Hardware data sheet section I

EtherCAT® slave controller

Section I – Technology

EtherCAT protocol, physical layer, EtherCAT processing unit, FMMU, SyncManager, SII EEPROM, distributed clocks

Section II – Register description

(Online at http://www.beckhoff.com)

Section III – Hardware description

(Online at http://www.beckhoff.com)

Version 2.5
Date: 2025-07-28

BECKHOFF

DOCUMENT ORGANIZATION

The Beckhoff EtherCAT slave controller (ESC) documentation covers the following Beckhoff ESCs:

- ET1200
- ET1100, ET1150
- EtherCAT IP core for FPGAs
- ESC20

The documentation is organized in three sections. Section I and section II are common for all Beckhoff ESCs, section III is specific for each ESC variant.

The latest documentation is available at the Beckhoff homepage (http://www.beckhoff.com).

Section I – technology (all ESCs)

Section I deals with the basic EtherCAT technology. Starting with the EtherCAT protocol itself, the frame processing inside EtherCAT slaves is described. The features and interfaces of the physical layer with its two alternatives Ethernet and EBUS are explained afterwards. Finally, the details of the functional units of an ESC like FMMU, SyncManager, distributed clocks, slave information interface, interrupts, watchdogs, and so on, are described.

Since section I is common for all Beckhoff ESCs, it might describe features which are not available in a specific ESC. Refer to the feature details overview in section III of a specific ESC to find out which features are available.

Section II – register description (all ESCs)

Section II contains detailed information about all ESC registers. This section is also common for all Beckhoff ESCs, thus registers, register bits, or features are described which might not be available in a specific ESC. Refer to the register overview and to the feature details overview in section III of a specific ESC to find out which registers and features are available.

Section III – hardware description (specific ESC)

Section III is ESC specific and contains detailed information about the ESC features, implemented registers, configuration, interfaces, pinout, usage, electrical and mechanical specification, and so on. Especially the process data interfaces (PDI) supported by the ESC are part of this section.

Additional documentation

Application notes and utilities can also be found at the Beckhoff homepage. Pinout configuration tools for EtherCAT ASICs are available. Additional information on EtherCAT IP Cores with latest updates regarding design flow compatibility, FPGA device support and known issues are also available.

DOCUMENT HISTORY

Version	Comment
1.0	Initial release
1.1	• Chapter Interrupts – AL event request: corrected AL Event mask register address to 0x0204:0x0207 • EtherCAT Datagram: Circulating frame bit has position 14 (not 13) • PHY addressing configuration changed • Loop control: a port using auto close mode is automatically opened if a valid Ethernet frame is received at this port • EEPROM read/write/reload example: steps 1 and 2 swapped • EEPROM: configured station alias (0x0012:0x0013) is only taken over at first EEPROM load after power-on or reset • SyncManager: watchdog trigger and interrupt generation in mailbox mode with single byte buffers requires alternating write and read accesses for some ESCs, thus buffered mode is required for digital I/O watchdog trigger generation • National Semiconductor DP83849I Ethernet PHY deprecated because of large link loss reaction time and delay • Added distinction between permanent ports and Bridge port (frame processing) • Added PDI chapter • PDI and DC Sync/Latch signals are high impedance until the SII EEPROM is successfully loaded • Editorial changes
1.2	• PHY address configuration revised. Refer to section III for ESC supported configurations • Added Ethernet link detection chapter • Added MI link detection and configuration, link detection descriptions updated • Added EEPROM Emulation for EtherCAT IP core • Added General Purpose Input chapter • Corrected minimum datagram sizes in EtherCAT header figure • Editorial changes
1.2.1	• Chapter 5.1.1: incompatible PHYs in footnote 1 deleted
1.3	• Added advisory for unused MII/RMII/EBUS ports • Ethernet PHY requirements revised: e.g., configuration by strapping options, recommendations enhanced. Footnote about compatible PHYs removed, information has moved to the EtherCAT slave controller application note “PHY Selection Guide”. • Frame error detection chapter enhanced • FIFO size reduction chapter enhanced • EBUS enhanced link detection chapter enhanced • Ethernet PHY link loss reaction time must be faster than 15 µs, otherwise use enhanced link detection • Enhanced link detection description corrected. Enhanced link detection does not remain active if it is disabled by EEPROM and EBUS handshake frames are received • ARMW/FRMW commands increase the working counter by 1 • Editorial changes
1.4	• Update to EtherCAT IP core release 2.1.0/2.01a • Added restriction to enhanced link configuration: RX_ER has to be asserted outside of frames (IEEE802 optional feature) • ESC power-on sequence for IP core corrected • Removed footnote on t_{Diff} figures, refer to section III for actual figures • Editorial changes

Version	Comment
1.5	• EEPROM read/write/Reload example: corrected register addresses • Updated/clarified PHY requirements, PHY link loss reaction time is mandatory • Enhanced link detection can be configured port-wise depending on ESC • Added DC Activation and DC Activation state features for some ESCs • ESC10 removed • Editorial changes
1.6	• Fill reserved EEPROM words of the ESC configuration area with 0 • Interrupt chapter: example for proper interrupt handling added • Use position addressing only for bus scanning at startup and to detect newly attached devices • System time PDI controlled: detailed description added • Added MII back-to-back connection example • Renamed Err(x) LED to PERR(x) • Editorial changes
1.7	• Link status description enhanced • Clarifications for DC system time and reference between clocks and registers • Chapter on avoiding unconnected port 0 configurations added • Direct ESC to standard Ethernet MAC MII connection added • MI link detection and configuration must not be used without link signals • Added criteria for detecting when DC synchronization is established • SII EEPROM interface is a point-to-point connection • PHY requirements: PHY startup should not rely on MDC clocking, ESD tolerance and baseline wander compensation recommendations added • Editorial changes
1.8	• Update to EtherCAT IP core release 2.3.0/2.03a • EEPROM acknowledge error (0x0502[13]) can also occur for a read access • ERR and STATE LED updated • Editorial changes
1.9	• EtherCAT state machine: additional AL status codes defined • EtherCAT protocol: LRD/LRW read data depends on bit mask • Updated EBUS enhanced link detection • Updated FMMU description • Loop control description updated • EtherCAT frame format (VLAN tag) description enhanced • Update to EtherCAT IP core release 2.3.2/2.03c
2.0	• Update to EtherCAT IP core release 2.4.0/2.04a • SII/ESI denotation now consistent with ETG • Updated AL status codes • Editorial changes
2.1	• Update to EtherCAT IP core release 3.0.0/3.00a • Update to ET1100-0003 and ET1200-0003 • RUN/ERR LED description enhanced • Added RGMII and FX operation • Added Gigabit Ethernet PHY chapter • Updated FIFO size configuration (default from SII) • Updated PHY address configuration • Added PDI register function acknowledge by write • Added propagation delay measurement in reverse mode (especially ET1200) • Enhanced ERR_LED description • Editorial changes
2.2	• Update to EtherCAT IP core release 3.0.6/3.00g • Added resetting distributed clocks time loop control filters to the synchronization steps • Extended Back-to-Back MII connection schematic • Clarified EBUS standard link detection restrictions • Editorial changes

Version	Comment
2.3	• Added EtherCAT P physical layer • Corrected PDI register function acknowledge by write for SyncManager Activation register 0x0806 • Added UDP/IP evaluation description, and checksum clearing • Added RGMII in-band signaling • Enhanced PHY/EBUS link detection chapters • Enhanced error counter description • Added optional features indicated in 0x0008:0x0009 • Added PHY reset and link partner notification/loop closing chapter • Altera is now Intel • Editorial changes
2.4	• Added second ESC configuration area, updated EEPROM and EEPROM emulation descriptions • Added second PDI (configuration, status, error counter, watchdog) • Added DC SyncSignals[3:2], and LatchSignals[3:2] • Added SyncManager deactivation delay • Added SyncManager sequential mode • Added EEPROM emulation example • Add SyncManager buffer description for ESCs with processing data width above 8 bit • Corrected description of register 0x0310 lost link counter • Further corrections to enhanced EBUS link detection chapter • Add RGMII back-to-back description • Intel is now Altera, Xilinx is now AMD • Editorial changes
2.5	• Update to EtherCAT IP core V4.0 • Added ET1150 • Add chapter on troubleshooting and FAQ (formerly part of an application note) • Editorial changes

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

µC	Microcontroller
ADR	Address
ADS	Automation Device Specification (Beckhoff)
AL	Application Layer
AMBA®	Advanced Microcontroller Bus Architecture from ARM®
APRD APWR APRW ARMW	Auto Increment Physical Read Auto Increment Physical Write Auto Increment Physical ReadWrite Auto Increment Physical Read Multiple Write
AoE	ADS over EtherCAT
ASIC	Application Specific Integrated Chip
Auto Crossover	Automatic detection of whether or not the send and receive lines are crossed.
Auto Negotiation	Automatic negotiation of transmission speeds between two stations.
Avalon®	On-chip bus for Altera® FPGAs
AXI™	Advanced eXtensible Interface Bus, an AMBA interconnect. Used as On-Chip-bus
Big Endian	Data format (also Motorola format). The more significant byte is transferred first when a word is transferred. However, for EtherCAT the least significant bit is the first on the wire.
BOOT	BOOT state of EtherCAT state machine
Boundary Clock	A station that is synchronized by another station and then passes this information on.
Bridge	A term for switches used in standards. Bridges are devices that pass on messages based on address information.
Broadcast	An unacknowledged transmission to an unspecified number of receivers.
BRD BWR BRW	Broadcast Read Broadcast Write Broadcast ReadWrite
Cat	Category – classification for cables that is also used in Ethernet. Cat 5 is the minimum required category for EtherCAT. However, Cat 6 and Cat 7 cables are available.
CoE	CAN® application layer over EtherCAT
Communication Stack	A communication software package that is generally divided into successive layers, which is why it is referred to as a stack.
Confirmed	Means that the initiator of a service receives a response.
CRC	Cyclic Redundancy Check, used for FCS

Cut Through	Procedure for cutting directly through an Ethernet frame by a switch before the complete message is received.
Cycle	Cycle in which data is to be exchanged in a system operating on a periodical basis.
DC	Distributed clocks Mechanism to synchronize EtherCAT slaves and master
Delay	Delays can be caused by run-times during transfer or internal delays of a network component.
Dest Addr	Destination address of a message (the destination can be an individual network station or a group (multicast).
DHCP	Dynamic Host Configuration Protocol, used to assign IP addresses (and other important startup parameter in the Internet context).
DL	Data Link Layer, also known as Layer 2. EtherCAT uses the data link layer of Ethernet, which is standardized as IEEE 802.3.
DNS	Domain Name Service, a protocol for domain name to IP addresses resolution.
EBUS	Based on LVDS (Low Voltage Differential Signaling) standard specified in ANSI/TIA/EIA-644-1995
ECAT	EtherCAT
EEPROM	Electrically Erasable Programmable Read Only Memory. Non-volatile memory used to store EtherCAT Slave Information (ESI). Connected to the SII.
EMC	ElectroMagnetic Compatibility, describes the robustness of a device with regard to electrical interference from the environment.
EMI	ElectroMagnetic Interference
Engineering	Here: All applications required to configure and program a machine.
EoE	Ethernet over EtherCAT
EOF	End of Frame
ERR	Error indicator for AL state
Err(x)	Physical Layer RX Error LED for debugging purposes
ESC	EtherCAT Slave Controller
ESI	EtherCAT Slave Information, stored in SII EEPROM
ESM	EtherCAT State Machine
ETG	EtherCAT Technology Group (http://www.ethercat.org)
EtherCAT	Real-time Standard for Industrial Ethernet Control Automation Technology (Ethernet for Control Automation Technology)
EtherCAT P	EtherCAT physical layer which combines EtherCAT communication and power in a single 4-wire standard Ethernet cable

EtherType	Identification of an Ethernet frame with a 16-bit number assigned by IEEE. For example, IP uses EtherType 0x0800 (hexadecimal) and the EtherCAT protocol uses 0x88A4.
EPU	EtherCAT Processing Unit. The logic core of an ESC containing e.g. registers, memory, and processing elements.
Fast Ethernet	Ethernet with a transmission speed of 100 Mbit/s.
FCC	Federal Communications Commission
FCS	Frame Check Sequence
FIFO	First In First Out
Firewall	Routers or other network component that acts as a gateway to the Internet and enables protection from unauthorized access.
FMMU	Fieldbus Memory Management Unit
FoE	File access over EtherCAT
Follow Up	Message that follows Sync and indicates when the Sync frame was sent from the last node (defined in IEEE 1588).
FPGA	Field Programmable Gate Array
FPRD FPWR FPRW FRMW	Configured Address Physical Read Configured Address Physical Write Configured Address Physical ReadWrite Configured Address Physical Read Multiple Write
Frame	See PDU
FTP	File Transfer Protocol
Get	Access method used by a client to read data from a device.
GND	Ground
GPI GPO	General Purpose Input General Purpose Output
HW	Hardware
I²C	Inter-Integrated Circuit, serial bus used for SII EEPROM connection
ICMP	Internet Control Message Protocol: Mechanisms for signaling IP errors.
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INIT	INIT state of EtherCAT state machine
Interval	Time span
IP	Internet Protocol: Ensures transfer of data on the Internet from end node to end node. Intellectual Property
IRQ	Interrupt Request

ISO	International Standard Organization
ISO/OSI Model	ISO Open Systems Interconnection Basic Reference Model (ISO 7498): describes the division of communication into 7 layers.
IT	Information Technology: Devices and methods required for computer-aided information processing.
LatchSignal	Signal for distributed clocks time stamping
LED	Light Emitting Diode, used as an indicator
Link/Act	Link/Activity Indicator (LED)
Little Endian	Data format (also Intel format). The less significant byte is transferred first when a word is transferred. With EtherCAT, the least significant bit is the first on the wire.
LLDP	Lower Layer Discovery Protocol – provides the basis for topology discovery and configuration definition (see IEEE802.1ab)
LRD LWR LRW	Logical Read Logical Write Logical ReadWrite
LVDS	Low Voltage Differential Signaling
M12	Connector used for industrial Ethernet
MAC	Media Access Control: Specifies station access to a communication medium. With full duplex Ethernet, any station can send data at any time; the order of access and the response to overload are defined at the network component level (switches).
MAC Address	Media Access Control Address: Also known as Ethernet address; used to identify an Ethernet node. The Ethernet address is 6 bytes long and is assigned by the IEEE.
Mandatory Services	Mandatory services, parameters, objects, or attributes. These must be implemented by every station.
MBX	Mailbox
MDI	Media Dependent Interface: Use of connector Pins and Signaling (PC side)
MDI-X	Media Dependent Interface (crossed): Use of connector Pins and Signaling with crossed lines (Switch/hub side)
MI	(PHY) Management Interface
MII	Media Independent Interface: Standardized interface between the Ethernet MAC and PHY.
Multicast	Transmission to multiple destination stations with a frame – generally uses a special address.
NOP	No Operation
NVRAM	Non-volatile random access memory, e.g. EEPROM or flash.

Octet	Term from IEC 61158 – one octet comprises exactly 8 bits.
OP	Operational state of EtherCAT state machine
OPB	On-Chip Peripheral Bus
Optional Service	Optional services can be fulfilled by a PROFINET station in addition to the mandatory services.
OSI	Open System Interconnect
OTP	One-time programmable memory, used for ESC configuration storage
OUI	Organizationally Unique Identifier – the first 3 Bytes of an Ethernet address that will be assign to companies or organizations and can be used for protocol identifiers as well (e.g. LLDP)
PDI	Process Data Interface or Physical Device Interface: an interface that allows access to ESC from the process side.
PDO	Process Data Object
PDU	Protocol Data Unit: Contains protocol information (src addr, dest addr, checksum and service parameter information) transferred from a protocol instance of transparent data to a subordinate level (the lower level contains the information being transferred).
PE	Protection Earth
PHY	Physical layer device that converts data from the Ethernet controller to electric or optical signals.
Ping	Frame that verifies whether the partner device is still available.
PLB	Processor Local Bus
PLL	Phase Locked Loop
PREOP	Pre-Operational state of EtherCAT state machine
Priority Tagging	Priority field inserted in an Ethernet frame.
Protocol	Rules for sequences – here, also the sequences (defined in state machines) and frame structures (described in encoding) of communication processes.
Provider	Device that sends data to other consumers in the form of a broadcast message.
PTP	Precision Time Protocol in accordance with IEEE 1588: precise time synchronization procedures.
PTP Master	Indicates time in a segment.
PTP Slave	Station synchronized by a PTP master.
Quad Cable	Cable type in which the two cable pairs are twisted together. This strengthens the electromagnetic resistance.
RAM	Random Access Memory. ESC have user RAM and process data RAM.
Read	Service enabling read access to an I/O device.

Real-Time	Real-time capability of a system to perform a task within a specific time.
Request	Call of a service in the sender/client.
Response	Response to a service on the client side.
RJ45	FCC Registered Jack, standard Ethernet connector (8P8C)
RMII	Reduced Media Independent Interface
Router	Network component acting as a gateway based on the interpretation of the IP address.
RSTP	Rapid Spanning Tree Protocol: Prevents packet from looping infinitely between switches; RSTP is specified in IEEE 802.1 D (Edition 2004)
RT	Real-time. Name for a real-time protocol that can be run in Ethernet controllers without special support.
RTC	Real-time Clock chip of PCs
RT Frames	EtherCAT Messages with EtherType 0x88A4.
RX	Receive
RXPDO	Receive PDO, i.e. Process Data that will be received by ESC
RUN	RUN indicator (LED) for application state
SAFEOP	Safe-Operational state of EtherCAT state machine
Safety	Safety function, implemented by an electric, electronic programmable fail-safe system that maintains the equipment in a safe state, even during certain critical external events.
Schedule	Determines what should be transferred and when.
Services	Interaction between two components to fulfill a specific task.
Set	Access method used by a client to write data to a server.
SII	Slave Information Interface
SM	SyncManager
SNMP	Simple Network Management Protocol: SNMP is the standard Internet protocol for management and diagnostics of network components (see also RFC 1157 and RFC 1156 at www.ietf.org).
SoE	Servo Profile over EtherCAT
SOF	Start of Frame Ethernet SOF delimiter at the end of the preamble of Ethernet frames
SPI	Serial Peripheral Interface
Src Addr	Source address: source address of a message.
Store and Forward	Currently the common operating mode in switches. Frames are first received in their entirety, the addresses are evaluated, and then they are forwarded. This

	result in considerable delays, but guarantees that defective frames are not forwarded, causing an unnecessary increase in the bus load.
STP	Shielded Twisted Pair: Shielded cable with at least 2 core pairs to be used as the standard EtherCAT cable.
Subnet mask	Divides the IP address into two parts: a subnet address (in an area separated from the rest by routers) and a network address.
Switch	Also known as Bridge. Active network component to connect different EtherCAT participants with each other. A switch only forwards the frames to the addressed participants.
SyncManager	ESC unit for coordinated data exchange between master and slave µController
SyncSignal	Signal generated by the distributed clocks unit
TCP	Transmission Control Protocol: Higher-level IP protocol that ensures secure data exchange and flow control.
TX	Transmit
TXPDO	Transmit PDO, i.e. process data that will be transmitted by ESC
UDP	User Datagram Protocol: Non-secure multicast/broadcast frame.
UTP	Unshielded Twisted Pair: unshielded cable with at least 2 core pairs are not recommended for industrial purpose but are commonly used in areas with low electro-magnetic interference.
VLAN	Virtual LAN
VoE	Vendor specific profile over EtherCAT
WD	Watchdog
WKC	Working Counter
XML	Extensible Markup Language: standardized definition language that can be interpreted by nearly all parsers.
XML Parser	Program for checking XML schemas.

1 EtherCAT slave controller overview

An EtherCAT slave controller (ESC) takes care of the EtherCAT communication as an interface between the EtherCAT fieldbus and the slave application. This document covers the following Beckhoff ESCs: ASIC implementations functionally fixed binary configurations for FPGAs, and configurable IP Cores for FPGAs.

Table 1: ESC main features

Feature	ET1200	ET1100	ET1150	IP core	ESC20
Ports	2-3 (each EBUS/MII, max. 1xMII)	2-4 (each EBUS/MII)	1-4 (each MII/ RGMII)	1-4 (each MII/RGMII/ RMII)	2 MII
FMMUs	3	8	16	0-16	4
SyncManagers	4	8	16	0-16	4
RAM [Kbyte]	1	8	15	0-60	4
Distributed clocks	64 bit	64 bit	64 bit	32/64 bit	32 bit
Process data interfaces	1	1	2	2	1
Digital I/O	16 bit	32 bit	32 bit	8-32 bit	32 bit
SPI slave	Yes	Yes	Yes	Yes	Yes
SPI master	-	-	Yes	-	-
µController	-	8/16 bit async/sync	8/16/32 bit async/sync	8/16/32 bit async	8/16 bit async
On-chip bus	-	-	-	Yes	-

The general functionality of an ESC is shown in Figure 1:

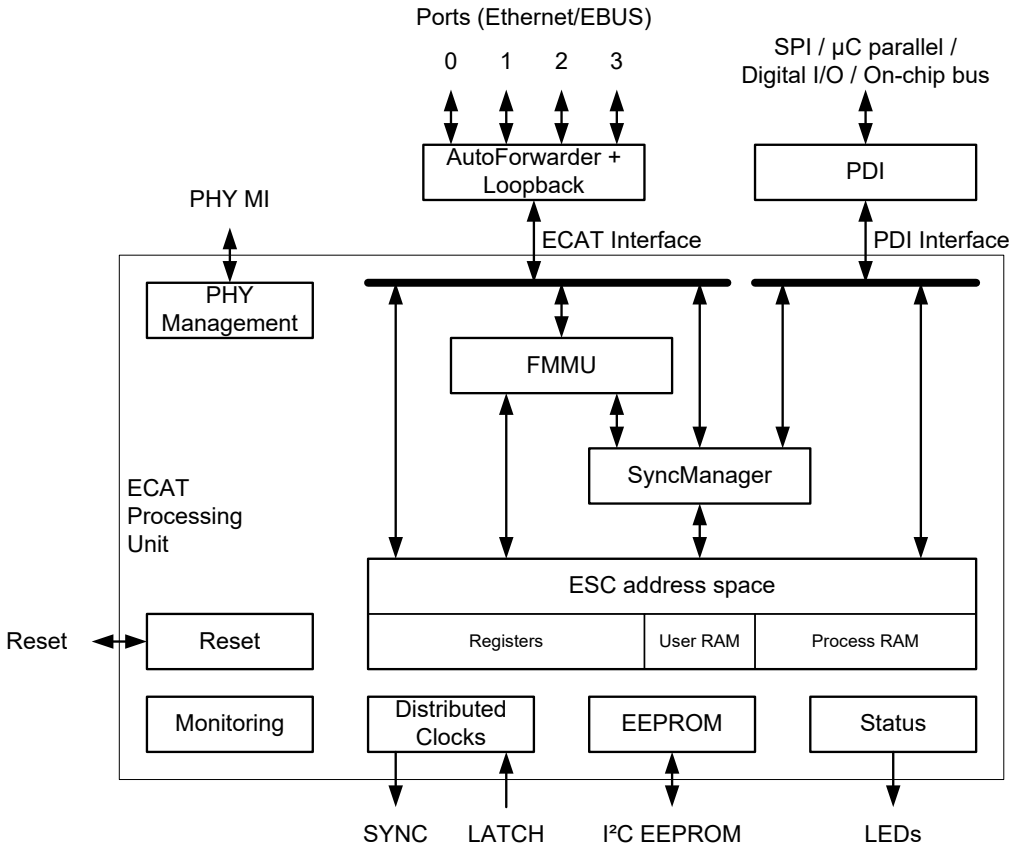


Figure 1: EtherCAT slave controller block diagram

1.1 EtherCAT slave controller function blocks

EtherCAT interfaces (Ethernet/EBUS)

The EtherCAT interfaces or ports connect the ESC to other EtherCAT slaves and the master. The MAC layer is integral part of the ESC. The physical layer may be Ethernet or EBUS. The physical layer for EBUS is fully integrated into the ASICs. For Ethernet ports, external Ethernet PHYs connect to the MII/RGMII/RMII ports of the ESC. Transmission speed for EtherCAT is fixed to 100 Mbit/s with full duplex communication. Link state and communication status are reported to the Monitoring device. EtherCAT slaves support 2-4 ports, the logical ports are numbered 0-1-2-3, formerly they were denoted by A-B-C-D.

EtherCAT processing unit

The EtherCAT processing unit (EPU) receives, analyses, and processes the EtherCAT data stream. It is logically located between port 0 and port 3. The main purpose of the EtherCAT processing unit is to enable and coordinate access to the internal registers and the memory space of the ESC, which can be addressed both from the EtherCAT master and from the local application via the PDI. Data exchange between master and slave application is comparable to a dual-ported memory (process memory), enhanced by special functions e.g. for consistency checking (SyncManager) and data mapping (FMMU). The EtherCAT processing Units contains the main function blocks of EtherCAT slaves besides auto-Forwarding, loop-back function, and PDI.

Auto-forwarder

The auto-forwarder receives the Ethernet frames, performs frame checking and forwards it to the loop-back function. Time stamps of received frames are generated by the auto-forwarder.

Loop-back function

The loop-back function forwards Ethernet frames to the next logical port if there is either no link at a port, or if the port is not available, or if the loop is closed for that port. The loop-back function of port 0 forwards the frames to the EtherCAT processing unit. The loop settings can be controlled by the EtherCAT master.

FMMU

Fieldbus memory management Units are used for bitwise mapping of logical addresses to physical addresses of the ESC.

SyncManager

SyncManagers are responsible for consistent data exchange and mailbox communication between EtherCAT master and slaves. The communication direction can be configured for each SyncManager. Read or write transactions may generate events for the EtherCAT master and an attached µController respectively. The SyncManagers are responsible for the main difference between and ESC and a dual-ported memory, because they map addresses to different buffers and block accesses depending on the SyncManager state. This is also a fundamental reason for bandwidth restrictions of the PDI.

Monitoring

The Monitoring unit contains error counters and watchdogs. The watchdogs are used for observing communication and returning to a safe state in case of an error. Error counters are used for error detection and analysis.

Reset

The integrated reset controller observes the supply voltage and controls external and internal resets (ET1100 and ET1200 ASICs only).

PHY management

The PHY management unit communicates with Ethernet PHYs via the MII management interface. This is either used by the master or by the slave. The MII management interface is used by the ESC itself for optionally restarting auto negotiation after receive errors with the enhanced link detection mechanism, and for the optional MI link detection and configuration feature.

Distributed clock

Distributed clocks (DC) allow for precisely synchronized generation of output signals and input sampling, as well as time stamp generation of events. The synchronization may span the entire EtherCAT network.

Memory
An EtherCAT slave can have an address space of up to 64Kbyte. The first block of 4 Kbyte (0x0000-0x0FFF) is used for registers and user memory. The memory space from address 0x1000 onwards is used as the process memory (up to 60 Kbyte). The size of process memory depends on the device. The ESC address range is directly addressable by the EtherCAT master and an attached µController.

Process data interface (PDI) or application interface
There are several types of PDIs available, depending on the ESC:

- Digital I/O (8-32 bit, unidirectional/bidirectional, with DC support)
- SPI slave
- SPI master
- 8/16/32 bit µController (asynchronous or synchronous)
- On-chip bus (e.g., Avalon® , PLB®, or AXI®, depending on target FPGA type and selection)
- General purpose I/O

The PDIs are described in section III of the particular ESC, since the PDI functions are highly depending on the ESC type.

SII EEPROM
One non-volatile memory is needed for EtherCAT slave information (ESI) storage, typically an I²C EEPROM. If the ESC is implemented as an FPGA, a second non-volatile memory is necessary for the FPGA configuration code.

Status / LEDs
The status block provides ESC and application status information. It controls external LEDs like the application RUN LED/ERR LED and port link/activity LEDs.

1.2 Further reading on EtherCAT and ESCs

For further information on EtherCAT, refer to the EtherCAT specification ETG.1000, available from the EtherCAT Technology Group (ETG, http://www.ethercat.org), and the IEC standard “Digital data communications for measurement and control – Fieldbus for use in industrial control systems”, IEC 61158 Type 12: EtherCAT, available from the IEC (http://www.iec.ch).

Additional documents on EtherCAT can be found on the EtherCAT Technology Group website (http://www.ethercat.org).

Documentation on Beckhoff Automation EtherCAT slave controllers is available at the Beckhoff website (http://www.beckhoff.com), e.g., data sheets, application notes, and ASIC pinout configuration tools.

1.3 Scope of section I

Section I deals with the basic EtherCAT technology. Starting with the EtherCAT protocol itself, the frame processing inside EtherCAT slaves is described. The features and interfaces of the physical layer with its two alternatives Ethernet and EBUS are explained afterwards. Finally, the details of the functional units of an ESC like FMMU, SyncManager, distributed clocks, slave information interface, interrupts, watchdogs, and so on, are described.

Since section I is common for all Beckhoff ESCs, it contains features which might not be available in every individual ESC. Refer to the feature details overview in section III of a specific ESC to find out which features are actually available.

The following Beckhoff ESCs are covered by section I:

- ET1200-0003
- ET1100-0003
- ET1150-0002
- EtherCAT IP core for FPGAs (V4.0)
- ESC20 (Build 22)

2 EtherCAT protocol

EtherCAT uses standard IEEE 802.3 Ethernet frames, thus a standard network controller can be used and no special hardware is required on master side.

EtherCAT has a reserved EtherType of 0x88A4 that distinguishes it from other Ethernet frames. Thus, EtherCAT can run in parallel to other Ethernet protocols[1].

EtherCAT does not require the IP protocol, however it can be encapsulated in IP/UDP. The EtherCAT slave controller processes the frame in hardware. Thus, communication performance is independent from processor power.

An EtherCAT frame is subdivided into the EtherCAT frame header followed by one or more EtherCAT datagrams. At least one EtherCAT datagram has to be in the frame. Only EtherCAT frames with type 1 in the EtherCAT Header are currently processed by the ESCs. The ESCs also support IEEE802.1Q VLAN tags, although the VLAN tag contents are not evaluated by the ESC.

If the Ethernet frame size falls below 64 byte, padding bytes have to be added until this size is reached. Otherwise the EtherCAT frame is exactly as large as the sum of all EtherCAT datagrams plus EtherCAT frame header.

2.1 EtherCAT header

Figure 2 shows how an Ethernet frame containing EtherCAT data is assembled.

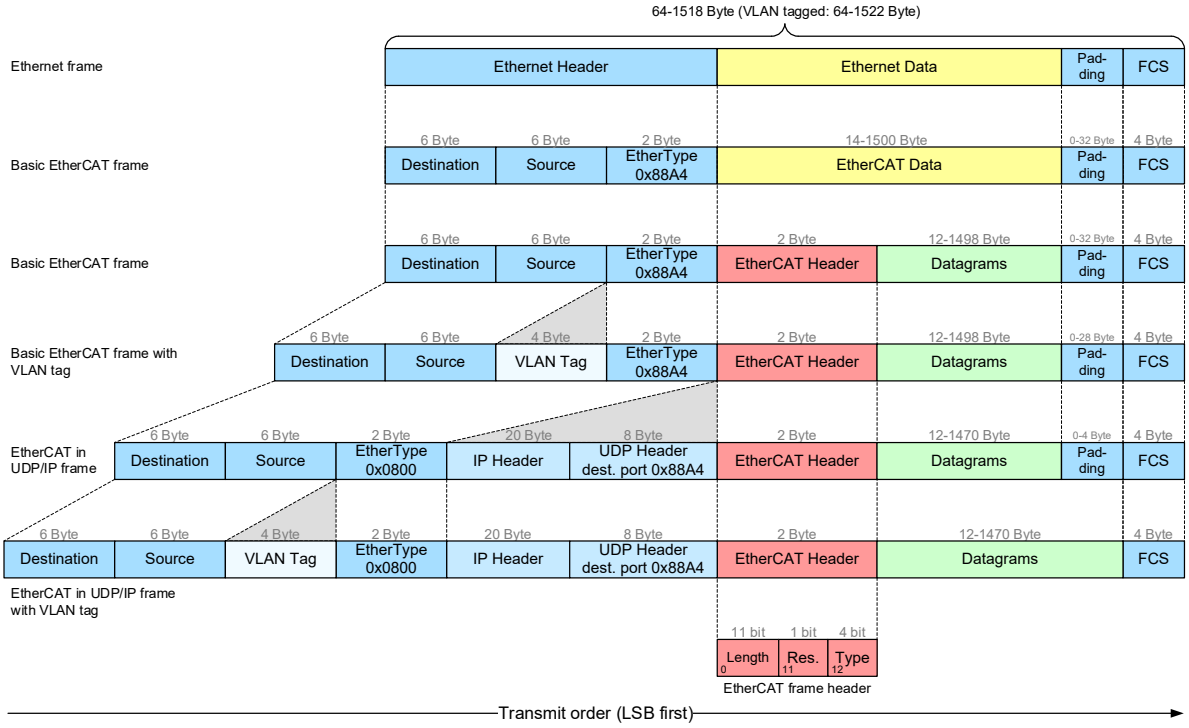


Figure 2: Ethernet frame with EtherCAT data

Table 2: EtherCAT frame header

Field	Data type	Header bits	Value/description
Length	11 bit	[10:0]	Length of the EtherCAT datagrams (excl. FCS)
Reserved	1 bit	[11]	Reserved, 0
Type	4 bit	[15:12]	Protocol type. Only EtherCAT commands (Type = 0x1) are supported by ESCs.

NOTE: The EtherCAT header length field is ignored by ESCs, they rely on the datagram length fields. Transmission order is least significant bit/byte first.

[1] ESCs have to be configured to forward non-EtherCAT frames via DL control register 0x0100[0].

2.2 EtherCAT datagram

Figure 3 shows the structure of an EtherCAT datagram.

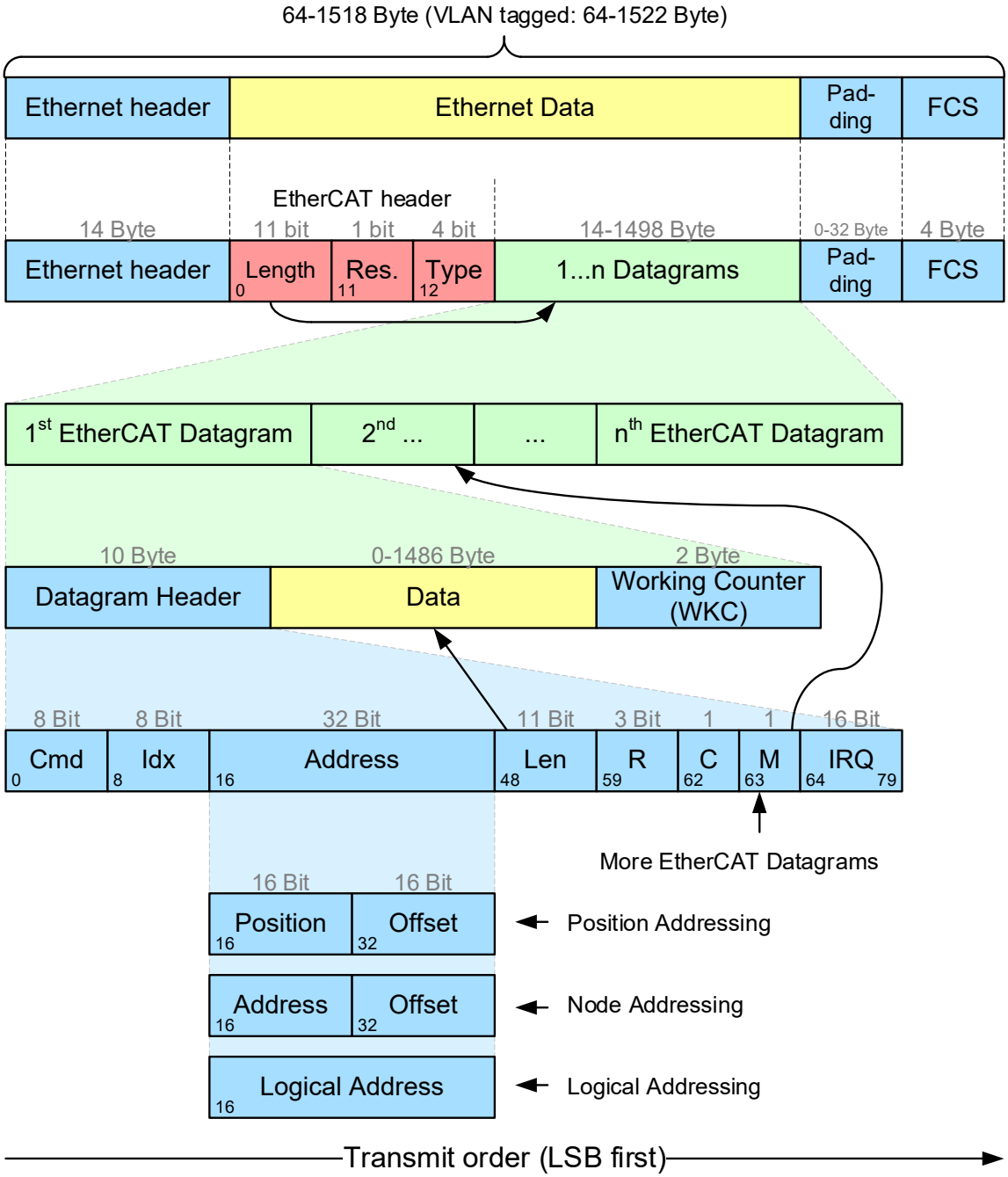


Figure 3: EtherCAT datagram

Table 3: EtherCAT datagram

Field	Data type	Header bits	Value/description
Cmd	BYTE	[7:0]	EtherCAT command type (see 2.5)
Idx	BYTE	[15:8]	The index is a numeric identifier used by the master for identification of duplicates/lost datagrams. It shall not be changed by EtherCAT slaves
Address	BYTE[4]	[47:16]	Address (auto increment, configured station address, or logical address, see 2.3)
Len	11 bit	[58:48]	Length of the following data within this datagram
R	3 bit	[61:59]	Reserved, 0
C	1 bit	[62]	Circulating frame (see 3.5): 0: Frame is not circulating 1: Frame has circulated once
M	1 bit	[63]	More EtherCAT datagrams 0: Last EtherCAT datagram 1: More EtherCAT datagrams will follow
IRQ	WORD	[79:64]	EtherCAT event request registers of all slaves combined with a logical OR
Data	BYTE[n]		Read/write data
WKC	WORD		Working counter (see 2.4)

2.3 EtherCAT addressing modes

Two addressing modes of EtherCAT devices are supported within one segment: device addressing and logical addressing. Three device addressing modes are available: auto increment addressing, configured station address, and broadcast. EtherCAT devices can have up to two configured station addresses, one is assigned by the master (configured station address), the other one is stored in the SII EEPROM and can be changed by the slave application (configured station alias address). The EEPROM setting for the configured station alias address is only taken over at the first EEPROM loading after power-on or reset.

Table 4: EtherCAT addressing modes

Mode	Field	Data Type	Value/description
Auto increment address	Position	WORD	Each slave increments position. Slave is addressed if position = 0.
	Offset	WORD	Local register or memory address of the ESC
Configured station address	Address	WORD	Slave is addressed if address matches configured station address or configured station alias (if enabled).
	Offset	WORD	Local register or memory address of the ESC
Broadcast	Position	WORD	Each slave increments position (not used for addressing)
	Offset	WORD	Local register or memory address of the ESC
Logical address	Address	DWORD	Logical address (configured by FMMUs) Slave is addressed if FMMU configuration matches address.

2.3.1 Device addressing

The device can be addressed via device position address (auto increment address), by node address (configured station address/configured station alias), or by a broadcast.

- Position address / auto increment address:
  The datagram holds the position address of the addressed slave as a negative value. Each slave increments the address. The slave which reads the address equal zero is addressed and will execute the appropriate command at receive.
  Position addressing should only be used during start-up of the EtherCAT system to scan the fieldbus and later only occasionally to detect newly attached slaves. Using position addressing is problematic if loops are closed temporarily due to hot connecting or link problems. position addresses are shifted in this case, and e.g., a mapping of error register values to devices becomes impossible, thus the faulty link cannot be localized.
- Node address / configured station address and configured station alias:
  The configured station address is assigned by the master during start up and cannot be changed by the EtherCAT slave. The configured station alias address is stored in the SII EEPROM and can be changed by the EtherCAT slave. The configured station alias has to be enabled by the master. The appropriate command action will be executed if node address matches with either configured station address or configured station alias.
  Node addressing is typically used for register access to individual and already identified devices.
- Broadcast:
  Each EtherCAT slave is addressed.
  Broadcast addressing is used e.g. for initialization of all slaves and for checking the status of all slaves if they are expected to be identical.

Each slave device has a 16 bit local address space (address range 0x0000:0x0FFF is dedicated for EtherCAT registers, address range 0x1000:0xFFFF is used as process memory) which is addressed via the offset field of the EtherCAT datagram. The process memory address space is used for application communication (e.g. mailbox access).

2.3.2 Logical addressing

All devices read from and write to the same logical 4 Gbyte address space (32 bit address field within the EtherCAT datagram). A slave uses a mapping unit (FMMU, Fieldbus memory management unit) to map data from the logical process data image to its local address space. During start up the master configures the FMMUs of each slave. The slave knows which parts of the logical process data image have to be mapped to which local address space using the configuration information of the FMMUs.

Logical addressing supports bit wise mapping. Logical addressing is a powerful mechanism to reduce the overhead of process data communication, thus it is typically used for accessing process data.

2.4 Working counter

Every EtherCAT datagram ends with a 16 Bit working counter (WKC). The working counter counts the number of devices that were successfully addressed by this EtherCAT datagram. Successfully means that the ESC is addressed and the addressed memory is accessible (e.g., protected SyncManager buffer). EtherCAT slave controllers increment the working counter in hardware. Each datagram should have an expected working counter value calculated by the master. The master can check the valid processing of EtherCAT datagrams by comparing the working counter with the expected value.

The working counter is increased if at least one byte/one bit of the whole multi-byte datagram was successfully read and/or written. For a multi-byte datagram, you cannot tell from the working counter value if all or only one byte was successfully read and/or written. This allows reading separated register areas using a single datagram by ignoring unused bytes.

The read-multiple-write commands ARMW and FRMW are either treated like a read command or like a write command, depending on the address match.

Table 5: Working counter increment

Command	Data type	Increment
Read command	No success	no change
	Successful read	+1
Write command	No success	no change
	Successful write	+1
Read-write command	No success	no change
	Successful read	+1
	Successful write	+2
	Successful read and write	+3

2.5 EtherCAT command types

All supported EtherCAT command types are listed in Table 6. For read-write operations, the read operation is performed before the write operation.

Table 6: EtherCAT command types

CMD	Abbr.	Name	Description
0	NOP	No operation	Slave ignores command
1	APRD	Auto increment read	Slave increments address. Slave puts read data into the EtherCAT datagram if received address is zero.
2	APWR	Auto increment write	Slave increments address. Slave writes data into memory location if received address is zero.
3	APRW	Auto increment read write	Slave increments address. Slave puts read data into the EtherCAT datagram and writes the data into the same memory location if received address is zero.
4	FPRD	Configured address read	Slave puts read data into the EtherCAT datagram if address matches with one of its configured addresses
5	FPWR	Configured address write	Slave writes data into memory location if address matches with one of its configured addresses
6	FPRW	Configured address read write	Slave puts read data into the EtherCAT datagram and writes data into the same memory location if address matches with one of its configured addresses
7	BRD	Broadcast read	All slaves put logical OR of data of the memory area and data of the EtherCAT datagram into the EtherCAT datagram. All slaves increment position field.
8	BWR	Broadcast write	All slaves write data into memory location. All slaves increment position field.
9	BRW	Broadcast read write	All slaves put logical OR of data of the memory area and data of the EtherCAT datagram into the EtherCAT datagram, and write data into memory location. BRW is typically not used. All slaves increment position field.
10	LRD	Logical memory read	Slave puts read data into the EtherCAT datagram if received address matches with one of the configured FMMU areas for reading.
11	LWR	Logical memory write	Slaves writes data to into memory location if received address matches with one of the configured FMMU areas for writing.
12	LRW	Logical memory read write	Slave puts read data into the EtherCAT datagram if received address matches with one of the configured FMMU areas for reading. Slaves writes data to into memory location if received address matches with one of the configured FMMU areas for writing.
13	ARMW	Auto increment read Multiple write	Slave increments address. Slave puts read data into the EtherCAT datagram if received address is zero, otherwise slave writes the data into memory location.
14	FRMW	Configured read multiple write	Slave puts read data into the EtherCAT datagram if address matches with one of its configured addresses, otherwise slave writes the data into memory location.
15-255		reserved	

Table 7: EtherCAT command details

CMD	High Addr. In	High Addr. Out	Low Addr.	Address Match	Data In	Data Out	WKC
NOP	untouched			none	untouched		
APRD	Position	Pos.+1	Offset	ADP=0	-	Read	+0/1
APWR	Position	Pos.+1	Offset	ADP=0	Write		+0/1
APRW	Position	Pos.+1	Offset	ADP=0	Write	Read	+0/1/2/3
FPRD	Address		Offset	ADP= conf. station address./alias	-	Read	+0/1
FPWR	Address		Offset	ADP= conf. station address./alias	Write		+0/1
FPRW	Address		Offset	ADP= conf. station address./alias	Write	Read	+0/1/2/3
BRD		High Add. In+1	Offset	all		Data In *OR* Read	+0/1
BWR		High Add. In+1	Offset	all	Write		+0/1
BRW		High Add. In+1	Offset	all	Write	Data In *OR* Read	+0/1/2/3
LRD	Logical address			FMMU	-	(Read *AND* bitmask[1]) *OR* (Data In *AND NOT* bit_mask[1])	+0/1
LWR	Logical address			FMMU	Write		+0/1
LRW	Logical address			FMMU	Write	(Read *AND* bit_mask[1]) *OR* (Data In *AND NOT* bit_mask[1])	+0/1/2/3
ARMW	Position	Pos.+1	Offset	Read: ADP=0	-	Read	+1
				Write: ADP/=0	Write		+1
FRMW	Address		Offset	Read: ADP= conf. station address./alias	-	Read	+1
				Write: ADP/= conf. station address./alias	Write		+1

NOTE: Working counter (WKC) increment depends on address match

[1] bit_mask depends on FMMU configuration if bit-wise mapping is used: only masked bits are actually addressed by the logical read/write command.

2.6 UDP/IP

The following header fields are evaluated by an ESC to detect an EtherCAT frame encapsulated in UDP/IP:

Table 8: EtherCAT UDP/IP encapsulation

Field	Expected value for EtherCAT
EtherType	0x0800 (IP)
IP version	4
IP header length	5
IP protocol	0x11 (UDP)
UDP destination port	0x88A4

All other fields if the IP and UDP header are not evaluated, and the UDP checksum is not checked.

Since EtherCAT frames are processed on the fly, the UDP checksum cannot be updated by an ESC when the frame content is modified. Instead, the ESC clears the UDP checksum for any EtherCAT frame (regardless of DL control register 0x0100[0]), which indicates that the checksum is not used. The UDP checksum is forwarded without modification for non-EtherCAT frames if DL control register 0x0100[0]=0.

3 Frame processing

The Beckhoff EtherCAT slave controllers only support direct mode addressing: neither a MAC address nor an IP address is assigned to the ESC, they process EtherCAT frames with any MAC or IP address.

It is not possible to use unmanaged switches between these ESCs or between master and the first slave, because source and destination MAC addresses are not evaluated or exchanged by the ESCs. Only the source MAC address is modified when using the default settings, so outgoing and incoming frames can be distinguished by the master.

NOTE: Attaching an ESC directly to an office network will result in network flooding, since the ESC will reflect any frame – especially broadcast frames – back into the network (broadcast storm).

The frames are processed by the ESC on the fly, i.e., they are not stored inside the ESC. Data is read and written as the bits are passing the ESC. The forwarding delay is minimized to achieve fast cycle times. The forwarding delay is defined by the receive FIFO size and the EtherCAT processing unit delay. A transmit FIFO is omitted to reduce delay times.

The ESCs support EtherCAT, UDP/IP, and VLAN tags. EtherCAT frames and UDP/IP frames containing EtherCAT datagrams are processed. Frames with VLAN tags are processed by the ESCs, the VLAN settings are ignored and the VLAN tag is not modified.

The source MAC address is changed for every frame passing the EtherCAT processing unit (SOURCE_MAC[1] is set to 1 – locally administered address). This helps to distinguish between frames transmitted by the master and frames received by the master.

3.1 Loop control and loop state

Each port of an ESC can be in one of two states: open or closed. If a port is open, frames are transmitted to other ESCs at this port, and frames from other ESCs are received. A port which is closed will not exchange frames with other ESCs, instead, the frames are forwarded internally to the next logical port, until an open port is reached.

The loop state of each port can be controlled by the master (ESC DL control register 0x0100). The ESCs supports four loop control settings, two manual configurations, and two automatic modes:

Manual open
The port is open regardless of the link state. If there is no link, outgoing frames will be lost.

Manual close
The port is closed regardless of the link state. No frames will be sent out or received at this port, even if there is a link with incoming frames.

Auto
The loop state of each port is determined by the link state of the port. The loop is open if there is a link, and it is closed without a link.

Auto close (manual open)

The port is closed depending on the link state, i.e., if the link is lost, the loop will be closed (auto close). If the link is established, the loop will not be automatically opened, instead, it will remain closed (closed wait state). Typically, the port has to be opened by the master explicitly by writing the loop configuration again to the ESC DL control register 0x0100. This write access has to enter the ESC via a different open port. There is an additional fallback option for opening the port: if a valid Ethernet frame is received from the external link at the closed port in auto close mode, it will also be opened after the CRC is received correctly. The content of the frame is not evaluated.

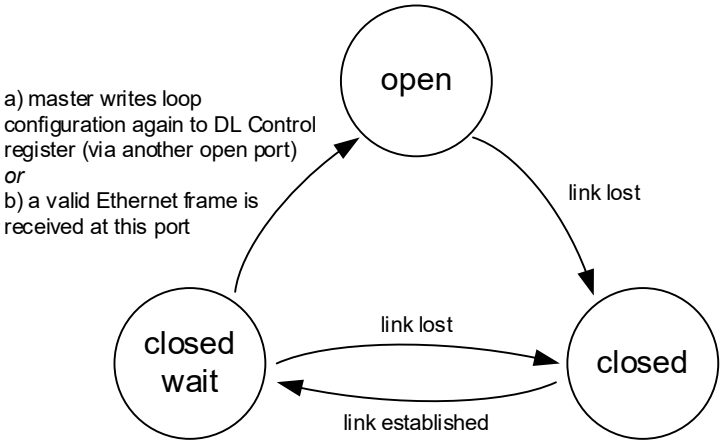


Figure 4: Auto close loop state transitions

A port is considered open if the port is available, i.e., it is enabled in the configuration, and one of the following conditions is met:

- The loop setting in the DL control register is auto and there is an active link at the port.
- The loop setting in the DL control register is auto close and there is an active link at the port and the DL control register was written again after the link was established.
- The loop setting in the DL control register is auto close and there is an active link at the port and a valid frame was received at this port after the link was established.
- The loop setting in the DL control register is always open

A port is considered closed if one of the following conditions is met:

- The port is not available or not enabled in the configuration.
- The loop setting in the DL control register is auto and there is no active link at the port.
- The loop setting in the DL control register is auto close and there is no active link at the port or the DL control register was not written again after the link was established
- The loop setting in the DL control register is always closed

NOTE: If all ports are closed (either manually or automatically), port 0 will be opened as the recovery port. Reading and writing via this port is possible, although the DL status register reflects the correct status. This can be used to correct DL control register settings.

Registers used for loop control and loop/link status are listed in Table 9.

Table 9: Registers for loop control and loop/link status

Register address	Name	Description
0x0100[15:8]	ESC DL control	Loop control/loop setting
0x0110[15:4]	ESC DL status	Loop and link status
0x0518:0x051B	PHY port status	PHY management link status

3.2 Frame processing order

The frame processing order of EtherCAT slave controllers depends on the number of ports (logical port numbers are used):

Table 10: Frame processing order

Number of ports	Frame processing order
1	0→EtherCAT processing unit→0
2	0→EtherCAT processing unit→1 / 1→0
3	0→EtherCAT processing unit→1 / 1→2 / 2→0 (log. ports 0,1, and 2) or 0→EtherCAT processing unit→3 / 3→1 / 1→0 (log. ports 0,1, and 3)
4	0→EtherCAT processing unit→3 / 3→1 / 1→2 / 2→0

The direction through an ESC including the EtherCAT processing unit is called “processing” direction, other directions without passing the EtherCAT processing unit are called “forwarding” direction.

Ports which are not implemented behave similar to closed ports, the frame is forwarded to the next port.

Figure 5 shows the frame processing in general:

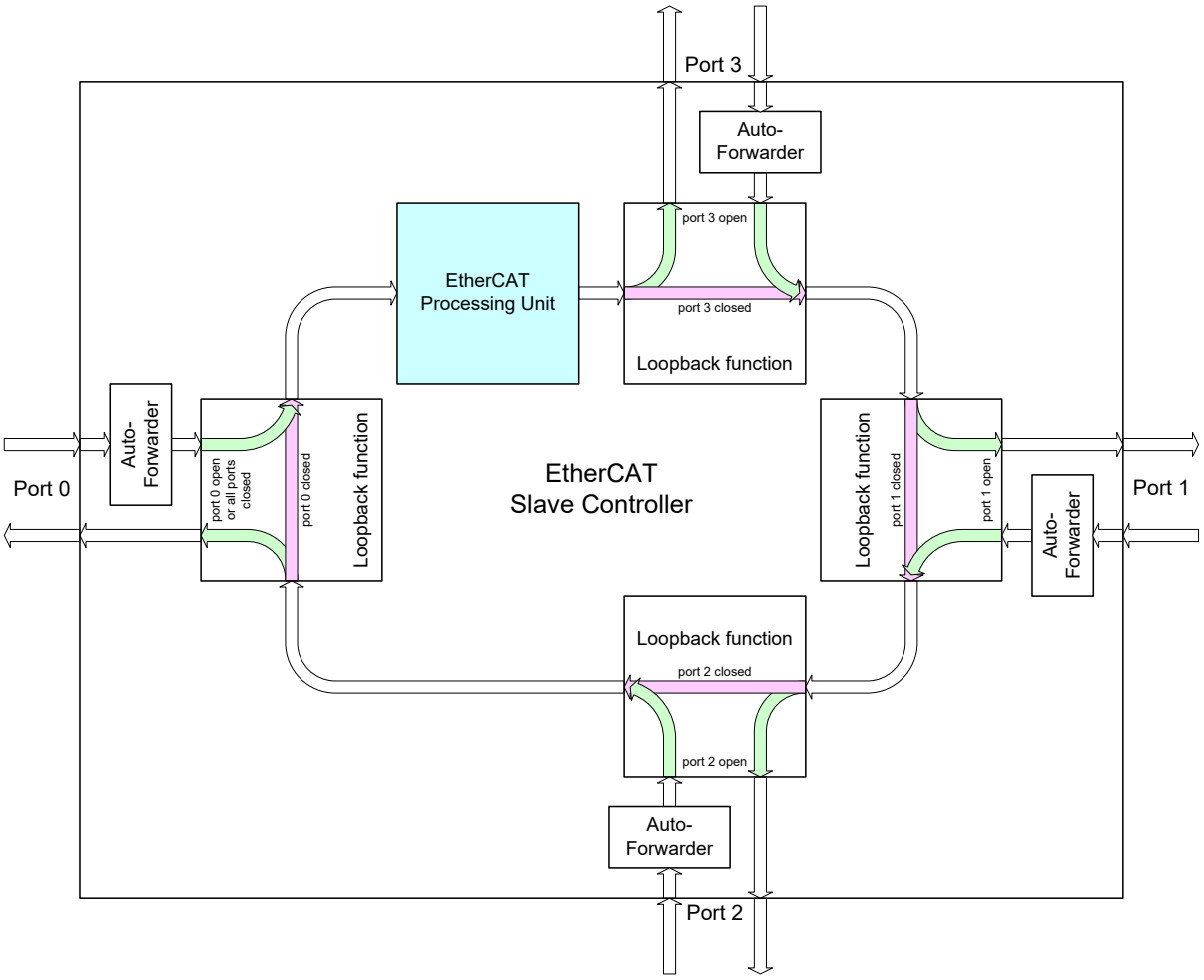


Figure 5: Frame processing

Example port configuration with ports 0, 1, and 2

If there are only ports 0, 1, and 2, a frame received at port 0 goes via the auto-forwarder and the loopback function to the EtherCAT processing unit which processes it. Then, the frame is sent to logical port 3 which is not configured, so the loopback function of port 3 forwards it to port 1. If port 1 is closed, the frame is forwarded by the loopback function to port 2. If port 1 is open, the frame is sent out at port 1. When the frame comes back into port 1, it is handled by the auto-forwarder and sent to port 2. Again, if port 2 is closed, the frame is forwarded to port 0, otherwise, it is sent out at port 2. When the frame comes back into port 2, it is handled by the auto-forwarder and then sent to the loopback function of port 0. Then it is handled by the loopback function and sent out at port 0 – back to the master.

3.3 Permanent ports and bridge port

The EtherCAT ports of an ESC are typically permanent ports, which are directly available after power-on. Permanent ports are initially configured for auto mode, i.e., they are opened after the link is established. Additionally, some ESCs support an EtherCAT bridge port (port 3), which is configured in the SII EEPROM like PDI interfaces. This bridge port becomes available if the EEPROM is loaded successfully, and it is closed initially, i.e., it has to be opened (or set to auto mode) explicitly by the EtherCAT master.

3.4 Shadow buffer for register write operations

The ESCs have shadow buffers for write operations to registers (0x0000 to 0x0F7F). During a frame, write data is stored in the shadow buffers. If the frame is received correctly, the values of the shadow buffers are transferred into the effective registers. Otherwise, the values of the shadow buffers are not taken over. As a consequence of this behavior, registers take their new value shortly after the FCS of an EtherCAT frame is received. SyncManagers also change the buffers after the frame was received correctly.

User and process memory do not have shadow buffers. Accesses to these areas are taking effect directly. If a SyncManager is configured to user memory or process memory, write data will be placed in the memory, but the buffer will not change in case of an error.

3.5 Circulating frames

The ESCs incorporate a mechanism for prevention of circulating frames. This mechanism is very important for proper watchdog functionality.

This is an example network with a link failure between slave 1 and slave 2:

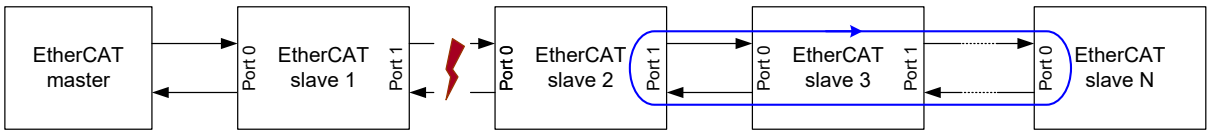


Figure 6: Circulating frames

Both slave 1 and slave 2 detect the link failure and close their ports (port 1 at slave 1 and port 0 at slave 2). A frame currently traveling through the ring at the right side of slave 2 might start circulating. If such a frame contains output data, it might trigger the built-in watchdog of the ESCs, so the watchdog never expires, although the EtherCAT master cannot update the outputs anymore.

To prevent this, a slave with loop closed at port 0 and loop control for port 0 set to auto or auto close (ESC DL control register 0x0100) will do the following inside the EtherCAT processing unit:

- If the circulating bit of the EtherCAT datagram is 0, set the Circulating bit to 1
- If the circulating bit is 1, do not process the frame and destroy it

The result is that circulating frames are detected and destroyed. Since the ESCs do not store the frames for processing, a fragment of the frame will still circulate triggering the link/activity LEDs. Nevertheless, the fragment is not processed.

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.

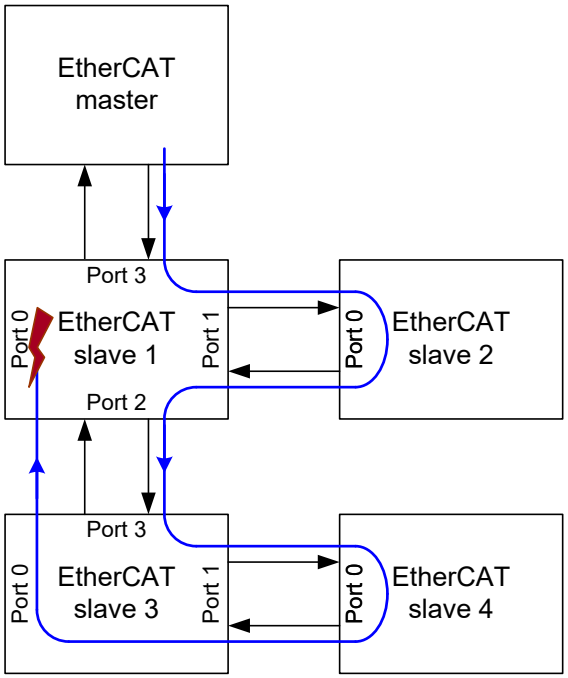


Figure 7: All frames are dropped because of circulating frame prevention

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

3.6 Non-EtherCAT protocols

If non-EtherCAT protocols are used, the forwarding rule in the ESC DL control register (0x0100[0]) has to be set to forward non-EtherCAT protocols. Otherwise they are destroyed by the ESC.

3.7 Special functions of port 0

Port 0 of each EtherCAT is characterized by some special functions in contrast to ports 1, 2, and 3:

- Port 0 leads to the master, i.e., port 0 is the *upstream* port, all other ports (1-3) are *downstream* ports (unless an error has occurred, and the network is in redundancy mode).
- The link state of port 0 influences the circulating frame bit, and frames are dropped at port 0 if the bit is set and the loop is automatically closed.
- Port 0 loop state is open if all ports are closed (either automatically or manually).
- Port 0 has a special behavior when using standard EBUS link detection.

4 Physical layer common features

EtherCAT supports two basic types of physical layers, Ethernet and EBUS. It requires 100 Mbit/s links with full duplex communication.

The Ethernet-based physical layer uses standard Ethernet physical layer devices (PHY) according to IEEE 802.3. It supports different flavors like cable (100BaseTX), optical (100BaseFX), and EtherCAT P (100BaseTX-like with power supply). EBUS is an LVDS-based physical layer, which is integrated into the EtherCAT ASICs.

The ESCs may use MII, RGMII, or RMII, for connecting to external Ethernet PHYs. The MII interface of Beckhoff ESCs is optimized e.g. for low processing/forwarding delay. The resulting additional requirements to Ethernet PHYs are described in the corresponding chapters.

4.1 Link status

The link status of each port is available in the ESC DL status register (0x0110:0x0111), most important are the “communication established” bits 15, 13, 11, and 9. Additional link information is available in the PHY port status register (0x0518:0x051B) if MI link detection and configuration is used. All other status bits are mainly for debugging purposes. The link status bits are described in the following table.

Table 11: Link status description

<table>
<tr><th rowspan="3">Status register</th><th colspan="4">MII/RMII/RGMII</th><th colspan="2" rowspan="2">EBUS</th></tr>
<tr><th colspan="2">Link signal or RGMII in-band signaling</th><th colspan="2">Management interface</th></tr>
<tr><th>Standard link detection</th><th>Enhanced link detection</th><th>Standard link detection</th><th>Enhanced link detection</th><th>Standard link detection</th><th>Enhanced link detection</th></tr>
<tr><td>ESC DL status: Physical link 0x0110[7:4]</td><td colspan="2">Link signal/in-band status</td><td colspan="2">Link signal/in-band status combined with MI link detection and configuration result</td><td colspan="2">Result of the standard link detection mechanism</td></tr>
<tr><td>ESC DL status: Communication established 0x0110[15,13,11,9]</td><td>Link signal/in-band status</td><td>Link signal/in-band status combined with RX_ERR threshold state</td><td>Link signal signal/in-band status combined with MI link detection and configuration result</td><td>Link signal signal/in-band status combined with RX_ERR threshold state and MI link detection and configuration result</td><td>Result of the standard link detection mechanism.</td><td>Result of the enhanced link detection mechanism.</td></tr>
<tr><td>PHY port status: physical link status 0x0518[0], 0x0519[0], 0x051A[0], 0x051B[0]</td><td colspan="2" rowspan="2">n.a.</td><td colspan="2">PHY has detected link (PHY status register 1[2])</td><td colspan="2" rowspan="2">n.a.</td></tr>
<tr><td>PHY port status: link status 0x0518[1], 0x0519[1], 0x051A[1], 0x051B[1]</td><td colspan="2">PHY has detected link, link is suitable for ECAT</td></tr>
</table>

If all ports are closed (either manually or automatically, e.g., because no port has a communication link), port 0 is automatically opened as the recovery port. Reading and writing via this port is possible, although the DL status register reflects the correct status. This can be used to correct erroneous DL control register settings or to fix link signal polarity configuration.

4.2 Selecting standard/enhanced link detection

Some ESCs distinguish between standard and enhanced link detection. Enhanced link detection provides additional security mechanisms regarding link establishment and surveillance. Using enhanced link detection is recommended for Ethernet PHY ports (refer to chapter 6.4.2 for compatibility issues with EBUS enhanced link detection). Some ESCs only support global enhanced link detection configuration for all ports, some support port-wise configuration.

After power-on, enhanced link detection is enabled by default. It is disabled or remains enabled after the SII EEPROM is loaded according to the EEPROM setting (register 0x0141). An invalid EEPROM content will also disable enhanced link detection.

The EEPROM setting for enhanced link detection is only taken over at the first EEPROM loading after power-on or reset. Changing the EEPROM and manually reloading it will not affect the enhanced link detection enable status (register 0x0110[2]), even if the EEPROM could not be read initially.

Registers used for enhanced link detection are listed in Table 12.

Table 12: Registers for enhanced link detection

Register address	Name	Description
0x0141[1]	ESC configuration	Enable/disable enhanced link detection for all ports
0x0141[7:4]	ESC configuration	Enable/disable enhanced link detection port-wise
0x0110[2]	ESC DL status	Enhanced link detection status

NOTE: Some of these register bits are set via SII EEPROM/IP core configuration. Some of the registers are not available in specific ESCs. Refer to section II and III for details.

4.3 FIFO size reduction

The ESCs incorporate a receive FIFO (RX FIFO) for decoupling receive clock and processing clock. The FIFO size is programmable by the EtherCAT master (ESC DL control register 0x0100). Some ESCs support a default value for the FIFO size loaded from the SII EEPROM.

The FIFO size values determine a reduction of the FIFO size, the FIFO cannot be disabled completely. The FIFO size can be reduced considering these three factors:

- Accuracy of the receiver's clock source
- Accuracy of the sender's clock source
- Maximum frame size

The default FIFO size is sufficient for maximum Ethernet frames and default Ethernet clock source accuracy (100 ppm). If the clock accuracy is 25 ppm or better, the FIFO size can be reduced to the minimum. If the FIFO size was accidentally reduced too much, a short 64 byte frame should be sent for resetting the FIFO size to the default value, since a smaller frame is not utilizing the FIFO as much as a larger frame.

The FIFO size can be reduced to minimum if both sender and receiver have 25 ppm accuracy of their clock sources, even with maximum frame size.

Since 25 ppm clock accuracy can typically not be guaranteed for the entire life-time of a clock source, the actual clock deviation has to be measured on a regular basis for FIFO size reduction. If a slave does not support distributed clocks or the actual deviation is larger than 25 ppm, the FIFO size of all neighbors and the slave itself cannot be reduced. The actual deviation can be measured using distributed clocks:

- Compare DC receive times over a period of time for slaves which only support DC receive times. Do not use this method if both slaves which are compared support DC time loop, since the measured deviation will approximate zero if the DC control loop has settled, but the actual deviation determining the FIFO size might be larger than 25 ppm.
- Compare calculated deviation from register speed counter Difference (0x0932:0x0933) for adjacent slaves with DC time loop support after the DC control loop has settled (i.e., system time difference 0x092C:0x092F is at its minimum).

NOTE: Be careful with FIFO size reduction at the first slave if off-the-shelf network interface cards without 25 ppm accuracy are used by master.

Table 13: Registers for FIFO size reduction

Register address	Name	Description
0x0100[18:16]	ESC DL control	Current FIFO size setting

NOTE: Some of these register bits are set via SII EEPROM. Some of the registers are not available in specific ESCs. Refer to section II and III for details.

4.4 Frame error detection

Refer to chapter 14 (error Counters) for details on frame error detection.

5 Ethernet physical layer

ESCs with Ethernet Physical Layer support use the MII interface, some do also support the RMII or RGMII interface. Since RMII/RGMII PHYs include FIFOs, they increase the forwarding delay of an EtherCAT slave device as well as the jitter. MII is recommended due to these reasons.

5.1 Requirements to Ethernet PHYs

EtherCAT and Beckhoff ESCs have some general requirements to Ethernet PHYs, which are typically fulfilled by state-of-the-art Ethernet PHYs.

The MII interfaces of Beckhoff ESCs are optimized for low processing/forwarding delays by **omitting a transmit FIFO**. To allow this, the Beckhoff ESCs have additional requirements to Ethernet PHYs, which are easily accomplished by several PHY vendors.

Refer to the EtherCAT slave controller application note “PHY selection guide” for Ethernet PHY requirements and example Ethernet PHYs.

Refer to section III for ESC specific information about supported features.

5.2 PHY reset and link partner notification/loop closing

The main principle of EtherCAT operation in case of link errors is disabling unreliable links by closing loops. This is automatically performed by the ESCs. The ESCs rely on a dedicated link signal from the PHYs for detecting the link state (LINK_MII, LINK_RMII, LINK_RGMII), or in case of RGMII, the in-band status signal.

It is crucial that a PHY does not establish a link while the ESC is not operating. Otherwise, the communication partner would also detect a physical link, causing it to open the communication link. Subsequently, all frames will get lost because the ESC is not operating.

So at least the following requirements have to be fulfilled, otherwise frames will be lost:

- ESC in reset state → PHY disabled

The recommended solution for this issue is to enable the PHY together with the ESC by using the ESC’s global reset signal for the PHY, too. If the ESC has individual PHY reset outputs, they should be used instead. This solution prevents the PHYs from establishing a link while the ESCs are not operating.

The individual PHY reset signals of an ESC have this functionality:

- Keep the PHY in reset while the ESC is in Reset
- If an FX link is used, and enhanced link detection is active, a reset for the PHY and the transceiver is applied when too many RX_ERR are received
- The PHY reset is extended to a certain time (refer to section III for details)

5.3 MII interface

Refer to section III for ESC specific MII information. Some MII signals are unused by Beckhoff ESCs:

Table 14: Special/unused MII interface signals

Signal	Direction at PHY	Description
TX_CLK	OUT	ESC20: TX_CLK of one PHY is used as clock source, TX_CLK of other PHY is unused, leave open. ESC dependent: TX_CLK is optionally used for automatic TX shift compensation. Other Beckhoff ESCs: Unused, leave unconnected.
COL	OUT	Collision detected. ESC20: Connected, but not used. Other Beckhoff ESCs: Unused. Leave unconnected.
CRS	OUT	Carrier sense. ESC20: Connected, but not used. Other Beckhoff ESCs: Unused. Leave unconnected
TX_ER	IN	Transmit error. ESC20: Connected, always driven low. Other Beckhoff ESCs: Connect to GND.

For more details about the MII interface, refer to IEEE Standard 802.3 (Clause 22), available from the IEEE.

5.3.1 Unused MII port

If an ESC MII interface is not used, the link signal has to be tied to a logic value which indicates no link to the ESC, and RX_CLK, RXD, RX_ER, and especially RX_DV have to be tied to GND. The TX outputs can be left unconnected, unless they are used for ESC configuration.

5.4 RMII interface

Refer to section III for ESC specific RMII information. For more details about the RMII interface, refer to the RMII specification, available from the RMII consortium.

5.4.1 Unused RMII ports

If an ESC RMII interface is not used, the link signal has to be tied to a logic value which indicates no link to the ESC, and RXD, RX_ER, and especially CRS_DV have to be tied to GND. The TX signals can be left unconnected, unless they are used for ESC configuration.

5.5 RGMII interface

Refer to section III for ESC specific RGMII information.

For more details about the RGMII interface, refer to the “Reduced Gigabit Media Independent Interface (RGMII)” specification version 2.0.

5.5.1 RGMII in-band link status

Some ESCs support RGMII PHYs. With these PHYs, it is possible (and recommended) to use the RGMII in-band status instead of the link signal. In this case, the link input of the ESC has to be tied to a value which indicates “no link”. The RGMII in-band status is checked to indicate a 100 Mbit/s full duplex link. The in-band status is equivalent to the link signal.

NOTE: Some PHYs require that RGMII in-band status is enabled by writing to the PHY management registers.

5.5.2 Unused RGMII port

If an ESC RGMII interface is not used, the link signal has to be tied to a logic value which indicates no link to the ESC, and RX_CLK, RX_CTL, and RXD have to be tied to GND. The TX signals can be left unconnected, unless they are used for ESC configuration.

5.6 Link detection

The ESCs support different mechanisms for link detection, and some of them are optional. The major goal of the link detection is to detect a link loss very fast. This is required to maintain communication in the rest of the network by appropriately closing communication loops.

The following link status sources are available for Ethernet PHYs:

- link signal from an Ethernet PHY
- RGMII in-band signaling from an Ethernet PHY with RGMII
- MI link detection and configuration (MII management interface)
- RX_ERR signal used by enhanced link detection

All ESCs support a link signal for fast link detection at each Ethernet port. Some ESCs additionally support link detection and configuration via the MII management interface, or RGMII in-band signaling.

The MI link detection is generally not fast enough for EtherCAT when a link loss happens. For many PHYs, the link signal or RGMII in-band signaling are fast enough, but others need enhanced link detection (RX_ERR evaluation).

The results of the different link status sources are evaluated inside the ESC according to the DL control loop setting (register 0x0100):

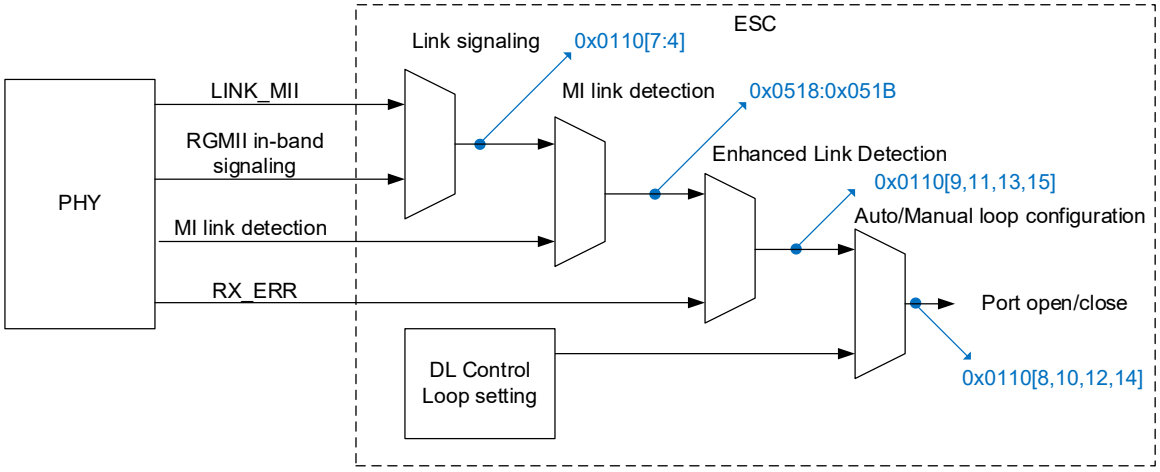


Figure 8: Link detection

At first, the link signal and the RGMII in-band status is combined. The result is available in the ESC DL status register 0x0110[7:4]. The link signal takes precedence over RGMII in-band signaling, if the link signal indicates a link. RGMII in-band signaling can be used without a link signal, the link signal is set to “no link”.

The result of the MI link detection is available in the PHY port status register 0x0518:0x051B, it is combined with the previous link result.

Enhanced link detection overrides the previous result, if it is enabled and too many RX_ERR have been detected. The result is reflected in the ESC DL status register (0x0110[15,13,11,9] – Communication established), and it is also used for LINKACT LED.

Finally, the DL control settings select if the link result is evaluated (and how). The loop state of each port is reflected in the ESC DL status register (0x0110[14,12,10,8] – loop open/closed).

Table 15: Registers used for Ethernet link detection

Register address	Name	Description
0x0100:0x0101	ESC DL control	Link configuration (manual/auto)
0x0110:0x0111	ESC DL status	Link status (link signal or RGMII in-band status)
0x0300:0x0307	RX error counter	RX_ERR counter for enhanced link detection
0x0518:0x051B	PHY port status	MI link detection results

5.6.1 Standard link detection

The results of the different sources for standard MII link detection are combined by a specific logic. The main combinations are shown in the following table.

Table 16: Ethernet link detection combination

Link signal	In-band signaling	MI link	MI no link	Link result	Description
no	no	no	no	no link	
yes	no	no	no	link	
no	yes	no	no	link	In-band signaling is used without link signal
yes	yes	no	no	link	Link signal is used while In-band signaling is available
yes (one or both)		yes	no	link	MI link detection found valid link, status of link signal/in-band signaling is accepted
don't care		no	yes	no link	MI link detection found invalid link status, either local or at link partner, overrides other status
link down event		yes	no	no link	A link down event (edge) overrides MI link detection
no	no	yes	no	link	Test only, because link-loss reaction time is too slow: MI link is used without link signal, and without in-band signaling

NOTE: "MI link" means that MI link detection successfully read the PHY, and the result is OK. "MI no link" means that MI link detection successfully read the PHY, but the result is not OK. If MI link detection could not properly read from the PHY, the result is undefined.

5.6.1.1 Link signal

The link signal used for link detection is typically an LED output signal of the Ethernet PHY. If available, the link signal should be connected to a combined signal indicating a 100 Mbit/s full duplex link. If such a signal is not available, a signal indicating a 100 Mbit/s link (speed LED) might be used. If only a general link signal is available (link LED), this might be used. Never use (combined) activity signals, e.g., link/activity LED outputs, because the link state will toggle upon activity.

The main advantage of using a dedicated link signal instead of reading out MII management interface registers is the fast reaction time in case of a link loss. This is crucial for redundancy operation, since only one lost frame is tolerated. The EtherCAT port of an ESC which loses a link has to be closed as fast as possible to maintain EtherCAT communication at the other ports and to reduce the number of lost frames.

5.6.1.2 RGMII in-band status

RGMII supports signaling of the link status (10/100/1000 Mbit/s, link up/down, full/half duplex) by using the receive data bus during inter-frame gaps. This is an optional feature of RGMII PHYs.

The EtherCAT IP core evaluates the in-band signaling. When the value is stable for a few clock cycles, it is considered valid. A valid link on the link signal takes precedence over the RGMII in-band signaling, so you should set LINK_RGMII to "no link" if you want to use RGMII in-band signaling only.

5.6.1.3 MI link detection and configuration

The EtherCAT IP core supports link detection and PHY configuration by using the MII management interface. Initially, the PHY configuration is checked and updated if necessary. Afterwards, the link status of each Ethernet port is cyclically polled. PHY accesses of the EtherCAT master are inferred upon request.

The MI link configuration mechanism configures the Ethernet PHYs to use auto-negotiation and advertise only 100BASE-TX full duplex connections. Some ESCs support using Gigabit Ethernet PHYs, which are restricted to 100 Mbit/s links by the MI link configuration. ESCs which support FX operation natively are configuring the FX ports to use fixed 100BASE-TX full duplex connections without auto-negotiation.

Depending on the physical layer configuration and the supported ESC features (e.g., TX vs. FX), the MI link detection will check if the link characteristics fulfill EtherCAT requirements (auto-negotiation, speed, duplex, etc.). If all conditions are met, an MI link is detected.

Since the MI link detection does not solely rely on the PHY link status bit (register 1[2]), the local PHY and the remote PHY may indicate a link, but the ESC refuses it because it does not fulfill EtherCAT requirements. The current MI link detection state is reflected in the MI interface registers (PHY port status 0x0518:0x051B).

MI link detection and configuration must not be used without link detection via link signal or RGMII in-band signaling, because link loss reaction time would otherwise be to slow for redundancy operation. Enhanced link detection might not be a suitable solution in this case if too few RX_ERR are issued to the ESC before the PHY takes down the link.

NOTE: For testing it is possible to use MI link detection without link signals: If the link signals are statically set to “no link”, only MI link detection is used.

The MI link detection and configuration checks the management communication with Ethernet PHYs. If communication is not possible – e.g. because no PHY is configured for the expected PHY address – the results are ignored. Take care of proper PHY address configuration to prevent erroneous behavior.

NOTE: Proper PHY address settings and PHY address offset configuration is crucial for MI link detection and configuration.

5.6.2 Enhanced link detection

For Ethernet, the enhanced MII link detection feature is a feature of link error detection and reaction. This has to be distinguished from the actual link detection, which tells the ESC if a physical link is available (i.e., the link signal or the MI link detection and configuration mechanism).

Enhanced MII link detection will additionally disconnect a link if at least 32 RX errors (RX_ER) occur in a fixed interval of time (~10 µs). The local loop is closed and the link partner is informed by restarting the auto-Negotiation mechanism via the MII management interface. This informs the link partner of the error condition, and the link partner will close the loop.

The ESC keeps the port closed until the link goes down during auto-negotiation and comes up again (the port remains closed if the link does not go down).

The availability of enhanced MII link detection depends on a supported PHY address configuration, otherwise it has to be disabled.

5.7 MII management interface (MI)

Most EtherCAT slave controllers with MII/RMII/RGMII ports use the MII management interface for communication with the Ethernet PHYs. The MII management interface can be used by the EtherCAT master – or the local µController if supported by the ESC. Enhanced MII link detection uses the management interface for configuration and restarting auto negotiation after communication errors occurred (TX PHYs only). Some ESCs can make use of the MII management interface for link detection and PHY configuration. For fast link detection, the ESCs require to use a separate signal, or RGMII in-band status.

Refer to chapter 5.6 for details about link detection with Ethernet PHYs. For more details about the MII management interface, refer to IEEE Standard 802.3 (Clause 22), available from the IEEE.

The ESCs support a shared MII management interface for all PHYs, i.e., MCLK and MDIO are connected to the ESC and to all PHYs.

5.7.1 PHY addressing/PHY address offset

Proper PHY address configuration is crucial for enhanced link detection and MI link detection and configuration, because the ESC itself needs to relate logical ports to the corresponding PHY addresses. The EtherCAT master can access the Ethernet PHYs using any address by means of the PHY address register 0x0513.

Typically, the logical port numbers match with the PHY addresses, i.e. the EtherCAT master and the ESC itself use PHY address 0 for accessing the PHY at port 0 (PHY address 1 for the PHY at port 1 and so on).

Depending on the ESC, there are two options for configuring the PHY addresses:

- PHY address offset:
  The ESC accesses a PHY at logical port x with the address “x + PHY address offset”. Depending on the ESC only some or all possible offsets are configurable.
  The EtherCAT master uses the PHY addresses 0-3 to access the logical ports 0-3. These addresses are incremented by the PHY address offset inside the ESC. If the master uses PHY addresses 4-31, these addresses are also incremented by the PHY address offset inside the ESC.
- Individual PHY address:
  The ESC accesses a PHY at each logical port with an individual PHY address.
  The EtherCAT master uses the PHY addresses 0-3 to access the logical ports 0-3. These addresses are replaced with the individual addresses inside the ESC. If the master uses PHY addresses 4-31, these addresses are not translated.

The PHY address configuration depends on the ESC.

Typically, the PHY address offset should be 0, and the logical port numbers match with the PHY addresses. Some Ethernet PHYs associate a special function with PHY address 0, e.g., address 0 is a broadcast PHY address. In these cases, PHY address 0 cannot be used. Instead, a PHY address offset different from 0 should be selected, as supported by the ESC. It is recommended that the PHY addresses are selected to be equal to the logical port number plus 1 in this case. If port 0 is EBUS, ports 1-3 should have PHY addresses 1-3, i.e., PHY address offset is 0.

If the PHY address offset configuration of an ESC reflects the actual PHY address settings, the EtherCAT master can use addresses 0-3 in PHY address register 0x0513 for accessing the PHYs of logical ports 0-3, regardless of the PHY address offset.

Table 17: PHY address configuration matches PHY address settings

Logical port	Configured address of the PHY		PHY address register value used by EtherCAT master
	PHY address offset = 0	PHY address offset = 16	
0	0	16	0
1	1	17	1
2	2	18	2
3	3	19	3
none	4-15	20-31	4-15
none	16-31	0-15	16-31

5.7.2 Logical interface

The MI of the ESC is typically controlled by EtherCAT via the MI registers[1].

Table 18: MII management interface register overview

Register address	Description
0x0510:0x0511	MII management control/status
0x0512	PHY address
0x0513	PHY register address
0x0514:0x0515	PHY data

The MI supports two commands: write to one PHY register or read one PHY register.

5.7.2.1 MI read/write example

The following steps have to be performed for a PHY register access:

1. Check if the busy bit of the MI status register is cleared and the MI is not busy.
2. Write PHY address to PHY address register.
3. Write PHY register number to be accessed into PHY register address register (0-31).
4. Write command only: put write data into PHY data register (1 word/2 byte).
5. Issue command by writing to control register.
   For read commands, write 1 into command register Read 0x0510[8].
   For write commands, write 1 into write Enable bit 0x0510[0] and also 1 into command register write 0x0510[9]. Both bits have to be written in one frame. The write enable bit realizes a write protection mechanism. It is valid for subsequent MI commands issued in the same frame and self-clearing afterwards.
6. The command is executed after the EOF, if the EtherCAT frame had no errors.
7. Wait until the busy bit of the MI status register is cleared.
8. Check the error bits of the MI status register. The command error bit is cleared with a valid command or by clearing the command register. The read error bit indicates a read error, e.g., a wrong PHY address. It is cleared by writing to the register.
9. Read command only: Read data is available in PHY data register.

NOTE: The command register bits are self-clearing. Manually clearing the command register will also clear the status information.

5.7.2.2 MI interface Assignment to ECAT/PDI

The EtherCAT master controls the MI interface (default) if the MII management PDI access state register 0x0517[0] is not set. The EtherCAT master can prevent PDI control over the MI interface, and it can force the PDI to release the MI interface control. After power-on, the PDI can take over MI interface control without any master transactions.

[1] ET1100 only: MI Control is transferred to PDI if the transparent mode is enabled. IP Core: MI control by PDI is possible.

5.7.3 MI protocol

Each MI access begins with a preamble of “Ones“ (32 without preamble suppression, less if both ESC and PHY support preamble suppression), followed by a start-of-frame (01) and the operation code (01 for write and 10 for read operations). Then the PHY address (5 bits) and the PHY register address (5 bits) are transmitted to the PHY. After a turnaround (10 for write and Z0 for read operations – Z means MDIO is high impedance), two bytes of data follow. The transfer finishes after the second data byte and at least one idle cycle.

5.7.4 Timing specifications

Table 19: MII management interface timing characteristics

Parameter	Comment
t_{Clk}	MDC period
t_{Write}	Write access time
t_{Read}	Read access time

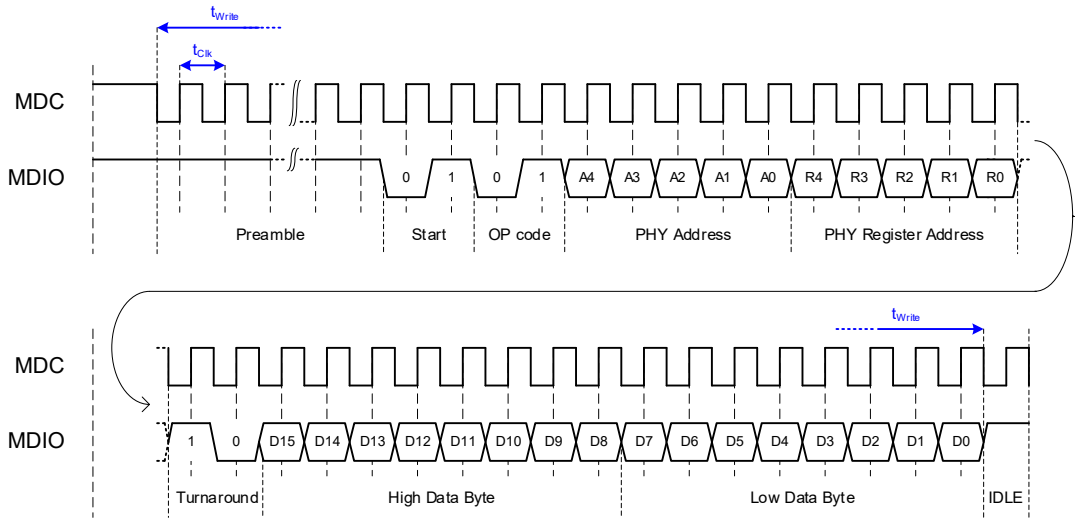


Figure 9: Write access

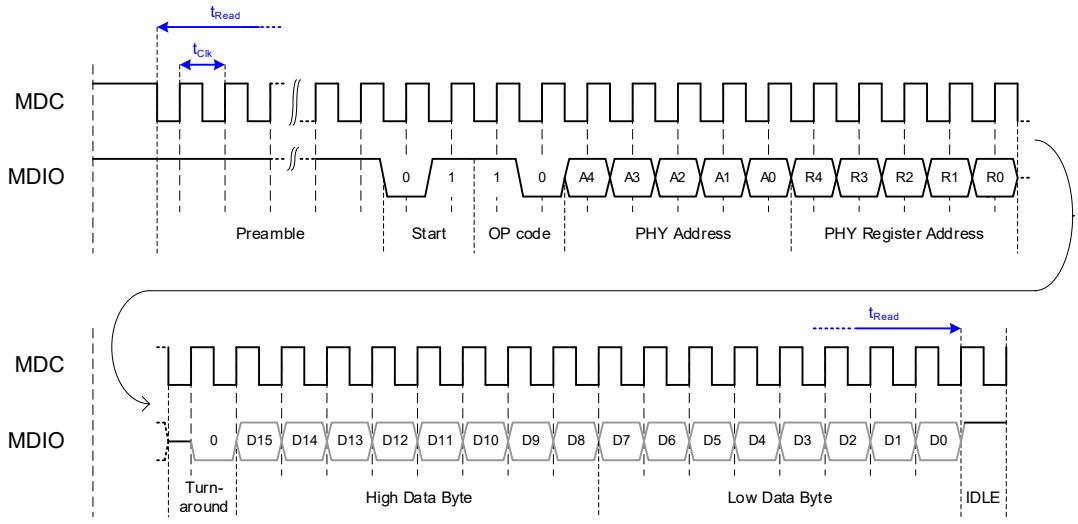


Figure 10: Read access

5.8 MII management example schematic

The MII management interface is a shared bus for all PHYs. The example schematic shows the connection between ESC and PHYs. Take care of proper PHY address configuration.

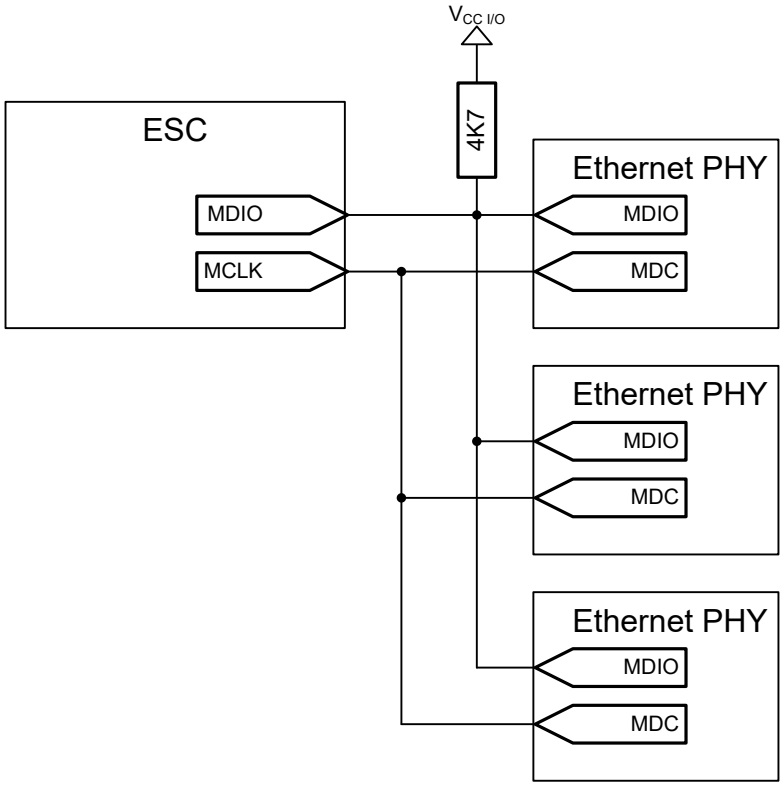


Figure 11: MII management example schematic

5.9 Back-to-back connection

5.9.1 MII: ESC to ESC connection

Two EtherCAT slave controllers can be connected back-to-back using MII as shown in the figure below. The timing of RX_DV and RXD with respect to RX_CLK has to be checked at both ESCs to be compliant with the IEEE 802.3 requirements of min. 10 ns setup time and min. 10 ns hold time. The timing can be adjusted by configuring the TX shift settings of each ESC, or using automatic TX shift.

Enhanced link detection must not be enabled for the back-to-back ports on both ESCs, and MI link detection and configuration must be disabled as well.

Other clocking options besides using quartz oscillators are also possible. If CLK25_OUT is not available, REF_CLK (the clock input of the ESC) can be used instead.

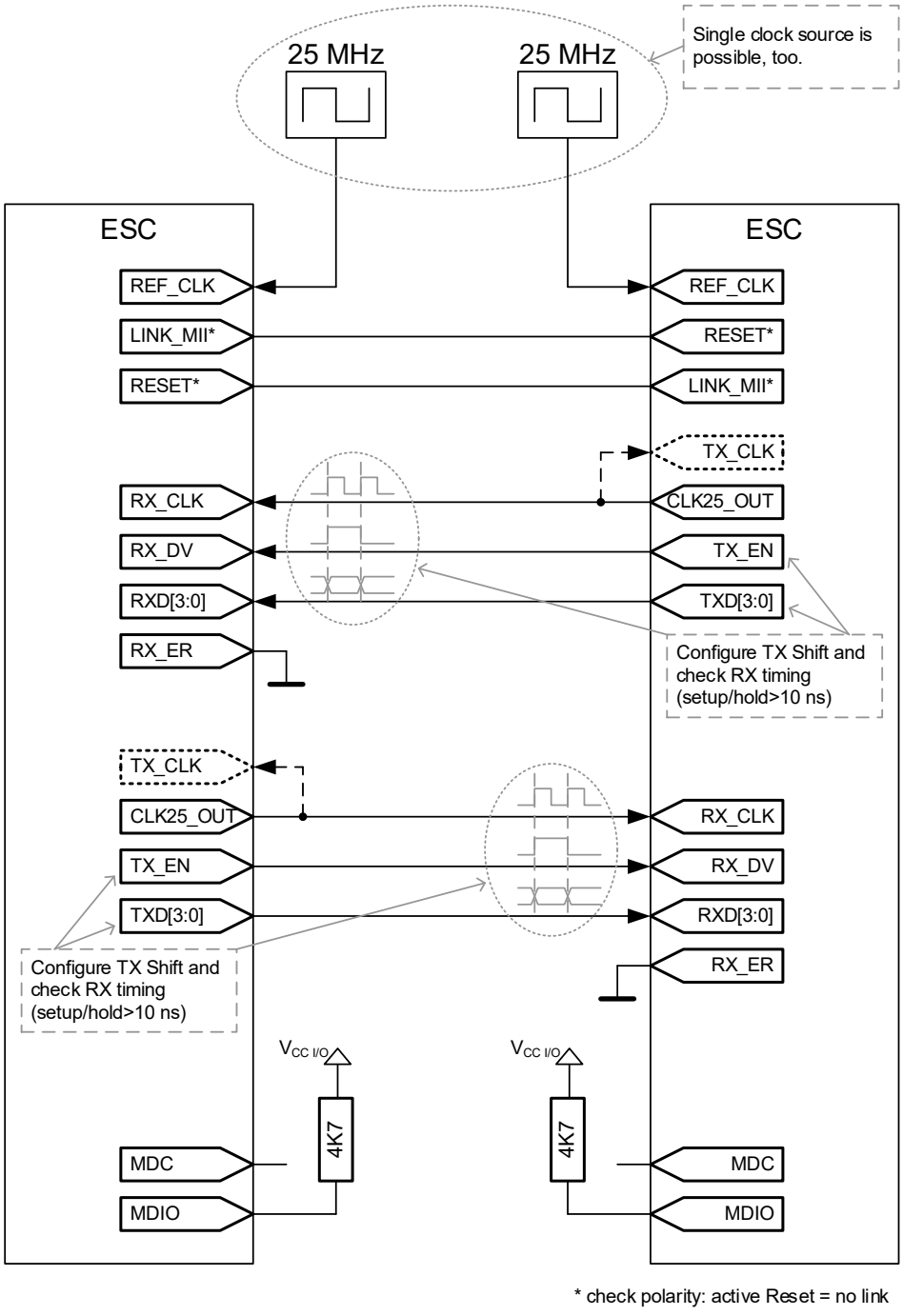


Figure 12: Back-to-back MII connection (two ESCs)

5.9.2 MII: ESC to standard Ethernet MAC

If an ESC is to be connected directly to a standard Ethernet MAC (e.g. µController or FPGA), RX timing at the ESC and the MAC has to be checked. Since TX shift configuration is not possible in the MAC, RX_CLK for the ESC has to be adjusted (delayed) to achieve proper RX timing at the ESC.

An EtherCAT slave controller can be directly connected to a standard Ethernet MAC using MII as shown in the figure below. The timing of RX_DV and RXD with respect to RX_CLK has to be checked at both ESC and MAC to be compliant with the IEEE 802.3 requirements of min. 10 ns setup time and min. 10 ns hold time. The timing can be adjusted by configuring the TX shift setting of the ESC and the clock buffer (e.g. using one or more buffers).

If the standard Ethernet MAC completely uses the IEEE 802.3 TX signal timing window of 0 to 25 ns, standard compliant RX timing (-10..10 ns) is impossible. Since most Beckhoff ESCs do not require the entire IEEE802.3 RX timing window (check in section III), a valid configuration is possible even in this case.

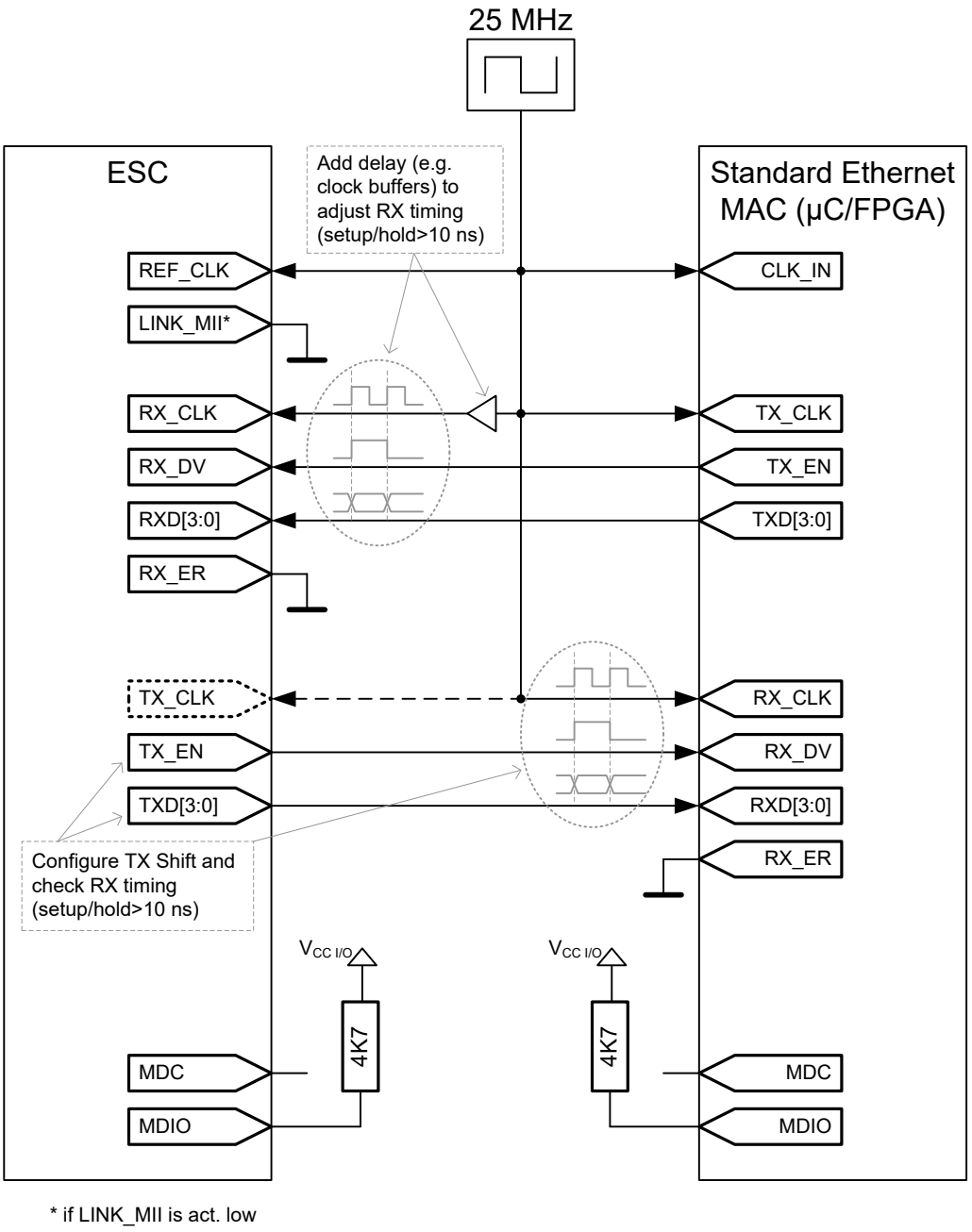


Figure 13: Back-to-back MII connection (ESC and standard MAC)

5.9.3 RGMII: ESC to ESC

In general, all RGMII TX signals are connected to their RGMII RX counterparts.

- The LINK_RGMII signal of both sides has to indicate “link up”. RGMII in-band status cannot be used, since the TX side will not transmit appropriate values.
- The timing relation between clock and data has to be observed. If the back-to-back connection is inside the FPGA, CLK25_2NS should be connected to CLK25. Then, TX_CLK and RX_CLK are also equal to CLK25, and the timing analyzer will verify that the signals are transferred synchronously.
- MI link detection and configuration cannot be enabled, it would try to configure the back-to-back port which has no PHY
- Enhanced Link Detection can be enabled, since the back-to-back port will not observe RX errors. To be sure, FX mode can be selected for the back-to-back ports, and the link signal input would be connected to the PHY reset output.

5.10 EtherCAT 100BaseTX physical layer

5.10.1 Ethernet connector (RJ45 / M12)

Fast Ethernet (100BaseTX) uses two pairs/four pins. An RJ45 connector (8P8C) or an M12 D-code connector can be used. The RJ45 connector is recommended to use MDI pinout (PC side) for all ports for uniformity reasons (standard EtherCAT; Fast Hot Connect ports use MDI at port 0, and MDI-X at ports 1-3). Standard Ethernet cables are used, not crossover cables. PHYs have to support MDI/MDI-X auto-crossover.

Table 20: Signals used for Fast Ethernet

Signal	Name	Core color	Contact assignment	
	MDI	(may change depending on cable)	RJ45	M12 D-code
TX+	Transmission data +	Yellow (light orange)	1	1
TX-	Transmission data -	Orange	2	3
RX+	Receive data +	Light green	3	2
RX-	Receive data -	Green	6	4

NOTE: MDI-X (Switches/Hubs) uses same outline as MDI but TX+ is RX+ and TX- is RX- and vice versa.

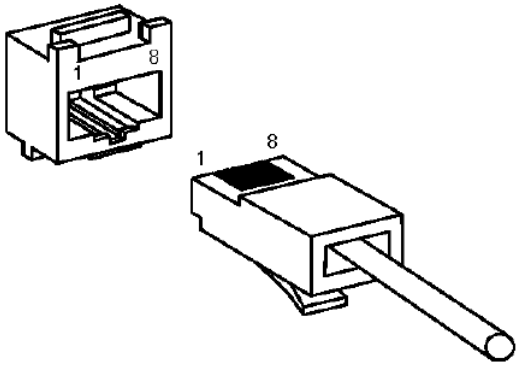


Figure 14: RJ45 connector

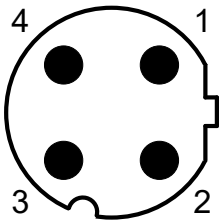


Figure 15: M12 D-code connector

5.10.2 Ethernet termination and grounding recommendation

This termination and grounding design recommendation may help to meet the overall requirements for industrial communication. Nevertheless, implementation may vary depending on other requirements like board layout, other capacities, common ground interconnection, and shield grounding.

Unused RJ-45 pins are terminated by 75Ω resistors which will be connected to virtual ground. Virtual GND is connected to Protection Earth (PE) by a 10nF/500V capacitor in parallel to a 1MΩ resistor. Shield is also connected to PE by a 10nF/500V capacitor in parallel to a 1MΩ resistor. Especially the values for the connection between virtual GND and PE are subject to change to meet industrial requirements (EMC). Shield is not directly connected with PE to avoid ground loops across large industrial plants with different PE potentials.

This design recommendation of termination and grounding is shown in Figure 16.

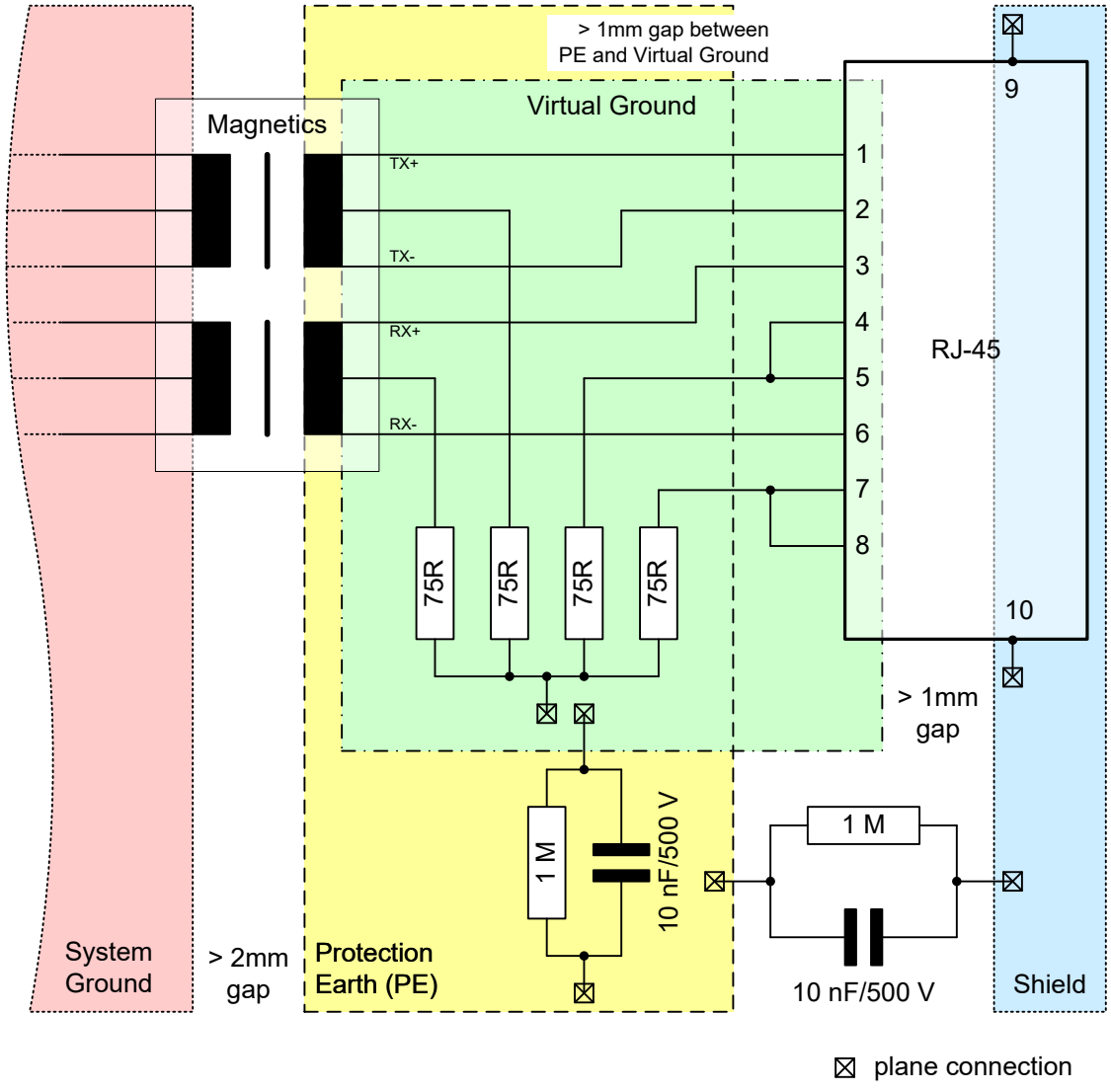


Figure 16: Termination and grounding recommendation

5.11 EtherCAT 100BaseFX physical layer

EtherCAT communication over optical links using Ethernet PHYs is possible, but some requirements of EtherCAT have to be respected, and some characteristics of EtherCAT slave controllers have to be considered.

Some ESCs are prepared for FX operation, others require external logic to achieve EtherCAT compatibility.

Refer to the EtherCAT slave controller application note “PHY selection guide” for details on FX PHY connection and example schematics. The application note is available at the Beckhoff website (http://www.beckhoff.com).

5.11.1 Link partner notification and loop closing

The main principle of EtherCAT operation in case of link errors is disabling unreliable links by closing loops. This is automatically performed by the ESCs. The ESCs rely on the link signal/in-band status from the PHYs for detecting the link state.

With FX connections, it could happen that the transceiver device is powered, while the PHY (and/or the ESC) is not active. The communication partner would detect a signal, causing him to open the link. All frames will get lost because the PHY (and/or the ESC) is not operating.

So at least the following two requirements have to be fulfilled, otherwise frames will be lost:

- ESC in reset state → transceiver disabled
- PHY in reset state → transceiver disabled

The recommended solution for this issue is to enable the transceiver together with the PHY by using the PHY’s reset signal for the transceiver, too. If the transceiver has no suitable input, the power supply of the transceiver can be switched off. Since the PHY’s reset should be controlled by the ESC’s reset output (either main reset output or individual PHY reset output), the transceiver will power down while the PHY is in the reset state and also while the ESC is in the reset state. Thus, the ESC and the PHY will be active when the transceiver gets active, and no frames are lost.

5.11.2 Far-end-fault (FEF)

Some FX PHYs offer a Far-end-fault (FEF) generation/detection feature. The intention is to inform the link partner of a bad link.

FEF generation
If an FEF-supporting PHY receives a signal with a quality which is not sufficient, the PHY will transmit a special FEF pattern to the link partner.

FEF detection
If an FEF-supporting PHY receives the FEF pattern with good signal quality, it will continue transmitting regularly, but it will indicate “no link” locally to the ESC, until the FEF pattern ends.

Conclusion
The FEF feature is advantageous for EtherCAT, because the PHYs will only indicate a link when the signal quality is high enough. Without FEF, the EtherCAT slave controllers have to rely on the enhanced link detection feature for detecting a low quality link.
Nevertheless, enhanced link detection becomes active only after the link is already established, thus, in case of a low quality link, the link status will be toggling on/off (link up → enhanced link detection tears down link → link up …). This is sufficient to locate an issue in the network, but it might disturb operation of the remaining network.

So, it is highly recommended to use PHYs which fully implement FEF generation and detection.

NOTE: Some PHYs are claiming FEF support, but they are either not supporting FEF generation or detection, or they require configuration commands via MI management interface, which cannot be issued by the ESCs automatically.

5.11.3 ESCs with native FX support

ESCs with native FX support have individual PHY reset outputs for each port. This PHY reset output is intended to hold the PHY and the transceiver in reset state while the ESC is in reset state, and additionally, to issue a reset cycle when a link failure is detected by the enhanced link detection mechanism.

If at least one port is configured for FX operation, all ports have to use the individual PHY reset outputs. This is especially important for enhanced link detection, since all the PHY reset outputs are used for link down signaling instead of auto-negotiation restart, which is not used anymore – regardless of the port using FX or TX.

5.11.4 ESCs without native FX support

ESCs without native FX support require special attention regarding reset connection, link down signaling and enhanced link detection. External logic might be required.

5.12 EtherCAT P physical layer

EtherCAT P specifies an alternative Ethernet-based physical layer for EtherCAT, which combines EtherCAT communication and power in a single 4-wire standard Ethernet cable. It uses standard Ethernet PHYs, and standard EtherCAT slave controllers, while adding power supply to the connectors and cables.

Please refer to the EtherCAT technology group website (http://www.ethercat.org/EtherCATP) for more details on EtherCAT P.

5.13 Gigabit Ethernet PHYs

Gigabit Ethernet PHYs can generally be used for EtherCAT, as long as the link speed is restricted to 100 Mbit/s, either by strapping options of the PHY or by using the auto-negotiation advertisement.

Some ESCs are capable of restricting the auto-negotiation advertisement of Gigabit Ethernet PHYs to 100 Mbit/s full-duplex if MI link detection and configuration is enabled.

Nevertheless, all other requirements of EtherCAT have to be fulfilled – especially the link loss reaction time (enhanced link detection might be required).

6 EBUS/LVDS physical layer

EBUS is an EtherCAT physical layer designed to reduce components and costs. It also reduces delay inside the ESC. The EBUS physical layer uses low voltage differential signaling according to the ANSI/TIA/EIA-644 “Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits” standard.

EBUS has a data rate of 100 Mbit/s to accomplish the Fast Ethernet data rate. The EBUS protocol simply encapsulates Ethernet frames, thus EBUS can carry any Ethernet frame – not only EtherCAT frames.

EBUS is intended to be used as a backplane bus, it is not qualified for wire connections.

6.1 Interface

Two LVDS signal pairs per EBUS link are used, one for reception and one for transmission of Ethernet/EtherCAT frames.

The EBUS interface has the following signals:

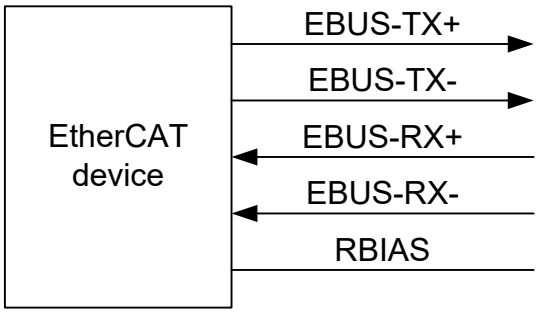


Figure 17: EBUS interface signals

Table 21: EBUS interface signals

Signal	Direction	Description
EBUS-TX+ EBUS-TX-	OUT	EBUS/LVDS transmit signals
EBUS-RX+ EBUS-RX-	IN	EBUS/LVDS receive signals
RBIAS		BIAS resistor for EBUS-TX current adjustment

Unused EBUS ports can be left unconnected. If required by the ESC, the LVDS termination resistor and the RBIAS resistor remain mandatory.

6.2 EBUS protocol

Ethernet/EtherCAT frames are Manchester encoded (Biphase L) and encapsulated in start-of-frame (SOF) and end-of-frame (EOF) identifiers. A beginning of a frame is detected if a Manchester violation with positive level (N+) followed by a ‘1’ bit occurs. The falling edge in the middle of the ‘1’ indicates the SOF to the ESC. This ‘1’ bit is also the first bit of the Ethernet preamble. Then the whole Ethernet frame is transmitted (including Ethernet SOF at the end of the preamble, up to the CRC). The frame finishes with a Manchester violation with negative level (N-), followed by a ‘0’ bit. This ‘0’ bit is also the first bit of the IDLE phase, which consists of ‘0’ bits.

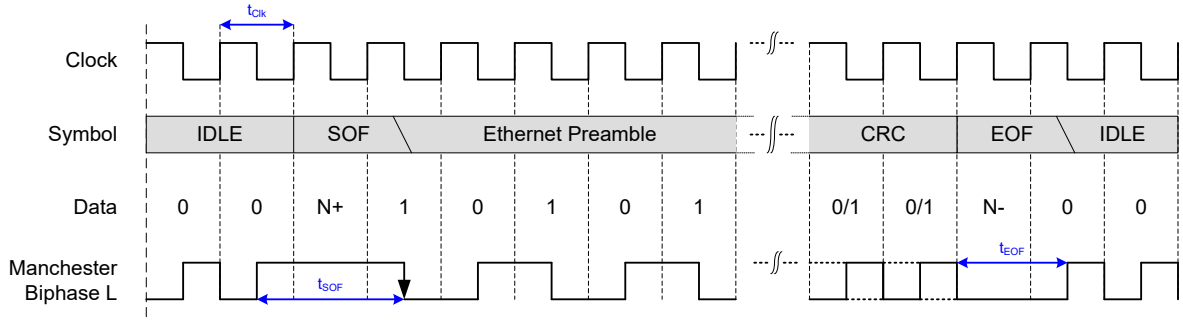


Figure 18: EBUS protocol

6.3 Timing characteristics

Table 22: EBUS timing characteristics

Parameter	Min	Typ	Max	Comment
t_{Clk}		10 ns		EBUS clock (100 Mbit/s)
t_{SOF}	15 ns			Positive level of SOF before falling edge
t_{EOF}	15 ns			Negative level of EOF after last edge

NOTE: After power-on, a receiver which receives IDLE symbols cannot distinguish incoming ‘0’ bits from ‘1’ bits, because it is not synchronized to the transmitters phase. Synchronization is established at the falling edge at the end of the EBUS SOF, which indicates the center of the first preamble bit. After synchronization, idle errors can be detected by the ESC.

NOTE: SOF is detected at a falling edge following a period of at least 15 ns (nominal) of positive level, EOF is detected after a period of at least 15 ns (nominal) of negative level. I.e., the length of SOF and EOF can be even longer.

6.4 EBUS link detection

The ESCs support different mechanisms for link detection, and some of them are optional. The major goal of the link detection is to detect a link loss very fast. This is required to maintain communication in the rest of the network by appropriately closing communication loops.

The following link status sources are available for EBUS:

- EBUS link detection
- EBUS enhanced link detection

The EBUS link detection is fast enough for EtherCAT when a link loss happens. The enhanced link detection provides an additional level of security.

The results of the different link status sources are evaluated inside the ESC according to the DL control loop setting (register 0x0100):

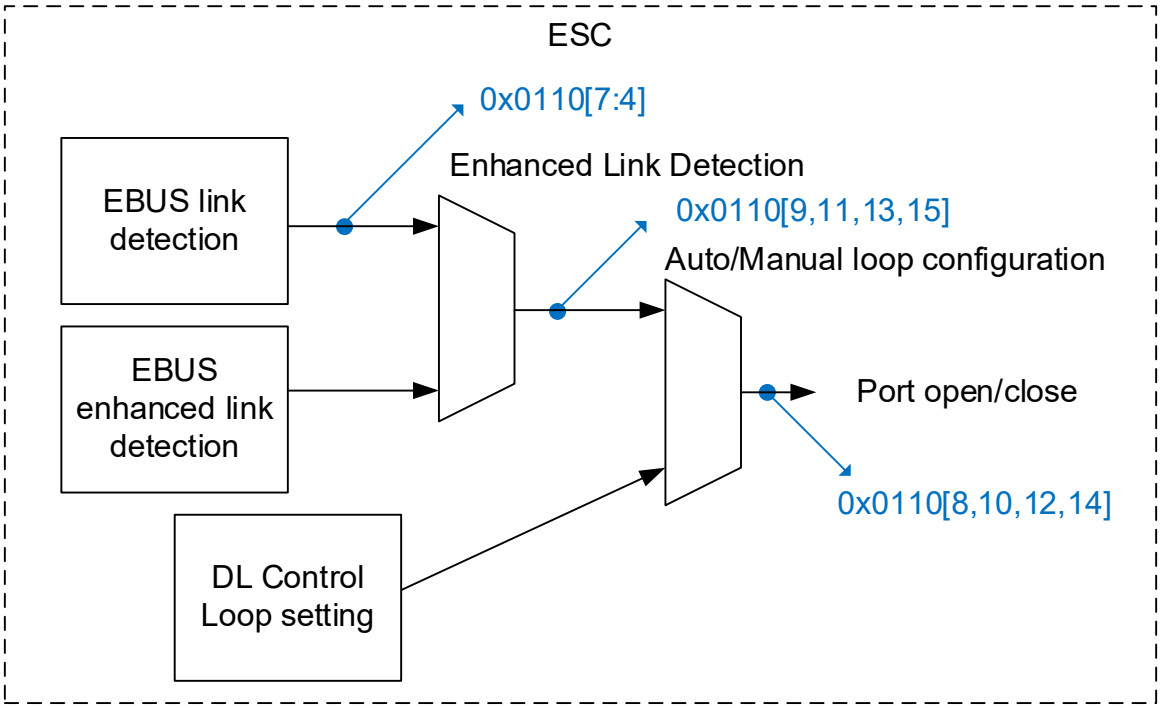


Figure 19: EBUS link detection

Enhanced link detection overrides the result of the standard EBUS link detection, if it is enabled. The result is reflected in the ESC DL status register (0x0110[15,13,11,9] – Communication established), and it is also used for LINKACT LED.

Finally, the DL control settings select if the link result is evaluated (and how). The loop state of each port is reflected in the ESC DL status register (0x0110[14,12,10,8] – loop open/closed).

Table 23: Registers used for Ethernet link detection

Register address	Name	Description
0x0100:0x0111	ESC DL control	Link configuration (manual/auto)
0x0110:0x0111	ESC DL status	Link status (link signal or RGMII in-band signaling)
0x0300:0x0307	RX error counter	RX_ERR counter for enhanced link detection

6.4.1 Standard EBUS link detection

Standard EBUS link detection is realized by counting the number of correctly detected data bits (without RX error) in a defined interval of time and comparing it to a given value. The link is established if the nominal number of correct bits was detected, using a small tolerance. The link is disconnected if too few bits were counted.

In order to handle partial link failures correctly, the following mechanism is used:

- An ESC transmits at port 0 only if a link is detected (e.g., IDLE symbols are received), otherwise it will transmit N- symbols.
- An ESC transmits at ports 1, 2, and 3 regardless of the link state (typically IDLE symbols if no frames are pending).

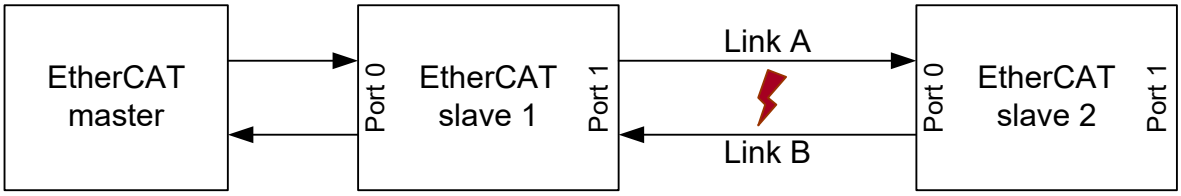


Figure 20: Example EtherCAT network

This method addresses these two cases of partial link failure (see Figure 20):

- A failure on link A will be detected by slave 2, which will stop transmitting anything on link B (and close the loop at port 0). This is detected by slave 1, which will close the loop at port 1. The master can still communicate with slave 1.
- A failure on link B will be detected by slave 1, which will close the loop at port 1. The master can still communicate with slave 1. This failure cannot be detected by slave 2, which will leave port 0 open.

Do not connect any EBUS port 0 to another EBUS port 0 (same ESC or different ESCs) using standard link detection, because standard link detection will not establish a link after it was down.

Do not connect any EBUS port 1-3 to another EBUS port 1-3 (same ESC or different ESCs) using standard link detection, because partial link failures will not result in closed ports.

NOTE: Standard link detection cannot cope with a specific partial link fault (link B failure), which affects redundancy operation (e.g., port 1 of slave 2 is connected to the master), because the master cannot communicate with slave 2 which leaves its port 0 open.

NOTE: Another advantage of this mechanism is that in case slave 2 is added to the network, at first port 0 of slave 2 is opened because there is activity on link A, then transmission on link B is started, and finally slave 1 opens port 1. This assures that no frames get lost during link establishment.

6.4.2 Enhanced EBUS link detection

Enhanced EBUS link detection uses the standard link detection mechanism and adds a simple link detection handshake protocol before the link is established.

With enhanced link detection, the ESC transmits on all ports regardless of the link state (typically IDLE symbols are transmitted if no frames are pending).

The handshake protocol consists of three phases:

1. Each device starts transmitting a 4 nibble link request frame with 0x55C4 regularly at each port. This frame has no SOF/CRC and would be discarded if it was accidentally received by standard Ethernet devices (e.g., masters).
2. If a link request frame (0x55C4) is received at a port, the ESC will transmit link acknowledge frames (0x55CC) instead of link request frames at this port.
3. If a link acknowledge frame (0x55CC) is received at a port, the link at the port is established. Both link partners know that the link is good and establish the link. No further link detection frames are transmitted (until the link is interrupted and the link detection handshake phases are starting from the beginning).

Link disconnection is signaled to the link partner by stopping transmission for a certain time. This will be detected by the standard EBUS link detection mechanism. The link gets disconnected at both sides, and both sides close their loops. After that, the first phase of the handshake protocol starts again.

EBUS enhanced link detection is not compatible with older devices which forward enhanced link detection handshake frames depending on the direction (e.g. ESC20 and bus terminals without ASICs): the handshake frames are not forwarded through the EtherCAT processing unit, but they are forwarded without modification alongside the EtherCAT processing unit.

A device using enhanced link detection will stop generating handshake frames after the link is established or the enhanced link detection is disabled by SII EEPROM setting. It will restart generating handshake frames shortly after a link is lost (unless enhanced link detection is disabled).

6.5 EBUS RX errors

The EBUS receiver detects the following RX errors:

- Missing edge in the middle of one bit (but not EBUS SOF/EOF)
- EBUS SOF inside a frame (two SOFs without EOF in between)
- EBUS EOF outside a frame (two EOFs without SOF in between)
- IDLE violation: ‘1’ outside a frame
- Too short pulses (less than ~3.5 ns)

A frame with an RX error is discarded. Too many RX errors in a defined interval of time will result in link disconnection.

Errors outside of a frame are counted only once, and errors inside a frame are also counted only once, because the Manchester decoding might have lost synchronization.

6.6 EBUS low jitter

In low jitter mode, the jitter of the forwarding delay (EBUS port to EBUS port) is reduced.

6.7 EBUS connection

The connection of two EBUS ports is shown in Figure 21. LVDS termination with 100 Ohm impedance is necessary between each pair of receive signals – located adjacent to the receive inputs, or integrated into the ESC.

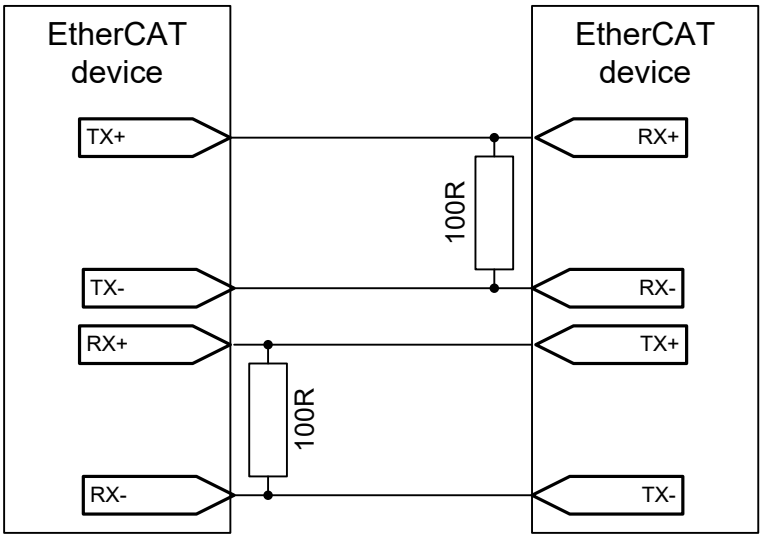


Figure 21: EBUS connection

7 FMMU

Fieldbus memory management units (FMMU) convert logical addresses into physical addresses by the means of internal address mapping. Thus, FMMUs allow to use logical addressing for data segments that span several slave devices: one datagram addresses data within several arbitrarily distributed ESCs. Each FMMU channel maps one continuous logical address space to one continuous physical address space of the slave. The FMMUs of Beckhoff ESCs support bit wise mapping, the number of supported FMMUs depends on the ESC. The access type supported by an FMMU is configurable to be either read, write, or read/write.

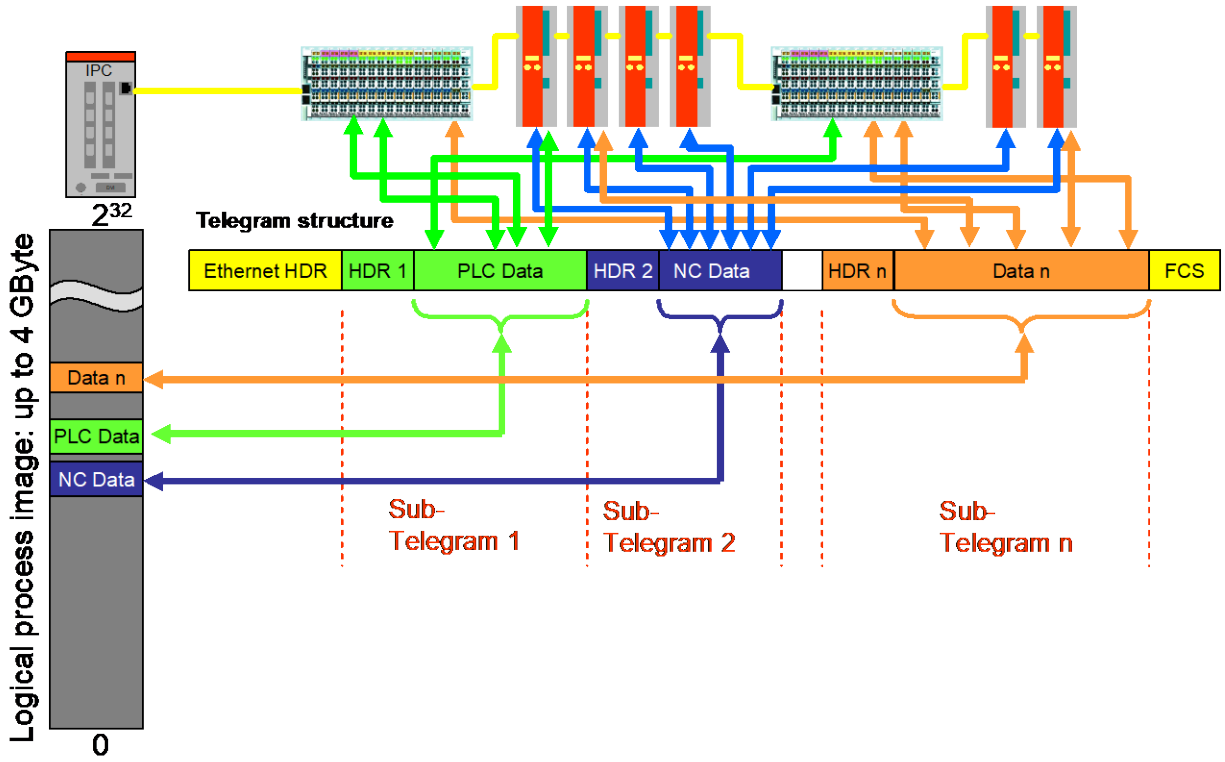


Figure 22: FMMU mapping principle

The following example illustrates the functions of an FMMU configured to map 14 bits from logical address 0x00010011.3 to 0x00010013[0] to the physical register bits 0x0F01[1] to 0x0F02[6]. The FMMU length is 3 byte, since the mapped bits span 3 Bytes of the logical address space. Length calculation begins with the first logical byte which contains mapped bits, and ends with the last logical byte which contains mapped bits.

Table 24: Example FMMU configuration

FMMU configuration register	FMMU reg. offset	Value
Logical start address	0x0:0x3	0x00010011
Length (bytes)	0x4:0x5	3
Logical start bit	0x6	3
Logical stop bit	0x7	0
Physical start address	0x8:0x9	0x0F01
Physical start bit	0xA	1
Type	0xB	read and/or write
Activate	0xC	1 (enabled)

NOTE: FMMU configuration registers start at address 0x0600.

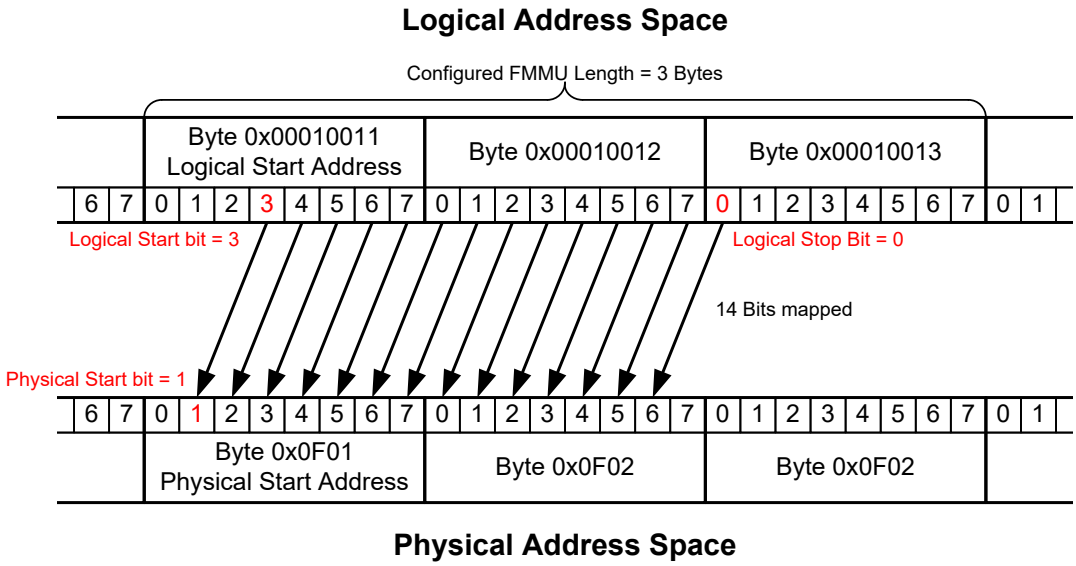


Figure 23: FMMU mapping example

Attention: This drawing of the bit string shows the least significant bit first, in a hexadecimal representation of the octets the least significant value is at the right place and the most significant on the left place (00110011 is represented as octet by 0xCC).

Restrictions on FMMU settings

The FMMUs of Beckhoff ESCs are subject to restrictions, since adjacent FMMU matches lead to internal read/write accesses, which cannot overlap.

The logical address ranges of two FMMUs of the same direction (read or write) in one ESC must be separated by at least 3 logical bytes not configured by any FMMU of the same type, if one of the FMMUs or both use bit-wise mapping (logical start bit ≠ 0, logical stop bit ≠ 7, or physical start bit ≠ 0). In the above example, the first logical address area after the one shown must have a logical start address of 0x00010017 or higher (the last byte of the example FMMU is 0x00010013, three bytes free 0x00010014-0x00010016).

If only byte-wise mapping is used (logical start bit = 0, logical stop bit = 7, or physical start bit = 0, the logical address ranges can be adjacent.

Bit-wise writing is only supported by the digital output register (0x0F00:0x0F03). All other registers and memories are always written byte-wise. If bit-wise mapping is used for writing into these areas, bits without mapping to logical addresses are written with undefined values (e.g., if only physical address bit 0x1000[0] is mapped by a write FMMU, the bits 1-7 are written with undefined values).

Additional FMMU characteristics

- Each logical address byte can at most be mapped either by one FMMU (read) plus one FMMU(write), or by one FMMU(read/write). If two or more FMMUs (with the same direction – read or write) are configured for the same logical byte, the FMMU with the lower number (lower configuration address space) is used, the other ones are ignored.
- One or more FMMUs may point to the same physical memory, all of them are used. Collisions cannot occur.
- It is the same to use one read/write FMMU or two FMMUs – one read, the other one write – for the same logical address.
- A read/write FMMU cannot be used together with SyncManagers, since independent read and write SyncManagers cannot be configured to use the same (or overlapping) physical address range.
- Bit-wise reading is supported at any address. Bits which are not mapped to logical addresses are not changed in the EtherCAT datagram. E.g., this allows for mapping bits from several ESCs into the same logical byte.
- A frame/datagram addressing a logical address space which is not configured in the ESC will not change data in the ESC, and no data from the ESC is placed in the frame/datagram.

8 SyncManager

The memory of an ESC can be used for exchanging data between the EtherCAT master and a local application (on a µController attached to the PDI) without any restrictions. Using the memory for communication like this has some drawbacks which are addressed by the SyncManagers inside the ESCs:

- Data consistency is not guaranteed. Semaphores have to be implemented in software for exchanging data in a coordinated way.
- Data security is not guaranteed. Security mechanisms have to be implemented in software.
- Both EtherCAT master and application have to poll the memory in order to find out when the access of the other side has finished.

SyncManagers enable consistent and secure data exchange between the EtherCAT master and the local application, and they generate interrupts to inform both sides of changes.

SyncManagers are configured by the EtherCAT master. The communication direction is configurable, as well as the communication mode (buffered mode and mailbox mode). SyncManagers use a buffer located in the memory area for exchanging data. Access to this buffer is controlled by the hardware of the SyncManagers.

A buffer has to be accessed beginning with the start address, otherwise the access is denied. After accessing the start address, the whole buffer can be accessed, even the start address again, either as a whole or in several strokes. A buffer access finishes by accessing the end address, the buffer state changes afterwards and an interrupt or a watchdog trigger pulse is generated (if configured). The end address cannot be accessed twice inside a frame.

It is recommended to align SyncManager buffer start addresses to 64 bit (8 byte) boundaries.

Two communication modes are supported by SyncManagers:

- Buffered mode
  - The buffered mode allows both sides, EtherCAT master and local application, to access the communication buffer at any time. The consumer always gets the latest consistent buffer which was written by the producer, and the producer can always update the content of the buffer. If the buffer is written faster than it is read out, old data will be dropped.
  - The buffered mode is typically used for cyclic process data.
- Mailbox mode
  - The mailbox mode implements a handshake mechanism for data exchange, so that no data will be lost. Each side, EtherCAT master or local application, will get access to the buffer only after the other side has finished its access. At first, the producer writes to the buffer. Then, the buffer is locked for writing until the consumer has read it out. Afterwards, the producer has write access again, while the buffer is locked for the consumer.
  - The mailbox mode is typically used for application layer protocols.

The SyncManagers accept buffer changes caused by the master only if the FCS of the frame is correct, thus, buffer changes take effect shortly after the end of the frame.

The configuration registers for SyncManagers are located beginning at register address 0x0800.

Table 25: SyncManager register overview

Description	Register address offset
Physical start address	0x0:0x1
Length	0x2:0x3
Control register	0x4
Status register	0x5
Activate	0x6
PDI control	0x7

8.1 Buffered mode

The buffered mode allows writing and reading data simultaneously without interference. If the buffer is written faster than it is read out, old data will be dropped. The buffered mode is also known as 3-buffer-mode.

Physically, 3 buffers of identical size are used for buffered mode. The start address and size of the first buffer is configured in the SyncManager configuration. The addresses of this buffer have to be used by the master and the local application for reading/writing the data. Depending on the SyncManager state, accesses to the first buffer's (0) address range are redirected to one of the 3 buffers. The memory used for buffers 1 and 2 cannot be used and should be taken into account for configuring other SyncManagers.

One buffer of the three buffers is allocated to the producer (for writing), one buffer to the consumer (for reading), and the third buffer keeps the last consistently written data of the producer.

The buffer interaction is shown in Figure 24:

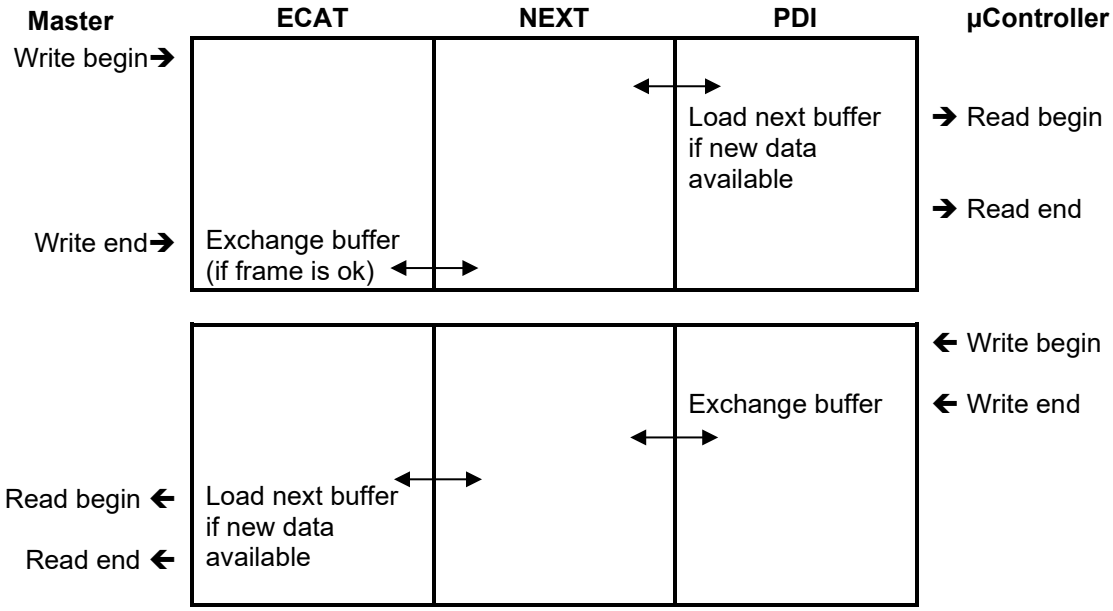


Figure 24: SyncManager buffered mode Interaction

The status register of the SyncManager reflects the current state. The last written buffer is indicated (informative only, access redirection is performed by the ESC), as well as the interrupt states. If the SyncManager buffer was not written before, the last written buffer is indicated to be 3 (start/empty).

8.1.1 Memory usage for ESCs with 8 bit processing data width

This example demonstrates a configuration with start address 0x1000 and length 0x100. The other buffers shall not be read or written. Access to the buffer is always directed to addresses in the range of buffer 0 (0x1000-0x10FF).

0x1000 - 0x10FF	**Buffer 0** (visible)	Required RAM for buffer 0
0x1100 - 0x11FF	Buffer 1 (invisible, cannot be used)	Required RAM for buffer 1
0x1200 - 0x12FF	Buffer 2 (invisible, cannot be used)	Required RAM for buffer 2
0x1300 …	Next usable RAM space	

All buffers are controlled by the SyncManager. Only buffer 0 is configured by the SyncManager and addressed by ECAT and µController.

Figure 25: SyncManager buffer allocation for ESCs with 8 bit processing data width

8.1.2 Memory usage for ESCs with 16/32/64 bit processing data width

ESCs which use an internal processing data width of more than 8 bit, e.g. 32 bit internal data width, will require special attention when planning the required memory area for SyncManager buffers. These ESCs treat each 32 bit word as a single address, and it is not allowed to configure multiple SyncManagers for the same 32 bit word.

Therefore, the start address of each SyncManager buffer has to be rounded down to the start of a 32 bit word, and the end address of each SyncManager buffer has to be rounded up to the end of a 32 bit word. Each buffer could occupy up to 3+3=6 bytes more for each buffer, so up to 3*6=18 bytes more for a 3-buffer SyncManager. In buffered mode, this extended memory area has to be multiplied by 3 (for buffer 1 and 2). The resulting memory area cannot be used for other SyncManagers.

Please note that there is no restriction on the SyncManager start address, end address, or length – none of them has to be 32 bit aligned.

This example demonstrates a configuration with start address 0x1003 and length 0xFA. The other buffers shall not be read or written. Access to the buffer is always directed to addresses in the range of buffer 0 (0x1003 – 0x10FC).

0x1000 - 0x1002	Alignment buffer 0 (cannot be used)	Required RAM for buffer 0
0x1003 - 0x10FC	**Buffer 0** (visible)	
0x10FD - 0x10FF	Alignment buffer 0 (cannot be used)	
0x1100 - 0x1102	Alignment buffer 1 (cannot be used)	Required RAM for buffer 1
0x1103- 0x11FC	**Buffer 1** (invisible, cannot be used)	
0x11FD - 0x11FF	Alignment buffer 1 (cannot be used)	
0x1200 - 0x1202	Alignment buffer 2 (cannot be used)	Required RAM for buffer 2
0x1203 – 0x12FC	**Buffer 2** (invisible, cannot be used)	
0x12FD - 0x12FF	Alignment buffer 2 (cannot be used)	
0x1300 …	Next usable RAM space	

All buffers are controlled by the SyncManager. Only buffer 0 is configured by the SyncManager and addressed by ECAT and µController.

Figure 26: SyncManager buffer allocation for ESCs with 32 bit processing data width

8.2 Mailbox mode

The mailbox mode only allows alternating reading and writing. This assures all data from the producer reaches the consumer. The mailbox mode uses just one buffer of the configured size.

At first, after initialization/activation, the buffer (mailbox, MBX) is writeable. Once it is written completely, write access is blocked, and the buffer can be read out by the other side. After it was completely read out, it can be written again.

The time it takes to read or write the mailbox does not matter.

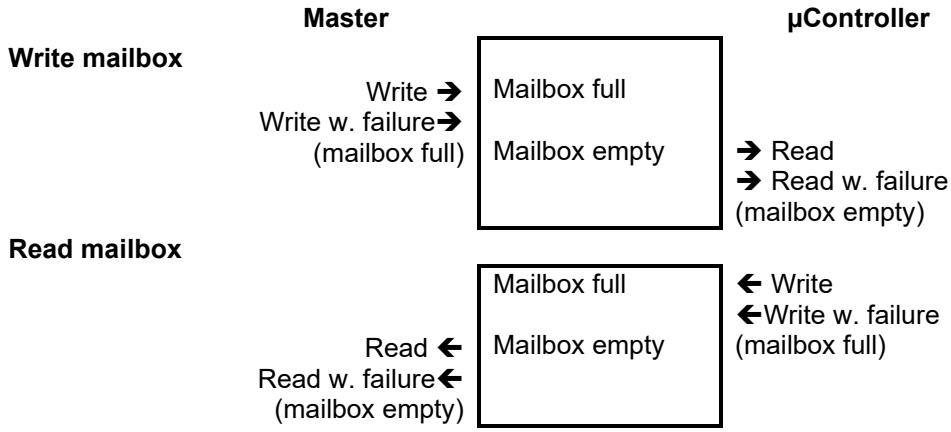


Figure 27: SyncManager mailbox Interaction

8.2.1 Memory usage for ESCs with 8 bit processing data width

This example demonstrates a configuration with start address 0x1000 and length 0x100.

0x1000 - 0x10FF	**Mailbox buffer** (visible)	Required RAM for buffer
0x1100 …	Next usable RAM space	

Figure 28: SyncManager buffer allocation for ESCs with 8 bit processing data width

8.2.2 Memory usage for ESCs with 16/32/64 bit processing data width

ESCs which use an internal processing data width of more than 8 bit, e.g. 32 bit internal data width, will require special attention when planning the required memory area for SyncManager buffers. These ESCs treat each 32 bit word as a single address, and it is not allowed to configure multiple SyncManagers for the same 32 bit word.

Therefore, the start address of each SyncManager buffer has to be rounded down to the start of a 32 bit word, and the end address of each SyncManager buffer has to be rounded up to the end of a 32 bit word. Each buffer could occupy up to 3+3=6 bytes more for each buffer, so up to 3*6=18 bytes more for a 3-buffer SyncManager. The resulting memory area cannot be used for other SyncManagers.

Please note that there is no restriction on the SyncManager start address, end address, or length – none of them has to be 32 bit aligned.

This example demonstrates a configuration with start address 0x1003 and length 0xFA..

0x1000 - 0x1002	Alignment (cannot be used)	Required RAM for buffer
0x1003 - 0x10FC	**Mailbox buffer** (visible)	
0x10FD - 0x10FF	Alignment (cannot be used)	
0x1100 …	Next usable RAM space	

Figure 29: SyncManager buffer allocation for ESCs with 32 bit processing data width

8.2.3 Mailbox communication protocols

This is only a brief overview of the mailbox communication protocols. For details on the mailbox protocols refer to the EtherCAT specification ETG.1000.6: “Application layer protocol specification” available from the EtherCAT Technology Group (http://www.ethercat.org) or to the IEC specification “Digital data communications for measurement and control – Fieldbus for use in industrial control systems”, IEC 61158 Type 12: EtherCAT, available from the IEC (http://www.iec.ch).

Ethernet over EtherCAT (EoE)
Defines a standard way to exchange or tunnel standard Ethernet frames over EtherCAT.

CAN application layer over EtherCAT (CoE)
Defines a standard way to access a CAN application layer object dictionary and to exchange CAN application layer Emergency and PDO messages on an event driven path.

File access over EtherCAT (FoE)
Defines a standard way to download and upload firmware and other ‘files’.

Servo profile over EtherCAT (SoE)
Defines a standard way to access the IEC 61800-7 identifier.

Vendor specific profile over EtherCAT (VoE)
A vendor specific protocol follows after a VoE header that identifies the vendor and a vendor specific type.

ADS over EtherCAT (AoE)
AoE defines a standard way to exchange automation device specification (ADS) messages over EtherCAT.

The content of an EtherCAT mailbox header is shown in Figure 30.

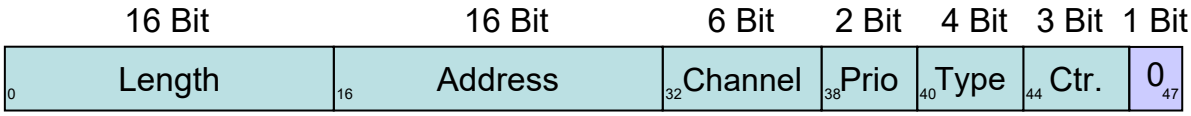


Figure 30: EtherCAT mailbox header (for all types)

Table 26: EtherCAT mailbox header

Field	Data type	Value/description
Length	WORD	Number of bytes of this mailbox command excluding the mailbox header
Address	WORD	Station address of originator
Channel	6 bit	0 – reserved for future use
Priority	2 bit	Priority between 0 (lowest) and 3 (highest)
Type	4 bit	Mailbox protocol types: 0x0: Error 0x1: Vendor specific (Beckhoff: AoE – ADS over EtherCAT) 0x2: EoE (Ethernet over EtherCAT) 0x3: CoE (CAN application layer over EtherCAT) 0x4: FoE (File access over EtherCAT) 0x5: SoE (Servo profile over EtherCAT) 0xF: Vendor specific (VoE)
Ctr.	3 bit	Sequence number that is used for detection of duplicated frames
Reserved	1 bit	0

8.3 PDI function acknowledge by write

Refer to chapter 16.3 for the background of this function, which only affects the SyncManager operation when the PDI reads a buffer. The EtherCAT operation is not influenced by this optional feature.

Reading a SyncManager buffer consists of the following steps, if PDI register function acknowledge by write is enabled:

- Read the first byte.
  This will open the buffer. Accidentally reading subsequent bytes has no influence, the buffer is not closed by just reading.
- Read any byte of the buffer.
  This includes the first byte, the last byte, and any inner byte. The buffer remains open, and the buffer can even be read multiple times. Accidentally reading the last byte has no influence.
- Write to the last byte of the buffer.
  Byte enable signals are used, write data is ignored (write 0). This write operation closes the buffer.

Accidentally reading the first byte of a second SyncManager buffer behind the one to be read is still possible, this will open the second SyncManager buffer. This can easily be prevented by aligning SyncManager buffer start addresses to the data width of the µController. It is recommended to align SyncManager buffer start addresses to 64 bit (8 byte) boundaries anyway.

8.4 Interrupt and watchdog trigger generation, latch event generation

Interrupts can be generated when a buffer was completely and successfully written or read. A watchdog trigger signal can be generated to rewind (trigger) the process data watchdog used for digital I/O after a buffer was completely and successfully written. Interrupt and watchdog trigger generation are configurable. The SyncManager status register reflects the current buffer state.

For debugging purposes it is also possible to trigger Distributed clocks latch events upon successful buffer accesses (for some ESCs).

8.5 Single byte buffer length / watchdog trigger for digital output PDI

If a SyncManager is configured for a length of 1 byte (or even 0), the buffer mechanism is disabled, i.e., read/write accesses to the configured address will pass the SyncManager without interference. The additional buffers 1 and 2 are not used in buffered mode, and alternation of write/read accesses is not necessary for mailbox mode. Consistency is not an issue for a single byte buffer.

Nevertheless, watchdog triggering is still possible if the buffer length is 1 byte (interrupt generation as well).

NOTE: For some ESCs in mailbox mode, watchdog and interrupt generation are depending on the alternation of write and read accesses, although the write/read accesses itself are executed without interference. I.e., buffered mode should be used for single byte buffers for watchdog generation.

Watchdog trigger generation with single byte SyncManagers is used for digital outputs, because the outputs are only driven with output data if the process data watchdog is triggered. One SyncManager has to be configured for each byte of the digital output register (0x0F00:0x0F03) which is used for outputs. The SyncManagers have to be configured like this:

- Buffered mode (otherwise the process data watchdog will not be wound up with some ESCs upon the second and following writes, because the digital I/O PDI does not read the addresses)
- Length of 1 byte
- EtherCAT write / PDI read
- Watchdog trigger enabled

For more details refer to the digital I/O PDI description of section III and the chapters about watchdog and digital output in this document.

NOTE: A SyncManager with length 0 behaves like a disabled SyncManager. It does not interfere accesses nor generate interrupt or watchdog trigger signals.

8.6 Repeating mailbox Communication

A lost datagram with mailbox data is handled by the application layer. The Repeat request/Repeat Acknowledge bits in the SyncManager Activation register (offset 0x06[1]) and the PDI control register (offset 0x07[1]) are used in mailbox mode for retransmissions of buffers from a slave to the master. If a mailbox read frame gets lost/broken on the way back to the master, the master can toggle the Repeat request bit. The slave polls this bit or receives an interrupt (SyncManager activation register changed, register 0x0220[4]) and writes the last buffer again to the SyncManager. Then the PDI toggles the Repeat Acknowledge bit in the PDI control register. The master will read out this bit and read the buffer content. Communication resumes afterwards.

This mechanism is shown in Figure 31 for a mailbox write service. The mailbox confirmation is lost on its way from the slave to the master and has to be repeated again.

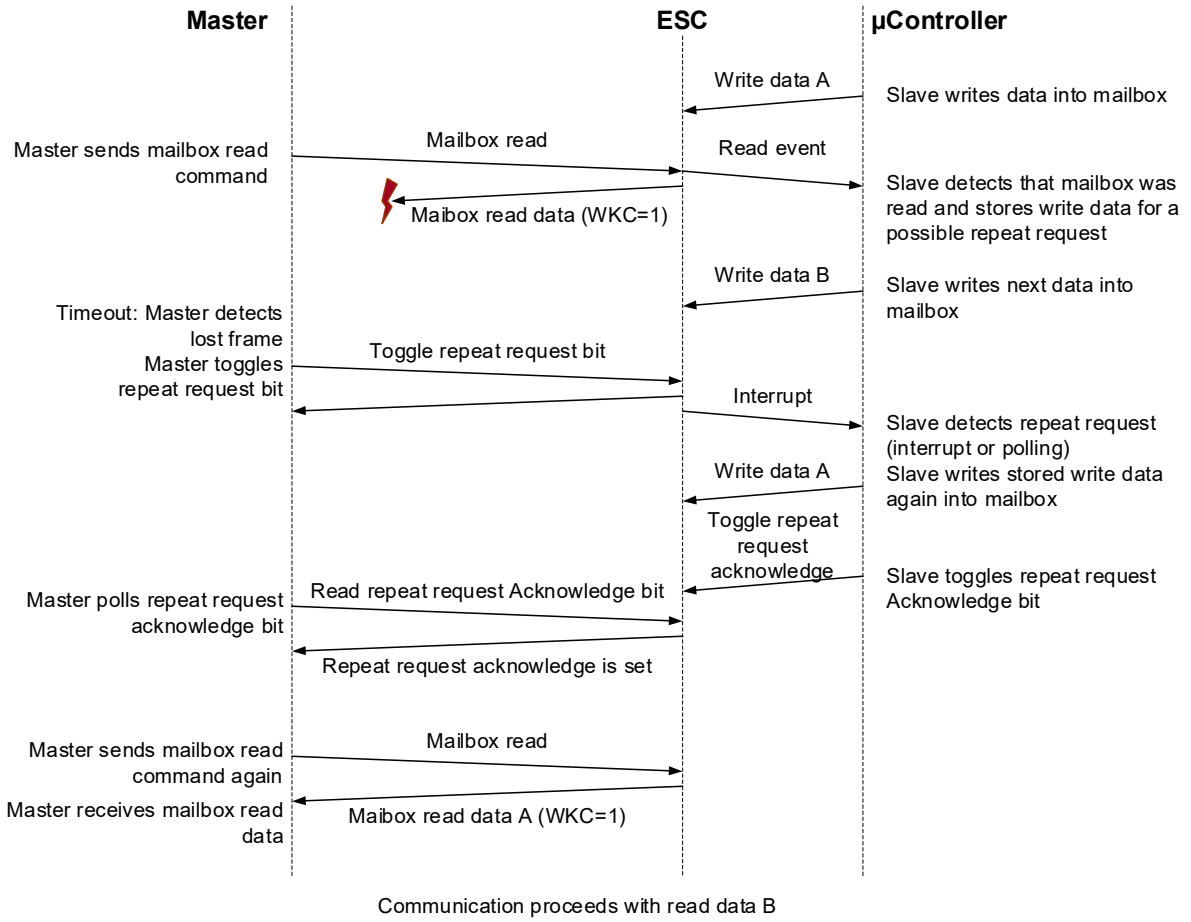


Figure 31: Handling of a repeat request with read mailbox

8.7 SyncManager deactivation by the PDI

A SyncManager can be deactivated by the PDI to inform the master of local problems (typically used in buffered mode only). The master can detect SyncManager deactivation by checking the working counter, which is not incremented if a deactivated SyncManager buffer is accessed. If a SyncManager is deactivated by the PDI (PDI control register 0x7[0]=1), the state of the SyncManager is reset, interrupts are cleared and the SyncManager has to be written first after re-activation. The entire SyncManager buffer area is read/write protected while the SyncManager is deactivated by the PDI.

8.8 SyncManager deactivation delay

When a SyncManager is deactivated by the EtherCAT master, the buffer management is immediately disabled. If the PDI is reading a buffer at that time, some bytes could come from buffer 1 or 2, while the last bytes come from buffer 0, resulting in inconsistent values. Typically, this does not occur: an EtherCAT master will at first leave the OPERATIONAL state, before it disables the SyncManagers. The local stack detects this, and discards the inconsistent read values before processing them.

For rare applications which require consistent data, this is not feasible. SyncManager deactivation delay addresses this. This optional feature can be enabled via the SII EEPROM or by PDI writing SyncManager PDI control register bit (0x0807+y*8)[6].

When deactivation delay is enabled for a SyncManager configured for PDI reading, and the master disables the SyncManager, the SyncManager enters a transitional state. This state allows the PDI side to read the last buffer, while it appears to be disabled for the master side (due to compatibility).

The transitional state is entered when all these conditions are fulfilled (registers are shown for SyncManager 0 only, all other SyncManagers have the same behavior):

- SyncManager deactivation delay is enabled (SM PDI control register 0x0807[6]=1)
- SyncManager is read by PDI
- SyncManager length is > 0
- Buffer has been written by ECAT
- SyncManager is disabled by ECAT

During the transitional state, the SyncManager exhibits the following behavior:

- ECAT reads SyncManager activate register 0x0806[0]=0 (SyncManager disabled)
- PDI reads SyncManager activate register 0x0806[0]=0 (SyncManager disabled, indicates that master requested disabling)
- Transitional state is shown in SyncManager PDI control register 0x0807[7]=1 (for ECAT and PDI)
- ECAT can reconfigure and reactivate the SyncManager
- ECAT cannot write to the buffer area (but it can read it)
- SyncManager buffer state is unchanged for PDI, PDI can continue reading the last buffer (or even start reading the latest buffer)
- ECAT reads the default value for SyncManager status register 0x0805 (unless SyncManager sequential mode is enabled – due to compatibility)
- PDI reads the actual value for SyncManager status register 0x0805

The transitional state is left when one of these conditions is met:

- ECAT re-activates the SyncManager (with the same or a different configuration)
- PDI disables the SyncManager (SyncManager PDI control register 0x0807[0]=1)
- PDI disables SyncManager deactivation delay (SyncManager PDI control register 0x0807[6]=0)
- PDI reads mailbox content completely

Although the transitional state could be maintained endlessly, ECAT could reactivate the SyncManager at any time, causing inconsistent values. Consequently, the PDI should finish reading the last buffer as fast as possible. The buffer should not be read again when 6 µs have elapsed since the master disabled the SyncManager. The PDI should stop the transitional state by disabling the SyncManager (SyncManager PDI control register 0x0807[0]=1), once it has read the buffer.

For debug purposes, the master can disable the transitional state by activating and deactivating the SyncManager.

Registers used for SyncManager deactivation delay:

Table 27: SyncManager deactivation delay register overview

Description	Register address offset
SyncManager deactivation delay enable	(0x0807+y*8)[6]
SyncManager deactivation delay status	(0x0807+y*8)[7]

8.9 SyncManager sequential mode

The basic architecture of a SyncManager uses the first and the last byte of the buffer for buffer change detection. Due to the maximum EtherCAT frame size of 1536 byte, a very large SyncManager buffer with more than ~3 KByte buffer size requires three or more frames to be written completely.

When the middle frame gets invalid, the ESC does not process it. If the invalid frame contains an access to at least one byte of a SyncManager buffer, the SyncManager buffer is considered invalid, and the buffer state is reset.

However, if the middle frame gets lost, or if the SyncManager cannot detect an access to the buffer area anymore, the SyncManager is not reset. Instead, when the last frame is received, the SyncManager buffer is taken over – although the data of the middle frame is missing. This results in inconsistent data.

The SyncManager sequential mode addresses consistency for large SyncManager buffers, by additionally checking if every byte has been written by the master sequentially.

The SyncManager sequential mode can be enabled by the EtherCAT master in the SyncManager control register 0x0804[7] (registers are shown for SyncManager 0 only, all other SyncManagers have the same behavior).

The SyncManager sequential mode only affects the EtherCAT writing side of a SyncManager. When it is enabled, every byte of the SyncManager buffer has to be written sequentially, starting with the first byte of the buffer.

If the last byte is written, while any previous byte was not written, the buffer will not be taken over, and an error flag will be set in SyncManager status register 0x0805[2].

An invalid frame which accesses the SyncManager buffer will completely reset the buffer, and require it to be written from the beginning again.

Registers used for SyncManager sequential mode:

Table 28: SyncManager sequential mode register overview

Description	Register address offset
SyncManager sequential mode supported	0x0008[12]
SyncManager sequential mode Enable	(0x0804+y*8)[7]
SyncManager sequential mode error	(0x0805+y*8)[2]

9 Distributed clocks

The distributed clocks (DC) unit of EtherCAT slave controllers supports the following features:

- Clock synchronization between the slaves (and the master)
- Generation of synchronous output signals (SyncSignals)
- Precise time stamping of input events (LatchSignals)
- Generation of synchronous interrupts
- Synchronous digital output updates
- Synchronous digital input sampling

9.1 Clock synchronization

DC clock synchronization enables all EtherCAT devices (master and slaves) to share the same EtherCAT system time. The EtherCAT devices can be synchronized to each other, and consequently, the local applications are synchronized as well.

For system synchronization all slaves are synchronized to one reference clock. Typically, the first ESC with distributed clocks capability after the master within one segment holds the reference time (system time). This system time is used as the reference clock to synchronize the DC slave clocks of other devices and of the master. The propagation delays, local clock sources drift, and local clock offsets are taken into account for the clock synchronization.

The ESCs can generate SyncSignals for local applications to be synchronized to the EtherCAT system time. SyncSignals can be used directly (e.g., as interrupts) or for digital output updating/digital input sampling. Additionally, LatchSignals can be time stamped with respect to the EtherCAT system time.

Definition of the system time

- Beginning on January, 1st 2000 at 0:00h
- Base unit is 1 ns
- 64 bit value (enough for more than 500 years)
- Lower 32 bits span over 4.2 seconds (typically enough for communication and time stamping). Some ESCs only have 32 bit DCs, which are compatible with 64 bit DCs.

Definition of the reference clock

One EtherCAT device will be used as a reference clock. Typically, the reference clock is the first ESC with DC capability between master and all the slaves to be synchronized (DC slaves). The reference clock might be adjusted to a “global” reference clock, e.g. to an IEEE 1588 grandmaster clock. The reference clock provides the system time.

Definition of the local clock

Each DC slave has a local clock, initially running independent of the reference clock. The difference between local clock and reference clock (offset) can be compensated, as well as clock drifts. The offset is compensated by adding it to the local clock value. The drift is compensated by measuring and adjusting the local clock speed.

Each DC slave holds a copy of the reference clock, which is calculated from the local clock and the local offset. The reference clock has a local clock, too.

Definition of the master clock

The reference clock is typically initialized by the EtherCAT master using the master clock to deliver the system time according to the system time definition. The EtherCAT master clock is typically bound to a global clock reference (RTC or the master PC, IEEE1588, GPS, etc.), which is either directly available to the master or indirectly by an EtherCAT slave providing access to the reference.

Propagation delay

The propagation delay between reference clock and slave clock has to be taken into account when the system time is distributed to the slaves.

Offset

The offset between local clock and reference clock has two reasons: the propagation delay from the ESC holding the reference clock to the device with the slave clock, and initial differences of the local times resulting from different times at which the ESCs have been powered up. This offset is compensated locally in each slave.

The ESC holding the reference clock derives the system time from its local time by adding a local offset. This offset represents the difference between local time (started at power-up) and master time (starting on January, 1st 2000 at 0:00h).

Drift

Since reference clock and DC slaves are typically not sourced by the same clock source (e.g., a quartz), their clock sources are subject to small deviations of the clock periods. The result is that one clock is running slightly faster than the other one, their local clocks are drifting apart.

ESC classification regarding DC support

Three classes of ESCs are distinguished regarding distributed clocks support:

1. Slaves supporting system time/time loop control unit:
   Receive time stamps and system time/time loop control unit available; SyncSignal generation, LatchSignal time stamping, and SyncManager event times are optionally supported depending on application.

2. Slaves supporting only propagation delay measurement:
   Mandatory for ESCs with 3 or more ports (topology devices like EK1100 and ET1100). Local clock and receive time stamps are supported.

3. Slaves without distributed clocks support:
   Slaves with max. 2 ports do not have to support DC features. Processing/forwarding delay of such slaves is treated like a “wire delay” by the surrounding DC capable slaves.

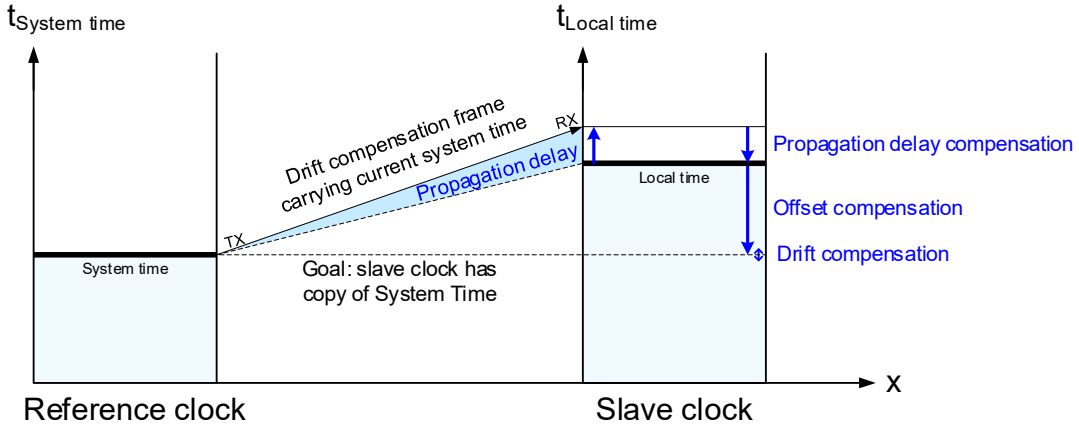
9.1.1 Clock synchronization process

The clock synchronization process consists of three steps:

1. Propagation delay measurement:
   The master initiates propagation delay measurement between all slaves in all directions. Each EtherCAT slave controller measures the receive time of the measurement frame. The master collects the time stamps afterwards and calculates the propagation delays between all slaves.
2. Offset compensation to reference clock (system time):
   The local time of each slave clock is compared to the system time. The difference is compensated individually by writing it to each slave. All devices get the same absolute system time.
3. Resetting the time control loop:
   The current filter status needs to be reset to eliminate influences of previous measurements and improve repeatability.
4. Drift compensation to reference clock:
   The drift between reference clock and local clock has to be compensated by regularly measuring the differences and readjusting the local clocks.

The following figure illustrates the compensation calculations for two cases, in the first case the system time is less than the slave's local time, in the second case, it is the other way around.

System time < local time



System time > local time

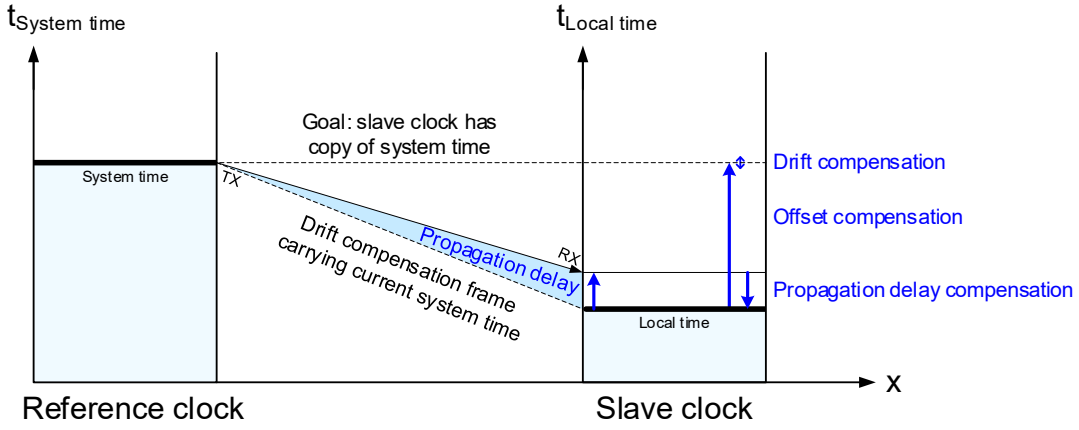


Figure 32: Propagation delay, offset, and drift compensation

9.1.2 Propagation delay measurement

Since each slave introduces a small processing/forwarding delay in each direction (within the device and also in the PHY), as well as the cable between the ESCs has a delay, the propagation delay between reference clock and the respective slave clock has to be considered for the synchronization of the slave clocks.

1. For measuring the propagation delay, the master sends a broadcast write to register DC receive time port 0 (at least the first byte).
2. Each slave device stores the time of its local clock when the first bit of the Ethernet preamble of the frame was received, separately for each port (receive time port 0-3 registers).
3. The master reads all time stamps and calculates the delay times with respect to the topology. The delay time between reference clock and an individual slave is written to the slave's system time delay register (0x0928:0x092B).

The receive time registers are used to sample the receive time of a specific frame (a broadcast write to receive time port 0 register).

The clocks must not be synchronized for the delay measurement, only local clock values are used. Since the local clocks of the slaves are not synchronized, there is no relation between the receive times of different slaves. So the propagation delay calculation has to be based on receive time differences between the ports of a slave.

Devices with two ports do not need to support distributed clocks at all, their delay is treated as an additional "wire delay" between the surrounding DC-capable slaves. Devices with more than 2 ports have to support at least propagation delay measurements (DC receive times).

NOTE: Some ESCs use the broadcast write to receive time port 0 register as an indicator to latch the receive times of the next frame at all ports other than port 0 (if port 0 is open). Thus, another frame which is still traveling around the ring might trigger the measurement, and the receive times do no correlate. For these ESCs, the ring has to be empty before the broadcast write is issued. Refer to section II receive time port x registers for further information.

Table 29: Registers for propagation delay measurement

Register address	Name	Description
0x0900:0x0903	Receive time port 0	Local time when receiving frame on port 0
0x0904:0x0907	Receive time port 1	Local time when receiving frame on port 1
0x0908:0x09B	Receive time port 2	Local time when receiving frame on port 2
0x090C:0x090F	Receive time port 3	Local time when receiving frame on port 3
0x0918:0x091F	Receive time ECAT processing unit	Local time when receiving frame at the ECAT processing unit
0x0936	Receive time latch mode	ET1200 only: receive time latching in forwarding or reverse mode

9.1.2.1 Propagation delay measurement in forwarding direction

For redundancy operation, it is necessary to perform propagation delay measurements either in processing direction (master connected to port 0), or in forwarding direction (master connected to port 1-3). For ET1200, care has to be taken because the measurement principle is slightly different for these two ESC types: DC receive time latch mode register 0x0936 has to be used. As newer ESCs do not require register 0x0936, the following rules can be used for propagation delay measurement in a mixed ET1200/other ESC environment (ESC20 does not support propagation delay measurement in forwarding direction mode).

- In processing direction, the master should write 0x00 to register 0x0936 of all slaves, and perform the delay measurement afterwards.
- In forwarding direction, the master should write 0x01 to register 0x0936 of all slaves, and perform the delay measurement afterwards.

Please refer to section II for more details on DC receive time latch mode register 0x0936 and receive time stamps in 0x0900:0x090F.

9.1.2.2 Propagation delay measurement example

The propagation delay between the local device and the reference clock device is calculated for the network example shown in Figure 33. The example assumes that slave A is the reference clock. The loops of slave D and F are closed internally. The wire delays are assumed to be symmetrical, and the processing and forwarding delays are assumed to be identical for all ESCs.

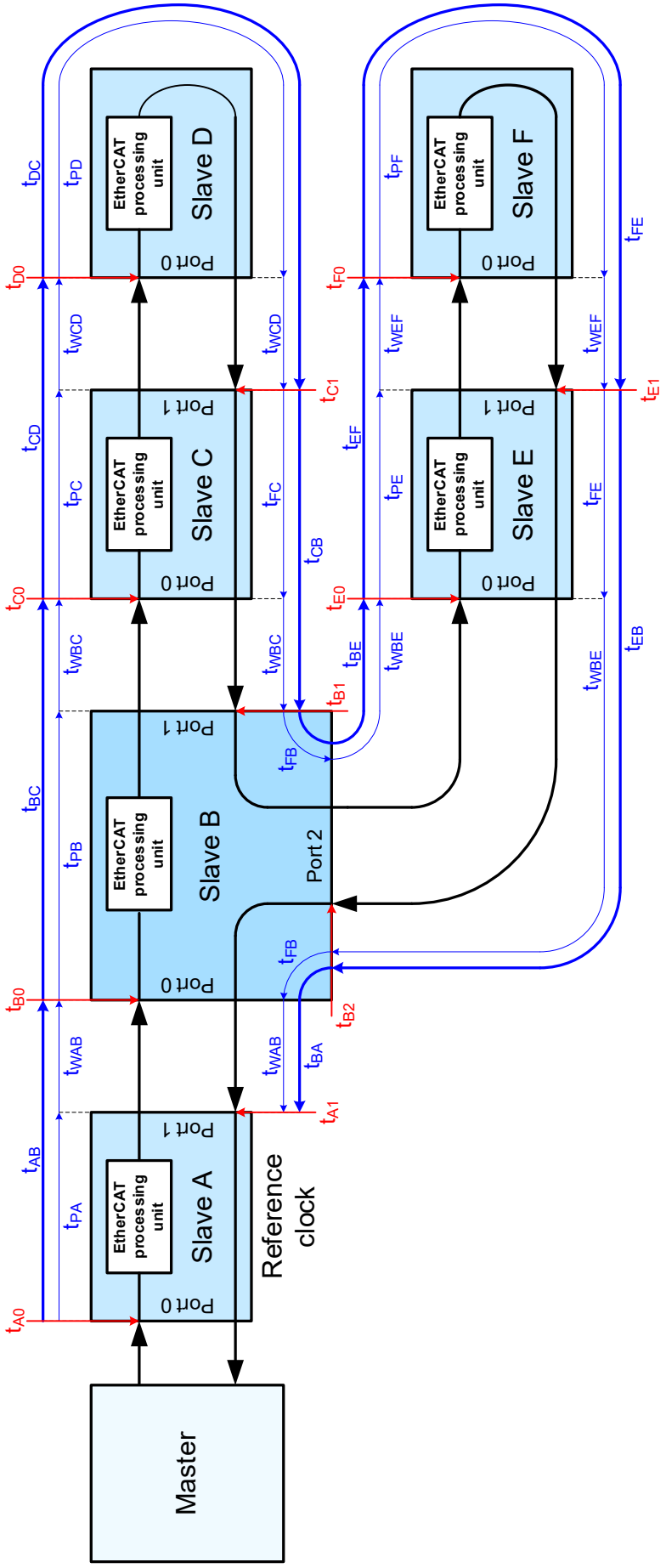


Figure 33: Propagation delay calculation

Parameters used for propagation delay calculation are listed in Table 30:

Table 30: Parameters for propagation delay calculation

Parameter	Description
t_{Px}	Processing delay of slave x (through EtherCAT processing unit, x=A-F)
t_{Fx}	Forwarding delay of slave x (alongside EtherCAT processing unit, x=A-F)
t_{xy}	Propagation delay from slave x to slave y (x/y=A-F)
t_{Wxy}	Wire propagation delay between slaves x and y (assumed to be symmetrical in both directions, x/y=A-F)
t_{x0}, t_{x1}, t_{x2}	Receive time port 0/1/2 values of slave x (time when first preamble bit is detected, x=A-F), measured with a write access to DC receive time 0 register.
t_P	Processing delay (through EtherCAT processing unit) if all slaves are identical
t_F	Forwarding delay (alongside EtherCAT processing unit) if all slaves are identical
t_{Diff}	Difference between processing delay and forwarding delay $t_{Diff} = t_P - t_F$ if all slaves are identical. ESC specific information, part of the ESI. Refer to section III for actual figures.
t_{Ref_x}	Propagation delay from reference clock (slave A) to slave x

Propagation delay between slave C and D

The propagation delays between slave C and D (t_{CD} and t_{DC}) consist of a processing delay and the wire delay:

$$t_{CD} = t_{PC} + t_{WCD}$$

$$t_{DC} = t_{PD} + t_{WCD}$$

Assuming that the processing delays of slave C and D are identical ($t_P = t_{PC} = t_{PD}$):

$$t_{CD} = t_{DC} = t_P + t_{WCD}$$

The two receive times of slave C have the following relation:

$$t_{C1} = t_{C0} + t_{CD} + t_{DC}$$

So the propagation delays between slave C and D are

$$t_{CD} = t_{DC} = (t_{C1} - t_{C0}) / 2$$

Propagation delay between slave B and C

The propagation delays between slave B and C (t_{BC} and t_{CB}) are calculated as follows:

$$t_{BC} = t_{PB} + t_{WBC}$$

$$t_{CB} = t_{FC} + t_{WBC}$$

Assuming that the processing delays of slaves B, C and D are identical ($t_P = t_{PB} = t_{PC} = t_{PD}$), and the difference between forwarding and processing delay of slave C is $t_{Diff} = t_{PC} - t_{FC}$:

$$t_{BC} = t_P + t_{WBC}$$

$$t_{CB} = t_{BC} - t_{Diff}$$

The receive times (port 0 and 1) of slave B have the following relation:

$$t_{B1} = t_{B0} + t_{BC} + t_{CD} + t_{DC} + t_{CB}$$

So the propagation delay between slave B and C is

$$2*t_{BC} - t_{Diff} = (t_{B1} - t_{B0}) - (t_{C1} - t_{C0})$$

$$t_{BC} = ((t_{B1} - t_{B0}) - (t_{C1} - t_{C0}) + t_{Diff}) / 2$$

And for the other direction:

$$t_{CB} = ((t_{B1} - t_{B0}) - (t_{C1} - t_{C0}) - t_{Diff}) / 2$$

Propagation delay between slave E and F

The propagation delays between slave E and F are calculated like the delays between slave C and D:

$$t_{EF} = t_{PE} + t_{WEF}$$

$$t_{FE} = t_{PF} + t_{WEF}$$

Assuming that the processing delays of slave E and F are identical ($t_P = t_{PE} = t_{PF}$):

$$t_{EF} = t_{FE} = (t_{E1} - t_{E0}) / 2$$

Propagation delay between slave B and E

The propagation delays between slave B and E (t_{BE} and t_{EB}) are calculated as follows:

$$t_{BE} = t_{FB} + t_{WBE}$$

$$t_{EB} = t_{FE} + t_{WBE}$$

Assuming that the processing delays of slaves B to F are identical ($t_P = t_{Px}$), and the difference between forwarding and processing delay of these slaves is $t_{Diff} = t_{Px} - t_{Fx}$:

$$t_{BE} = t_{EB} = t_P - t_{Diff} + t_{WBE}$$

The receive times port 1 and 2 of slave B have the following relation:

$$t_{B2} = t_{B1} + t_{BE} + t_{EF} + t_{FE} + t_{EB}$$

So the propagation delay between slave B and E is

$$2*t_{BE} = (t_{B2} - t_{B1}) - t_{EF} - t_{FE}$$

$$t_{BE} = t_{EB} = ((t_{B2} - t_{B1}) - (t_{E1} - t_{E0})) / 2$$

Propagation delay between slave A and B

The propagation delays between slave A and B are calculated as follows:

$$t_{AB} = t_{PA} + t_{WAB}$$

$$t_{BA} = t_{FB} + t_{WAB}$$

Assuming that the processing delays of all slaves are identical ($t_P = t_{Px}$), and the difference between forwarding and processing delay of these slaves is $t_{Diff} = t_{Px} - t_{Fx}$:

$$t_{AB} = t_P + t_{WAB}$$

$$t_{BA} = t_{AB} - t_{Diff}$$

The receive times of slave A have the following relation:

$$t_{A1} = t_{A0} + t_{AB} + (t_{B1} - t_{B0}) + (t_{B2} - t_{B1}) + t_{BA}$$

So the propagation delay between slave A and B is

$$t_{AB} = ((t_{A1} - t_{A0}) - (t_{B2} - t_{B0}) + t_{Diff}) / 2$$

And for the other direction:

$$t_{BA} = ((t_{A1} - t_{A0}) - (t_{B2} - t_{B0}) - t_{Diff}) / 2$$

Summary of propagation delay calculation between slaves

$t_{AB} = ((t_{A1} - t_{A0}) - (t_{B2} - t_{B0}) + t_{Diff}) / 2$

$t_{BA} = ((t_{A1} - t_{A0}) - (t_{B2} - t_{B0}) - t_{Diff}) / 2$

$t_{BC} = ((t_{B1} - t_{B0}) - (t_{C1} - t_{C0}) + t_{Diff}) / 2$

$t_{CB} = ((t_{B1} - t_{B0}) - (t_{C1} - t_{C0}) - t_{Diff}) / 2$

$t_{CD} = t_{DC} = (t_{C1} - t_{C0}) / 2$

$t_{EF} = t_{FE} = (t_{E1} - t_{E0}) / 2$

$t_{BE} = t_{EB} = ((t_{B2} - t_{B1}) - (t_{E1} - t_{E0})) / 2$

Propagation delays between reference clock and slave clocks

The system time delay register of each slave clock takes the propagation delay from the reference clock to the slave. This delay is calculated like this:

$t_{Ref\_B} = t_{AB}$

$t_{Ref\_C} = t_{AB} + t_{BC}$

$t_{Ref\_D} = t_{AB} + t_{BC} + t_{CD}$

$t_{Ref\_E} = t_{AB} + t_{BC} + t_{CD} + t_{DC} + t_{CB} + t_{BE}$

$t_{Ref\_F} = t_{AB} + t_{BC} + t_{CD} + t_{DC} + t_{CB} + t_{BE} + t_{EF}$

9.1.3 System time propagation

9.1.3.1 Offset compensation

The local time of each device is a free-running clock which typically will not have the same time as the reference clock. To achieve the same absolute system time in all devices, the offset between the reference clock and every slave device's clock is calculated by the master. The offset time is written to register system time offset to adjust the local time for every individual device. Small offset errors are eliminated by the drift compensation after some time, but this time might become extremely high for large offset errors – especially for 64 bit DCs.

Each DC slave calculates its local copy of the system time using its local time and the local offset value:

$t_{Local\ copy\ of\ system\ time} = t_{Local\ time} + t_{Offset}$

This time is used for SyncSignal generation and time stamping of LatchSignals. It is also provided to the PDI for use by µControllers.

The system time of the reference clock is bound to the master clock by calculating the difference and compensating it using the system time offset of the reference clock.

Registers used for offset compensation are listed in Table 31.

Table 31: Registers for offset compensation

Register address	Name	Description
0x142[8]	ESC configuration	Enable/disable DC system time unit (power saving)
0x0910:0x0917	System time	Local copy of system time (read from PDI)
0x0918:0x091F	Receive time ECAT processing unit	Local time ($t_{Local\ time}$)
0x0920:0x0927	System time offset	Difference between local time and system time (t_{Offset})

9.1.3.2 Resetting the time control loop

Before starting drift compensation, the internal filters of the time control loop must be reset. Their current status is typically unknown, and they can have negative impact on the settling time. The filters are reset by writing the speed counter start value to the speed counter start register (0x0930:0x0931). Writing the current value of the register again is sufficient to reset the filters.

Registers used for resetting the time control loop filters are listed in Table 31.

Table 32: Registers for resetting the time control loop

Register address	Name	Description
0x0930:0x0931	Speed counter start	Bandwidth for adjustment of local copy of system time Writing a value resets the internal filters.

9.1.3.3 Drift compensation

After the delay time between the reference clock and the slave clocks has been measured, and the offset between both clocks has been compensated, the natural drift of every local clock (emerging from quartz variations between the reference clock's quartz and the local quartz) is compensated by the time control loop which is integrated into each ESC.

For drift compensation, the master distributes the system time from the reference clock to all slave clocks periodically. The ARMW or FRMW commands can be used for this purpose. The time control loop of each slave takes the lower 32 bit of the system time received from the reference clock and compares it to its local copy of the system time. For this difference, the propagation delay has to be taken into account:

$$\Delta t = (t_{\text{Local time}} + t_{\text{Offset}} - t_{\text{Propagation delay}}) - t_{\text{Received system time}}$$

If Δt is positive, the local time is running faster than the system time, and has to be slowed down. If Δt is negative, the local time is running slower than the system time, and has to speed up. The time control loop adjusts the speed of the local clock.

For a fast compensation of the static deviations of the clock speeds, the master should initially send many ARMW/FRMW commands (e.g. 15,000) for drift compensation in separate frames after initialization of the propagation delays and offsets. The control loops compensate the static deviations and the distributed clocks are synchronized. Afterwards, the drift compensation frames are sent periodically for compensation of dynamic clock drifts.

NOTE: The system time offset allows fast compensation of differences between local copy of the system time and the system time, the drift compensation is very slow. Thus, shortly before drift compensation is started, the offset should be roughly compensated using the system time offset register. Otherwise settling time might become very high.

NOTE: Δt must not exceed 2^{30} ns (~ 1 second), otherwise stability is not guaranteed. For fast settling times, Δt should be as low as possible.

Time control loop configuration and status

The time control loop has some configuration and status registers (system time difference, speed counter start, speed counter difference, system time difference filter depth, and speed counter filter depth). The default settings of these registers are sufficient for proper operation of the drift compensation. Setting the speed counter filter depth (0x0935) to 0 improves control loop behavior.

The system time difference register (0x092C:0x092F) contains the mean value of the difference between local copy of the system time and the system time (Δt). This value converges to zero when both times are identical.

The speed counter start register (0x0930:0x0931) represents the bandwidth of the drift compensation. The value of the speed counter difference register (0x0932:0x0933) represents the deviation between the clock periods of the reference clock and the local ESC.

The system time difference filter depth register (0x0934) and the speed counter filter depth register (0x0935) set filter depths for mean value calculation of the received system times and of the calculated clock period deviations.

Registers used for control loop/drift compensation are listed in Table 33.

Table 33: Registers for drift compensation

Register address	Name	Description
0x0900:0x090F	Receive time port n	Local time when receiving frame on port n
0x0918:0x091F	Receive time ECAT processing unit	Local time when receiving frame for ECAT processing unit
0x0910:0x0917	System time	Local copy of system time (equals local time if system time offset=0, control loop filters are in reset state, and no write access to 0x0910 is performed)
0x0920:0x0927	System time offset	time difference between system time and local time
0x0928:0x092B	System time delay	delay between reference clock and the ESC
0x092C:0x092F	System time difference	Mean difference between local copy of system time and received system time values
0x0930:0x0931	Speed counter start	Bandwidth for adjustment of local copy of system time
0x0932:0x0933	Speed counter difference	Deviation between local clock period and reference clock's clock period
0x0934	System time difference filter depth	Filter depth for averaging the received system time deviation
0x0935	Speed counter filter depth	Filter depth for averaging the clock period deviation

9.1.3.4 Reference between DC registers/functions and clocks

Table 34: Reference between DC registers/functions and clocks

Register address	Name	Referring to clock
0x0900:0x090F	Receive time port n	Local time ($t_{Local\ time}$)
0x0918:0x091F	Receive time ECAT processing unit	Local time ($t_{Local\ time}$)
0x0910:0x0917	System time	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x0990:0y0997	SYNC0 start time	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x0998:0x099F	NEXT SYNC1 pulse	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x0950:0x0957	NEXT SYNC2 pulse	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x0958:0x095F	NEXT SYNC3 pulse	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09B0:0x09B7	Latch0 time positive edge	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09B8:0x09BF	Latch0 time negative edge	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09C0:0x09C7	Latch1 time positive edge	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09C8:0x09CF	Latch1 time negative edge	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09D0:0x09D7	Latch2 time positive edge	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09D8:0x09DF	Latch2 time negative edge	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09E0:0x09E7	Latch3 time positive edge	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09E8:0x09EF	Latch3 time negative edge	Local copy of system time ($t_{Local\ time} + t_{Offset}$)
0x09F0:0x09F3	EtherCAT buffer change event time	Local time ($t_{Local\ time}$)
0x09F8:0x09FB	PDI buffer start event time	Local time ($t_{Local\ time}$)
0x09FC:0x09FF	PDI buffer change event time	Local time ($t_{Local\ time}$)

9.1.3.5 When is synchronization established?

There are two possibilities to detect if DC synchronization of a slave is established:

- Read system time difference (0x92C:0x092F):
  If the difference is below an application specific threshold, DC has locked.
  Advantage: Can be read using a single BRD command for the entire network: if the upper N bits are zero, synchronization is established.
  Recommended if an EtherCAT slave is the reference clock. If the master is the reference clock, the threshold has to be increased to accomplish for the master jitter, which could make this solution unusable.
- Read speed counter difference (0x0932:0x0933):
  If the value is stable (within an application-specific range), DC has locked.
  Disadvantage: Loss of lock is recognized late.

9.1.3.6 Clock synchronization initialization example

The initialization procedure of clock synchronization including propagation delay measurement, offset compensation, filter reset, and drift compensation is shown in the following. After initialization, all DC slaves are synchronized with the reference clock.

1. Master reads the DL status register of all slaves and calculates the network topology.
2. Master sends a broadcast write to receive time port 0 register (at least first byte). All slaves latch the local time of the first preamble bit of this frame at all ports and at the ECAT processing unit. Some ESCs need the EtherCAT network to be free of frames before the broadcast write is sent.
3. Master waits until the broadcast write frame has returned.
4. Master reads all receive time port 0-3 registers (depending on the topology and the receive time ECAT processing unit register (0x0918:0x091F) which contains the upper 32 bits of the receive times.
5. Master calculates individual propagation delays and writes them to system time delay registers of the slaves. Possible overruns of the 32 bit receive times have to be checked and taken into account.
6. Master sets system time offset register of the reference clock so that the reference clock is bound to the master time. The offset for the reference clock is master time minus receive time ECAT processing unit (local time) of the reference clock.
7. Master calculates system time offsets for all DC slaves and writes them to the system time offset registers. The offset of each slave is receive time ECAT processing unit from reference clock minus receive time ECAT processing unit from each DC slave.
8. Master resets all slaves' time control loop filters by writing to the speed counter start register (0x0930:0x0931).
9. For static drift compensation, the master sends many separate ARMW or FRMW drift compensation frames (e.g., 15,000 frames) to distribute the system time of the reference clock to all DC slaves.
10. For dynamic drift compensation, the master sends ARMW or FRMW commands periodically to distribute the system time of the reference clock to all DC slaves. The rate of the drift compensation commands depends on the acceptable maximum deviation.

9.1.3.7 System time PDI controlled

Sometimes distributed clocks of different EtherCAT networks have to be synchronized. One solution is master-master communication, the other one is based on a physical device which is present in both EtherCAT networks. One of the networks contains the DC reference clock (DC source), the other one – DC destination – is synchronized to the reference clock in the DC source network.

Some ESCs support such a synchronization by a different functionality of the system time register (0x0910:0x0913). In normal operation mode, a write access initiated by the EtherCAT master to the system time register triggers the synchronization: the written value is compared to the local copy of the system time, and the difference is fed into the control loop. If the system time is PDI controlled, the PDI writes the system time register, and the written value is compared to the DC Latch0 time positive edge register (0x09B0:0x09B3). This feature makes the accuracy of the synchronization independent of the µController/PDI response times.

The following figures illustrate how the system time is transferred from the DC source to the DC destination. ESC 1 and ESC 2 are located in different EtherCAT networks. The EtherCAT network of ESC 1 contains the reference clock, the network of ESC 2 will become synchronized to this reference clock. ESC 2 is the "reference clock" of its EtherCAT network. There are two options for synchronization, which has to be performed on a regular basis.

The first option is to let the µController generate a trigger pulse for ESC 1 and 2. The time of the rising edge is stored in the Latch0 time positive edge register both in ESC 1 and 2. Afterwards, the µController reads this time from ESC 1 and writes it into the system time register of ESC 2. The difference of the Latch0 times is used to feed the control loop.

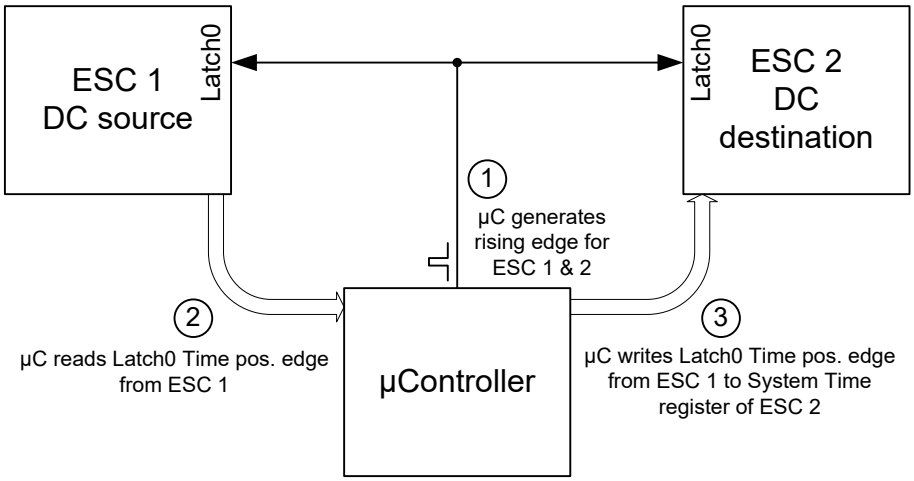


Figure 34: System time PDI controlled with three steps

The second option uses a SyncSignal output of ESC 1 to trigger Latch0 at ESC 2 and an interrupt at the µController. Upon receiving an interrupt, the µController writes the time of the SyncSignal pulse to the system time register of ESC 2. The µController has to calculate the time of the SyncSignal based upon start time cyclic operation and SYNC cycle time configuration of ESC 1 from interrupt to interrupt. The advantage of the second solution is less communication, the disadvantages are more calculation overhead and error detection/troubleshooting.

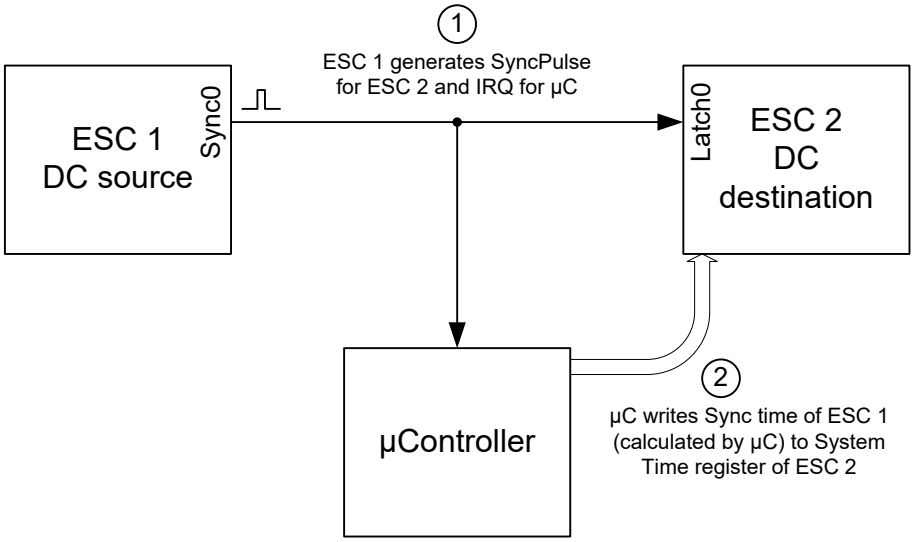


Figure 35: System time PDI controlled with two steps

9.1.4 Direct control of system time

In special situations, it can be useful for a master or a local µController to directly control the speed of the system time. This is possible using the speed counter diff direct control register.

Table 35: Registers for direct control of system time

Register address	Name	Description
0x0938:0x0938	Speed counter diff direct control	Direct control of local copy of system time

9.2 SyncSignals and LatchSignals

ESCs with distributed clocks support generation of SyncSignals and time stamping of LatchSignals. The SyncSignals can be used internally for

- Interrupt generation (mapping to AL event request register 0x0220:0x0223 and PDI IRQ)
- PDI digital output update events
- PDI digital input latch events

The SyncSignals can also be directly mapped to output signals (SYNC[3:0]) for use by external devices, e.g., as interrupt signals (less jitter than PDI IRQ, no interrupt source decoding).

The latch event unit supports time stamping of up to two LatchSignals (LATCH[3:0], rising and falling edge separately), and time stamping of SyncManager events for debugging purposes.

9.2.1 Interface

The distributed clocks unit has the following signals (depending on the ESC and the ESC configuration):

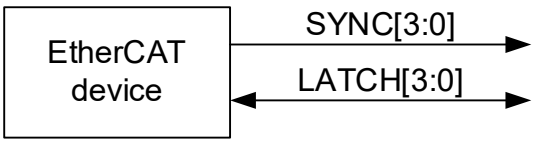


Figure 36: Distributed clocks signals

Table 36: Distributed clocks signals

Signal	Direction	Description
SYNC[3:0]	OUT	SyncSignals, also named SYNC0/SYNC1/SYNC2/SYNC3
LATCH[3:0]	IN	LatchSignals, also named LATCH0/LATCH1/LATCH2/LATCH3

Not all of these signals might be available depending on the ESC and its hardware configuration. Some ESCs have combined SYNC/LATCH signals, the selection is made by the setting of the Sync/Latch PDI configuration register 0x0151.

9.2.2 Configuration

The SyncSignals are internally available for interrupt generation, digital I/O synchronization, and SPI master start events, regardless of the availability at external signals. The mapping of SyncSignals to the AL event request register is controlled by the Sync/Latch PDI configuration register 0x0151.

The length of a SyncSignal pulse is defined in the DC pulse length of SYNC signals register (0x0982:0x0983). A value of 0 selects acknowledged modes.

Some ESCs support power saving options (partly disabling DC function units) at configuration, or using SII EEPROM. The possible functions which can be enabled are:

- Receive time measurement (mandatory for > 2 ports)
- System time (requires receive time measurement)
- SyncSignal [0] (requires system time)
- SyncSignal [1] (requires SyncSignal [0])
- SyncSignal [2] (requires SyncSignal [1])
- SyncSignal [3] (requires SyncSignal [2])
- LatchSignal [0] (requires system time)
- LatchSignal [1] (requires LatchSignal[0])
- LatchSignal [2] (requires LatchSignal[1])
- LatchSignal [3] (requires LatchSignal[2])

The SyncSignals are not driven (high-impedance) by some ESCs until the SII EEPROM is successfully loaded. Refer to section III for details. Take care of proper SyncSignal usage while the EEPROM is not loaded (e.g. pull-down/pull-up resistors).

9.2.3 SyncSignal generation

The DC cyclic unit / Sync unit supports the generation of a base SyncSignal SYNC0 and the dependent SyncSignals SYNC1-SYNC3. The SyncSignals can both be used internally and externally of the ESC. SyncSignals can be generated at a specific system time. Four operation modes are supported: cyclic generation, single shot, cyclic acknowledge, and single shot acknowledge mode. The acknowledged modes are typically used for interrupt generation. The interrupts have to be acknowledged by a µController.

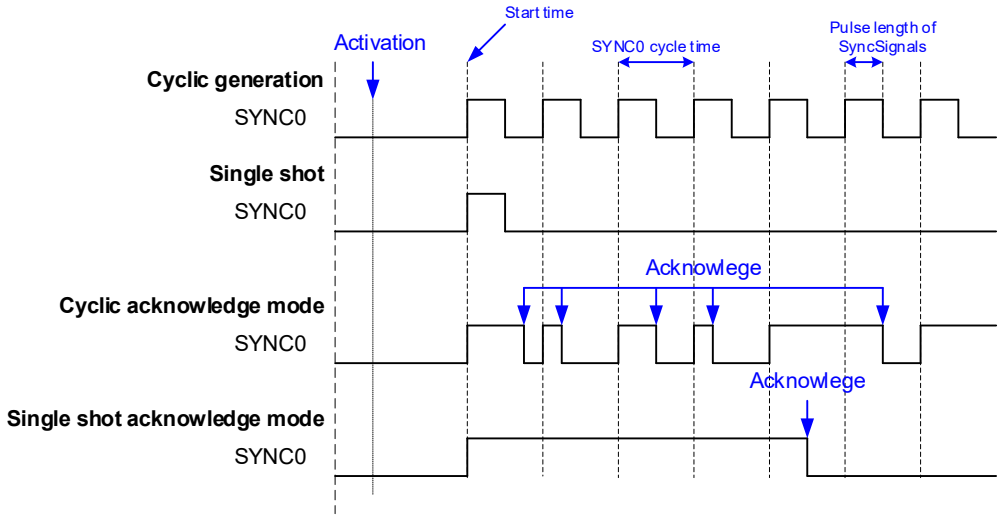


Figure 37: SyncSignal generation modes

The SyncSignal operation mode is selected by the configuration of the pulse length and the SYNC0 Cycle time, according to the following table:

Table 37: SyncSignal generation mode selection

Pulse length of SYNC signals (0x0982:0x0983)	SYNC0 cycle time (0x09A0:0x09A3)	
	> 0	= 0
> 0	Cyclic generation	Single shot
= 0	Cyclic acknowledge	Single shot acknowledge

The cycle time of the SYNC0 signal is configured in the SYNC0 Cycle time register (0x09A0:0x09A3), the start time is set in the start time cyclic operation register (0x0990:0x0997). After the Sync unit is activated and the output of the SYNC0/1 signals is enabled (DC Activation register 0x0981), the Sync unit waits until the start time is reached and generates the first SYNC0 pulse.

Some ESCs support additional activation options like auto-activation when the start time is written, or 64 bit extension if only 32 bit of the start time is written. Other options are to detect invalid start times and provide debug output of SyncSignals.

Internally, the SyncSignals are generated with an update rate of 100 MHz (10 ns update cycle). The jitter of the internal SyncSignal generation in comparison to the local copy of the system time is 12 ns.

The registers used for SyncSignal generation are shown in Table 38.

Table 38: Registers for SyncSignal generation

Register address	Name	Description
0x0141[3:2]	ESC configuration	Enable/disable DC sync/latch units (power saving)
0x0151	Sync/latch PDI configuration	Configuration of shared SYNC/LATCH pins, configuration of AL event mapping
0x0980[3:0]	Cyclic unit control	Assignment of SYNC unit control to EtherCAT or PDI
0x0981	Activation 0+1	Activation of cyclic function and SYNC[1:0]
0x0982:0x0983	Pulse length of SYNC signals	Length of SYNC impulse length
0x0984	Activation status	Activation status of SYNC[3:0]
0x0986	Activation 2+3	Activation of SYNC[3:2]
0x098E	SYNC0 status	Status of SYNC0 signal
0x098F	SYNC1 status	Status of SYNC1 signal
0x098C	SYNC2 status	Status of SYNC2 signal
0x098D	SYNC3 status	Status of SYNC3 signal
0x0990:0y0997	SYNC0 start time	Start system time of cyclic operation/ system time of next SYNC0 pulse
0x0998:0x099F	Next SYNC1 pulse	System time of next SYNC1 pulse
0x0950:0x0957	Next SYNC1 pulse	System time of next SYNC2 pulse
0x0958:0x095F	Next SYNC1 pulse	System time of next SYNC3 pulse
0x09A0:0x09A3	SYNC0 cycle time	Cycle time of SYNC0
0x09A4:0x09A7	SYNC1 cycle time	Cycle time of SYNC1
0x0940:0x0943	SYNC2 cycle time	Cycle time of SYNC2
0x0944:0x0947	SYNC3 cycle time	Cycle time of SYNC3

NOTE: Some of these registers are set via SII EEPROM/IP core configuration, or they are not available in specific ESCs. Refer to section II for details.

9.2.3.1 Cyclic generation

In cyclic generation mode, the SYNC unit generates isochronous SyncSignals after the start time. The generation ends if the cyclic unit is deactivated. The cycle times are determined by the SYNC[3:0] Cycle time registers. The pulse length of the SYNC signals has to be greater than 0. If the pulse length is greater than the cycle time, the SyncSignal will always be activated after the start time.

9.2.3.2 Single shot mode

In single shot mode (SYNC0 cycle time set to 0), only one SyncSignal pulse is generated after the start time is reached. Another pulse can only be generated by deactivating the cyclic unit (0x0981[0]=0), reprogramming the start time, and reactivation of the cyclic unit.

9.2.3.3 Cyclic acknowledge mode

The cyclic acknowledge mode is typically used for generation of isochronous interrupts. The acknowledged modes are selected by setting the pulse length of SYNC Signals to 0 (0x0982:0x0983). Each SyncSignal pulse remains active until it is acknowledged – typically by a µController – by reading the appropriate SYNC[3:0] status register (0x098D - 0x098F). The first pulse is generated after the start time is reached, following pulses are generated when the next regular SYNC[3:0] event would occur.

9.2.3.4 Single shot acknowledge mode

In single shot acknowledge mode (both pulse length of SyncSignals and SYNC0 cycle time are 0), only one pulse is generated when the start time is reached. The pulse remains active until it is acknowledged by reading the appropriate SYNC[3:0] status registers. Another pulse can only be generated by deactivating the cyclic unit (0x0981[0]=0), reprogramming the start time, and reactivation of the cyclic unit.

9.2.3.5 SYNC[3:1] generation

The derived SyncSignals (SYNC1/SYNC2/SYNC3) depend on SYNC0, they can be generated with a predefined delay after SYNC0 pulses. The delay is configured in the SYNC[3:1] cycle time registers individually for each SYNC[3:1]. The SYNC[3:1] signals are treated equally, the following description shows SYNC1 as an example – SYNC2 and SYNC3 are configured accordingly.

If the SYNC1 cycle time is larger than the SYNC0 cycle time, it will be generated as follows: when the start time cyclic operation is reached, a SYNC0 pulse is generated. The SYNC1 pulse is generated after the SYNC0 pulse with a delay of SYNC1 cycle time. The next SYNC1 pulse is generated when the next SYNC0 pulse was generated, plus the SYNC1 cycle time.

Some example configurations are shown in the following figure:

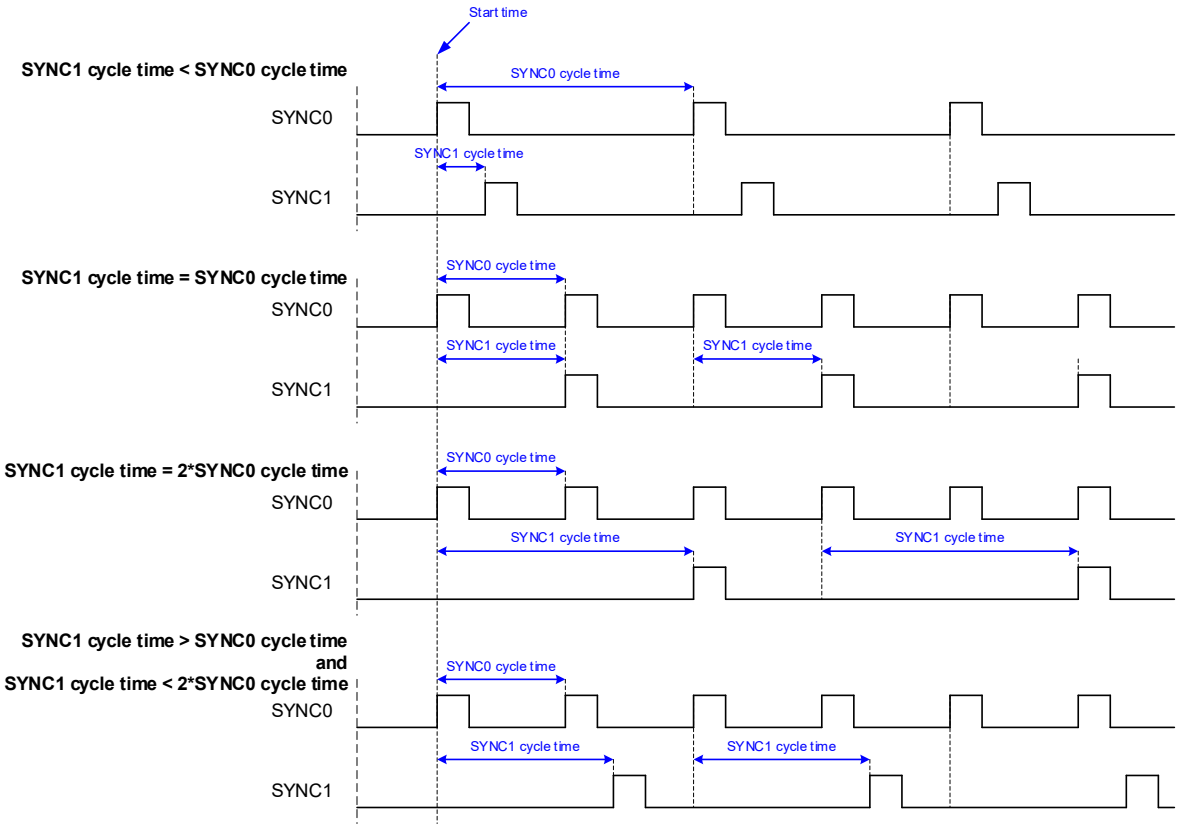


Figure 38: SYNC0/1 cycle time examples

NOTE: If The SYNC1 cycle time is 0, SYNC1 reflects SYNC0.

9.2.3.6 SyncSignal initialization example

The SyncSignal generation is initialized with the following procedure:

1. Enable DC SYNC unit (ESC specific)
2. Configure SyncSignal output to pins (ESC specific)
3. Set pulse length register (0x0982:0x0983, initialized by EEPROM) to pulse length of SYNC signals. Select a value > 0 ns for cyclic repetition of the SyncSignals
4. Assign SYNC unit to ECAT or PDI (0x0980, part of ESI)
5. Set cycle time of SYNC0 signal (0x09A0:0x09A3) and for SYNC1 signal (0x09A4:0x09A7). SYNC2/SYNC3 accordingly.
6. Set start time of cyclic operation (0x0990:0x0997) to a time later than the time the cyclic generation will be activated (end of activation frame; e.g., read the system time and add the time for writing start time and activation). For 32 bit DCs, the SyncSignal generation will start at worst after a turn-over of the system time (~ 4 s), but with 64 bit DCs, SyncSignal generation may start in hundreds of years.
7. Activate cyclic operation (0x0981[0]=1) to start cyclic generation of SyncSignals and activate SYNC[3:0] generation (0x0981[2:1]=0x7, 0x0986[1:0]=0x3). The Sync unit waits until the start time of cyclic operation is reached for the generation of the first SYNC0 pulse.

Register start time of cyclic operation and register next SYNC1 pulse can be read to get the time of the next output event. In the acknowledged modes, the Sync0/1 status registers (0x098E:0x098F) give the status of the SyncSignals. The SyncSignals are acknowledged by reading the SYNC0/1 status registers.

9.2.4 LatchSignals

The DC latch unit enables time stamping of LatchSignal events for up to four external signals, LATCH[3:0]. Both rising edge and falling edge time stamps are recorded for each signal. Additionally, time stamping of SyncManager events is possible with some ESCs.

LatchSignals are sampled with a sample rate of 100 MHz, the corresponding time stamp has an internal jitter of 11 ns.

The state of the LatchSignals can be read from the latch status registers (0x09AC - 0x09AF) – if supported by the ESC.

The DC latch unit supports two modes: single event or continuous mode, configured in the latch control registers (0x09A8 - 0x09AB).

The registers used for LatchSignal event time stamping are shown in Table 39:

Table 39: Registers for latch input events

Register address	Name	Description
0x0141[3:2]	ESC configuration	Enable/disable DC sync/latch units (power saving)
0x0151	Sync/Latch PDI configuration	Configuration of shared SYNC/LATCH pins, configuration of AL event mapping
0x0980[7:4]	Cyclic unit control	Assignment of LATCH unit control to EtherCAT or PDI
0x09A8	Latch0 control	Latch unit configuration for LATCH[0]
0x09A9	Latch1 control	Latch unit configuration for LATCH[1]
0x09AA	Latch2 control	Latch unit configuration for LATCH[2]
0x09AB	Latch3 control	Latch unit configuration for LATCH[3]
0x09AE	Latch0 status	Latch status LATCH[0]
0x09AF	Latch1 status	Latch status LATCH[1]
0x09AC	Latch2 status	Latch status LATCH[2]
0x09AD	Latch3 status	Latch status LATCH[3]
0x09B0:0x09B7	Latch0 time positive edge	System time at positive edge LATCH[0]
0x09B8:0x09BF	Latch0 time negative edge	System time at negative edge LATCH[0]
0x09C0:0x09C7	Latch1 time positive edge	System time at positive edge LATCH[1]
0x09C8:0x09CF	Latch1 time negative edge	System time at negative edge LATCH[1]
0x09D0:0x09D7	Latch2 time positive edge	System time at positive edge LATCH[2]
0x09D8:0x09DF	Latch2 time negative edge	System time at negative edge LATCH[2]
0x09E0:0x09E7	Latch3 time positive edge	System time at positive edge LATCH[3]
0x09E8:0x09EF	Latch3 time negative edge	System time at negative edge LATCH[3]
0x09F0:0x09F3	EtherCAT buffer change event time	Local time at beginning of frame causing ECAT SyncManager buffer change event
0x09F8:0x09FB	PDI buffer start event time	Local time at PDI SyncManager buffer start event
0x09FC:0x09FF	PDI buffer change event time	Local time at PDI SyncManager buffer change event

NOTE: Some of these registers are set via SII EEPROM/IP core configuration, or they are not available in specific ESCs. Refer to section II for details.

9.2.4.1 Single event mode

In single event mode, only the timestamps of the first rising and the first falling edge of the LatchSignals are recorded. The latch status registers (0x09AC - 0x09AF) contain information about the events which already have occurred. The latch time registers (0x09B0 to 0x09EF) contain the time stamps.

Each event is acknowledged by reading the corresponding latch time register. After reading the latch time register, the latch unit is waiting for the next event. Latch events are mapped to the AL event request register in single event mode.

9.2.4.2 Continuous mode

In continuous mode, each event is stored in the latch time registers. At reading, the time stamp of the last event is read. The latch status registers (0x09AC - 0x09AF) do not reflect the latch event states in continuous mode.

9.2.4.3 SyncManager event

Some ESCs support debugging of SyncManager interactions with time stamps for buffer events. The last event can be read out at the SyncManager event time registers (0x09F0:0x09FF), if the SyncManager is configured appropriately.

9.2.5 ECAT or PDI control

The SyncSignal units and the LatchSignal units of the distributed clocks entity can be assigned independently by the master to be controlled either by ECAT or a local µController (PDI) using the cyclic unit control register 0x0980. With PDI control, a µController can e.g. set up cyclic interrupts for itself.

9.3 Communication timing

Three communication modes are possible:

1. Free run
   EtherCAT communication and application are running independently from each other.
2. Synchronized to output event
   The slave application is synchronized to an output event. If no outputs are used the input event is used for synchronization.
3. Synchronized to SyncSignal
   Application is synchronized to the SyncSignal.

For further information please refer to the corresponding section within the EtherCAT information system.

The communication timing with use of distributed clocks is explained in Figure 39.

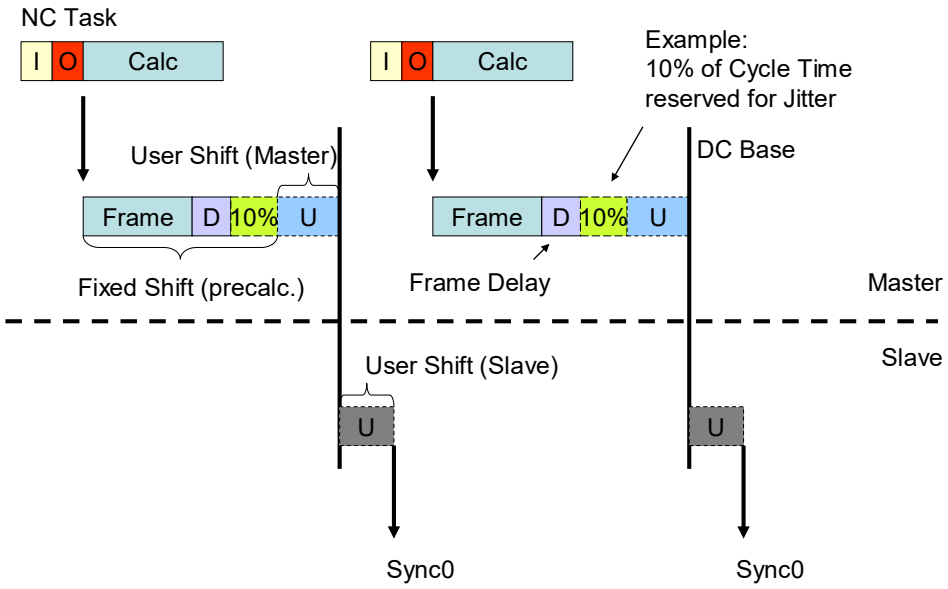


Figure 39: DC timing signals in relation to communication

I/O(master)
Time to load I/O data to communication buffer and vice versa.

Calc(master)
Processing time of the master.

Frame(communication)
Time to transmit the I/O data frame (about 5µs overhead plus 80ns per byte of data).

D(communication)
Delay time of the EtherCAT slaves to transfer data (approx. 1 µs with 100BASE-TX, plus line delay of approx. 5ns per m).

Jitter(communication)
Depends mostly on master timing quality.

U(communication-master)
Shift time that is adjusted internally by the master to deal with delays needed by the master and adjust the cycle time.

U(slave)
Delay time of the EtherCAT slaves. This can be set by each slave individually and is usually 0. There is a need to set this parameter in case of timing inaccuracies of the slave or to deal with slaves that have a slow output method compared to others with high speed output.

Cycle time jitter
Cycle time jitter is application-specific and depends on the jitter of the master system, the used infrastructure components and the slaves. This example assumes a time of 10% of the cycle time for jitter compensation.

10 EtherCAT state machine

The EtherCAT state machine (ESM) is responsible for the coordination of master and slave applications at start up and during operation. State changes are typically initiated by requests of the master. They are acknowledged by the local application after the associated operations have been executed. Unsolicited state changes of the local application are also possible.

Simple devices which do not support the EtherCAT state machine can be configured to use EtherCAT state machine emulation (device emulation) instead. These devices simply accept and acknowledge any state change automatically. Typically, devices using the digital I/O PDI use device emulation (ESM emulation).

There are four states an EtherCAT slave shall support, plus one optional state:

- Init (state after reset)
- Pre-Operational
- Safe-Operational
- Operational
- Bootstrap (optional)

The states and the allowed state changes are shown in Figure 40:

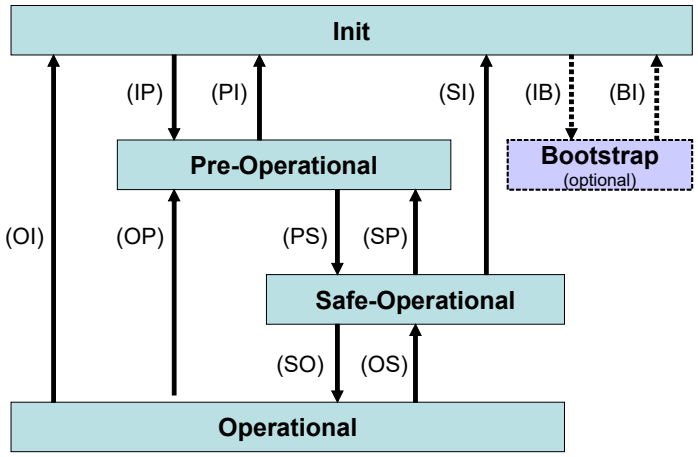


Figure 40: EtherCAT state machine

NOTE: Not all state changes are possible, e.g., the transition from ‘Init’ to ‘Operational’ requires the following sequence: Init → Pre-Operational → Save-Operational → Operational.

Each state defines required services. Before a state change is confirmed by the slave all services required for the requested state have to be provided or stopped respectively.

For more details please refer to the EtherCAT technology group website (http://www.ethercat.org).

10.1 EtherCAT state machine registers

The state machine is controlled and monitored via registers within the ESC. The master requests state changes by writing to the AL control register. The slave indicates its state in the AL status register and puts error codes into the AL status Code register.

Table 40: Registers for the EtherCAT state machine

Register address	Name	Description
0x0120:0x0121	AL control	Requested state by the master
0x0130:0x0131	AL status	AL status of the slave application
0x0134:0x0135	AL status Code	Error codes from the slave application
0x0141[0]	ESC configuration	Device emulation configuration

NOTE: The PDI control register is set via SII EEPROM/IP core configuration, other registers might not be available in specific ESCs. Refer to section II and section III for details.

10.1.1 AL control and AL status register

Writing the AL control register (0x0120:0x0121) initiates a state transition of the device state machine. The AL status register (0x0130:0x0131) reflects the current state of the slave.

Table 41: AL control and AL status register values

Register [3:0]	AL control register 0x0120	AL status register 0x0130
1	Request Init state	Init state
3	Request Bootstrap state (optional)	Bootstrap state (optional)
2	Request Pre-Operational state	Pre-Operational state
4	Request SAFE-Operational state	SAFE-Operational state
8	Request Operational state	Operational state

10.1.2 Device emulation (ESM emulation)

Simple devices (without µController) have the device emulation enabled (0x0141[0]=1). The AL control register is directly copied into the AL status register by the ESC. The master should not set the error indication acknowledge bit for such slaves at all, because setting this bit would result in setting the error indication bit – although no error occurred.

10.1.3 Error indication and AL status code register

The slave indicates errors during a state transition by setting the error indication flag (0x0130[4]=1) and writing an error description into the AL status code register (0x0134:0x0135). The master acknowledges the error indication flag of the slave by setting the error indication acknowledge flag (0x0120[4]).

For more information on defined AL status Codes, refer to the EtherCAT knowledge base available at the EtherCAT technology group website (http://www.ethercat.org, ETG.1020 Protocol enhancements).

11 SII EEPROM

EtherCAT slave controllers use a mandatory NVRAM (typically a serial EEPROM with I²C interface) to store EtherCAT slave information (ESI). EEPROM sizes from 1 Kbit up to 4 Mbit are supported, depending on the ESC.

The EtherCAT IP core supports omitting the serial I²C EEPROM if a µController with read/write access to an NVRAM (e.g., the one which contains the µController's program and data, or the FPGA configuration EPPROM) is used to emulate the EEPROM transactions. Since the logical interface is the same in this case, the EEPROM emulation is treated to be equivalent to the typical I²C EEPROM solution throughout this chapter. Refer to chapter 11.2.4 for more details about EEPROM emulation.

The EEPROM structure is shown in Figure 41. The SII uses word addressing.

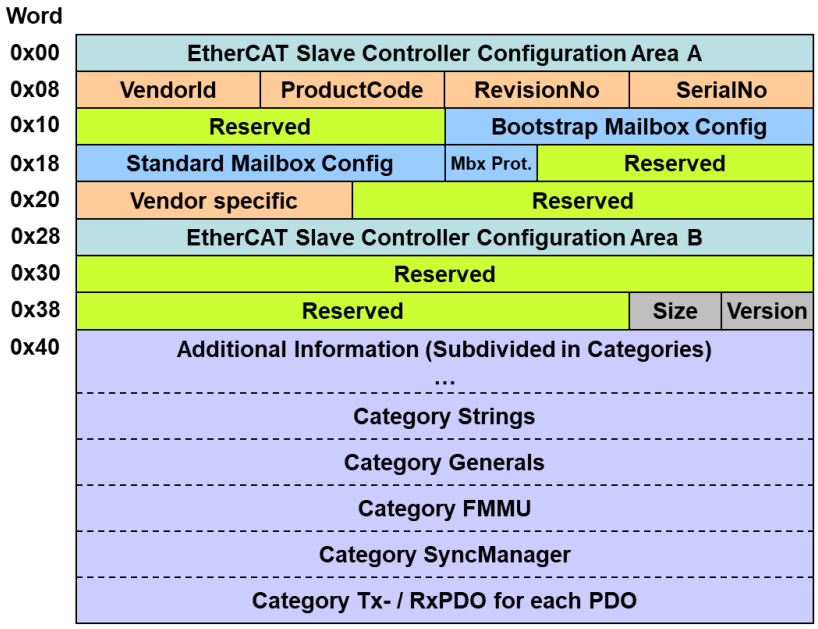


Figure 41: SII EEPROM layout

At least the information stored in the address range from word 0 to 63 (0x00 to 0x3F) is mandatory, as well as the general category (→ absolute minimum SII EEPROM size is 2Kbit, complex devices with many categories should be equipped with 32 Kbit EEPROMs or larger). The ESC configuration areas A and B are used by the ESC for configuration, area B is optional (existence is indicated in area A). All other parts are used by the master or the local application.

For a more detailed description of the ESI and other mandatory parts refer to the ETG.2000 EtherCAT slave information (ESI) specification, available from the download section of the EtherCAT technology group website (http://www.ethercat.org).

An EtherCAT slave must not be powered down/reset within 10 seconds after an EEPROM write access occurred, to allow internal storage (both for I²C devices and EEPROM emulation).

11.1 SII EEPROM content

The ESC configuration area A (EEPROM word addresses 0 to 7) is automatically read by the ESC after power-on or reset. If a flag in area A indicates that area B is present, ESC configuration area B (EEPROM word addresses 40 to 47) is also loaded by the ESC (as long as the ESC supports a second area). The ESC configuration areas contain the configurations for PDI0 and PDI1, DC settings, and the configured station alias. The consistency of the ESC configuration area data is secured with a checksum for each area.

For SII coding refer to ETG.1000 EtherCAT specification, part 6, clause 5.4, available in the download area of the EtherCAT technology group website (http://www.ethercat.org).

The EtherCAT master can invoke reloading the EEPROM content. In this case the configured station alias register 0x0012:0x0013 and ESC configuration register bits 0x0141[1,4,5,6,7] (enhanced link detection) are not transferred into the registers again, they are only transferred at the initial EEPROM loading after power-on or reset.

The ESC configuration areas are shown in Table 42.

Table 42: ESC configuration area

Word address	Register name	Parameter	Description	Register address
ESC configuration area A				
0x0	A0	PDI0 control/ ESC configuration A0	PDI0 selection, ESC configuration	0x0140:0x0141
0x1	A1	PDI0 configuration	Initialization value for PDI0 configuration register	0x0150:0x0151
0x2	A2	Pulse length of SYNC Signals	Initialization value for pulse length of SYNC Signals register	0x0982:0x0983
0x3	A3	Extended PDI0 configuration	Initialization value for extended PDI0 configuration register	0x0152:0x0153
0x4	A4	Configured station alias	Initialization value for configured station alias address register	0x0012:0x0013
0x5	A5	ESC configuration A5	ESC configuration	0x0142:0x0143
0x6	A6	ESC configuration A6	ESC configuration area B present, other ESC configuration	0x0144:0x0144
0x7	A7	Checksum A	Low byte contains remainder of division of word 0 to word 6 as unsigned number divided by the polynomial x^8+x^2+x+1(initial value 0xFF). NOTE: For debugging purposes it is possible to disable the checksum validation with a checksum value of 0x88A4. Never use this for production!	-
ESC configuration area B (optional)				
0x28	B0	PDI1 control	PDI1 selection, ESC configuration	0x0180:0x0181
0x29	B1	PDI1 configuration	Initialization value for PDI1 configuration register	0x0190:0x0191
0x2A	B2	Reserved		-
0x2B	B3	Extended PDI1Configuration	Initialization value for extended PDI1 configuration register	0x0192:0x0193
0x2C	B4	ESC configuration B4	ESC configuration	0x0188:0x0189
0x2D	B5	ESC configuration B5	ESC configuration	0x0182:0x0183
0x2E	B6	ESC configuration B6	ESC configuration	0x0184:0x0185
0x2F	B7	Checksum B	Low byte contains remainder of division of word 40 to word 46, similar to A7	

NOTE: Reserved words or reserved bits of the ESC configuration areas shall be filled with 0.

An excerpt of the SII EEPROM content following the ESC configuration area is shown in Table 43. For more information, refer to the ETG.2010 EtherCAT slave information interface (SII) specification, available from the download section of the EtherCAT technology group website (http://www.ethercat.org).

Table 43: SII EEPROM content excerpt

Word Address	Parameter	Word Address	Parameter
0x0	PDI0 control	0x20:22	Vendor specific
0x1	PDI0 configuration	0x23:0x27	Reserved
0x2	Pulse length of SYNC Signals	0x28	PDI1 control
0x3	Extended PDI0 configuration	0x29	PDI1 configuration
0x4	Configured station alias	0x2A	Reserved
0x5	ESC configuration A5	0x2B	Extended PDI1 configuration
0x6	ESC configuration A6	0x2C	ESC configuration B4
0x7	Checksum A	0x2D	ESC configuration B5
0x8:0x9	Vendor ID	0x2E	ESC configuration B6
0xA:0xB	Product code	0x2F	Checksum B
0xC:0xD	Revision number	0x30:0x3D	Reserved
0xE:0xF	Serial number	0x3E	Size
0x10:0x13	Reserved	0x3F	Version
0x14	Bootstrap receive mailbox offset	0x40	Category header
0x15	Bootstrap receive mailbox Size	0x41	Category word size
0x16	Bootstrap send mailbox offset	0x42	Category data
0x17	Bootstrap send mailbox Size	…	Second category header …
0x18	Standard receive mailbox offset		
0x19	Standard receive mailbox Size		
0x1A	Standard send mailbox offset		
0x1B	Standard send mailbox Size		
0x1C	Mailbox protocol		
0x1D:0x1F	Reserved		

11.2 SII EEPROM logical interface

The SII EEPROM interface of the ESC is either controlled by EtherCAT or by the PDI. Initially, EtherCAT has EEPROM interface access, but it can transfer access to the PDI.

Table 44: SII EEPROM interface register Overview

Register address	Description
0x0500	EEPROM configuration
0x0501	EEPROM PDI access state
0x0502:0x0503	EEPROM control/status
0x0504:0x0507	EEPROM address
0x0508:0x050F	EEPROM data

The EEPROM interface supports three commands: write to one EEPROM address (1 word), read from EEPROM (2 or 4 words, depending on ESC), or reload ESC configuration from EEPROM.

11.2.1 SII EEPROM errors

The ESC retries reading the EEPROM after power-on or reset once if an error has occurred (missing acknowledge, wrong checksum). If reading the ESC configuration areas fails twice, the error Device information bit is set, and the PDI Operational bit in the ESC DL status register (0x0110[0]) remains clear and the EEPROM_Loaded signal (if available) remains inactive. The process memory is not accessible until the ESC configuration areas are loaded successfully.

All registers initialized by the ESC configuration areas keep their values in case of an error. This is also true for the error device information bit as well as the PDI operational bit. Only if the EEPROM was loaded/reloaded successfully, the registers take over the new values (except for configured station alias register 0x0012:0x0013 and ESC configuration register bits 0x0141[1,4,5,6,7] – enhanced link detection).

The SII EEPROM interface has these error status bits in register EEPROM control/status (0x0502:0x0503):

Table 45: SII EEPROM interface errors

Bit	Name	Description
11	Checksum error	ESC configuration area checksum is wrong (after device initialization or EEPROM reload). Registers initialized with EEPROM values keep their value. Reason: CRC error Solution: Check CRC
12	Error device information	ESC configuration not loaded Reasons: Checksum error, acknowledge error, EEPROM missing Solution: Check other error bits
13	Error acknowledge/ command	Missing Acknowledge or invalid command Reason: a) Missing acknowledge from EEPROM chip (see below). EEPROM chip is busy performing operations internally or EEPROM chip is not available. b) Invalid command issued Solution: a) Retry access. EEPROM device does not acknowledge if it is internally busy. b) Use valid commands EEPROM emulation only: Missing acknowledge error indicates a temporary failure. Invalid command error is automatically supported by the EEPROM interface.
14	Error write enable	Write without write enable (ECAT control only): Reason: ECAT issued a write command without write enable bit set (0x0502[0]) Solution: Set write enable bit in the same frame as the write command

11.2.1.1 Missing acknowledge

Missing acknowledges from the EEPROM chip are a common issue, especially if a fast PDI uses the EEPROM interface. E.g., a write access to the EEPROM with missing acknowledge may look like this:

1. ECAT/PDI issue write command (first command)
2. ESC is busy transferring the write data to the EEPROM chip.
3. ESC is not busy anymore. EEPROM chip is internally busy transferring data from input buffer to storage area.
4. ECAT/PDI issue a second command.
5. ESC is busy transferring the write data to the EEPROM chip. EEPROM chip does not acknowledge any access until internal transfer has finished (may take up to several ms).
6. ESC is not busy anymore. error acknowledge/command bit is set. (ESC has to re-issue the second command after EEPROM chip is finished and the command is acknowledged).
7. EEPROM chip finishes internal transfer.
8. ESC re-issues the second command, the command is acknowledged and executed successful.

This is also possible for a read access, because some EEPROM chips require an idle period between any two accesses. During this idle period, they do not acknowledge any access.

11.2.2 SII EEPROM interface assignment to ECAT/PDI

The EtherCAT master controls the EEPROM interface (default) if EEPROM configuration register 0x0500[0]=0 and EEPROM PDI access register 0x0501[0]=0, otherwise PDI controls the EEPROM interface. These access rights should be checked by both sides before using the EEPROM interface.

A typical EEPROM interface control hand-over is as follows:

1. ECAT assigns EEPROM interface to PDI by writing 0x0500[0]=1
2. If PDI wishes to access EEPROM, it takes over EEPROM control by writing 0x0501[0]=1.
3. PDI issues EEPROM commands.
4. After PDI has finished EEPROM commands, PDI releases EEPROM control by writing 0x0501[0]=0.
5. ECAT may take back the EEPROM interface by writing 0x0500[0]=0
6. ECAT checks EEPROM control by reading 0x0501
7. ECAT issues EEPROM commands.

If the PDI does not release EEPROM control (e.g. because of a software failure), ECAT can force releasing the access:

1. ECAT writes 0x02 to register 0x0500 (as the result, 0x0501[0] is cleared)
2. ECAT writes 0x00 to register 0x0500
3. ECAT has control over EEPROM interface

11.2.3 Read/write/reload example

The following steps have to be performed for an SII EEPROM access:

1. Check if the busy bit of the EEPROM status register is cleared (0x0502[15]=0) and the EEPROM interface is not busy, otherwise wait until the EEPROM interface is not busy anymore.
2. Check if the error bits of the EEPROM status register are cleared. If not, write “000” to the command register (register 0x0502 bits [10:8]).
3. Write EEPROM word address to EEPROM address register.
4. Write command only: put write data into EEPROM data register (1 word/2 byte only).
5. Issue command by writing to control register.
   a) For a read command, write 001 into command register 0x0502[10:8].
   b) For a write command, write 1 into write Enable bit 0x0502[0] and 010 into command register 0x0502[10:8]. Both bits have to be written in one frame. The write enable bit realizes a write protection mechanism. It is valid for subsequent EEPROM commands issued in the same frame and self-clearing afterwards. The write enable bit needs not to be written from PDI if it controls the EEPROM interface.
   c) For a reload command, write 100 into command register 0x0502[10:8].
6. The command is executed after the EOF if the EtherCAT frame had no errors. With PDI control, the command is executed immediately.
7. Wait until the busy bit of the EEPROM status register is cleared.
8. Check the error bits of the EEPROM status register. The error bits are cleared by clearing the command register. Retry command (back to step 5) if EEPROM acknowledge was missing. If necessary, wait some time before retrying to allow slow EEPROMs to store the data internally.
9. a) For a read command: Read data is available in EEPROM data registers (2 or 4 Words, depending on ESC – check register 0x0502[6]).
   b) For a reload command: ESC configuration is reloaded into appropriate registers.

NOTE: The command register bits are self-clearing. Manually clearing the command register will also clear the status information.

11.2.4 EEPROM emulation

The EEPROM emulation mode is used in ESCs with a non-volatile memory (NVRAM) attached to a µController. The ESC configuration and the device description can be stored in the NVRAM of the µController, e.g., together with the program or FPGA configuration code. An additional I²C EEPROM chip for the ESC is not needed anymore if EEPROM emulation is used.

The µController performs the read and write actions to the NVRAM upon request of the ESC.

From the EtherCAT master's point of view, there is no difference between EEPROM emulation mode and I²C EEPROM: the master just issues EEPROM commands and waits until the EEPROM interface is not busy anymore.

In EEPROM emulation mode, the ESC issues an interrupt to the µController if an EEPROM command is pending and it automatically sets the busy bit 0x0502[15]. While the busy bit is set, the µController can read out the command and the EEPROM address. For a write access, write data is present in the data register. For a read command, read data has to be stored in the data register by the µController.

A reload command requires, read data has to be stored in the data register by the µController, depending on the EEPROM data register size:

- 32 bit (0x0502[6]=0): the µController has to place specifically formatted reload data in the data register 0x0508.
- 64 bit (0x0502[6]=1): the µController can treat the reload command as a read command.

Once the µController has finished reading/writing the EEPROM data register, it acknowledges the command by writing to the EEPROM command register bits. The µController has to write the command value it has executed into the EEPROM command register. Errors can be indicated using two of the error bits. After acknowledging the command, the EEPROM busy bit 0x0502[15] is automatically cleared, and the interrupt is released.

It is acceptable for the µController to use a cached image of the EEPROM content, to reduce access times and write cycles to the NVRAM. This is especially recommended for flash implementations.

The initial content of the NVRAM is relevant for ESCs with 32 bit EEPROM data register (0x0502[6]=0), because it does not support CRC checking: it should either be a valid ESI content, or all zero. The initial content for ESCs with 64 bit EEPROM data register is not relevant, as long as the CRC (word A7) is invalid. It should be either a valid ESI content, all zero, or all ones.

NOTE: The EEPROM can be assigned to the PDI even if EEPROM emulation is used. Depending on the busy bit 0x0502[15], the current interface mode is changing (issuing command vs. performing access).

11.2.4.1 EEPROM emulation example for 32 bit EEPROM data register size (0x0502[6]=0)

This chapter only applies for ESCs with 32 bit EEPROM data register size (0x0502[6]=0), i.e., the EtherCAT IP cores for FPGAs.

The following steps have to be performed to process EEPROM commands by the µController. Instead of interrupt operation as shown here, polling of EEPROM busy status bit (0x502[15]) can be used.

1) µController waits for EEPROM command:
   a) Interrupt driven:
      i) Once: enable EEPROM emulation interrupt by setting AL event mask register 0x0204[5] to 1.
      ii) Wait for interrupt from ESC (if not already pending), and read AL event request register. 0x220[5]=1 indicates an EEPROM command is pending, which is processed as following.
   b) Poll EEPROM busy status bit: 0x502[15]=1 indicates an EEPROM command is pending.
2) Read EEPROM command register (0x502[10:8]).
3) Execute command:
   a) Read command (0x502[10:8] =“001”):
      Fetch 4 bytes of data from NVRAM according to word address value in EEPROM address register (0x504:0x0507), and place data in EEPROM data register (0x0508:0x050B).

   b) Reload command (0x502[10:8] = “100”):
      Fetch first 8 words from NVRAM (ESC configuration area A, starting at address 0). Reformat data according to the following table (or section II, table “EEPROM data for EEPROM Emulation Reload IP core (0x0508:0x050B)”), and write result to EEPROM data register (0x0508:0x050B):

Table 46: SII EEPROM emulation reload data formatting when 0x0502[6]=0

Copy data from NVRAM	Copy data to EEPROM data register	Description	ESC will automatically transfer data to register
Word A4[15:0]	0x0508[15:0]	Configured station alias	0x0012[15:0]
Word A0[9]	0x0508[16]	Enhanced link detection all ports	0x0141[1]
Word A0[15:12]	0x0508[20:17]	Enhanced link detection ports 3:0	0x0141[7:4]
Word A5[7:4]	0x0508[24:21]	ESC DL configuration	0x0100[23:20]
Word A5[11:9]	0x0508[27:25]	FIFO size reduction	0x0100[18:16]
N.A.	0x0508[31:28]	Reserved, write 0	N.A.

   c) Write command (0x502[10:8] = “010”):
      Read data from EEPROM data register (0x0508:0x0509), and store data in NVRAM according to word address value in EEPROM address register (0x504:0x0507).

      NOTE: It is allowed to cache write data for NVRAM for a short time, before actually writing them into the NVRAM. This enables combining several word writes into a single page write, e.g. for flash memory.

4) Acknowledge command by writing to EEPROM command register 0x0502[15:8]:
   Write value for executed command back into EEPROM command register (0x502[10:8]).
   Write 0 to 0x502[13] (error acknowledge/command), if the access to NVRAM was successful, otherwise write 1. Write 0 to all other bits.
5) Read EEPROM busy status bit (0x502[15]) to check if command is completely processed.
6) Repeat from step 1) until 0x502[15]=0. Interrupt is gone if busy is gone.

11.2.4.2 EEPROM emulation example for 64 bit EEPROM data register size (0x0502[6]=1)

This chapter only applies for ESCs with 64 bit EEPROM data register size (0x0502[6]=1), i.e., ET1150.

The following steps have to be performed to process EEPROM commands by the µController.

1) µController waits for EEPROM command:
   a) Interrupt driven:
      i) Once: enable EEPROM emulation interrupt by setting AL event mask register 0x0204[5] to 1.
      ii) Wait for interrupt from ESC (if not already pending), and read AL event request register. 0x220[5]=1 indicates an EEPROM command is pending, which is processed as following.

   b) Poll EEPROM busy status bit: 0x502[15]=1 indicates an EEPROM command is pending.

2) Read EEPROM command register (0x502[10:8]).
3) Execute command:
   a) Read command (0x502[10:8] =“001”), or reload command (0x502[10:8] = “100”):
   Fetch 8 bytes of data from NVRAM according to word address value in EEPROM address register (0x504:0x0507), and place data in EEPROM data register (0x0508:0x050F).

   b) Write command (0x502[10:8] = “010”):
   Read 2 bytes of data from EEPROM data register (0x0508:0x0509), and store data in NVRAM according to word address value in EEPROM address register (0x504:0x0507).

   NOTE: It is allowed to cache write data for NVRAM for a short time, before actually writing them into the NVRAM. This enables combining several word writes into a single page write, e.g. for flash memory.

4) Acknowledge command by writing to EEPROM command register 0x0502[15:8]:
   Write value for executed command back into EEPROM command register (0x502[10:8]).
   Write 0 to 0x502[13] (error acknowledge/command), if the access to NVRAM was successful, otherwise write 1. Write 0 to all other bits.
5) Read EEPROM busy status bit (0x502[15]) to check if command is completely processed.
6) Repeat from step 1) until 0x502[15]=0. Interrupt is gone if busy is gone.

11.2.4.3 Caching EEPROM content for EEPROM emulation

Some NVRAM implementations (e.g., flash) have too long access times, which are not accepted by EtherCAT masters. In this case, it is recommended to cache the EEPROM content.

During startup, the cache is initially filled from the NVRAM content. During operation, master accesses are serviced by the cache. When the master writes into the cache, this data has to be transferred to the NVRAM shortly after write accesses from the master are finished. Typically, a master will write multiple words of the EEPROM. To reduce NVRAM erase and write cycles, a timeout of 100 ms (±20%) after the last write access should be used to trigger transfer to the NVRAM. The transfer should be completed within 1 second, and it must be completed within 5 seconds after the last write access. During NVRAM erase/write, further EEPROM commands by the EtherCAT master/PDI shall be accepted, i.e., the NVRAM erase/write operation is invisible to the master, as it occurs in the background. status information about words which have been updated by the master, and words written to NVRAM is probably required.

Since the EEPROM content is not required for EtherCAT frame processing, the risk of destructing the EEPROM content by powering down during transfer from cache to NVRAM is accepted. The EEPROM has to be written again in such cases. An EtherCAT slave must not be powered down/reset in the first 10 seconds after an EEPROM write access occurred.

NOTE: Do not rely on a certain write sequence of the master, this is master dependent and may change in the future.

11.3 SII EEPROM electrical interface (I²C)

The SII EEPROM interface is intended to be a point-to-point interface between the ESC and the I²C EEPROM. If other I²C masters are required to access the I²C bus, the ESC must be held in reset state (e.g. for in-circuit-programming of the EEPROM), otherwise access collisions are possible.

The SII EEPROM interface has the following signals[1]:

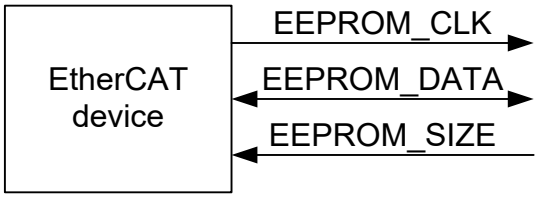


Figure 42: I²C EEPROM signals

Table 47: I²C EEPROM signals

Signal	Direction	Description
EEPROM_CLK	OUT	I²C clock
EEPROM_DATA	BIDIR	I²C data
EEPROM_SIZE	IN	EEPROM size configuration

Both EEPROM_CLK and EEPROM_DATA must have a pull-up resistor (4.7 kΩ recommended for ESCs), either integrated into the ESC or connected externally.

11.3.1 Addressing

EtherCAT and ESCs use word addressing when accessing the EEPROM, although the I²C interface actually uses byte addressing. The lowest address bit A[0] is added internally by the EEPROM interface controller of the ESCs. I.e., the EEPROM address register (0x0504:0x0507) reflects the physical EEPROM address bits A[18:1] (higher address bits are reserved/are zero).

SII EEPROM word 0 is typically located at I²C address 0, i.e., the I²C device address has to be set to 0. Some ESCs support an offset in the I²C base address.

11.3.2 EEPROM size

Depending on the EEPROM size, one out of two EEPROM algorithms has to be selected with the EEPROM_SIZE configuration signal. Smaller EEPROMs need only one address byte, larger ones need two address bytes:

Table 48: EEPROM size

EEPROM size	Address bytes	Max. I²C address bits	EEPROM_SIZE signal
1 Kbit – 16 Kbit	1	11	0
32 Kbit – 4 Mbit	2	19	1

[1] The availability of the EEPROM signals as well as their names depend on the specific ESC.

11.3.3 I²C access protocol

Each EEPROM access begins with a start condition and ends with a stop condition. Data is transferred byte-wise, and each byte is acknowledged by the recipient.

The start condition is a falling edge on EEPROM_DATA while EEPROM_CLK is high, the stop condition is a rising edge on EEPROM_DATA while EEPROM_CLK is high. In all other cases, EEPROM_DATA has to remain stable while EEPROM_CLK is high, as this indicates valid data. A byte transfer is acknowledged in an additional bit, which is driven low by the recipient of the byte transfer if it acknowledges the byte.

NOTE: If the EEPROM does not acknowledge an access (ack bit=high), it might be busy internally. Especially if the EERPOM interface is handled by a µController via the PDI, this situation may come up, because many µControllers can write to the EEPROM interface much faster than many EEPROMs can transfer the data from their input registers into their NVRAM.

The first byte of an I²C access is the control byte (Bit 7/MSB is transferred first):

Table 49: I²C control byte

Bit	Description
0	Read/write access: 0: Write 1: Read
[3:1]	Chip select bits/highest address bits
[7:4]	Control code: 1010

Depending on the access, either read data will follow or additional address bytes and write data. This is described in the following chapters.

The EEPROM has an internal byte pointer, which is incremented automatically after each data byte transfer.

For more details about the I²C protocol, refer to “The I²C-Bus Specification”, available from NXP (http://www.nxp.com, document number 39340011) and http://www.i2c-bus.org.

11.3.3.1 Write access

An EEPROM write access always writes one word (2 bytes) to the EEPROM. In this case, page boundaries are not relevant, because they will not be violated.

The ESC will perform the following steps for a write access to the EERPOM:

Table 50: I²C write access

Step	Description	Up to 16 Kbit	32 Kbit – 4 Mbit
1	Start condition		
2	Control byte (write)	A[10:8]	A[18:16]
3*	High address byte	Not preset	A[15:8]
4	Low address byte	A[7:0]	A[7:0]
5	Low data byte	D[7:0]	
6	High data byte	D[15:8]	
7	Stop condition		

* This step is only for EEPROMs larger than 16 Kbit.

11.3.3.2 Read access

An EEPROM read access reads 2 or 4 words (4 or 8 bytes, depending on device capabilities) from the EEPROM, the load or reload EEPROM access typically reads 8 words (16 bytes). The address wrap-around at the end of the EEPROM address space has to be taken into account by the application; the ESC has no knowledge about it.

The ESC will perform the following steps for a read access to the EERPOM. At first, the address is written to the EEPROM, then the data is read (N=3 or N=7):

Table 51: I²C read access

Step	Description	Up to 16 Kbit	32 Kbit – 4 Mbit
1	Start condition		
2	Control byte (write)	A[10:8]	A[18:16]
3*	High address byte	Not present	A[15:8]
4	Low address byte	A[7:0]	A[7:0]
5	Start condition		
6	Control byte (read)	A[10:8]	A[18:16]
7	Data byte 0	D0 [7:0]	
8	Data byte 1	D1 [7:0]	
…	…	…	
N+7	Data byte N	DN [7:0]	
N+8	Stop condition		

* This step is only for EEPROMs larger than 16 Kbit.

11.3.4 Timing specifications

Table 52: EEPROM timing characteristics

Parameter	Comment
t_{Clk}	EEPROM clock period
t_{Write}	Write access time (without errors)
t_{Read}	Read access time (without errors)
t_{Delay}	Time until configuration loading begins after Reset is gone

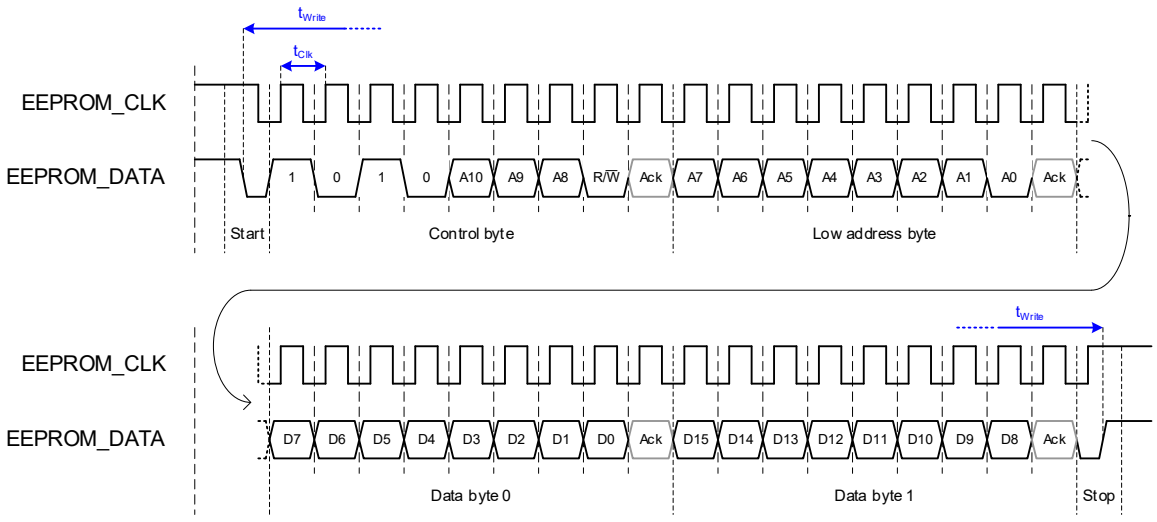


Figure 43: Write access (1 address byte, up to 16 Kbit EEPROMs)

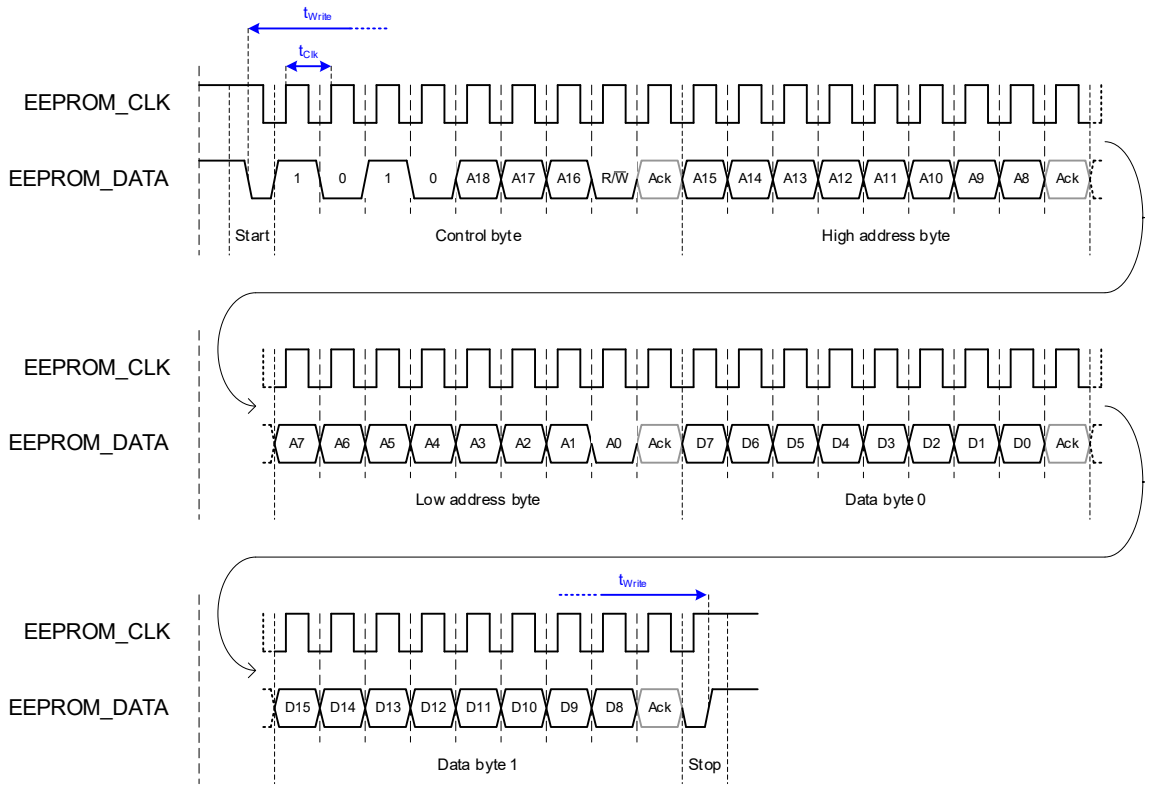


Figure 44: Write access (2 address bytes, 32 Kbit - 4 Mbit EEPROMs)

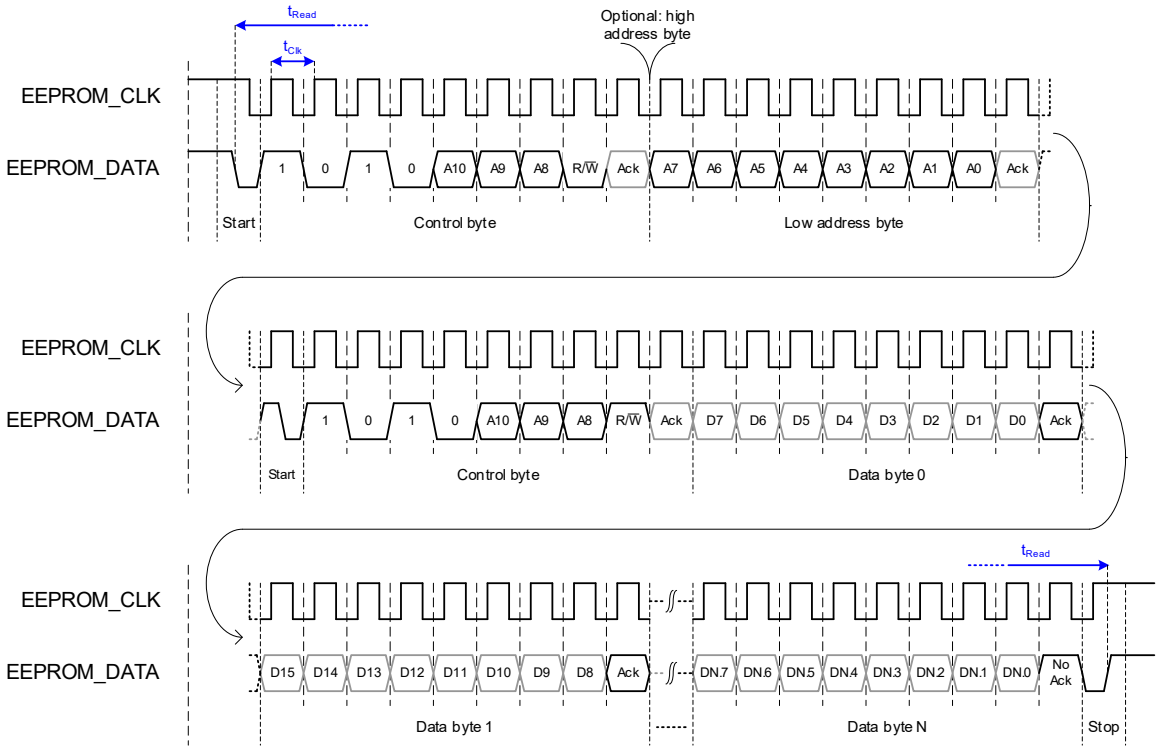


Figure 45: Read access (1 address byte, up to 16 Kbit EEPROMs)

12 Interrupts

ESCs support two types of interrupts: AL event requests, targeted at a µController, and ECAT event requests, targeted at the EtherCAT master. Additionally, the distributed clocks SyncSignals can be used as interrupts for a µController as well.

12.1 AL event request (PDI interrupt)

AL event requests can be signaled to a µController using the PDI interrupt request signal (IRQ/SPI_IRQ, etc.). For IRQ generation, the AL event request register (0x0220:0x0223) is combined with the AL event mask register (0x0204:0x0207) using a logical AND operation, then all resulting bits are combined (logical OR) into one interrupt signal. The output driver characteristics of the IRQ signal are configurable using the SYNC/LATCH PDI configuration register (0x0151). The AL event mask register allows for selecting the interrupts which are relevant for the µController and handled by the application.

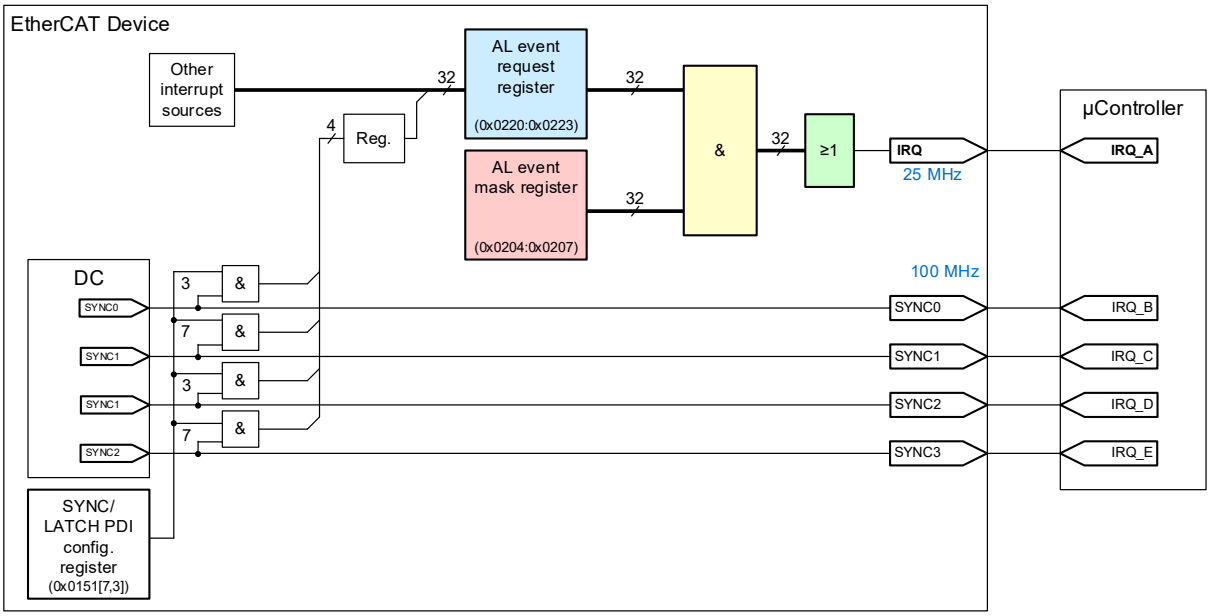


Figure 46: PDI interrupt masking and interrupt signals

The DC SyncSignals can be used for interrupt generation in two ways:

- The DC SYNC signals are mapped into the AL event request register (configured with SYNC/LATCH PDI configuration register 0x0151.3/7). In this case, all interrupts from the ESC to the µController are combined into one IRQ signal. The IRQ signal has a jitter of ~40 ns.
- The DC SyncSignals are directly connected to µController interrupt inputs. The µController can react on DC SyncSignal interrupts faster (without reading AL request register), but it needs more interrupt inputs. The jitter of the SyncSignals is ~12 ns.

Registers used for AL event requests are described in Table 53:

Table 53: Registers for AL event request configuration

Register address	Name	Description
0x0150	PDI0 configuration	PDI0 IRQ driver characteristics, depending on PDI0
0x0190	PDI1 configuration	PDI1 IRQ driver characteristics, depending on PDI1
0x0151	SYNC/LATCH PDI configuration	Mapping DC SyncSignals to interrupts
0x0204:0x0207	AL event mask	Mask register
0x0220:0x0223	AL event request	Pending Interrupts
0x0804 + N*8	SyncManager control	Mapping SyncManager interrupts

NOTE: Some of these registers are set via SII EEPROM/IP core configuration, or they are not available in specific ESCs. Refer to section II for details.

12.2 ECAT event request (ECAT interrupt)

ECAT event requests are used to inform the EtherCAT master of slave events. ECAT events make use of the IRQ field inside EtherCAT datagrams. The ECAT event request register (0x0210:0x0211) is combined with the ECAT event mask register (0x0200:0x0201) using a logical AND operation. The resulting interrupt bits are combined with the incoming ECAT IRQ field using a logical OR operation, and written into the outgoing ECAT IRQ field. The ECAT event mask register allows for selecting the interrupts which are relevant for the EtherCAT master and handled by the master application.

NOTE: The master cannot distinguish which slave (or even more than one) was the origin of an interrupt.

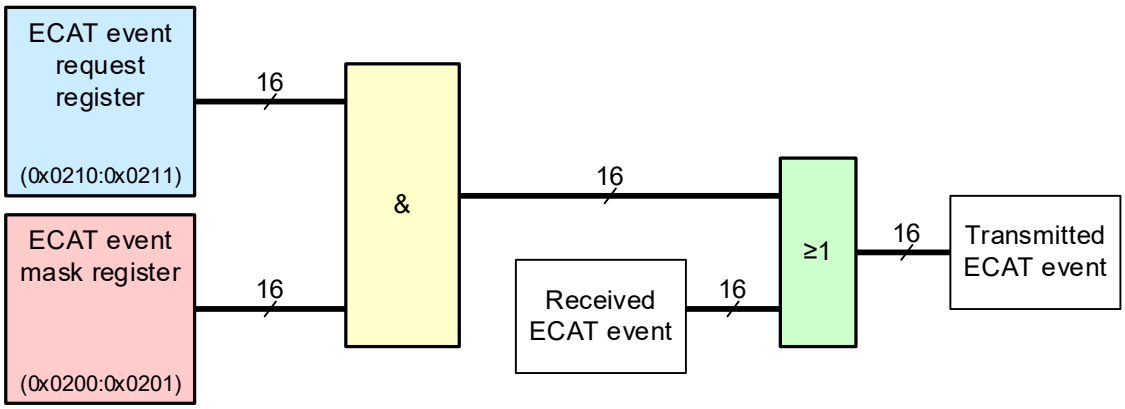


Figure 47: ECAT interrupt masking

Registers used for ECAT Interrupts are described in Table 54:

Table 54: Registers for ECAT event request configuration

Register address	Name	Description
0x0200:0x0201	ECAT event mask	Mask register
0x0210:0x0211	ECAT event request	Pending interrupts
0x0804 + N*8	SyncManager control	Mapping SyncManager interrupts

NOTE: Some of these registers are not available in specific ESCs. Refer to section II for details.

12.3 Clearing interrupts accidentally

Event request registers and register actions which clear interrupts are intended to be accessed independently, i.e., with separate EtherCAT frames or separate PDI accesses. Otherwise it may happen that interrupts and/or data are missed.

Examples:

- Using SPI to read a SyncManager buffer: polling SyncManager buffers and interrupts delivered at the beginning of each SPI access in the same access can lead to missed interrupts/data. Fault scenario: the interrupt is not pending while the interrupts delivered at the beginning of the access are sampled. The µController gets the information “no interrupt”, but it continues reading the SyncManager buffer because the read command cannot be stopped without causing a PDI error. If the SyncManager interrupt occurs in the time windows between interrupt sampling and buffer reading, new buffer data will be delivered and the interrupt is acknowledged. As a consequence, the µController application will ignore the new data because no interrupt was set.
  Solution: read the SyncManager buffer only if the IRQ signal indicates a pending interrupt or if a **preceding** access indicates pending interrupts.
- Using a single ECAT frame to read DC Latch[3:0] status and latch time registers: the status registers may indicate no event, but if the event occurs in the time window between reading status and time registers, the new latch time will be delivered and the corresponding interrupt is cleared directly. The master gets the information “no interrupt”, but new latch times, so it will ignore the time values and the interrupt/data is missed.
  Solution: read DC latch time registers only if an ECAT event was indicated in a previous frame or if the DC latch status registers were polled in a **previous** frame.

13 Watchdogs

The ESCs support up to three internal watchdogs (WD), a process data watchdog used for monitoring process data accesses, and a PDI watchdog monitoring PDI activity for each of the two PDIs.

The timeout for these watchdogs can be configured individually, but they share a single watchdog divider (WD_DIV, register 0x0400:0x0401). The watchdog timeout is calculated from the watchdog Divider settings multiplied with the watchdog time settings for PDI0 (WD_PDI0, register 0x0410:0x0411), PDI1 (WD_PDI1, register 0x0412:0x0413), or process data (WD_PD, register 0x0420:0x0421). Base time unit is 40 ns. The watchdog timeout jitters, the jitter depends on the watchdog Divider settings. I.e., selecting smaller watchdog Divider settings results in smaller jitter.

The following equations are used for a quick estimation of the watchdog timeout (they are not exact in terms of nanoseconds):

t_{WD_Div} = (WD_DIV + 2) * 40ns

t_{WD_PDI} = [t_{WD_Div} * WD_PDI ; t_{WD_Div} * WD_PDI + t_{WD_Div}]

t_{WD_PD} = [t_{WD_Div} * WD_PD ; t_{WD_Div} * WD_PD + t_{WD_Div}]

Registers used for watchdogs are described in Table 55:

Table 55: Registers for watchdogs

Register address	Name	Description
0x0110[1]	ESC DL status	Combined status PDI watchdog
0x0181[1:0]	ESC configuration B0	Combined status configuration
0x0400:0x0401	Watchdog divider	Watchdog divider (WD_DIV)
0x0410:0x0411	Watchdog time PDI0	Watchdog time PDI0 (WD_PDI0)
0x0412:0x0413	Watchdog time PDI1	Watchdog time PDI1 (WD_PDI1)
0x0420:0x0421	Watchdog time process data	Watchdog time process data (WD_PD)
0x0440:0x0441	Watchdog status process data	Status process data watchdog
0x0442	Watchdog counter process data	Watchdog expiration counter process data
0x0443	Watchdog counter PDI0	Watchdog expiration counter PDI0
0x0444	Watchdog counter PDI1	Watchdog expiration counter PDI1
0x0448	Watchdog status PDI	Status PDI0 + PDI1 watchdog
0x0804 +N*8	SyncManager control	Watchdog trigger enable

NOTE: Some of these registers are not available in specific ESCs. Refer to section II for details.

13.1 Process data watchdog

The process data watchdog is rewound (triggered) by a write access to a SyncManager buffer area if the SyncManager is configured to generate a watchdog trigger signal (SyncManager control register 0x0804[6] for SyncManager 0, etc.). The watchdog trigger signal is generated after the buffer was completely and successfully written (similar to the interrupt write of a SyncManager).

The process data watchdog can be disabled by setting the process data watchdog time to 0.

A timeout of the process data watchdog has these consequences:

- Watchdog status register for process data (0x0440[0]) reflects the watchdog status.
- The digital I/O PDI takes back digital output data, either by not driving the signals anymore or by driving them low (ESC and configuration dependent).
- The watchdog counter process data (0x0442) is incremented.

13.2 PDI watchdogs

Each PDI watchdog is rewound (triggered) by any correct read or write access by the PDI. Each PDI watchdog can be disabled by setting the PDI watchdog time to 0.

ESC DL status register (0x0110[1]) reflects a combined watchdog status, which is derived from WD_PDI0 and WD_PDI1, based on ESC configuration B0 register [1:0]. Possible combinations are:

- PDI0 watchdog status only
- PDI1 watchdog status only
- PDI0 or PDI1 watchdog status
- PDI0 and PDI1 watchdog status

A timeout of a PDI watchdog has these consequences:

- ESC DL status register (0x0110[1]) reflects the combined watchdog status. This combined status can be mapped to the ECAT interrupt to inform the master.
- The watchdog counter PDI0 (0x0443) or PDI1 (0x0444) is incremented.

NOTE: The digital I/O PDI0 only triggers the PDI0 watchdog upon input events.

14 Error counters

The ESCs have numerous error counters which help in detecting and locating errors. All error counters are saturated at 0xFF (no wrap-around) and they are cleared individually or group-wise by writing any value to them.

Table 56: Error counter overview

Error counter	Register	Description
Port error Counters		Errors counted at the auto-forwarder (per port):
Invalid frame counter	0x0300/2/4/6	Invalid frame initially detected (includes RX errors)
RX error counter	0x0301/3/5/7	Physical layer RX errors (inside/outside frame, open ports): Ethernet: RX_ER EBUS: Manchester violations
Extended RX error counter	0x0314:0x0317	Physical layer RX errors (inside/outside frame, open/closed ports), link failure events
Forwarded RX error counter	0x0308:0x030B	Invalid frame with marking from previous ESC detected (per port)
ECAT processing unit error counter	0x030C	Invalid frame passing the EtherCAT processing unit (additional checks by processing unit)
PDI0 error counter	0x030D	Physical errors detected by the PDI0
PDI0 error code	0x030E:0x030F	Error code for PDI0 error counter
Lost link counter	0x0310:0x0313	Link lost events (per port), counts only if port is in auto mode
RX error code	0x0320:0x0327	Link error code or RX error code, associated with Extended RX error counter
PDI1 error counter	0x0340	Physical errors detected by the PDI1
PDI1 error code	0x0341:0x0342	Error code for PDI1 error counter
Watchdog counter process data	0x0442	Watchdog timeout events
Watchdog counter PDI0	0x0443	Watchdog timeout events
Watchdog counter PDI1	0x0444	Watchdog timeout events

NOTE: Some errors will be counted in multiple registers. E.g., a physical layer RX error received at port 0 is counted in registers 0x0300, 0x0301, 0x030C, and 0x0314. A forwarded error received at port 0 is counted in registers 0x0308, 0x030C, and 0x0314.
Some of these registers are not available in specific ESCs. Refer to section II for details.

14.1 Frame error detection

EtherCAT frame error detection takes place at three functional blocks, at the physical layer (device), inside the auto-forwarder and inside the EtherCAT processing unit. The following errors are detected and counted in these registers:

Table 57: Errors and corresponding error counters

Error type	RX error counter Registers 0x301/3/5/7 0x314/5/6/7	Invalid frame counter Registers 0x300/2/4/6	Forwarded error counter Register 0x308/9/A/B	ECAT processing unit error counter Register 0x30C
Counter increments by one for	every received symbol with RX_ER	every frame with error	every frame with error	every frame with error
Physical layer error MII/RMII: RX_ER event EBUS: EBUS/manchester code violations (refer to chapter 6.5)	yes	if inside frame only	no	EPU inbound port only
FIFO overrun/underrun	no	yes	no	EPU inbound port only
Frames without Ethernet SOF	no	yes	no	EPU inbound port only
Too long frames (> ~ 2000 byte)	no	yes	no	EPU inbound port only
CRC error without odd nibble	no	yes	no	EPU inbound port only
CRC error with odd nibble	no	no	yes	EPU inbound port only
Too short frames (< 64 byte)	no	no	no	EPU inbound port only
EtherCAT frame length errors (e.g., frame ends although more header/data bytes are expected, padding > 64 byte)	no	no	no	EPU inbound port only
Non-EtherCAT frames if register 0x0100[0]=1	no	no	no	EPU inbound port only
Circulating bit=1 and port 0 automatically closed	no	no	no	EPU inbound port only

Any of the above errors will have these consequences:

- The frame transmission is aborted (a frame with an RX error at the beginning is truncated). The CRC of the transmitted data is modified (or appended) so that it becomes bad. A special marking for forwarded errors is added.
- The EtherCAT processing unit discards register operations, e.g., write operations to registers, SyncManager buffer changes, etc.. RAM areas will be written, because they do not have a shadow buffer for write data like registers.
- Error counters are increased.

Refer to application note “Frequently asked questions and troubleshooting” for more advice regarding error counter interpretation. The application note is available at the Beckhoff website (http://www.beckhoff.com).

14.2 Errors and forwarded errors

The ESCs distinguish errors initially detected by an ESC and forwarded errors detected by a previous ESC. This is useful for error location when interpreting the RX error/forwarded RX error counters.

The first device detecting an error (e.g., a CRC error or an RX error of the physical layer), will discard register operations and count a port error (0x0300-0x0307). The outgoing frame gets a special marking, consisting of one extra nibble added after the (invalid) CRC.

A device receiving a frame with a CRC error and an additional nibble will also discard register operations, but it will count one forwarded RX error instead of a normal port error.

NOTE: A forwarded error is sometimes called “green error”, the initial error is sometimes called “red error”. A physical layer RX error is always a “red error”, because it could not have been forwarded.

15 LED signals

EtherCAT slave controllers support different LEDs regarding link state and AL status. For details about EtherCAT indicators refer to the ETG.1300 EtherCAT indicator and labeling specification, available from the download section of the EtherCAT technology group website (http://www.ethercat.org).

15.1 RUN LED

The AL status is displayed with the RUN LED (green). The RUN output of an ESC is controlled by the AL status register (0x0130) and supports the following states, which are automatically translated into blink codes:

Table 58: RUN LED state indication

RUN LED	Description
Off	The device is in state INIT
Blinking (slow)	The device is in state PRE-OPERATIONAL
Single flash	The device is in state SAFE-OPERATIONAL
On	The device is in state OPERATIONAL
Flickering (fast)	The device is in state BOOTSTRAP or loading the SII EEPROM

The following RUN LED states can be automatically generated by an ESC, without interaction of an application controller. The support of individual RUN LED states is ESC specific.

Table 59: Automatic ESC RUN LED state indication

RUN LED	Description	Examples
Flickering (fast)	SII EEPROM loading	During first loading

15.1.1 RUN LED override

Some ESCs support optional RUN LED outputs by overriding the state indication of the RUN LED. The output can be set by master or local application. This can be used e.g. for locating a specific slave by forcing the RUN LED to indicate a triple flash (device identification).

The RUN LED override is disabled automatically by a following ESM state change.

Table 60: Registers for RUN LED control

Register address	Name	Description
0x0130	AL status register	After a state change, the RUN LED indicates the current ESM state.
0x0138	RUN LED override	Direct control of the RUN LED from ECAT and/or PDI

15.2 ERR LED

The ERR LED indicates local errors and application errors. It is either connected to the application controller or to the ESC (optional ESC feature). If it is connected to the ESC, some errors are automatically indicated by the ESC, other error states are detected by the application controller and indicated by writing to the ERR LED override register 0x0139.

The following ERR LED states can be automatically generated by an ESC, without interaction of an application controller. The support of individual error states is ESC specific.

Table 61: Automatic ESC ERR LED state indication

ERR LED	Description	Examples
Off	No error	
Flickering (fast)	SII EEPROM loading error	no SII EEPROM device, SII CRC error
Blinking (slow)	Invalid hardware configuration	ESC pin sharing violation
Single flash	AL status register error indication bit 0x0130[4] is set while device emulation is disabled	Local application controller sets error indication bit. NOTE: E.g., if the µController makes a state change with error indication bit 0x0130[4] set after a process data watchdog timeout, it has to manually set the ERR LED to Double Flash again (otherwise the ESC would generate a single flash due to the error indication bit automatically).
Double flash	Process data watchdog timeout (edge detected) while device is OPERATIONAL	master is disconnected/not sending process data any more
On	PDI watchdog timeout (edge detected) or build-in self-test error	local application controller failure or self-test failure

NOTE: Do not confuse the application ERR LED with the port receive error LEDs (PERR(x)) supported by some ESCs.

15.2.1 ERR LED override

If the ESC supports the ERR LED, the local application (and the master) is able to control the ERR LED via the ERR LED override register.

The ERR LED override is disabled automatically if an error is detected by the ESC which is associated with an ERR LED blink code.

Table 62: Registers for ERR LED control

Register address	Name	Description
0x0139	ERR LED override	Direct control of the ERR LED from ECAT and/or PDI

15.3 STATE LED and STATE_RUN LED signal

The STATE LED is a bicolor-LED combining RUN and ERR LED. Since the RUN LED part of the STATE LED must be turned off while the ERR LED part is active, the RUN and ERR LED signals cannot be simply combined to drive the bicolor LED. Some ESCs support a STATE_RUN signal, which is turned off while ERR LED is on, so STATE_RUN and ERR signals can be used to drive the bicolor STATE LED. Otherwise the logical combination “RUN and not(ERR)” has to be used to control the RUN LED part of the STATE LED.

15.4 LINKACT LED

The link/activity state of each port is displayed with the LINKACT LED (green).

Table 63: LINKACT LED states

LINKACT LED	Description
Off	No link
Blinking	Link and activity
On	Link without activity

It is highly recommended to use the LINKACT LED signals of the ESCs for driving visible LEDs, instead of the link/activity LED signals of the PHY, because the ESC signals reflect the actual link/activity state of the device – not only the state of the PHYs –, and the ESC signals adhere to the ETG.1300 EtherCAT indicator and labeling specification.

The LINKACT LED indicates the link status after evaluation of the enhanced link detection plus MI link detection and configuration, it corresponds with the ESC DL status register (0x0110[15,13,11,9] – Communication established.

15.5 Port error LED (PERR)

Some ESCs support port receive error indicators PERR(x), which display physical layer RX errors. The PERR(x) LEDs are not part of the ETG.1300 EtherCAT indicator and labeling specification. They are only intended for testing and debugging. The PERR(x) LEDs flash once if a physical layer RX error occurs.

16 Process data interface (PDI)

A process data interface (PDI) realizes the connection between slave application and ESC. Several types of PDIs are defined, e.g., serial and parallel µController interfaces and digital I/O interfaces. Table 64 gives an overview of the available PDI types for each ESC.

Details on individual PDI functionality can be found in section III of the hardware data sheet for a specific ESC.

Table 64: Available PDIs depending on ESC

PDI control	PDI name	ESC20	IP core	ET1100	ET1150	ET1200
0x00	Interface deactivated	x	x	x	x	x
0x04	Digital I/O		x	x	x	x
0x05	SPI slave (number of MISO/MOSI signals)	1	1, 2, 4, 8	1	1, 2, 4, 8	1
0x07	EtherCAT bridge (port 3)					x
0x08	16/64 Bit asynchronous µController interface	16	16	16	16	
0x09	8/32 Bit asynchronous µController interface	8	8/32	8	8	
0x0A	16/64 Bit synchronous µController interface			16	16	
0x0B	8/32 Bit synchronous µController interface			8	8	
0x0C	16/64 Bit multiplexed asynchronous µController interface				16	
0x0D	8/32 Bit multiplexed asynchronous µController interface				8/32	
0x0E	16/64 Bit multiplexed synchronous µController interface				16	
0x0F	8/32 Bit multiplexed synchronous µController interface					
0x10	32 digital input/0 digital output	x				
0x11	24 digital input/8 digital output	x				
0x12	16 digital input/16 digital output	x				
0x13	8 digital input/24 digital output	x				
0x14	0 digital input/32 digital output	x				
0x15	SPI master				x	
0x80	On-chip bus		x			
Others	Reserved					

NOTE: On-Chip bus: different kinds of on-chip buses are supported by the EtherCAT IP cores for FPGAs, depending on the target FPGA vendor.

16.1 PDI selection and configuration

Typically, the PDI selection and configuration is part of the ESC configuration area of the SII EEPROM.

Some ESCs (e.g. IP core) have the PDI selected and configured at power-on time. In this case, the ESC configuration area should reflect the actual settings, although they are not evaluated by the ESC itself.

For the other ESCs, the PDI becomes active after the SII EEPROM is successfully loaded. All PDI pins are inactive (high impedance) until then (as well as the DC Sync/Latch signals). Some ESCs and PDIs provide an EEPROM_Loaded signal, which indicates that the EEPROM is successfully loaded and the PDI can be used. Attach a pull-down resistor to the EEPROM_Loaded pin, because it is also not driven (high impedance) until the EEPROM is successfully loaded. The PDI of an IP core is active after reset is released, which enables e.g. EEPROM emulation by a µController.

The optional PDI information registers 0x014E/0x018E can be used to determine activation status of the PDIs.

Take care of digital output signals and DC SyncSignals while the EEPROM is not loaded to achieve proper output behavior.

Table 65: Registers used for PDI configuration and PDI status

Register address	Name	Description
0x0110[0]	ESC DL control	PDI operational/EEPROM loaded
0x0140	PDI0 control	PDI0 selection
0x014E	PDI0 information	PDI0 activation status
0x0150 0x0152:0x0153	PDI0 configuration	PDI0 configuration
0x0180	PDI1 control	PDI1 selection
0x018E	PDI1 information	PDI0 activation status
0x0190 0x0192:0x0193	PDI1 configuration	PDI1 configuration

16.2 Multiple PDI interfaces

Some ESCs support two PDI interfaces. Both PDIs can access all registers and RAM, without limitations, they have to operate cooperatively. If two µControllers are connected using the two PDIs, it is up to the software stacks which µController performs certain tasks, e.g., handling of the EtherCAT state machine, and process data exchange. Internally, both PDIs share the same internal interface, and read/write accesses are multiplexed to registers and RAM - the internal bandwidth is shared between both PDIs.

Some features are available for each PDI individually:

- PDI control and configuration registers
- PDI watchdog, watchdog status, and PDI watchdog counter
- PDI error counter and PDI error code

The PDI0 and PDI1 watchdogs are combined to feed the ESC DL status register (0x0110[1]), using a configurable function (ESC configuration B0[1:0]).

Table 66: Registers used for two PDIs

Register address	Name	Description
0x0110[1]	ESC DL status	Combined PDI watchdog status
0x014E	PDI0 information	PDI0 activation status
0x018E	PDI1 information	PDI0 activation status
0x0181	ESC configuration B0	PDI watchdog selection for 0x0110[1]
0x0182:0x0183	ESC configuration B5	Private RAM configuration

16.2.1 Private RAM

If µControllers connected via two PDIs need to exchange data for cooperation, it is possible to use the ESC process data RAM for this purpose. To avoid disturbance by any EtherCAT master, a certain area of the process data RAM can be reserved for the PDIs, the “private RAM”.

The private RAM area has a configurable number of Kbytes at the end of the available process data RAM (ESC configuration B5[5:0]). This private RAM is not accessible by the EtherCAT master, the indicated RAM size in register 0x0006 is reduced accordingly. Both PDIs have unlimited access to this RAM.

16.3 PDI function acknowledge by write

Some ESC functions are triggered by writing or reading individual byte addresses, e.g., SyncManager buffer change or AL event request acknowledge. With an increasing data bus width of the µControllers, this can lead to restrictions or even problems.

Since most µControllers are using byte enable signals for write accesses, there is no restriction for functions which are triggered by writes. But many µControllers are not using the byte enable signals for read accesses, they expect to get a whole data bus width of read data. Reading individual bytes is not possible. This can lead to problems especially by accidentally reading byte addresses which trigger certain ESC functions. Consider a SyncManager buffer area from 0x1000-0x1005. A 32 bit µController application might read the buffer byte-wise. The first access to 0x1000 would open the buffer, and it would also read 0x1001-0x1003. The second access would read 0x1001, and also 0x1000/0x1002-0x1003. The problem occurs when address 0x1004 is to be read, because this would also read 0x1005. The data of 0x1005 is discarded, but the buffer is closed. When the µC reads 0x1005, it will always get 0 – the data seems to be corrupted. A similar issue occurs for DC SyncSignal acknowledging (registers 0x098E and 0x098F). A 32 bit µController would always acknowledge SYNC0 and SYNC1 at the same time, it is not possible to acknowledge them separately.

This problem can be overcome by enabling PDI function acknowledge by write. In this mode, all functions which are originally triggered by read access are now triggered by corresponding write accesses – which use byte enables and thus can be restricted to certain bytes.

This feature is enabled by IP core configuration. The current status has to be checked by the µController application in PDI information register 0x014E[0]/0x018E[0, before using this function.

This feature affects reading of SyncManager buffers and reading of certain registers from PDI side. There is no change to the EtherCAT master side at all. Refer to chapter 8 for SyncManager behavior.

Some ESCs allow independent activation of this feature for SyncManager buffers, and registers.

Table 67: Registers used PDI register function acknowledge by write

Register address	Name	Description
0x014E	PDI0 information	PDI0 function acknowledge activation status
0x018E	PDI1 information	PDI1 function acknowledge activation status

NOTE: Current ESCs do not support different configurations for two PDIs.

The following registers are affected by the PDI function acknowledge by write feature:

Table 68: Functions/registers affected by PDI function acknowledge by write

Address	Name	Trigger function
SyncManager function acknowledge by write		
any	SyncManager buffer end address	Read SyncManager buffer, then write to buffer end address to acknowledge buffer reading.
Register function acknowledge by write		
0x0120:0x0121	AL control	Read 0x0120:0x0121 after AL control changes, then write to 0x0120 to acknowledge reading.
0x0440	Watchdog status process Data	Read 0x0440, then write to 0x0440 to clear AL event request 0x0220[6]
0x0806+y*16	SyncManager activate	Read 0x0806+y*16, then write to 0x0806 (SyncManager 0) only to clear AL event request 0x0220[4] for all SyncManagers
0x098E	SYNC0 status	Read 0x098E, then write to 0x098E to acknowledge DC Sync[0] status 0x098E[0]
0x098F	SYNC1 status	Read 0x098E, then write to 0x098E to acknowledge DC Sync[1] status 0x098F[0]
0x098C	SYNC2 status	Read 0x098C, then write to 0x098E to acknowledge DC Sync[2] status 0x098C[0]
0x098D	SYNC3 status	Read 0x098D, then write to 0x098E to acknowledge DC Sync[3] status 0x098D[0]
0x09B0:0x09B7	Latch0 time positive edge	Read 0x09B0:0x09B7, then write to 0x09B0 to clear DC Latch[0] status 0x09AE[0]
0x09B8:0x09BF	Latch0 time negative edge	Read 0x09B8:0x09BF, then write to 0x09B8 to clear DC Latch[0] status 0x09AE[1]
0x09C0:0x09C7	Latch1 time positive edge	Read 0x09C0:0x09C7, then write to 0x09C0 to clear DC Latch[1] status 0x09AF[0]
0x09C8:0x09CF	Latch1 time negative edge	Read 0x09C8:0x09CF, then write to 0x09C8 to clear DC Latch[1] status 0x09AF[1]
0x09D0:0x09D7	Latch2 time positive edge	Read 0x09D0:0x09D7, then write to 0x09D0 to clear DC Latch[2] status 0x09AC[0]
0x09D8:0x09DF	Latch2 time negative edge	Read 0x09D8:0x09DF, then write to 0x09D8 to clear DC Latch[2] status 0x09AC[1]
0x09E0:0x09E7	Latch3 time positive edge	Read 0x09E0:0x09E7, then write to 0x09E0 to clear DC Latch[3] status 0x09AD[0]
0x09E8:0x09EF	Latch4 time negative edge	Read 0x09E8:0x09EF, then write to 0x09E8 to clear DC Latch[3] status 0x09AD[1]

16.4 General purpose I/O

Some ESCs support general purpose inputs, outputs, or both. General purpose I/O is much different from digital I/O, because of missing consistency and security features, and no bit-write support.

16.4.1 General purpose inputs

The general purpose inputs are directly mapped into the General purpose Input registers. Consistency of the general purpose inputs is not provided.

16.4.2 General purpose output

The general purpose output signals reflect the values of the general purpose output register without watchdog protection. The general purpose output register can be written both by ECAT and PDI. The general purpose outputs are intended e.g. for application specific LED outputs. General purpose outputs are updated at the end of an EtherCAT frame or at the end of a PDI access.

Consistency of the general purpose outputs is not provided, and they are also not watchdog-secured. Additionally, they do not support bit-wise modification with FMMUs.

16.4.3 Bidirectional general purpose output

Some ESCs support bidirectional general purpose I/O, using some general purpose output register values as driver enable, and other general purpose output register values as output data. Inputs are mapped to general purpose input registers.

17 Additional information

17.1 ESC clock source

The initial accuracy of the ESC clock sources has to be 25 ppm or better (at room temperature). This enables FIFO size reduction, i.e., forwarding delay reduction, and supports fast DC locking.

17.2 Power-on sequence

The power-on sequence of ESCs looks like this:

Table 69: ESC power-on sequence

No.	Step	Result
1	Power-on	Voltages reach proper levels ASICs only: power-on values are sampled
2	FPGA only: Loading FPGA configuration	FPGA loads its hardware configuration
3	PLL locks	Clocks are generated properly
4	Some ESCs only: Build-in self-test is performed	ESC is ok
5	Some ESCs only: Loading OTP configuration	OTP configuration valid
6	Release RESET	ESC operation begins. Process memory is not accessible until the SII EEPROM is loaded, as well as any function depending on ESC configuration data. IP core: PDI is operational; others: PDI is not operational until EEPROM is loaded.
7a*	Links are established	EtherCAT communication begins, master can access ESC registers
7b*	Loading ESC EEPROM	Only upon successful EEPROM loading: • ESC configuration registers initialized • PDI is activated (not IP core: active after RESET) • PDI operation begins • Register bit 0x0110[0] turns to 1 • Process data RAM becomes accessible • Some PDIs: EEPROM_Loaded signal is driven high • ESC is in Init state
8	Example: master proceeds to Operational state	ESC proceeds to Operational state

* Steps 5 and 6 are executed in parallel.

NOTE: Usually, the PDI signals are not driven until the ESC EEPROM is loaded successfully, especially the EEPROM_Loaded signal is not driven and needs a pull-down resistor if it is used.

17.3 Optional features, optional registers

ESCs have different features, which an EtherCAT master has to know about. Some features are indicated directly by registers:

Table 70: Features indicated by registers

Register address	Feature	Description
0x0004	FMMUs supported	Existence of FMMUs and FMMU registers in 0x0600:0x06FF
0x0005	SyncManagers supported	Existence of SyncManagers and SyncManager registers in 0x0800:0x087F
0x0006	RAM size	Size of process data RAM from 0x1000 to 0xFFFF
0x0007	Port descriptor	Type of communication port 0-3
0x0008[0]	FMMU operation	Bitwise FMMU mapping
0x0008[2]	Distributed clocks	Existence of distributed clocks and DC registers in 0x0900-0x09FF (DC functions are optional)
0x0008[3]	Distributed clocks width	Existence of 64 bit extensions of some DC registers
0x0008[4]	Low jitter EBUS	Support of low jitter EBUS and existence of 0x0100[19]
0x0008[5]	Enhanced link detection EBUS	Support of enhanced link detection for EBUS and existence of 0x0110[2]
0x0008[6]	Enhanced link detection MII	Support of enhanced link detection for MII/RMII/RGMII and existence of 0x0110[2]
0x0008[7]	Separate handling of FCS Errors	Support of forwarded errors and existence of registers 0x0308:0x030B (if port is available)
0x0008[8]	Enhanced DC activation	Existence of 0x0981[7:3] and 0x0984
0x0008[9]	EtherCAT LRW command	Support of LRW command
0x0008[10]	EtherCAT BRW/APRW/FPRW command	Support of BRW/APRW/FPRW command
0x0008[11]	Fixed FMMU/SyncManager configuration	Support of flexible FMMU/SyncManager configuration
0x0008[12]	SyncManager Sequential Mode	Support of SyncManager sequential mode and 0x0804[7] and 0x0805[2] for each SyncManager
0x0502[5]	EEPROM emulation	I2C device or µController serving SII EEPROM
0x0502[6]	Number of EEPROM read bytes	Support of 0x05C:0x050F
0x0502[7]	EEPROM algorithm	Max. I²C EEPROM size
0x0510[1]	MII control by PDI	Support of MII control by PDI and existence of 0x0516:0x0517
0x0510[2]	MI link detection	General support of MI link detection and existence of 0x0518:0x051B (if port is available)
0x0510[12]	MI clause 45 support	Support for clause 45 commands
0x0518[6]	MI link detection	Port specific support of MI link detection

Other optional features are not indicated directly, but the EtherCAT master is still possible to find out the existence of a feature: The basic principle is to read the first byte of the register, and evaluate the working counter (WKC). If it increments, the feature is present.

The read values of unsupported/reserved registers or registers bits must be ignored. Registers which are not supported, should not be written. If absolutely necessary, the write value must be 0.

17.4 Write protection

Some ESCs are capable of register write protection or entire ESC write protection.

Registers used for write protection are described in Table 71:

Table 71: Registers for write protection

Register address	Name	Description
0x0020	Register write enable	Temporarily release register write protection
0x0021	Register write protection	Activate register write protection
0x0030	ESC write enable	Temporarily release ESC write protection
0x0031	ESC write protection	Activate ESC write protection

NOTE: Some of these registers are not available in specific ESCs. Refer to section II for details.

17.4.1 Register write protection

With register write protection, only the register area 0x0000 to 0x0F7F is write protected (except for registers 0x0020 and 0x0030). User RAM (0x0F80:0x0FFF) and process data RAM (0x1000:0xFFFF) are not protected.

If register write protection is enabled (register 0x0021[0]=1), the register write enable bit (0x0020[0]) has to be set in the same frame before any register write operations. This is also true for disabling the register write protection. Otherwise, write operations to registers are discarded.

17.4.2 ESC write protection

ESC write protection disables write operations to any memory location (except for registers 0x0020 and 0x0030).

If ESC write protection is enabled (register 0x0031[0]=1), the ESC write enable bit (0x0030[0]) has to be set in the same frame before any write operations. This is also true for disabling the ESC write protection as well as the register write protection. Otherwise, write operations are discarded.

NOTE: If both register write protection and ESC write protection are enabled (not recommended), both enable bits have to be set before the write operations are allowed.

17.5 ESC reset

Some ESCs are capable of issuing a hardware reset by the EtherCAT master or even via the PDI. A special sequence of three independent and consecutive frames/commands has to be sent to the slave (Reset register ECAT 0x0040, or PDI 0x0041). Afterwards, the slave is reset.

NOTE: It is likely that the last frame of the sequence will not return to the master (depending on the topology), because the links to and from the slave which is reset will go down.

17.6 Register and RAM access options

There are four different ways to access an ESC register or the ESC RAM from the EtherCAT master:

- Directly
- SyncManager (the target address is covered by a SyncManager)
- FMMU (a logical command and an FMMU are used to access a physical address)
- SyncManager and FMMU (a logical command and an FMMU are used to access a physical address, which is covered by a SyncManager)

NOTE: For debugging, it could be beneficial to disable a SyncManager temporarily. The buffer content can be read, while consistency is sacrificed.

18 Troubleshooting and FAQ

18.1 ESC clock source accuracy: Is 25 ppm necessary?

Since standard PCs can be used as a master, the clock source deviation of the master's Ethernet PHY can be up to 100ppm (according to IEEE 802.3). This is tolerated by the ESCs if the receive FIFO size is remaining in default configuration. Nevertheless, EtherCAT requires 25 ppm clock accuracy for the EtherCAT slaves in order to enable FIFO size reduction.

FIFO size reduction is possible in any slave with high accuracy except for the first one directly attached to the master. This can reduce network latency significantly, so, 25 ppm is mandatory. Clock accuracy can be measured by an EtherCAT master using distributed clocks, allowing for FIFO size reduction and faster round-trip-times. Additionally, the clock accuracy has influence on the DC settling time and the forwarding latency jitter.

18.2 Why should port 0 never be an unused port?

Refer to the special features of port 0: If a frame travels through a slave with port 0 automatically closed (because it is unused, the link state is fixed at no link), the circulating frame bit is set in the frame. If this frame comes to a second device which has also port 0 automatically closed (because unused), the frame will be destroyed.

So, any network containing more than one of the slaves with port 0 unused will experience impossible communication. Additionally, in case of network trouble, masters might try to close ports in order to isolate faulty network segments and to maintain communication in the rest of the network. If port 0 is not used, the master is connected via port 1, 2 or 3. The master might try to close ports 1, 2, or 3 because it expects to be located at port 0, which results in network isolation.

18.3 Link/activity LEDs shows strange behavior

Link/activity LED is on without anything attached, LED is off with other device attached. If all ports are connected, communication with the slave is possible, although following slaves are not seen.

This happens if the link signal polarity is incorrect, i.e. the link information is inverted for the ESC. The communication with the slave is possible if all ports are connected, because the ESC detects that no port has a link, and then it automatically opens port 0. You can check this in ESC DL status register 0x0110: no physical link at all ports (although you can obviously read through port 0).

18.4 Can slaves communicate without SII EEPROM / invalid SII EEPROM content?

Yes! EtherCAT communication is possible; all frames can be forwarded without a problem. Register reading/writing is possible, but process data exchange is impossible. The ESC blocks process data memory access until the EEPROM is successfully loaded (typically no PDI is configured unless the SII EEPROM is loaded successfully). You can do the initial programming of the SII EEPROM via EtherCAT (e.g. for hundreds of slaves with "empty" SII EEPROM in parallel (broadcast).

18.5 Do I need a complete XML ESI description for simple PDI read/write tests?

No, you do not need a complete ESI for PDI read/write tests. Only the first 8 words of the SII EEPROM need to be written (e.g. via the network with values from the pinout configurator, the IP core configurator, or the ESI EEPROM CRC calculator). This enables the PDI interface. Such a device cannot be correctly identified by an EtherCAT master, but basic frame exchange (register view etc.) is possible.

18.6 What do I do with unused ports?

EBUS
Attach RX resistor, leave TX signals open. If EBUS is not used at all, the tolerance of the RBIAS resistor is increased to 1kΩ (i.e., you could set RBIAS to 10 kΩ).

MII/RMII/RGMII
Set link signal to a value indicating no link (value depends on link polarity configuration), leave TX signals open, pull-down RX signals. For RGMII in-band status, a link signal might not be used, but pulling down RX signals also indicates no link.
Leave pull-up for MDC/MDIO in the design, so that the interface can detect the state correctly.

18.7 Resetting ESC, PHYs, and µController

All Ethernet PHYs should be hold in reset state until the ESC leaves reset state. With the EtherCAT ASICs, make use of the nRESET signal. With FPGAs, you should route the nRESET signal out of the FPGA. Make sure the PHYs are in reset state while the FPGA is loading. If FX transceivers are used, make also sure that the transceivers are disabled (powered down) while the ESC is not operational (refer to the application note “PHY selection guide” for more details).

The reason is that otherwise the PHYs will establish a link with the neighbours, these neighbours will open their link, but communication is not possible because the ESC is not operational. The result will be frame loss, which is not desired.

If EEPROM emulation is used, the µController should be able to process EEPROM commands once communication is possible, otherwise an EtherCAT master might time-out.

A µController attached to the PDI should also be reset by the ESC (e.g. by a soft reset via register 0x0040:0x0041). The intention of the reset register is not to regularly reset any slave, but to get as much parts of a slave into a known state in exceptional situations (strange errors which otherwise require a power-cycle).

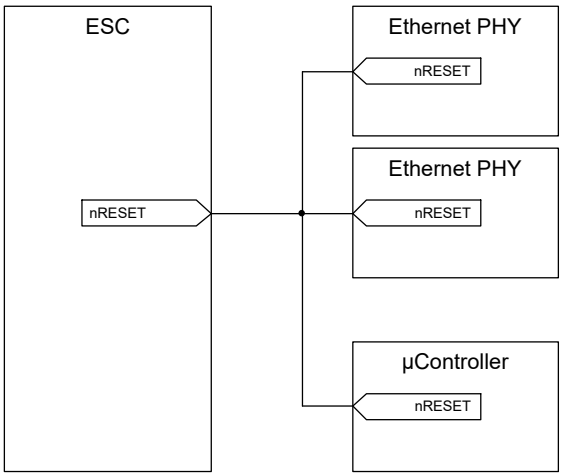


Figure 48: Reset connection principle

Refer to the ESC datasheet and the AN PHY selection guide for more information.

18.8 Should I enable enhanced link detection?

For MII/RMII/RGMII ports, a precondition for enhanced link detection is that the MII management interface is communicating with all PHYs (PHY address configuration ok), which is strongly recommended anyway. Enhanced link detection is recommended regardless of the PHY link loss reaction time. Nevertheless, the demand of fast link loss reaction times is application specific. Therefore, enhanced link detection is enabled in the SII EEPROM, for applications which require fast link-down detection (e.g. with redundancy), while it can be disabled in applications which tolerate potential disturbance and frame loss.

For hardware debugging, it is recommended to disable enhanced link detection, because it could be easier to identify the root cause then.

18.9 Why must I configure the PHYs for auto-negotiation instead of forcing 100 Mbit+FD?

EtherCAT requires that the Ethernet PHYs are configured to use auto-negotiation, and not 100 Mbit + full duplex in force mode. The reason is interoperability.

While two devices which are forced to 100 Mbit + full duplex (FD) are perfectly operating together with EtherCAT, a problem arises if one device uses auto-negotiation, and the other one forced mode. In this case, the auto-negotiation device will fall back to 100 Mbit + half duplex (HD), because it has no knowledge of the capabilities of the link partner. The half-duplex operation can lead to communication issues especially at the master, but probably also at the slaves which check the link configuration via MII management interface. That is why auto-negotiation is mandatory for EtherCAT PHYs.

Another issue is the Auto-MDIX feature of the PHYs, which often depends on auto-negotiation being enabled. So, without auto-negotiation, some EtherCAT links would require a cross-over cable, others a straight cable, which complicates the situation even more.

Refer to the AN PHY selection guide for more information.

18.10 What is TX shift and auto TX shift?

Beckhoff ESCs are not incorporating transmit (TX) FIFOs for the MII interfaces to reduce the latency. Nevertheless, the PHYs have certain timing requirements for the TX signals (setup/hold), which have to be fulfilled. The TX shift configuration is used to move the TX signal timing to a valid window manually. With auto TX shift, the ESC automatically figures out the correct shift value and shifts the TX signals accordingly. Nevertheless, auto TX shift is not the same as a TX FIFO, it still requires synchronous clocking of the ESC and the PHY.

Refer to the ESC datasheet section III for more information about TX shift and auto TX shift.

18.11 Frames are lost / communication errors are counted

During power-up, it is a normal behaviour of the ESCs to count a few communication errors. You can typically ignore them, only new errors which occur during operation are significant. An exception is the PDI error counter, which is expected to be zero after power-up.

Generally, no real bus system is completely error-free, i.e., the bit error rate (BER) is never zero. A few individual errors over time are tolerated by EtherCAT. It needs several errors or lost frames in a row to interrupt the operation of a part of the network. Again, the PDI error counter is an exception, it has to be zero all the time.

If frames are lost or error counters are increasing, configure the EtherCAT master to show the ESC error counters and use the error counter interpretation guide in this chapter.

Refer to the ESC datasheet section II for more information on the ESC error counters.

18.11.1 Configuring TwinCAT to show the ESC error counters

Refer to the appendix (chapter 19.2) for details on configuring TwinCAT for displaying all the ESC error counters. This helps in identifying the problem.

18.11.2 Reduce the complexity

Try reducing the complexity of the problem, which helps in localizing the problem cause:

- Try reducing the network size to a minimum, while the error is still present.
- Disable enhanced link detection for testing to improve error visibility.
- Disable MI link detection and configuration if possible.
- Swap devices in the network to isolate the error source.

18.11.3 Basic error counter interpretation

First of all, check the ESC configuration, e.g. by viewing the POR registers 0x0E00:0x0E01 or the IP core configuration in the user RAM 0x0F80. If the configuration is wrong, check the strapping options and power-on signal values during POR sampling.

Then, check the link status in the ESC DL status register 0x0110. If there is something wrong with the link status, check the link signal. If this is also ok, the problem is in the area PHY – connector – cable – connector – PHY.

If an RX Error counter (0x0301, 0x0303, 0x0305, 0x0307) is increasing, there is a problem in the area PHY – connector – cable – connector – PHY. Measure the RX_ERR signal from the PHY to prove the error counter: if it is sometimes high, check the cabling, connectors, and PHY. Measure the RX_ERR signal from the PHY. The ESC is not the source of the problem.

If there is no RX error, there is probably a master issue. Use a proven master, attach also proven slaves. Remove DUT; move the DUT in the network by swapping positions and check if the error is moving with the DUT.

If the issue is not a master issue, it must be in the area PHY – ESC – PHY, i.e., check the interface configuration of the PHY, the connections itself, the configuration of the ESC, and – in case of the IP core –, if the ESC is correctly constrained and the timing is achieved.

Also think of power supply break-down, clock jitter, and glitches on the reset signal.

18.11.4 Error counter interpretation guide

The following diagram can be used to identify possible error causes. At first, find out which error counters are increasing in your network (or device), and at the master. Compare this with each of the cases shown in the main table of Figure 49 and find the most similar case. Follow the arrows at the bottom of each column to the possible error causes. Use Table 72 for comments and error localization.

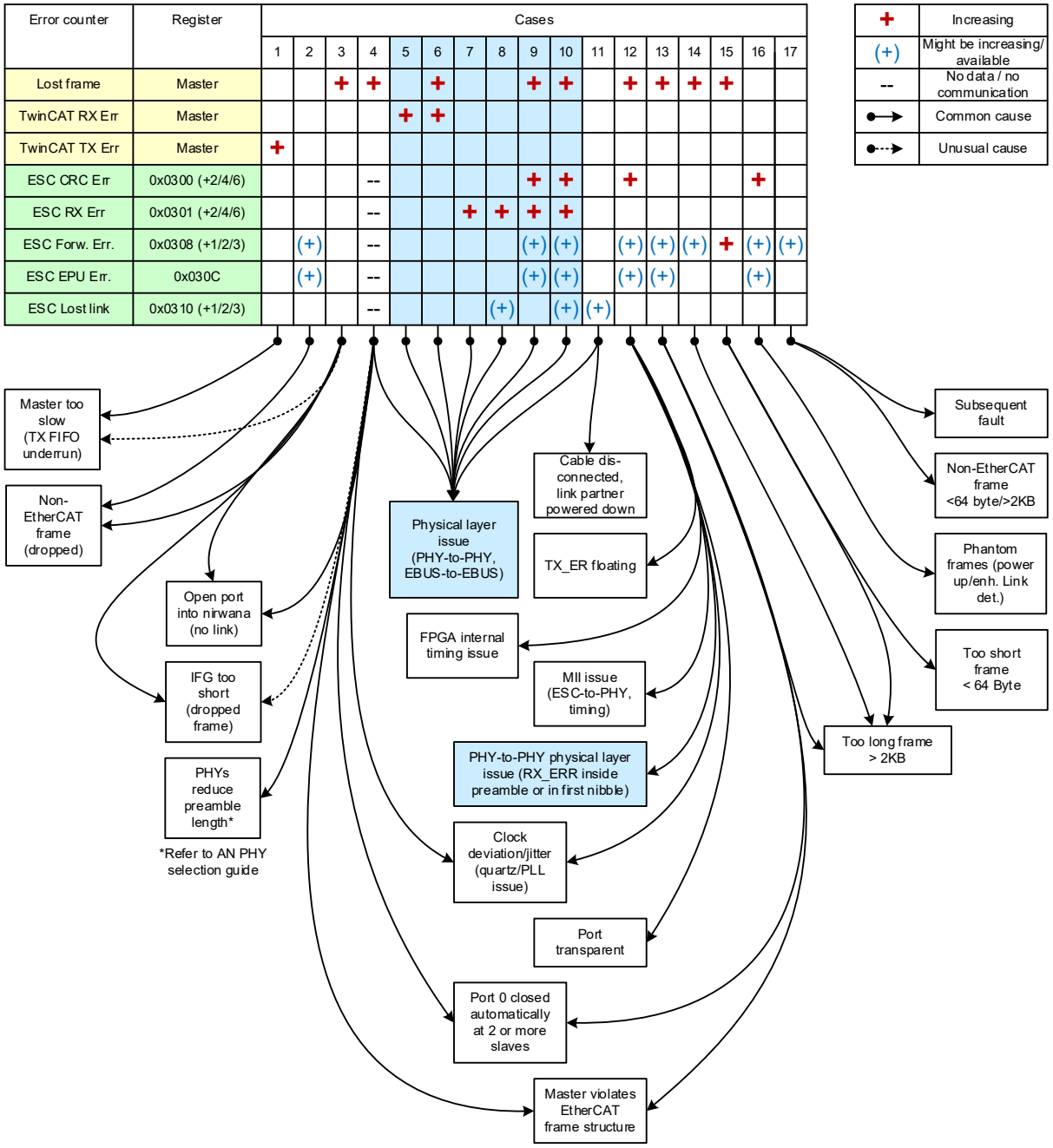


Figure 49: Error counters interpretation guide

Table 72: Error counters interpretation comments

Case	Comment	Localization
1	Master issue	Master
2		Master/slave to slave with increasing EPU error cnt., or between the two slaves in front of slave with increasing Forw. error cnt.
3		
4	no communication to the slave, error counters cannot be read	Last slave with stable communication to slave without communication
5	RX_ER during IDLE	from 1st slave back to master
6	RX_ER during frame	from 1st slave back to master
7	RX_ER during IDLE	Slave to slave with RX err. cnt. increasing
8	RX_ER during IDLE, link down by Enhanced Link Detection	Slave to slave with RX err. cnt. increasing, or last slave with stable communication to slave without communication
9	RX_ER during frame	Slave to slave with RX err. cnt. increasing
10	RX_ER during frame, link down by Enhanced Link Detection	Slave to slave with RX err. cnt. increasing, or last slave with stable communication to slave without communication
11		Slave to slave with Lost link cnt. increasing
12		Slave to slave with CRC err. cnt. increasing, or between the two slaves in front of slave with increasing Forw. error cnt.
13		Slave to slave with increasing or Forw. error cnt., or between the two slaves in front of slave with increasing Forw. error cnt.
14		Any slave to slave at or before the slave with increasing Forw. error cnt.
15		Any slave to slave at or before the slave with increasing Forw. error cnt.
16		Slave to slave with increasing or CRC error cnt
17	Subsequent fault, typically not source of errors	Between the two slaves in front of slave with increasing Forw. error cnt.

18.12 PDI performance

The PDI interface performance cannot be compared with the performance of a simple dual-ported memory, mainly because of the SyncManagers. The SyncManager buffer mechanism coordinates EtherCAT and PDI access to the memory and registers by the means of buffer re-mapping, enabling/disabling accesses, and interrupt/error handling.

18.12.1 ET1100, ET1200, EtherCAT IP core versions 1 and 2

For these ESCs, especially the dependency between EtherCAT frame processing and PDI reduces PDI performance.

The theoretical maximum throughput of any PDI is (1 Byte / 80 ns) = 12.5 Mbyte/s (equal to the maximum Ethernet throughput). Additional latency is required for synchronization (clock domain crossing, EtherCAT frame start) and read/write collision detection.

The datasheet figures are worst case timing, min./max. and average read/write times for the asynchronous µC PDI while using the BUSY signal are (ET1100, preliminary):

Read 16 bit:	min. 195 ns,	average 235 ns,	max. 575 ns
Write 16 bit:	min. 160 ns,	average 200 ns,	max. 280 ns

The maximum read time includes a postponed 16 bit write access, a 16 bit read access, phase alignment and read/write collision. A µController has to cope with this maximum latency, but it is not the average latency! If large amounts of data are to be transferred in one direction (either read or write), the latency is coming close to the average value.

All PDIs are using the same internal interface, thus the maximum throughputs of the PDIs are very similar, and even the on-chip bus PDIs are comparable. That's because they are all limited by the internal PDI/SyncManager realization.

The synchronous µC interface is based on the asynchronous µC interface, plus additional output synchronization, i.e., it is slightly slower. The SPI PDI achieves the maximum throughput for accesses with a large number of bytes. The maximum times for the on-chip bus interfaces (Avalon/OPB/PLB) are slightly better, because there is no need of synchronization (the throughput degrades with smaller access width and with a faster bus clock), although the average throughput is similar.

In most situations, the PDI performance is sufficient, at least if the read and write accesses are properly grouped and timed in relation to the network cycle time.

18.12.2 EtherCAT IP core version 3

The theoretical maximum throughput of any PDI is (1 Byte / 40 ns) = 25 Mbyte/s either for read or for write accesses, in total 50 Mbyte/s for parallel read/write accesses. Additional latency is required for synchronization (clock domain crossing).

The datasheet figures are worst case timing, min./max. and average read/write times for the asynchronous µC PDI while using the BUSY signal are (preliminary):

Read 16 bit:	min. 115 ns,	average 155 ns,	max. 285 ns
Write 16 bit:	min. 0 ns,	average 85 ns,	max. 95 ns

The maximum read time includes a postponed 16 bit write access. If large amounts of data are to be transferred in one direction (either read or write), the latency is coming close to the average value.

All PDIs are using the same internal interface, thus the maximum throughputs of the PDIs are very similar, and even the on-chip bus PDIs are comparable. Only the AXI PDIs can make use of parallel read/write accesses and pipelining.

In most situations, the PDI performance is sufficient, at least if the read and write accesses are properly grouped and timed in relation to the network cycle time.

18.13 Interrupts

18.13.1 µControllers with edge-triggered Interrupt / only the first interrupt is processed

The Beckhoff EtherCAT ESCs generate a level-triggered interrupt signal, which combines many internal interrupt sources (AL event request) into one interrupt signal (according to the AL event mask).

If a µController with edge-triggered interrupt input is connected to this ESC, care has to be taken that all interrupts are processed by the interrupt service routine before it is left. It is especially possible that additional interrupts are coming up while the ISR is processed. If the ISR does not check the AL event request to be empty before it is left, the ISR might not be called anymore at all, because the interrupt signal is continuously active, without any further triggering edge. In such a case, the ISR has to check the AL event request register before the ISR is left, to guarantee that the interrupt signal is cleared.

18.13.2 Polling and interrupt handling

Some functions of EtherCAT slave controllers can be used either by polling or by interrupting. This could be:

- SyncManager reading or writing (mailbox, 3 buffer mode)
- AL control changes
- Distributed clocks SyncSignal events
- etc.

Typically, only one of the two methods (polling or interrupting) should be chosen. Using both methods simultaneously is potentially dangerous, because interrupts could be missed when the polling access “acknowledges” the interrupt immediately when it occurs, and the interrupt service routine does not get called (because the interrupt is already gone or was never seen on the interrupt signal).

18.14 Distributed clocks: resolution, precision, accuracy

The distributed clocks system is operating internally with a 100 MHz clock source, i.e. 10 ns cycle time. Nevertheless, most DC values are represented by multiples of 1 ns (only the pulse length of DC SyncSignals is counted in multiples of 10 ns).

Question: How good is DC, 10 ns or 1 ns?

To answer this question, different terms have to be considered. The short answer is that the resolution of most values is 1 ns, but the precision is basically 10 ns. By averaging, the accuracy can be increased so that it can come close the 1 ns resolution. This is a more detailed answer:

Resolution

The resolution is the smallest value you can read or set in a DC register. The resolution of the DC values is 1 ns, only the pulse length of DC SyncSignals has a resolution of 10 ns.

Precision

The precision is somehow the “quality” of a single event or measurement, i.e. the deviation between actual time and the ideal time. The precision of the DCs is mainly caused by jitter, e.g. due to the 10 ns / 100 MHz operating clock of the DCs.

Accuracy

The accuracy is like the “average of the precision”, i.e. the average deviation between a couple of measurements and the ideal value. The accuracy of most DC values gets better and better the more measurements are made. This is a statistical effect, with better accuracy for an increased number of measurements.

Ideal reference

For most DC functions, the precision and jitter reference is the ideal and continuous time value of the reference clock, the global system time.
The system time value read from the reference clock registers is already subject to a 10 ns jitter because of the 100 MHz operating clock for DC. So the precision of the system time read from the reference clock is 10 ns, the accuracy of the system time at the reference clock is 0 ns.

Synchronization

This system time value is spread over the network into the slaves, which requires knowledge of the propagation delays. These delays are calculated from the DC receive times. The precision of the receive times at the ports and the core depends on the physical layer. For EBUS, the precision is 10 ns, for MII it is 40 ns (because of the 25 MHz MII receive clock). By averaging the calculated delays, their accuracy can get close to the 1 ns resolution.

The system time of a synchronized slave is generated by averaging system time differences. The accuracy of a synchronized slave’s system time depends on the accuracy of the delay measurement (down to 1 ns), but it can only be read with a precision of 10 ns due to the 100 MHz clock source. This is nearly the same for all synchronized DC slaves.

SyncSignal start time and LatchSignal time stamps

The SyncSignal generation and the LatchSignal time stamps depend on the local representation of the system time. The start time of a SyncSignal is subject to a jitter of 10 ns. This adds to the jitter of the local system time, so the total precision of the SyncSignal start time is about 20 ns. Averaging many SyncSignal start times will lead to accuracy near the resolution limit. The precision of LatchSignal time stamps is also about 20 ns, because of the input synchronization stage with 10 ns jitter and the local system time with another 10 ns jitter. The accuracy of multiple latch events gets down to the resolution limit.

NOTE: The precision/jitter and accuracy values are based on the algorithms; real hardware and real clock sources further increase the jitter by a small fraction, and inaccuracies can sum up.

18.15 Hardware is not working

If the hardware is not working, the following functions should be tested, e.g., by external measurement equipment.

General

- Are all power lanes supplied correctly?
- Is the clock running?
- Reset: is the reset signal released?
- Is the ESC accessing the EEPROM after power-up? No: check connection, power, clock, reset, ESC strapping options/configuration again.

Communication

- Link signals: are the PHYs signaling a link? Is the link speed correct? Is the link polarity for the ASIC/FPGA configured correctly?
- The SII EEPROM is not required for communication. Reading registers from EtherCAT master is possible without SII EEPROM at all.
- Turn off enhanced link detection for testing.
- Force the link signal to indicate link up for port 0 and try again (force all other ports to link down). If this does not help, set all ports either to link up, or all ports to link down, and try to communicate via port 0.
- Is PDI access possible? Have a look at the registers to solve the problem. Try reading out the PHY status via MI management.

PDI

- Check if the PDI is active and that the settings are correct (PDI control, PDI configuration)
- Check the PDI error counter (if available)
- Try user RAM access (0x0F80:0FFF), because the process RAM might be blocked if the EEPROM is not loaded correctly or if SyncManagers are active.
- Watch the RAM area via EtherCAT network while accessing it with the PDI.
- Try writing to registers, e.g., AL event mask. Registers have additional write checks in contrast to RAM. If only register writing is impossible, check for access errors (data sheet).
- Check if the signal state during an access (are the appropriate signals driven by the µC, does the ESC drive the appropriate signals), maybe there is an issue with floating signals.

19 Appendix

19.1 Logging error counters in TwinCAT

By default, TwinCAT reads out the ESC error counter registers 0x0300/0x0302/0x0304/0x0306 (CRC error port 0-3) regularly, and clears them. The results are accumulated in software and displayed in the CRC column for each port of the ESC (separated by commas).

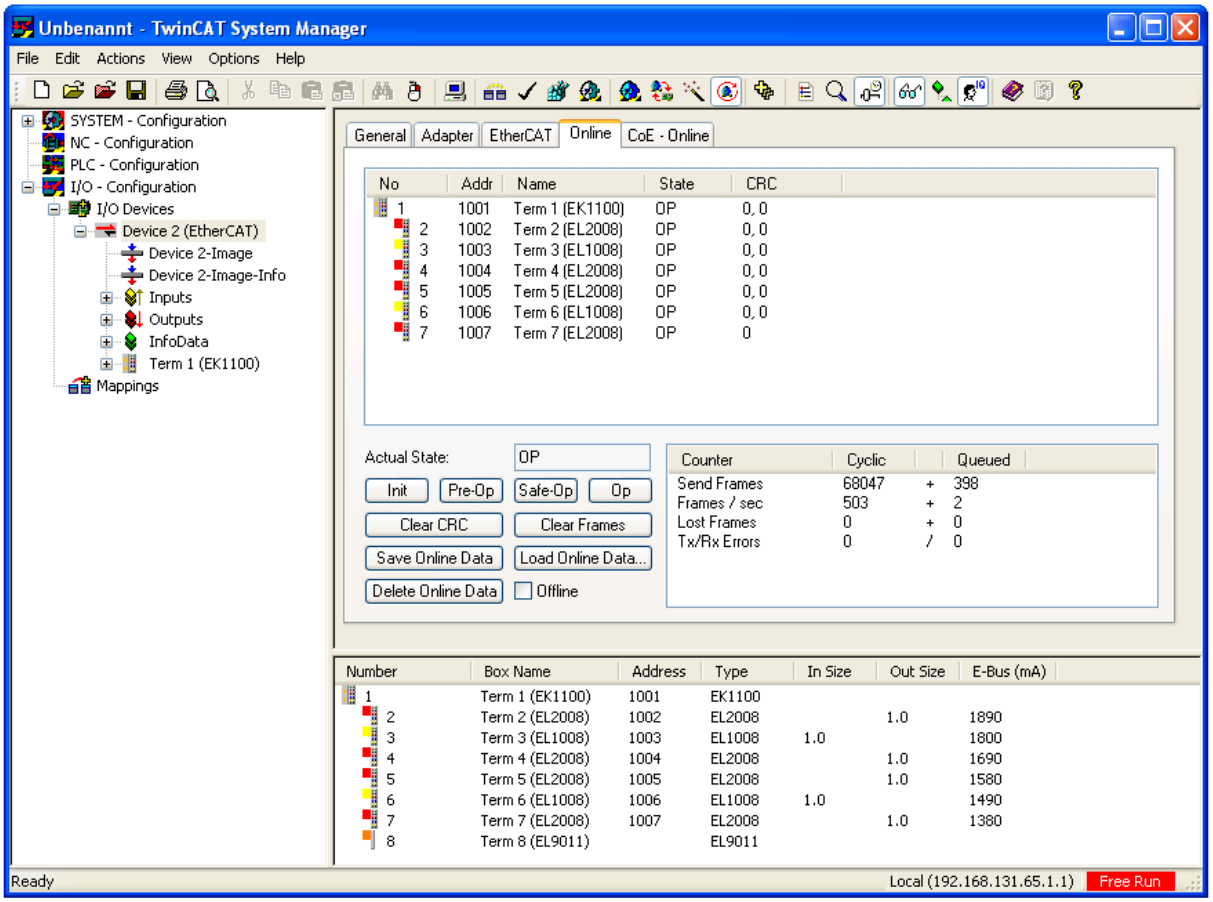


Figure 50: TwinCAT logging CRC error counters

By clearing the CRC error counters, TwinCAT also clears the RX error counter registers 0x0301/0x0303/0x0305/0x0307. The other error counters are also not evaluated by TwinCAT. For a better identification of the error source, it is necessary to disable the CRC error logging and have a look at all the error counters. Follow these steps to disable CRC error logging and displaying the error counters.

1. Go to the “EtherCAT” tab and open “Advanced Settings…”:

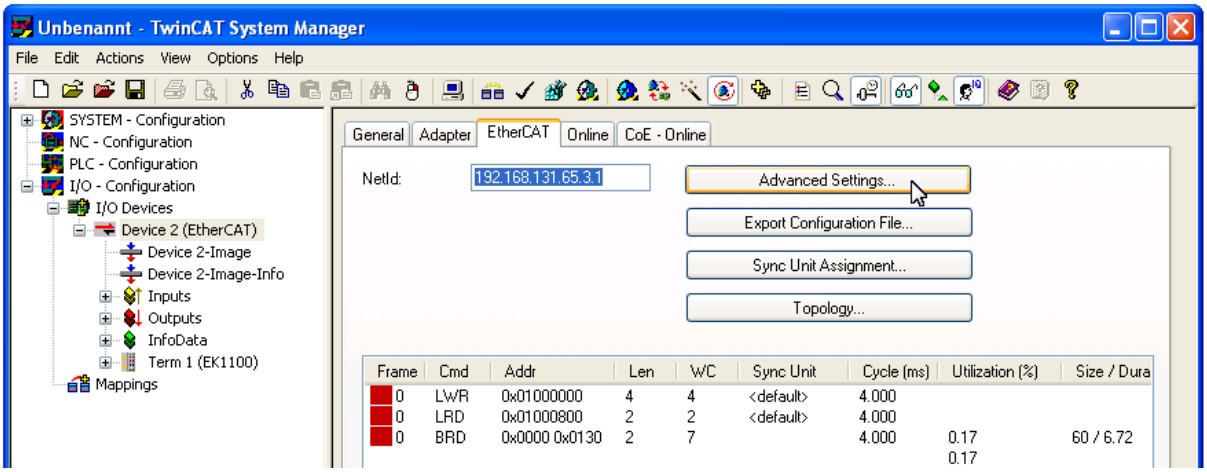


Figure 51: Go to EtherCAT advanced settings

2. Unselect “Log CRC Counters” and press OK.

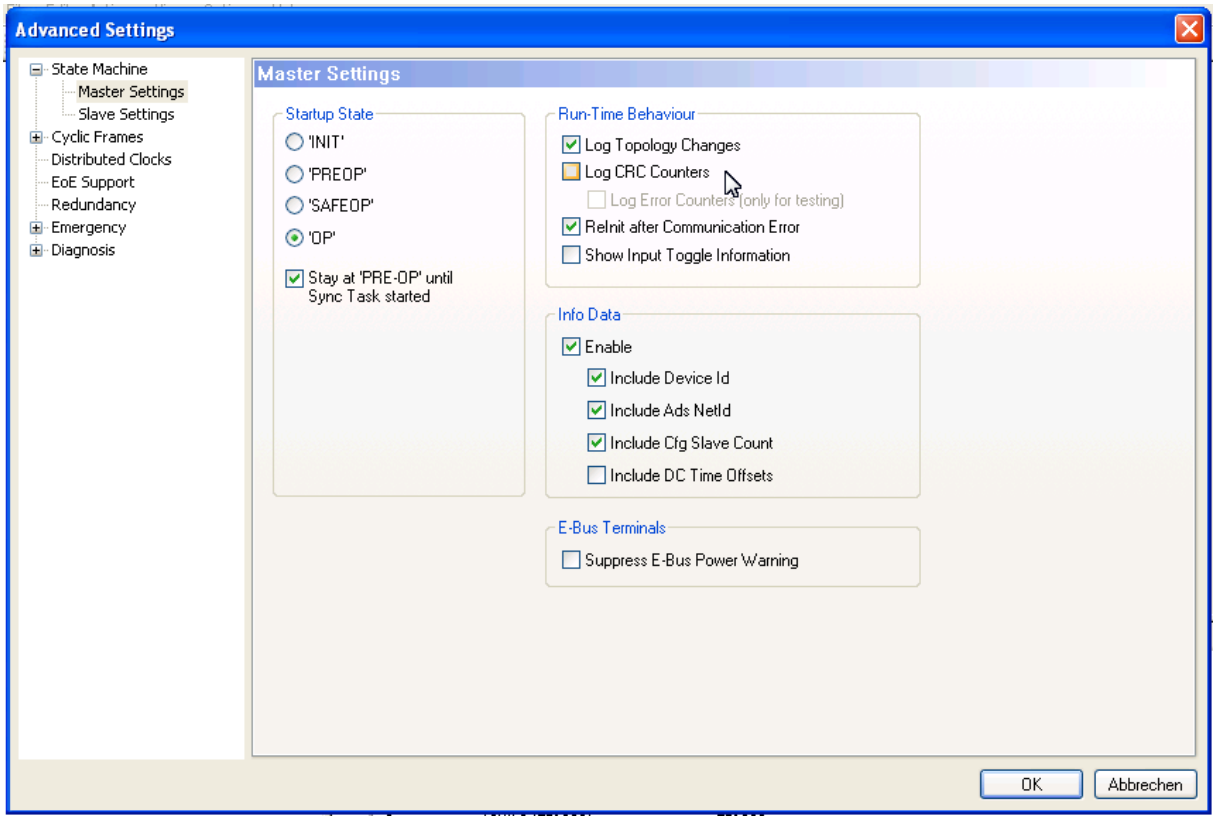


Figure 52: Unselect CRC counter logging

3. Although the CRC column is not visible anymore, TwinCAT still reads out and clears the CRC error counter registers. Activate the new settings by reloading I/O devices (press F4 or use the button).

   It is very important to reload I/O devices (F4) after disabling "Log CRC Counters", otherwise the error counters will still be cleared automatically by TwinCAT.

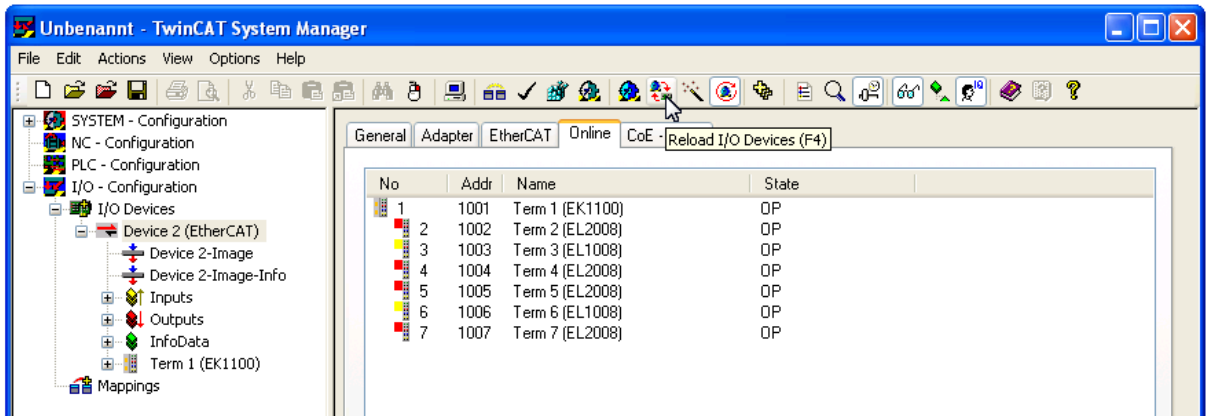


Figure 53: Reload I/O devices (F4) after disabling CRC logging

4. Now you can add the error counters to the online view. Right-click into the Online view area and select "Properties…"

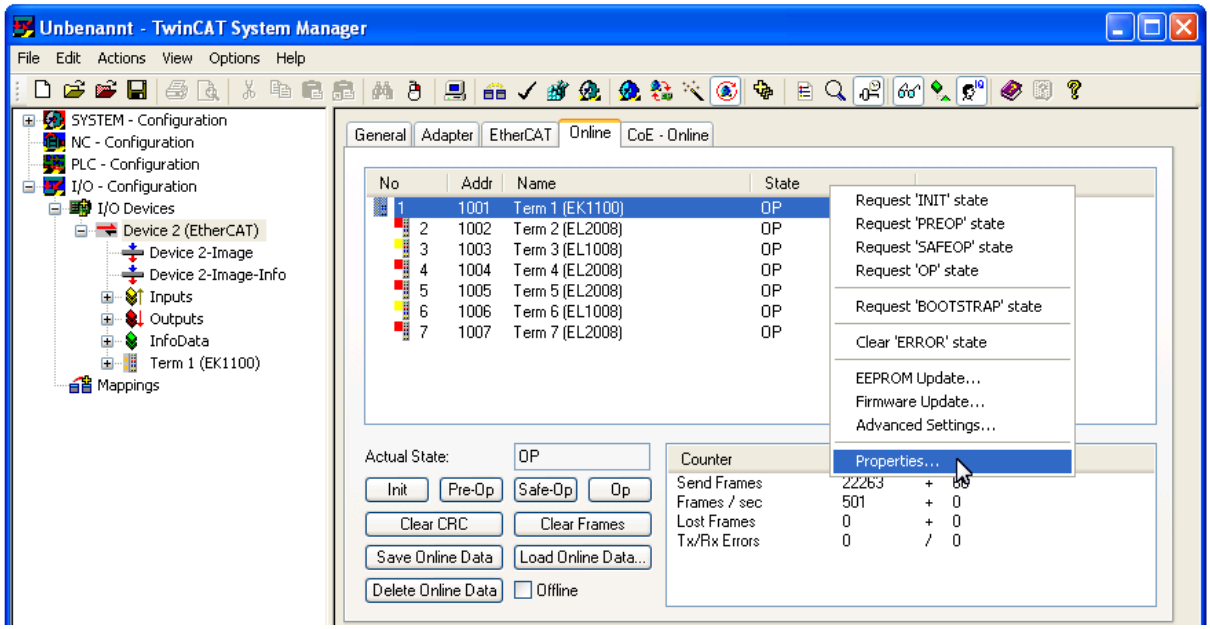


Figure 54: Online view properties

5. Select the following registers for online viewing, and press OK:
   2 port ESCs:
   - “0300 CRC A”,
   - “0302 CRC B”,
   - “0308 Forw. CRC A/B”,
   - “030C Proc. CRC/PDI Err”, and
   - “0310 Link Lost A/B”

   3-4 port port ESCs: additionally select
   - “0304 CRC C”,
   - “0306 CRC D”,
   - “030A Forw. CRC C/D”, and
   - “0312 Link Lost C/D”

   Optionally Enable
   - “0000 ESC Rev./Type” and
   - “0002 ESC Build”

   to identify the type of ESC.

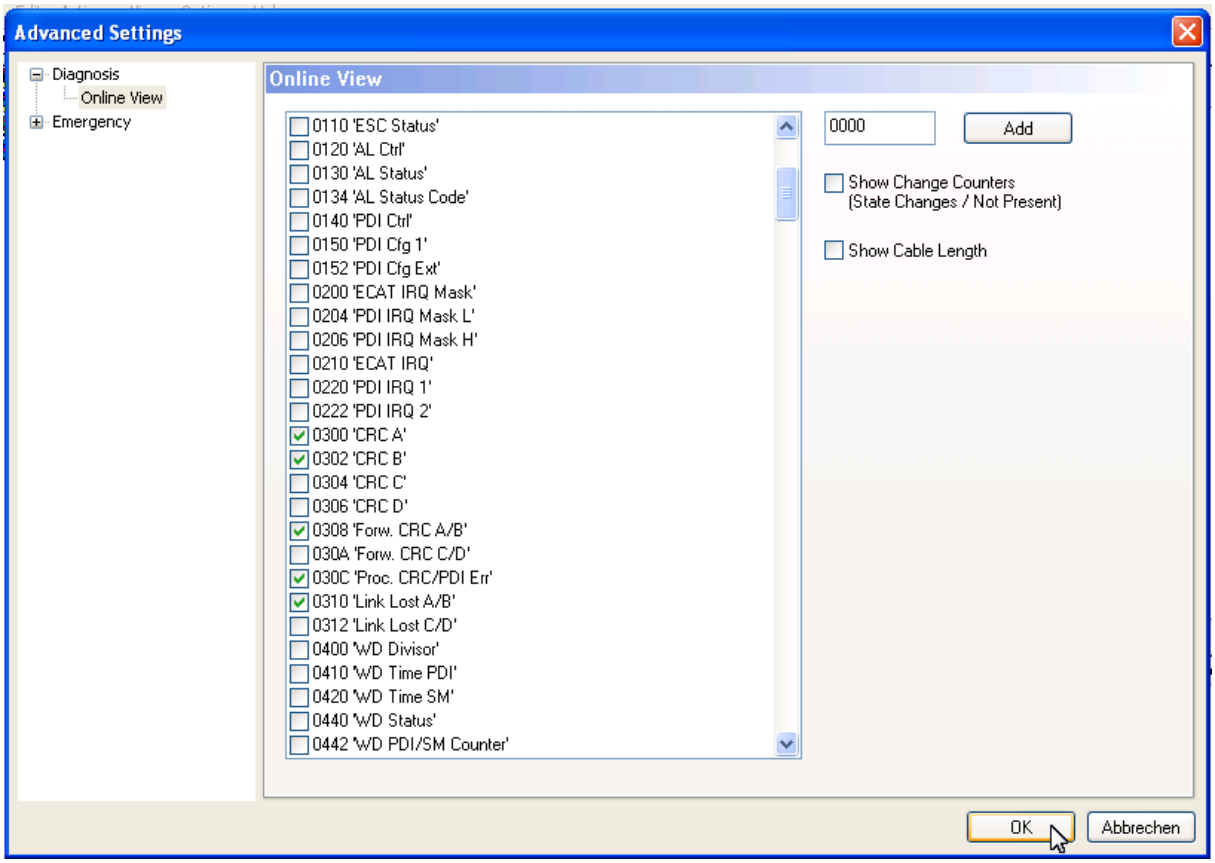


Figure 55: Select registers for online view

NOTE: TwinCAT uses A/B/C/D to enumerate the ESC ports 0/1/2/3.

6. Now you might see some errors in registers 0x030C-0x0313 resulting from power-up. These errors have no significance, only errors increasing during operation are important.
   TwinCAT reads out and clears only the error counters in register 0x0300-0x0307, and pressing the button “Clear CRC” also clears only these counters. The other counters are still accumulating from power-up until now.
   Please note that the error counters are shown as 16 bit values, i.e., the column called “Reg. 0300” shows in fact registers 0x0300 (lower 8 bit, right side) and 0x0301 (higher 8 bit, left side).

   Press the buttons “Clear CRC” and “Clear Frames” to clear the TwinCAT counters in the lower right area (Lost frames, Tx/Rx Errors).

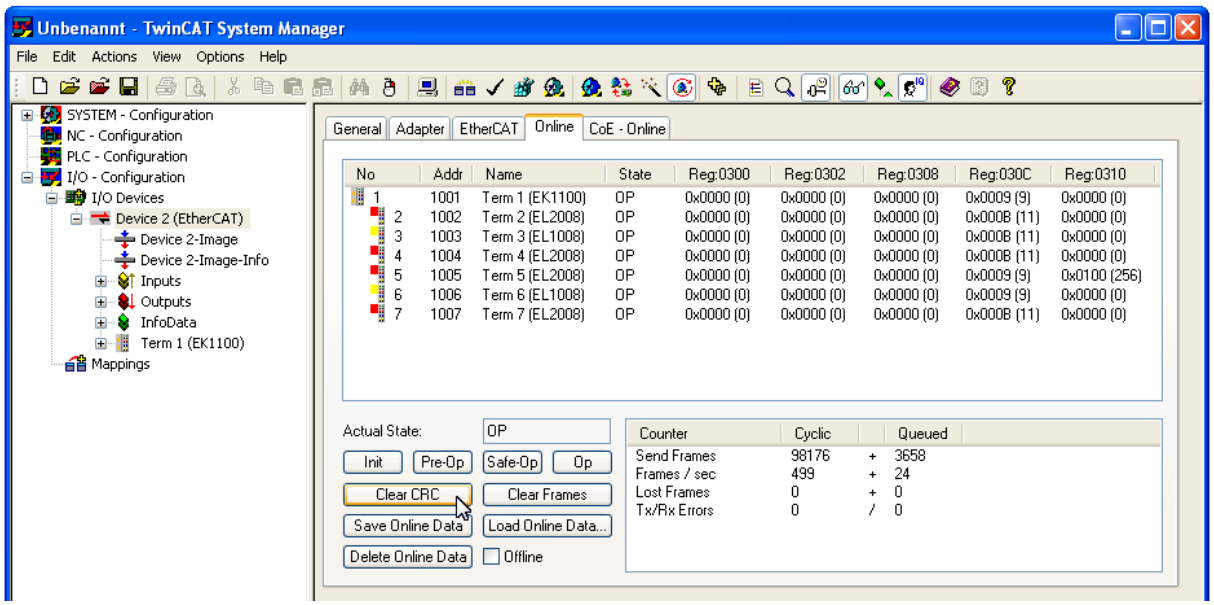


Figure 56: Some power-up errors without significance

7. If TwinCAT does not clear all the error counters (older versions), follow these steps for each ESC: At first, highlight the ESC with the error counters you want to clear. Right-click into the row and select “Advanced Settings…” from the context menu.

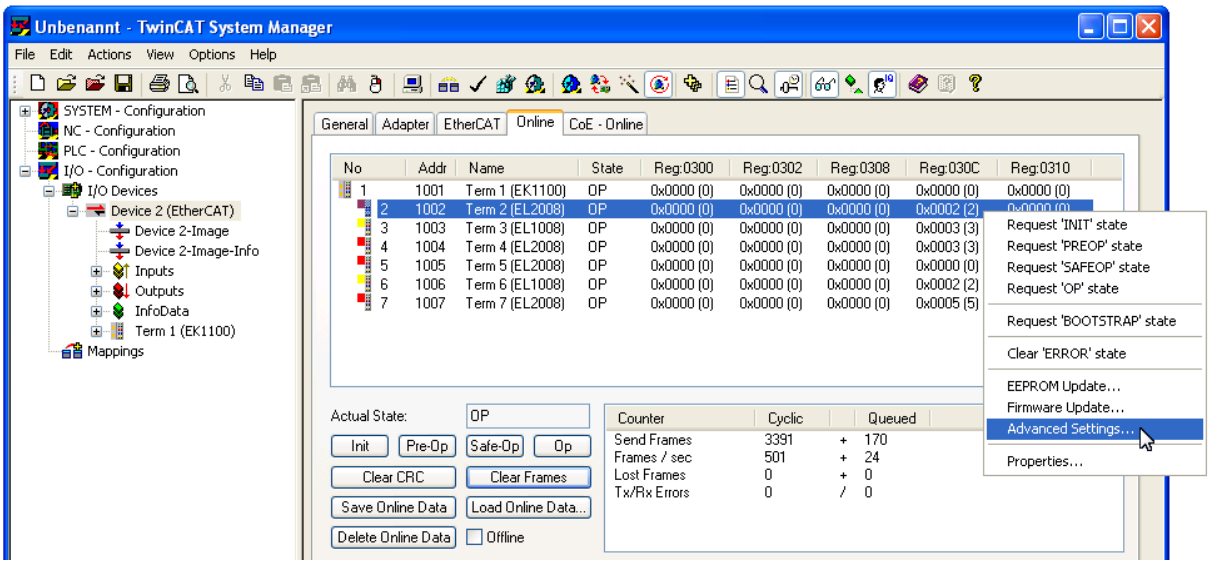


Figure 57: Clear error counters manually in the advanced settings

8. Enter “Start Offset” of 300, activate “Auto Reload” and “Compact view”. Press the “Reload” button to see the error counters from 0x0300 ff. The error counters are again shown as 16 bit values.

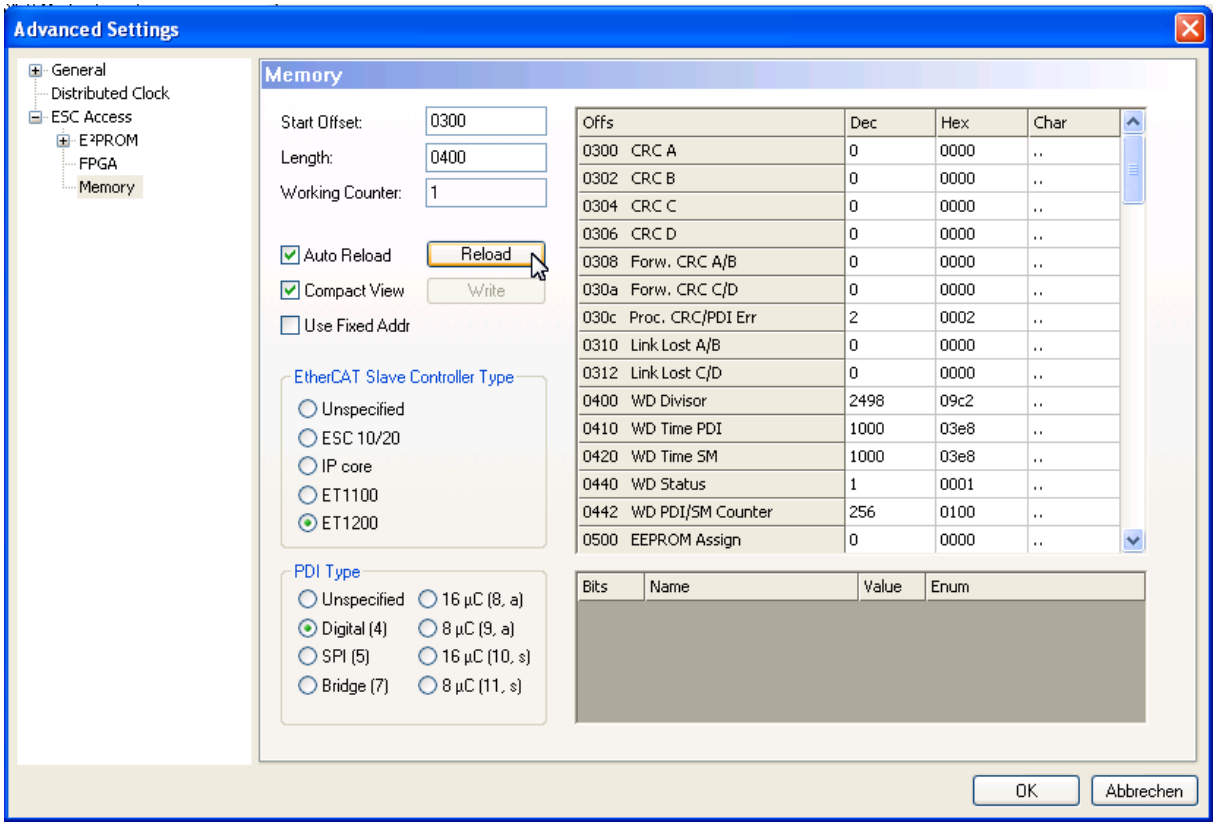


Figure 58: Error counters in the Advanced Settings window

9. To clear an error counter, write any value into it, it is the write access and not the value which clears the error counter. Refer to the ESC data sheet section II to find out which error counters are cleared in a group.

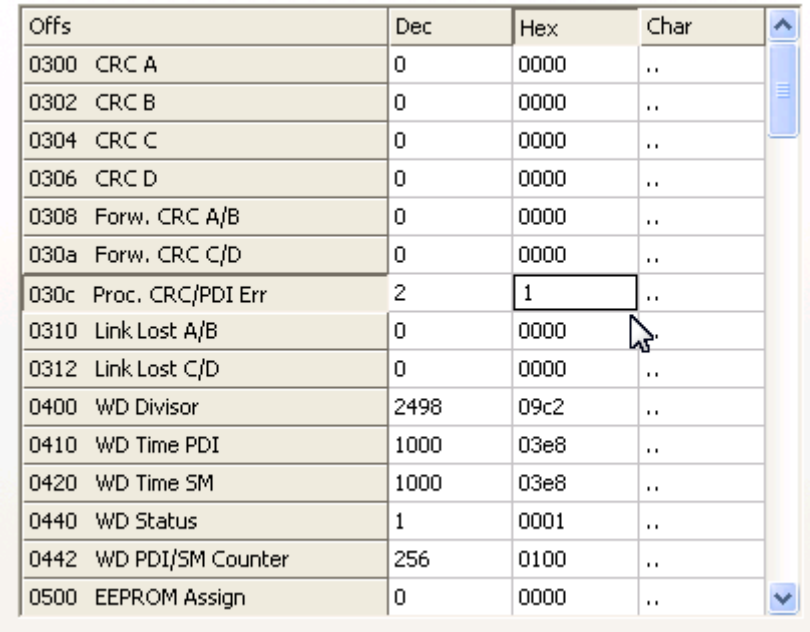


Figure 59: Write any value into an error counter to clear it.

10. Press the “Write” button to write the new value, which in fact clears the error counter.

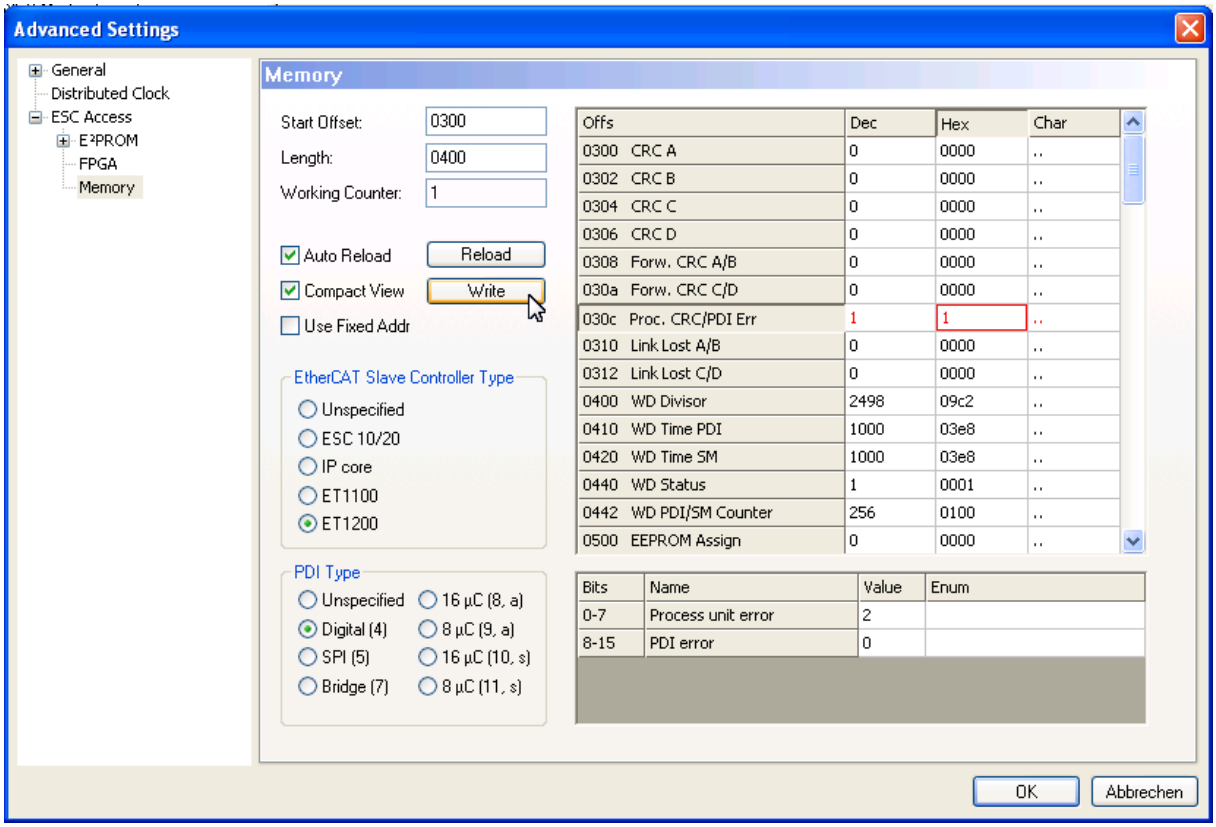


Figure 60: Write to the error counter to clear it

19.2 TwinCAT hints

If communication errors are present, TwinCAT tries to close ports automatically to maintain stable network operation. If you want to observe the instable network part, you have to re-open the ports manually (e.g., by resetting the devices, or writing to the appropriate ESC registers). If TwinCAT closes the ports again, try setting TwinCAT to INIT mode.

19.3 Support and service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

19.3.1 Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages: http://www.beckhoff.com

You will also find further documentation for Beckhoff components there.

19.4 Beckhoff headquarters

Beckhoff Automation GmbH & Co. KG
Huelshorstweg 20
33415 Verl
Germany

phone: + 49 (0) 5246/963-0

fax: + 49 (0) 5246/963-198

e-mail: info@beckhoff.com

web: www.beckhoff.com

Beckhoff support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- world-wide support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

hotline: + 49 (0) 5246/963-157

fax: + 49 (0) 5246/963-9157

e-mail: support@beckhoff.com

Beckhoff service

The Beckhoff service center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

hotline: + 49 (0) 5246/963-460

fax: + 49 (0) 5246/963-479

e-mail: service@beckhoff.com