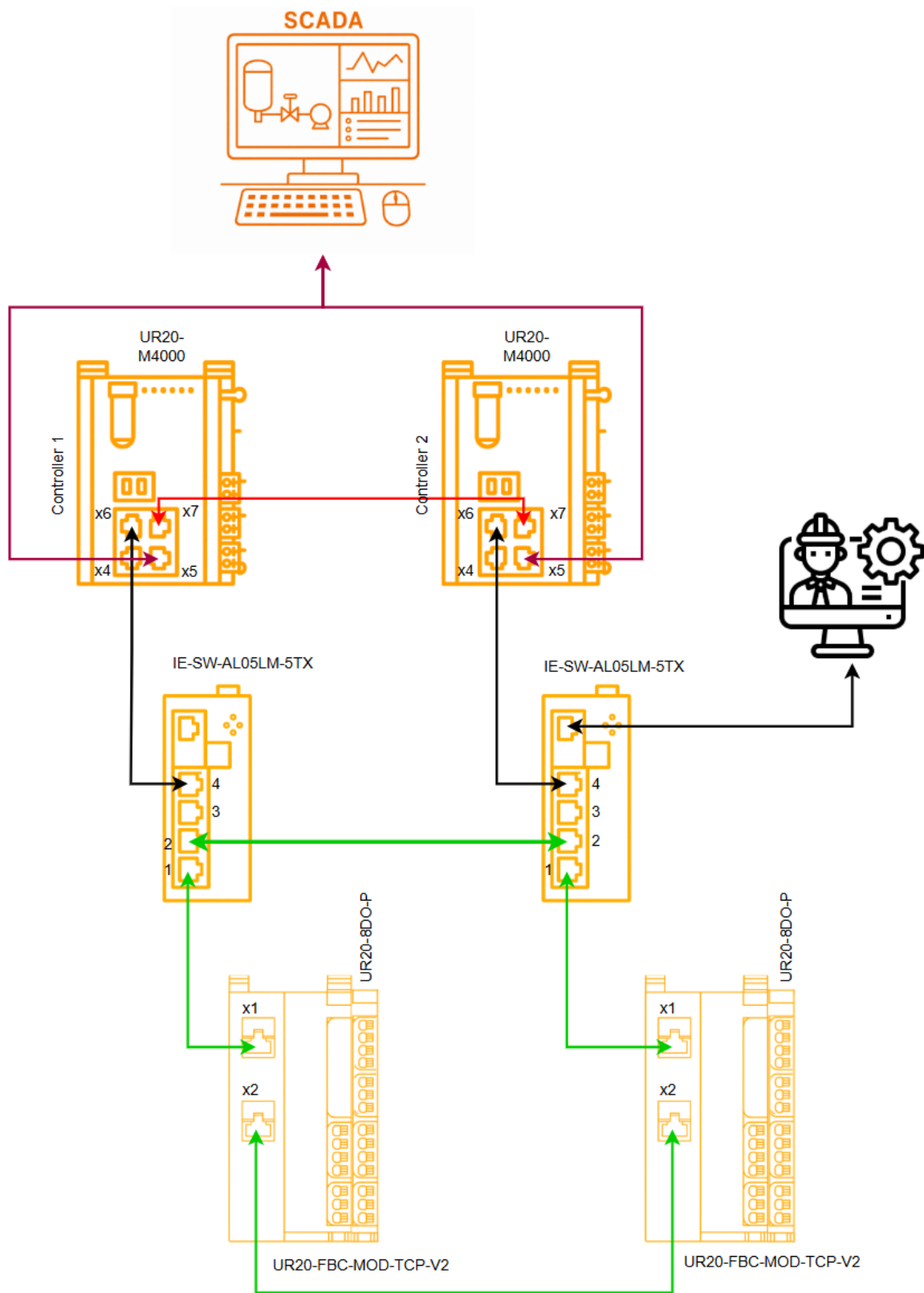


Figure 1: Network Topology

Symbol	Description
	CODESYS Redundancy Link – Synchronization channel between the primary and secondary UC20 controller. Used for redundancy management and data synchronization.
	Controller Ethernet Connection – Ethernet connection between the UC20 controller and the local managed switch.
	Modbus TCP Network (RSTP Ring) – Redundant Ethernet network used for communication between the controllers and the UR20 Modbus TCP fieldbus couplers. Rapid Spanning Tree Protocol (RSTP) provides automatic path redundancy in case of a network failure.
	SCADA / OPC UA Communication – Logical communication path between the SCADA system and the redundant controller pair. A common IP address is dynamically provided by the controller responsible for the service, allowing the SCADA system to access the application through a single IP address independent of which controller is currently active.

3.3 CODESYS Redundancy Principle (System Redundancy)

The controller redundancy used in this reference application is based on the CODESYS Redundancy Configuration. Two UC20 controllers execute the same application while one controller operates as the active PLC and the second controller operates as the standby PLC. Application data is synchronized between both controllers via a dedicated redundancy link.

If the active controller becomes unavailable, the standby controller can take over application execution. The current redundancy state is also available to the application and is used by the reference functions described in this application note to coordinate communication monitoring and service availability.



Note:

The setup and configuration of CODESYS Controller Redundancy and redundant Modbus TCP communication are not part of this application note. These functions must be configured beforehand as described in **AN0141 – Setting Up CODESYS Controller Redundancy**.

3.4 Media Redundancy

The fieldbus network shown in Chapter 3.2 uses Rapid Spanning Tree Protocol (RSTP) to provide an alternative communication path between the controllers and the Modbus TCP fieldbus couplers in case of a network link failure.

Media redundancy operates independently from CODESYS Controller Redundancy. While controller redundancy maintains application execution after a controller failure, media redundancy maintains the Ethernet communication path after a network failure.

The reference application described in this document operates on top of this redundant network infrastructure and assumes that the RSTP network has already been configured and verified.



Note:

The configuration and verification of the RSTP network are described separately in **AN0145 - Setting Up Media Redundancy using RSTP**.

3.5 Reference Application Functions

The CODESYS redundancy mechanisms described in the previous chapters provide the basic functionality for controller synchronization, controller switchover and redundant network communication. The reference application extends this setup with additional application-level functions to improve communication availability and simplify the integration of higher-level systems.

The reference application provides two main functions:

Local Modbus TCP Link Monitoring

The Local Modbus TCP Link Monitoring function monitors the communication between the active controller and the Modbus TCP fieldbus couplers.

If the active controller loses communication with the fieldbus couplers, the standby controller verifies whether the devices are still reachable through its own network connection. If a valid communication path is available from the standby controller, the application can initiate a controller switchover.

This mechanism addresses failure scenarios in which the active controller itself remains operational but has lost its communication path to the fieldbus devices.

IPAliasManagement

The Common IP Address Management function provides a single logical IP address for communication with higher-level systems such as SCADA or OPC UA clients.

The common IP address is managed according to the current redundancy state and is provided by the controller responsible for the service. After a controller switchover, the common IP address is transferred accordingly.

This allows higher-level systems to communicate with the redundant controller pair through a single IP address without requiring knowledge of which physical controller is currently active.

Reference Function	Purpose
Local Modbus TCP Link Monitoring	Monitors fieldbus communication and allows a controller switchover to be initiated if the active controller loses access to the fieldbus devices while they remain reachable from the standby controller.
IPAliasManagement	Provides a single logical IP address for SCADA or OPC UA communication independent of which controller currently provides the service.



Note:

These functions extend the basic redundancy mechanisms and are implemented at application level. They are provided as reference implementations and may need to be adapted and validated according to the requirements of the target application.

4 Reference Application Architecture

4.1 Application Structure

The reference application implements the functions introduced in Chapter 3.5 as separate application-level components running on both controllers of the redundant PLC pair.

The implementation uses the CODESYS redundancy system as its underlying source of information about the local PLC identity and the current redundancy state. Based on this information, the individual application components execute different actions depending on whether the local controller is operating as Active, Standby or in another redundancy state.

The main application components are:

PRG_LocalMTSLinkMonitor

Implements the Local Modbus TCP Link Monitoring function. The program monitors the availability of the Modbus TCP communication path and, if required, verifies whether the Standby controller provides an alternative path to the fieldbus devices. Based on this information, a controller switchover can be requested. The implementation is described in Chapter 5.

PRG_IPAliasManagement

Implements the Common IP Address Management function. The program manages the Common IP according to the current redundancy state and the physical controller identity. In Standalone operation, additional reachability checks are used to prevent both controllers from providing the Common IP simultaneously. The implementation is described in Chapter 6.

Both programs are executed locally on each controller. Their behavior is determined dynamically from the local CODESYS redundancy state. This allows the same application to be deployed on both controllers while the individual program instances perform different functions according to the current redundancy role.

5 Local Modbus TCP Link Monitoring

5.1 Purpose and Monitored Failure Scenario

CODESYS Redundancy provides controller redundancy by operating two PLCs as an Active/Standby pair. If the Active PLC fails, the Standby PLC can take over application execution. However, the controller itself may remain operational while its communication path to the Modbus TCP fieldbus devices is no longer available.

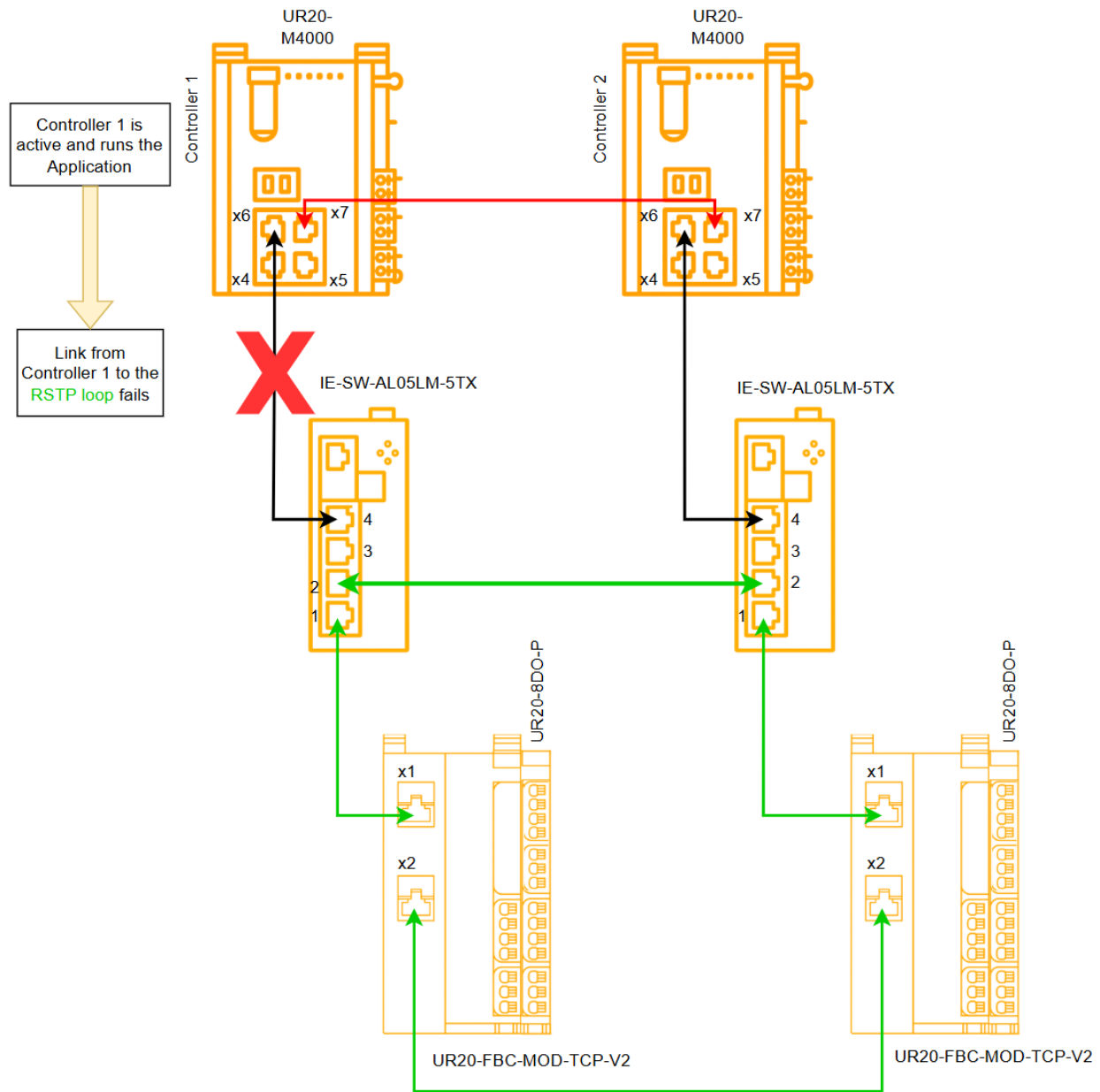
Such a failure is particularly relevant in a redundant network architecture. The Active PLC may still be running and remain connected to the Standby PLC via the CODESYS Redundancy Link, while its local Modbus TCP communication is no longer operational. At the same time, the Standby PLC may still have a valid communication path to the fieldbus devices.

The **Local Modbus TCP Link Monitoring** function extends the standard redundancy behavior to cover this failure scenario. It monitors the Modbus TCP communication status on the Active PLC and makes this information available to the Standby PLC.

If the Active PLC reports that both monitored Modbus TCP connections are unavailable, the Standby PLC does not immediately initiate a controller switchover. Instead, it first verifies whether the fieldbus devices are reachable through its own network connection.

Reference Application for CODESYS Controller Redundancy

The following figure illustrates the monitored failure scenario.



Link from Controller 1 to the RSTP loop fails

If at least one of the monitored fieldbus devices can be reached from the Standby PLC, the reference application requests that the Standby PLC becomes Active. If the devices cannot be reached from the Standby PLC either, no switchover is requested and the reachability check is repeated after a defined retry time.

This additional verification prevents a controller switchover when the Standby PLC would not provide a better communication path to the fieldbus devices.



Note:

Local Modbus TCP Link Monitoring complements the CODESYS controller redundancy and RSTP media redundancy mechanisms. It does not replace either mechanism.

This additional monitoring is required due to the topology of the reference system. The controllers do not provide integrated switching or RSTP functionality and are each connected to the redundant RSTP network through a single local Ethernet connection. Consequently, a failure of the connection between a controller and its local managed switch cannot be compensated by RSTP for that controller, even though the remaining RSTP network may still be fully operational.

If this failure occurs on the Active controller, the controller itself and the CODESYS Redundancy Link may remain operational while access to the Modbus TCP fieldbus devices is lost. The Local Modbus TCP Link Monitoring function detects this condition and allows a switchover to be initiated when the Standby controller still provides a usable communication path to the fieldbus devices.

5.2 Functional Principle

The **Local Modbus TCP Link Monitoring** function is implemented as a local monitoring mechanism on both redundant controllers. Its purpose is to determine whether the controller currently operating as Active still has a usable communication path to the Modbus TCP fieldbus devices and, if not, whether the Standby controller can provide an alternative communication path.

The monitoring logic is implemented in **PRG_LocalMTSLinkMonitor**. The program is executed cyclically in the **NonReduConfig** task with a cycle time of **20 ms**. This task is not configured as the CODESYS Redundancy Task. Consequently, the internal state of **PRG_LocalMTSLinkMonitor**, including its state machine and function block instances, is maintained locally on each controller and is not synchronized between the two PLCs.

This behavior is required because the two instances of the Link Monitor perform different functions depending on the current redundancy role:

- On the **Active controller**, the Link Monitor monitors the local Modbus TCP communication.
- On the **Standby controller**, the Link Monitor evaluates the communication status reported by the Active controller and performs an independent reachability check if a communication failure is detected.

NonReduConfig X

Configuration

Priority (0..31): 2 Task group IEC-Tasks

Type: Cyclic Interval (e.g. t#200ms) 20

Watchdog: ☐ Enable Time (e.g. t#200ms) Sensitivity 1

+ Add Call X Remove Call Change Call Move Up Move Down Open POU

POU	Comment
PRG_IPAliasManagement	
PRG_LocalMTSLinkMonitor	

CODESYS task configuration - PRG_LocalMTSLinkMonitor assigned to NonReduConfig

Redundanzkonfiguration X

Redundancy State
Redundancy Settings
Registered Areas
Applications
Files
Log
PLC Shell

Object	Update behaviour
Device.Application.PRG_ModbusManagement	☐ Cyclic update
Device.Application.PRG_Init_Modbus_Config	☐ Cyclic update
☒ Device.Application.PRG_General_Synced	☒ Cyclic update
Device.Application.PRG_IPAliasManagement	☐ Cyclic update
☐ Device.Application.PRG_LocalMTSLinkMonitor	☐ Cyclic update
☒ Device.Application.GVL_ModbusIP_Config	☒ Cyclic update
☒ Device.Application.GVL_General	☒ Cyclic update
☒ Device.Application.GVL_ModbusIP_CH_CR_ProcessValues	☒ Cyclic update
☐ Device.Application.PersistentVars	☐ Cyclic update
☒ Device.Application.GVL_IPAlias	☒ Cyclic update
☒ Device.Application.GVL_IPAliasStatus	☒ Cyclic update
☒ Device.Application.GVL_IpMTSLinkMonitoring	☒ Cyclic update

Registered Areas - GVL_IpMTSLinkMonitoring enabled for cyclic synchronization

Only information required for coordination between the two local monitoring instances is synchronized through the CODESYS redundancy mechanism. In particular, **GVL_IpMTSLinkMonitoring** is included in the registered redundancy areas and therefore provides the status information required by both controllers.



Note:

The execution of **PRG_LocalMTSLinkMonitor** and the synchronization of its coordination data are intentionally separated. The program itself operates locally on each controller, while selected data in **GVL_IpMTSLinkMonitoring** is synchronized

between the redundant controllers. This allows the Active and Standby controllers to maintain different local monitoring states while exchanging the information required for a coordinated switchover decision.

Redundancy-Dependent Monitoring

Each local instance of **PRG_LocalMTSLinkMonitor** retrieves the current CODESYS redundancy state using **GetRedundancyState()**.

The current state is available through:

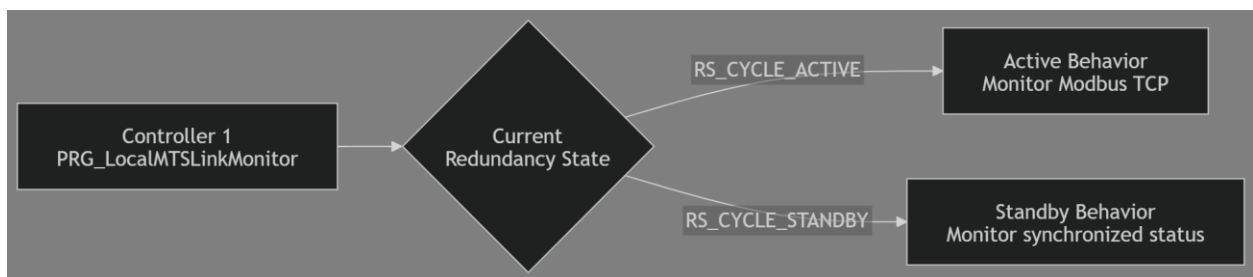
rsRedundancyState.eRedundancyState

The Link Monitor uses this information to determine which monitoring function must be performed locally.

The two primary operating states relevant to normal redundant operation are:

CODESYS Redundancy State	Local Link Monitor Function
RS_CYCLE_ACTIVE	Monitor the local Modbus TCP communication and provide the resulting status to the Standby controller.
RS_CYCLE_STANDBY	Evaluate the synchronized communication status of the Active controller and initiate a local reachability check if required.

This results in two independent instances of the same monitoring program operating with different responsibilities:



The same principle applies to the second controller. After a redundancy role change, the local Link Monitor detects the changed redundancy state and adapts its behavior accordingly.

The controller identity itself is independent of this role. **GetPlcIdent()** is therefore used to identify whether the local device is PLC 1 or PLC 2, whereas **GetRedundancyState()** determines whether that PLC currently operates as Active or Standby.

Monitoring on the Active Controller

When the local controller is in **RS_CYCLE_ACTIVE**, the Link Monitor evaluates the communication state of the two configured Modbus TCP connections.

The individual communication states are combined into the global status:

GVL_IpMTSLinkMonitoring.xMTSHandlerOK

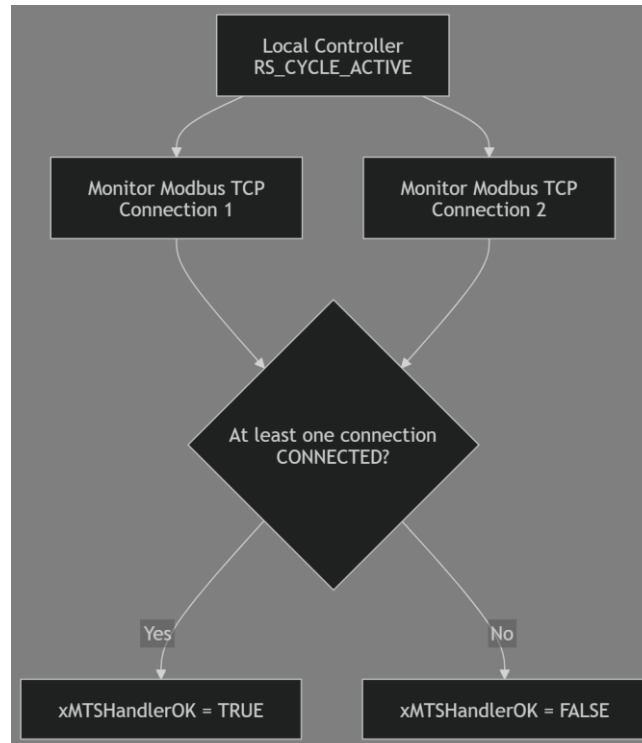
The reference application considers the Modbus TCP communication available as long as **at least one of the two monitored connections remains connected**.

The resulting logic is:

Modbus TCP Connection 1	Modbus TCP Connection 2	xMTSHandlerOK
Connected	Connected	TRUE
Connected	Not connected	TRUE
Not connected	Connected	TRUE
Not connected	Not connected	FALSE

Therefore, the failure of a single monitored Modbus TCP connection does not cause the Link Monitor to initiate the additional failover procedure. The monitored condition is reached only when **both connections are unavailable**.

Conceptually, the Active-side monitoring can be represented as follows:



Because this GVL is included in the registered redundancy areas, the status becomes available to the other controller through the CODESYS redundancy synchronization mechanism.

Status Exchange Between Active and Standby

The synchronization of `GVL_IpMTSLinkMonitoring` provides the communication channel required between the two local instances of `PRG_LocalMTSLinkMonitor`.

Conceptually, the information flow is:



The important distinction is that **the monitoring program itself is not synchronized**. Only the information required by the other controller is synchronized.

For example, local variables and instances such as the following remain controller-specific:

- `etLinkMonitorState`
- `etFSMAction`
- `fbIsIpExtern1`
- `fbIsIpExtern2`
- `tonRetryMTSPing`

This allows one controller to execute the **active** monitoring state while the other controller simultaneously executes the **passive** monitoring state.

Local Reachability Check on the Standby Controller

The local reachability check is implemented using two instances of **FB_IsIpExtern**:

fbIsIpExtern1

fbIsIpExtern2

Each instance is configured with the IP address of one of the monitored Modbus TCP fieldbus devices. The parameter structure **ST_IsIpExternParams** contains:

Parameter	Description
arbIPtoTest	IPv4 address of the fieldbus device to be checked
udiPingTimeout	Maximum time to wait for the ping response

In the reference application, the ping timeout is defined by:

c_udiPingTimeout := 1000 ms

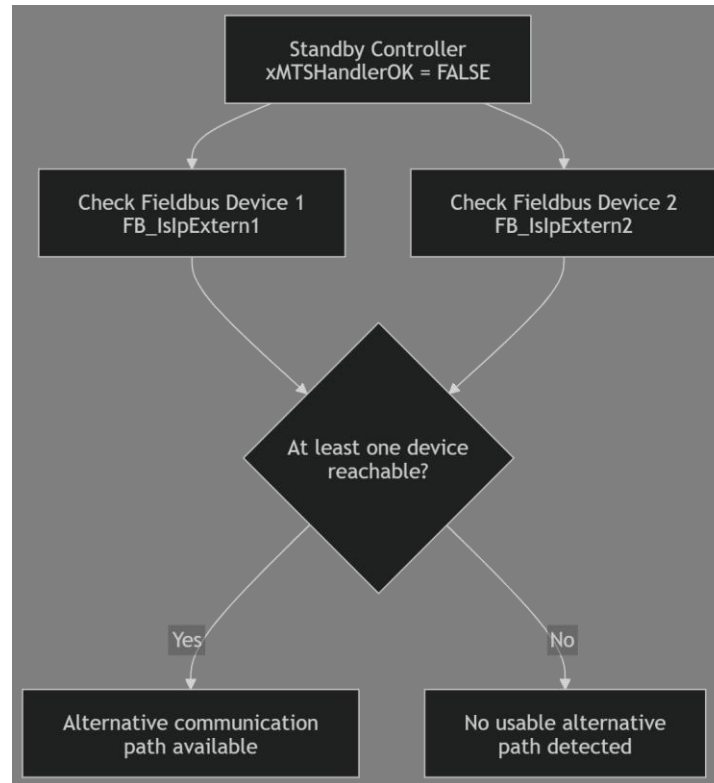
FB_IsIpExtern performs a ping-based reachability check and additionally verifies that the responding IP address is not a local IP address of the controller.

The result is provided through:

q_xIpIsExtern

A **TRUE** result therefore indicates that the configured external device can be reached from the local controller.

The Standby controller evaluates both reachability checks. At least one of the monitored devices must be reachable before a switchover is requested.



This check is performed locally by the Standby controller and therefore verifies its **own communication path** rather than relying solely on information received from the Active controller.

Switchover Decision

A controller switchover is requested only if both of the following conditions are fulfilled:

1. The Active controller reports that both monitored Modbus TCP connections are unavailable.
2. The Standby controller can reach at least one of the monitored fieldbus devices through its own network connection.

If both conditions are fulfilled, the Standby controller calls:

RDCY.SwitchToActive()

SwitchToActive() requests a transition of the local PLC from **RS_CYCLE_STANDBY** to **RS_CYCLE_ACTIVE**.

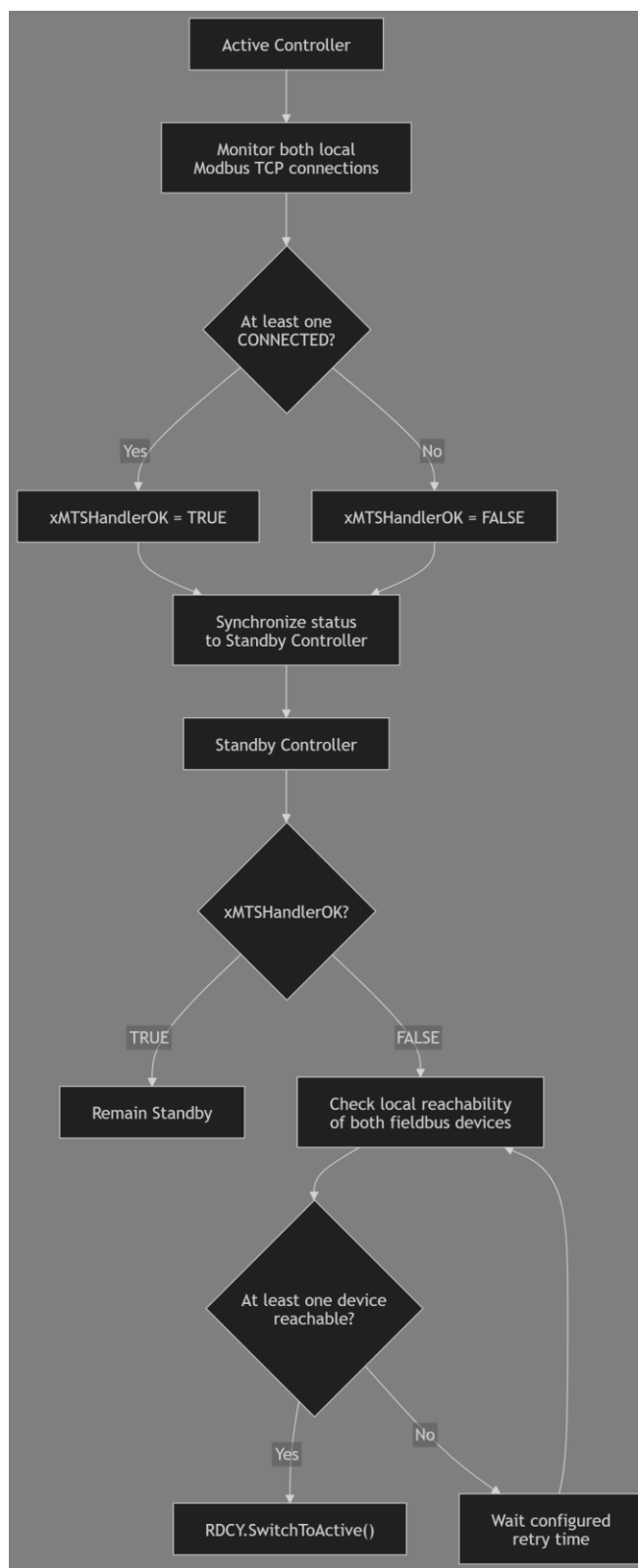
The application itself does not directly assign the new redundancy state. The actual state transition remains under control of the CODESYS redundancy system.

If neither fieldbus device can be reached from the Standby controller, no switchover is requested. Instead, the application waits for the configured retry interval:

c_tRetryMTSPing := T#1s

and subsequently repeats the reachability check.

The decision principle can therefore be summarized as follows:



Functional decision sequence of the Local Modbus TCP Link Monitoring

The additional reachability check prevents an unnecessary role change if the communication problem is not limited to the Active controller's local network path. A switchover is therefore only requested when the Standby controller has evidence that it can provide a usable alternative path to the fieldbus network.

**Note:**

FB_IsIpExtern verifies IP reachability and does not perform a complete Modbus TCP functional check. A successful result confirms that the configured external IP address is reachable from the Standby controller; it does not by itself confirm that the Modbus TCP application protocol is operating correctly.

5.3 Link Monitor State Machine

The behavior of PRG_LocalMTSLinkMonitor is implemented as a finite state machine. The state machine allows each controller to execute different monitoring actions depending on its current CODESYS redundancy state and the availability of the monitored Modbus TCP communication.

The current state of the Link Monitor is stored in:

etLinkMonitorState : ET_LinkMonitorState

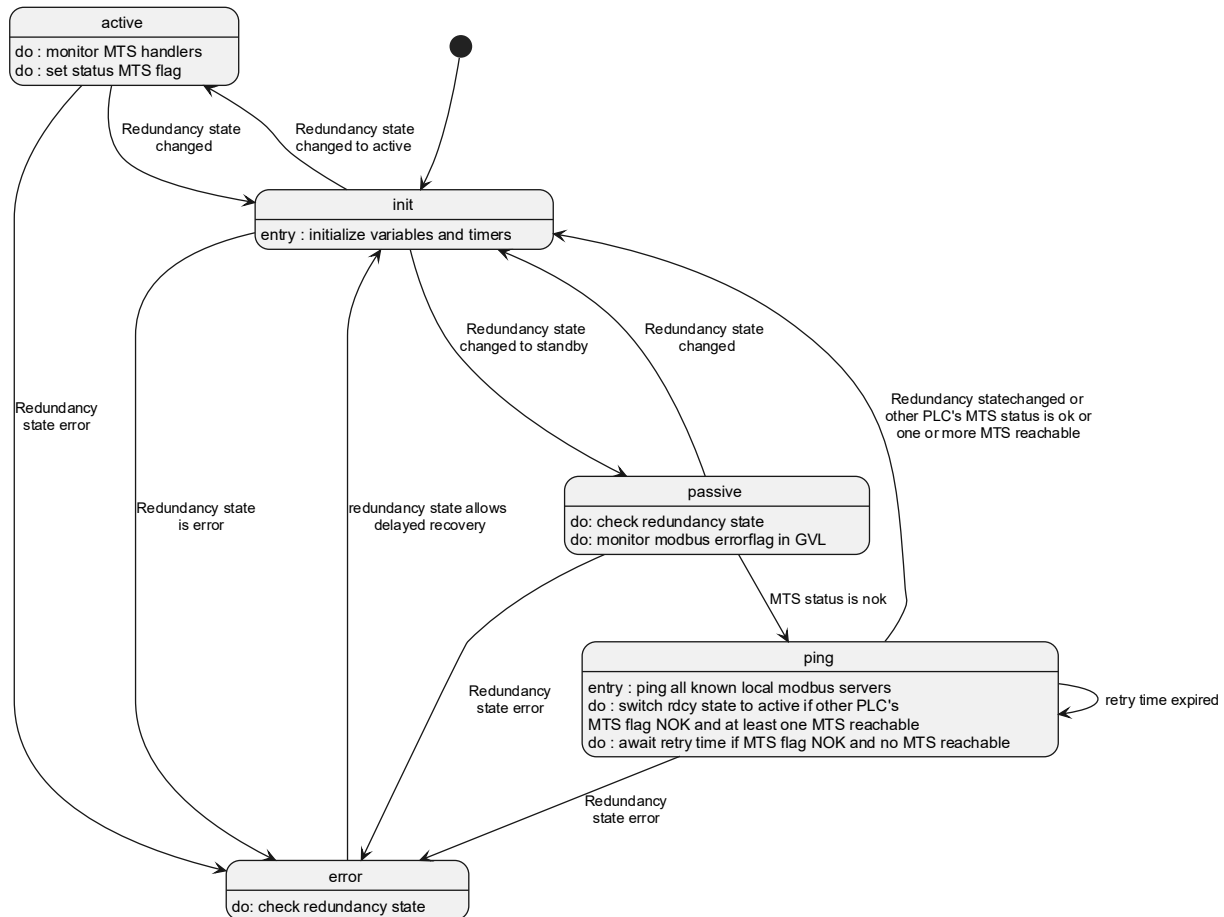
Each state is additionally divided into the actions **_entry**, **_do** and **_exit** using **ET_FSMAction**. This separates initialization activities from cyclic processing and state transitions.

The following states are defined for the Link Monitor:

State	Purpose
init	Initializes the Link Monitor and determines the required operating state based on the current CODESYS redundancy state.
active	Monitors the local Modbus TCP communication while the local PLC operates as Active.
passive	Monitors the synchronized Modbus TCP status reported by the Active controller while the local PLC operates as Standby.
ping	Verifies whether the monitored fieldbus devices are reachable through the local network connection of the Standby controller.
switch_redcy_state	Defined in the state enumeration but not used as a separate state in the current reference implementation.
error	Stores diagnostic information and handles error recovery.

State Machine Overview

The normal operating sequence of the Link Monitor is shown below.



State machine of PRG_LocalMTSLinkMonitor

The state machine continuously reacts to changes of the CODESYS redundancy state. A controller that changes from Standby to Active, or vice versa, first returns to **init**. This ensures that the local monitoring function is reinitialized according to the new redundancy role.

State Action Concept

Each Link Monitor state uses a second state variable:

etFSMAction : ET_FSMAction

The following actions are available:

Action	Purpose
_entry	Executes once when entering a state and initializes the functions required by that state.
_do	Contains the cyclic processing that is executed while the Link Monitor remains in the state.
_exit	Performs actions required before leaving a state, where required.

This structure prevents initialization actions from being executed repeatedly during every task cycle.

For example, when entering the **ping** state, the two reachability checks are activated during **_entry**. Their results are subsequently evaluated during **_do**.

Init State

The **init** state prepares the Local Modbus TCP Link Monitor for operation and determines which monitoring behavior is required on the local controller.

During **_entry**, the program:

- resets internal status and error flags,
- resets the retry and automatic error recovery timers,
- retrieves the local PLC identity using **GetPlcIdent()**,
- verifies that the PLC identity is within the expected range,
- copies the configured Modbus TCP fieldbus device IP addresses into the parameter structures of **FB_IsIpExtern**,
- configures the ping timeout,
- enables both **FB_IsIpExtern** instances, and
- leaves their execution inputs inactive until a reachability check is required.

The program then waits until both **FB_IsIpExtern** instances have reached their Standby state.

After initialization has completed, the current CODESYS redundancy state determines the next Link Monitor state:

CODESYS State	Link Monitor Behavior
RS_CYCLE_ACTIVE	Transition to active
RS_CYCLE_STANDBY	Transition to passive
RS_CYCLE_STANDALONE	Remain in init and wait
Redundancy error state	Transition to error

The **RS_CYCLE_STANDALONE** state is therefore not treated as an Active controller state by the Link Monitor. The program remains in **init** until the CODESYS redundancy state changes.

Active State

The **active** state is entered when the local PLC operates in **RS_CYCLE_ACTIVE**.

In this state, the Link Monitor continuously evaluates the communication states of the two configured Modbus TCP connections.

The result is written to:

GVL_IpMTSLinkMonitoring.xMTSHandlerOK

The status remains **TRUE** as long as at least one of the two monitored Modbus TCP connections is in the **CONNECTED** state.

Only when both connections are no longer connected is:

xMTSHandlerOK := FALSE

The synchronized status therefore informs the Standby controller that the Active controller has lost both monitored Modbus TCP communication paths.

If the CODESYS redundancy state changes from **RS_CYCLE_ACTIVE** to **RS_CYCLE_STANDBY** or **RS_CYCLE_STANDALONE**, the Link Monitor returns to **init** and determines its new local monitoring behavior.

A redundancy error causes a transition to the **error** state.

Passive State

The passive state is used while the local PLC operates in **RS_CYCLE_STANDBY**.

The Standby controller does not continuously ping the fieldbus devices. Instead, it initially monitors the synchronized status:

GVL_IpMTSLinkMonitoring.xMTSHandlerOK

As long as the value remains TRUE, no additional action is required.

If the Active controller sets:

xMTSHandlerOK := FALSE

the local Link Monitor transitions from **passive** to **ping**.

This two-stage approach avoids unnecessary network reachability checks during normal operation. A local ping is only initiated after the Active controller has reported the loss of both monitored Modbus TCP connections.

If the local redundancy state changes to Active or Standalone while the controller is in **passive**, the Link Monitor returns to **init**.

Ping State

The ping state performs the independent fieldbus reachability check from the Standby controller.

During **_entry**, both instances of **FB_IsIpExtern** are activated:

- **fbIsIpExtern1** checks the first configured fieldbus device.
- **fbIsIpExtern2** checks the second configured fieldbus device.

The reachability checks are only started if the local PLC is still in **RS_CYCLE_STANDBY**.

During **_do**, the Link Monitor first verifies whether the conditions that caused the reachability check are still valid.

If the Active controller has recovered its Modbus TCP communication and the synchronized value returns to:

xMTSHand1erOK = TRUE

the check is aborted and the Link Monitor returns to **init**.

If both **FB_IsIpExtern** instances have completed their checks, the results are evaluated.

At Least One Fieldbus Device Is Reachable

If either:

fbIsIpExtern1.q_xIpIsExtern = TRUE

or:

fbIsIpExtern2.q_xIpIsExtern = TRUE

the Standby controller has confirmed that at least one monitored fieldbus device is reachable through its own local network connection.

The application then calls:

RDCY.SwitchToActive()

This requests that the local Standby PLC becomes Active.

After requesting the switchover, the Link Monitor returns to **init**. The actual redundancy state transition remains controlled by CODESYS Redundancy.

No Fieldbus Device Is Reachable

If neither fieldbus device is reachable, a controller switchover would not provide an improved communication path.

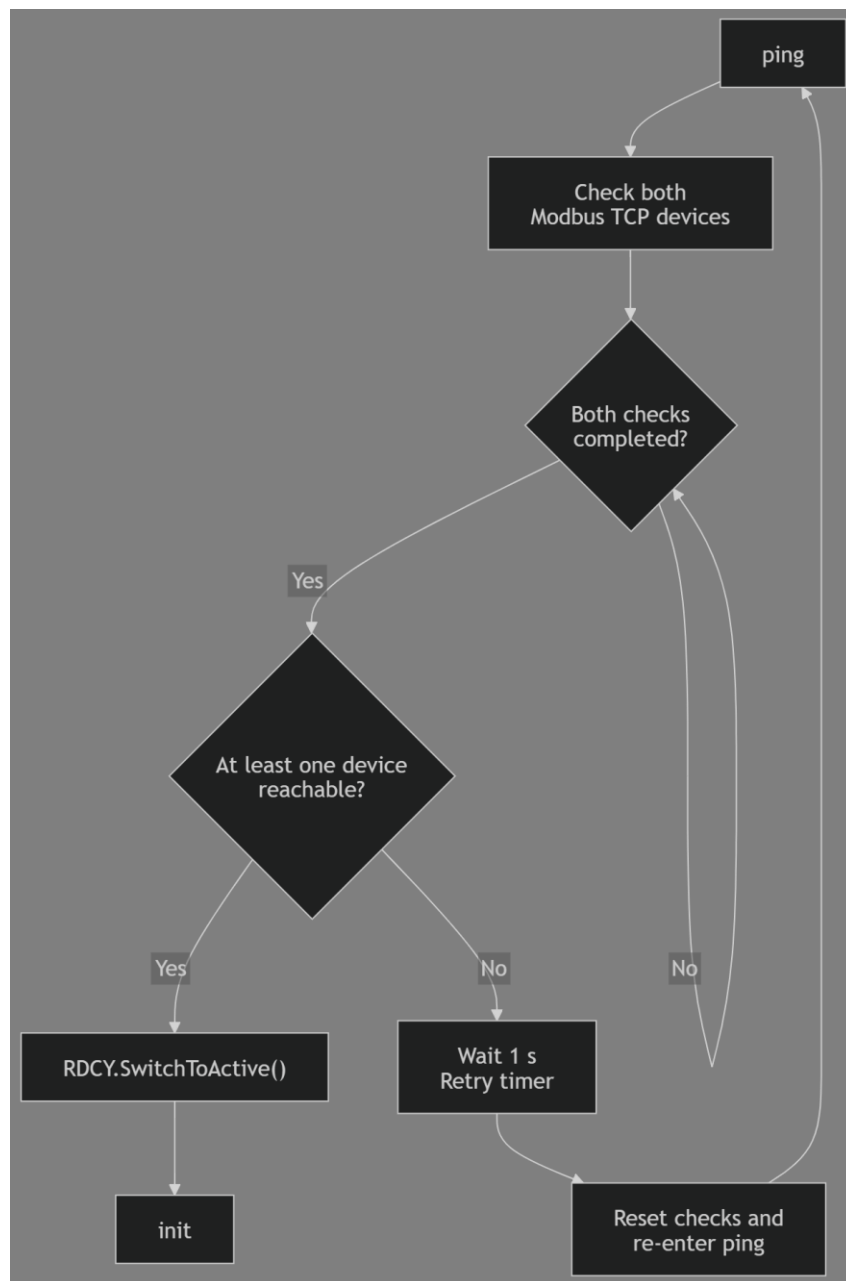
No switchover is therefore requested.

Instead, the retry timer is started. After:

c_tRetryMTSPing := T#1s

has elapsed, the previous reachability check is reset and the Link Monitor starts a new **ping** cycle.

The resulting behavior is:



Reachability check and retry sequence in the ping state

This prevents repeated controller switchovers when the communication failure affects both controllers or the fieldbus network itself.

Error State

The **error** state handles errors detected by the Link Monitor, its function blocks or the CODESYS redundancy system.

When entering the state, the program stores diagnostic information including:

- the Link Monitor state in which the error occurred,
- the current FSM action,
- the CODESYS redundancy state at the time of the error, and
- detailed error information provided by the affected function block.

The diagnostic information is made available through **GVL_IpMTSLinkMonitoring**.

Errors originating from **FB_IsIpExtern** are copied into the common error structure before the corresponding function block is disabled.

If the error was caused by a CODESYS redundancy state, the application evaluates whether automatic recovery is possible.

For redundancy states that permit recovery, an automatic recovery timer is started. The reference application uses:

c_tErrorAutoAck := T#10S

After the timer has elapsed, the error is acknowledged and the Link Monitor is reinitialized.

If automatic recovery is not permitted for the current redundancy state, the Link Monitor remains in the error condition until an appropriate recovery action is performed.

After successful acknowledgment:

- the stored error information is cleared,
- the **FB_IsIpExtern** instances are disabled and reset, and
- the state machine returns to **init**.

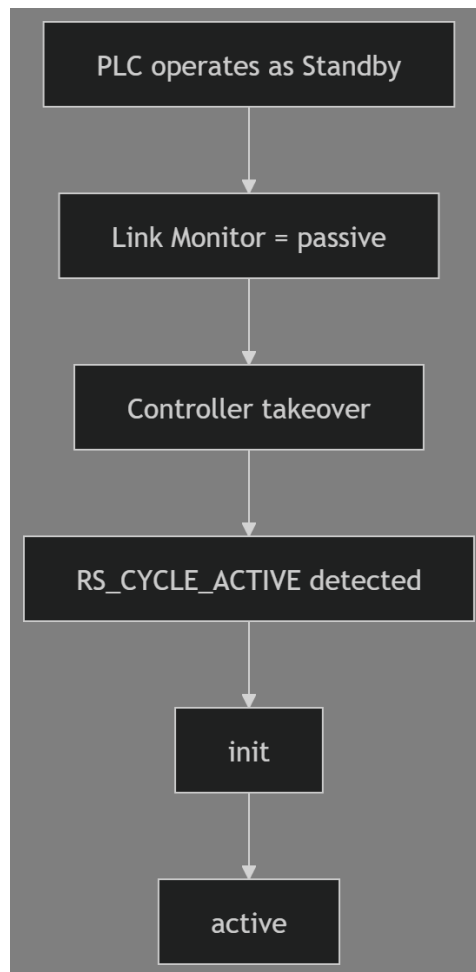
Handling of Redundancy State Changes

A central design principle of the Link Monitor is that the local monitoring behavior always follows the current CODESYS redundancy role.

The redundancy state is therefore read cyclically using **GetRedundancyState()** and compared with the previously detected state.

If the role changes during normal operation, the current monitoring sequence is terminated and the state machine returns to **init**.

For example:



Link Monitor state transition after controller takeover

This ensures that the same instance of PRG_LocalMTSLinkMonitor automatically changes its behavior after a controller switchover without requiring a restart of the application.

5.4 Function Blocks and Interfaces

The Local Modbus TCP Link Monitor uses several function blocks and functions to interact with the CODESYS redundancy system, manage the redundant Modbus TCP communication, and verify the local reachability of the fieldbus devices.

The following table provides an overview of the main interfaces used by PRG_LocalMTSLinkMonitor.

Function / Function Block	Library / Namespace	Purpose in the Reference Application
GetRedundancyState()	RDCY	Retrieves the current CODESYS redundancy state of the local controller.
GetPlcIdent()	RDCY	Retrieves the fixed identity of the local controller.
SwitchToActive()	RDCY	Requests a transition of the local Standby controller to Active.
RedundancyManager	Modbus TCP redundancy functionality	Coordinates the Modbus TCP client with the CODESYS redundancy state.
FB_IsIpExtern	libWilpAlias	Verifies whether a configured external IP address is reachable from the local controller.

The interfaces have different responsibilities within the monitoring mechanism. **GetRedundancyState()** and **GetPlcIdent()** provide information from the redundancy system to the application, while **SwitchToActive()** allows the application to request a redundancy role change. **FB_IsIpExtern** provides the additional network reachability information required for the switchover decision.

CODESYS Redundancy Interfaces

GetRedundancyState()

GetRedundancyState() retrieves the current redundancy state information of the local controller and stores it in a **RedundancyState** structure.

The Link Monitor calls this function cyclically and evaluates:

rsRedundancyState.eRedundancyState

The most relevant states for normal Link Monitor operation are:

Redundancy State	Meaning for the Link Monitor
RS_CYCLE_ACTIVE	The local controller executes the active monitoring behavior.
RS_CYCLE_STANDBY	The local controller executes the passive monitoring behavior and may perform reachability checks.
RS_CYCLE_STANDALONE	The Link Monitor returns to or remains in init.
Error states	The Link Monitor transitions to its error state.

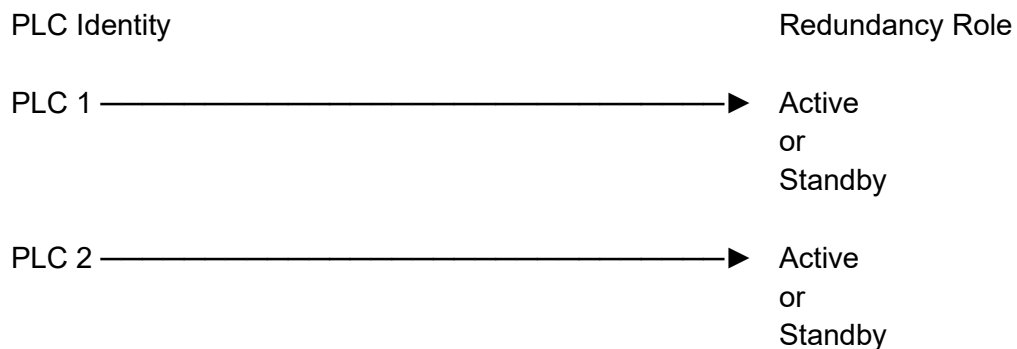
The **RedundancyState** structure additionally provides **eRedundancyError** and **stRedundancyError**, which contain diagnostic information about the current redundancy error.

GetPlcIdent()

GetPlcIdent() returns the fixed redundancy identity of the local PLC.

This identity must not be confused with the current Active/Standby role.

The PLC identity identifies the physical controller, whereas the redundancy state identifies its current operating role:



A controller therefore retains its PLC identity after a redundancy switchover.

PRG_LocalMTSLinkMonitor retrieves the identity during initialization and uses it, among other purposes, for controller-specific diagnostic information.

SwitchToActive()

SwitchToActive() provides the interface from the application back to the CODESYS redundancy system.

The function is called by the Link Monitor only after the Standby controller has verified that at least one monitored fieldbus device is reachable through its own local network path.

The call:

RDCY.SwitchToActive()

requests that the local controller changes from:

RS_CYCLE_STANDBY

to:

RS_CYCLE_ACTIVE.

The state transition is not performed directly by **PRG_LocalMTSLinkMonitor**. The application only requests the change; the actual transition is subsequently coordinated by the CODESYS redundancy system.



Note:

The Link Monitor does not directly assign or manipulate the CODESYS redundancy state. **SwitchToActive()** is the defined interface used to request a controlled role change.

RedundancyManager

The RedundancyManager function block is used to coordinate the Modbus TCP client with the CODESYS redundancy system.

The reference application declares:

fbModbusTcpMgn : RedundancyManager

and connects the corresponding Modbus TCP client instance to the function block.

The RedundancyManager provides information including:

Output	Purpose
xRedundancyActive	Indicates whether CODESYS PLC Redundancy is available.
PLC_ID	Provides the PLC identity used by the redundancy system.
RedundancyState	Provides the current CODESYS redundancy state.
xConnected	Indicates whether the two redundant controllers can communicate through the configured redundancy link.
xSync	Indicates whether the redundant controllers are synchronized.
xPrimary	Indicates that the associated Modbus TCP client currently operates as Primary.
xBackup	Indicates that the associated Modbus TCP client currently operates as Backup.

The responsibility of **RedundancyManager** must be distinguished from the Local Modbus TCP Link Monitoring function.

RedundancyManager manages the Modbus TCP client according to the CODESYS redundancy state. It does **not** perform the additional local network reachability evaluation described in Chapter 5.2 and does not independently decide whether a Standby controller should take over because its network path is better.

This additional decision logic is implemented by **PRG_LocalMTSLinkMonitor**.



Note:

According to the function block documentation, **RedundancyManager** does not resolve the condition in which both PLCs operate in Standalone mode. In such a condition, Modbus TCP communication may become active on both PLCs. This behavior must be considered when adapting the reference application to another system architecture.

FB_IsIpExtern

FB_IsIpExtern is used by the Standby controller to determine whether a configured fieldbus device can be reached through its local network connection.

Two instances are used in the reference application:

fbIsIpExtern1

fbIsIpExtern2

Each instance receives a parameter structure of type **ST_IsIpExternParams**.

The relevant parameters are:

Parameter	Type	Purpose
arbIPtoTest	ARRAY [0..3] OF BYTE	IPv4 address of the fieldbus device to be checked.
udiPingTimeout	UDINT	Maximum duration of the ping operation in milliseconds.

When activated, **FB_IsIpExtern** executes a ping to the configured IP address and evaluates whether the responding IP address represents an external device rather than a local address of the controller.

The most important output for the Link Monitor is:

q_xIpIsExtern

The result is interpreted as follows:

Result	Meaning
q_xIpIsExtern = TRUE	The configured external device is reachable.
q_xIpIsExtern = FALSE	No reachable external device was detected, for example due to a ping timeout or unavailable network connection.
q_xError = TRUE	The reachability check itself encountered an error.

The Link Monitor waits until **both instances report q_xDone = TRUE** before evaluating their results.

A switchover may be requested when:

fbIsIpExtern1.q_xIpIsExtern OR fbIsIpExtern2.q_xIpIsExtern

evaluates to TRUE.

Consequently, one reachable fieldbus device is sufficient to demonstrate that the Standby controller has a usable local path into the fieldbus network.

Required Runtime Permission

FB_IsIpExtern executes Linux commands on the controller. CODESYS must therefore be permitted to execute the required system processes.

The following configuration must be added to `/etc/CODESYSControl1_User.cfg`:

```
[SysProcess]
Command=AllowAll
```

The PLC must be restarted after changing the configuration.



Note:

FB_IsIpExtern performs an IP reachability check. A successful result confirms that the configured external IP address is reachable from the local controller. It does not verify the complete Modbus TCP application communication.

5.5 Configuration and Parameters

The Local Modbus TCP Link Monitoring function requires application-specific parameters as well as configuration on the CODESYS runtime and redundancy system.

The reference application contains predefined values for the monitored fieldbus devices and the monitoring intervals. These values are suitable for the reference topology and must be reviewed when the function is transferred to another application.

The following sections describe the relevant configuration parameters and system requirements.

Fieldbus Device Configuration

The Link Monitor verifies the local network path by checking the reachability of the two Modbus TCP fieldbus devices used in the reference topology.

Two parameter structures are assigned to the corresponding **FB_IsIpExtern** instances:

- **stIsIpExtern1Params**
- **stIsIpExtern2Params**

Each structure is of type **ST_IsIpExternParams** and contains the target IPv4 address and the ping timeout:

Parameter	Type	Description
arbIPtoTest	ARRAY [0..3] OF BYTE	IPv4 address of the fieldbus device used for the reachability check.
udiPingTimeout	UDINT	Maximum time in milliseconds to wait for the reachability check to complete.

During initialization of **PRG_LocalMTSLinkMonitor**, the IP addresses of the two monitored Modbus TCP devices are assigned to the corresponding parameter structures.

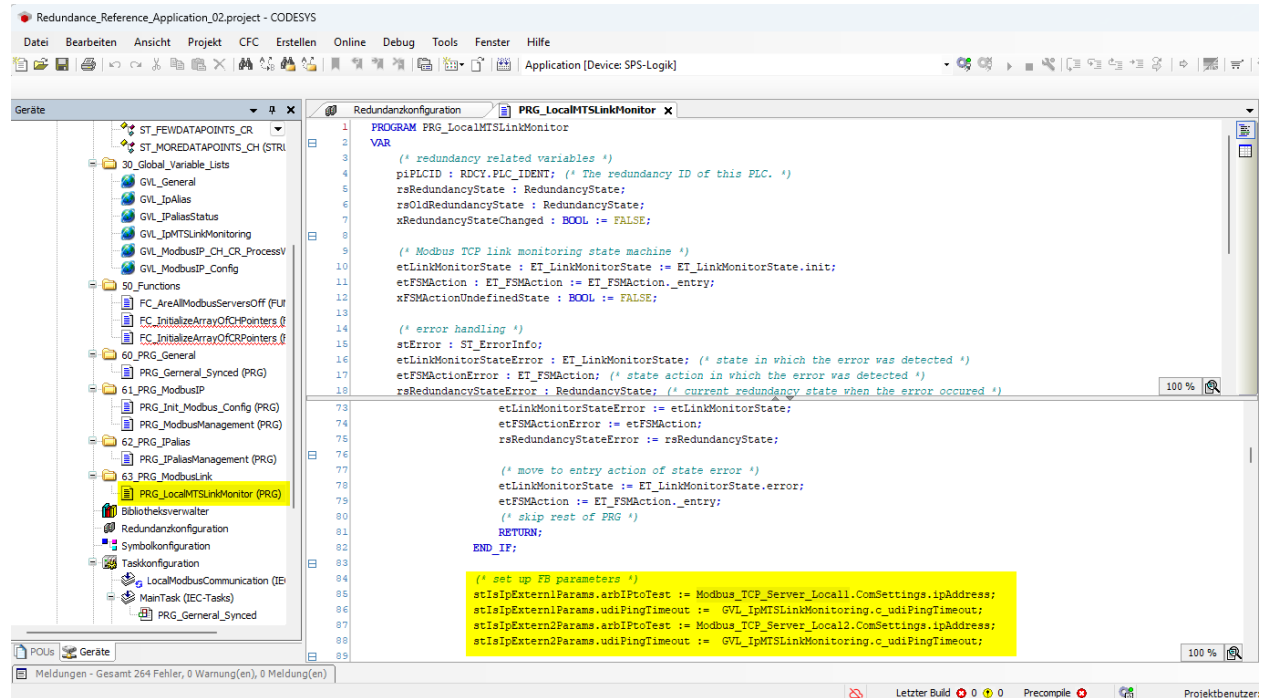
The configured target addresses must represent devices that are reachable through the network path whose availability is relevant for the redundancy decision.

For the reference topology, the two Modbus TCP fieldbus couplers are used for this purpose. If at least one of these devices is reachable from the Standby controller, the Link Monitor considers the Standby controller to have a usable path to the Modbus TCP network.



Note:

When adapting the reference application, the monitored IP addresses must be selected according to the actual network topology. The devices should be located behind the network path that is intended to be verified. Selecting an unrelated reachable device would not provide valid information for the controller switchover decision.



Configuration of the monitored Modbus TCP device IP addresses



Note:

In the provided reference application, the IP addresses used by the Link Monitor are obtained automatically from the configured Modbus TCP server instances. The application reads the configured server addresses from **Modbus_TCP_Server_Local1.ComSettings.ipAddress** and **Modbus_TCP_Server_Local2.ComSettings.ipAddress** and assigns them to the corresponding **FB_IsIpExtern** parameter structures. Therefore, the IP addresses do not need to be configured separately for the Link Monitor. When adapting the reference application, ensure that the configured Modbus TCP server addresses correspond to the fieldbus devices that shall be used for the reachability check.

Monitoring and Timing Parameters

The timing behavior of the Link Monitor is controlled by constants defined in **GVL_IpMTSLinkMonitoring**.

The reference application uses the following values:

Parameter	Reference Value	Description
c_udiPingTimeout	1000 ms	Maximum duration of an individual FB_IsIpExtern reachability check.
c_tRetryMTSPing	1 s	Delay before repeating the reachability check if neither fieldbus device is reachable.
c_tErrorAutoAck	10 s	Delay used by the automatic error recovery mechanism.

Ping Timeout

c_udiPingTimeout defines how long **FB_IsIpExtern** waits for the configured device to respond. The reference value is:

```
c_udiPingTimeout := 1000
```

A shorter timeout can reduce the time required to detect an unavailable device but may increase the probability of interpreting temporarily delayed responses as unavailable.

A longer timeout increases tolerance to delayed responses but also increases the time required to complete the reachability check.

Because both **FB_IsIpExtern** instances must complete before their results are evaluated, the configured timeout directly influences the response time of the Link Monitor.

Ping Retry Time

If both reachability checks complete and neither fieldbus device is reachable, the Link Monitor does not request a controller switchover.

Instead, the retry timer is started using:

```
c_tRetryMTSPing := T#1s
```

After the timer expires, both **FB_IsIpExtern** instances are reset and a new reachability check is initiated.

The reference value therefore prevents continuous ping execution while still allowing the Link Monitor to detect when the network path becomes available again.

Error Auto-Acknowledge Time

The constant:

```
c_tErrorAutoAck := T#10s
```

is used by the Link Monitor error recovery mechanism.

It defines the delay before an error that permits automatic recovery is acknowledged and the Link Monitor is reinitialized.

The detailed error handling behavior is described in Chapter 5.6.



Note:

The reference values should not automatically be considered suitable for every application. Network latency, required failover time, fieldbus availability requirements and expected transient network disturbances should be considered when selecting the timing parameters.

Required Controller and Redundancy Configuration

In addition to the application parameters, the Local Modbus TCP Link Monitoring function depends on two system-level configuration requirements.

CODESYS Redundancy Data Synchronization

PRG_LocalMTSLinkMonitor is executed locally in the **NonReduConfig** task and its internal state is not synchronized between the controllers.

However, the information required for coordination between the local monitoring instances must be available on both controllers.

For this purpose:

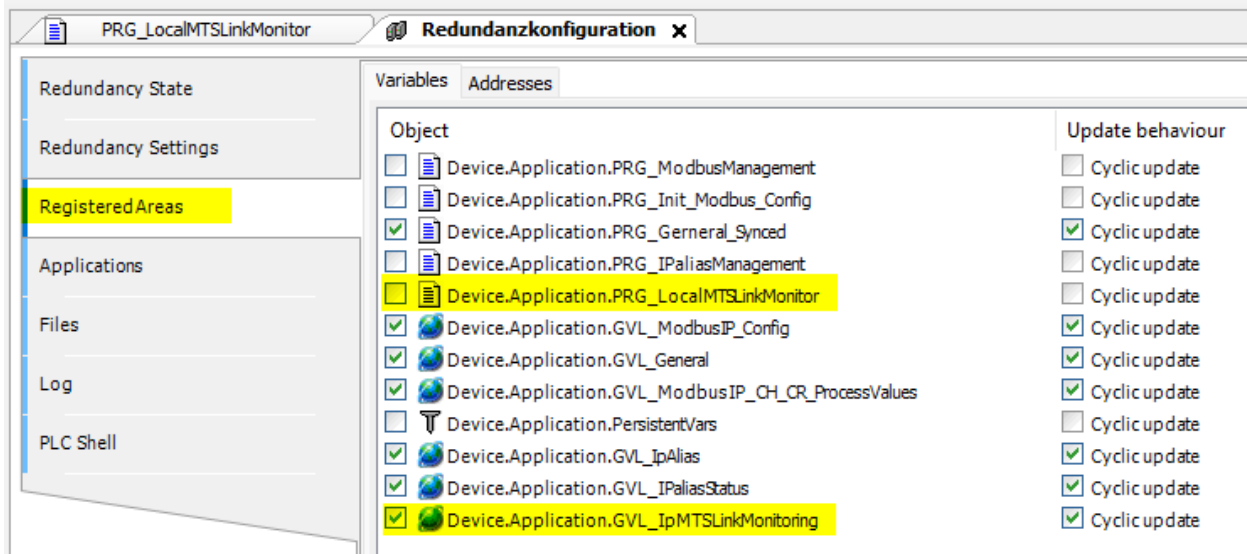
GVL_IpMTSLinkMonitoring

is included in the **Registered Areas** of the CODESYS Redundancy Configuration with cyclic synchronization enabled.

This includes the communication status:

xMTSHandlerOK

which allows the Standby controller to evaluate the Modbus TCP communication status reported by the Active controller.



CODESYS Redundancy Configuration - Registered Areas for Local Modbus TCP Link Monitoring



Note:

PRQ_LocalMTSLinkMonitor is intentionally not included in the registered redundancy areas. The program is executed locally on both controllers and maintains its own state machine, timers, and function block instances.

GVL_IPMTSLinkMonitoring, however, is registered with **Cyclic update** enabled. This allows the information required for coordination between the two local Link Monitor instances, in particular **xMTSHandlerOK**, to be synchronized between the controllers.

CODESYS Runtime Permission for FB_IsIpExtern

FB_IsIpExtern performs the reachability check by executing Linux commands on the controller.

The CODESYS runtime must therefore be permitted to execute the required system processes. Add the following configuration to:

`/etc/CODESYSControl_User.cfg`

```
[SysProcess]
Command=AllowAll
```

The controller must be restarted after modifying the configuration file.

Without this permission, **FB_IsIpExtern** cannot execute the required command and reports a command execution error.



Note:

Command=AllowAll permits the CODESYS runtime to execute system commands and therefore increases the privileges available to the application. This setting should be considered as part of the system security concept and only be enabled where required by the application.

5.6 Error Handling and Recovery

PRG_LocalMTSLinkMonitor contains dedicated error handling to distinguish between expected communication failures and internal errors that prevent the monitoring function from operating correctly.

A loss of the monitored Modbus TCP connections or an unsuccessful reachability check is not treated as an internal error. These conditions are part of the normal Link Monitor behavior and are handled by the **active**, **passive**, and **ping** states.

The Link Monitor enters the **error** state when the monitoring function itself cannot continue its intended operation. This can be caused, for example, by an error reported by **FB_IsIpExtern**, an invalid state within the Link Monitor state machine, or a CODESYS redundancy state that indicates an abnormal operating condition.

The following distinction is therefore important:

Condition	Handling
Both monitored Modbus TCP connections are unavailable	Normal monitoring condition → Standby enters ping
Neither fieldbus device responds to the reachability check	Normal monitoring condition → wait and retry
FB_IsIpExtern reports an execution error	Transition to error
CODESYS redundancy state indicates an error condition	Transition to error
Undefined Link Monitor or FSM action state	Transition to error

Error Detection and Diagnostic Information

When the Link Monitor enters the **error** state, it first determines the source of the error and stores the available diagnostic information.

If either **fbIsIpExtern1** or **fbIsIpExtern2** reports an error with valid error information, the corresponding **q_stError** structure is copied to:

GVL_IpMTSLinkMonitoring.stError

The CODESYS redundancy state that was present when the error occurred is additionally stored in:

GVL_IpMTSLinkMonitoring.rsErrorRedundancyStates[piPLCID]

and the common error flag is set:

GVL_IpMTSLinkMonitoring.xError := TRUE

The affected **FB_IsIpExtern** instance is then disabled before the Link Monitor continues with the cyclic error-handling logic.

If the error did not originate from either **FB_IsIpExtern** instance, the internally generated **stError** structure is stored instead. This is used, for example, for invalid redundancy states or undefined state-machine conditions.

The Link Monitor stores additional context internally to simplify diagnosis:

Variable	Stored Information
etLinkMonitorStateError	Link Monitor state in which the error was detected
etFSMActionError	FSM action (_entry , _do , or _exit) active when the error occurred
rsRedundancyStateError	CODESYS redundancy state at the time of the error
stError	Detailed application-level error information



Note:

A negative reachability result from **FB_IsIpExtern** must not be confused with a function block error. If a device does not respond to ping, **q_xIpIsExtern** is evaluated as **FALSE** and the Link Monitor remains within the normal **ping** retry sequence. The **error** state is only entered if the reachability check itself reports an execution or internal error.

Automatic Recovery from Redundancy State Errors

The Link Monitor distinguishes errors caused by the CODESYS redundancy state from other monitoring errors.

If a redundancy state caused the transition to error, the internal flag:

xRedundancyStateRaisedError

is set.

While the Link Monitor remains in the error state, the current redundancy state is continuously evaluated. Automatic recovery is permitted when the redundancy system subsequently reaches one of the following states:

- **RS_CYCLE_ACTIVE**
- **RS_CYCLE_STANDBY**
- **RS_CYCLE_STANDALONE**
- **RS_SHUTDOWN_ACTIVE**
- **RS_SHUTDOWN_STANDBY**

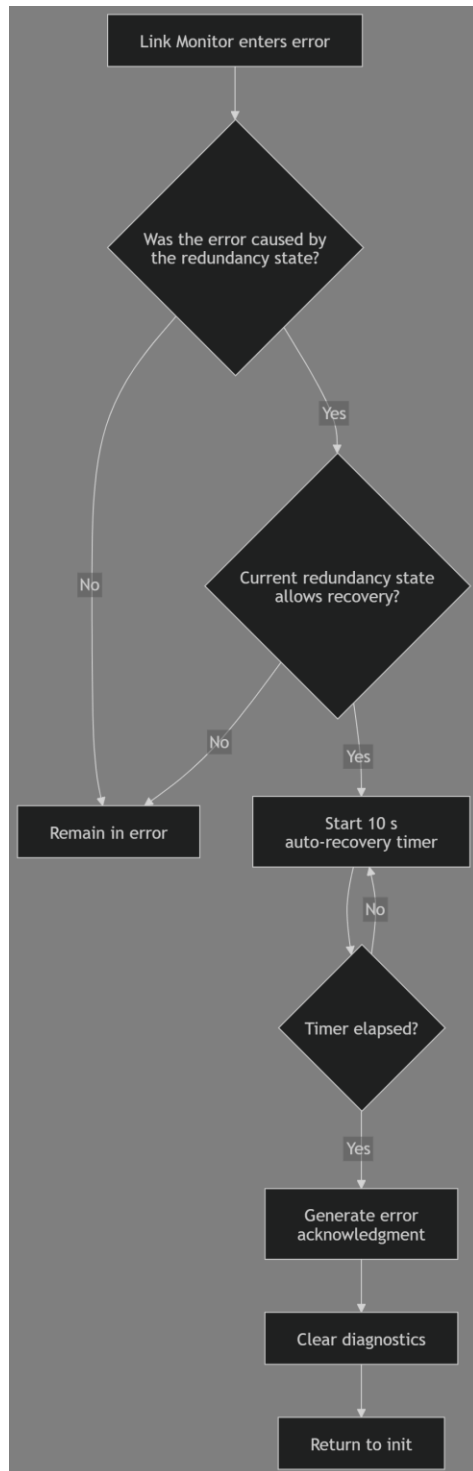
When one of these states is detected, the automatic recovery timer is started.

The reference application uses:

c_tErrorAutoAck := T#10s

If the redundancy state remains suitable for recovery until this timer expires, the Link Monitor generates an internal error acknowledgment.

The sequence can be represented as follows:



The delay prevents the Link Monitor from immediately restarting while the redundancy system is still transitioning between states.



Note:

The 10-second automatic recovery mechanism is only activated when **xRedundancyStateRaisedError = TRUE**. Errors originating from **FB_IsIpExtern** or other internal Link Monitor errors do not automatically start this recovery timer.

Error Acknowledgment and Reinitialization

The actual recovery process is triggered by a rising edge of the error acknowledgment signal.

When an acknowledgment is detected, the Link Monitor clears the shared diagnostic information:

- **GVL_IpMTSLinkMonitoring.xError** is reset.
- The stored redundancy state is reset to **RS_START**.
- The stored redundancy error is reset to **NO_ERROR**.
- The redundancy error text is cleared.
- The application error module, source, reason, parameters, and timestamp are reset.
- **xErrorDataValid** is reset.

Both **FB_IsIpExtern** instances are then completely deactivated:

```
i_xEnable := FALSE
```

```
i_xExecute := FALSE
```

Finally, the Link Monitor returns to:

```
ET_LinkMonitorState.init
```

with:

```
ET_FSMAction._entry
```

The complete monitoring function is therefore initialized again before normal Active/Standby behavior resumes.



Note:

The current implementation explicitly generates **xErrorAcknowledge** automatically for recoverable redundancy-state errors. The shown source code does not provide a separate application interface for automatically acknowledging errors originating from **FB_IsIpExtern** or other internal errors. If manual recovery of these errors is required in the target application, the acknowledgment concept should be reviewed and adapted accordingly.

6 Common IP Address Management

6.1 Purpose and Monitored Failure Scenario

In a redundant controller system, both controllers have their own individual IP address. Higher-level systems such as SCADA systems or OPC UA clients would therefore normally have to address a specific controller. After a redundancy switchover, the communication partner would either have to connect to a different IP address or implement additional logic to determine which controller is currently providing the required service.

The reference application avoids this dependency by providing a **Common IP address** for communication with the redundant controller pair. From the perspective of the higher-level system, the redundant controllers therefore appear as a single communication endpoint. The client can continue to use the same destination IP address regardless of which controller is currently Active.

During normal redundant operation, ownership of the Common IP follows the CODESYS redundancy role. The Active controller provides the Common IP, while the Standby controller does not provide the Common IP. Both controllers retain their individually configured IP addresses and therefore remain directly accessible through these addresses, for example for engineering access or the controller Web UI. The Common IP is provided as an additional IP address on the same Ethernet interface of the controller currently responsible for the service. Consequently, this interface can listen on both the individual controller IP address and the Common IP at the same time. When the redundancy roles change, only ownership of the Common IP is transferred to the new Active controller; the individual controller IP addresses remain unchanged.



Note:

When secure OPC UA communication is used, the security and certificate configuration must be implemented independently on both controllers of the redundant system. Although the higher-level system always connects to the same Common IP address, the OPC UA service may be provided by either physical controller after a redundancy switchover.

Both controllers must therefore be configured with the same OPC UA security requirements, for example the same Security Policy and Message Security Mode such as “Sign & Encrypt”. However, each controller should use its own OPC UA server certificate and corresponding private key.

The trust relationships must be configured symmetrically:

- “Controller 1” must trust the certificate of the SCADA or OPC UA client.
- “Controller 2” must trust the certificate of the SCADA or OPC UA client.
- The “SCADA or OPC UA client” must trust the server certificate of Controller 1.
- The “SCADA or OPC UA client” must also trust the server certificate of Controller 2.

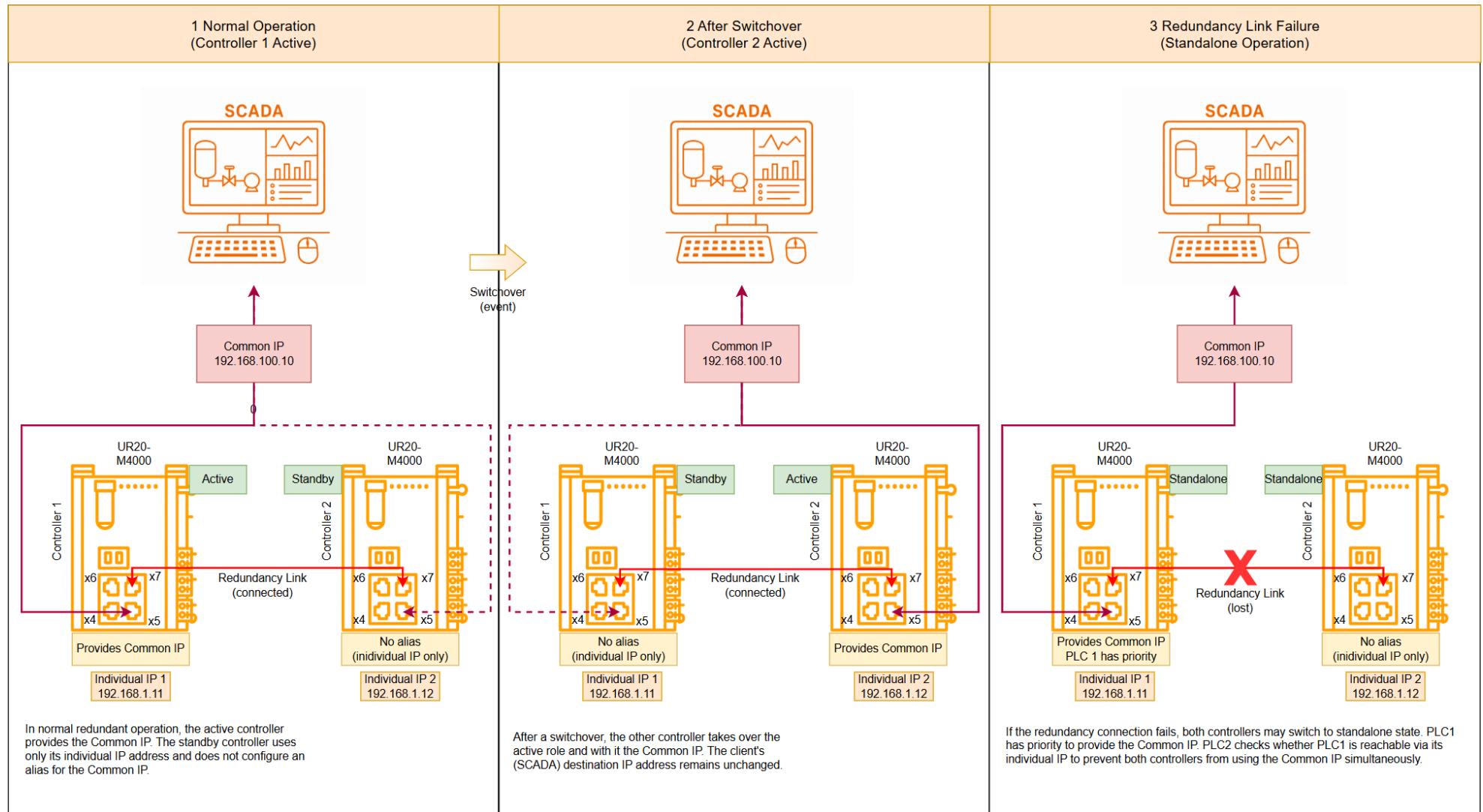
This configuration is required because a controller switchover changes the physical OPC UA server that terminates the secure connection. The Common IP address remains unchanged, but the newly Active controller presents its own server certificate. If this certificate is not already trusted by the client, the OPC UA connection may be rejected after the switchover.

The certificates used by both controllers must also be valid for the configured OPC UA endpoint. If the OPC UA client validates the host name or IP address against the server certificate, the Common IP address or the corresponding common DNS name must be considered when the certificates are created.

Certificate deployment, renewal and replacement must therefore be coordinated across both controllers and the higher-level system. New certificates and trust relationships should be installed on all relevant communication partners before an existing certificate is replaced or removed. This prevents a redundancy switchover from causing a communication failure due to an incomplete trust configuration.

After a controller switchover, the OPC UA client must establish a new secure connection to the newly Active controller. The Common IP mechanism provides a constant network endpoint, but it does not synchronize OPC UA certificates, security settings, trust lists, SecureChannels or sessions between the two controllers.

Reference Application for CODESYS Controller Redundancy



Symbol / Color	Meaning
 Orange	Physical devices and hardware elements, such as the SCADA station and UR20-M4000 controllers.
 Light green status field	Current CODESYS redundancy state of the controller: Active, Standby, or Standalone.
 Magenta field	Common IP address used by the higher-level system.
 Magenta solid line	Active Common IP communication path. It shows which controller currently provides the Common IP to SCADA.
 Magenta dashed line	Inactive Common IP path. The corresponding controller does not currently provide the Common IP alias.
 Bright red X	Lost redundancy connection between Controller 1 and Controller 2.
 Light yellow field	Controller-specific IP information or description of the current IP behavior, such as Provides Common IP, No alias, or the individual controller IP address.

A particular failure scenario must be considered when the redundancy connection between the controllers is lost. Depending on the resulting redundancy state, the controllers can operate in RS_CYCLE_STANDALONE. In this condition, the normal Active/Standby relationship can no longer be used as the sole criterion for assigning the Common IP. The application must prevent both controllers from independently providing the same address.

For this reason, the reference application implements additional collision-avoidance logic for Standalone operation. PLC1 has priority for providing the Common IP. PLC2 is only permitted to provide the Common IP if PLC1 cannot be reached using its individual controller address. If PLC1 becomes reachable again while PLC2 is providing the Common IP, PLC2 withdraws the Common IP and returns to passive operation.

The Common IP Address Management therefore addresses two related requirements:

- **Transparent controller switchover:** Higher-level communication partners can use a constant destination IP address independent of the current Active controller.
- **IP collision avoidance:** The application prevents, as far as possible, simultaneous use of the Common IP by both controllers when normal redundancy coordination is unavailable.



Note:

The Common IP Address Management does not replace or control CODESYS Redundancy. The Active/Standby roles are determined by the CODESYS redundancy mechanism. The reference application uses the resulting redundancy state and additional reachability checks to determine which controller may provide the Common IP.

6.2 Functional Principle

The Common IP Address Management is implemented by **PRG_IPAliasManagement**. The program is executed locally on both controllers and evaluates the current CODESYS redundancy state to determine whether the local controller shall provide the Common IP or operate without the IP alias.

The physical controller identity is determined using **GetPlcIdent()**, while **GetRedundancyState()** provides the current redundancy state. The PLC identity remains fixed for each controller and is therefore independent of whether the controller is currently Active, Standby, or Standalone.

During normal redundant operation, the assignment of the Common IP directly follows the redundancy role:

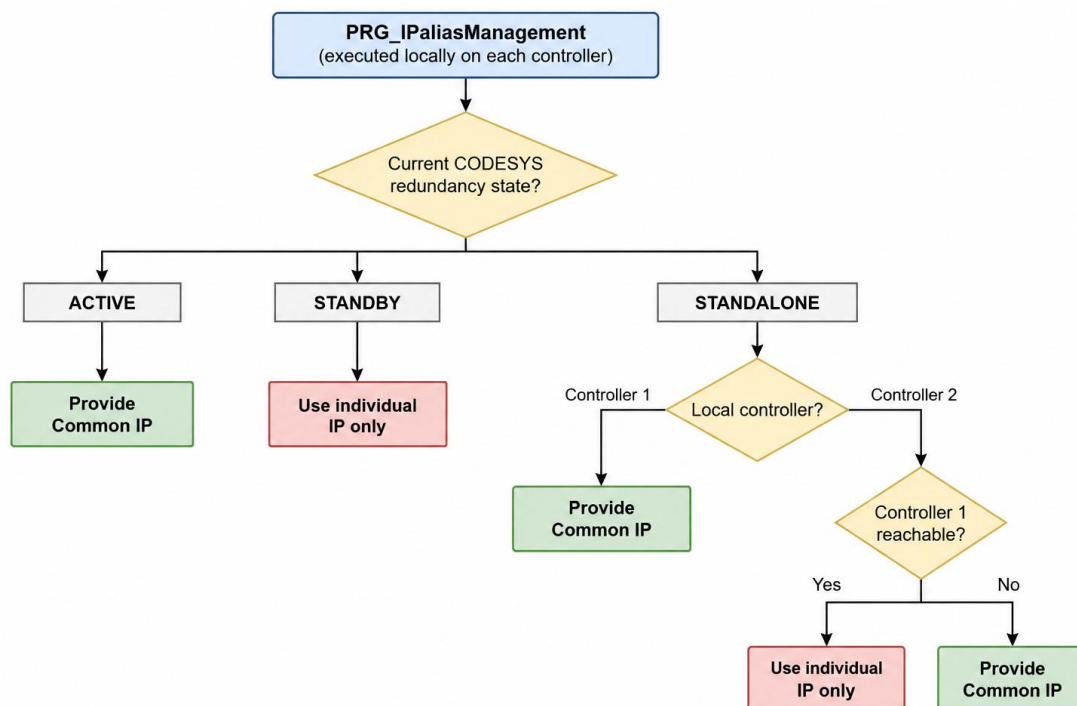
Redundancy State	Common IP Behavior
RS_CYCLE_ACTIVE	The controller provides the Common IP.
RS_CYCLE_STANDBY	The controller does not provide the Common IP and uses its individual IP address.
RS_CYCLE_STANDALONE	The controller identity and reachability of PLC1 are additionally evaluated to determine Common IP ownership.

After initialization, an Active controller enters the **offer_server** operating state, while a Standby controller enters the **passive** state.

A special case occurs in **RS_CYCLE_STANDALONE**. In this state, PLC1 has priority for providing the Common IP. PLC2 first verifies whether PLC1 is reachable using its individual IP address. If PLC1 is reachable, PLC2 remains passive and uses only its individual IP address. If PLC1 cannot be reached, PLC2 may provide the Common IP.

While PLC2 provides the Common IP in Standalone operation, the reachability of PLC1 continues to be checked. If PLC1 becomes reachable again, PLC2 withdraws the Common IP and returns to passive operation.

The resulting decision logic for Common IP ownership is summarized in the following figure.



6.3 IP Alias State Machine

State Machine Overview

The behavior of PRG_IPAliasManagement is implemented as a finite state machine. The state machine controls the local IP configuration according to the current CODESYS redundancy state and the fixed physical controller identity.

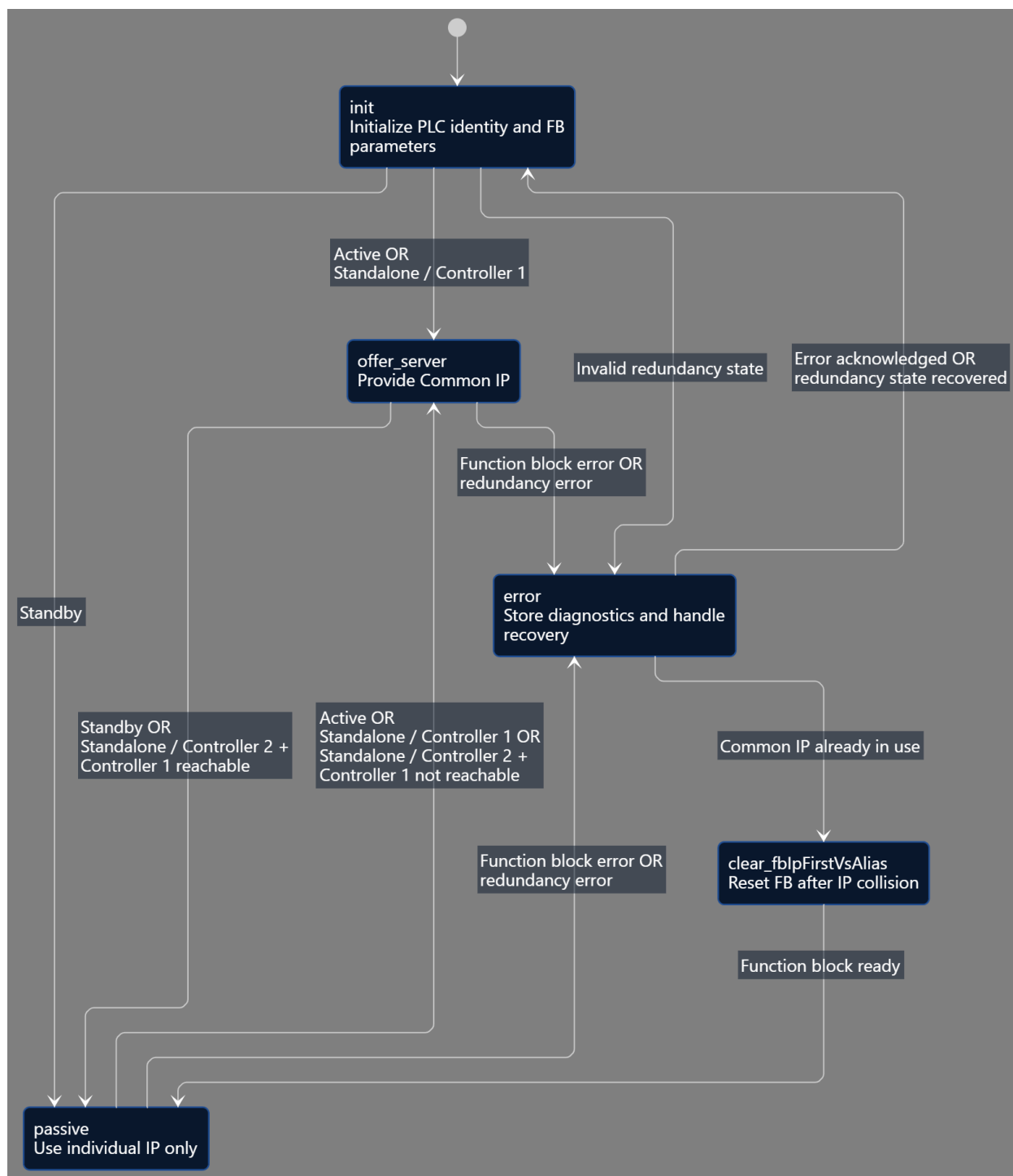
The current state is stored in:

etLRCState : ET_LRCState

Each state is additionally divided into the actions `_entry`, `_do`, and `_exit` using `ET_FSMAction`. This separates one-time initialization activities from cyclic processing and state-transition handling.

The following states are used by the IP Alias Management function:

State	Purpose
init	Initializes the local controller identity, function block parameters, and determines the required operating state.
passive	Ensures that the local controller operates with its individual IP address and does not provide the Common IP.
offer_server	Provides the Common IP on the local controller.
clear_fbIpFirstVsAlias	Clears a detected Common IP collision condition and prepares the controller to return to passive operation.
error	Stores diagnostic information and handles recovery from function block, state-machine, or redundancy-related errors.



State Action Concept

The IP Alias Management state machine uses the secondary state variable:

etFSMAction : ET_FSMAction

Each state is divided into the actions **_entry**, **_do**, and **_exit**.

- **_entry** is used for one-time initialization when entering a state.
- **_do** contains the cyclic processing performed while the state remains active.
- **_exit** is used for actions required before leaving the state.

This structure ensures that initialization and reset operations are not executed repeatedly in every task cycle.

Init State

The **init** state prepares the complete IP Alias Management function for operation.

During **_entry**, the program first resets internal state-machine, error-handling, and IP-collision-check flags. It then retrieves the fixed local PLC identity using **GetPlcIdent()**. Only the identities PLC1 and PLC2 are accepted. An invalid identity generates application error information and causes an immediate transition to **error**.

The parameter structures of the three IP-related function blocks are then initialized.

FB_IpFirstVsAlias receives the Common IP address, its subnet mask, the Ethernet interface name, and the configured ping timeout:

- **GVL_IpAlias.c_OPClientExpectsIpAddress**
- **GVL_IpAlias.c_OPClientExpectsIpAddressMask**
- **GVL_IpAlias.c_strEthItfName**
- **GVL_IpAlias.c_udiPingTimeout**

FB_IpChange is configured with the individual IP address and subnet mask belonging to the local physical controller. The PLC identity is used as the index into **GVL_IpAlias.c_IpAddresses** and **GVL_IpAlias.c_IpAddressMasks**.

FB_IsIpExtern is configured with the individual IP address of PLC1. This address is later used by PLC2 for the Standalone collision-avoidance check.

After parameterization, all three function blocks are enabled and the state machine enters **_do**.

During **_do**, initialization remains active until all three function blocks report **q_xStandby = TRUE**. Only after this condition is fulfilled does the state machine proceed to **_exit**.

During **_exit**, the current CODESYS redundancy state determines the next operating state:

CODESYS Redundancy State	Next IP Alias State
RS_CYCLE_ACTIVE	offer_server
RS_CYCLE_STANDBY	passive
RS_CYCLE_STANDALONE	offer_server or passive , depending on PLC identity and reachability logic
Redundancy error state	error

An important implementation detail is that **RS_CYCLE_STANDALONE** initially leads to **offer_server** on both physical controllers. If the local controller is PLC2, the collision-avoidance logic subsequently verifies whether PLC1 is present and withdraws the Common IP if necessary.

Passive State

The **passive** state ensures that the local controller operates with its individual IP address and does not provide the Common IP.

During **_entry**, the state machine starts **FB_IpChange** by setting:

```
fbIpChange.i_xExecute := TRUE
```

The function block applies the individual IP configuration prepared during **init**.

During **_do**, the state machine waits until **fbIpChange.q_xDone = TRUE**. Once the IP change has completed, the execution input is reset and the state machine proceeds to **_exit**.

During **_exit**, the program waits until **FB_IpChange** has returned to its Standby state. Only then is the current redundancy state evaluated.

If the controller is now **RS_CYCLE_ACTIVE**, the state machine transitions directly to **offer_server**.

If the controller is **RS_CYCLE_STANDALONE**, the physical PLC identity is evaluated:

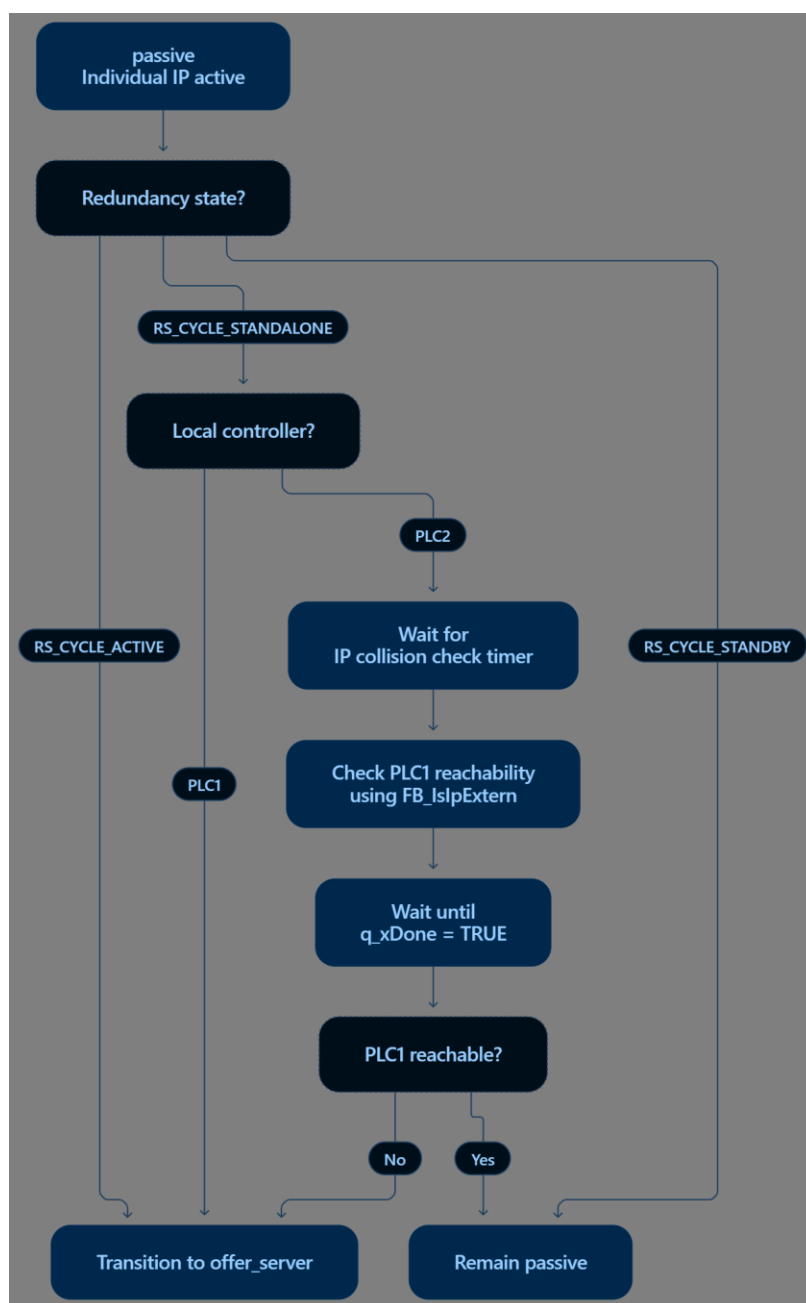
- **PLC1**: transitions directly to **offer_server**.
- **PLC2**: starts the IP-collision-check procedure.

For PLC2, the collision-check timer is activated. After the configured delay has elapsed, **FB_IsIpExtern** checks whether PLC1 can be reached using its individual IP address.

If PLC1 is **not reachable**, PLC2 transitions to **offer_server** and may take over the Common IP.

If PLC1 is **reachable**, PLC2 remains in **passive** and therefore continues using only its individual IP address.

If the redundancy state remains **RS_CYCLE_STANDBY** or enters one of the handled shutdown states, no additional state transition is initiated. Defined abnormal redundancy states cause a transition to **error**.



Offer Server State

The **offer_server** state is responsible for providing the Common IP on the local controller.

During **_entry**, **FB_IpFirstVsAlias** is activated:

```
fbIpFirstVsAlias.i_xExecute := TRUE
```

The state machine then enters **_do**.

The function block first performs the address handling required to provide the configured Common IP. The state machine waits until **fbIpFirstVsAlias.q_xDone = TRUE** before evaluating the current redundancy state.

If the controller is **RS_CYCLE_ACTIVE**, no further action is required and the Common IP remains provided by the local controller.

If the controller is **RS_CYCLE_STANDALONE**, additional behavior is required only on PLC2.

PLC2 periodically starts an IP-collision check using **FB_IsIpExtern**. The target of this check is the individual IP address of PLC1.

If PLC1 responds, PLC2 assumes that PLC1 is present in the network and should retain priority for the Common IP. PLC2 therefore:

1. deactivates **FB_IpFirstVsAlias**,
2. selects **passive** as the next state, and
3. enters the **_exit** action of **offer_server**.

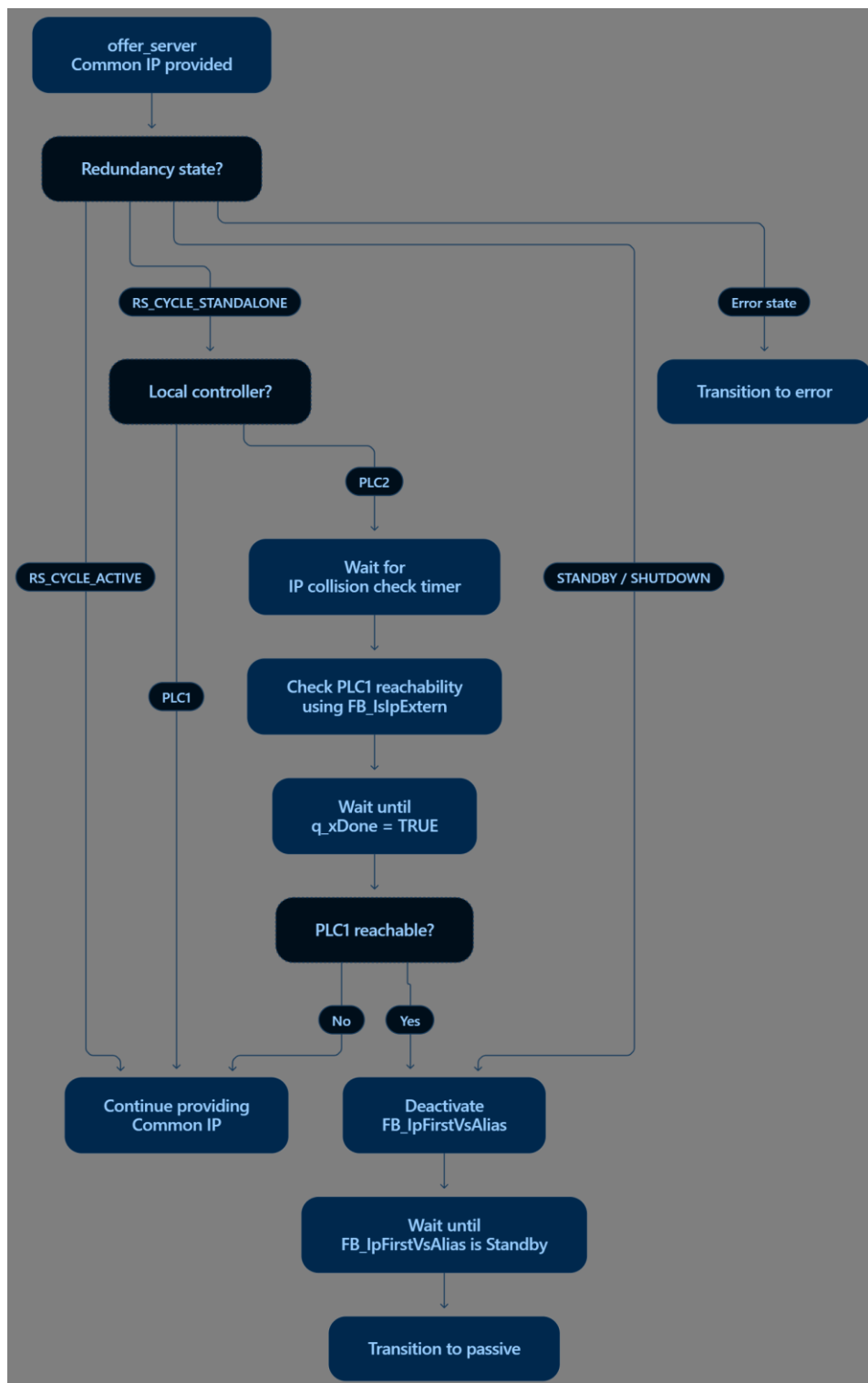
If PLC1 cannot be reached, PLC2 continues providing the Common IP.

If the redundancy state changes to **RS_CYCLE_STANDBY** or one of the handled shutdown states, the Common IP is also withdrawn. **FB_IpFirstVsAlias** is deactivated and **passive** is selected as the next state.

For abnormal redundancy states, the function block is likewise deactivated, but the selected next state is error.

During **_exit**, the state machine waits until **FB_IpFirstVsAlias** has returned to its Standby state. Only then does it enter the previously selected state stored in **etLRCNewState**.

This prevents a transition to another operating state while the previous Common IP operation is still being terminated.



Clear FB_IpFirstVsAlias State

`clear_fbIpFirstVsAlias` is a helper state used for a specific collision scenario.

FB_IpFirstVsAlias can report that the requested Common IP already exists elsewhere in the network:

IP address already exists in the network and is not locally set

Within this reference application, this condition can occur when the other controller already provides the Common IP during Standalone operation. Instead of reporting this condition as a general application error, the state machine interprets it as an expected collision-avoidance condition. **FB_IpFirstVsAlias** is therefore deactivated and disabled, and the state machine enters `clear_fbIpFirstVsAlias`.

During `_entry`, the state waits until the function block is completely idle:

- `q_xActive = FALSE`
- `q_xStandby = FALSE`
- `q_xError = FALSE`
-

The function block is then enabled again.

During `_do`, the state machine waits until `q_xStandby = TRUE`. Once the function block has been successfully reinitialized, the state machine returns to **passive**.

No separate `_exit` action is required.

Error State

The **error** state handles errors reported by:

- **FB_IpFirstVsAlias**,
- **FB_IpChange**,
- **FB_IsIpExtern**,
- invalid internal FSM states, and
- abnormal CODESYS redundancy states.

When entering the state, the application determines the source of the error and copies the corresponding diagnostic information to **GVL_IPAliasStatus**. The redundancy state that was active when the error occurred is also stored for diagnostic purposes.

The special case in which **FB_IpFirstVsAlias** detects the Common IP elsewhere in the network is excluded from normal error reporting and handled through **clear_fbIpFirstVsAlias** as described above.

For errors caused by the CODESYS redundancy state, the application evaluates whether a subsequent valid redundancy state permits automatic recovery. When recovery is permitted, an internal error acknowledgment is generated and the state machine is reinitialized.

After an acknowledgment, the stored diagnostic information is cleared, all IP-related function blocks are disabled and reset, and the state machine returns to:

ET_LRCState.init

with:

ET_FSMAction._entry

The detailed error information, recovery conditions, and diagnostic interfaces are described separately in Chapter 6.5.

6.4 Function Blocks and Configuration

The Common IP Address Management function uses several functions and function blocks to evaluate the CODESYS redundancy state, manage the IP configuration of the local controller, and perform the additional reachability check required for collision avoidance in Standalone operation.

The following table provides an overview of the main interfaces used by **PRG_IPAliasManagement**.

Function / Function Block	Purpose in the Reference Application
GetRedundancyState()	Retrieves the current CODESYS redundancy state of the local controller.
GetPlcIdent()	Retrieves the fixed physical identity of the local controller.
FB_IpFirstVsAlias	Provides the Common IP and verifies that the requested address is not already used by another device.
FB_IpChange	Applies the individual IP address assigned to the local controller.
FB_IsIpExtern	Verifies whether PLC1 is reachable during the Standalone collision-avoidance procedure.

GetRedundancyState() and **GetPlcIdent()** are common interfaces of the CODESYS redundancy system and are already described in Chapter 5.4. Their meaning is identical within the Common IP Address Management function: **GetPlcIdent()** identifies the physical controller, whereas **GetRedundancyState()** determines its current redundancy role.

The detailed interaction of the function blocks with the IP Alias Management state machine is described in Chapter 6.3.

FB_IpFirstVsAlias

FB_IpFirstVsAlias is responsible for providing the Common IP address on the configured Ethernet interface.

During initialization of **PRG_IPAliasManagement**, the function block is parameterized using:

- the Common IP address,
- the subnet mask associated with the Common IP,
- the Ethernet interface name, and
- the configured ping timeout.

The corresponding parameters are taken from **GVL_IpAlias**:

```

c_OPCUAClientExpectsIpAddress
c_OPCUAClientExpectsIpAddressMask
c_strEthItfName
c_udiPingTimeout

```

The parameter assignment is performed during the **init** state before the function block is enabled.

FB_IpFirstVsAlias is primarily used in the **offer_server** state described in Chapter 6.3. When the local controller is permitted to provide the Common IP, the state machine activates the function block by setting its execution input.

The function block does not simply assign the configured IP address. It also verifies whether the address is already present elsewhere in the network. This behavior is important because the Common IP may already be provided by the other redundant controller.

If the Common IP is available, the function block completes its operation and the local controller continues to provide the address.

If the function block detects that the Common IP is already used by another device, it reports an error condition. The reference application evaluates the reported reason and treats the specific condition

**Note:**

Recognized IP address already exists in the network and is not locally set

differently from a general application error. This condition is interpreted as an indication that another controller already provides the Common IP. The function block is reset and the local controller returns to passive operation through the `clear_fbIpFirstVsAlias` state.

The handling of this special condition within the state machine is described in Chapter 6.3. General function block errors and diagnostic reporting are described in Chapter 6.5.

FB_IpChange

FB_IpChange is responsible for applying the individual IP configuration of the local physical controller.

During initialization, the parameter structure is configured using the fixed PLC identity:

```
GVL_IpAlias.c_IpAddresses[piPLCID]
GVL_IpAlias.c_IpAddressMasks[piPLCID]
```

The Ethernet interface and ping timeout are configured using:

```
GVL_IpAlias.c_strEthItfName
GVL_IpAlias.c_udiPingTimeout
```

This means that the same application code can be executed on both controllers while the fixed PLC identity determines which individual IP configuration is applied locally.

FB_IpChange is primarily used in the **passive** state. When the controller shall no longer provide the Common IP, the function block is executed to ensure that the individual controller IP is active.

The state machine waits until the operation has completed and subsequently waits for the function block to return to its Standby state before evaluating the next redundancy-state transition.

The detailed execution sequence is described in the **passive** state section of Chapter 6.3.

FB_IsIpExtern

FB_IsIpExtern is reused from the Local Modbus TCP Link Monitoring function described in Chapter 5.4.

The function block performs an IP-based reachability check and indicates whether the configured address belongs to a reachable external device. Its general behavior, outputs, runtime requirements, and limitations are therefore not repeated in this chapter.

Within the Common IP Address Management function, the purpose of **FB_IsIpExtern** is different from its use in the Local Modbus TCP Link Monitor.

The target IP address is configured with the individual IP address of PLC1:

```
stIsIpExternParams.arbIPtoTest := GVL_IpAlias.c_IpAddresses[1];
```

and the timeout is configured using:

```
stIsIpExternParams.udIPingTimeout := GVL_IpAlias.c_udIPingTimeout;
```

The reachability check is relevant only to the Standalone collision-avoidance logic of PLC2.

PLC2 uses the result to determine whether PLC1 is still present in the surrounding network:

FB_IsIpExtern Result	Interpretation in Common IP Management
q_xIpIsExtern = TRUE	PLC1 is reachable. PLC2 must not retain or take over the Common IP.
q_xIpIsExtern = FALSE	PLC1 is not reachable. PLC2 may provide the Common IP if the remaining state-machine conditions permit it.
q_xError = TRUE	The reachability check itself failed and the state machine enters error handling.



Note:

As described in Chapter 5.4, **FB_IsIpExtern** verifies IP reachability. It does not verify the functional state of an application protocol. Within Common IP Address Management, the result is therefore used only as an indication of whether PLC1 is reachable through its individual IP address.

Configuration Parameters

The IP Alias Management configuration is centralized in **GVL_IpAlias**.

The following parameters are used by **PRG_IPAliasManagement**:

Parameter	Purpose
c_OPUClientExpectsIpAddress	Common IP address used by the higher-level communication partner.
c_OPUClientExpectsIpAddressMask	Subnet mask associated with the Common IP.
c_IpAddresses[1]	Individual IP address of PLC1.
c_IpAddresses[2]	Individual IP address of PLC2.
c_IpAddressMasks[1..2]	Subnet masks associated with the individual controller IP addresses.
c_strEthItfName	Ethernet interface on which the IP configuration is managed. (Appearance expl.: "eth-x4")
c_udiPingTimeout	Timeout used by the IP-related reachability checks.
c_tIPCollisionCheckTime	Interval used before performing the Standalone IP collision check.

The configured values are copied into the individual function block parameter structures during the **init** state.

The IP collision timer is executed using:

```
tonIPCollisionCheck(
    IN := xIPCollisionCheck,
    PT := GVL_IpAlias.c_tIPCollisionCheckTime
);
```

The collision-check interval determines how frequently PLC2 verifies the presence of PLC1 while Standalone coordination is required. The timer is reset whenever the CODESYS redundancy state changes because an ongoing check is no longer considered valid in the context of the new redundancy state.

The Common IP, individual controller IP addresses, subnet masks, and Ethernet interface name must be adapted to the target network before the reference application is used in another system.



Note:

The Common IP and both individual controller IP addresses must be unique within the intended network configuration. The collision-avoidance mechanism described in Chapters 6.2 and 6.3 reduces the risk that both redundant controllers provide the Common IP simultaneously, but it does not replace correct network and IP address planning.

Required Runtime Configuration

The IP management function blocks execute operating-system-level network operations on the controller. The runtime configuration therefore has to permit the required system commands.

The required runtime permission is the same as for **FB_IsIpExtern** described in Chapter 5.4 and Chapter 5.5 and is therefore not repeated in detail here.

When the reference application is transferred to another controller, runtime version, or network configuration, the required permissions and the configured Ethernet interface name must be verified before enabling Common IP Address Management.

6.5 Error Handling and Recovery

PRG_IPAliasManagement contains dedicated error handling to distinguish between expected IP-collision conditions and errors that prevent the Common IP Address Management function from continuing its intended operation.

A detected use of the Common IP by another device is not always treated as a general application error. If **FB_IpFirstVsAlias** reports that the Common IP already exists elsewhere in the network, the reference application interprets this condition as part of the Standalone collision-avoidance mechanism. The function block is reset through **clear_fbIpFirstVsAlias**, after which the local controller returns to passive operation.

Other errors reported by **FB_IpFirstVsAlias**, **FB_IpChange**, or **FB_IsIpExtern**, abnormal CODESYS redundancy states, and undefined state-machine conditions cause a transition to the error state.

Condition	Handling
Common IP already exists elsewhere in the network	Expected collision-avoidance condition → clear_fbIpFirstVsAlias
FB_IpFirstVsAlias reports another error	Transition to error
FB_IpChange reports an error	Transition to error
FB_IsIpExtern reports an execution or internal error	Transition to error
CODESYS redundancy state indicates an abnormal condition	Transition to error
Undefined IP Alias or FSM action state	Transition to error

Error Detection and Diagnostic Information

When the state machine enters **error**, the application determines the error source and stores the available diagnostic information in **GVL_IPAliasStatus**.

If one of the IP-related function blocks reports an error with valid diagnostic information, its **q_stError** structure is copied to:

GVL_IPAliasStatus.stError

The redundancy state that was active when the error occurred is additionally stored in:

GVL_IPAliasStatus.rsErrorRedundancyStates[piPLCID]

and the common error flag is set:

GVL_IPAliasStatus.xError := TRUE

The affected function block is disabled before the state machine continues with the cyclic error-handling logic.

If the error did not originate from one of the function blocks, the internally generated **stError** structure is stored instead. This is used, for example, for invalid redundancy states or undefined state-machine conditions.

The state machine additionally stores the following diagnostic context:

Variable	Stored Information
etLRCStateError	IP Alias state in which the error was detected
etFSMActionError	FSM action (_entry , _do , or _exit) active when the error occurred
rsRedundancyStateError	CODESYS redundancy state at the time of the error
stError	Detailed application-level error information

Automatic Recovery from Redundancy State Errors

Errors caused by an abnormal CODESYS redundancy state are treated separately from function block and internal application errors.

If a redundancy state caused the transition to error, the internal flag:

xRedundancyStateRaisedError

is set.

While the IP Alias Management function remains in the error state, the current CODESYS redundancy state is continuously evaluated. Automatic recovery is permitted when the redundancy system subsequently reaches one of the following states:

- **RS_CYCLE_ACTIVE**
- **RS_CYCLE_STANDBY**
- **RS_CYCLE_STANDALONE**
- **RS_SHUTDOWN_ACTIVE**
- **RS_SHUTDOWN_STANDBY**

When one of these states is detected, the application generates an internal error acknowledgment and starts reinitialization of the IP Alias Management function.

Error Acknowledgment and Reinitialization

The recovery process is triggered by a rising edge of the error acknowledgment signal.

When an acknowledgment is detected, the stored diagnostic information is cleared:

- **GVL_IPAliasStatus.xError** is reset.
- The stored redundancy state is reset to **RS_START**.
- The stored redundancy error is reset to **NO_ERROR**.
- The redundancy error text is cleared.
- The application error module, source, reason, parameters, and timestamp are reset.
- **xErrorDataValid** is reset.

The IP-related function blocks are then disabled and reset:

- **fbIpFirstVsAlias**
- **fbIpChange**
- **fbIsIpExtern**

Finally, the state machine returns to:

ET_LRCState.init

with:

ET_FSMAction._entry

The complete Common IP Address Management function is therefore initialized again before normal operation resumes.



Note:

Automatic recovery is only generated for errors that were raised by the CODESYS redundancy state. Errors originating from **FB_IpFirstVsAlias**, **FB_IpChange**, **FB_IsIpExtern**, or internal state-machine errors do not automatically trigger the same recovery path in the shown implementation. If manual recovery of such errors is required, the acknowledgment concept should be reviewed and adapted for the target application.