acontis technologies GmbH

SOFTWARE

EC-Master

EtherCAT® Master Stack
for embedded Operating Systems

Master Redundancy

Version: 1.16
Date: 2020-07-23

Document history

Version	Date / Chapter	Author	Comment / Changes
1.0	27 Feb 15	PBN	Created
1.1	16 Mar 15	PBN	Added *Feature Overview* (mandatory and optional) Added *Collision Detection* description and that *each Link Layer Instance must have a unique MAC address*. Added *Failure Detection* with *watchdog*. Added *Acyclic Master-To-Master Communication*. Added *example* why *Cyclic Shadow Frame* are needed. Added *impact to bandwidth* for *Cyclic Shadow Frame*. Added INACTIVE Master *reads* Process Data. Added *Master-To-Master* description in *Process Data*. Added *RAS cycle time* (RAS API latency) is configurable. Added *EC-Master parameters* and *ENI* files must be identical.
1.2		PBN	Complete Revision Shadowing in Cyclic Master Red Frames only for Slave Process Data, not for Master-To-Master Process Data. Added API Topology restriction: Slaves may be connected on one line only. Switchover is up to the application. Failure detection is up to the application. Reviewed Master-To-Master acyclic communication. Renamed *Cyclic Shadow Frame* to *Cyclic Master Red Frame*.
1.3		PBN	New chapter "Getting started" and "Proof of Concept"
1.4	27 Jan 16	PBN	Refining Proof Of Concept, Shadow Process Data Added Recovery from Failover
1.5	03 Jun 16	PBN	Added Frame Processing at Failover FSOE Proof Of Concept only needs slave INPUTS exchanged. Added Proof Of Concept Test Network description. Fixed Startup API order Added chapter Frame Processing Added details to Process Data Synchronization Replaced EcMasterDemo, EcMasterDemoSyncSm with *EcMasterDemoMasterRed .* Removed *ecatConfigureMasterRed.* Renamed *eRspErr_WRONG_SRC_MAC* to *eRspErr_FOREIGN_SRC_MAC*

1.6		**PBN**	**Added requirement list**
1.7		**PBN**	**Clean-up**
1.8		**PBN**	**Added hint about configuring ENI without LRW.**
1.9		**PBN**	**Slave OUTPUT Process Data is part of CSF, not CMF.** **Update MasterRed parameters (EC_T_MASTER_RED_PARMS)** **Removed Proof-Of-Concept Test network description** **Add Interrupt Mode used (removed ProcessAllRxFrames)** **Revision of chapter 12 "Frame Processing"**
1.10		**PBN**	**Update MasterRed parameters**
1.11		**PBN**	**Remove PoC references** **Review synchronization descriptions**
1.12		**PBN**	**Review requirements** **Add restrictions** **Remove PoC references** **Add denied APIs at INACTIVE**
1.13		**PBN**	**Review Getting Started**
1.14		**PBN**	**Update MasterRed parameters**
1.15		**PBN**	**Add circulating frame prevention (Master Forwarding Bit Mask 0x10)**
1.16		**PBN**	**Circulating master red frame prevention (Master Forwarding Bit Mask 0x84)**

Content - compact

Content

Figures

Literature

[1] Beckhoff Automation GmbH, „ET1100 Hardware Data Sheet (Version 1.7),“ Eiserstr. 5, 33415 Verl, 2010.

[2] acontis GmbH, „EC-Master Class B Documentation“.

[3] acontis GmbH, „Feature Pack EoE Endpoint Interface Manual“.

[4] wikipedia, „TUN/TAP,“ 2015 Februar 21. [Online]. Available: http://en.wikipedia.org/wiki/TUN/TAP. [Zugriff am 2015 Februar 27].

1 Definitions

1.1 Terms

Absent Slave	*Config Slave* not found on *Bus*.
ACTIVE	Key value of *Master Redundancy State*
Acyclic Master Red Frames	*Frames* sent by *ACTIVE* to *INACTIVE Master* for acyclic Master-to-Master communication.
Application	*EC-Master Application*
Application Layer Status	Enumeration of *INIT, BOOT, PRE-OP, SAFE-OP, OP* of the *ESM*.
Arbitration	Procedure leading to decision **which** *Master* must become/be *ACTIVE*.
Bus	EtherCAT Network
Bus owner	*Master* that won *Arbitration*, got *ACTIVE* and may *control the bus*.
Bus scan	Get or update topology knowledge by communication with *Slaves*.
Bus Slave	*Slave* that can be addressed by *Cmds*.
Connection to Master	*Master-Slave*-Communication: *Slave* can (directly or indirectly) receive *Frames* from *Master*. *Application-Master*-Communication: *RAS* connection via TCP.
Control the Bus	Send *Cmd*s to the *Bus* to directly interact with *Slaves*.
Config Slave	*Slave* configured in ENI
Cyclic Master Red Frames	*Frames* sent cyclically by *ACTIVE* to *INACTIVE Master* containing *Shadow Process Data* and Master Redundancy Process Data.
Cyclic Slave Frames	*Frames* sent by *ACTIVE Master* to Slaves containing Process Data. Forwarded by *INACTIVE Master.*
EC-Master Application	Program executing EC-Master API functions.
EC-Master Instance	Stack Instance controlled by *Application*(s) and interfacing *NIC*(s).
EtherCAT device	*Slave* or *Master*.
EtherCAT Network Information	EtherCAT XML Master Configuration
Ethernet source address semantic	Merge of MAC address and *Master State* information.
Failover	Switching from previously *ACTIVE Master* to *STANDBY Master* upon the failure of previously *ACTIVE Master*.
Frame	EtherCAT/Ethernet Frame.
Frame loss	*Frame* sent by *Master* but not received in next cycle or lost at all.
Job Task	Cyclical execution of *Master APIs* (“Jobs”) to send, receive and process frames.
Hybrid Access	Executing *Master APIs* from EC-Master Library and RAS Client Library.
INACTIVE	Key value of *Master Redundancy State*
Isolated Slave	*Slave* without *Connection to Master*.
Link Layer Instance	*NIC* exclusively assigned to and interfaced by a *Master*.
Log Message	Hint from *Master* to *Application* as formatted, zero-terminated char array of arbitrary length.
MAIN Frame	*Frame* sent on first *Link Layer Instance*.
Master API	Functions exported from the EC-Master Library (EcMaster.dll) to control an *EC-Master Instance*.
Master State	State of the *Master*’s internal *ESM*
Master	*EC-Master Instance*
Master Redundancy Configuration	Set *Master Redundancy Parameters* to *Master* in order to apply *Master Redundancy*.
Master Redundancy Parameters	Structured parameter set needed for *Master Redundancy Configuration*.
Master Redundancy State	Enumeration of *ACTIVE, INACTIVE*
Master-To-Master Process data	*ACTIVE to INACTIVE / INACTIVE to ACTIVE Master Process Data*

Message	Implemented as *Notification* or *Log Message*.
Network TAP	Virtual network adapter to OS, freely available from OpenVPN.
Notification	Parametrized *Message* of well-defined length identified by ID. Typically, but not necessarily raised by the *Master*. *Application*(s) can subscribe to by registering as client.
Master Redundancy	Appliance implementing e.g. *Failover* to increase system's availability.
Present Slave	*Config Slave* found on *Bus*.
Port	*NIC* at *Master* or Port A, B, C or D *at ESC.*
Port Link	Direct connection between two *Port*s.
Process Data Image	RAM allocated by each *Master* and cyclically synchronized with *Slaves* and other *Master*. Size defined in ENI.
RED Frame	*Frame* sent on second *Link Layer Instance*.
Redundancy Frame ID	Used for *Split and Merge*.
Shadow Process Data	Slave Process Data from last cycle as result from *Split and Merge.*
Slave	EtherCAT slave containing an *ESC* connected to the *Bus*.
Set Master Redundancy State	Request *Master Redundancy State* change, e.g. in order to become *ACTIVE* or *INACTIVE*.
Split and Merge	The Master uses *RED Frames* **and** *MAIN Frames* to *Control the Bus*.
STANDBY Master	*INACTIVE Master* ready for *Takeover*.
Store and Forward	Receive a complete *Frame*, evaluate (modify) it and forward it.
Switchover	Intentionally switching from previously *ACTIVE Master* to *STANDBY Master* **not** upon the failure of previously *ACTIVE Master*.
Stop	Becoming *INACTIVE* or *terminate*.
System	Set of *Applications*, *Masters*, *Slaves* interacting with each other (HW, SW)
Takeover	Becoming *ACTIVE*.
Terminate	Exit *Application*.
Topology Knowledge	Information about present *Slaves*' presence and mismatching *Slaves* including *Port Link*s.
Trace Data	Process Data in Cyclic Slave Frames, but not addressed to slaves.
Visible EtherCAT device	*EtherCAT device* connected to the *Bus* receiving and processing *Frame*s. → In case of *Frame Loss* there may be *Slave*s that are not visible, but receiving and processing *Frame*s.

1.2 Abbreviations

AL	Application Layer
AL Status	*Application Layer Status*
BOOT	Bootstrap State (ESM)
Cmd	Command (EtherCAT)
CMF	Cyclic Master Red Frames (s.a. CSF)
CSF	Cyclic Slave Frames (s.a. CMF)
EC	EtherCAT
ECAT	EtherCAT
ENI	EtherCAT Network Information
ESC	EtherCAT Slave Controller
ESM	EtherCAT State Machine
EoE	Ethernet over EtherCAT
FoE	File access over EtherCAT
FPRD	Configured Address Physical Read (EtherCAT command)
INIT	Init State (ESM, Master Redundancy State)
Msg	Message, e.g. log msg from master to application.
NIC	Network Interface Card
OP	Operational State (ESM)
OS	Operating System
PoC	Proof of Concept
PREOP	Pre-Operational State (ESM)

PD	Process Data
PD Image	Process Data Image
RAS	Remote Access Service
TAP	Network tap
SAFEOP	Safe-Operational State (ESM)
WKC	Working Counter

2 Overview

2.1 Master Redundancy Architecture

EtherCAT systems provide a high level of availability even in case of device failures. A typical EtherCAT system with cable redundancy for high availability consists of a dual-NIC Master Device and Slaves:

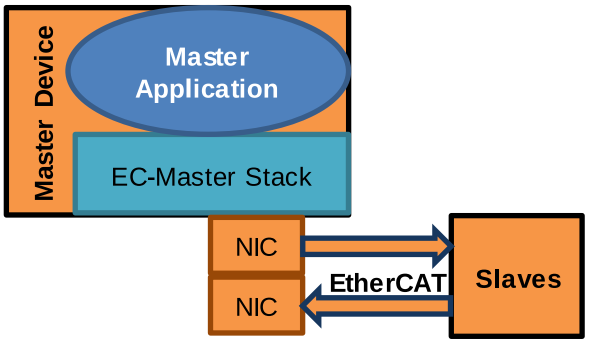


Figure 1: A typical high available single-Master System (Cable Redundancy)

Without Master Redundancy, the Master device exposes a single point of failure to the system in case of failing which affects the availability of the complete EtherCAT system:

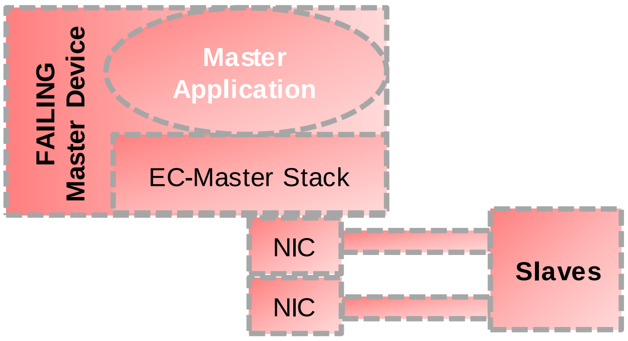


Figure 2: System failure in case of Master Device failure

In order to increase the availability of the EtherCAT system, a redundant Master capable for Failover is needed as backup. The Backup Master is INACTIVE until it needs to takeover and control the bus. The following diagram shows two redundant Master Devices. Each Master Device has two network interface cards (NICs) for EtherCAT communication:

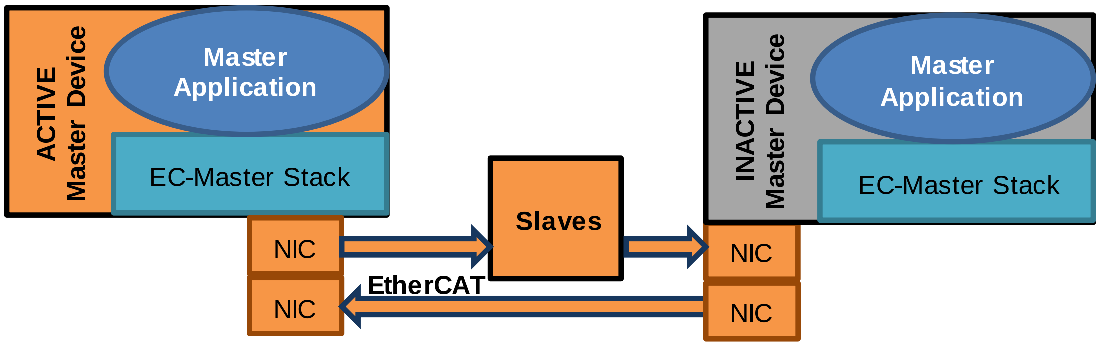


Figure 3: Schematic overview of dual-Master (Master Redundancy)

If the ACTIVE Master fails, the Backup Master can takeover. Because the INACTIVE Master is connected to the network, it can control the bus in case of Failover:

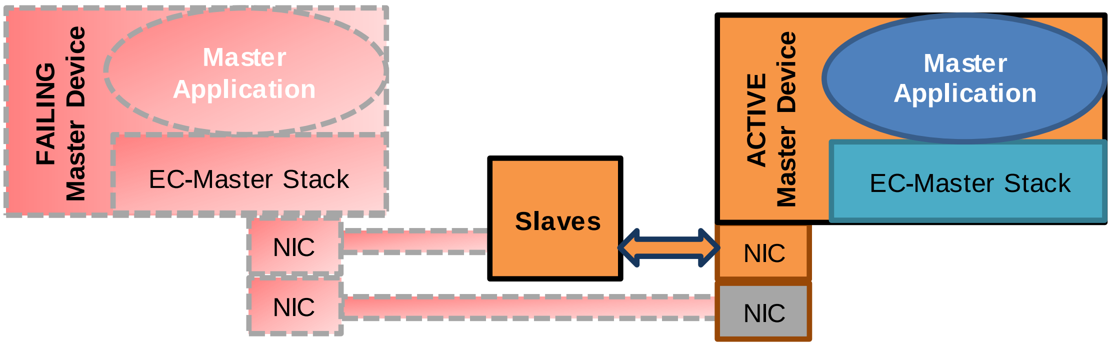


Figure 4: Failover: The Backup Master takes over.

2.2 Master Device Architecture

Each Master Device consists of the Master application and the EC-Master stack as shown in the following figure:

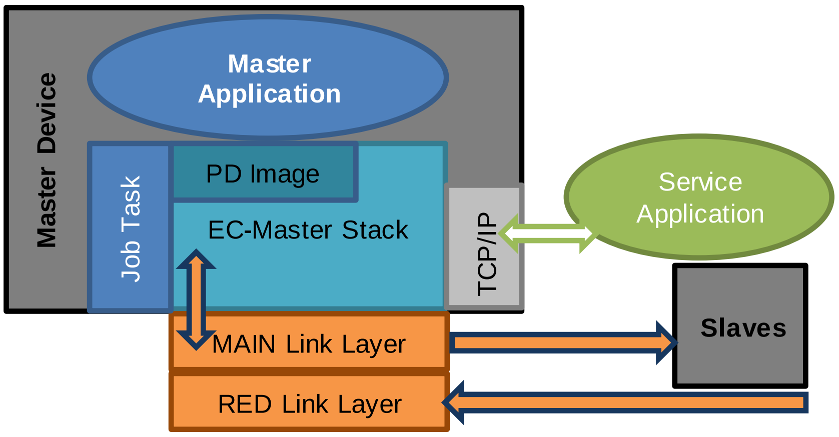


Figure 5: Schematic overview of a Master Device

2.3 Features

2.3.1 Master Redundancy State

A Master on the network can be ACTIVE or INACTIVE. ACTIVE means that this Master controls the bus. The Application can get or set its Master's Redundancy State at any time using an EC-Master API.

2.3.2 Failover and Switchover

The transition from INACTIVE to ACTIVE is called Takeover.
The application can initiate the Takeover by calling an EC-Master API. The EC-Master automatically changes to ACTIVE / INACTIVE upon Application's request.

Takeover is called **Failover** if the previously ACTIVE Master failed and the previously INACTIVE Master got ACTIVE and no INACTIVE Master remains.

Takeover is the key functionality of Master Redundancy.

Failover implies the need to **detect the failure** which must be implemented by the Application.

Takeover is called **Switchover** if there is no error at the ACTIVE Master, but the Application switches the ACTIVE Master to INACTIVE and the INACTIVE Master to ACTIVE. Switchover is needed as one way to prove that failover is possible. The Applications can do a Switchover by coordinated setting the Masters to ACTIVE and INACTIVE.

2.3.3 "Hot Standby", "Warm Standby" and "Cold Standby"

"Hot Standby" means that a Switchover or Failover from STANDBY to ACTIVE is processed usually without notice to the slaves.

"Warm Standby" means that the devices recognize the failover, typically because there are several steps, like scanning the bus, needed in order that the INACTIVE Master becomes ACTIVE. The decision between the redundancy types "Hot Standby" and "Warm Standby" is determined by the Application which is responsible to detect failure and control the reaction. Going through EtherCAT States INIT, PREOP, SAFE-OP, OP leads to "Warm Standby".

"Cold Standby" means that there is an offline device, a spare part to replace the failing component changed manually.

2.3.4 Store and Forward

The INACTIVE Master receives and forwards frames from the ACTIVE Master. See also Figure 9: Job Task timing.

2.3.5 Master State Synchronization

The INACTIVE Master cyclically gets the current EtherCAT state from the ACTIVE Master.

2.3.6 Process Data Synchronization

The INACTIVE Master cyclically gets the Process Data from the ACTIVE Master. The ACTIVE Master can also cyclically exchange Master specific Process Data with the INACTIVE Master.

2.3.7 Acyclic Master-To-Master Communication

The Masters can acyclically communicate which each other.

2.3.8 Collision Detection

The EC-Master detects if another Master sends frames to the network. This is called a collision and the EC-Master reports it to the Application

3 Specification

3.1 Requirements

The following requirements are referenced in this document:

No	Requirement	Response[1]
REQ001	The ACTIVE Master must control the bus	Master
REQ002	Expose Master Redundancy State	Master
REQ003	Support changing Master Redundancy State	Master
REQ004	Choose one of the Masters as Bus owner (Arbitration) (see also REQ035)	Application
REQ005	If the Master system fails there must be a backup Master that can control the bus	System
REQ006	Layer Instance must have a unique MAC address	System
REQ007	The Link Layer must support Interrupt Mode.	Master
REQ008	Configure a Network TAP with unique MAC and IP Address assignment at each Master	System
REQ009	Provide parameters for operation	Application
REQ010	EC-Master parameters and ENI files must be identical	Application
REQ011	Port 0 must be connected	System
REQ012	Links at failing Master must go down	System
REQ013	Slaves must be connected on same line	System
REQ014	Become ACTIVE/INACTIVE on request (see also REQ015)	Master
REQ015	Request ACTIVE/INACTIVE	Application
REQ016	Request INACTIVE on collision	Application
REQ017	Report collision	Master
REQ018	Detected communication time-out	Application
REQ019	Support Interrupt mode	Master
REQ020	Configure Interrupt mode (polling mode not supported).	Application
REQ021	Listen if another master sending frames	Application
REQ022	Perform the appropriate activities based on the INACTIVE/ACTIVE State.	Master
REQ023	Call OnMasterTimer exactly once per cycle	Application
REQ024	Call ecatSetMasterState at Bus owner	Application
REQ025	Report failing Master return	Master
REQ026	Handle failing Master return	Application
REQ027	The Master-Master Frame must be used to guarantee consistency	Master
REQ028	Report Process Data communication errors	Master

[1] See chapter 1.1 “Terms” for declaration of *Master / Application / System*.

REQ029	Consistency and integrity constraints checks of datagrams	Master
REQ030	Consistency and integrity constraints checks of the overall Process Data Image	Application
REQ031	The INACTIVE EC-Master must synchronize the EtherCAT State from the Cyclic Master Red Frame sent by the ACTIVE EC-Master	Master
REQ032	Provide current EtherCAT State at ACTIVE and INACTIVE Master (ecatGetMasterState)	Master
REQ034	Communication between the Masters must be possible at any state without additional hardware needed to provide a TCP/IP connection between the ACTIVE and the INACTIVE Master	Master
REQ035	ACTIVE Master (else EtherCAT frame sending stops) (see also REQ014)	System
REQ036	Register an EoE endpoint as network adapter	Application
REQ037	Start RAS Server	Application
REQ038	INACTIVE Master must be always connected to the network	System
REQ039	Reduce TAP adapter's MTU	System
REQ040	Encapsulate TAP communication (acyclic frames)	Master
REQ041	Configure Master Redundancy (ecatInitMaster)	Application
REQ042	Retry denied mailbox transfers (e.g. Switchover)	Application
REQ043	Slave State Sync	Master
REQ044	Hot Connect	Master
REQ045	Diagnosis Image Sync	Master
REQ046	Support *Master-To-Master Process data* hosting with Memory Provider	Master
REQ047	ENI without LRW	System
REQ048	Handle merging of two independent EtherCAT networks	Application

3.2 Design Features

Master Redundancy needs and provides the following set of features, referenced in this document:

No	Feature	Chapter
1	Master Redundancy State	3.3
2	Failover	3.6
3	Hot Standby	2.3.3
4	Store and Forward	4.4, 6.1
5	Master State Synchronization	4.1
6	Slave Process Data Synchronization	4.1, 4.4
7	Acyclic Master-To-Master Communication	4.4, 5.4
8	Remote Access	5.5

9	Cable Redundancy	6.1.2
10	Collision Detection	3.4
13	ACTIVE to INACTIVE Master Process Data	4.1
14	INACTIVE to ACTIVE Master Process Data	4.1

3.3 Master Redundancy State: Current state and Control

The behavior of the EC-Master is determined by its Master Redundancy State which can be either ACTIVE or INACTIVE. Initial value is INACTIVE and can be set to ACTIVE by the Application.

If the Master system fails there must be a backup Master that can control the bus (REQ005). The Master that must control the bus (REQ001) is called the ACTIVE Master (Bus owner) and the backup Master is called INACTIVE Master.

The EC-Master exposes its current Master Redundancy State (REQ002) and a means to change it by the API ecatGetMasterRedState and ecatSetMasterRedStateReq (REQ003, REQ014). Changes are signalled with notification EC_NOTIFY_MASTER_RED_STATECHANGED, for e.g. diagnostic over RAS.

- Calling ecatSetMasterRedStateReq with same state again and state didn't change will have no effect.
- Calling ecatSetMasterRedStateReq more often than once per cycle makes no sense.
- Collisions are detected within one cycle.
- MACs are NIC dependent and therefore will not change on reboot (only if you replace the NIC).
- Foreign Frames are only forwarded if the Master is INACTIVE. So the application must set the Master INACTIVE on collision to only lose frames from one cycle and not from several cycles.
- Even in case of collission, frame is still provided to application

3.4 Exclusive bus control

It is not possible to concurrently control the bus. One Master must exclusively control the bus (REQ001, REQ015). The EC-Master detects collisions in case of concurrent bus control and sends the notification EC_NOTIFY_FRAME_RESPONSE_ERROR with parameter eRspErr_FOREIGN_SRC_MAC to the Application (REQ016, REQ017). Collision Detection needs each Link Layer Instance to have a unique MAC address (REQ006).

If there are more than one Master able to control the bus, Arbitration is needed to determine which Master must Control the Bus (Bus owner) the other Master may not control the Bus and stay INACTIVE. The application must choose one of the Masters as Bus owner and set it ACTIVE and the other to INACTIVE issuing ecatSetMasterRedStateReq, e.g. by waiting a random delay before Takeover, see below (REQ004).

If both masters are INACTIVE, no master-Master communication is possible.

3.5 Failure Detection

The ACTIVE Master cyclically sends frames to the INACTIVE Master. If it stops sending frames it is a symptom for system failure. The INACTIVE Master cannot know if the communication from the ACTIVE Master stops, this must be detected by the application (REQ018). See also Implementation Hints.

3.6 Startup

A starting Master cannot know if it is the first one or not so the startup procedure for all Masters on the network is the same (REQ022, REQ023, REQ024, REQ027, REQ028, REQ029, REQ030). It is as follows:

- Call ecatInitMaster with Link Layer and Master Redundancy parameters (REQ009, REQ010). Initial state is INACTIVE with automatic Store and Forward as soon as link layers are opened. The Master parameters must be identical at each Master (REQ010).

- Start JobTask cyclically calling ecatExecJob and enable EtherCAT link to other EtherCAT devices
- Call ecatConfigureMaster with ENI file
- Master Redundancy requires use of interrupt mode (REQ007, REQ019, REQ020). The Application must provide a call back function. Call EC_IOCTL_REGISTER_CYCFRAME_RX_CB to register the callback function to trigger on ACTIVE Master's communication at application level of ACTIVE and INACTIVE Master.
- Call ecatRegisterClient to enable collision detection (REQ016)
- The Master reports collisions to the application. The application must arbitrate (REQ004).
- The Application must, before trying to get active, listen on the bus if there is another master sending frames (REQ021).

Store and Forward and Master-To-Master communication is now established. On System startup there is no ACTIVE Master.

3.7 Expected behaviour

The ACTIVE Master automatically polls the INACTIVE Master for information. The ACTIVE Master does not forward frames.

The INACTIVE Master does not send frames on its own, but forwards frames from ACTIVE Master. Master to Master communication is polled like Slave to Master communication.

Master Redundancy State change requests are applied when the Application's JobTask calls OnMasterTimer. The Application's JobTask must call OnMasterTimer exactly once per cycle therefore the delay of applying the Master Redundancy State change request is at worst case 1 cycle.

INACTIVE Master denies acyclic communication to slaves, because it may not control the bus.

On Takeover, Mailbox transfers may be denied by the ACTIVE Master until it knows the slave is present. The first call may be not successful, because Mailbox communication was interrupted by the Takeover at the slave. The Application can retry at a later time, at earliest at the next cycle. (REQ042)

⚠ CAUTION: Don't interrupt a slave firmware update, it may destroy the device!

3.8 Failover

To Takeover (becoming ACTIVE) the application must call:

- ecatSetMasterRedStateReq(EC_TRUE) to become ACTIVE
- ecatSetMasterState, e.g. OPERATIONAL if needed, e.g. at startup

If the ACTIVE Master fails, the RED port gets deactivated and following the EtherCAT standard the frames from MAIN return on MAIN:

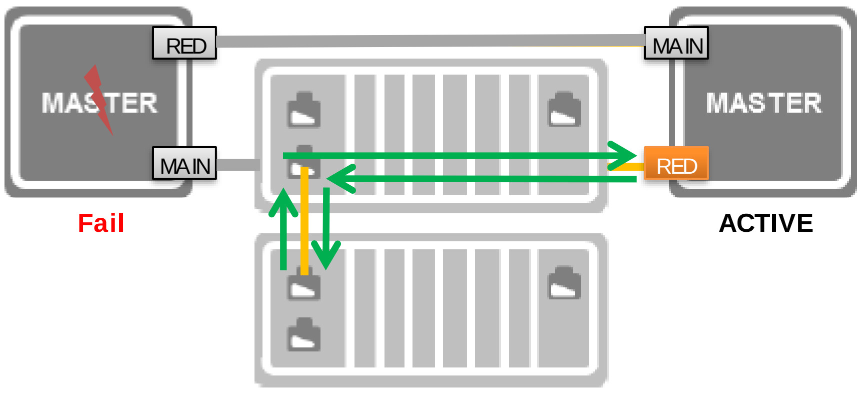


Figure 6: Failover Frame Processing Order

3.9 Recovery from Failover

In case the failing Master returns (REQ025, REQ026), the Application at the INACTIVE Master will notice the communication from the ACTIVE Master and the INACTIVE Application must stay INACTIVE.

In case both Masters got separated in two independent EtherCAT networks by a double line break as shown below, the recovery from Failover by repairing the cables, will lead to collision with two ACTIVE Masters which must (REQ048) be handled by the Application at both ACTIVE Masters:

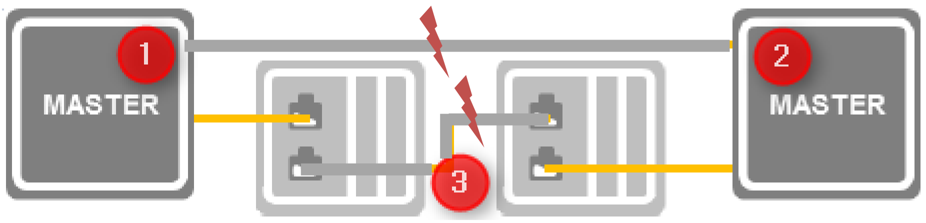


Figure 7: Failover dividing slaves into independent EtherCAT networks

3.10 Restrictions

The following known restrictions apply to Master Redundancy:

- Polling Mode is not supported, because the INACTIVE Master must forward frames in the same cycle.
- Cable Redundancy is mandatory for Master Redundancy. This implies restrictions to ENI.
- At INACTIVE Master WKC mismatches are not notified, but mismatches can be alternatives detected using the Diagnosis Image (REQ045).
- All Slaves Must Reach Master State False is only supported at start-up, but not on takeover in PREOP / SAFEOP / OP.
- Distributed Clocks needs special handling for port propagation delay measurement (currently not supported)

4 Master-Master Synchronization

4.1 Process Data areas

The INPUTs and OUTPUTs of Slaves are part of the Process Data Image. The Slaves' Process Data size (cyclic datagrams) and content structure (variables) is defined by the ENI file. The Process Data Image can be extended with *Master-To-Master Process Data* by API, see **MasterRedParms** at *ecatInitMaster*.

Master-To-Master Process data is asymmetrically, bi-directionally implemented as *ACTIVE to INACTIVE Master Process Data* and *INACTIVE to ACTIVE Master Process Data*.

Each Process Data area has a dedicated pointer exposed by API:

Slave OUTPUTs	ecatGetProcessImageOutputPtr
ACTIVE to INACTIVE Master (OUTPUTs)	ecatGetMasterRedProcessImageOutputPtr
Slave INPUTs	ecatGetProcessImageInputPtr
INACTIVE to ACTIVE Master (INPUTs)	ecatGetMasterRedProcessImageInputPtr

The memory is by default allocated by *ecatInitMaster*, but can also be provided by the application, see structure **EC_T_MEMPROV_DESC** at EC_IOCTL_REGISTER_PDMEMORYPROVIDER.

4.1.1 Synchronization

The Application must perform the same sequence on both masters. The Master shall perform the appropriate activities based on the INACTIVE/ACTIVE State.
The Process Data Image is cyclically updated in the following order, see e.g. *EcMasterDemoMasterRed*:

1. *ProcessAllRxFrames* updates the Process Data Image with contents of the last received frames. This contains OUTPUTs to Slaves, INPUTs from Slaves, and INACTIVE to ACTIVE Master Process Data.
   The data was polled by *SendAllCycFrames* within the last cycle.
2. The ACTIVE application changes the OUTPUTs in the Process Data Image, e.g. at *myAppWorkPd*.
3. *SendAllCycFrames* at the ACTIVE Master pushes the following:

- Slave OUTPUTs from the Process Data Image of this cycle
- ACTIVE to INACTIVE Master OUTPUTs from the Process Data Image of this cycle
- Merged Slave INPUTs from the last cycle to the INACTIVE Master
  SendAllCycFrames polls the INPUTs for the next cycle.

4.1.2 Consistency

EtherCAT ensures consistency and integrity constraints on EtherCAT datagram level. Process data is distributed across different datagrams in different frames. The ACTIVE Master merges the received frames from MAIN and RED Link Layer before further processing. The EC-Master must report Process Data communication errors to the Application using EC_NOTIFY_CYCCMD_WKC_ERROR, see *EC-Master Class B Documentation*. The ACTIVE Master sends the merged Slave INPUTs to the INACTIVE Master as *Shadow Process Data* to ensure consistency, see below.

Independent of Master Redundancy: Consistency and integrity constraints checks of the overall Process Data Image that is synchronized across multiple datagrams must be ensured by the Application, as the EC-Master checks on EtherCAT datagram level.

4.1.3 Memory Provider

The EC-Master by default allocates the Process Data Image. The EC-Master alternatively supports registering a Memory Provider so that the Application can allocate the Process Data Image, see *EC-Master Class B Documentation*. Multi-threaded Process Data access by the Application must be synchronized by the Application. This is not Master Redundancy specific and also not needed for single threaded Process Data handling in Job Task.

4.2 State Synchronization

The INACTIVE Master assumes the Master State (INIT, PREOP, SAFEOP, OP) from the Cyclic Master Red Frame in order to immediately continue sending Process Data to Slaves on *Takeover*. Therefor the INACTIVE Application cannot tell the INACTIVE EC-Master about the current EtherCAT State (INIT, PREOP, SAFE-OP or OP). The ACTIVE and INACTIVE Application can get the current EtherCAT State by calling ecatGetMasterState. (REQ031, REQ032)

4.3 Topology Knowledge

Slaves' presence and Slaves' ESM State (INIT, PREOP, SAFEOP, OP) information is stored in the Slave State Image that is automatically synchronized in CMF.
On Takeover the new ACTIVE Master immediately exchanges process data, but it needs some time to validate the Slave State Image to know which slaves are present. The System's requirements constrain the delay until the Application must have the knowledge of Slaves' presence after Takeover with time-out. A regular bus scan is time expensive due to e.g. EEPROM reading from the slaves. Because the Bus Slave Info cache is extensive, bus slaves are recorded explicitly and APIs like ecatGetBusSlaveInfo return EC_E_MASTER_RED_STATE_INACTIVE until the Master has read the Slave's EEPROM which is only possible while the Master is Bus Owner.

4.4 Shadow Process Data

The ACTIVE Master always sends Frames on both Link Layer Instances to implement *Cable Redundancy* and merges the returning data when processing the frames. This is called *Split and Merge*.

The INACTIVE Master implements *Store and Forward*, but in case of a line break, it does not see all INPUT data from Slaves in Cyclic Slave Frames, see 6.1.3. The data from *Cyclic Slave Frames* seen by the INACTIVE Master is therefore in case of line-break inconsistent and incomplete. Therefore the ACTIVE Master sends the merge result as *Shadow Process Data* in *Cyclic Master Red Frames* to the INACTIVE Master on both Link Layer Instances to supply consistent *Process Data*.
The INACTIVE Master reads the Process Data INPUTs, OUTPUTs from the *Shadow Process Data*.

4.5 Cyclic Master Red Frames

The ACTIVE Master sends the *Cyclic Master Red Frames* to the INACTIVE Master containing *Shadow Process Data* from slaves and *Master-To-Master Process Data* when the Application calls SendAllCycFrames. The INACTIVE Master automatically processes (parses) the frames from the ACTIVE Master and returns them.

The *Cyclic Master Red Frames* are conforming to the EtherCAT standard, but do not address any slave. Additionally they contain *Master-To-Master Process Data*. Because *Cyclic Master Red Frames* are additional frames they must be considered for bus load and CPU load.

4.6 EtherCAT Network Information

Because Hot Standby must be implemented the ENIs must match each other, especially in respect of Fixed Address assignments, Topology declaration, Sync Manager Settings, FMMU settings, etc. The EtherCAT Network Information (ENI) is **not** automatically synchronized between the Masters. The Application has to ensure the correct ENI is loaded.

5 Master Device Architecture

5.1 Overview

The following diagram shows the EC-Master integration to the Application.

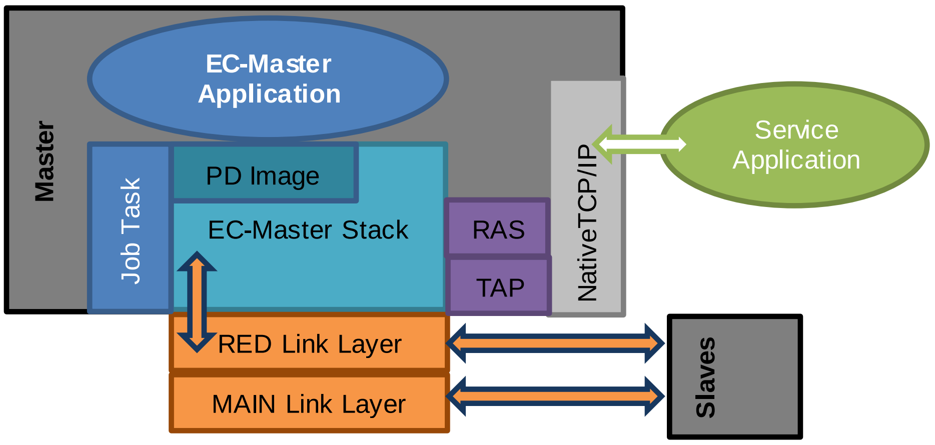


Figure 8: Schematic Master Overview

5.2 Job Task

The JobTask of the ACTIVE Master's application generates the ECAT cycles by a local timer. The frame processing is triggered on frame interrupt (Link Layer Mode: Interrupt). Because both Masters operate on frame receiving interrupt, their *myAppWorkPd* implementations ("App. Task") automatically do not drift. For reference see *EcMasterDemoMasterRed*.

The following figure shows the Job Task timing at ACTIVE and INACTIVE:

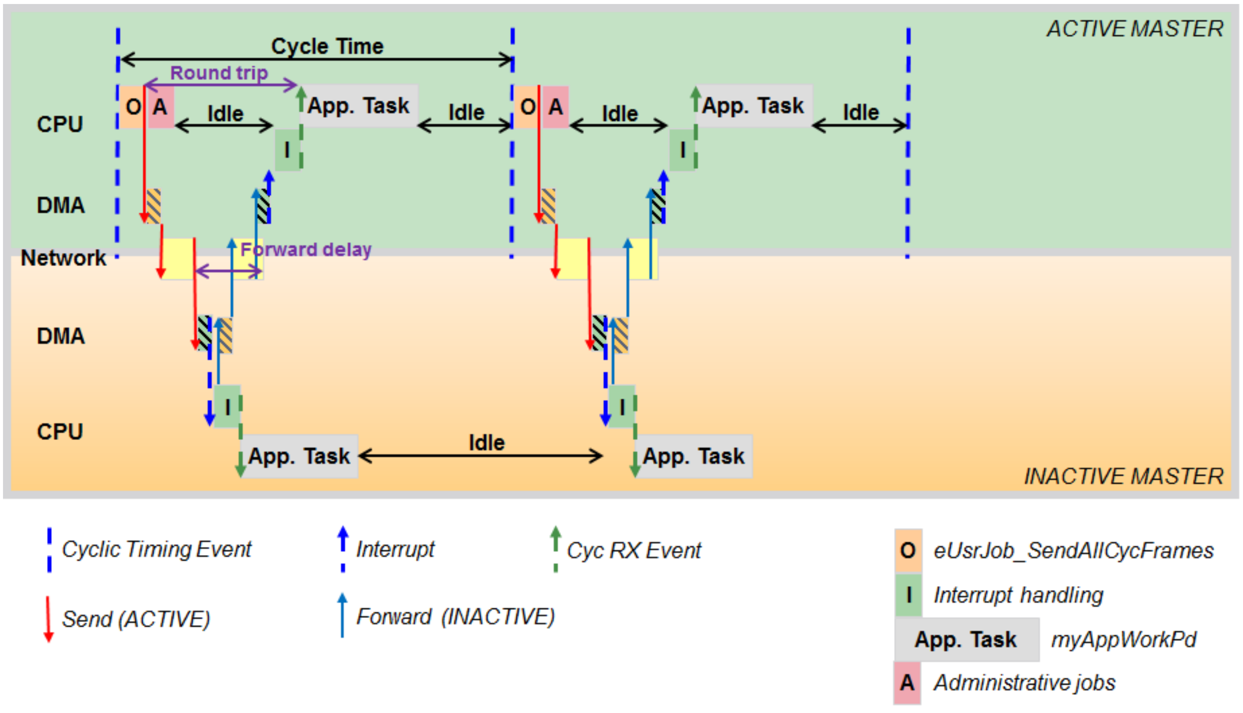


Figure 9: Job Task timing

5.3 Process Data (PD Image)

The Process Data Image contains Slave Process Data and *Master-To-Master Process Data*, see chapter 4.1, “Process Data”. The ACTIVE Application’s Job Task cyclically triggers the EC-Master to send it and the INACTIVE Master reads it cyclically.

5.4 Acyclic Master-To-Master Communication

The ACTIVE Master and the INACTIVE Master can communicate with each other using TCP/IP. Push data from ACTIVE to INACTIVE Master and to pull data from INACTIVE to ACTIVE Master with acyclic frame is part of the “Acyclic Master-To-Master Communication”.
The network load can be tuned by means of the MTU configuration at the TAP adapter.
The Acyclic Master-To-Master Communication data flow is as follows:

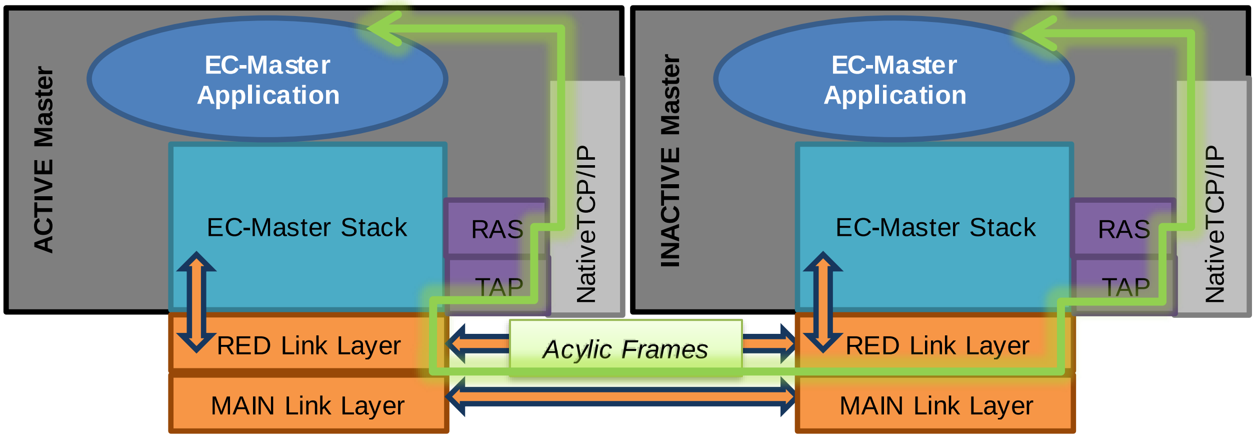


Figure 10: Acyclic Master-To-Master Communication data flow

5.5 Remote Access

5.5.1 Accessing the EC-Master via TCP/IP

It is possible to remotely access the EoE network using the following applications:

- EC-Engineer
- EC-Lyser
- EC-EoE-Gateway
- EC-Master *FP RAS* with *FP EoE Endpoint*

The EC-Master RAS Server (AtemRasSrv.dll / libAtemRasSrv.a) is a library to provide access to the Master via TCP/IP.

TCP/IP communication between the ACTIVE and the INACTIVE Master must be possible at any state without additional hardware. As long as there is no ACTIVE Master, the communication stops, because no Master sends EtherCAT frames. (REQ034, REQ035).

An EoE endpoint must be registered as network adapter to the EC-Master. The integration depends on the Master's operating system, e.g. for Windows, Windows CE and Linux the TAP driver which is freely available from OpenVPN is needed. The installation description and an example is part of the *Feature Pack EoE Endpoint Interface Manual*. (REQ036)

Data from the INACTIVE Master is queued. The poll size must be configurable.
The INACTIVE Master must transmit EoE data from the Frames of the ACTIVE Master to its EoE-Endpoint.

The RAS cycle time impacting the RAS API latency is configurable.

5.5.2 Concurrent access to ACTIVE and INACTIVE Masters

Diagnosis and remote control is possible at any state using a RAS connection to the ACTIVE Master. The Application must start the RAS Server for this functionality. (REQ037)

The EC-Engineer and EC-Lyser offer diagnosis of multiple connections in parallel, configured with individual IP addresses:

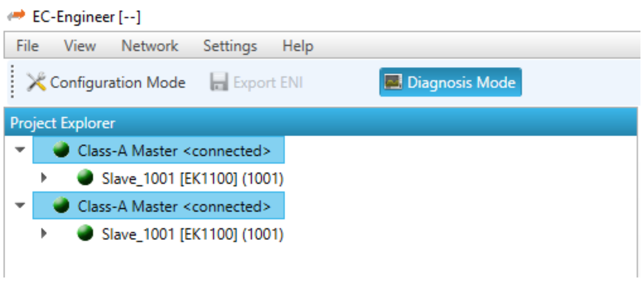

EC-Engineer [--]
File
View
Network
Settings
Help
Configuration Mode
Export ENI
Diagnosis Mode
Project Explorer
Class-A Master <connected>
Slave_1001 [EK1100] (1001)
Class-A Master <connected>
Slave_1001 [EK1100] (1001)

Figure 11: Multiple connections in EC-Engineer

6 Topology

The Masters have the two Link Layer Instances “MAIN” and “RED”. The Slaves are connected between the ACTIVE and INACTIVE Master so each Master:

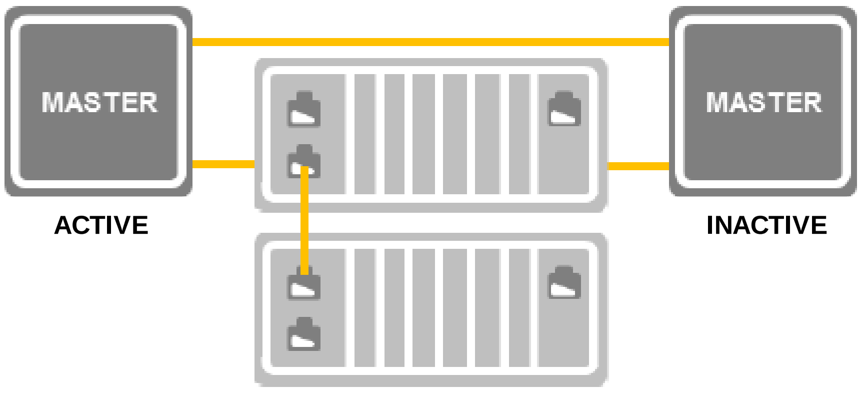


Figure 12: Masters and Slaves in Topology

This topology can apply Cable Redundancy and Master Redundancy.

The System needs permanent access to the ACTIVE and the INACTIVE Master, therefore the INACTIVE Master must be always connected to the network. (REQ038) The INACTIVE Master may not drop the Frames because the ACTIVE Master needs the Frame returning to detect slaves, retrieve input and detect data inconsistency. The NICs at the Master cannot process Frames on the fly. The Frames must be stored and forwarded. Hence the EC-Master functionality must be extended.

Slaves must be connected on same line (REQ013) else it will not match the ordering in the ENI file, so the following configuration is invalid:

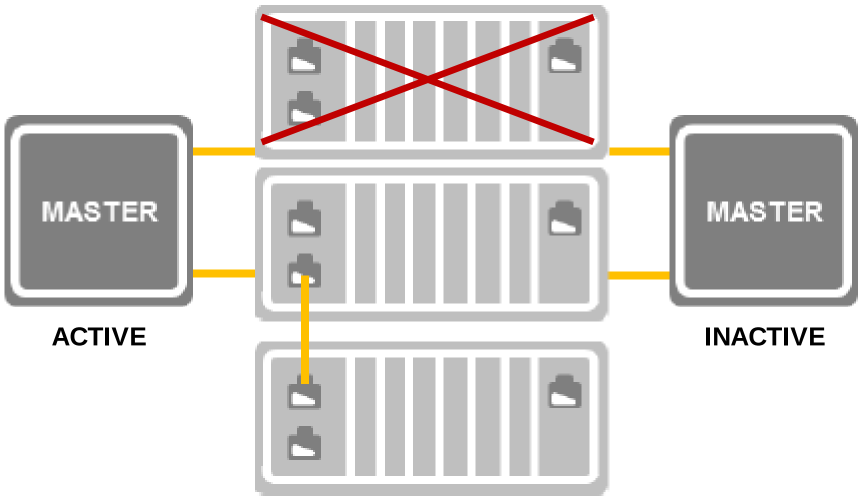


Figure 13: Slaves must be on same line connected

Only the ACTIVE Master controls all Slaves that are visible to it. The INACTIVE Master does not control any Slave, but receives and stores all Frames and forwards all foreign Frames from one Link Layer Instance to the other. Destroyed Frames are dropped. These two different functionalities are called *Controlling the bus* and *Store and Forward*.

The Master-To-Master communication is considered to be encapsulated in the Frames so there is no additional line for native TCP/IP support needed.

6.1 Frame Processing Order with Store and Forward

The Bus is owned by the ACTIVE Master that cyclically receives and sends Frames at its two Link Layer Instances. All frames traverse INACTIVE Masters and Slaves and are evaluated there respectively.

The ACTIVE Master regularly sends its data using the MAIN Link Layer Instance. All Slaves and the INACTIVE Master forward the Frames until the ACTIVE Master receives them at the RED Link Layer Instance. To accomplish Cable Redundancy purposes the ACTIVE Master uses its RED Link Layer Instance regularly for sending frames, too.
Process data is sent at the ACTIVE Master at MAIN and returns at ACTIVE Master at RED. The Master notifies the Application if frames are not received as expected and handles the error.

The ACTIVE and the INACTIVE Master act on their Link Layer Instances by default as follows:

	ACTIVE Master	**INACTIVE Master**
MAIN Link Layer Instance	Controlling the bus	Store and Forward
RED Link Layer Instance	Controlling the bus	Store and Forward

See also Figure 9: Job Task timing.

6.1.1 Frame Processing without Line-Break

The following figure shows in green the Frame processing order of ACTIVE Master's MAIN Link Layer Instance and in red the Frame processing order of the RED Link Layer Instance as on regular operation without line break. The slaves forward, but don't process the RED frame.

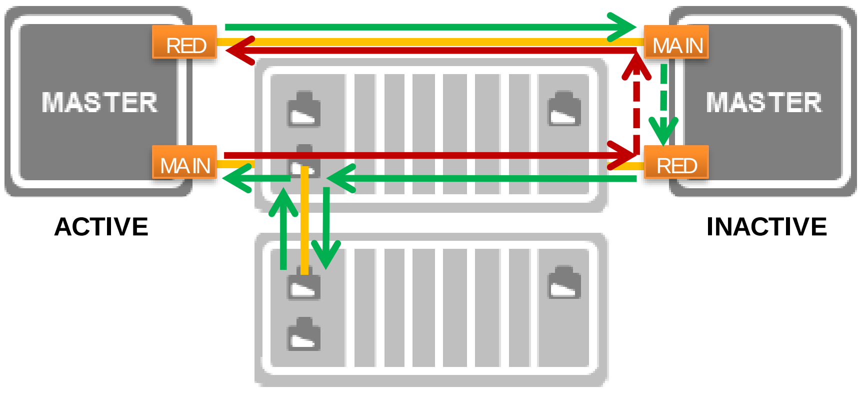


Figure 14: Regular Frame Processing Order

6.1.2 Frame Processing at Failover

If the ACTIVE Master fails, the RED port gets deactivated and the frames from MAIN return on MAIN:

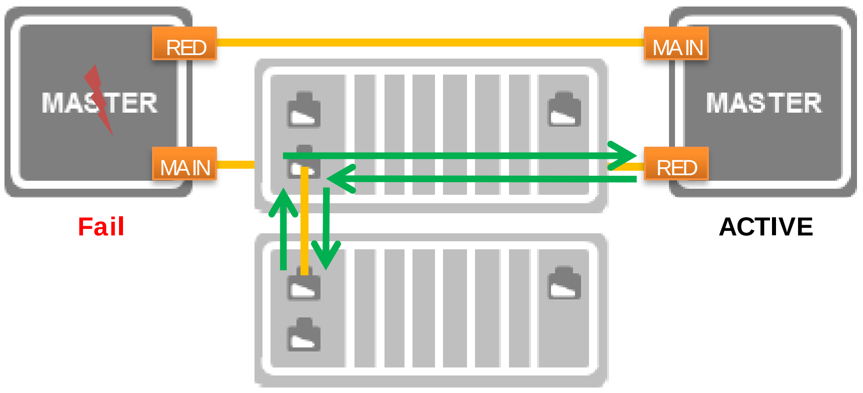


Figure 15: Failover Frame Processing Order

6.1.3 Frame Processing with Line-Break

If a line breaks or if the ACTIVE Master fails, the system handles the error without dividing the Bus into independent sub networks:

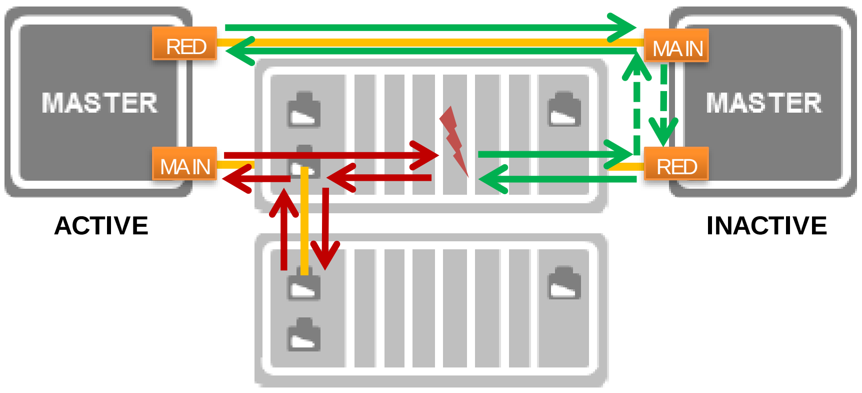


Figure 16: Line-Break Frame Processing Order

6.2 Store and forward delay time

The Master cannot process the Frames on the fly. The INACTIVE Master must Store and Forward the Frame. This increases the roundtrip time significantly. The roundtrip time furthermore depends on the delay introduced by the INACTIVE Master between storing the Frame and sending it again. This has to be considered when calculating the available bandwidth.

6.3 Bus scan timing at fail-over

The sequence at the take-over Master on fail-over is as follows:

- Application detects communication timeout
- Application sets Master ACTIVE (ecatSetMasterRedStateReq)
- Master starts sending frames, exchanges process data, scans the network

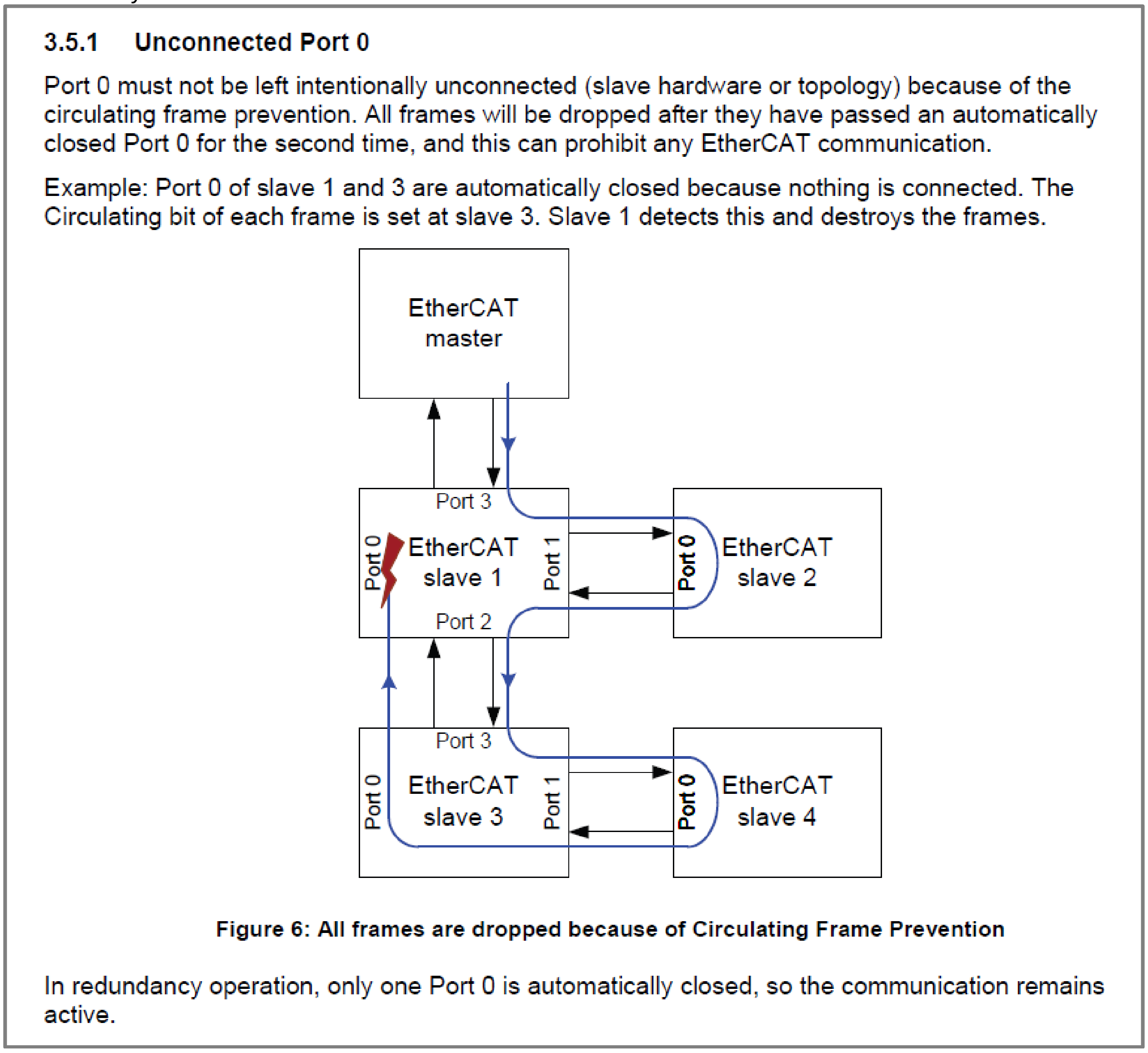
A sample fail-over profiling without optimizing the Bus Scan shows the following timing:

Cycle time: 5 ms
Process Data Payload: 5kB
Communication timeout: 15 ms
Topology: 33 slaves
Time Offset: Master 2 received last frame from Master 1

Time Offset [ms]	**Event at Master 2 since Last Frame Recv From Master 1**	**Duration**
14.890	Communication timeout detected	15 ms
14.910	Became Active	20 us
14.940	First Frame Sent	30 us
15.040	Process Data Received	100 us
329.910	Bus Scan done, Topology Known (further scans follow)	315 ms
	Thereof Readout EEPROM Information	110 ms
340.020	Coe Upload Done	10 ms

6.4 Port 0 must be connected

Chapter 3.5 of the ET1100 datasheet ([1]) explains the circulating frame prevention. If there is more than one Slave with unconnected port 0, frames are destroyed. Therefore according to chapter 3.5.1, port 0 of Slaves always has to be connected to the bus:



3.5.1 Unconnected Port 0

Port 0 must not be left intentionally unconnected (slave hardware or topology) because of the circulating frame prevention. All frames will be dropped after they have passed an automatically closed Port 0 for the second time, and this can prohibit any EtherCAT communication.

Example: Port 0 of slave 1 and 3 are automatically closed because nothing is connected. The Circulating bit of each frame is set at slave 3. Slave 1 detects this and destroys the frames.

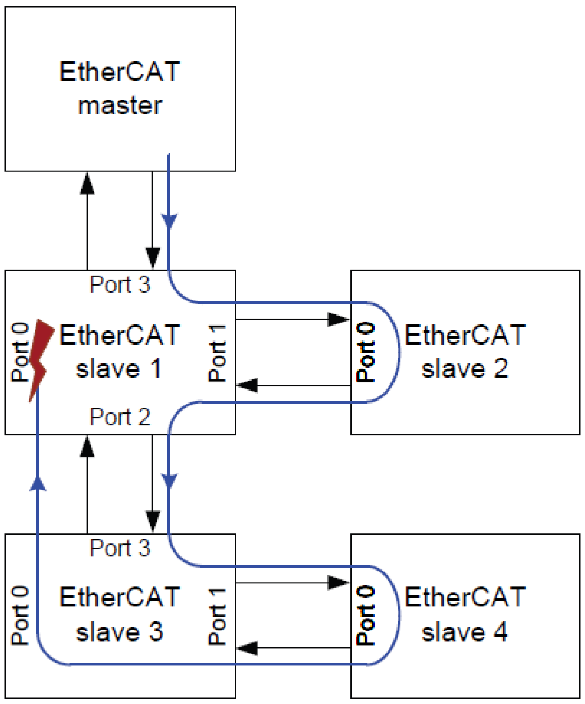


Figure 6: All frames are dropped because of Circulating Frame Prevention

In redundancy operation, only one Port 0 is automatically closed, so the communication remains active.

Figure 17: Unconnected port 0 excerpt from ET1100 datasheet

Destroyed Frames are not passed from the Link Layer Instance to the Master and therefore lost from the Master's point of view!

This must be considered especially for Master Redundancy (REQ011).

7 System Configuration

The application must provide parameters for operation. The EC-Master initialization parameters and ENI files of INACTIVE and ACTIVE Master must be identical (REQ010), see below.

7.1 Operating System Configuration

7.1.1 Link Layer Instances

The Operating System may not use the Link Layer Instances while they are assigned to the EC-Master. This is realized by either unbinding the Link Layer Instances before starting the EC-Master or removing the Operating Systems device driver handling the Link Layer Instance. Each Link Layer Instance must have a unique MAC address. The Link Layer must support Interrupt Mode.

7.1.2 Remote Access

The System must configure a Network TAP with unique MAC and IP Address assignment at each Master as described in the *Feature Pack EoE Endpoint Interface Manual*. (REQ008)

7.2 EC-Master configuration

7.2.1 EC-Master initialization parameters

The *EcMasterDemoMasterRed* demonstrates how to configure the EC-Master including the Link Layer Instances. The *EC-Master Class B Documentation* describes the EC-Master initialization parameters.

7.2.2 EC-Master Master Redundancy Parameters

The EC-Master Master Redundancy Parameters EC_T_MASTER_RED_PARMS contain:

- INACTIVE to ACTIVE Master Process Data size (INPUTs)
- ACTIVE to INACTIVE Master Process Data size (OUTPUTs)
- The max. amount of Queued Acyclic Master Red Frames send per cycle

7.2.3 EtherCAT Network Information File

The EC-Master has to know about the EtherCAT bus topology and the cyclic/acyclic frames to exchange with the slaves. This configuration is determined in a configuration file which has to be available in the **EtherCAT Network Information Format** (ENI). This format is completely independent from EtherCAT slave vendors and from EtherCAT configuration tools. Thus inter-operability between those vendors is guaranteed.

The Slaves need to be explicitly configured within the ENI-file.

The Process Data definition is also part of the ENI file and structures the bidirectional data exchanged between the EC-Master and the Slaves.

If the Masters do have an internal ESC, it must be configured within the ENI file.

The Applications must provide identical ENI files (REQ010).

Applying cable redundancy forbids usage of LRW EtherCAT commands. This can be configured using the EC-Engineer by means of applying “Disable LRW” to all slaves from the *Advanced Options* tab in the *Expert View*.

The Sync Manager Watchdog settings of the Slaves should be checked to match the system’s needs.

8 Getting started

The *EcMasterDemoMasterRed* that is part of the *EC-Master Feature Pack Master Redundancy* rudimentary demonstrates how to apply Master Redundancy.

DISCLAIMER: THE *EcMasterDemoMasterRed* MAY LEAD TO SYSTEM FAILURE, CRASHES OR OTHER HARM AND IS NOT INTENDED FOR USAGE IN PRODUCTIVE ENVIRONMENTS!

The *EcMasterDemoMasterRed* can be used to demonstrate and analyze the behavior at fail-over. Furthermore the network payload and content like process data exchange can be analyzed using e.g. Wireshark.

8.1 Topology

The following figure shows the failure locations that are demonstrated within this chapter. As shown in the figure a diagnostic PC can connect to the *EcMasterDemoMasterRed* at e.g. Master 1. This can be used to establish a TCP/IP connection to Master 2. Therefore the *EcMasterDemoMasterRed* must be started with parameter "-sp".

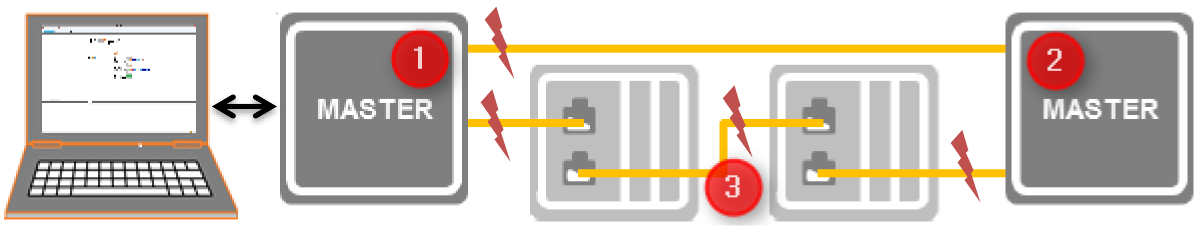


Figure 18: Master Redundancy demo topology

The topology contains two stations each consistent of e.g. an EK1100, EL2004, EL1004. To demonstrate INPUT update at INACTIVE Master, physically connect the IOs of EL2004 and EL1004:

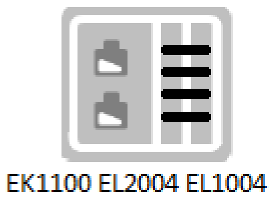


Figure 19: Physically connect OUTs of EL2004 to INs of EL1004

The network description must be provided as ENI file to *EcMasterDemoMasterRed* with parameter "-f".

8.2 Virtual Networking

In order to apply networking using Master-To-Master EtherCAT communication the *EcMasterDemoMasterRed* needs a TAP adapter. On Microsoft© Windows the Tap-windows installer from https://openvpn.net/index.php/open-source/downloads.html and on Linux the package uml-utilities provide this.

The IP addresses of the Masters must be different and setup at the network adapter properties of the Master's operating system.

The default MTU of 1500 **must be reduced** to because fragmentation is not supported at encapsulating the Master-To-Master communication in EtherCAT! (REQ039)

8.3 Simple Arbitration

The *EcMasterDemoMasterRed* awaits on start-up communication from an eventual ACTIVE Master. After waiting this time, the *EcMasterDemoMasterRed* gets ACTIVE. Just in case that two instances got ACTIVE at the same time, collisions are handled with a simple arbitration by randomly sleeping 1 … 10 ms. Therefore if both Masters are simultaneously starting up, one will win the arbitration and become ACTIVE (OPERATIONAL) and the other will fall back to INACTIVE.

When the Master is OPERATIONAL, the *EcMasterDemoMasterRed* immediately exits on collisions:

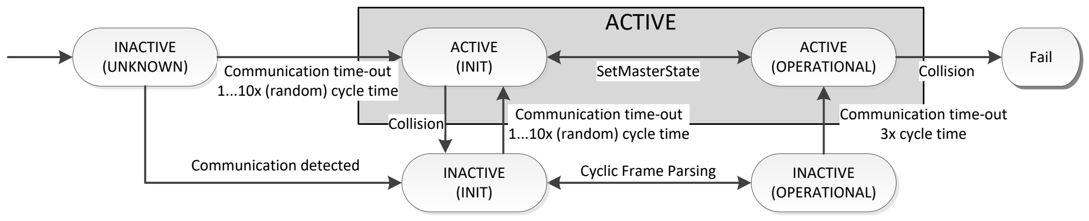


Figure 20: State changes with simple arbitration

8.4 Fail-over

8.4.1 Fail-over with failing Master

The following steps demonstrate fail-over with line break between slaves:

1. On Master 1, start *EcMasterDemoMasterRed*. It will become ACTIVE.
2. Master 1 reports EtherCAT State OPERATIONAL.
3. **Master 1** changes OUTPUTs automatically.
4. Master 1 logs INPUT changes automatically.
5. On Master 2, start *EcMasterDemoMasterRed*. It will be INACTIVE (receives frames from ACTIVE).
6. Master 2 updates its Process Data Image OUTPUTs and INPUTs when processing frames from Master 1 at Store and Forward.
7. Master 2 reports EtherCAT State OPERATIONAL, too.
8. Master 2 logs INPUT changes, too.
9. Disconnect cable between Master 1 and Master 2.
10. Disconnect cable between Master 1 and the slaves.
11. Master 1 becomes INACTIVE (because all links lost) and Master 2 becomes ACTIVE (due to communication time-out).

➔ Master 1 will report EtherCAT State UNKNOWN.
➔ **Master 2 changes OUTPUTs automatically (Fail-over done).**
➔ On reconnecting the cables, Master 1 will synchronize again with Master 2.

8.4.2 Fail-over with failing Master and failing Slaves

The following steps demonstrate fail-over with line break between slaves:

12. On Master 1, start *EcMasterDemoMasterRed*. It will become ACTIVE.
13. **Master 1** changes OUTPUTs automatically.
14. Master 1 logs INPUT changes automatically.
15. On Master 2, start *EcMasterDemoMasterRed*. It will be INACTIVE (receives frames from ACTIVE).
16. Master 2 updates its Process Data Image OUTPUTs and INPUTs when processing frames from Master 1 at Store and Forward.
17. Master 2 logs INPUT changes automatically, too.
18. Disconnect cable between the two slave stations.
19. **The network is still full OPERATIONAL due to Cable Redundancy and Store and Forward.**
20. Disconnect cable between Master 1 and the slaves. The disconnected slaves disappear, but the others are still reachable.
21. Disconnect cable between Master 1 and Master 2.
22. Master 1 becomes INACTIVE (because all links lost) and Master 2 becomes ACTIVE (due to communication time-out).
23. Master 1 will report EtherCAT State UNKNOWN.

➔ **Master 2** changes OUTPUTs automatically (Fail-over).
➔ Master 2 scans the network. If the missing slaves are mandatory according to the network configuration (ENI), Master 2 reports **Bus Mismatch**.
➔ **Master 2 fully applied the topology change (Fail-over done).**
➔ **Re-connecting the cable as *EcMasterDemoMasterRed* needs collision handling!**

8.5 Master-To-Master networking

Accessing the target operating system using TCP/IP needs EoE-Endpoints installed. See pre-processor define “INCLUDE_TAPED” in chapter **Fehler! Verweisquelle konnte nicht gefunden werden.** “**Fehler! Verweisquelle konnte nicht gefunden werden.**”.

The following steps establish a TCP connection between Master 1 and Master 2:

1. On Master 1, start *EcMasterDemoMasterRed*. It will become ACTIVE.
2. On Master 2, start *EcMasterDemoMasterRed*. It will be INACTIVE.
3. It is now possible to ping Master 2 from Master 1 and vice versa.

IP communication is now possible with the following restrictions:
- The MTU must be reduced so the Master can encapsulate the IP frames from TAP in EtherCAT.
- The communication has reduced bandwidth as it must be encapsulated in acyclic frames. (REQ040)
- The transmission timing is determined by the EtherCAT cycle.

8.6 Remote connection

The following steps establish a RAS connection to Master 2 via Master 1:

1. On Master 1, start *EcMasterDemoMasterRed*. It will become ACTIVE.
2. On Master 2, start *EcMasterDemoMasterRed*. It will be INACTIVE.
3. Start the EC-Engineer or EC-Lyser and connect to Master 1 with EoE over RAS enabled.
4. It is now possible to connect to Master 2 from another EC-Engineer instance.

9 EcMasterDemoMasterRed

The FP Master Redundancy contains the *EcMasterDemoMasterRed* including its sources.

9.1 Setting up and running EcMasterDemoMasterRed in Hot Standby

The virtual network feature needs a TAP interface. To install and configure execute the following steps:

1. Install uml-utilities on your Linux distribution, e.g.
```
> apt-get install uml-utilities
```
2. Create the TAP interface
```
> tunctl -t tap0
```
3. Configure the interface, e.g.
```
> ip link set tap0 up
> ip addr add 10.0.0.2/24 dev tap0
> ifconfig tap0 mtu 200 up
```
4. Add static ip-route(if necessary)
```
> ip route add xxx.xxx.xx.x/24 via 10.0.0.2
> echo 1 > /proc/sys/net/ipv4/ip_forward
```

The EcMasterDemoMasterRed will install the tap driver if compiled with `INCLUDE_TAPED` defined.

To start EcMasterDemoMasterRed the full path and file name of the configuration file has to be given as a command line parameter as well as the appropriate Main/Red Link Layers in interrupt mode.

Example:
```
./EcMasterDemoMasterRed -f MasterRedENI.xml -i8254x 0x01bbddff 0 -i8254x 0x01bbddff 0
-b 50000 -t 0 -sp [-standby]
```

For more information about the command line parameters see EC-Master Class B documentation chapter *2.4.2. Command line parameters* and chapter *2.5.4.2.6 Intel Pro/1000 EtherCAT driver emllI8254x*.

Parameter `-standby` prevents the master to get ACTIVE. As soon as the Masters are ready they can be switched from Permanent Standby to Hot Standby using `EC_NOTIFY_APP_MASTER_RED_UNFORCE_STANDBY` ,see below.

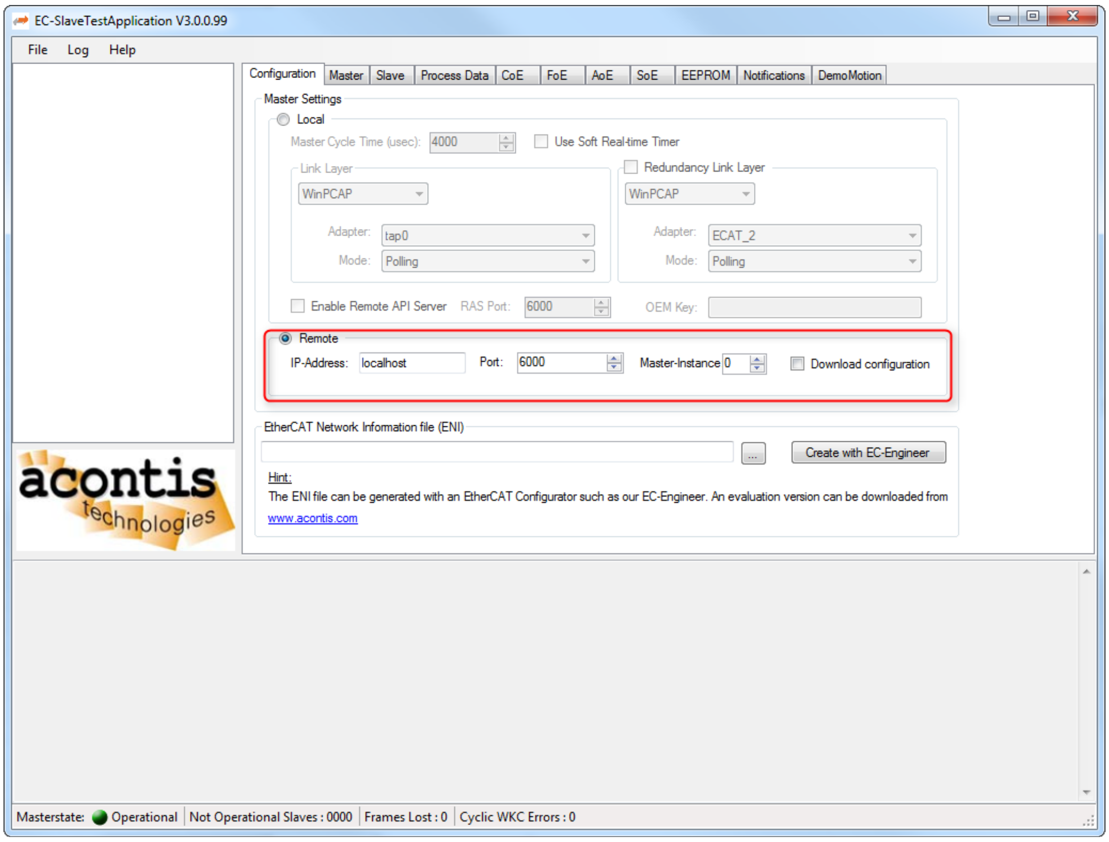
9.2 Switchover EcMasterDemoMasterRed

The EcMasterDemoMasterRed demo program implements the functionality to switchover the ACTIVE Master to INACTIVE and the INACTIVE Master to ACTIVE via notifications.

To coordinate a switchover the following steps are required:

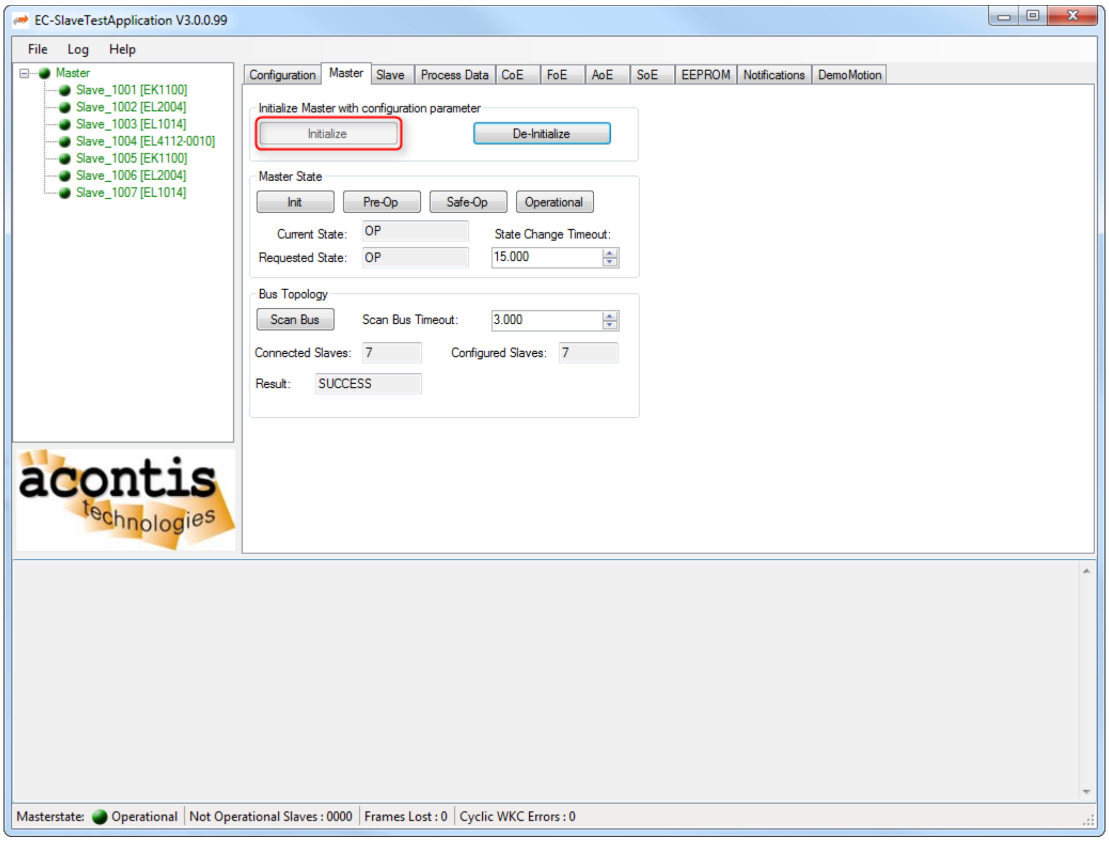
1. Start Master 1 demo in permanent INACTIVE state (Parameter `-standby`).
2. Start Master 2 demo which will become ACTIVE.
3. Unforce permanent standby with `EC_NOTIFY_APP_MASTER_RED_UNFORCE_STANDBY(Code = 2)` notification sent to Master 1.
4. Set Master 2 in INACTIVE state with notification `EC_NOTIFY_APP_MASTER_RED_FORCE_STANDBY(Code = 1)`. Master 1 will automatically get ACTIVE on communication timeout.

The notifications can be send with the EC-SlaveTestApplication via a remote connection to the masters see Figures 16-18.
To enable remote connection start EcMasterDemoMasterRed with command line parameter `-sp`.



Master Settings
Local
Master Cycle Time (usec): 4000
Use Soft Real-time Timer
Link Layer: WinPCAP
Adapter: tap0
Mode: Polling
Redundancy Link Layer: WinPCAP
Adapter: ECAT_2
Mode: Polling
Enable Remote API Server
RAS Port: 6000
OEM Key:
Remote
IP-Address: localhost
Port: 6000
Master-Instance 0
Download configuration
EtherCAT Network Information file (ENI)
Hint:
The ENI file can be generated with an EtherCAT Configurator such as our EC-Engineer. An evaluation version can be downloaded from www.acontis.com
Masterstate: ● Operational | Not Operational Slaves : 0000 | Frames Lost : 0 | Cyclic WKC Errors : 0

Figure 16: Configure EC-STA


EC-SlaveTestApplication V3.0.0.99

File Log Help

Master
- Slave_1001 [EK1100]
- Slave_1002 [EL2004]
- Slave_1003 [EL1014]
- Slave_1004 [EL4112-0010]
- Slave_1005 [EK1100]
- Slave_1006 [EL2004]
- Slave_1007 [EL1014]

Configuration | Master | Slave | Process Data | CoE | FoE | AoE | SoE | EEPROM | Notifications | DemoMotion

Initialize Master with configuration parameter: Initialize | De-Initialize

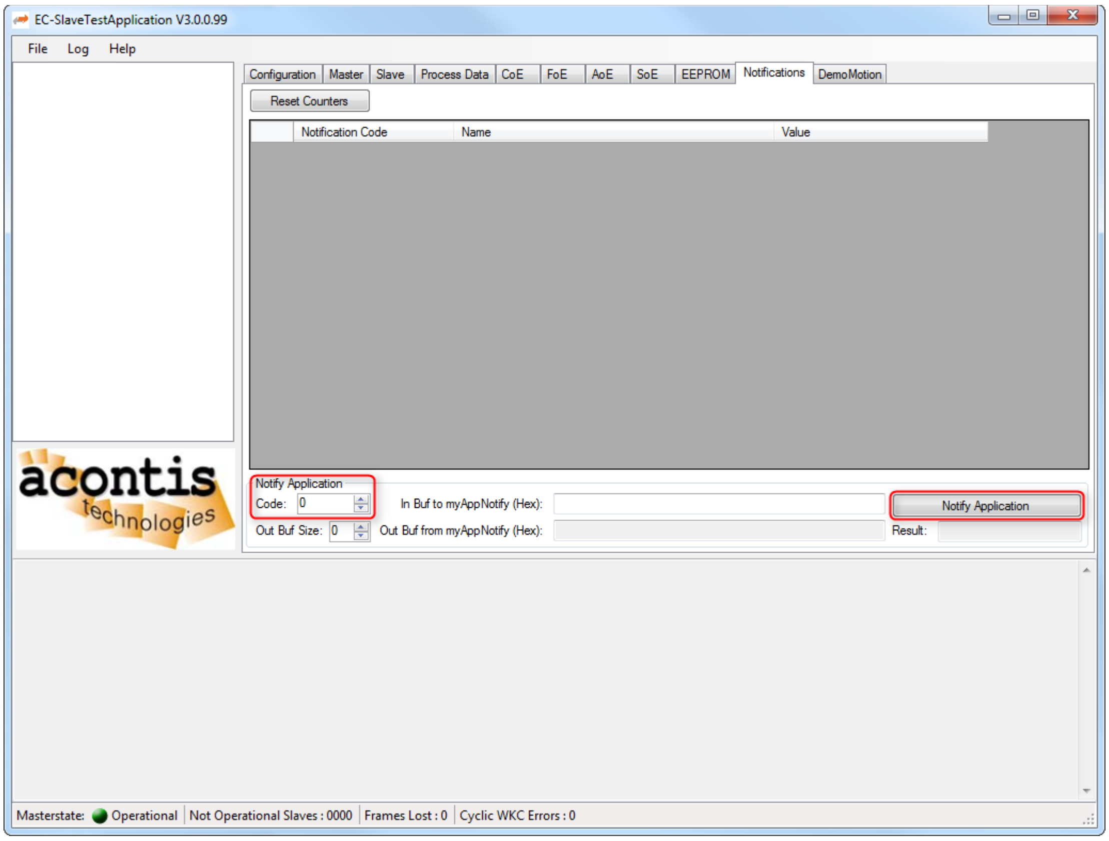
Master State: Init | Pre-Op | Safe-Op | Operational

Current State: OP
Requested State: OP
State Change Timeout: 15.000

Bus Topology: Scan Bus
Scan Bus Timeout: 3.000
Connected Slaves: 7
Configured Slaves: 7
Result: SUCCESS

Masterstate: Operational | Not Operational Slaves : 0000 | Frames Lost : 0 | Cyclic WKC Errors : 0

Figure 17: Initialize Master EC-STA


EC-SlaveTestApplication V3.0.0.99

File Log Help

Configuration | Master | Slave | Process Data | CoE | FoE | AoE | SoE | EEPROM | Notifications | DemoMotion

Reset Counters

Notification Code	Name	Value

Notify Application
Code: 0
Out Buf Size: 0
In Buf to myAppNotify (Hex):
Out Buf from myAppNotify (Hex):
Notify Application
Result:

Masterstate: Operational | Not Operational Slaves : 0000 | Frames Lost : 0 | Cyclic WKC Errors : 0

FIGURE 18: Notifications EC-STA

10 Application programming interface

10.1 Configure Master Redundancy in ecatInitMaster

The application must fill **EC_T_MASTER_RED_PARMS** when call *ecatInitMaster*.

```
typedef struct _EC_T_MASTER_RED_PARMS {
    EC_T_BOOL  bEnabled;                   /* set to EC_TRUE if using Master Redundancy */
    EC_T_WORD  wMasterPdOutSize;           /* ACTIVE to INACTIVE Master (in bytes) */
    EC_T_WORD  wMasterPdInSize;            /* INACTIVE to ACTIVE Master (in bytes) */
    EC_T_DWORD dwMaxAcycFramesPerCycle;    /* max. acyclic Master Red frames sent per cycle */
    EC_T_BOOL  bUpdateSlavePdOut;          /* Sync slave OUTPUTs to INACTIVE (from CSF) */
    EC_T_BOOL  bUpdateSlavePdIn;           /* Sync slave INPUTs to INACTIVE (from CMF) */
} EC_T_MASTER_RED_PARMS;
```

Example

```
{
    EC_T_INIT_MASTER_PARMS oInitParms;

    OsMemset(&oInitParms, 0, sizeof(EC_T_INIT_MASTER_PARMS));
    …
    oInitParms.MasterRedParms.bEnabled = EC_TRUE;
    oInitParms.MasterRedParms.bUpdateSlavePdIn = EC_TRUE;
    oInitParms.MasterRedParms.bUpdateSlavePdOut = EC_TRUE;
    oInitParms.MasterRedParms.wMasterPdInSize = 10;
    oInitParms.MasterRedParms.wMasterPdOutSize = 20;
    oInitParms.MasterRedParms.dwMaxAcycFramesPerCycle = 3;
    …
    dwRes = ecatInitMaster(&oInitParms);
    …
}
```

10.2 ecatSetMasterRedStateReq

Requests Master Redundancy State ACTIVE / INACTIVE.

EC_T_DWORD ecatSetMasterRedStateReq(EC_T_BOOL bActive);

Parameters

bActive

[in] EC_TRUE: ACTIVE, EC_FALSE: INACTIVE

Return

EC_E_NOERROR or error code
EC_E_INVALIDSTATE if MasterRedParms.bEnabled = EC_FALSE or Master not initialized.
EC_E_NOTSUPPORTED if EC-Master stack does not include Master Redundancy support.

Example

```
dwRes = ecatSetMasterRedStateReq(EC_TRUE);
if (dwRes != EC_E_NOERROR)
{
    …
}
```

10.3 ecatGetMasterRedState

Gets Master Redundancy State (ACTIVE / INACTIVE).

EC_T_DWORD ecatGetMasterRedState(EC_T_BOOL* pbActive);

Parameters

pbActive

[out] Pointer to variable of type EC_T_BOOL. Contains Master Redundancy State on success.

Return

EC_E_NOERROR or error code
EC_E_INVALIDSTATE if MasterRedParms.bEnabled = EC_FALSE or Master not initialized.
EC_E_NOTSUPPORTED if EC-Master stack does not include Master Redundancy support.

Comment

-

Example

```
EC_T_BOOL bMasterActive = EC_FALSE;
dwRes = ecatGetMasterRedState(&bMasterActive);
if (dwRes != EC_E_NOERROR)
{
    …
}
```

10.4 ecatGetMasterRedProcessImageInputPtr

Gets pointer to INACTIVE to ACTIVE Master Process Data.

EC_T_BYTE* ecatGetMasterRedProcessImageInputPtr(EC_T_VOID);

Parameters

-

Return

- Pointer to INACTIVE to ACTIVE Master Process Data
- EC_NULL if MasterRedParms.bEnabled = EC_FALSE or Master not initialized or MasterRedParms.wProcessImageInputSize = 0 or EC-Master stack does not include Master Redundancy support.

Comment

-

Example

```
EC_T_BYTE* pInactiveToActivePd = ecatGetMasterRedProcessImageInputPtr();
if (EC_NULL != pInactiveToActivePd)
{
    …
}
```

10.5 ecatGetMasterRedProcessImageOutputPtr

Gets pointer to ACTIVE to INACTIVE Master Process Data.

EC_T_BYTE* ecatGetMasterRedProcessImageOutputPtr(EC_T_VOID);

Parameters

-

Return

- Pointer to ACTIVE to INACTIVE Master Process Data
- EC_NULL if MasterRedParms.bEnabled = EC_FALSE or Master not initialized or MasterRedParms.wProcessImageOutputSize = 0 or EC-Master stack does not include Master Redundancy support.

Comment

-

Example

```
EC_T_BYTE* pActiveToInactivePd = ecatGetMasterRedProcessImageOutputPtr();
if (EC_NULL != pActiveToInactivePd)
{
    …
}
```

10.6 EC_NOTIFY_FRAME_RESPONSE_ERROR

See *EC-Master Class B Documentation*, especially parameter eRspErr_FOREIGN_SRC_MAC.

10.7 EC_NOTIFY_MASTER_RED_STATECHANGED

Notification about a change of the Master Redundancy State.

Parameters

pbyInBuf
[in] Pointer to data of type EC_T_BOOL which contains the new Master Redundancy State.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

10.8 APIs at ACTIVE / INACTIVE

The behaviour of APIs may be different depending on the Master Redundancy State. Some functionality may be blocked and the API returns EC_E_MASTER_RED_STATE_ACTIVE or EC_E_MASTER_RED_STATE_INACTIVE accordingly. The lists are subject to be extended.

The following APIs are blocked if the Master is INACTIVE:

ecatCoeSdoDownloadReq, ecatCoeSdoUploadReq, ecatCoeGetODList, ecatCoeGetObjectDesc, ecatCoeGetEntryDesc, ecatSetSlaveState, ecatTferSingleRawCmd, ecatSetMasterState, ecatQueueRawCmd, ecatCoeRxPdoTfer, ecatFoeFileUpload, ecatFoeFileDownload, ecatFoeUploadReq, ecatFoeDownloadReq, ecatCoeSdoDownload, ecatCoeSdoUpload, ecatReadSlaveEEPRom, ecatWriteSlaveEEPRom, ecatAssignSlaveEEPRom, ecatActiveSlaveEEPRom, ecatClntQueueRawCmd, ecatReadSlaveRegister, ecatWriteSlaveRegister, ecatVoeRead, ecatVoeWrite, ecatVoeWriteReq, ecatEthDbgMsg, ecatAoeRead, ecatAoeWrite, ecatAoeWriteControl, ecatAoeReadReq, ecatAoeWriteReq, ecatSoeRead, ecatSoeWrite, ecatSoeAbortProcCmd, ecatAoeReadWrite, ecatMbxTferAbort, ecatSoeReadReq, ecatSoeWriteReq, ecatSetSlavePortState, ecatBlockNode, ecatOpenBlockedPorts, ecatForceTopologyChange, ecatReloadSlaveEEPRom, ecatResetSlaveController, ecatGetSlaveInfo, catIsTopologyChangeDetected, ecatScanBus, ecatClntSendRawMbx, ecatFoeSegmentedDownloadReq, ecatFoeSegmentedUploadReq, ecatReadSlaveIdentification, ecatRescueScan, ecatGetSlaveStatistics, ecatClearSlaveStatistics

The following IOCTLs are blocked if the Master is INACTIVE:
EC_IOCTL_GET_SLVSTATISTICS, EC_IOCTL_SET_SLVSTAT_PERIOD, EC_IOCTL_FORCE_SLVSTAT_COLLECTION, EC_IOCTL_SB_RESTART.

The following APIs are enabled for ACTIVE / INACTIVE, but (may) behave differently:

- ecatIoControl
- ecatExecJob
- ecatIsSlavePresent
- ecatGetSlaveState
- ecatGetMasterState
- ecatCoeSdoUpload, ecatCoeSdoDownload (at INACTIVE only Master OD is accessible)

The following APIs are independent of the Master Redundancy State:

- OS Layer APIs like OsInit, OsSleep, OsAuxClkInit, …
- Link Layer APIs like EcLinkSendFrame, …
- Trace APIs like ecatSetTraceMask, …
- Performance Measurement APIs like ecatPerfMeasStart, ecatPerfMeasEnd
- ecatGetSlaveId
- ecatMbxTferCreate, ecatMbxTferDelete
- ecatConfigureMaster
- ecatGetSlaveId

11 Implementation Hints

11.1 Startup API order

Configuring Master Redundancy must be done when initializing the Master, before starting the Job Task to prevent from unintentional collisions and frame loss at ACTIVE Master. (REQ041)

The client registration is needed for collision detection and therefore must take place, before starting the Job Task.

The client registration must be done immediately after configuring the master else it will be dropped.

Master Redundancy needs an ENI file given.

The Application must provide the Job Task executing the Master APIs at ACTIVE **and** INACTIVE Master.

11.2 Communication needs an ACTIVE Master

In order to keep real-time constraints the INACTIVE Master does not autonomously send frames so if the application decides to set both Master INACTIVE the communication on network will stop.

11.3 Links at failing Master must go down

If a Master is not able to send or forward any frames, the links at the connected EtherCAT devices must go down else the EtherCAT communication is destroyed. (REQ012)
The link at the INACTIVE Master may only be established as soon as it can forward frames!

Link loss to the failing Master could be used for early Master failure detection.

11.4 Mailbox transfers and switchover

The EC-Master cancels pending Mailbox transfers on getting INACTIVE. The Application must handle the case that a mailbox transfer is denied after a failover or switchover, e.g. retry the transfer.

12 Frame Processing

12.1 Cyclic Slave Frames and Cyclic Master Red Frames

Process Data exchange is distributed to *Cyclic Slave Frames* (CSF) and *Cyclic Master Red Frames* (CMF). The ACTIVE Master sends the Slave Process Data in *Cyclic Slave Frames* according to the ENI file. The ACTIVE Master merges the Slave INPUT Process Data from last cycle and sends it together with the ACTIVE to INACTIVE Master Process Data in Cyclic Master Frames to the INACTIVE Master. The INACTIVE Master gets the Slave OUTPUT Process Data from the *Cyclic Slave Frames* and forwards the Cyclic Slave Frames. The INACTIVE Master gets the ACTIVE to INACTIVE Master Process Data from the *Cyclic Master Red Frames*, inserts the INACTIVE to ACTIVE Master Process Data accordingly and forwards the *Cyclic Master Red Frames* to the ACTIVE Master.

Slave		**ACTIVE Process Data Image**		**INACTIVE Process Data Image**
INPUTs	CSF →	Slave INPUTs	CMF →	Slave INPUTs
OUTPUTs	← CSF	Slave OUTPUTs	CSF →	Slave OUTPUTs
		INACTIVE to ACTIVE Master	← CMF	INACTIVE to ACTIVE Master
		ACTIVE to INACTIVE Master	CMF →	ACTIVE to INACTIVE Master

The process order is as follows.

At ACTIVE Master:

1. The ACTIVE Master sends the CSF (write Slave OUTPUT Process Data, read Slave INPUT Process Data) to MAIN and RED
2. The ACTIVE Master send the CMF (write Slave INPUT Process Data and from last cycle and ACTIVE to INACTIVE Master Process Data) to MAIN and RED
3. The ACTIVE Master sends acyclic Frames to MAIN and RED

At INACTIVE Master:

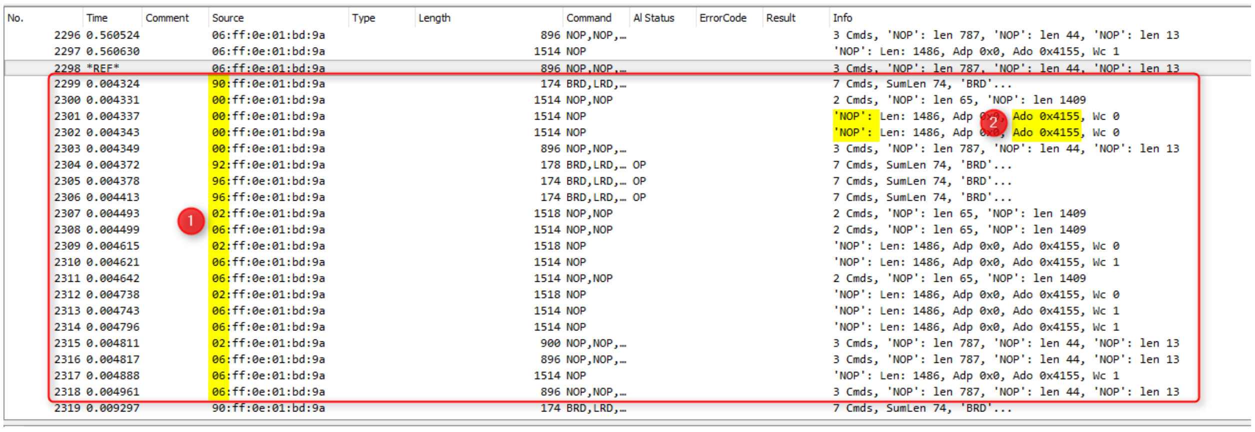
4. The INACTIVE Master receive each frame from MAIN and RED
5. The INACTIVE Master parses the CMF and gets the Slave INPUT Process Data from last cycle and ACTIVE to INACTIVE Master Process Data and it inserts INACTIVE to ACTIVE Master Process Data to CMF
6. The INACTIVE Master forwards each received frame and stores it for further processing
7. The INACTIVE Master parses each frame received from MAIN and RED and gets the Slave OUTPUT Process Data
8. The INACTIVE Master notifies the application about all cyclic data available as soon as both last cyclic frames (CMF from MAIN and RED) are processed

At ACTIVE Master:

9. The ACTIVE Master stores each received frame for processing
10. Each frame received from MAIN and RED gets processed (update Slave INPUTs and INACTIVE to ACTIVE Master Process Data)
11. The ACTIVE Master notifies the application about all cyclic data available as soon as both last cyclic frames (CMF from MAIN and RED) are processed

12.2 Example Trace

The Wireshark screenshot below shows the combined cyclic EtherCAT traffic recorded by two EcMasterDemoMasterRed instances in OPERATIONAL The Source address (1) contains flags stating where the frame was sent / forwarded / received and if it hit slaves, see also below.
Cmds at Ado 0x4155 (2) contain cyclic Master Master data:



No.	Time	Comment	Source	Type	Length	Command	Al Status	ErrorCode	Result	Info
2296	0.560524		06:ff:0e:01:bd:9a		896	NOP,NOP,…				3 Cmds, 'NOP': len 787, 'NOP': len 44, 'NOP': len 13
2297	0.560630		06:ff:0e:01:bd:9a		1514	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 1
2298	*REF*		06:ff:0e:01:bd:9a		896	NOP,NOP,…				3 Cmds, 'NOP': len 787, 'NOP': len 44, 'NOP': len 13
2299	0.004324		90:ff:0e:01:bd:9a		174	BRD,LRD,…				7 Cmds, SumLen 74, 'BRD'...
2300	0.004331		00:ff:0e:01:bd:9a		1514	NOP,NOP				2 Cmds, 'NOP': len 65, 'NOP': len 1409
2301	0.004337		00:ff:0e:01:bd:9a		1514	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 0
2302	0.004343		00:ff:0e:01:bd:9a		1514	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 0
2303	0.004349		00:ff:0e:01:bd:9a		896	NOP,NOP,…				3 Cmds, 'NOP': len 787, 'NOP': len 44, 'NOP': len 13
2304	0.004372		92:ff:0e:01:bd:9a		178	BRD,LRD,…	OP			7 Cmds, SumLen 74, 'BRD'...
2305	0.004378		96:ff:0e:01:bd:9a		174	BRD,LRD,…	OP			7 Cmds, SumLen 74, 'BRD'...
2306	0.004413		96:ff:0e:01:bd:9a		174	BRD,LRD,…	OP			7 Cmds, SumLen 74, 'BRD'...
2307	0.004493		02:ff:0e:01:bd:9a		1518	NOP,NOP				2 Cmds, 'NOP': len 65, 'NOP': len 1409
2308	0.004499		06:ff:0e:01:bd:9a		1514	NOP,NOP				2 Cmds, 'NOP': len 65, 'NOP': len 1409
2309	0.004615		02:ff:0e:01:bd:9a		1518	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 0
2310	0.004621		06:ff:0e:01:bd:9a		1514	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 1
2311	0.004642		06:ff:0e:01:bd:9a		1514	NOP,NOP				2 Cmds, 'NOP': len 65, 'NOP': len 1409
2312	0.004738		02:ff:0e:01:bd:9a		1518	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 0
2313	0.004743		06:ff:0e:01:bd:9a		1514	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 1
2314	0.004796		06:ff:0e:01:bd:9a		1514	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 1
2315	0.004811		02:ff:0e:01:bd:9a		900	NOP,NOP,…				3 Cmds, 'NOP': len 787, 'NOP': len 44, 'NOP': len 13
2316	0.004817		06:ff:0e:01:bd:9a		896	NOP,NOP,…				3 Cmds, 'NOP': len 787, 'NOP': len 44, 'NOP': len 13
2317	0.004888		06:ff:0e:01:bd:9a		1514	NOP				'NOP': Len: 1486, Adp 0x0, Ado 0x4155, Wc 1
2318	0.004961		06:ff:0e:01:bd:9a		896	NOP,NOP,…				3 Cmds, 'NOP': len 787, 'NOP': len 44, 'NOP': len 13
2319	0.009297		90:ff:0e:01:bd:9a		174	BRD,LRD,…				7 Cmds, SumLen 74, 'BRD'...

12.3 Split and Merge

The frames are sent from and received on both interfaces. If somewhere in the middle a line breaks up, the loop is closed within the “bordering” slaves and both bus stubs are still communicating with the EC-Master.

The EC-Master knows of each Slave if it is visible to the MAIN or RED EtherCAT interface. The datagrams are included in the frame corresponding to the EtherCAT interface that the Slave is visible to and the datagrams are merged as soon as they return.

The API calls still operate the same way as before the bus split up and therefore, from an application’s point of view, the bus still operates as usual (as long as all slaves are still powered and operating). Especially Auto Increment addressed and logical commands used within API calls are still unchanged after the bus split. Therefore the application developer doesn’t need to change any calls or addresses during runtime.

In case a line break is detected, a notification is thrown to the application, which enables a front-end or application user to be informed about the redundancy situation. When the bus is reconnected another notification is thrown to inform about the absence of the error situation.

12.4 Frame processing at INACTIVE Master

The INACTIVE Master automatically issues Store and Forwards of the received frames.
The frames are processed immediately after receiving (Interrupt Mode used). After both frames (from MAIN and RED) are received they are automatically processed.

12.5 Ethernet source address semantic

The destination MAC address of the EtherCAT frames are fixed to the EtherCAT unicast address 01:01:05:01:00:00 or broadcasted to ff:ff:ff:ff:ff:ff.
The source MAC address of frames sent from any EtherCAT interface of the Master contains the *Network Interface Controller (NIC) Specific* bytes of the EtherCAT interface MAC address and frame state information like the Redundancy Bit, Retry Index and the Redundancy Frame ID. Furthermore it contains the Forwarding Rule bit 0x02 that the Slaves set if the ESC is configured to drop non-EtherCAT frames (see ET1100 datasheet).
The Redundancy Bit 0x08 is set for all frames sent at the RED EtherCAT interface and it is not set for all frames sent at the MAIN EtherCAT interface. The retry index is set to indicate sending retries of asynchronous frames. It is incremented according to the retries. The Redundancy Frame ID is incremented with every frame per interface.

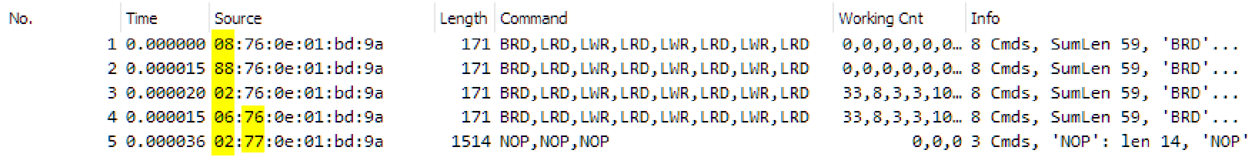
The frame is structured the following way:

Retry Index Mask 0x70	**Red. Bit** Mask 0x08	**Master Forwarding** Mask 0x84	**Slave Forwarding** Mask 0x02	**RedFrameId** Byte 1	**LL Reserved** Byte 2	**MAC Byte 3-5**

When receiving frames, the EC-Master performs consistency checks and fast error detection.
The EC-Master detects by means of the Redundancy Bit if there is a line break (disregarding local rings). It expects for frames received at RED that they were sent at MAIN and therefore their Redundancy Bit is not set. Frames received at MAIN are marked with the Redundancy Bit if the line is not broken (disregarding local rings).
The Redundancy Frame ID is used to merge received frames and to detect frame loss.

The Master Forwarding Bits are used to prevent circulating forwarded frames, see chapter 12.7, "*Circulating frame prevention*".

The following sample trace of EtherCAT frames with MAIN and RED link shows in first source byte the Redundancy bit 0x08 and the Forwarding Rule bits 0x08, 0x04 and 0x02 and in the second source byte the incrementing RedFrameId (0x76, 0x77):


```
No.      Time      Source              Length Command                          Working Cnt   Info
       1 0.000000 08:76:0e:01:bd:9a       171 BRD,LRD,LWR,LRD,LWR,LRD,LWR,LRD  0,0,0,0,0,0… 8 Cmds, SumLen 59, 'BRD'...
       2 0.000015 88:76:0e:01:bd:9a       171 BRD,LRD,LWR,LRD,LWR,LRD,LWR,LRD  0,0,0,0,0,0… 8 Cmds, SumLen 59, 'BRD'...
       3 0.000020 02:76:0e:01:bd:9a       171 BRD,LRD,LWR,LRD,LWR,LRD,LWR,LRD  33,8,3,3,10… 8 Cmds, SumLen 59, 'BRD'...
       4 0.000015 06:76:0e:01:bd:9a       171 BRD,LRD,LWR,LRD,LWR,LRD,LWR,LRD  33,8,3,3,10… 8 Cmds, SumLen 59, 'BRD'...
       5 0.000036 02:77:0e:01:bd:9a      1514 NOP,NOP,NOP                            0,0,0 3 Cmds, 'NOP': len 14, 'NOP'
```
Figure 21: Wireshark trace demonstrating Source MAC address semantic

E.g. from the Wireshark above the semantics are as follows:
Frame 1: 0x08: Received at INACTIVE MAIN from ACTIVE RED (Red. Bit)
Frame 2: 0x88: Received at ACTIVE MAIN from INACTIVE RED (Red. Bit + Master Forwarding RED)
Frame 3: 0x02: Received at INACTIVE RED from ACTIVE RED and Slaves (Slave Forwarding)
Frame 4: 0x06: Received at ACTIVE RED from INACTIVE MAIN and Slaves (Slave Forwarding + Master Forwarding MAIN)
Frame 5: 0x02: Received at INACTIVE RED from ACTIVE RED and Slaves (Slave Forwarding)

12.6 Data consistency

The WKC field of EtherCAT datagrams are used to ensure the consist datagram processing of each addressed Slave.

If there is a line break the ACTIVE Master has to be aware of the EtherCAT interface, each Slave is visible to. Using Split & Merge, the Slaves that are not visible to MAIN anymore due to the line break are visible to RED. In order to reach all Slaves disregarding if they are visible at MAIN or RED in case of a line break, the EC-Master has to send all frames on both lines. If there is a line break, a bitwise OR operation has to be performed by the EC-Master when merging the processed frames. If a datagram is addressed to several Slaves and those are distributed to both EtherCAT interfaces MAIN and RED, the WKC of each frame is evaluated accordingly after merging the frames from MAIN with the corresponding frames from RED.

As long as there is no line break, all Slaves operate on the same frames and increase the WKC fields whenever they are processing datagrams. In contrast to Split & Merge, only the WKCs of MAIN need to be considered to check the data consistency.

EtherCAT LRW commands may not be sent if there is a line break, because the addressed Slave to be read from may be at the other EtherCAT interface than the Slaves to be written to. The EC-Master respects this to ensure the integrity, e.g. configurations containing LRW commands are refused by the EC-Master when applying Cable Redundancy.

The INACTIVE Master increments at the NOP cmds in the CMF the WKC on ACTIVE to INACTE Master Process Data access by 2 and for INACTIVE to ACTIVE Master Process Data by 1.

12.7 Circulating frame prevention

EtherCAT provides a circulating frame prevention mechanism. A frame may hit a device with unconnected port 0 at most one time. Allowing to hit once is needed to handle line break. If it hits a device with unconnected port 0 a second time, the frame will be destroyed to stop delivering outdated data. This is realized at ESCs by setting the *Circulating Frame bit* in the EtherCAT command. The mechanism is documented in the “EtherCAT Slave Controller Hardware Data Sheet Section I” chapter “3.5 Circulating Frames”. As the *Circulating Frame bit* semantic cannot be fully applied for Master Redundancy to handle all circulating frame scenarios, the two different Master Forwarding Bits are considered to drop frames that are scheduled to be forwarded for the same network adapter again, in order to prevent circulating frames.

The following figure shows the regular Master Redundancy setup with ACTIVE and INACTIVE Master:

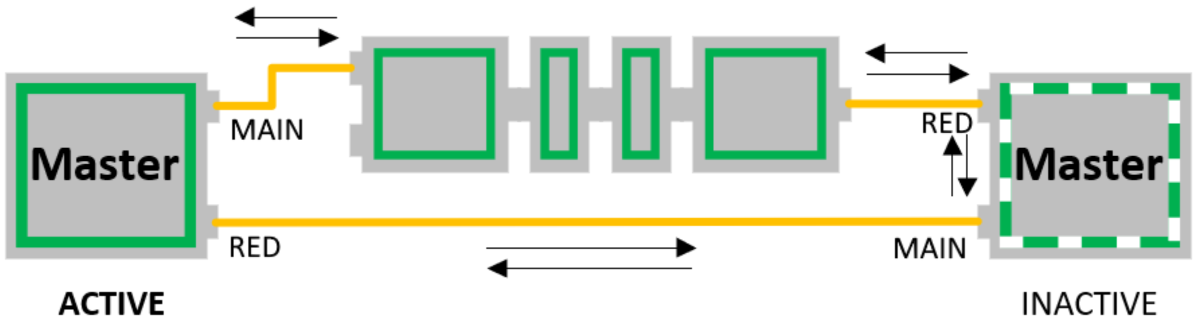


Figure 22: MAIN and RED network adapters of Masters

If the ACTIVE Master got disconnected from the network and the INACTIVE Master is about to forward frames of the current cycle, each frame may only be forwarded once per adapter in order to prevent circulating frames:

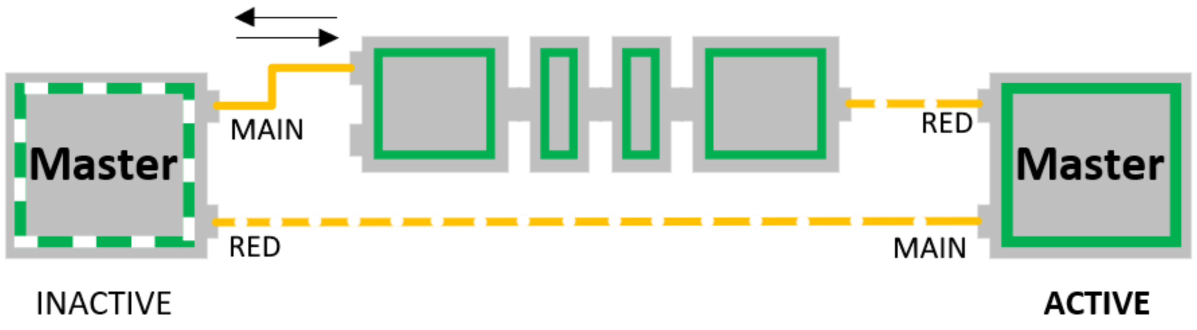


Figure 23: ACTIVE Master removed from network

In this case the forwarded frame of the previously connected ACTIVE Master is marked to be forwarded on MAIN (Master Forwarding Bit Mask 0x04 = 1). On returning it is dropped as else it would be forwarded to MAIN again. The same applies to the RED network adapter (Master Forwarding Bit Mask 0x80 = 1).