BECKHOFF

TR3077

“EtherCAT – Diagnostics and Extended Functionalities”

23-Jan-2020

BECKHOFF

Day 1:

- Resume of EtherCAT Functional Principle
- Synchronization & Timings
- Cable Redundancy
- Hot Connect & Swap Prevention

Day 2:

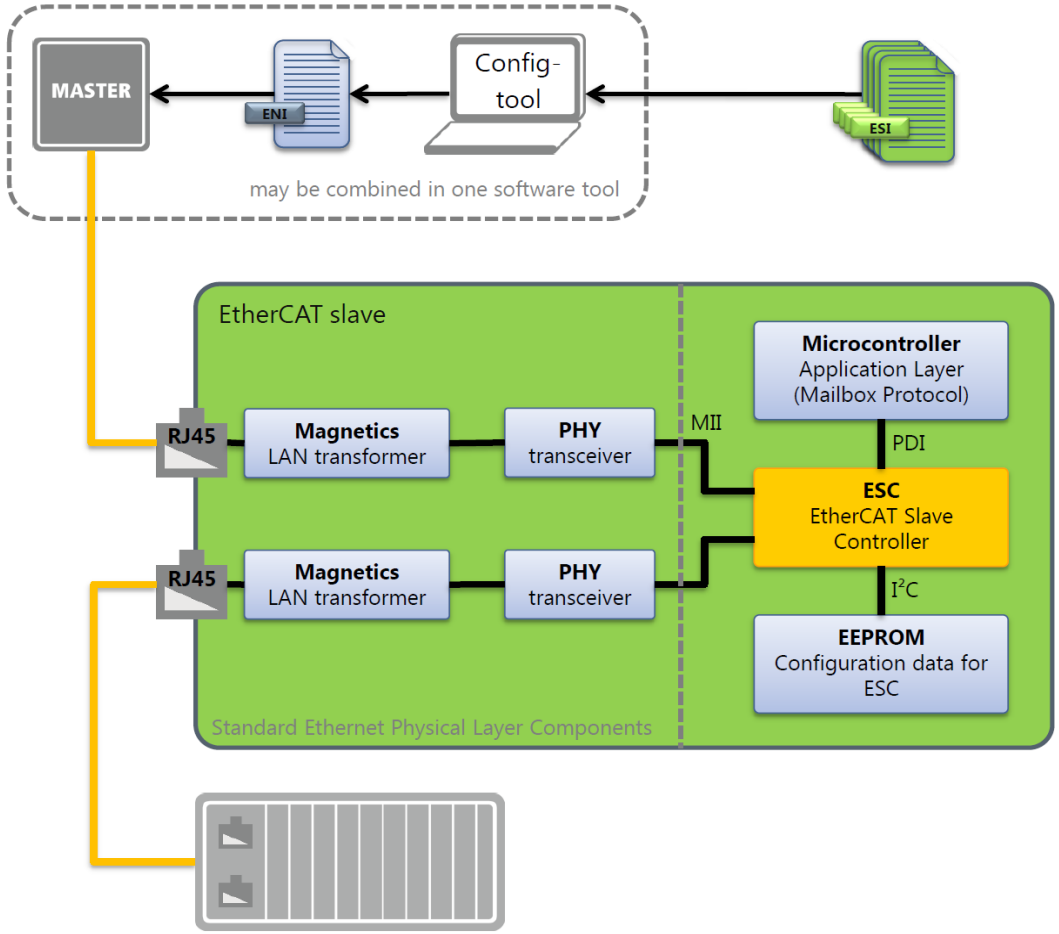
- Hardware and Software Dagnosis
- Ethernet over EtherCAT

Structure of EtherCAT network

BECKHOFF

An **EtherCAT network** consists of:

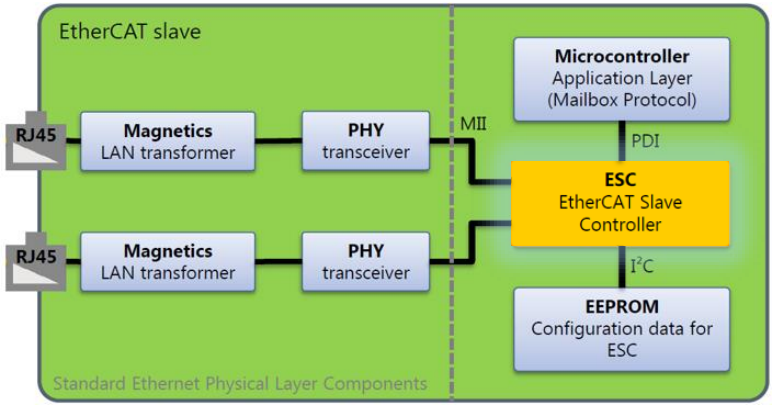
- One **master device**:
- Several **slave devices**: (max. 65535)



Internal structure of ESC

BECKHOFF

The EtherCAT-specific building block, integrated in all EtherCAT slaves, supporting the most time-critical functions is called EtherCAT Slave Controller (**ESC**) :



- Hardware routing of frames on the network (no software addressing like MAC-Address or IP-Address is needed).
- On-the-fly processing of data within frames (no store-and-forward of frames, predictable delays).
- Hardware support for precise synchronization of local times in network devices.

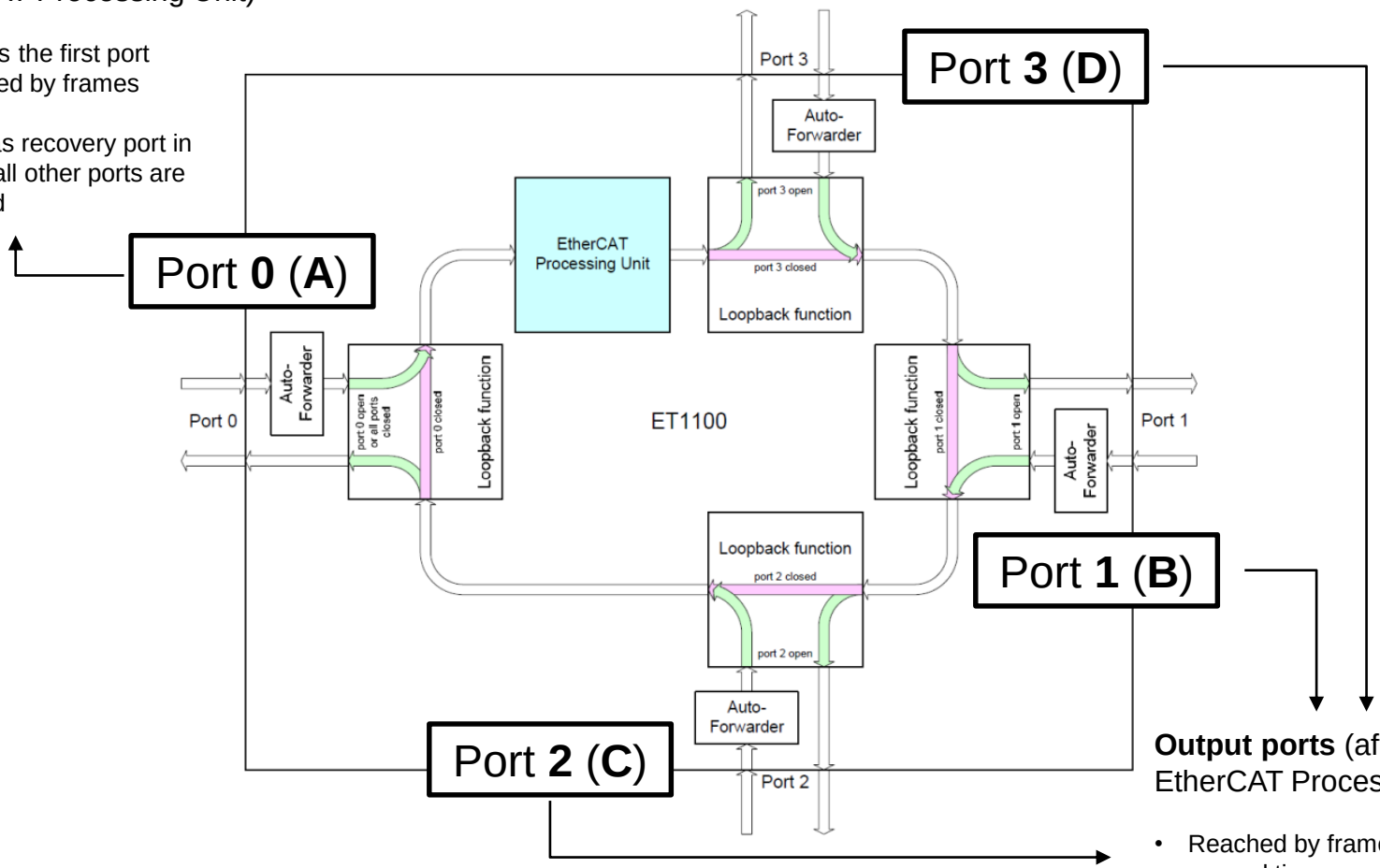
All ESCs support the same basic functionalities, independently on their manufacturer.

BECKHOFF

ESCs can support **2 ÷ 4 ports**:

Input port (before EtherCAT Processing Unit)

- Always the first port reached by frames
- Acts as recovery port in case all other ports are closed



Port **0** (**A**)

Port **3** (**D**)

Port **1** (**B**)

Port **2** (**C**)

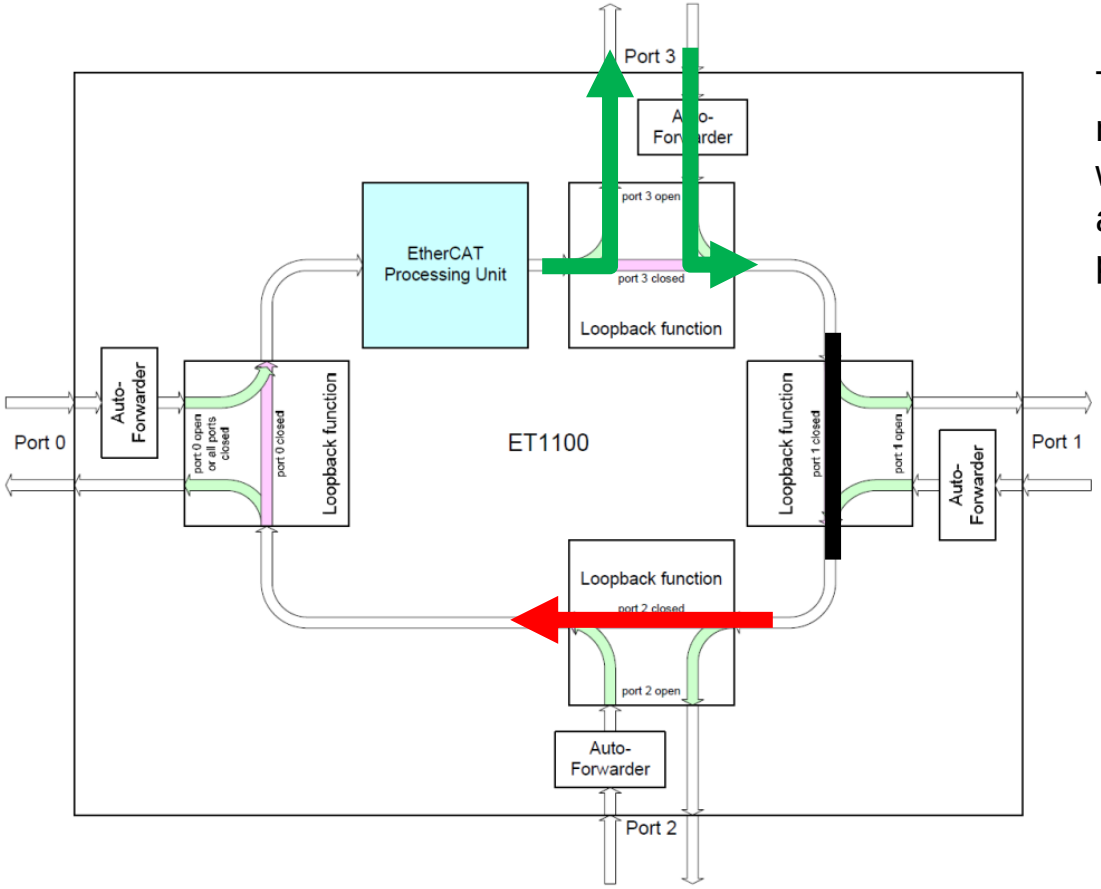
Output ports (after EtherCAT Processing Unit)

- Reached by frames in a second time

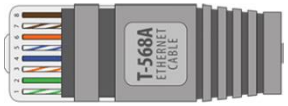
BECKHOFF

Each port can be in one of **two possible states**:

- **Port open** : frames are routed outwards and come back to the Slave through that port
- **Port closed**: frames are forwarded internally to next port of the same ESC

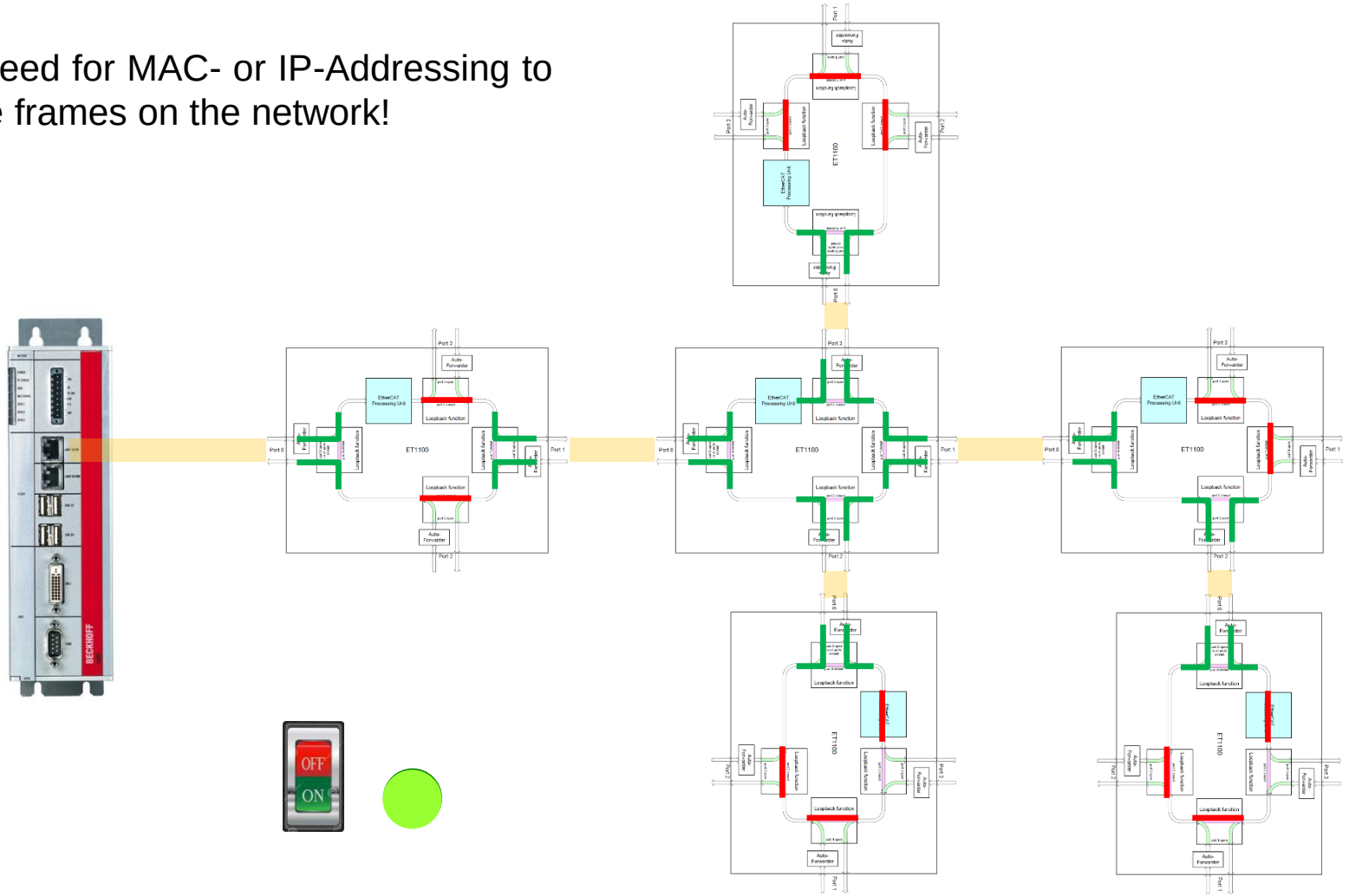


The **default** port setting is **Auto** mode: a port automatically opens when the physical link is present, and automatically closes when no physical link is detected.



Due to this default mode, the **network** builds-up itself **automatically** at power-on:

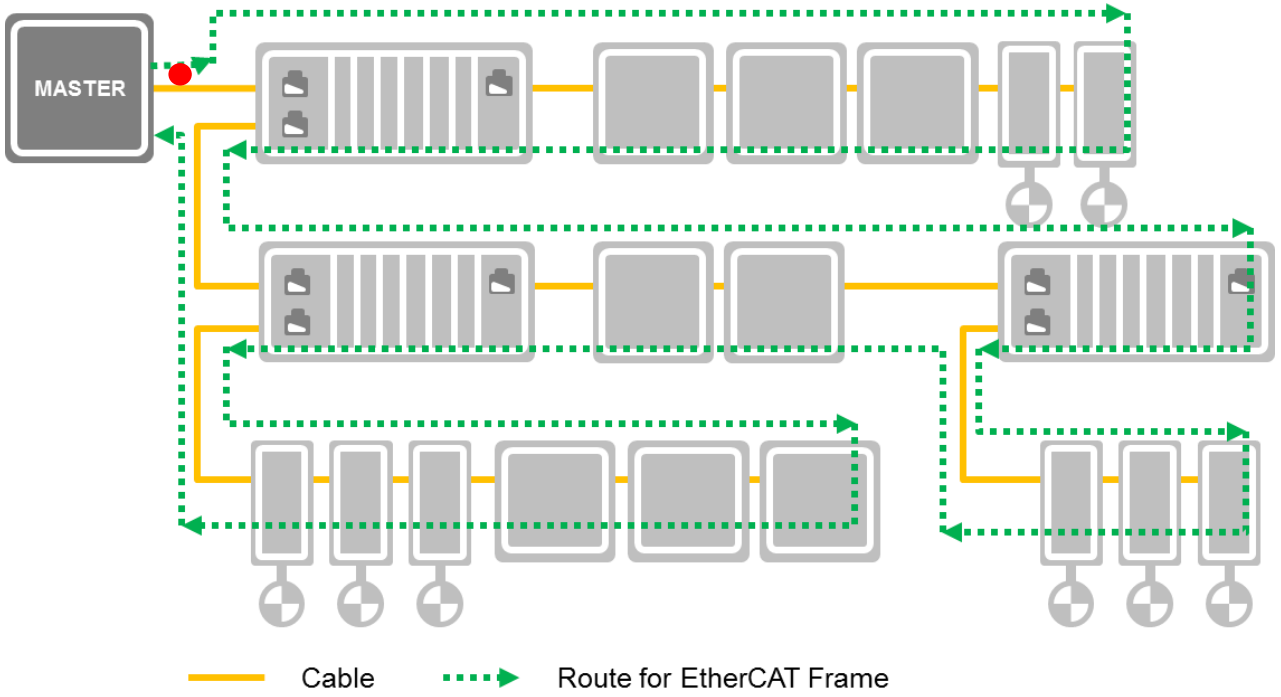
No need for MAC- or IP-Addressing to route frames on the network!



Frame routing through the network

BECKHOFF

All frames propagate in a “**logical loop**” way within the network independently from the hardware topology of the network (which can be line, daisy-chain, star, tree):

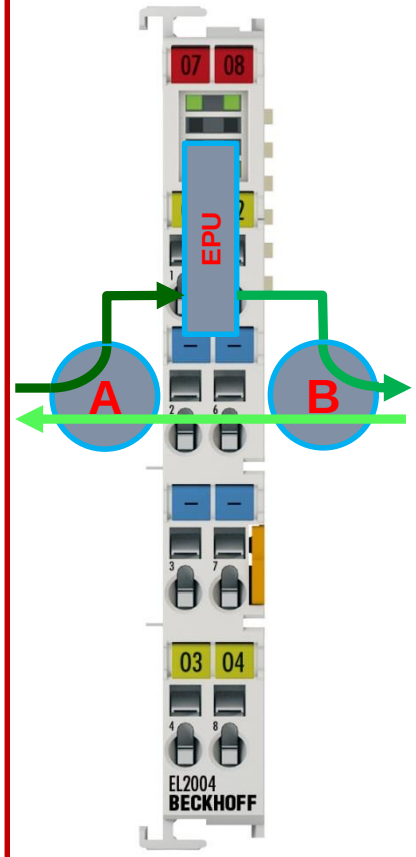


- **All frames** are sent by the master, go **through all** the network **slaves** in a well-defined order, and go back to the master after completing the “loop”.
- All frames go **through** the **EtherCAT Processing Unit** of each slaves **only once**.

Frame routing in Beckhoff Topology Components

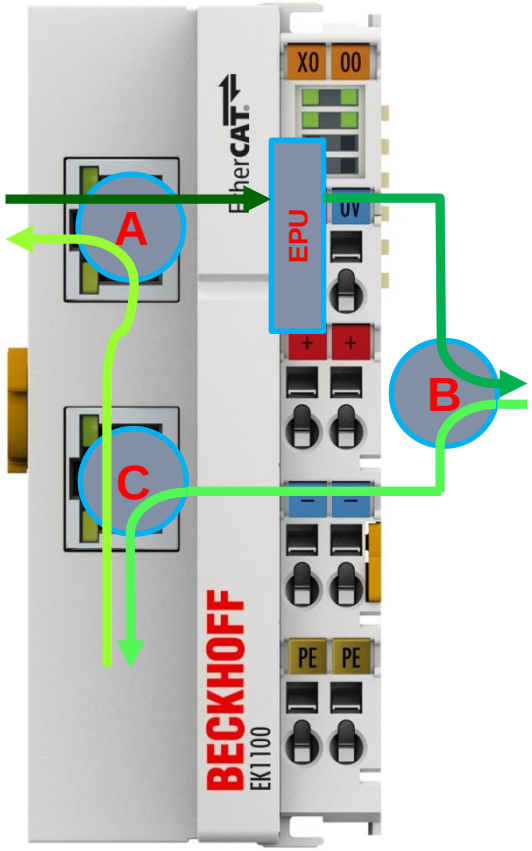
BECKHOFF

ELxxxx



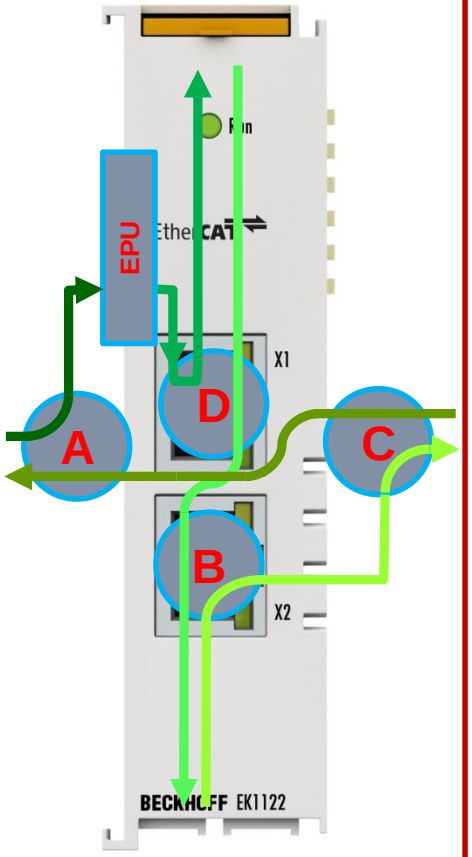
2 Ports (A,B)

EK110x



3 Ports (A,B,C)

EK1122



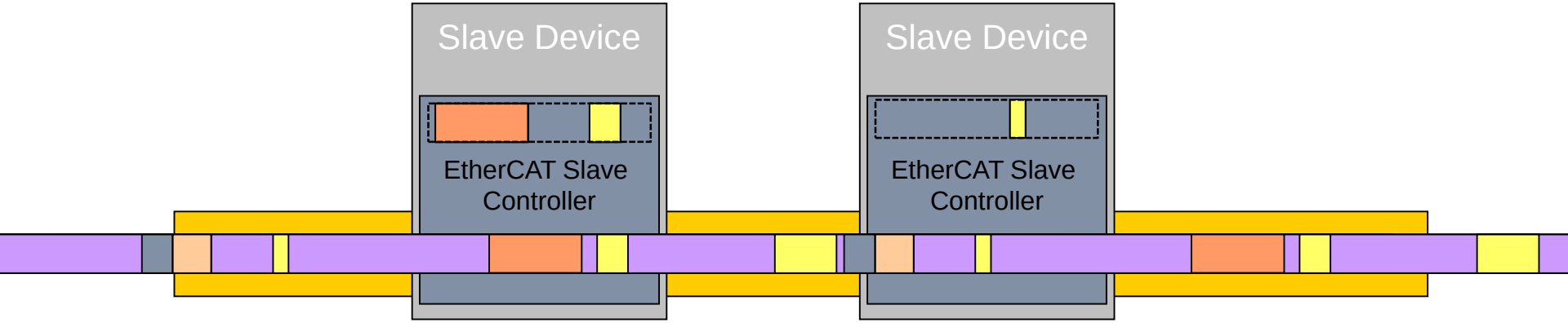
4 Ports (A,B,C,D)

EtherCAT Processing Unit

BECKHOFF

The **EtherCAT Processing Unit** is always located between port 0 and other ports, and processes data “on-the-fly” :

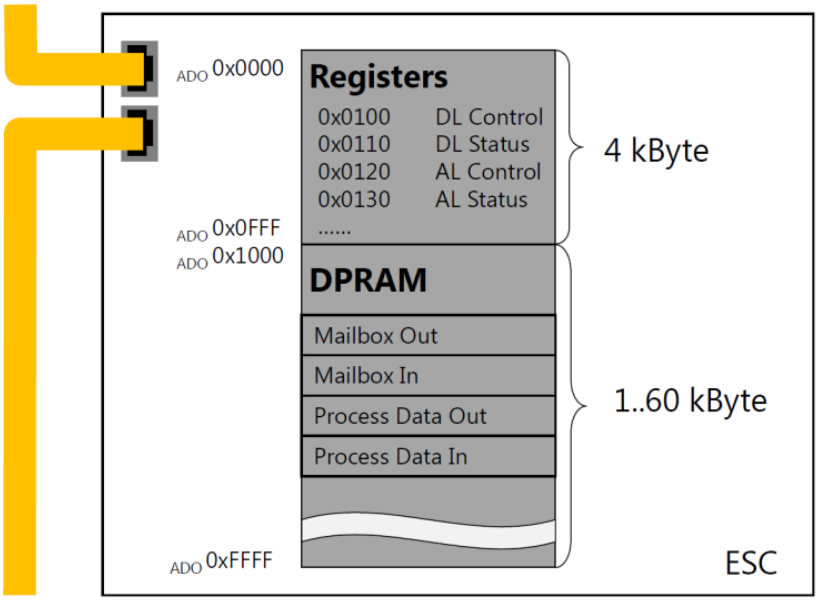
- no store-and-forward of frames
- no software stack delays



Data are written to and/or read from the ESC **Dual-Ported RAM** (accessible both from master and from local µC):

The DPRAM is divided in 2 ranges:

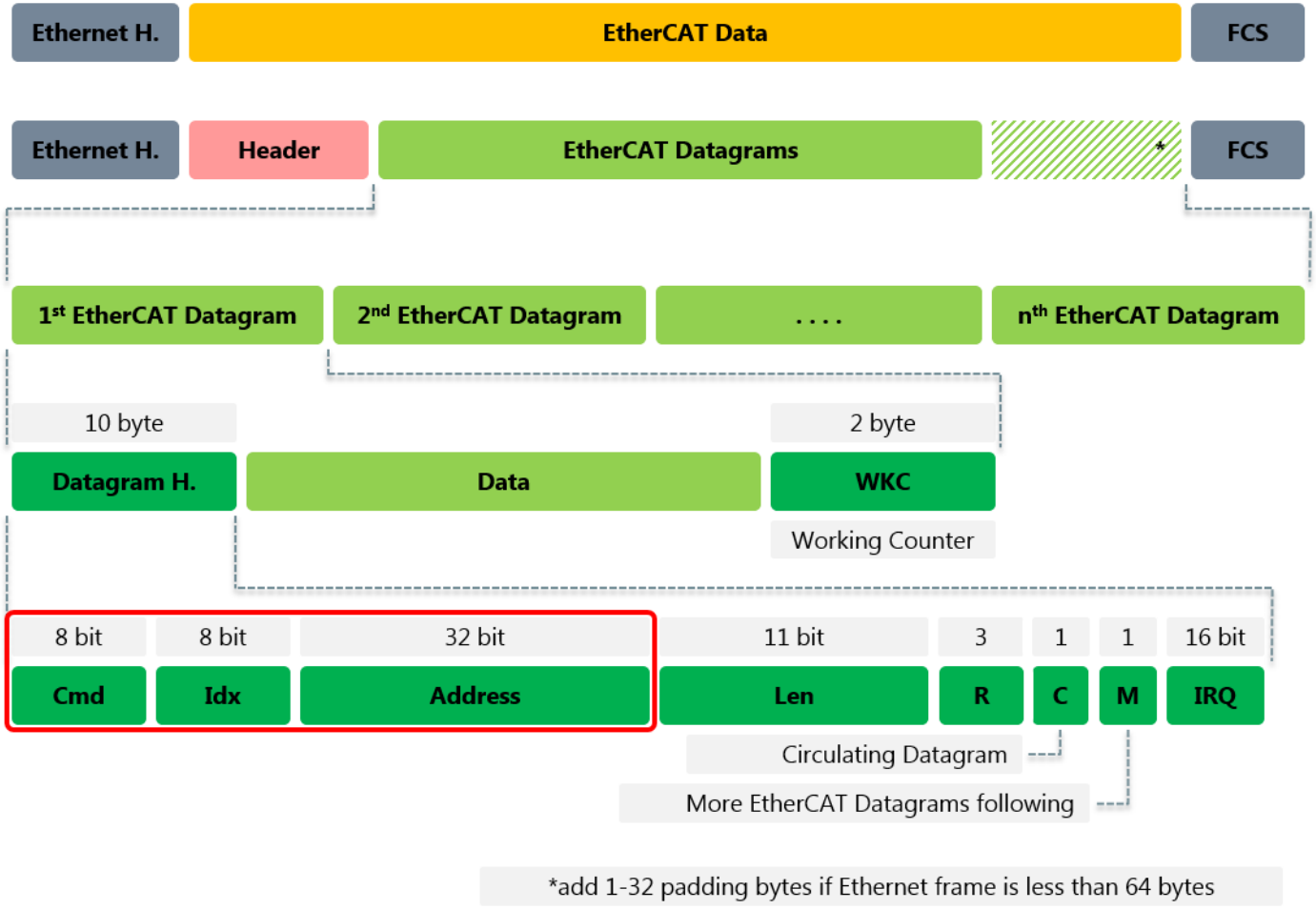
- **Registers** [0x0000÷0x0FFF]: contains low-level settings and diagnostic information.
- **Process Data RAM** [≥ 0x1000]: used for communication between master and slave at application level (Process Data and mailbox).



EtherCAT Frames and Datagrams

BECKHOFF

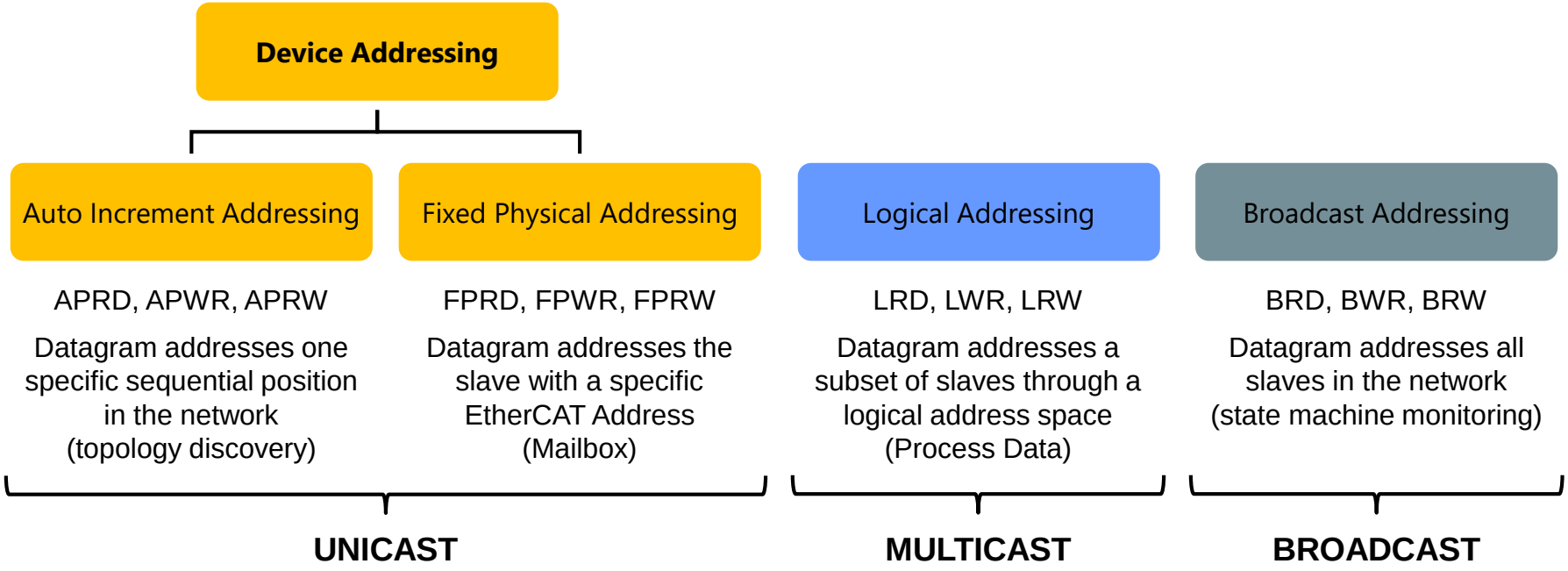
In an EtherCAT network, the payload of Ethernet frames are so-called **datagrams**.



EtherCAT Addressing Modes

BECKHOFF

Each datagram requests a memory access to the DPRAM of one, more or all slaves (according to the **addressing mode** used):



Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)	Utilization (%)	Size / Duration ...
0	NOP	0x0000 0x0900	4			2.000		
0	ARMW	0xfff4 0x0910	4			2.000		
0	LRD	0x09000000	3			2.000		
0	LRW	0x01000000	48	24	<default>	2.000		
0	LWR	0x01000800	34	9	<default>	2.000		
0	LRD	0x01001000	192	39	<default>	2.000		
0	BRD	0x0000 0x0130	2	56		2.000	1.64	387 / 32.88

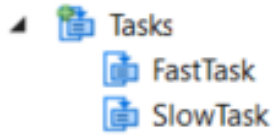
Synchronization and Timings

Synchronization at Application Layer

BECKHOFF

At application level, both the master and the slave device consist of a **cyclically** executed **software** code:

- **master application**: PLC program, NC task, …



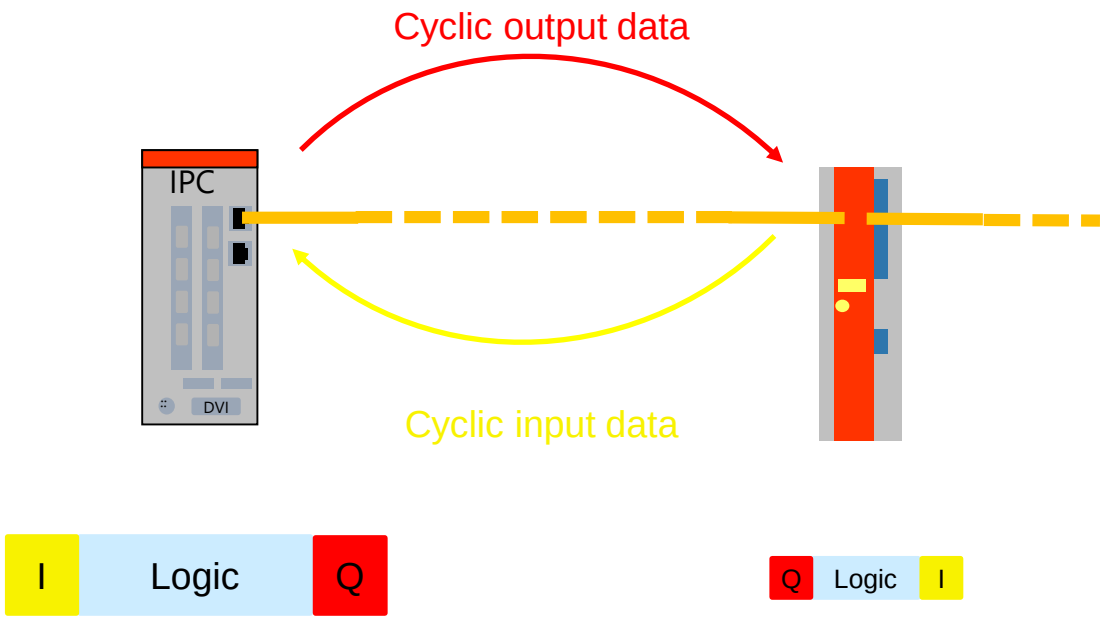
- **slave application**: the slave firmware

Synchronization at Application Layer

BECKHOFF

What does **synchronizing the master and the slave** mean from the point of view of the corresponding **applications**?

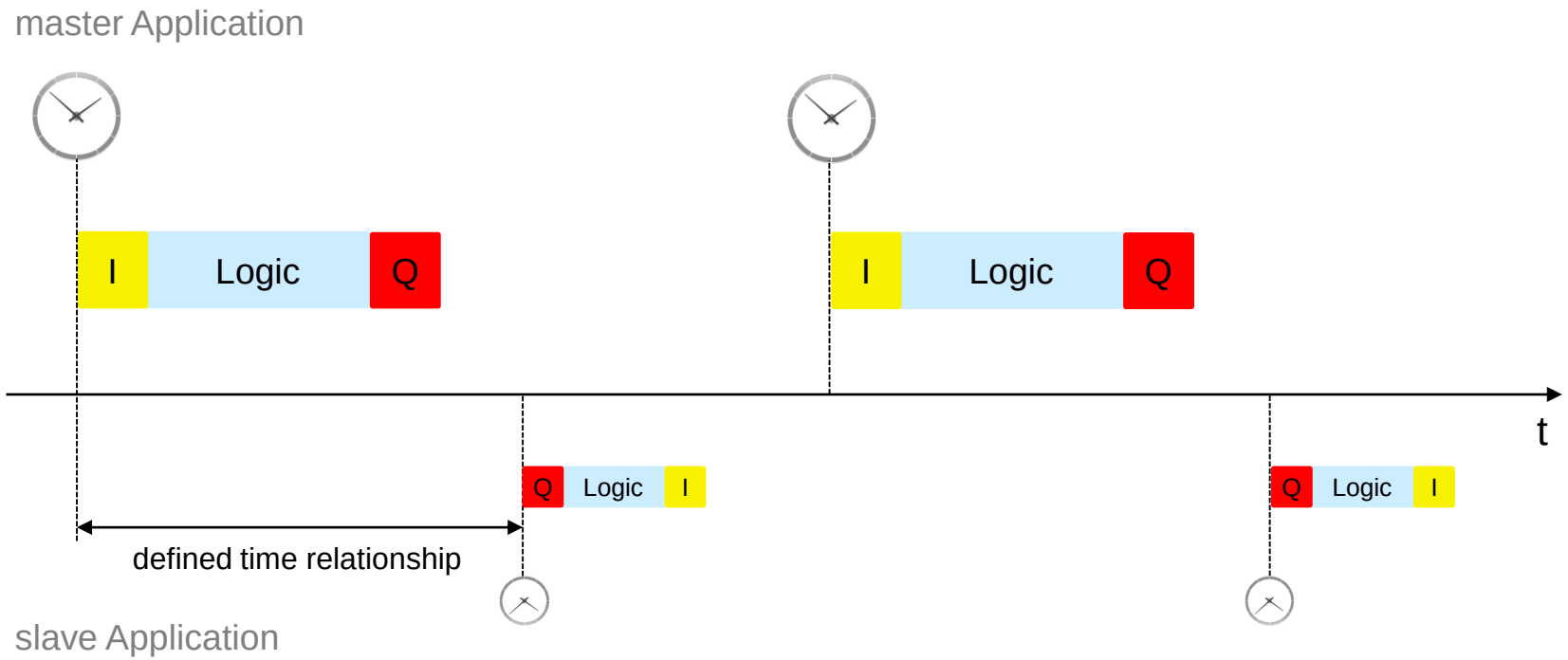
The master and each slave application cyclically exchange **process data** in both directions:



Synchronization at Application Layer

BECKHOFF

Synchronizing master and slave applications basically means defining a **time relationship** between the start times of the cyclic code handling process data in master and in slave:

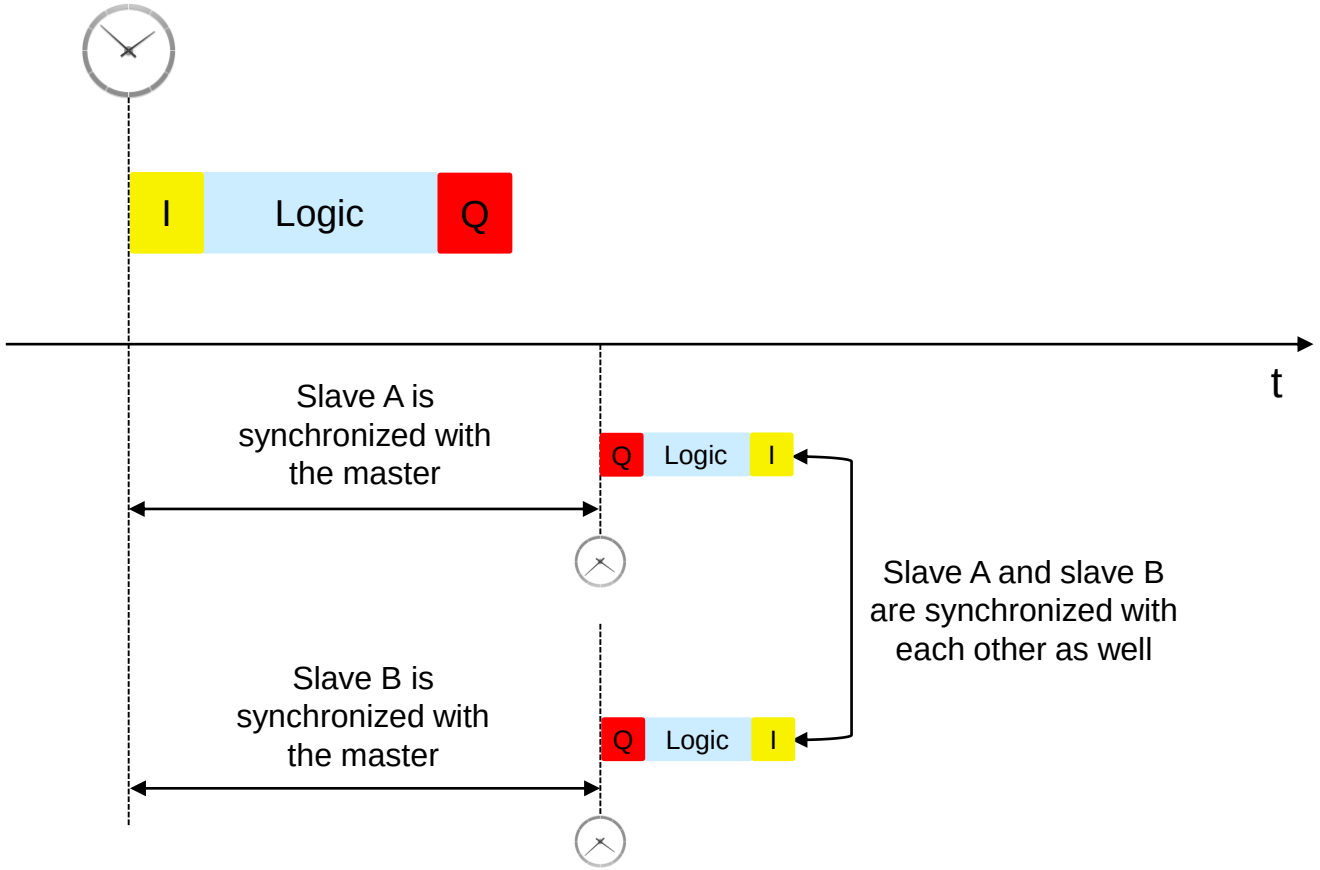


Goal is to time the cyclic data exchange, the setting of cyclic outputs and the latching of cyclic inputs in the way required by the application/user.

Synchronization at Application Layer

BECKHOFF

Synchronizing each slave to the master implicitly leads to the synchronization of different slaves with each other:



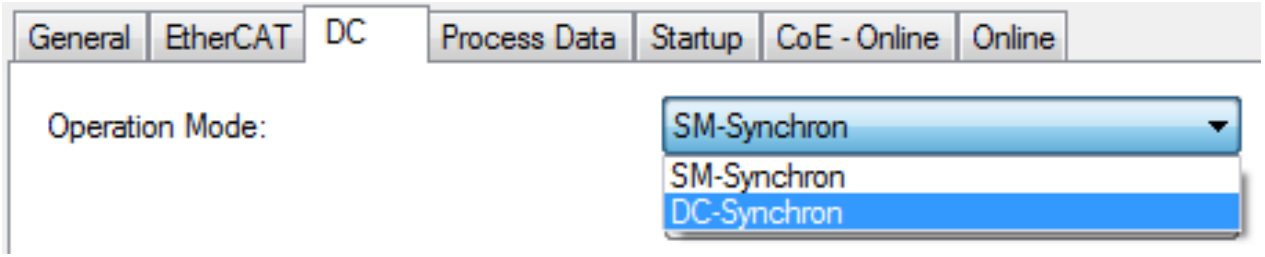
Synchronization Modes at Application Layer

BECKHOFF

EtherCAT defines **three** main time relationships of each slave application with respect to the master cycle (**synchronization modes**):

- **Free Run** (no synchronization): process data handling in the slave is started by an internal event having no defined time relation with the master cycle.
- **SM-Synchronous**: process data handling in the slave is started by a hardware interrupt event generated when the cyclic frame carrying the process data is received.
- **DC-Synchronous**: process data handling in the slave is started by a hardware interrupt event based on the Distributed Clocks and on the corresponding System Time.

When configurable, the synchronization mode is set in TwinCAT in “DC” tab (independently for each slave):

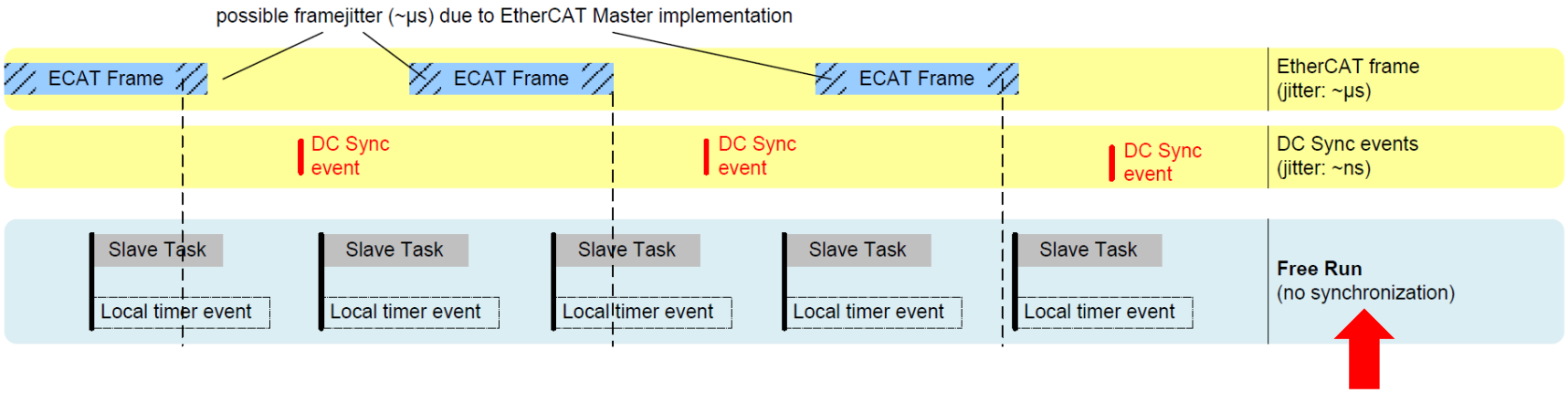


Free Run

BECKHOFF

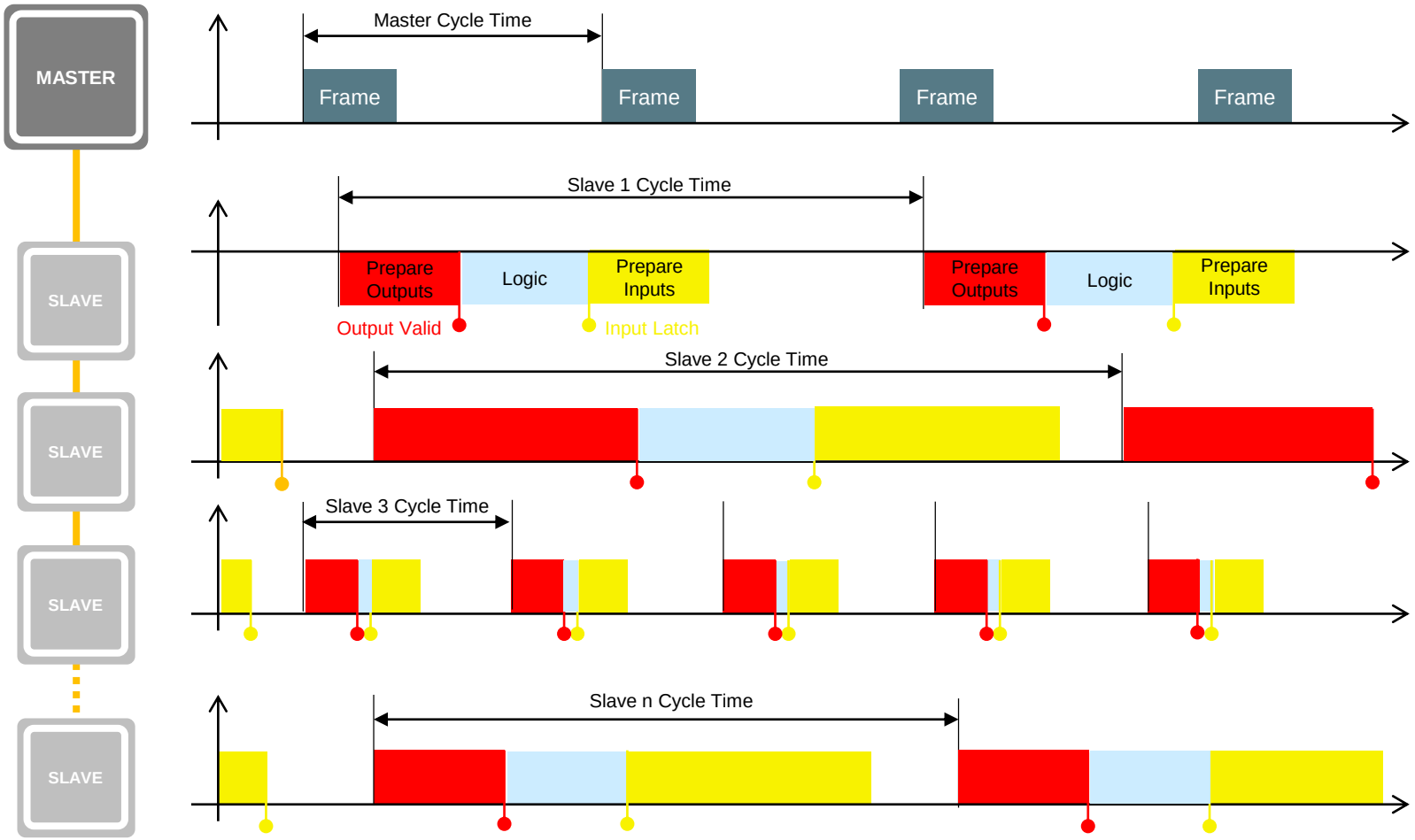
Free Run. The process data handling in the slave is triggered by an internal event:

- No defined time relationship between cyclic frames and local application
- Time offset among different “Free Run” slaves is undefined
- Intended for e.g. I/O devices handling slow-varying signals (temperature, …)



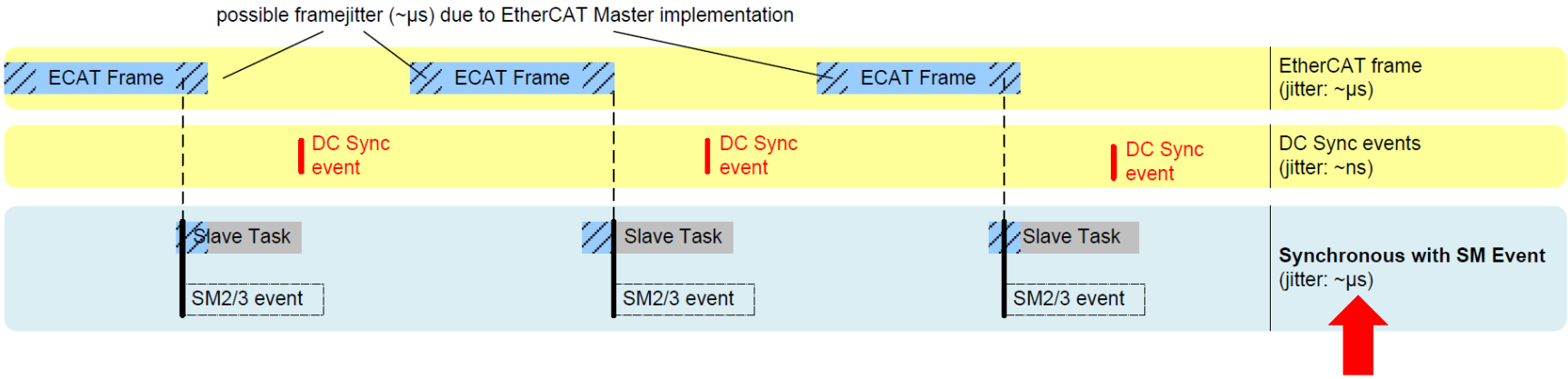
Free Run

BECKHOFF



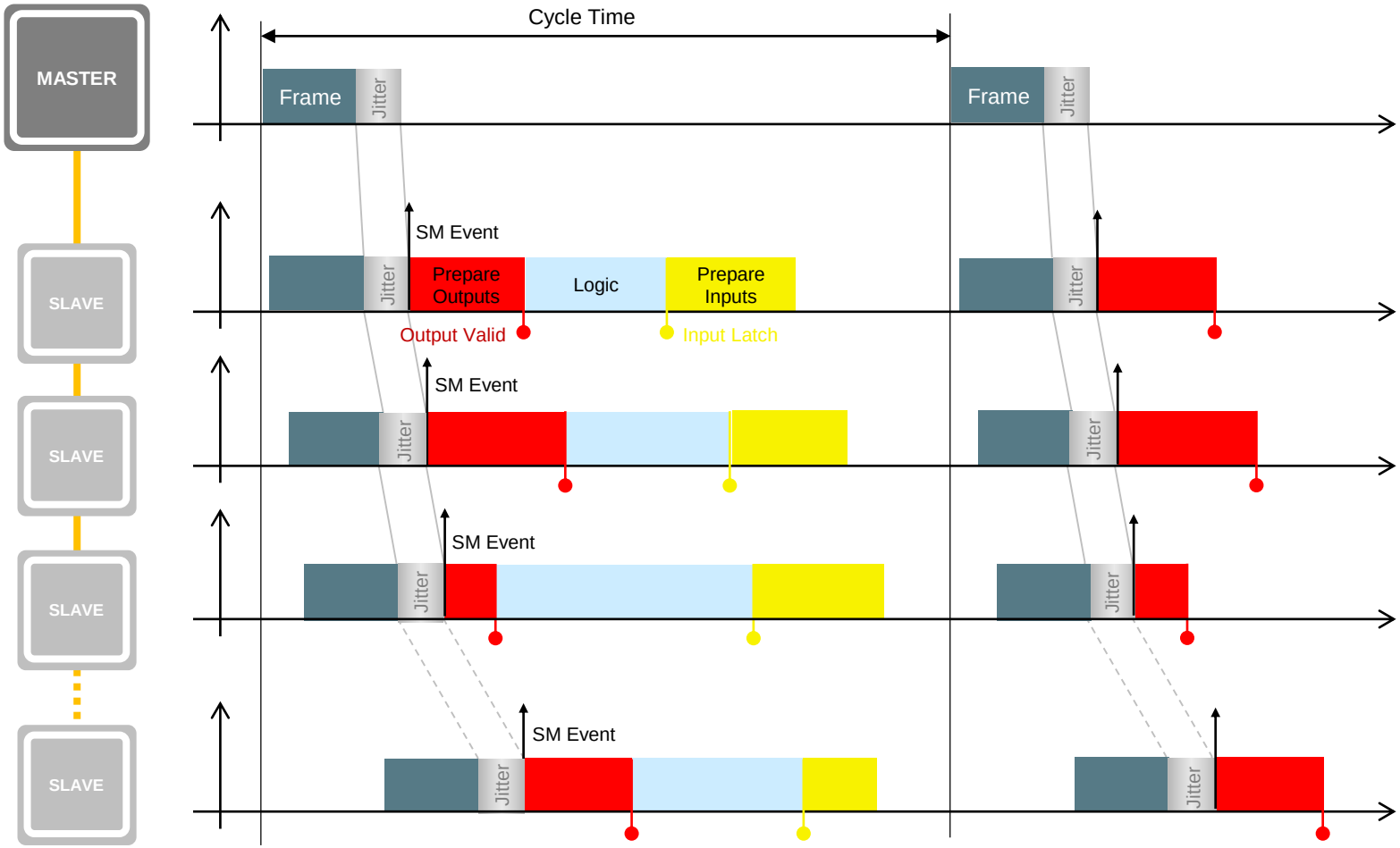
SM-Synchronous. The process data handling in the slave is triggered when the cyclic frames are received:

- First cause of synchronization inaccuracy: cyclic frames are received by slaves with the same jitter which affects the master in sending them.
- Second cause of synchronization inaccuracy : even with no jitter, due to finite hardware propagation delays the last slaves will receive the cyclic frames later with respect to the first ones.
- Intended for e.g. positioners controlled by a PLC task



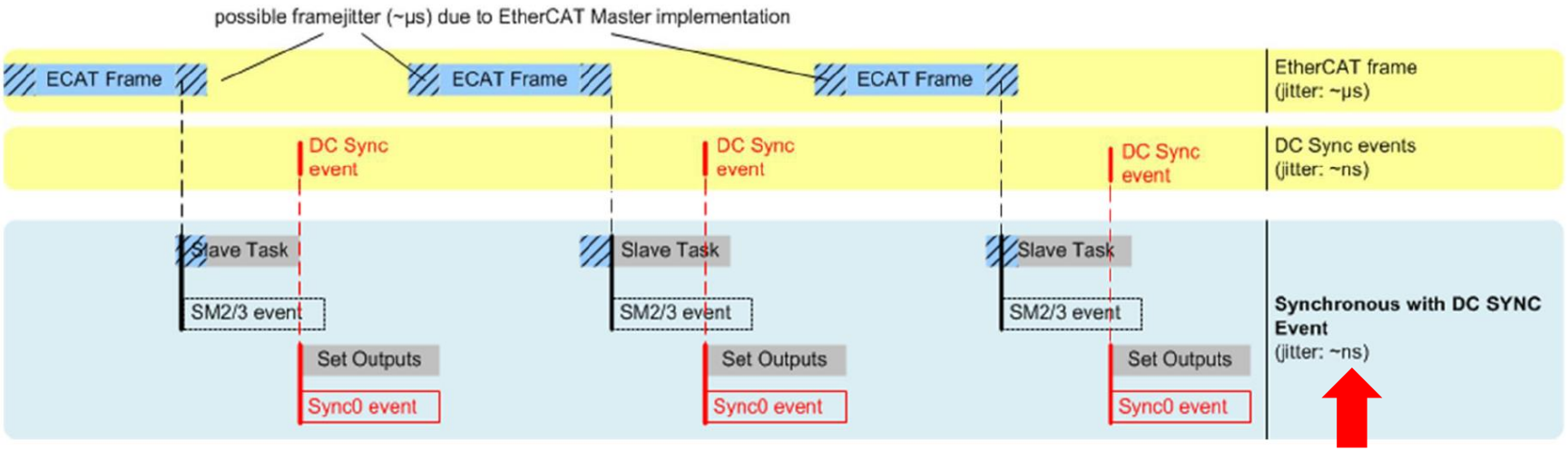
SM-Synchronous

BECKHOFF



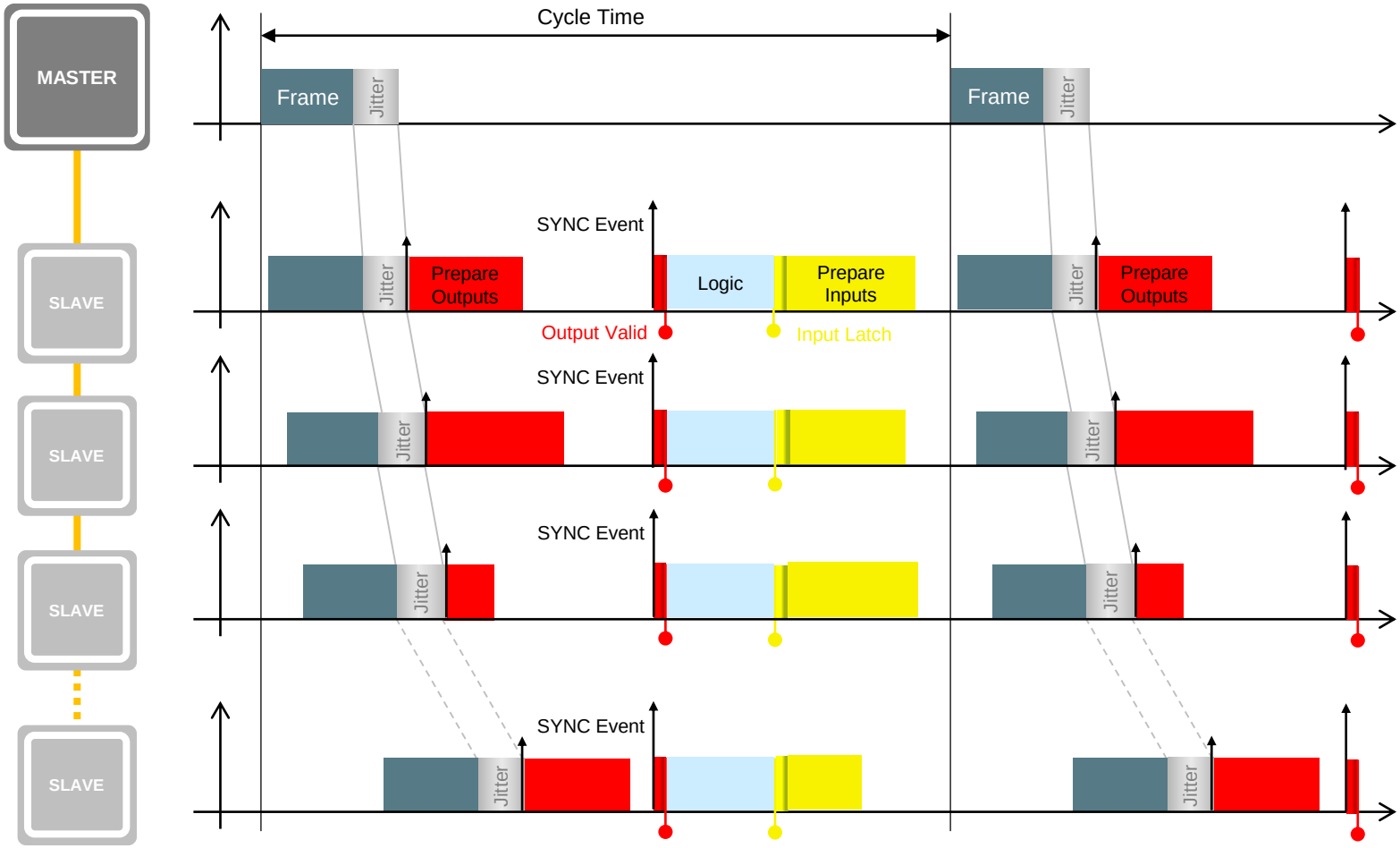
DC-Synchronous. The process data handling in the slave is triggered by the hardware SYNC events generated in the slave based on the DC System Time:

- Based on DC System Time, hardware SYNC events are generated within each slave.
- The triggering event in each slave is not affected by master jitter or propagation delays.
- Intended for e.g. drives controlled by the NC task or oversampling I/O devices.



DC-Synchronous

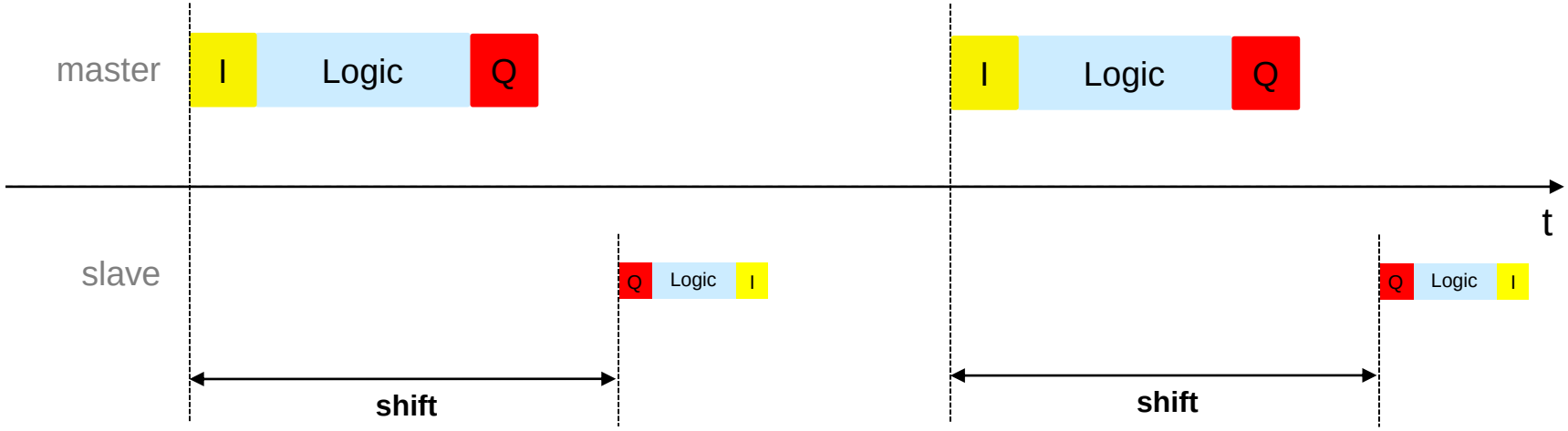
BECKHOFF



When Working in a Synchronous Mode: Time Shift

BECKHOFF

In SM- and DC-synchronous mode a certain **shift** is always needed between the master and slave application start times, in order to be enable the communication partner to receive the data before its cycle begins:

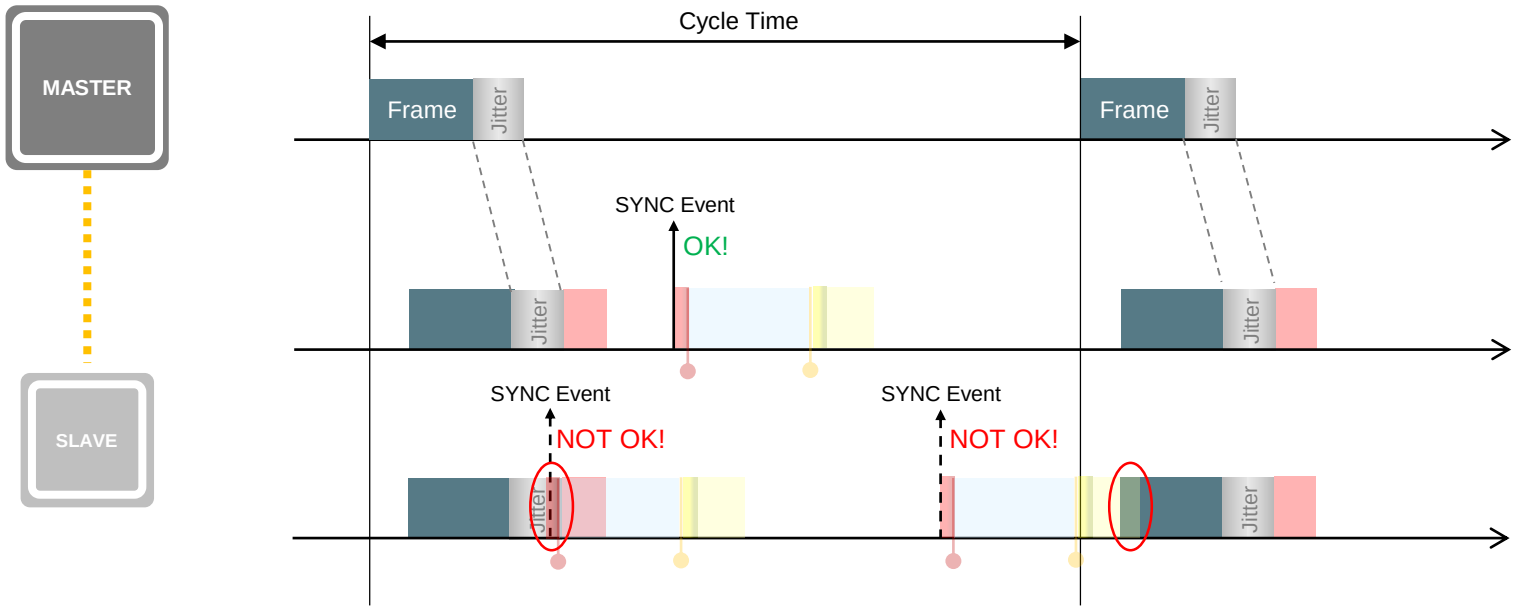


In **SM-Synchronous** mode, the shift is automatically determined by the way itself the synchronization mode works (no parameter configuration is needed).

In **DC-synchronous** mode, the shift between the SYNC interrupt and the master cycle is set by the master during the start-up phase, and can be changed by users if needed.

DC-Synchronous

BECKHOFF



In DC-Synchronous mode, the shift shall be set in order to guarantee that…

- All slaves are able to receive the cyclic outputs, before the local cycle in each slave is started by the SYNC event.
- All slaves are able to provide cyclic inputs, before the master fetches them with the next frame.

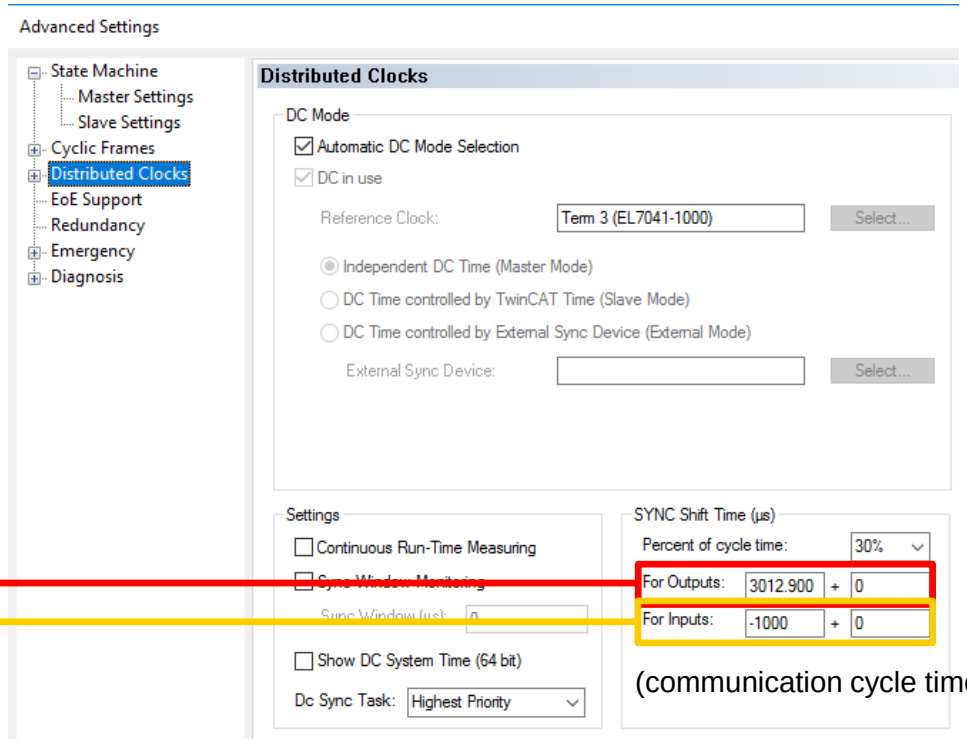
… and this despite communication jitter, propagation delays, number of slaves.

Master SYNC Shift Times

BECKHOFF

The time shift is set for all DC-Synchronous slaves through the **SYNC Shift Time**. Two different SYNC Shift Time parameters can be distinguished:

- **For Outputs**: typically used for slaves supporting cyclic outputs in their process image
- **For Inputs**: typically used for input-only slaves (no cyclic outputs in the process image)



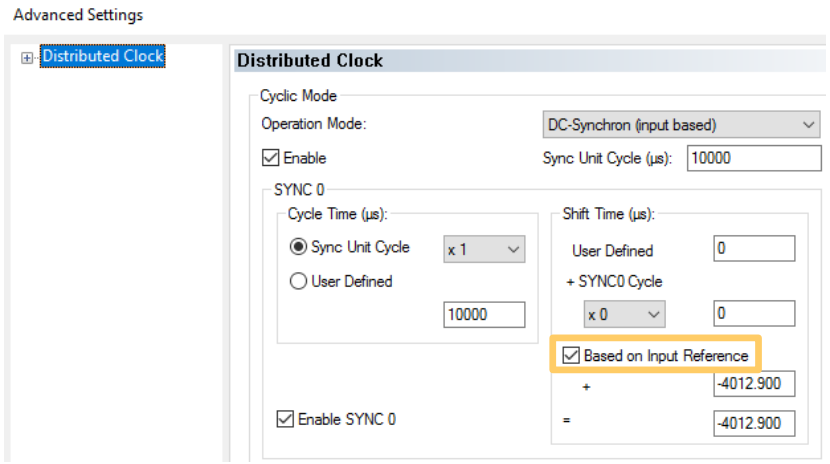
(communication cycle time = 10 ms)

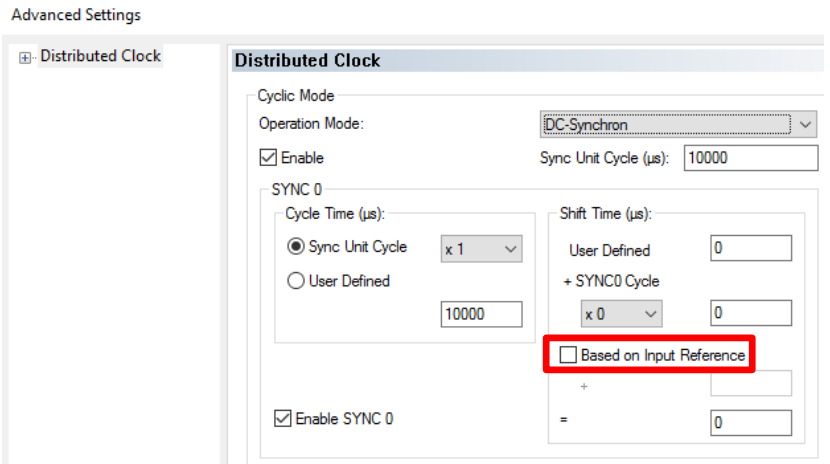
SYNC Shift Time for Outputs or for Inputs

BECKHOFF

Each DC-Synchronous slave is configured **either** using the SYNC Shift Time for Outputs **or** the SYNC Shift Time for Inputs, never with both of them. Which SYNC Shift Time is used is reported in the ESI file and shown in the TwinCAT among the slave settings.

- If flag "**Based on Input Reference**" in the slave Advanced Settings is set by default, the master will use the SYNC Shift Time for Inputs when initializing the slave …
- … otherwise the SYNC Shift Time for Outputs will be used.

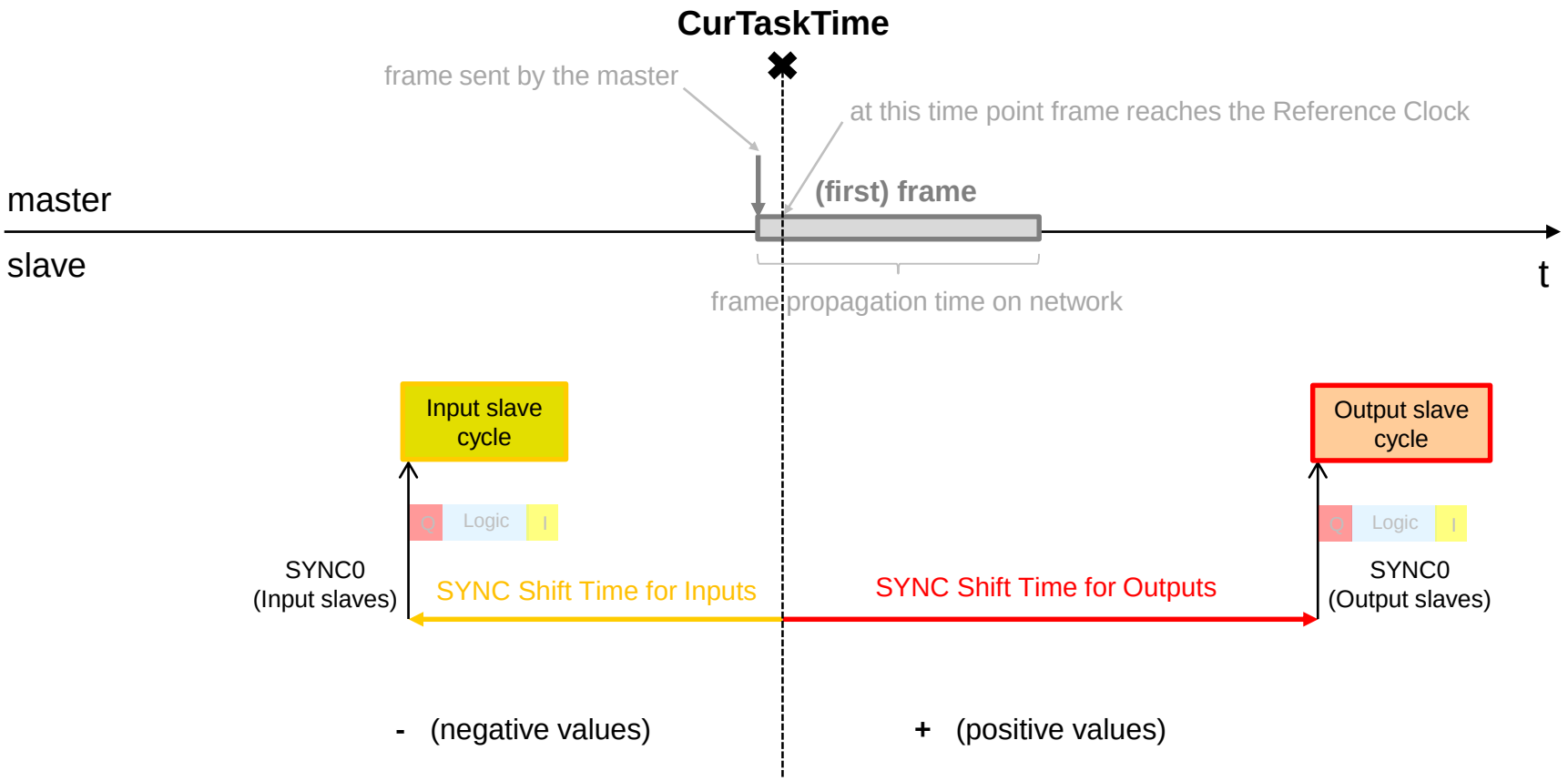




Reference Point for SYNC Shifts Times

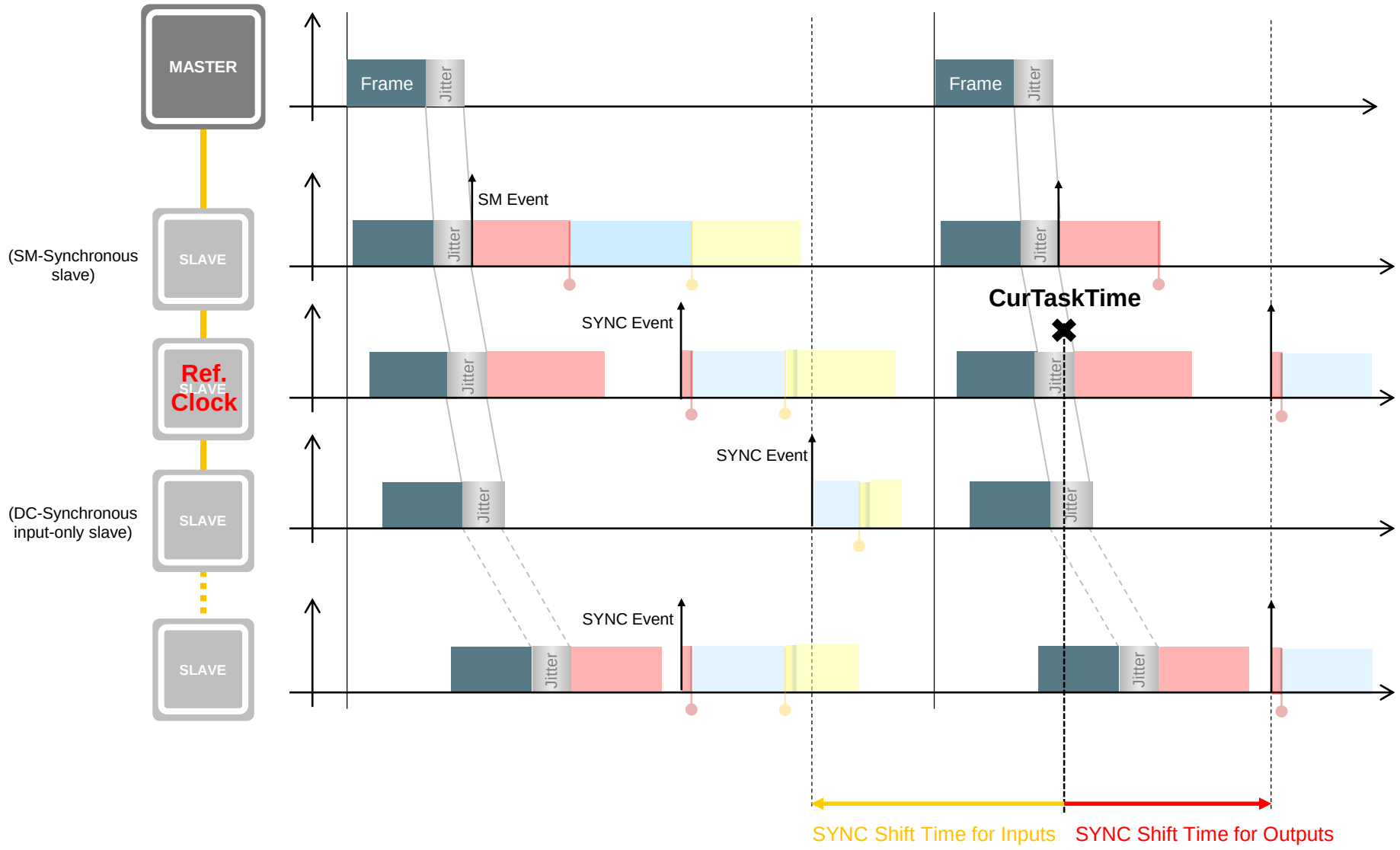
BECKHOFF

The **reference point** for the SYNC Shift Time values in TwinCAT is called **CurTaskTime**, and corresponds to the time when the (first) cyclic frame reaches the Reference Clock device (first DC-Synchronous slave in the network):



Reference Point for SYNC Shifts Times

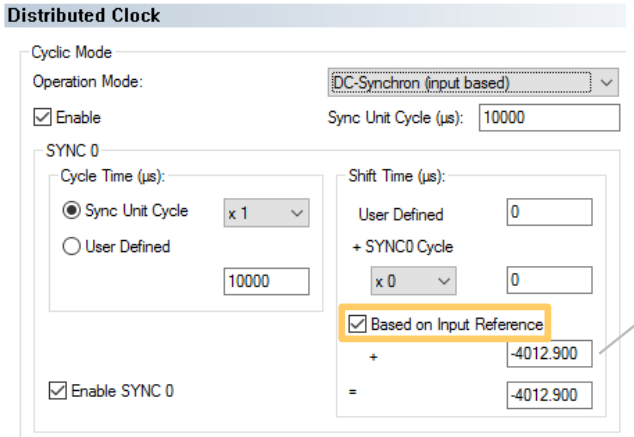
BECKHOFF



Shift Time Value in Slave Advanced Settings

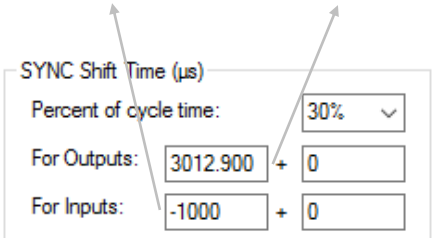
BECKHOFF

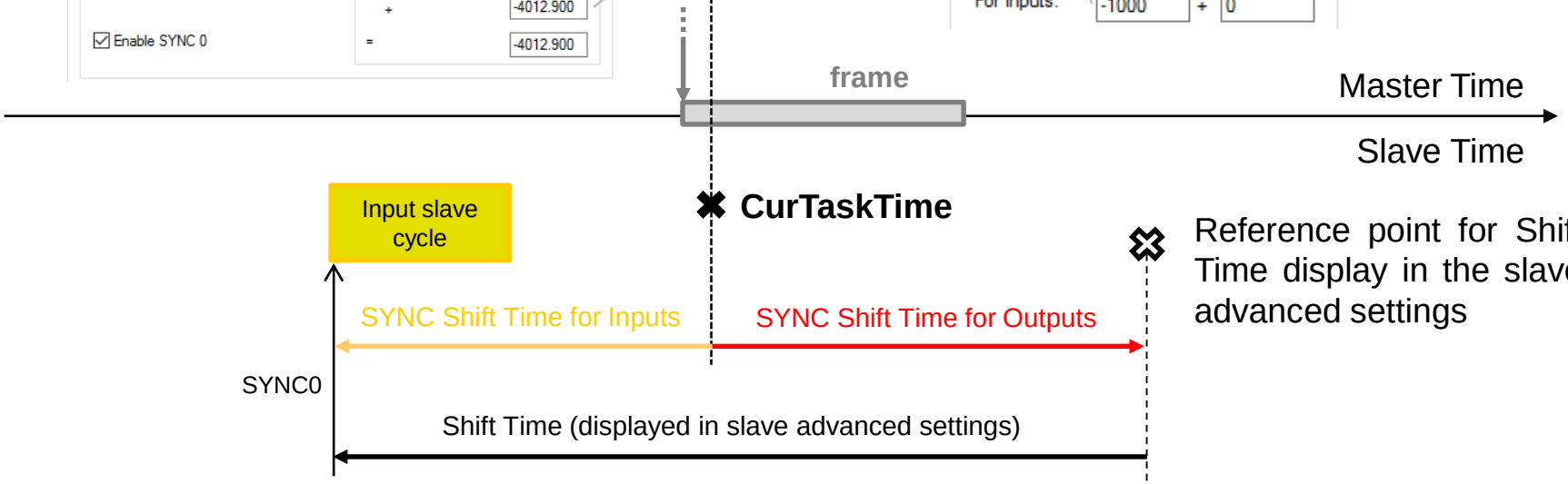
In the **slave** Advanced Settings the Shift Time value is **displayed** with respect to the SYNC Shift Time for Outputs. Yet, taken into account the different reference point, the absolute shift value is consistent with the [master Advanced Settings] (but if the slave has a User Defined contribution):



Example (slave “Based on Input Reference”):

– 4012.900 µs = – 1000.000 µs – (+3012.900 µs)





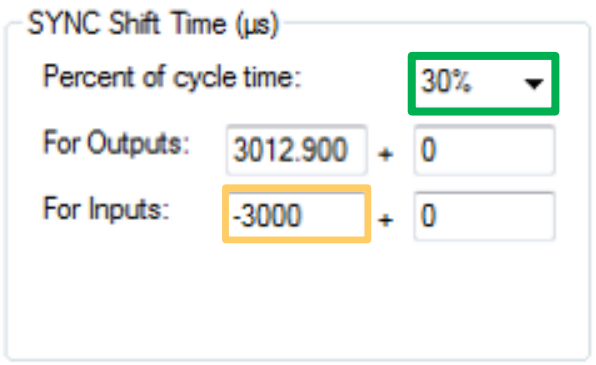
Reference point for Shift Time display in the slave advanced settings

Default Values of SYNC Shift Times

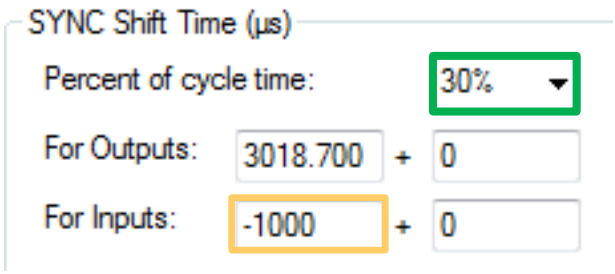
BECKHOFF

The default value for the SYNC Shift Time for Outputs is automatically set by TwinCAT to **30% of** the **communication cycle time** (the corresponding value for SYNC Shift Time for Inputs depends on the TwinCAT version):

TwinCAT 2



TwinCAT 3



This default value is a good trade-off which guarantees proper synchronization in most real applications (no parameter adaptation needed for users).

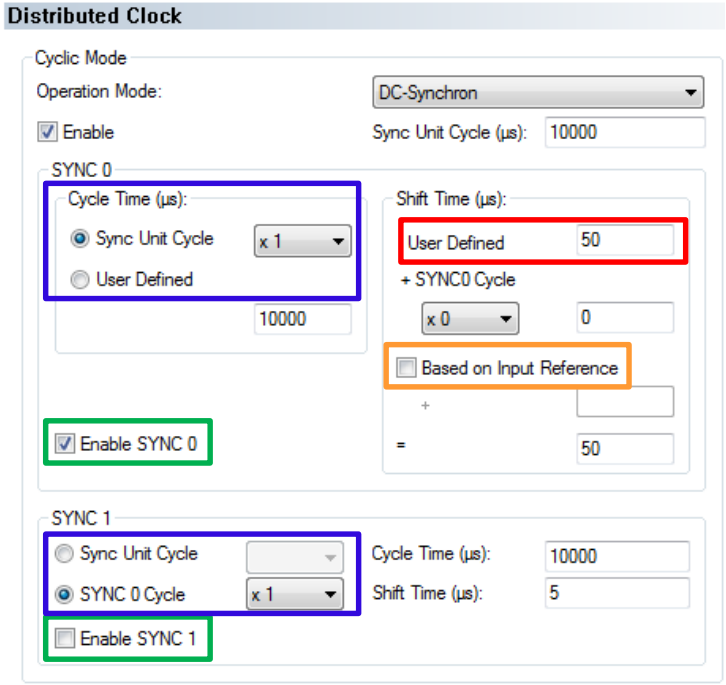
Only for very challenging applications (low cycle times, high number of slaves), or in case synchronization errors show-up after the network has been started, an more precise estimation and in case an adjustment of the default values for SYNC Shift Time should be performed, for example as shown in the following slides.

In Depth - Slave DC Advanced Settings

BECKHOFF

Unlike global SYNC Shift Time parameters in the master settings, most of slave DC Advanced Settings derive their default values from the ESI file: they were determined by the slave manufacturer to guarantee proper operation, and should **never** be **changed** by users.

Which periodicity each enabled SYNC signal shall have.



Additional contribution to SYNC Shift Time which will be added by master to the global one when the slave is initialized.

Which SYNC signals are enabled and which are not

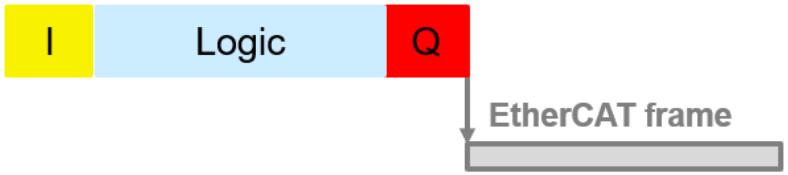
Whether the master uses the SYNC Shift Time for Outputs or the SYNC Shift Time for Inputs when the slave is initialized.

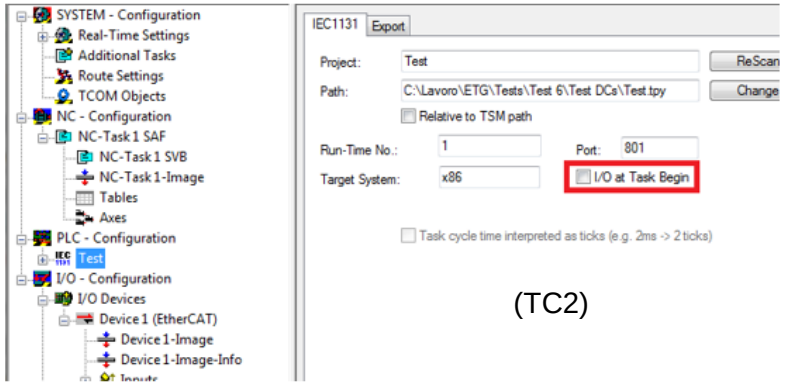
TwinCAT Settings - I/O at Task End and I/O at Task Begin

BECKHOFF

EtherCAT frames can be sent by TwinCAT with two different time relationships to the task:

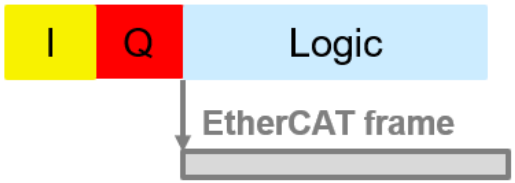
I/O at Task End

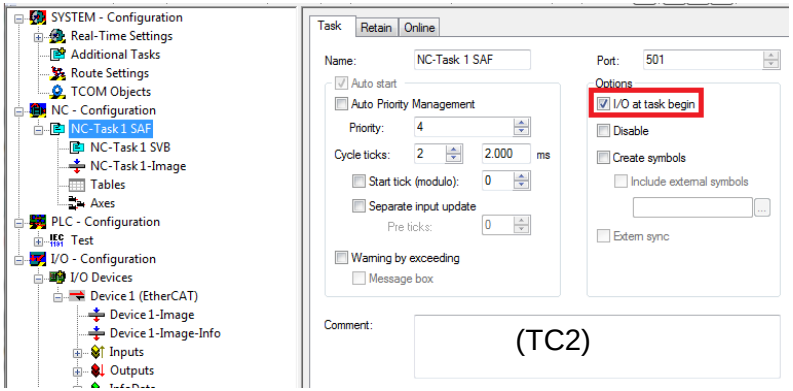




(TC2)

I/O at Task Begin





(TC2)

- “I/O at Task Begin” is the default for the NC task (and should not be changed)
- “I/O at Task End” is the default for PLC tasks as well as IO tasks

In TwinCAT 3, neither PLC Tasks nor C++ Modules provide the flag “I/O at Task Begin”. In these case, the same behaviour can be obtained by using the **TcCallAfterOutputUpdate** attribute:

PLC

Attribute TcCallAfterOutputUpdate

The pragma **{attribute 'TcCallAfterOutputUpdate'}** defines whether a program is to be executed after an output update. This attribute replaces the TwinCAT 2 functionality of the option "IO at Task begin".

Syntax:

{attribute 'TcCallAfterOutputUpdate' }

This attribute must be added to all program POUs, which are to be called after the output update.

Example:

```
{attribute 'TcCallAfterOutputUpdate'}
PROGRAM MAIN
VAR
END_VAR
```

C/C++

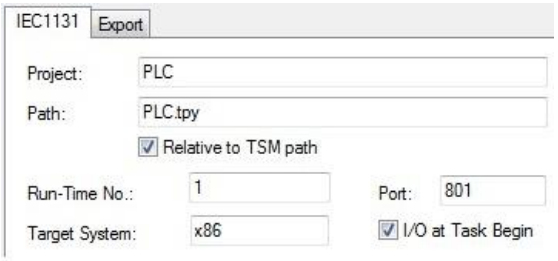
TcCallAfterOutputUpdate for C++ modules

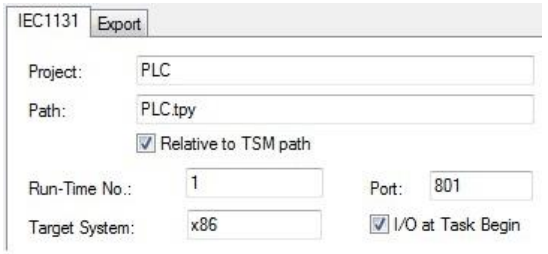
Comparable to PLC Attribute TcCallAfterOutputUpdate C++ modules could be called after the output update.
Equivalent to the ITcCyclic Interface: Please make use of the ITcPostCyclic Interface

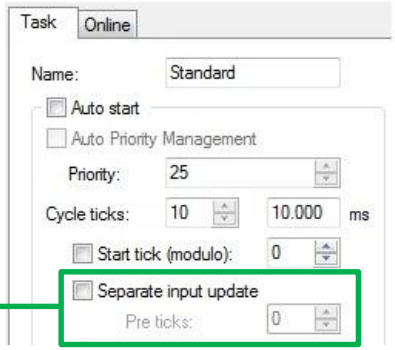
TwinCAT Settings – Separate Input Update

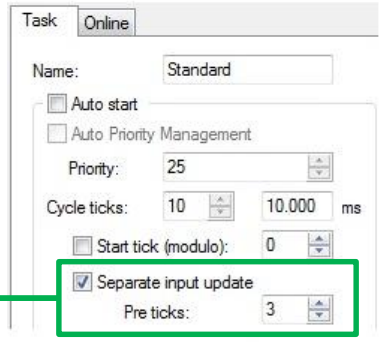
BECKHOFF

TwinCAT can optionally send a separate frame to collect the cyclic inputs. If “**Separate input update**” is enabled, a separate frame will be configured to read input process data (this setting can be used to reduce the input-output reaction times):







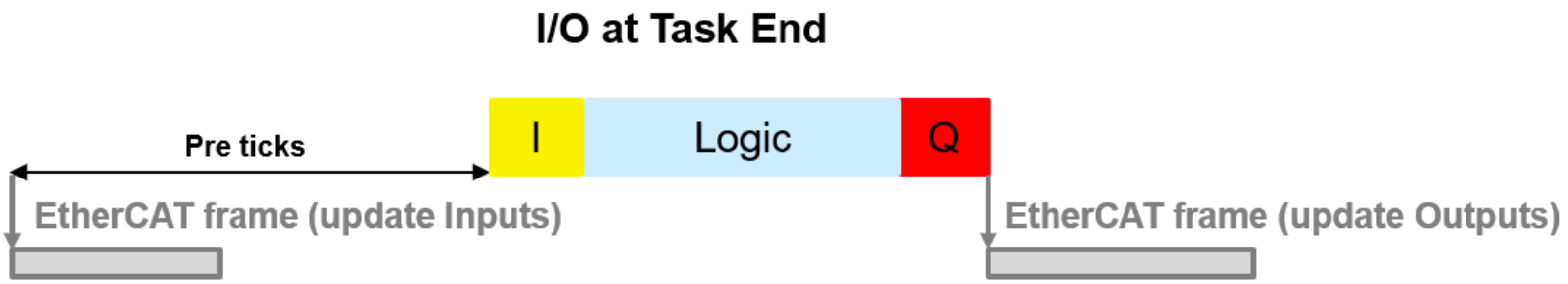


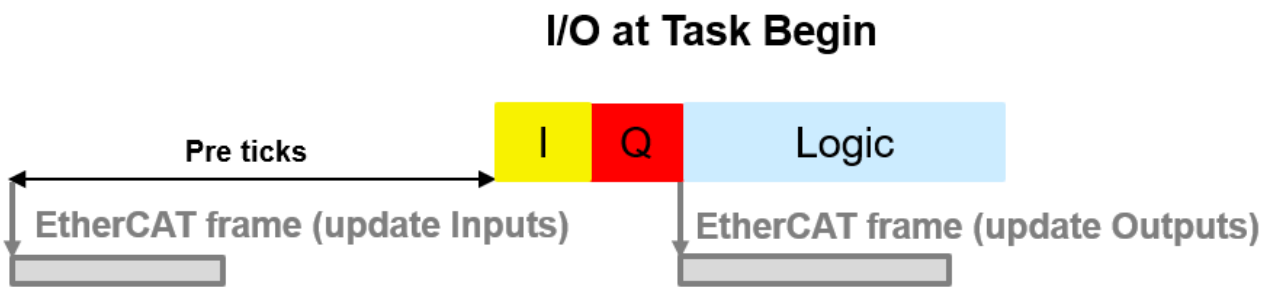
Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)
0	LWR	0x01000000	1	1	<default>	10.000
0	LRD	0x01000800	1	1	<default>	10.000
0	BRD	0x0000 0x0130	2	3		10.000

Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)
0	LRD	0x01000000	1	1	<default>	10.000
1	LWR	0x01000800	1	1	<default>	10.000
1	BRD	0x0000 0x0130	2	3		10.000

BECKHOFF

The **Pre ticks** parameter is expressed as multiple integer of TwinCAT Base Time, and indicates how much earlier the input data will be fetched before the software task starts.



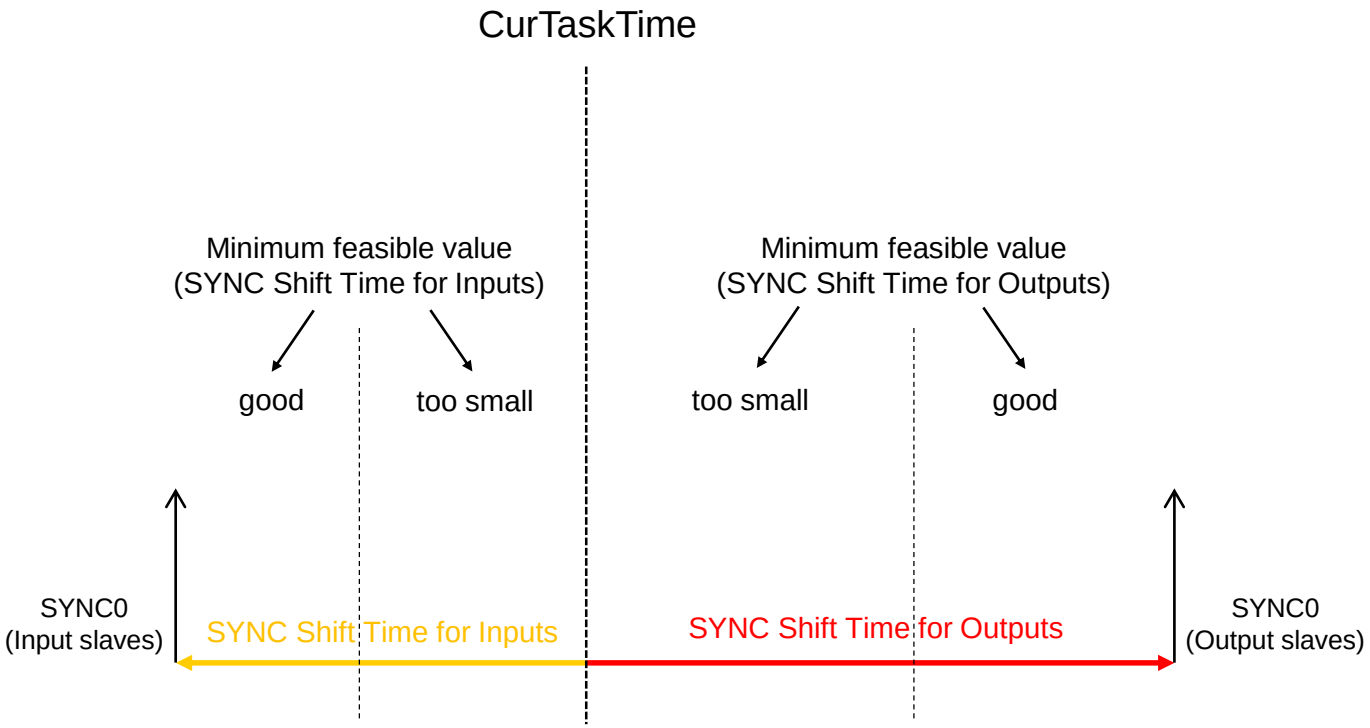


The TwinCAT Base Time can be reduced in order to obtain a finer granularity in the Pre ticks.

Estimation Example of SYNC Shift Times

BECKHOFF

How is it possible to perform a quick, rule-of-thumb estimation of minimum feasible corner values for the SYNC Shift Times?



Estimation Example of “SYNC Shift for Outputs”

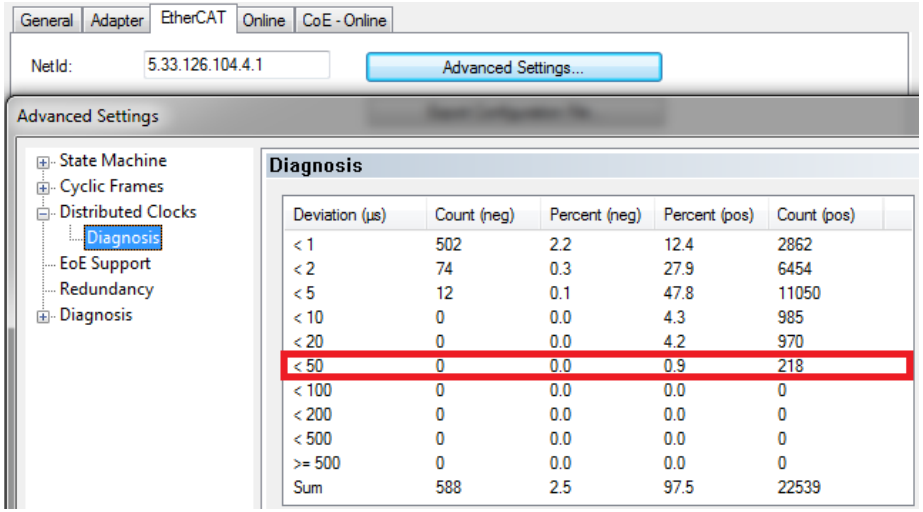
BECKHOFF

Whenever an optimization is needed, an estimation of the minimum value for the “SYNC Shift for Outputs” can be obtained as **algebraic sum** of different contributions:

+ Hardware delay introduced by the slaves internally (starting from Reference Clock):
  - 1 µs * number of slaves with MII Ports + 0,3 µs * number of slaves with only EBUS Ports

+ Hardware delay introduced by the cables (starting from Reference Clock):
  - 5,3 ns/m * total length of the copper cables in the network [expressed in m] * 2

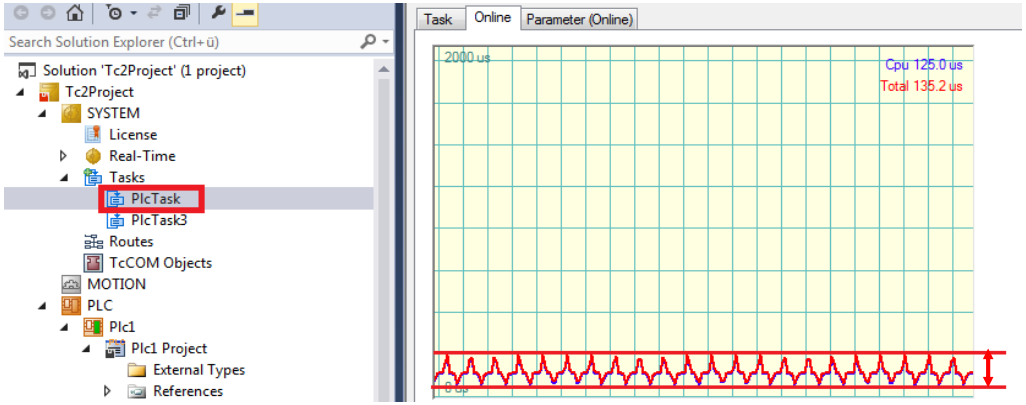
+ Worst case for master jitter (positive deviations):



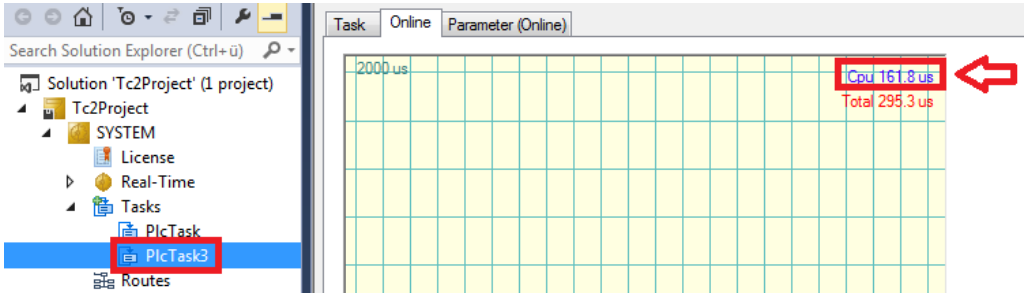
Estimation Example of “SYNC Shift for Outputs”

BECKHOFF

+ Worst case execution jitter of the first software task (only if this has “I/O at Task End”).



+ In multi-task configurations, Σ execution times of tasks from the second to the last having links with DC-Synchronous slaves (cycle time of the first task shall be included in the sum as well only if the first task is configured as “I/O at Task Begin”).



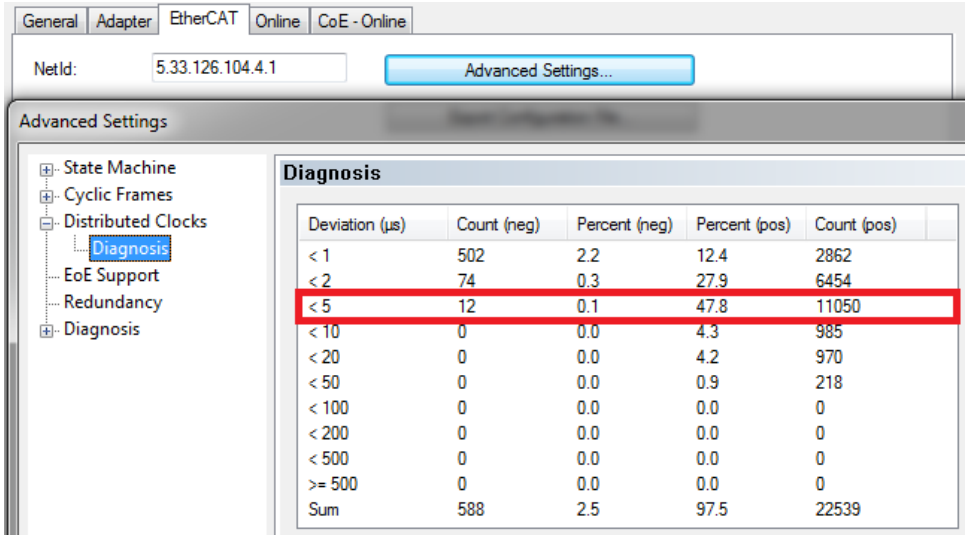
+ Certain tolerance margin…

Estimation Example of “SYNC Shift for Inputs”

BECKHOFF

When needed, an estimation of the minimum value for the “SYNC Shift for Inputs” can be obtained as **algebraic sum** of different contributions :

- Worst case for master jitter (negative deviations):



Deviation (µs)	Count (neg)	Percent (neg)	Percent (pos)	Count (pos)
< 1	502	2.2	12.4	2862
< 2	74	0.3	27.9	6454
< 5	12	0.1	47.8	11050
< 10	0	0.0	4.3	985
< 20	0	0.0	4.2	970
< 50	0	0.0	0.9	218
< 100	0	0.0	0.0	0
< 200	0	0.0	0.0	0
< 500	0	0.0	0.0	0
>= 500	0	0.0	0.0	0
Sum	588	2.5	97.5	22539

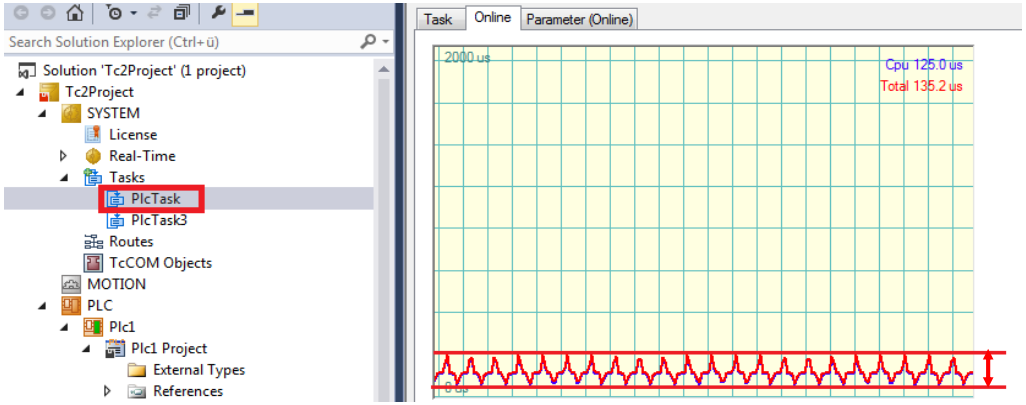
- Highest value among the minimum cycle times (0x1C33:05) for those slaves which work “Based on Input Reference”.

- Time value corresponding to the Pre ticks (only if “Separate input update” is active).

Estimation Example of “SYNC Shift for Inputs”

BECKHOFF

- Worst case execution jitter for the first software task (only if the first task is configured as “I/O at Task End”).



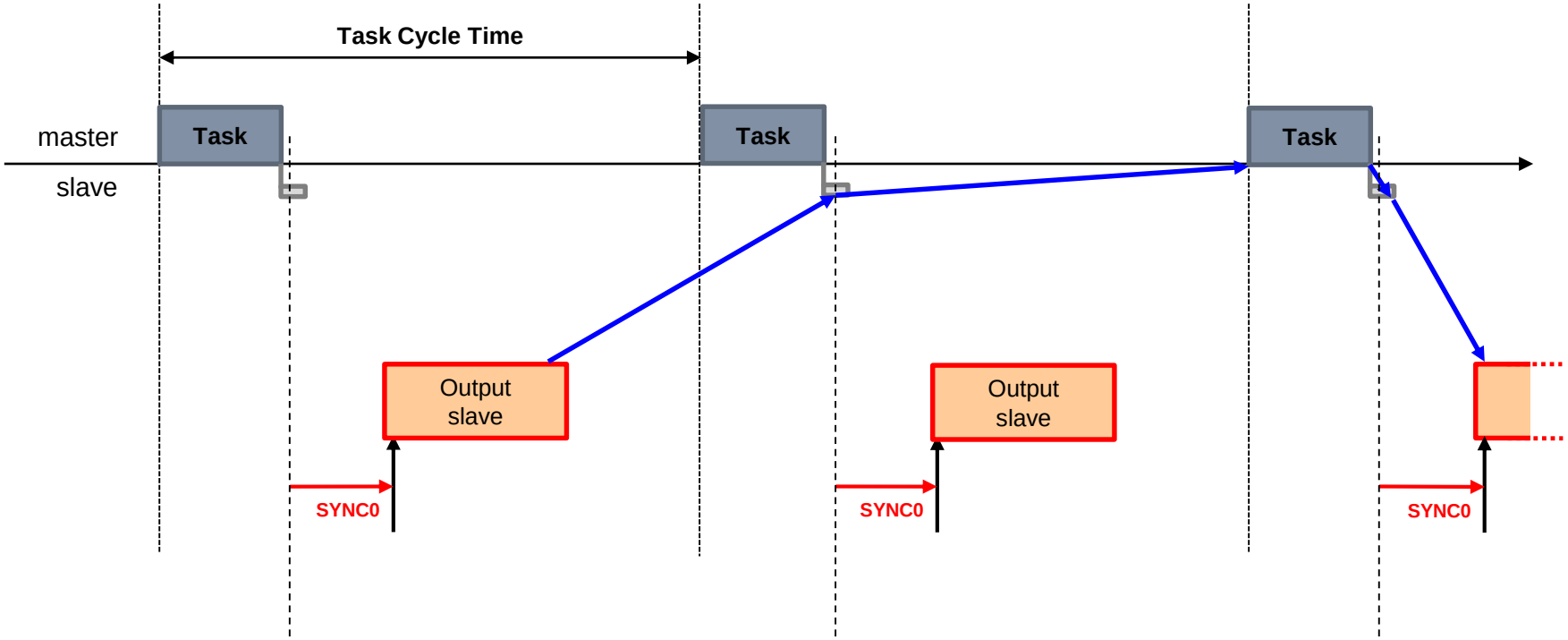
- Certain tolerance margin…

In Depth - Timing Example 2

BECKHOFF

This example corresponds to the following settings:

- Task with “I/O at Task End”
- inputs read and outputs written by an input+output slave

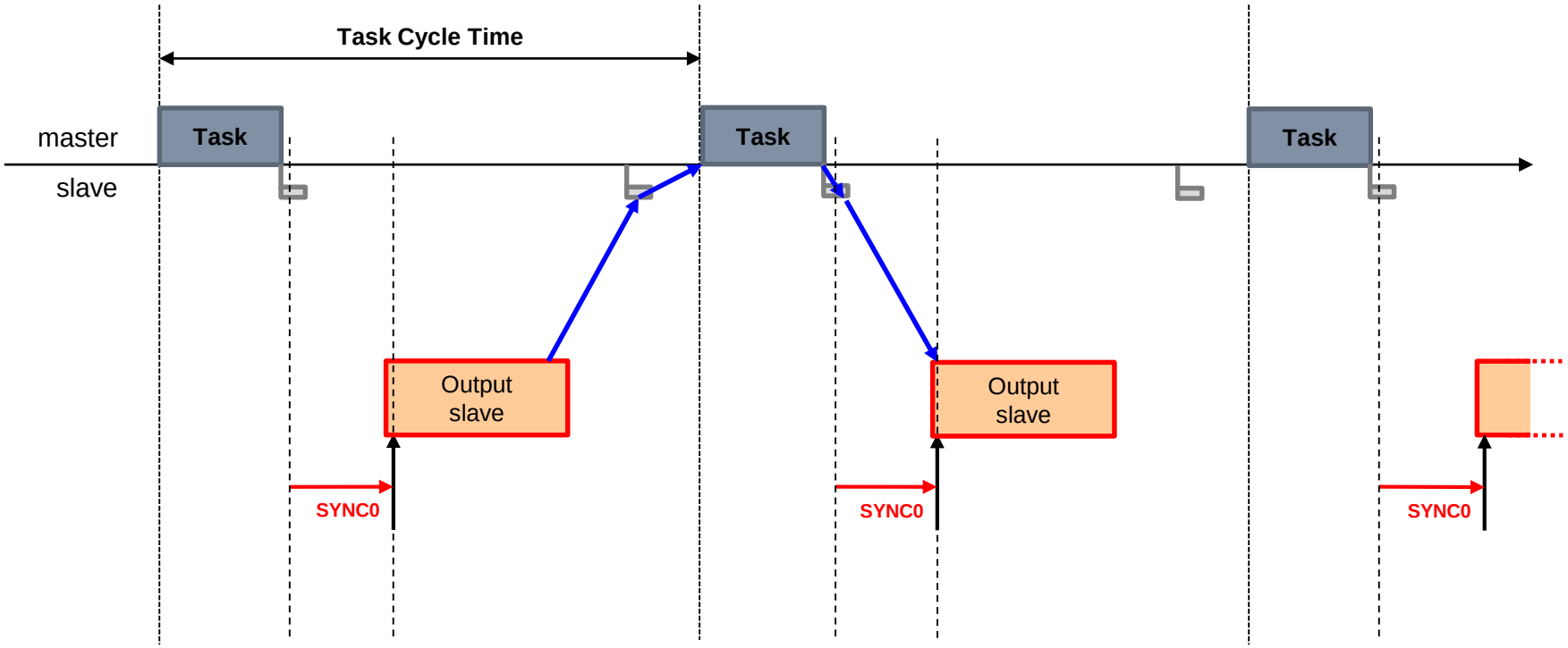


In Depth - Timing Example 3

BECKHOFF

This example corresponds to the following settings:

- Task with “I/O at Task End” with “Separate Input Update”
- inputs read and outputs written by an input+output slave

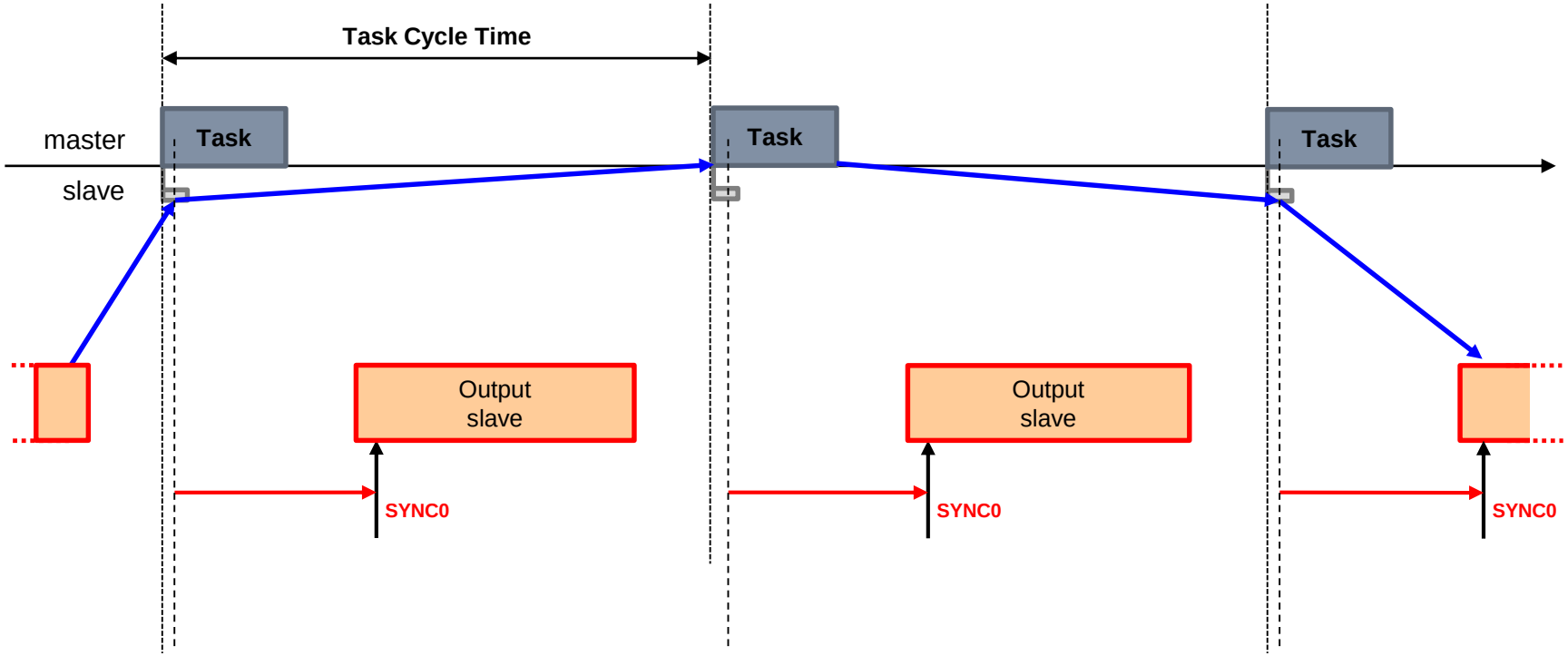


In Depth - Timing Example 5

BECKHOFF

This example refers to the following conditions:

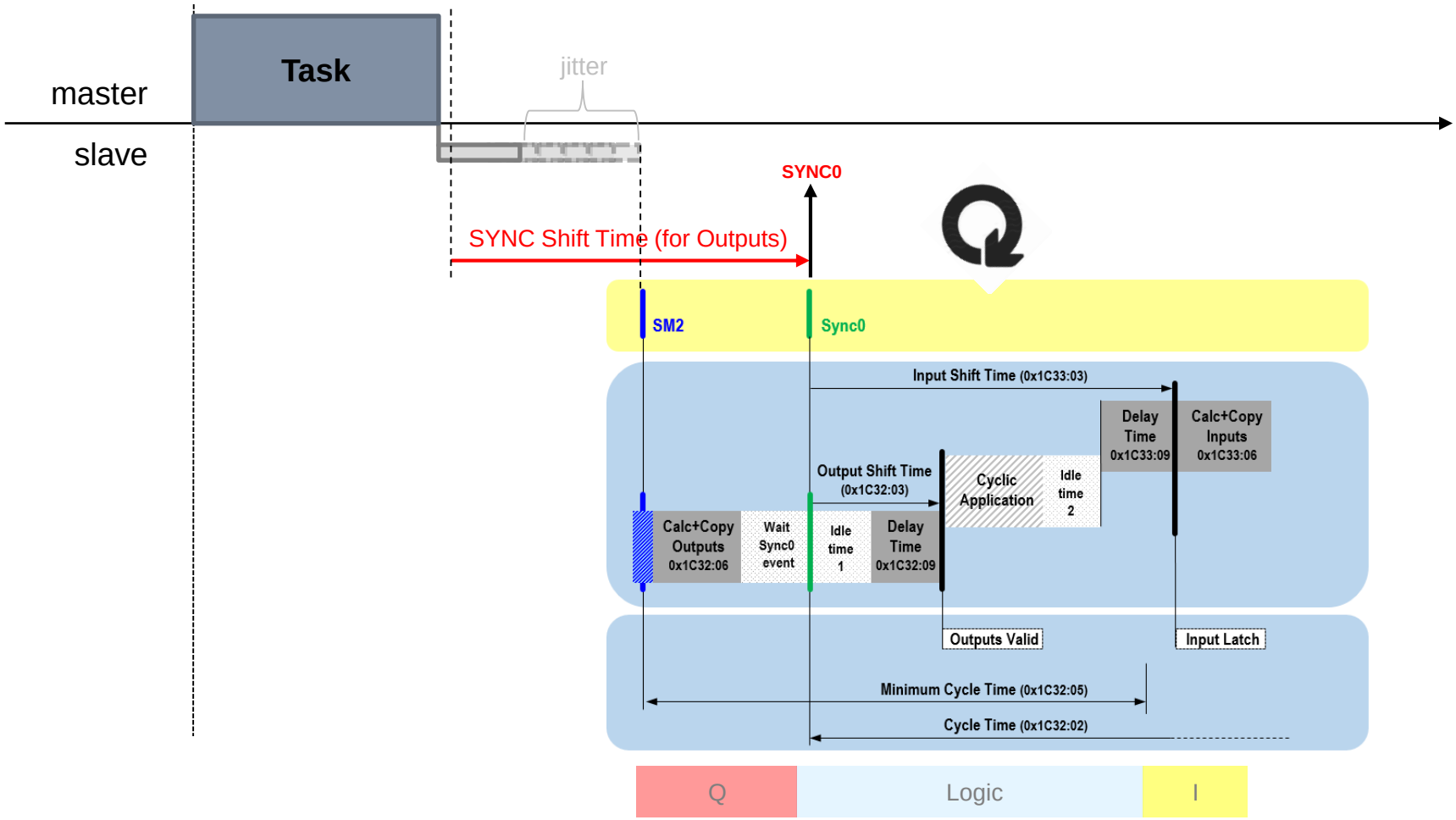
- Task with “I/O at Task Begin”
- inputs read and outputs written by an input+output slave



SYNC Shift Time and Slave Internal Timings

BECKHOFF

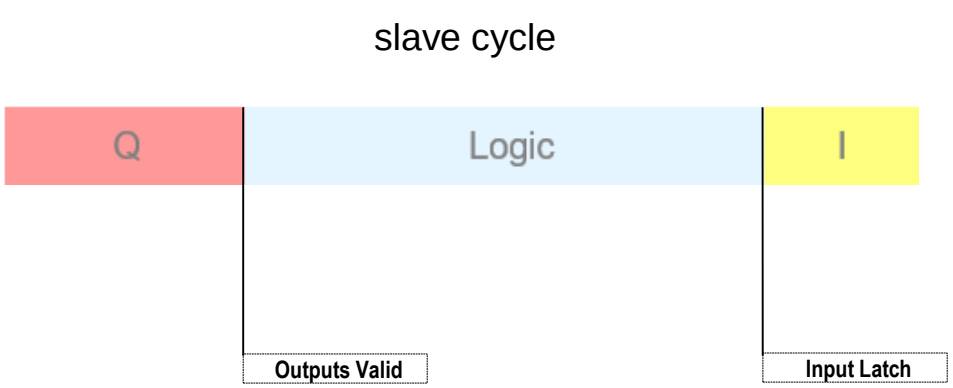
Each DC-Synchronous slave defines **internal timings** for local events. The reference point for the local timings is the SYNC event, and this is in turn configured via the SYNC Shift Time:



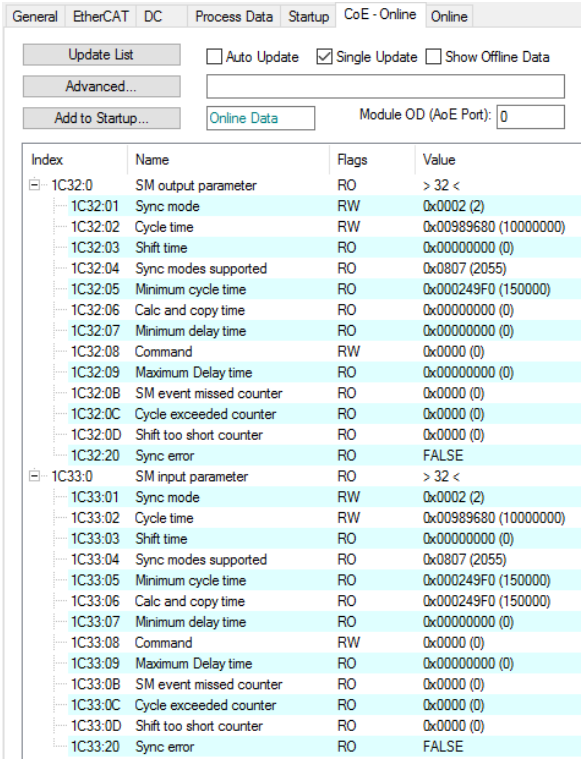
Synchronization Objects 0x1C32/33

In particular, the two most important local events within the internal slave cycle are:

- **Output Valid**: instant when the cyclic outputs received from the master in the last communication cycle are physically set in the field.
- **Input Latch**: instant when the cyclic inputs are physically read from the field before being transmitted to the master in the next communication cycle.



CoE slaves describe internal timings via standard objects **0x1C32** (for cyclic outputs) and **0x1C33** (for cyclic inputs):



Index	Name	Flags	Value
1C32:0	SM output parameter	RO	> 32 <
1C32:01	Sync mode	RW	0x0002 (2)
1C32:02	Cycle time	RW	0x00989680 (10000000)
1C32:03	Shift time	RO	0x00000000 (0)
1C32:04	Sync modes supported	RO	0x0807 (2055)
1C32:05	Minimum cycle time	RO	0x000249F0 (150000)
1C32:06	Calc and copy time	RO	0x00000000 (0)
1C32:07	Minimum delay time	RO	0x00000000 (0)
1C32:08	Command	RW	0x0000 (0)
1C32:09	Maximum Delay time	RO	0x00000000 (0)
1C32:0B	SM event missed counter	RO	0x0000 (0)
1C32:0C	Cycle exceeded counter	RO	0x0000 (0)
1C32:0D	Shift too short counter	RO	0x0000 (0)
1C32:20	Sync error	RO	FALSE
1C33:0	SM input parameter	RO	> 32 <
1C33:01	Sync mode	RW	0x0002 (2)
1C33:02	Cycle time	RW	0x00989680 (10000000)
1C33:03	Shift time	RO	0x00000000 (0)
1C33:04	Sync modes supported	RO	0x0807 (2055)
1C33:05	Minimum cycle time	RO	0x000249F0 (150000)
1C33:06	Calc and copy time	RO	0x000249F0 (150000)
1C33:07	Minimum delay time	RO	0x00000000 (0)
1C33:08	Command	RW	0x0000 (0)
1C33:09	Maximum Delay time	RO	0x00000000 (0)
1C33:0B	SM event missed counter	RO	0x0000 (0)
1C33:0C	Cycle exceeded counter	RO	0x0000 (0)
1C33:0D	Shift too short counter	RO	0x0000 (0)
1C33:20	Sync error	RO	FALSE

Synchronization Objects 0x1C32/33

BECKHOFF

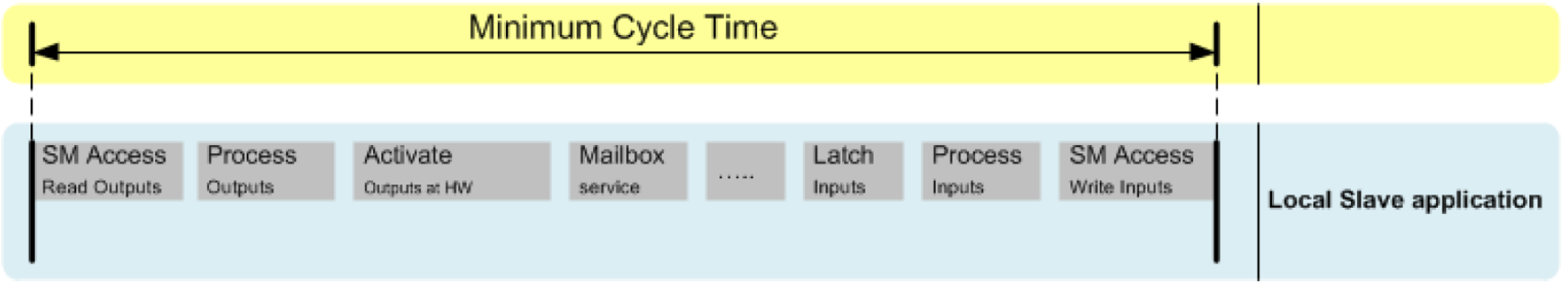
The most **relevant subindexes** in objects 0x1C32/33 (listed in logical, not in progressive order) are:

Subindex	Name	Which information is provided to master/user?	What can be set by the master/user?
05	Minimum Cycle Time	Which is the smallest communication cycle time which can ensure proper operation for the slave?	
03	Shift Time	Does the slave apply a software shift between the main application triggering event (SM2, SYNC0) and Output Valid/Input Latch?	If subindex writable, the master/user can change the software shift between the main interrupt event and Output Valid/Input Latch.
08	Get Cycle Time		If Dynamic Cycle Times supported (see Subindex :04), writing value 1 triggers a dynamic measurement of internal timings in the slave (like Minimum Cycle Time).
04	Synchronization Types Supported	Which synchronization modes and shifting options for Output Valid and Input Latch are supported by the slave?	

Minimum Cycle Time (0x1C32/33:05)

BECKHOFF

In order to process all its internal tasks (process data exchange, mailbox communication, state machine handling, application-specific functions), each slave needs a **Minimum Cycle Time** which can depend on several factors like complexity of the application or amount of process data exchanged.



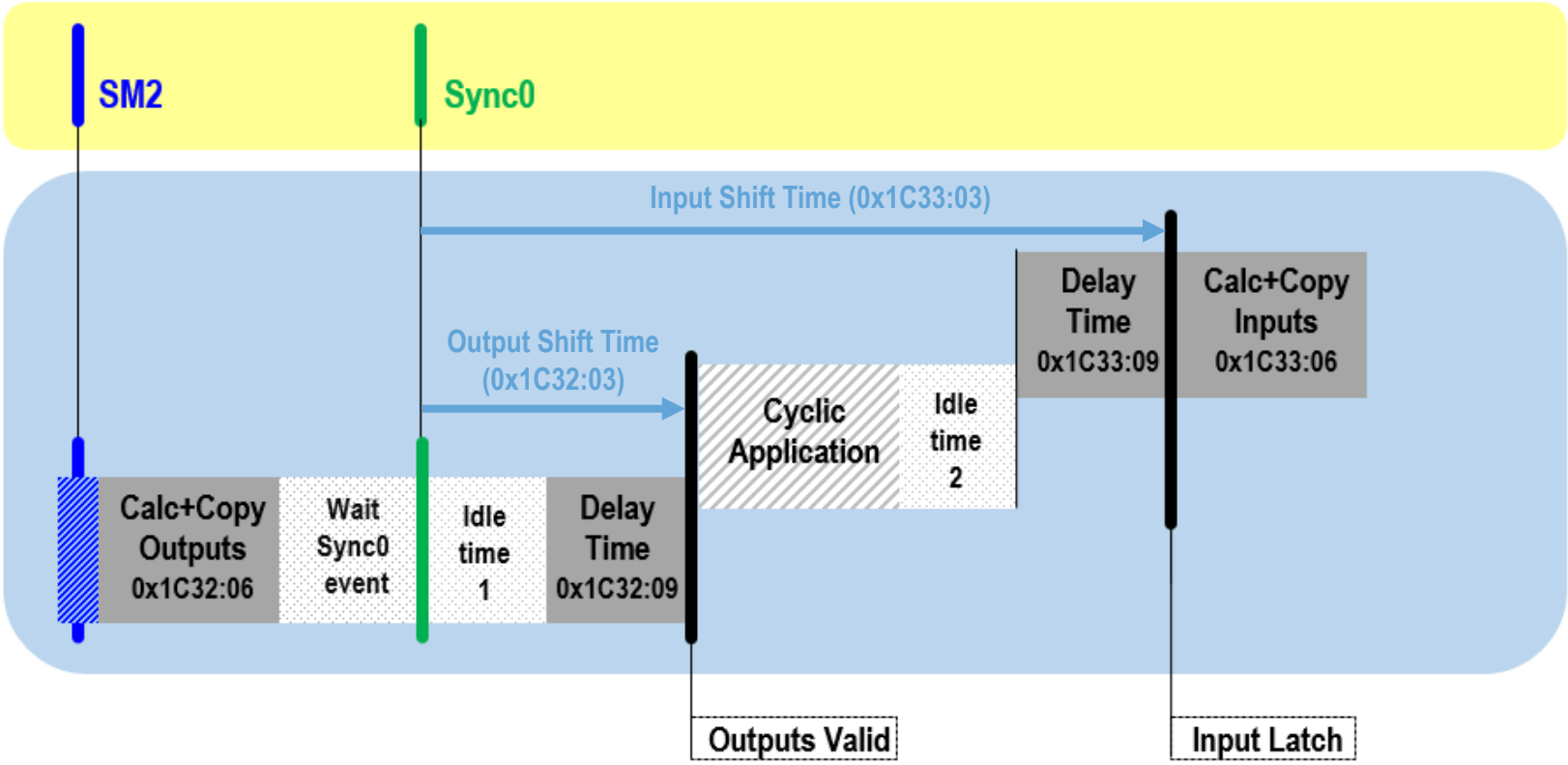
For the task linked to a DC-Synchronous slave it should be guaranteed that Communication Cycle Time ≥ Minimum Cycle Time, otherwise the slave application would not keep the pace.

In case this represents a too heavy limitation for the operation of other (DC- or non DC-Synchronous) slaves, it is suggested to create multiple tasks with different cycle times, and to link the process data of each slave to the task having the suitable cycle time.

Output/Input Shift Time (0x1C32/33:03)

BECKHOFF

Slaves can optionally support a software **Shift Time** of Output Valid and/or Input Latch.



If this subindex is writable, the master/user can finely tune the Output Valid and/or the Input Latching events within the slave cycle (default shift values shall guarantee proper operation, their adjustment is an advanced optimization).

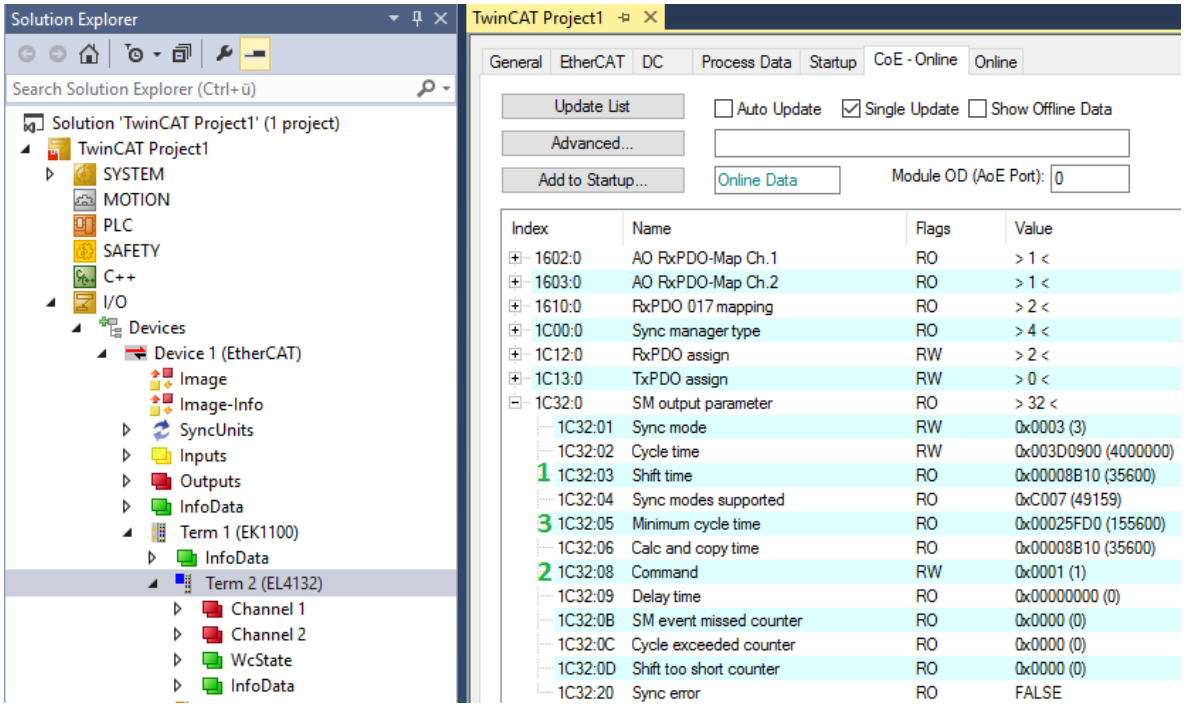
Get Cycle Time (0x1C32/33:08)

BECKHOFF

Some slaves can dynamically calculate their internal timings (e.g. the Minimum Cycle Time), according to the current configuration (for these slaves, bit 14 in 0x1C32/33:04 is set to 1).

The calculation is triggered by setting object **Command** (0x1C32/33:08) to 1:

1. Check if bit 14 0x1C32/33:04 = 1
2. Set 0x1C32/33:08 = 1
3. Timings 0x1C32/33:05+06+09



For slaves supporting Dynamic Cycle Times, triggering the calculation is recommended during commissioning in order to know the slave actual timings.

Synchronization Modes Supported (0x1C32/33:04)

BECKHOFF

Synchronization modes and shifting options supported by a slave are summarized in the **Synchronization Types supported** subindex.

0x1C32 (outputs)

Synchronization Types supported	UNSIGNED16	Bit 0: Free Run supported Bit 1: Synchronous supported Bit 4:2 : DC Type supported: 000 = No DC 001 = DC Sync0 010 = DC Sync1 100 = Subordinated Application with fixed Sync0
	1	
	2	Bit 6:5: Shift Settings 00 = No Output Shift supported 01 = Output Shift with local timer (Shift Time) 10 = Output Shift with Sync1
		Bit 9…6: Reserved for future use Bit 10: Delay Times should be measured (because they depend on the configuration) Bit 11: Delay Time is fix (synchronization is done by hardware) Bit 13…11: Reserved for future use
	3	Bit 14: Dynamic Cycle Times Times described in 0x1C32 are variable (depend on the actual configuration) This is used for e.g. EtherCAT gateway devices. The slave shall support cycle time measurement in OP state. The cycle time measurement is started by writing 1 to 0x1C32:08. If this bit is set, the default values of the times to be measured (Minimum Cycle Time, Calc And Copy Time, Delay Time) could be 0. The default values could be set in INIT and PREOP state. Bit 14 should only be set, if the slave cannot calculate the cycle time after receiving all Start-up SDO in transition PS.
		Bit 15: Reserved for future use

0x1C33 (inputs)

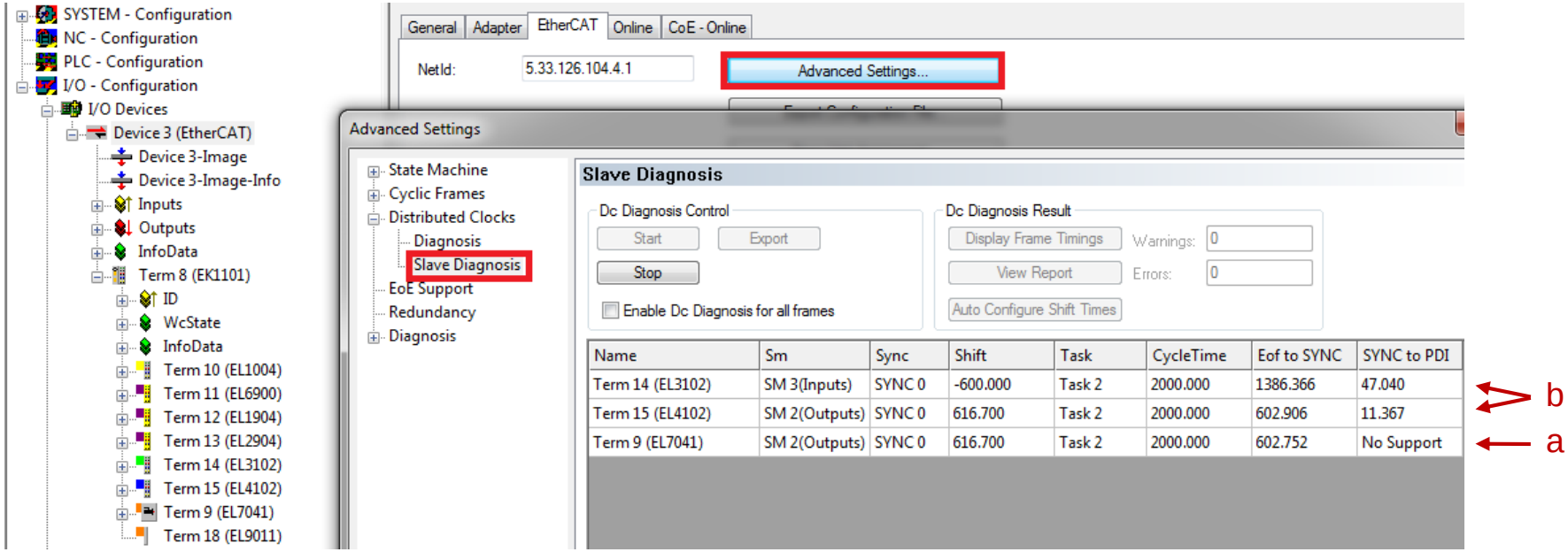
Synchronization Types supported	UNSIGNED16	Bit 0: Free Run supported Bit 1: Synchronous supported Bit 4:2 : DC Type supported: 000 = No DC 001 = DC Sync0 010 = DC Sync1 100 = Subordinated Application with fixed Sync0
	1	
	2	Bit 6:5: Shift Settings 00 = No Input Shift supported 01 = Input Shift with local timer (Shift Time) 10 = Input Shift with Sync1
		Bit 13:6: Reserved for future use
	3	Bit 14: Dynamic Cycle Times Same meaning as in 0x1C32:04
		Bit 15: Reserved for future use

1. Main triggering event for the slave application (identifies synchronization mode).
2. Shifting options for Output Valid/Input Latch.
3. Support for dynamic calculation of internal timings.

Timing Diagnosis (Master)

BECKHOFF

TwinCAT enables to graphically **monitor** the **timing** of all DC-Synchronous slaves, and to check their proper setting (e.g. suitable SYNC Shift Time value):

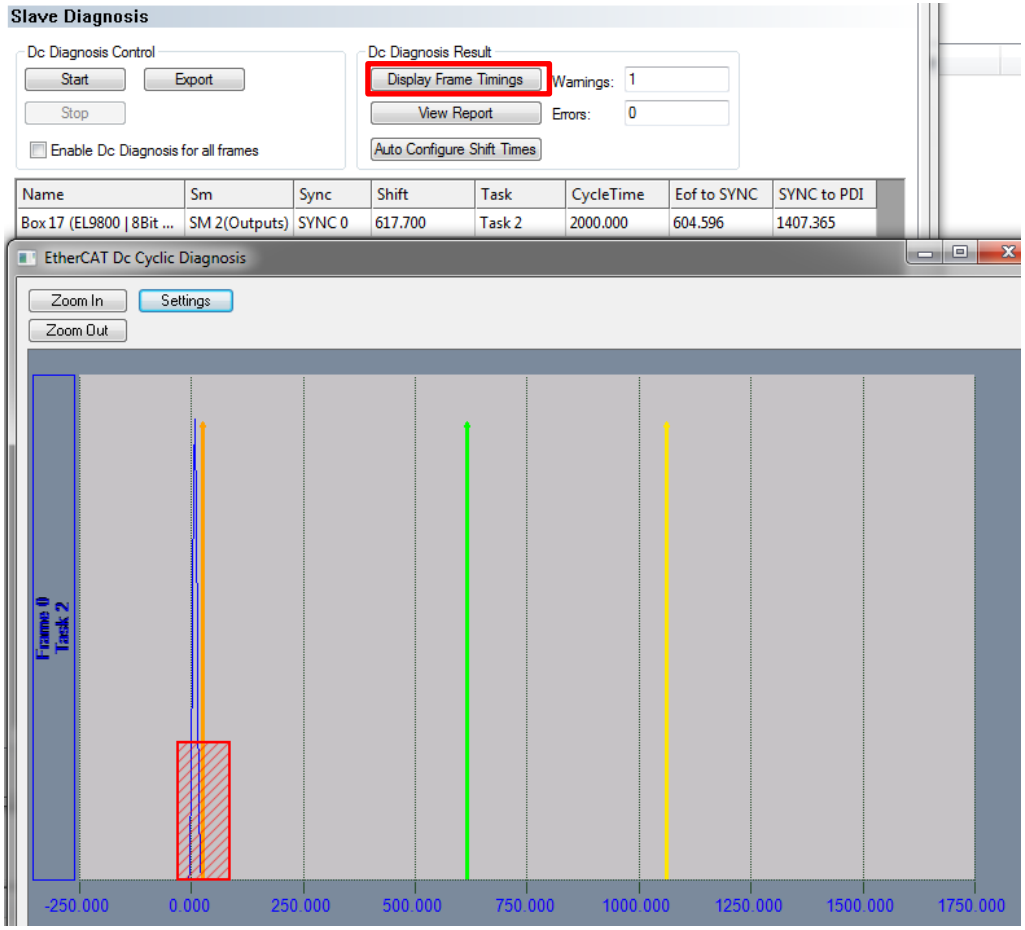


a. Slaves reporting “No Support” in the SYNC to PDI column enable to monitor the time relationship between cyclic frames and SYNC interrupts only.

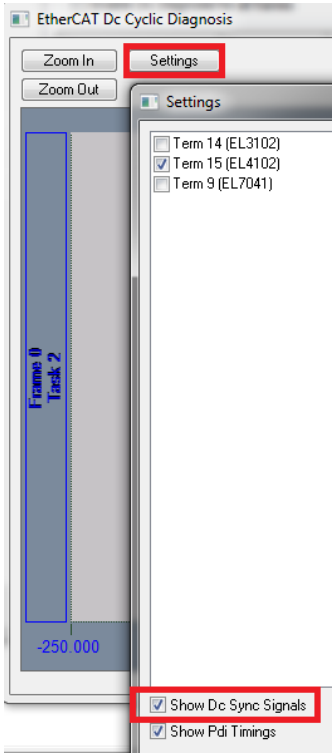
b. Other slaves enable to monitor the internal timings between SYNC interrupts and Output/Input PDI access as well.

Timing Diagnosis (Master)

BECKHOFF

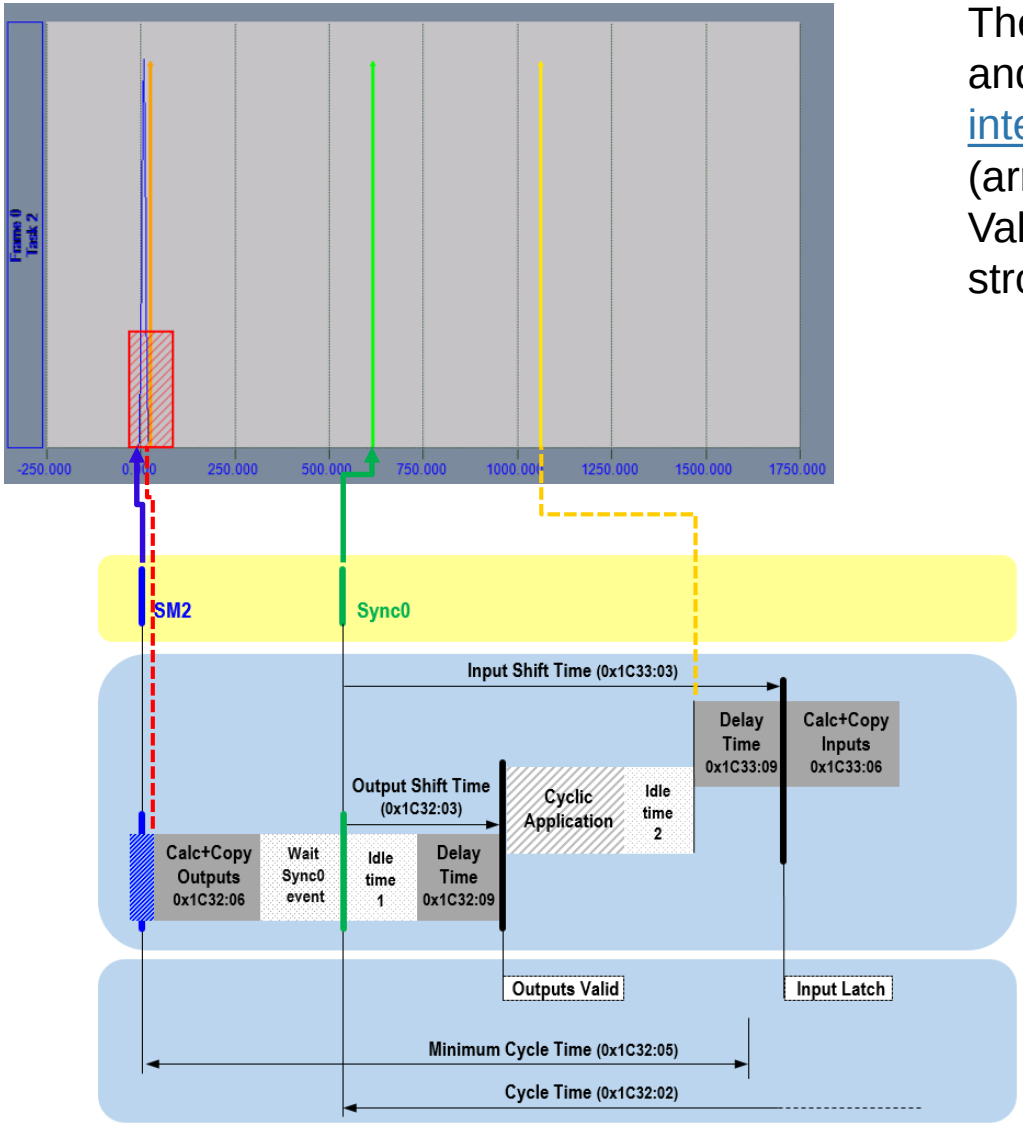


Visualization of SYNC interrupts shall be **manually** enabled:



- Cyclic frame is received (including jitter and propagation delay down to the last displayed slave)
- µC reads cyclic outputs out of ESC
- SYNC interrupt is generated
- µC writes cyclic inputs into ESC

Timing Diagnosis (Master)

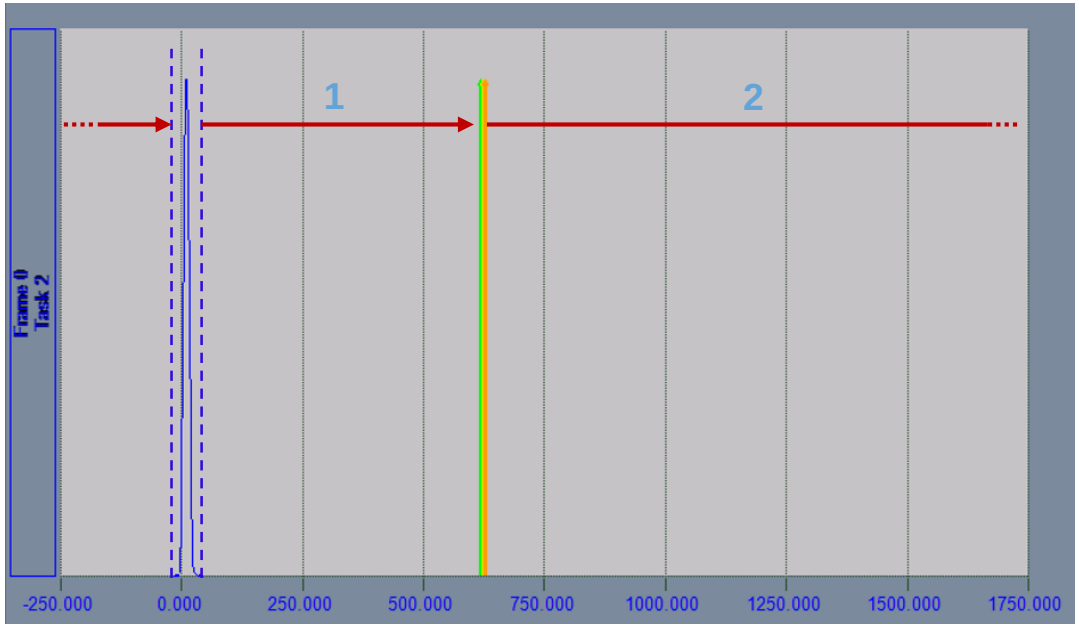


The graphical description corresponds, and can be interpreted according to the internal timing schemes of slave devices (arrows do not correspond to the Output Valid and Input Latch events, yet they are strongly related to them).

Timing Diagnosis (Master)

BECKHOFF

The time diagram is normalized within a communication cycle time.



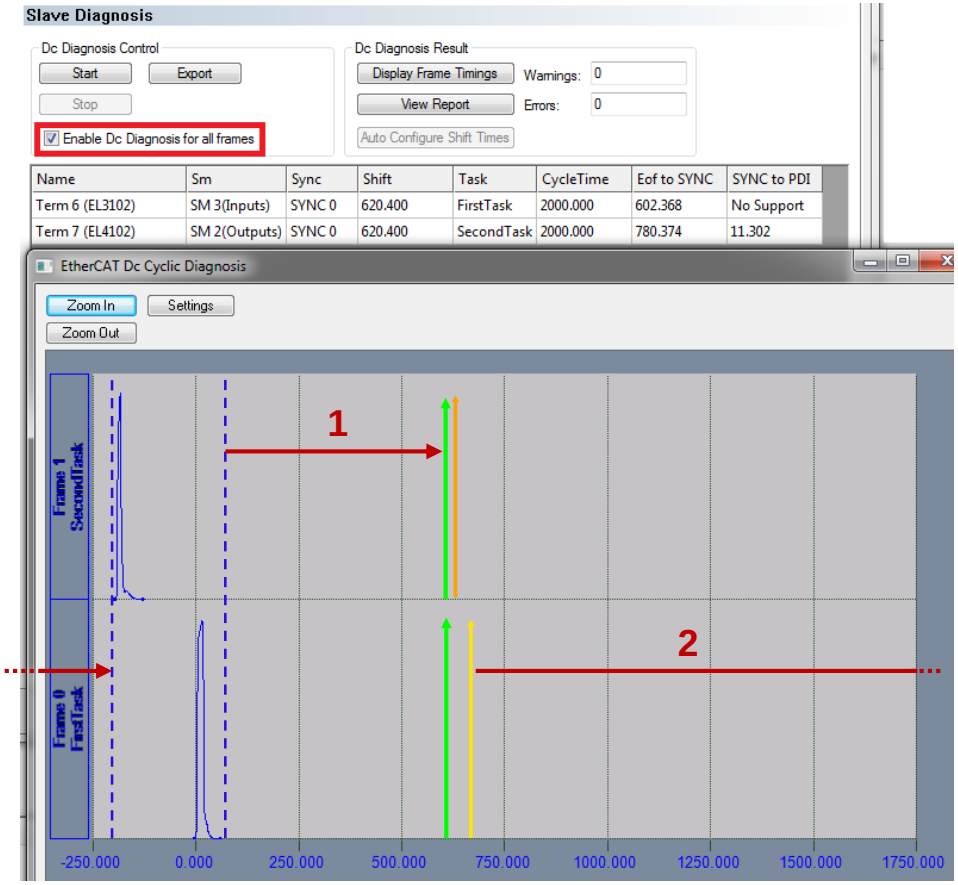
Typically no deep analysis of these timings is necessary: only aspects to be checked are:

1. All DC-Synchronous slaves generate the SYNC interrupt after the cyclic frame was received.
2. All DC-Synchronous slaves perform the last PDI access before the next cyclic frame is received.

Timing Diagnosis (Master)

BECKHOFF

In multi-task configurations (DC-Synchronous slaves linked to different software tasks), timings should be checked for all frames.



In this case:

1. Point 1) of previous slide should be guaranteed for the last cyclic frame carrying data for DC-Synchronous slaves.
2. Point 2) of previous slide should be guaranteed for the first cyclic frame carrying data for DC-Synchronous slaves.

WARNING. The correct time display in case of "Separate Input Update" is at the moment not yet supported (in particular, Pre ticks are not taken into account when displaying the frame fetching cyclic inputs).

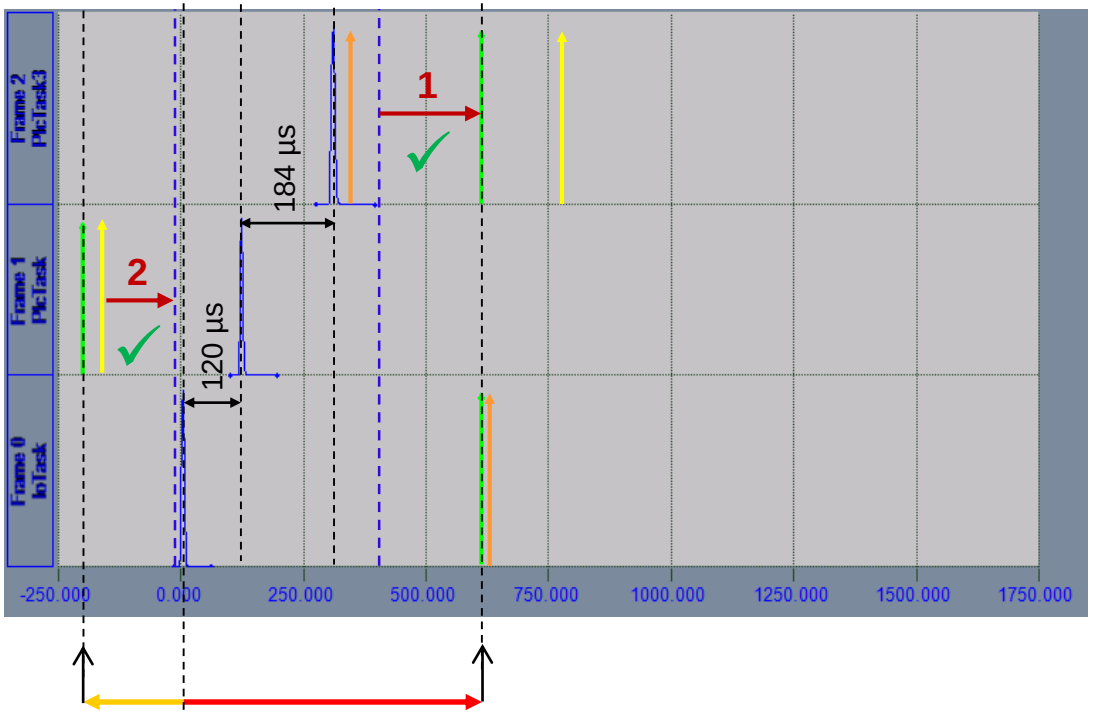
In Depth – Example of Timing Diagnosis (Master)

BECKHOFF

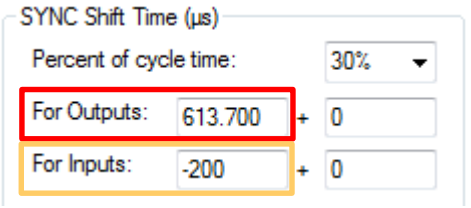
Example of configuration with 3 tasks:

- Task 0 (**IoTask**): cycle-time 2 ms, priority 19, I/O at Task end, actual execution time 16 µs, linked to output-only slave
- Task 1 (**PlcTask**): cycle-time 2 ms, priority 20, I/O at Task end, actual execution time 120 µs, linked to input-only slave
- Task 2 (**PlcTask3**): cycle-time 2 ms, priority 21, I/O at Task end, actual execution time 184 µs, linked to output+input slave

Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)
0	NOP	0x0000 0x0900	4			2.000
0	ARMW	0x0000 0x0910	4			2.000
0	LRD	0x09000000	1			2.000
0	LRW	0x01000000	8	3	<default>	2.000
1	ARMW	0x0000 0x0910	4			2.000
1	LRD	0x02000000	8	1	<default>	2.000
2	ARMW	0x0000 0x0910	4			2.000
2	LRW	0x03000000	16	9	<default>	2.000
2	LWR	0x03000800	4	1	<default>	2.000
2	LRD	0x03001000	3	2	<default>	2.000
2	BRD	0x0000 0x0130	2	8		2.000



Standard SYNC Shift settings for TC3 are used:



AL Status Codes in Case of Synchronization Loss

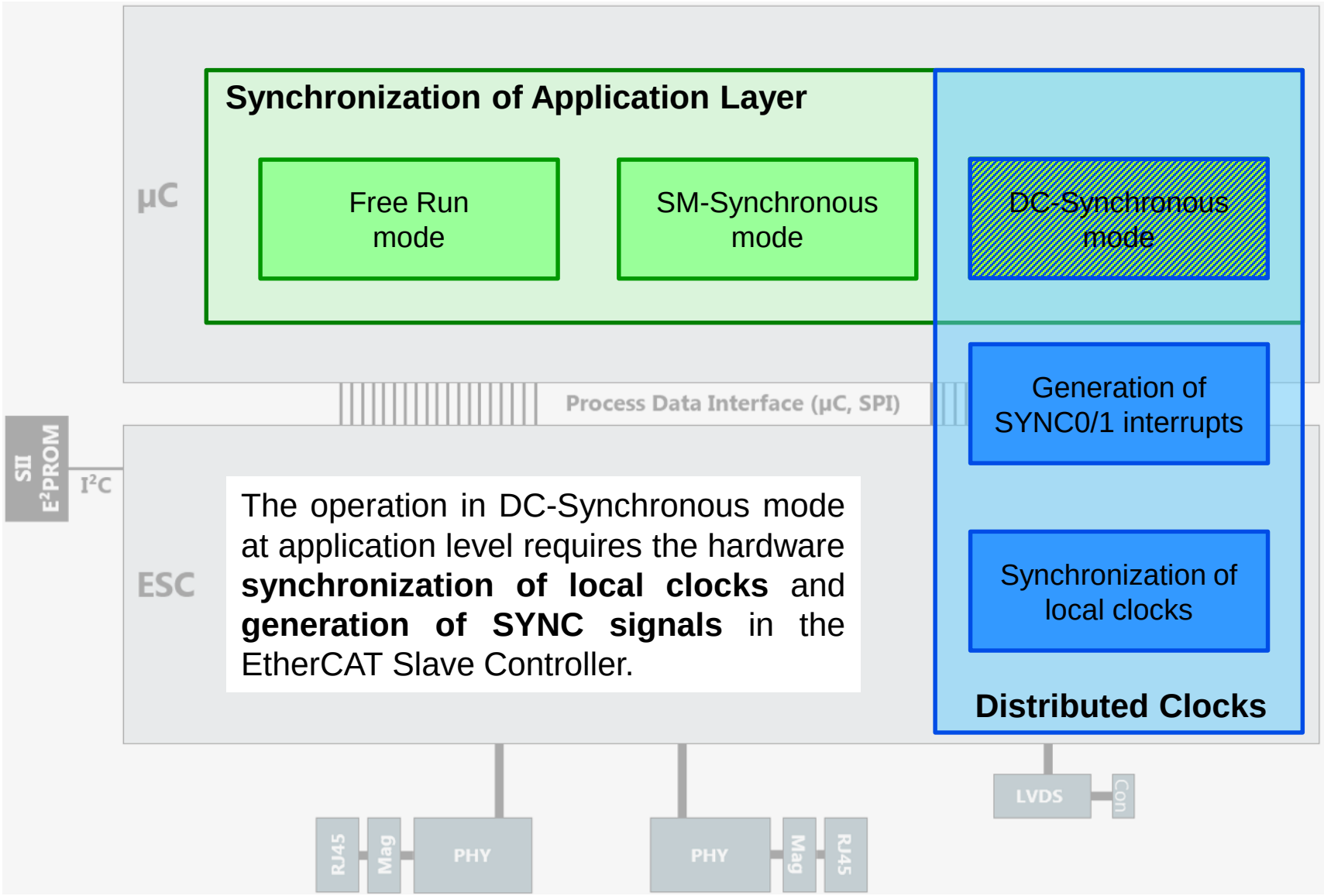
BECKHOFF

When a DC-Synchronous slave loses synchronization, it shall step back to Safe-Operational reporting one of the following AL Status Codes:

AL Status Code	Name	Meaning and reliminary checks suggested
0x35 (0x36, 0x37)	DC Invalid SYNC (SYNC0, SYNC1) cycle time	Cycle time configured by master is invalid for the slave application. **Action** → Check 0x1C32/33:05, and in case increase the cycle time of the linked task. Check if slave only supports a discrete set of allowed cycle times (see slave documentation).
0x30	Invalid DC SYNC configuration	SYNC settings downloaded by master are invalid for the slave application. **Action** → Check that the DC SYNC settings were not changed manually, and in case delete and manually append the slave in the configuration again.
0x2D	No SYNC Error	No SYNC signal is generated within the SafeOP → OP transition. **Action** → Check that the DC SYNC settings were not changed manually, and in case delete and manually append the slave in the configuration again, then take a Wireshark trace in order to check that the master properly configures the SYNC Start Time.
0x32	PLL Error	Slave application cannot synchronize itself to the communication cycle. **Action** → Check master jitter performances.
0x1A	Synchronization Error	Slave application cannot synchronize itself to the communication cycle. **Action** → Check master jitter performances.
0x33	DC Sync IO Error	
0x34	DC Sync Timeout Error	

Synchronization at Application Layer

BECKHOFF



Distributed Clocks

BECKHOFF

The Distributed Clocks technology enables synchronization of local clocks in EtherCAT devices (slaves and master) with maximum deviations ≤ **100 ns**.

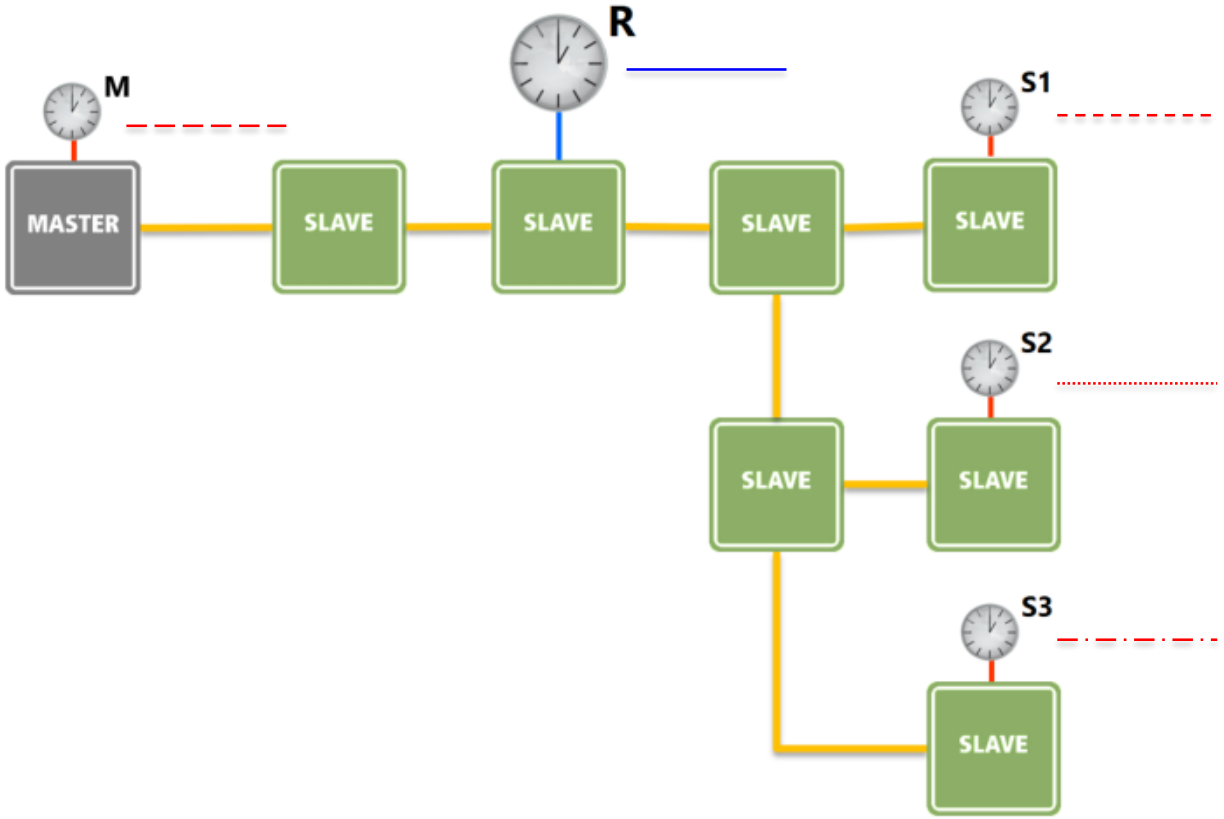
- **System Time**: time shared by all the DC-synchronous devices:
  - Beginning on January, 1st 2000 at 00:00h
  - Base unit is 1 ns per digit
  - Size is 64-bit (enough for more than 500 years)
  - Lower 32-bits span over 4.2 seconds (normally enough for synchronization and time stamping

- **Reference Clock**: device holding the System Time, is by default the first slave in the network configured in DC-Synchronous mode (setting not changeable):
  - “slave”, in order to have a fully-hardware, highly precise time reference.
  - “first”, in order to distribute the System Time to all other DC-synchronous network devices (slaves and master) within one single communication cycle.

BECKHOFF

DC-Synchronous devices in the network are therefore classified in:

- **Reference Clock**: first DC-Synchronous slave
- **Slave Clocks**: all other DC-Synchronous slave + (by default) master

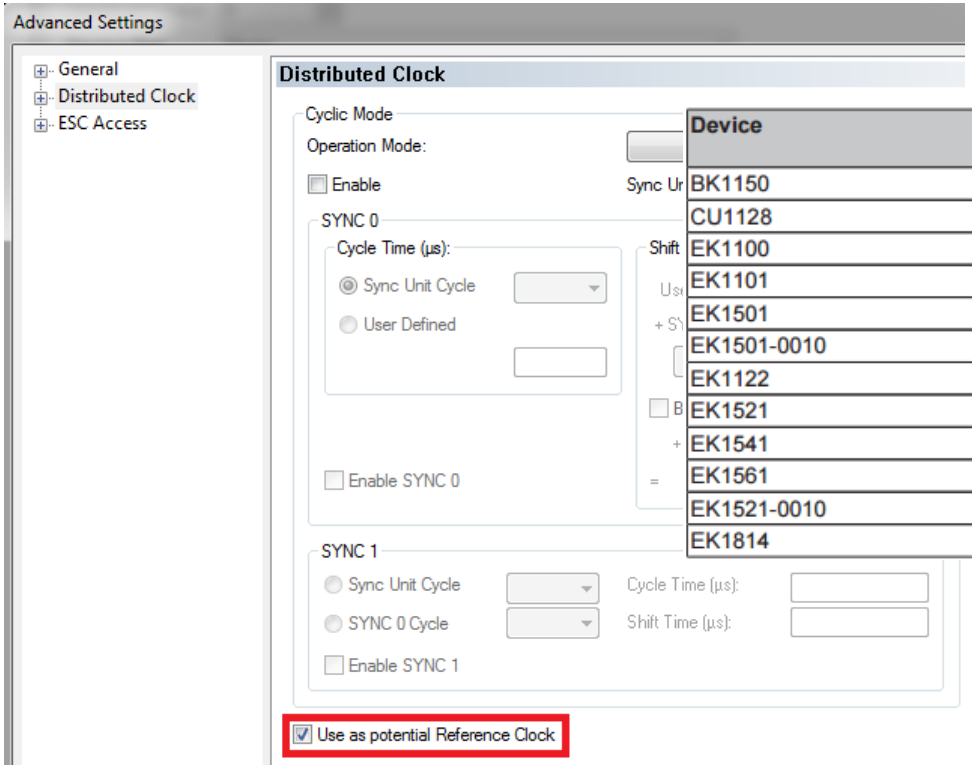


In Depth – Potential Reference Clock

BECKHOFF

The Reference Clock is not necessarily the very first slave in the network, yet it is suggested to have it not too far away from the master.

In case doubts arise concerning the good quality of the Reference Clock time reference (e.g. why synchronization errors are reported by several DC-Synchronous slaves), it is suggested to enable the closest topological component to the master as Reference Clock:



Device	XML revision in the configura-tion	Serial number of the component
BK1150	from BK1150-0000-0016	from firmware 01: xxxx01yy
CU1128	from CU1128-0000-0000	from firmware 00: xxxx00yy
EK1100	from EK1100-0000-0017	from firmware 06: xxxx06yy
EK1101	from EK1101-0000-0017	from firmware 01: xxxx01yy
EK1501	from EK1501-0000-0017	from firmware 01: xxxx01yy
EK1501-0010	from EK1501-0010-0017	from firmware 02: xxxx02yy
EK1122	from EK1122-0000-0017	from firmware 01: xxxx02yy
EK1521	from EK1521-0000-0018	from firmware 03: xxxx03yy
EK1541	from EK1541-0000-0016	from firmware 01: xxxx01yy
EK1561	from EK1561-0000-0016	from firmware 01: xxxx01yy
EK1521-0010	from EK1521-0010-0018	from firmware 03: xxxx03yy
EK1814	from EK1814-0000-0016	from firmware 00: xxxx00yy

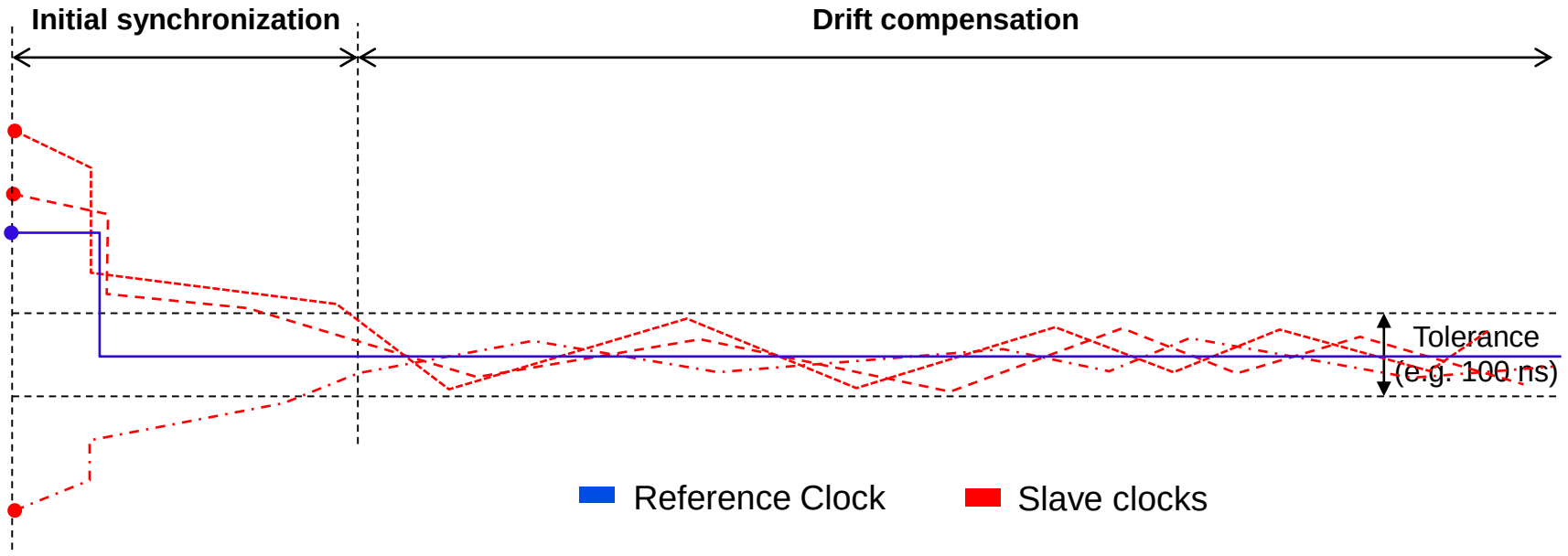
Synchronization Procedure

BECKHOFF

The hardware synchronization of local clocks distinguishes **2 phases:**

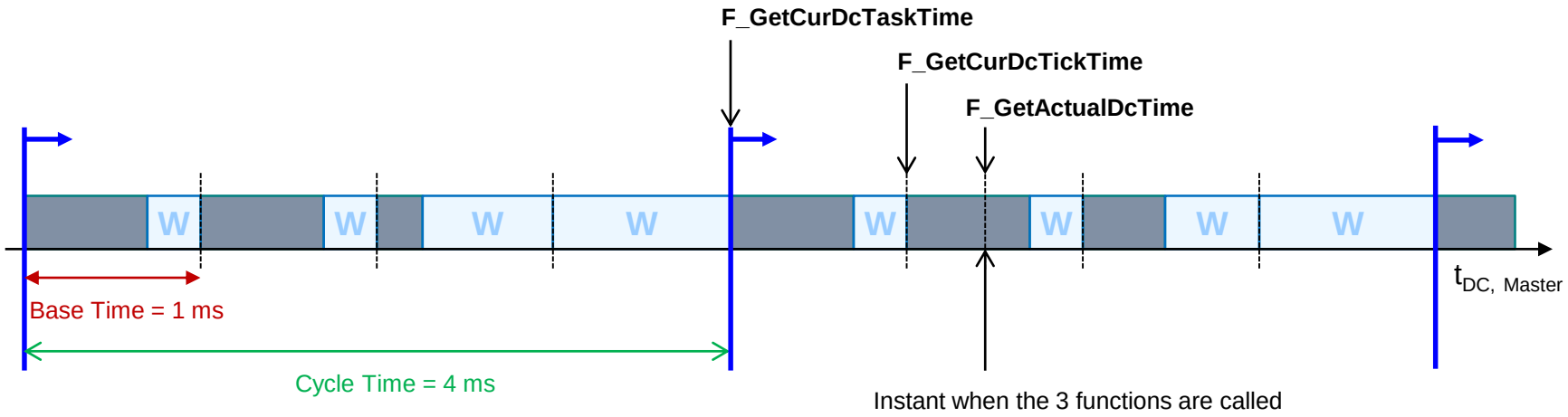
1. **Initial synchronization**: the master aligns the local clocks to the maximum tolerated deviation (typically < 100 ns).

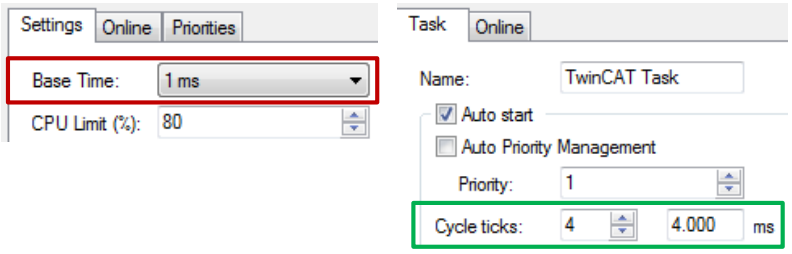
2. **Drift compensation:** local clocks are kept synchronized against drifts (slightly different quartz frequencies, thermal effects or ageing)



The PLC program can handle DC times thanks to three functions in **TcEtherCAT.lib** library:

- **F_GetCurDcTaskTime** : returns the DC time when the current task cycle started
- **F_GetCurDcTickTime** : returns the DC time of the last TwinCAT Base Time tick
- **F_GetActualDcTime** : returns the DC time when the function is called



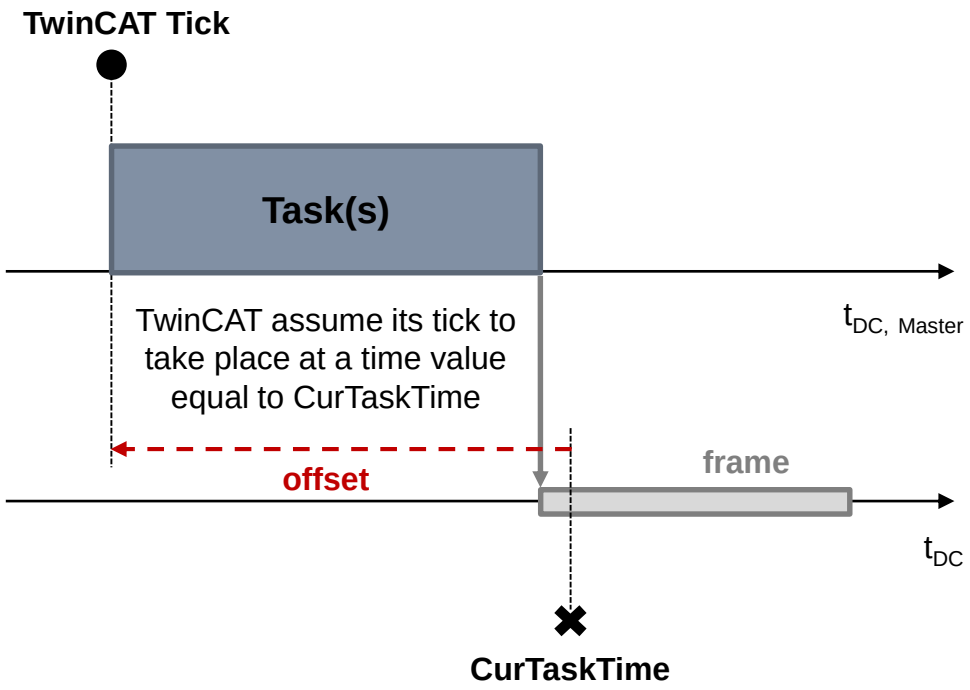


The time information returned by the three functions is based on the $t_{DC,\ Master}$ time base.

BECKHOFF

By default, whenever a DC-Synchronous EtherCAT network is configured TwinCAT adjusts its real-time clock to the System Time, yet the two time references are not identical.

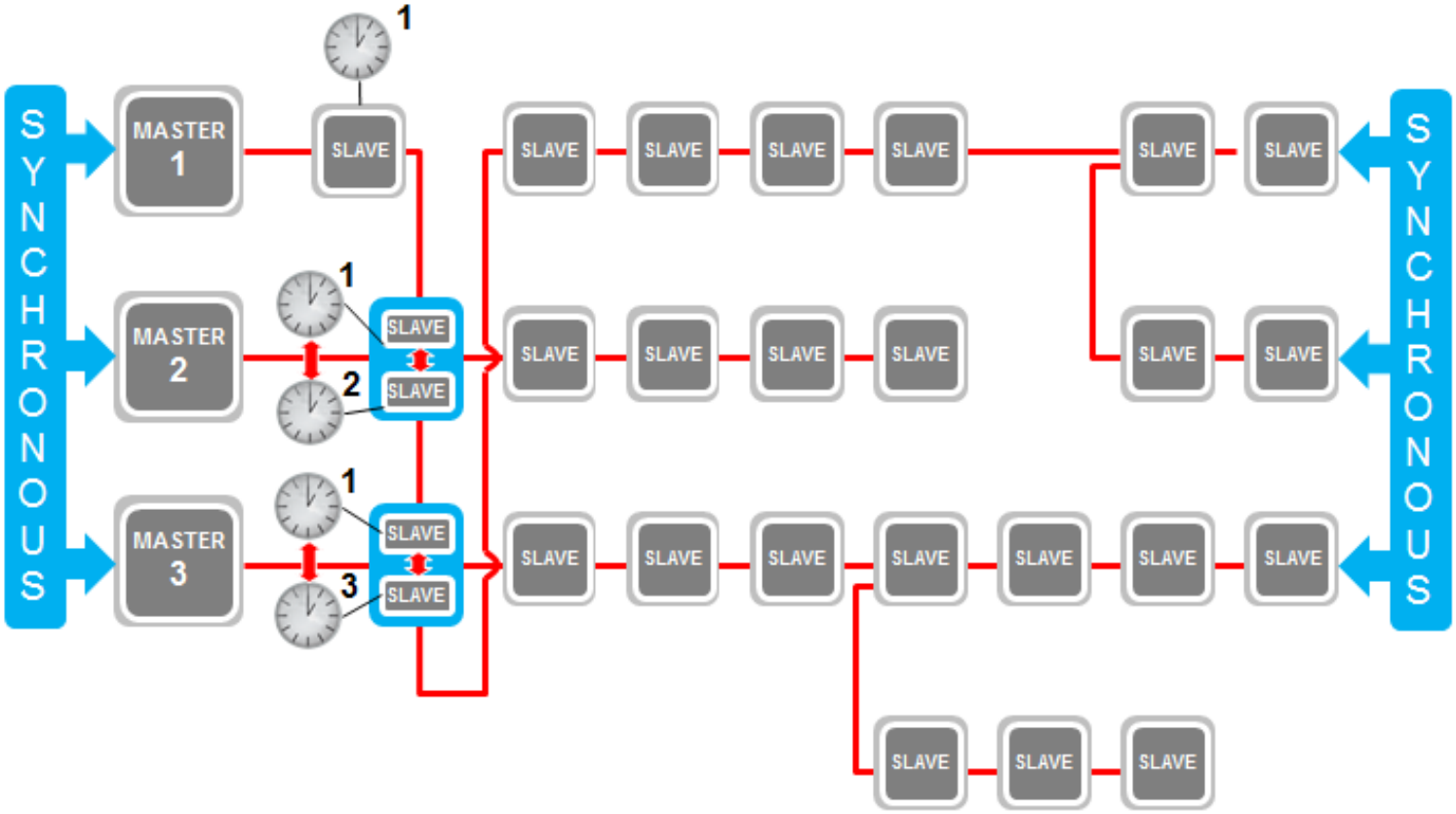
For calculation simplicity, TwinCAT assumes internally that the time the software task is started is equal to the CurTaskTime, what leads to an **offset** between the hardware DC time (t_{DC}) and the DC-adjusted TwinCAT time ($t_{DC, Master}$).



DC Synchronization Between Different Networks

BECKHOFF

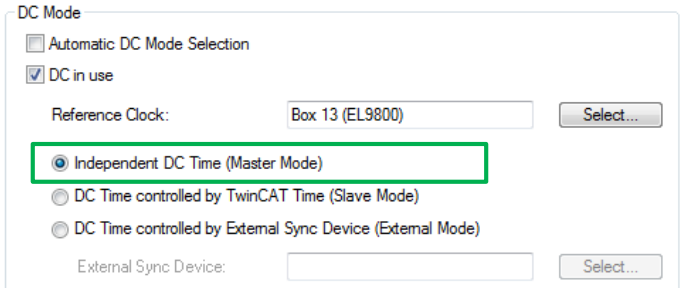
Besides providing synchronization within one single EtherCAT network, the Distributed Clocks technology enables to synchronize the DCs of different EtherCAT networks, or to synchronize the DCs of an EtherCAT network to an external IEEE 1588 reference as well:



DC Synchronization Between Different Networks

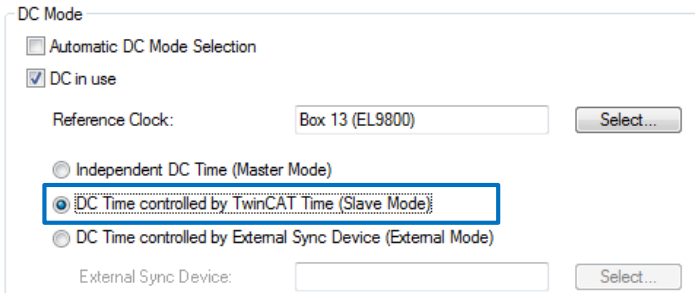
BECKHOFF

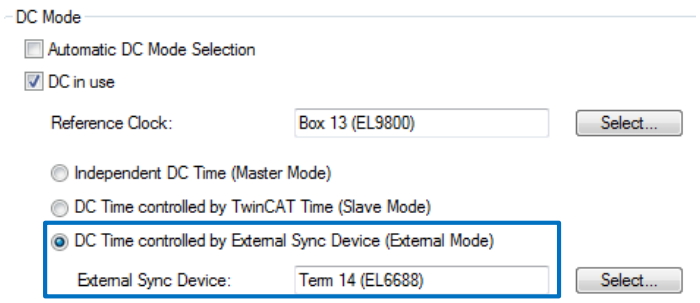
A network whose Reference Clock runs “untouched” (default case with only one EtherCAT network configured in a TwinCAT project), works **Master Mode**.



In some cases yet, the Reference Clock can also be adjusted by another time source:

- **Slave Mode**: the time source is the clock of the master device (i.e. TwinCAT clock).
- **External Mode**, the time source (e.g. the Reference Clock of another network, or an IEEE1588 signal) is provided through a dedicated slave device.



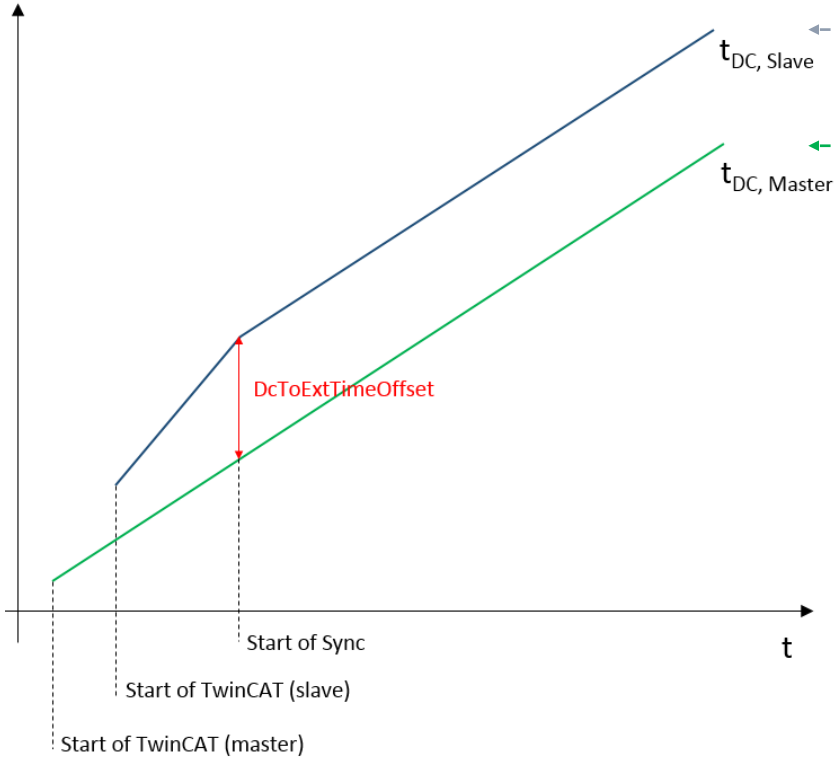


In Depth - DcToExtTimeOffset

BECKHOFF

Whenever the local Reference Clock of an EtherCAT network is adjusted to an external time source, **frequencies** are synchronized. Yet the absolute values of the local System Time can in general have an offset with respect to those of the external time source.

This offset shall be considered if time information want to be referred to the same time base:



Synchronized DC Time = $t_{DC, Slave}$ + DcToExtTimeOffset

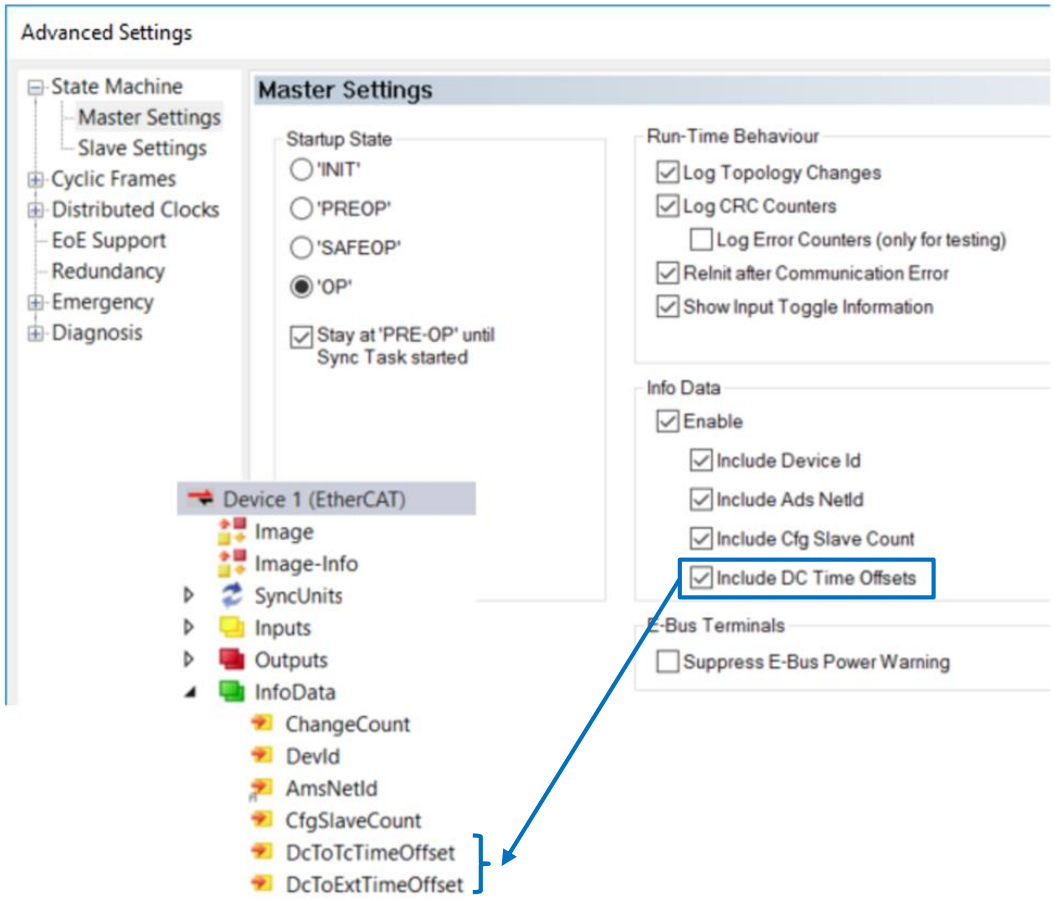
Synchronized DC Time = $t_{DC, Master}$

In case of cascaded synchronized systems

Synchronized DC Time = $t_{DC, Slave} + \Sigma_i DcToExtTimeOffset_i$

BECKHOFF

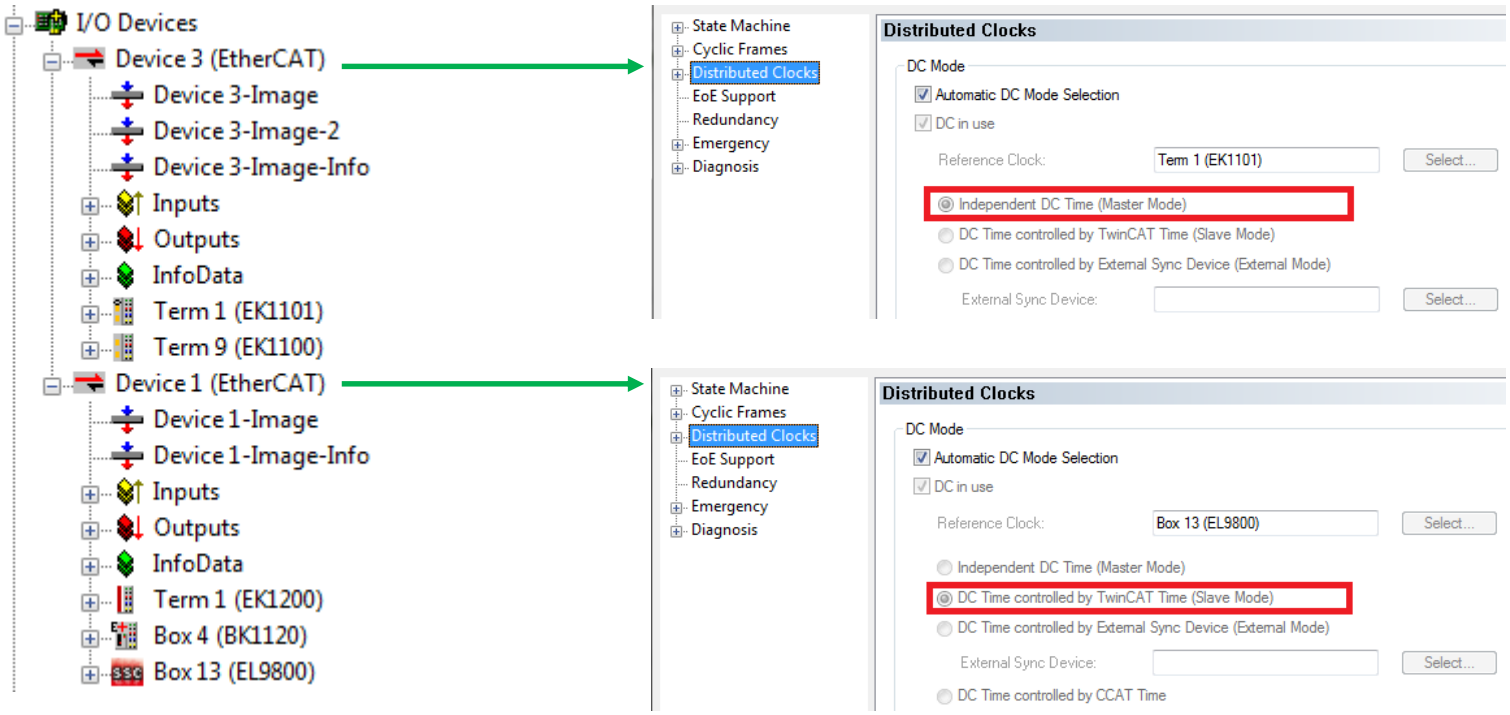
TwinCAT provides the **DcToExtTimeOffset** information as cyclic input which can be linked to a PLC program for time calculations (shall be manually enabled).



Synchronization of Several Networks on Same PC

BECKHOFF

If multiple DC-synchronous EtherCAT networks are run by the same TwinCAT controller, synchronization is automatically guaranteed (no external synchronization device needed).

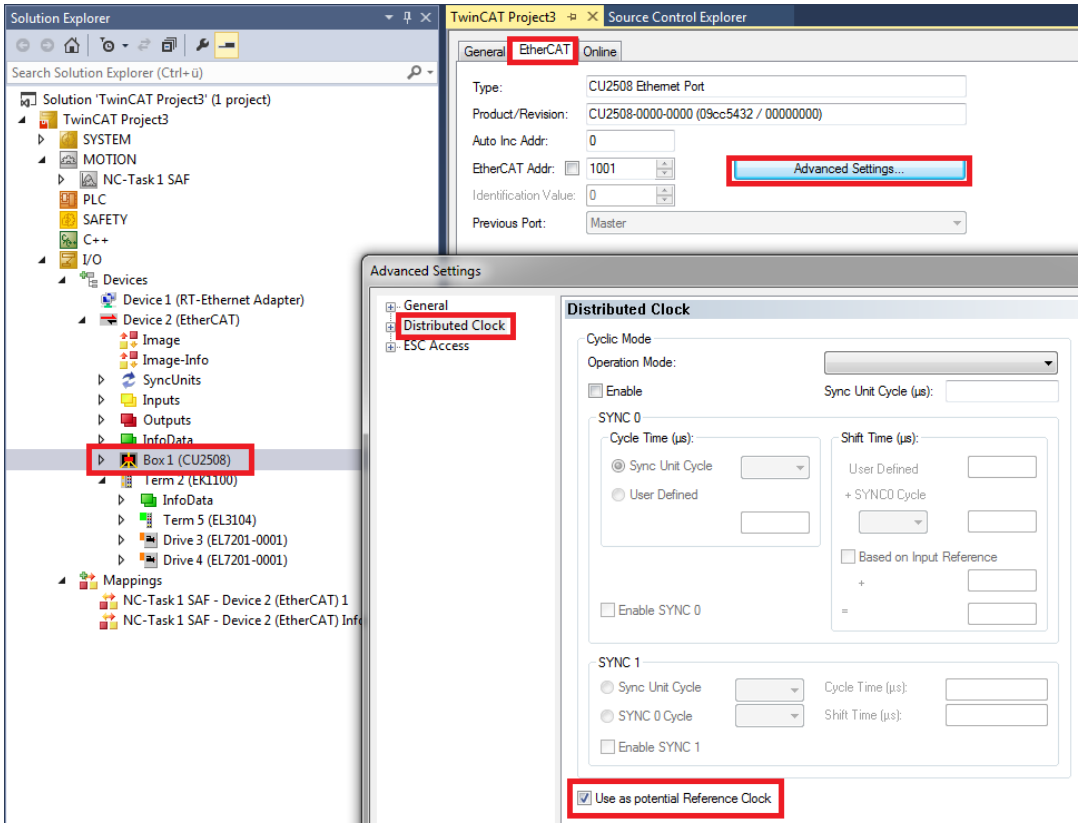


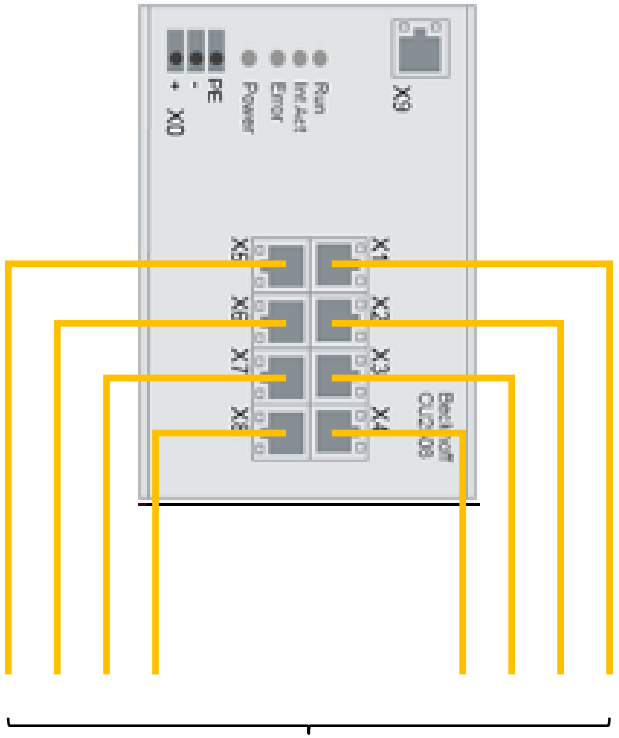
The first appended network will be automatically set in Master Mode, all others in Slave Mode.

In Depth – Synchronization via Port Multiplier

BECKHOFF

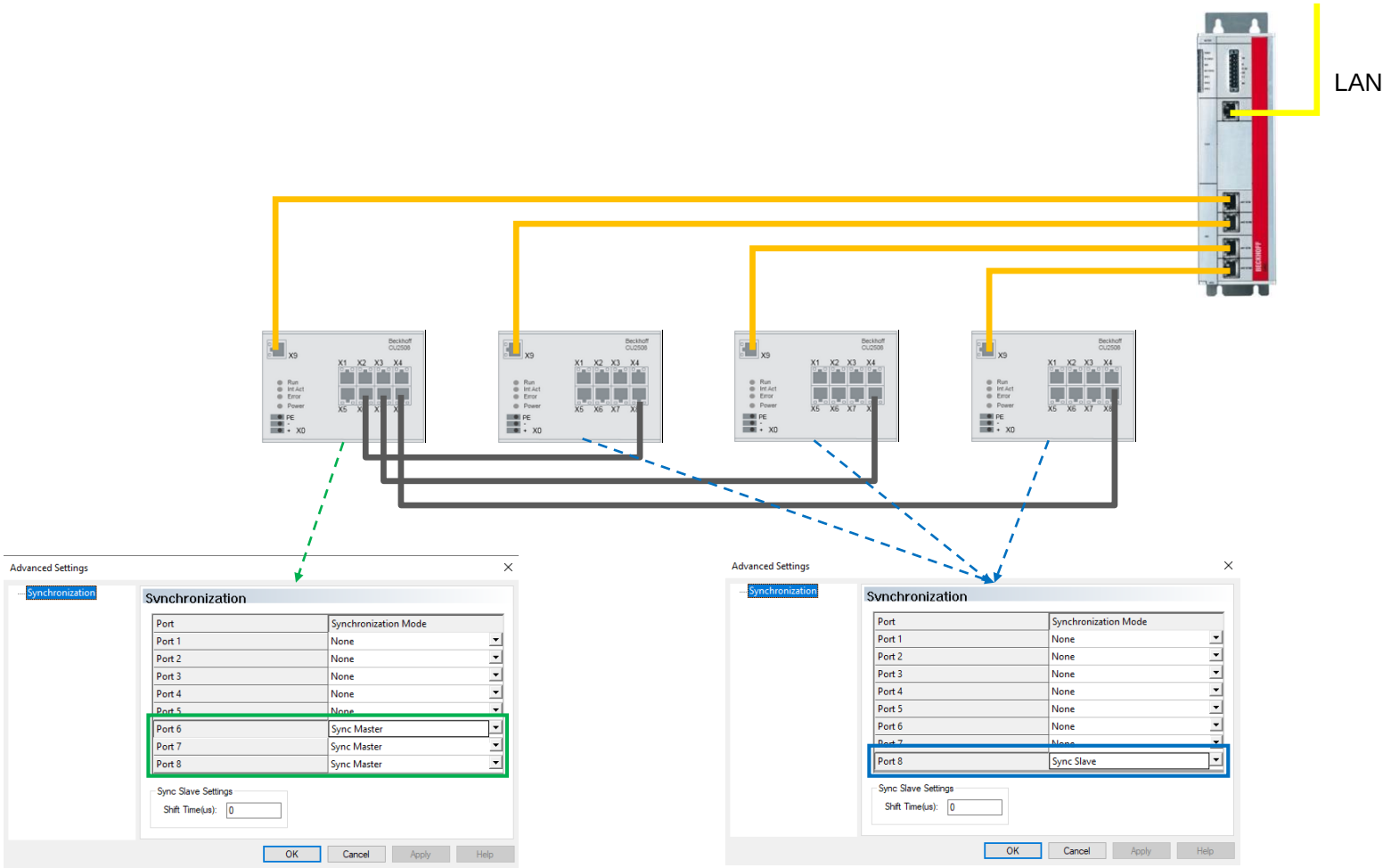
In order for multiple EtherCAT networks connected to the controller via a Port Multiplier to work synchronously with each other the **CU2508** slave shall be manually configured as **Reference Clock** in **each** network (only using CU2508 the DC synchronization can be enabled together with cable redundancy!).



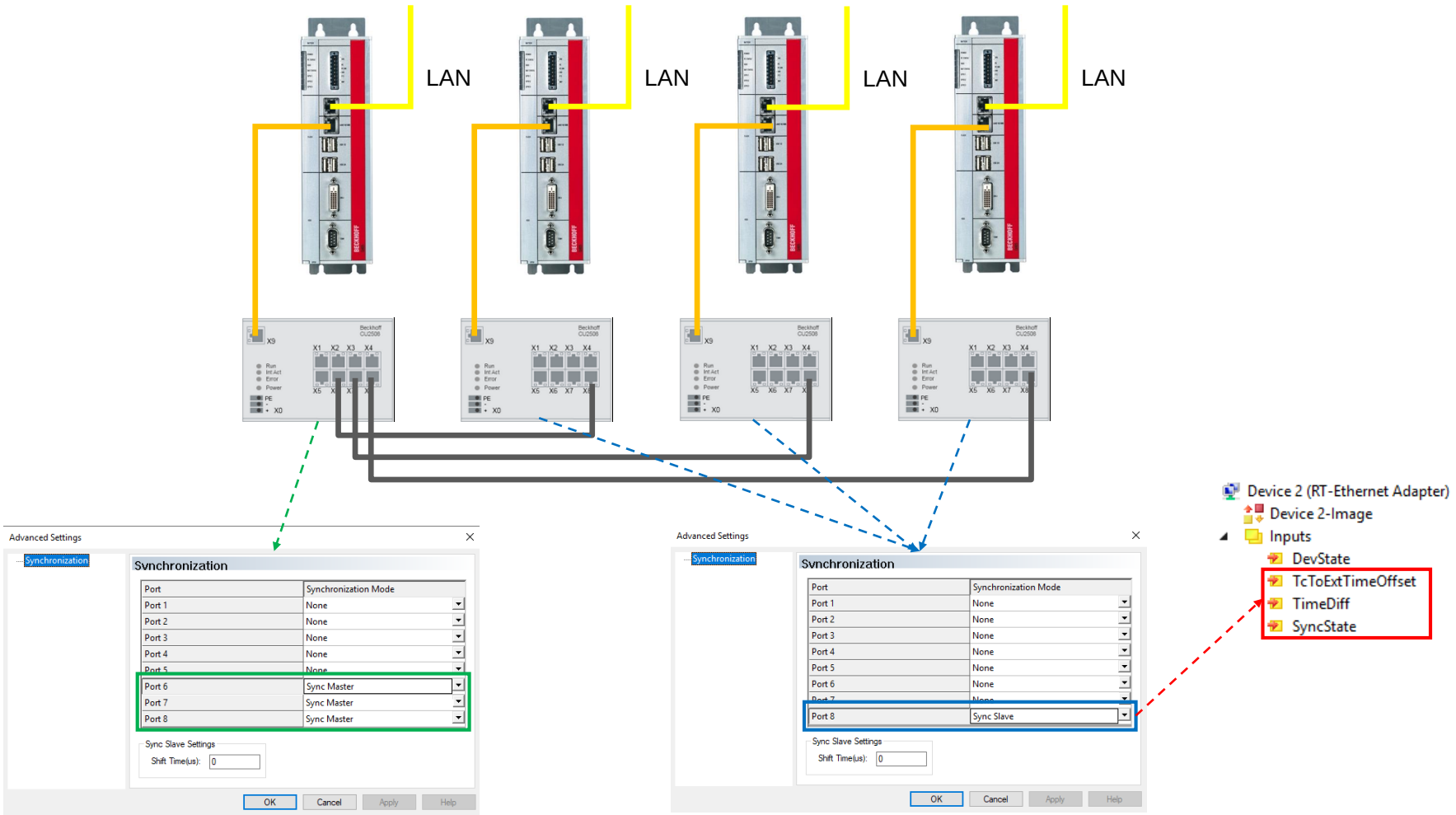


In Depth – Synchronization of Several Port Multipliers

If several CU2508s are used (for example in XTS applications), the clocks of different CU2508 shall be synchronized with each other as well:



The synchronization of several CU2508s works also if these are connected to different PCs:

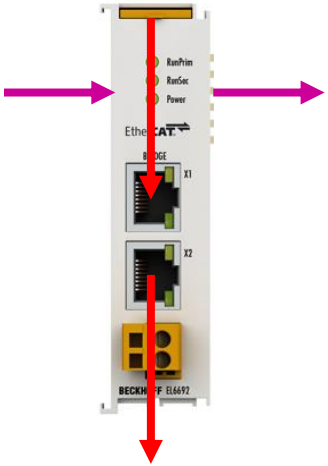


Synchronization via EtherCAT Bridge (EL6692/EL6695)

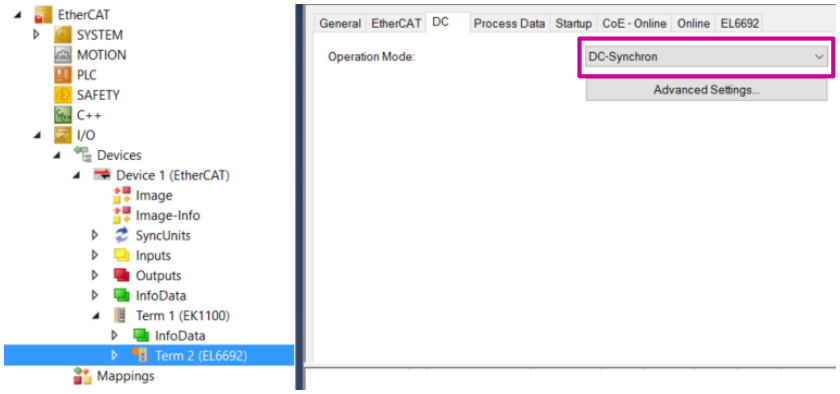
BECKHOFF

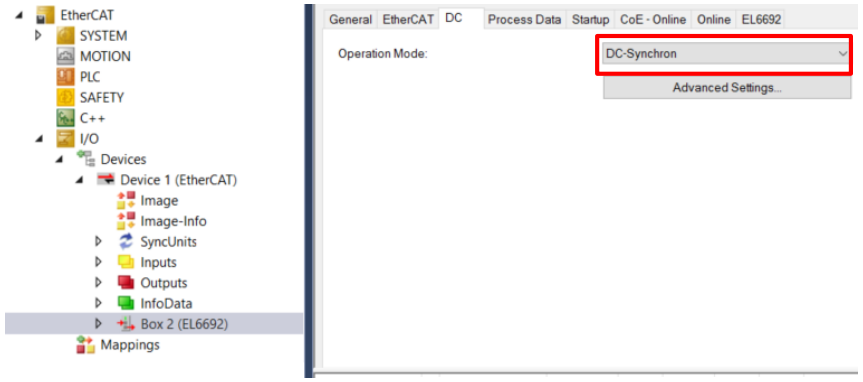
The EL6692/EL6695 bridge device is slave of two networks at the same time:

- **Primary Side** : EBUS
- **Secondary Side** : RJ45



For DC synchronization, both bridge sides shall be configured in DC-Synchronous mode:

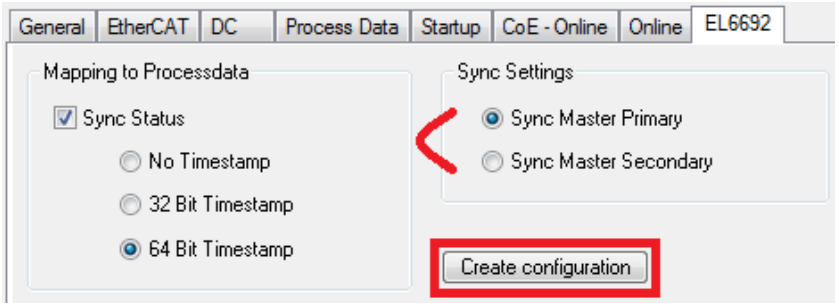


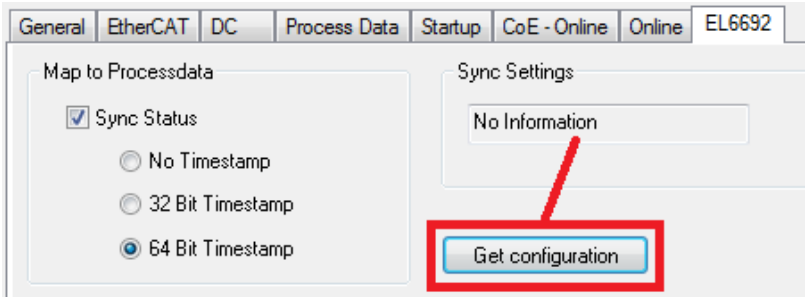


Synchronization via EtherCAT Bridge (EL6692)

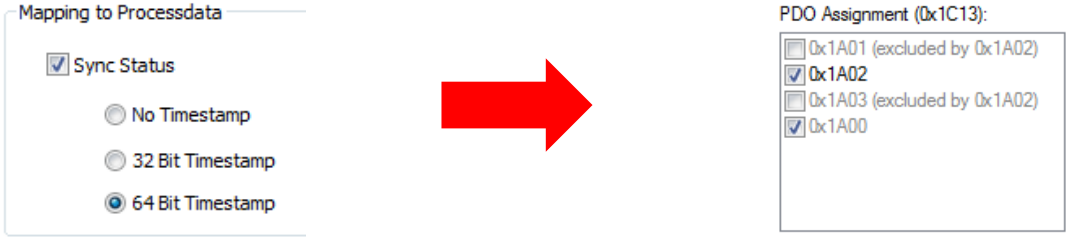
BECKHOFF

For EL6692, when configuring the Primary Side it has to be defined whether the Primary Side or the Secondary Side will be **Sync Master** (i.e. the side whose local System Time is not adjusted). The Secondary Side (which should be configured later) can upload this setting from the Primary Side (button **Get configuration**).





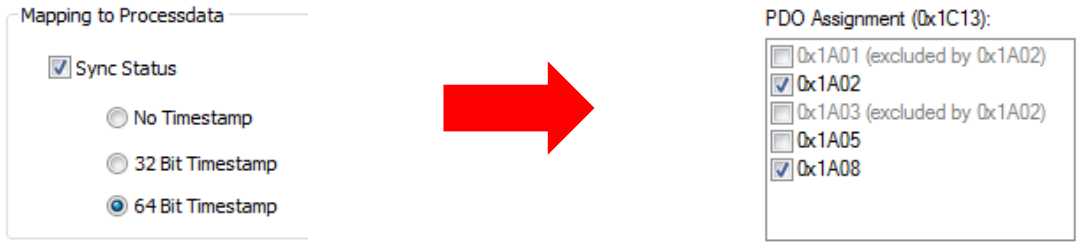
On both sides, the correct PDO assignment needed to transmit the time information is automatically set when the Timestamp format is selected.



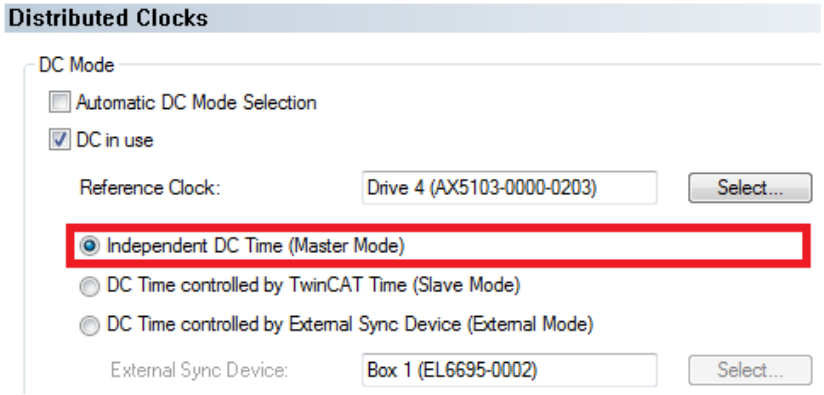
Synchronization via EtherCAT Bridge (EL6695)

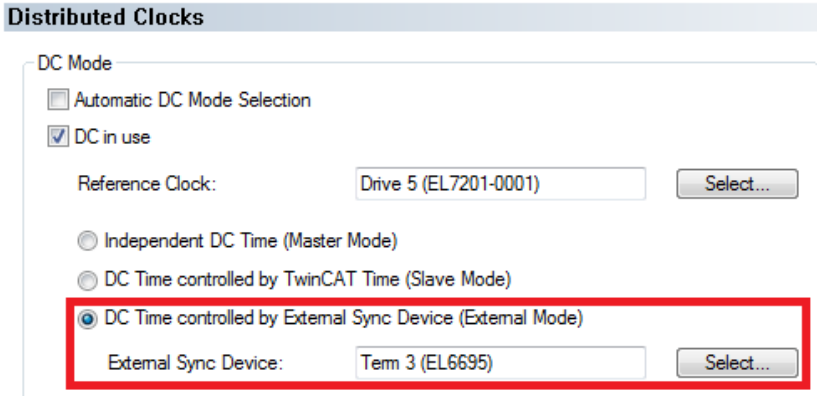
BECKHOFF

For EL6695, only the Timestamp format can be configured in the dedicated tab (the required PDOs are automatically mapped in this case too):



Which side is Sync Master and which is Sync Slave shall instead be configured directly among the Advanced Settings of the corresponding master device.

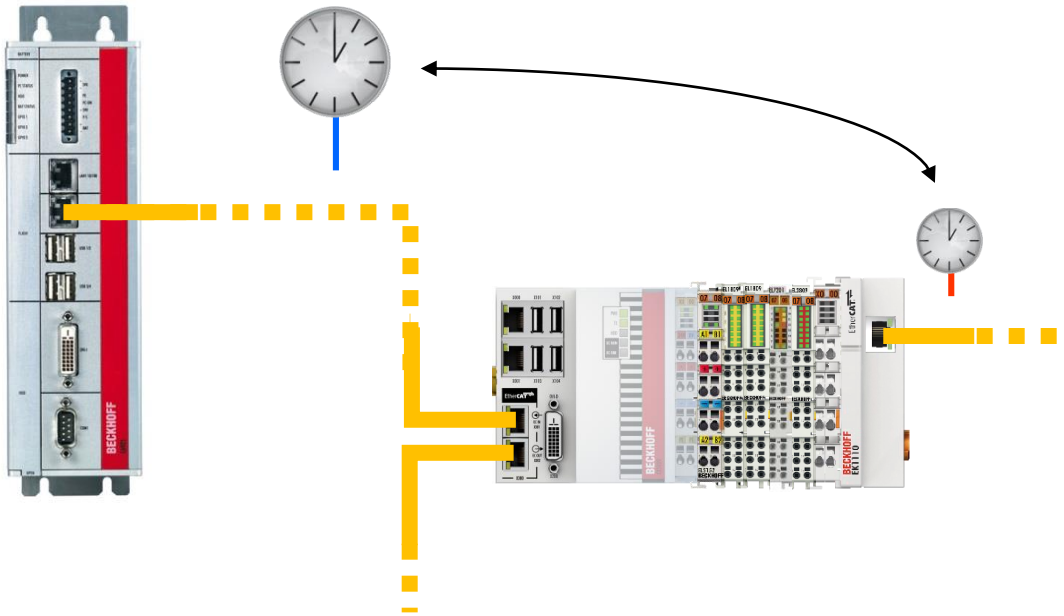




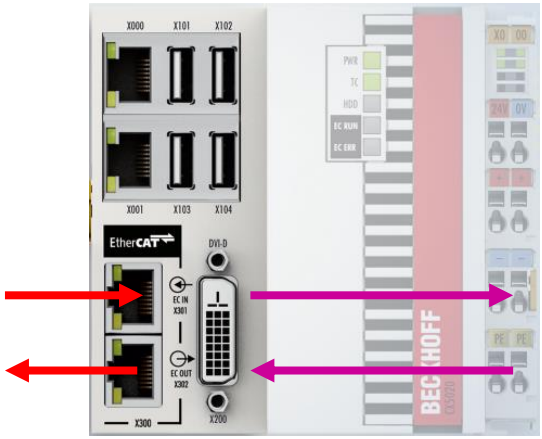
Synchronization via CCAT Interface (CX –B110)

BECKHOFF

The local EtherCAT EBUS master interface of a CX device can be synchronized with a superior EtherCAT network via the **–B110 slave** interface.



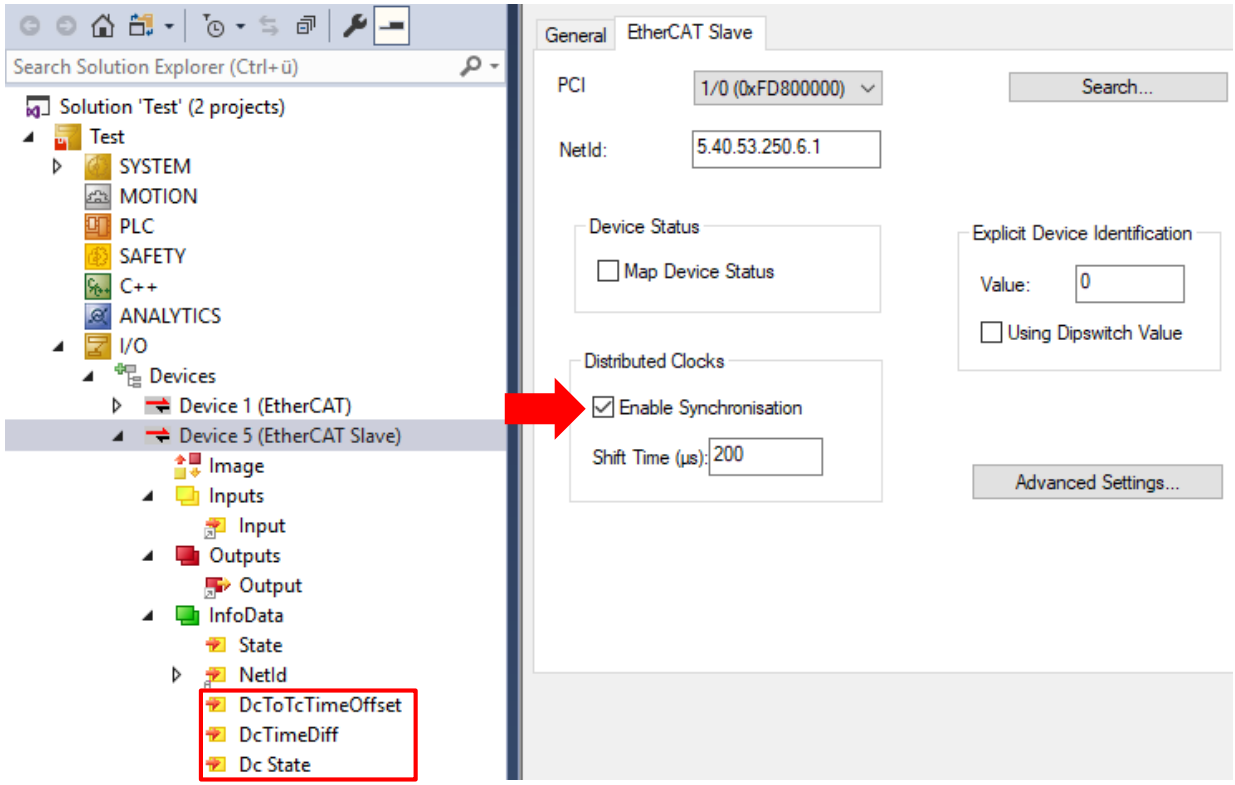
- **Primary Side** : CCAT
- **Secondary Side** : RJ45



Synchronization via CCAT Interface (CX –B110)

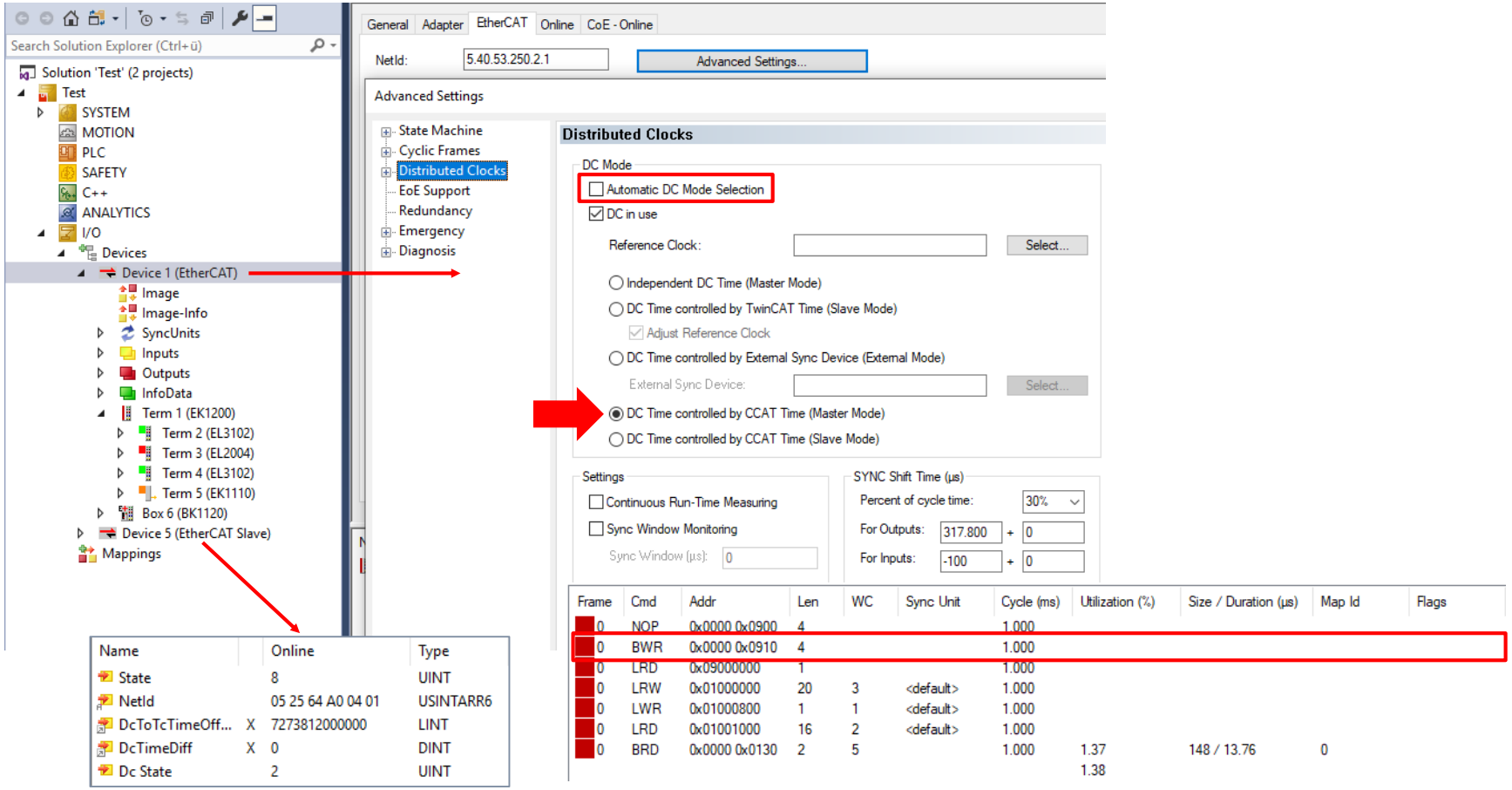
BECKHOFF

DC synchronization shall be activated in the –B110 slave interface (this will also enable the cyclic variables in the process image of the slave interface itself).



Synchronization via CCAT Interface (CX –B110)

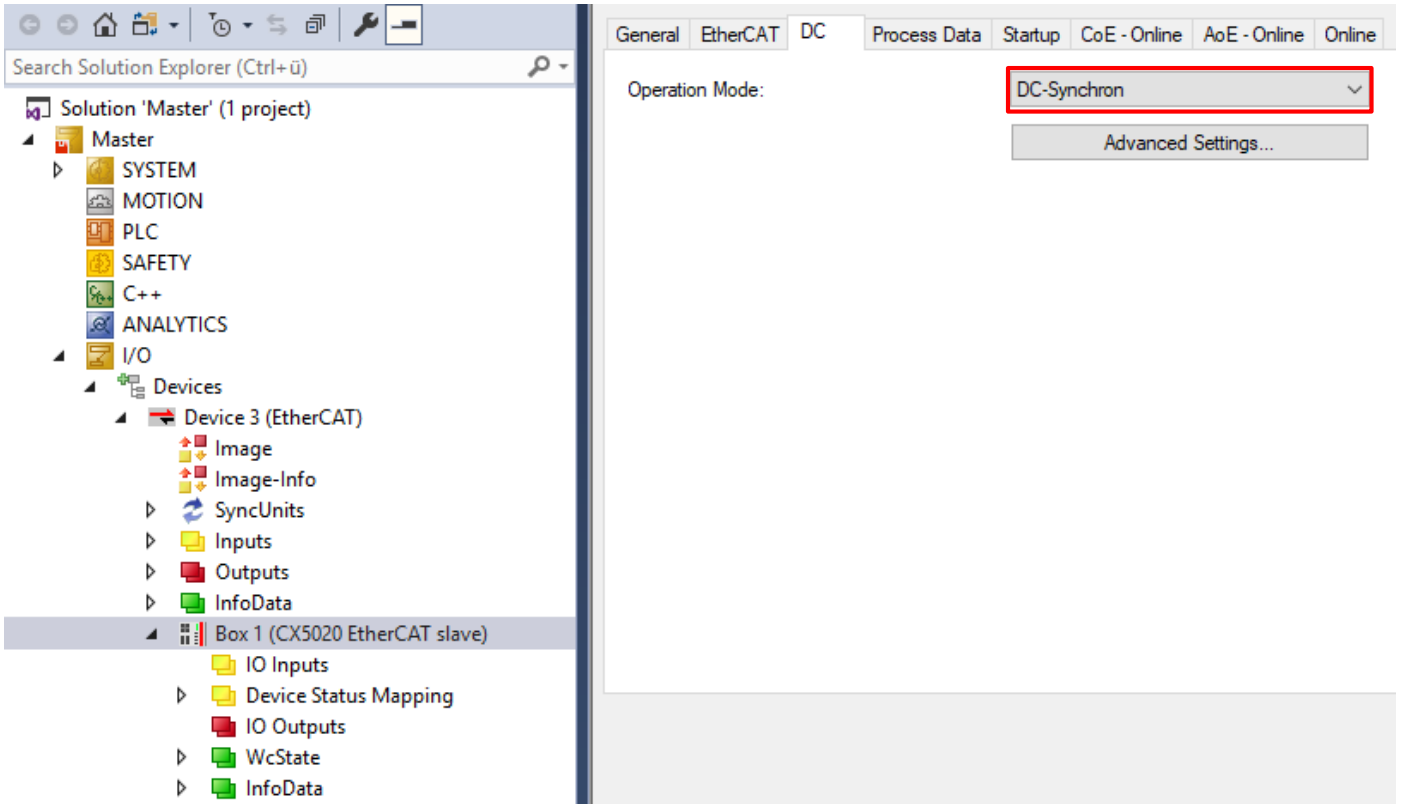
DCs in the local EtherCAT master on the CX shall be manually set to “DC Time Controlled by CCAT Time (Master Mode)” → CCAT is DC slave on secondary side and DC master on primary side”.



Synchronization via CCAT Interface (CX –B110)

BECKHOFF

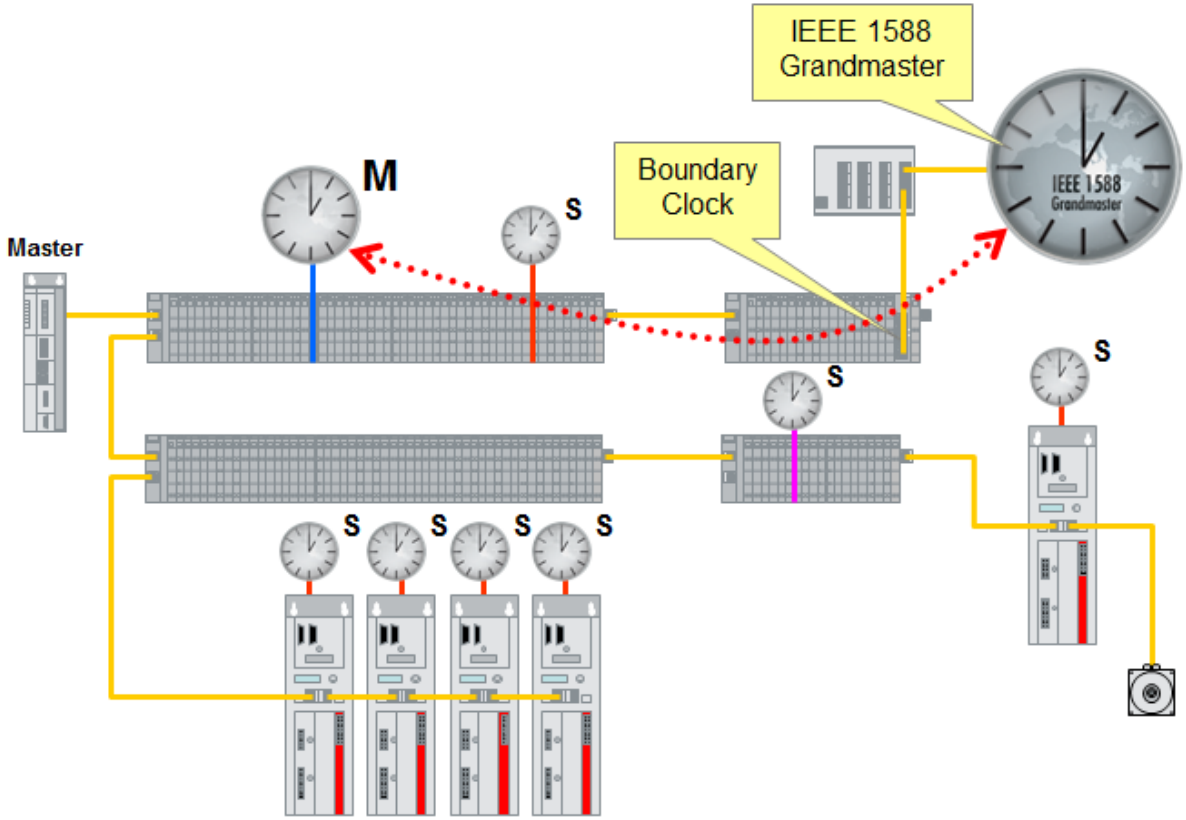
The –B110 slave interface of the CX shall be finally configured in “DC-Synchron” mode on the side of the superior EtherCAT network.



Synchronization via IEEE 1588 Grandmaster (EL6688)

BECKHOFF

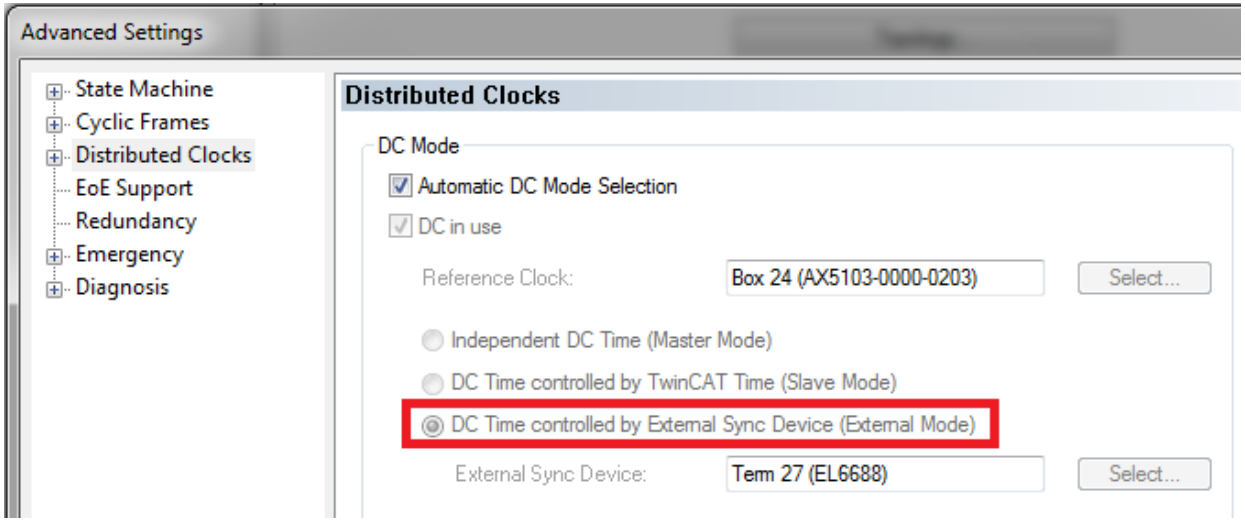
The Reference Clock of an EtherCAT network can be synchronized to an IEEE 1588 **Grandmaster Clock**.



Synchronization via IEEE 1588 Grandmaster (EL6688)

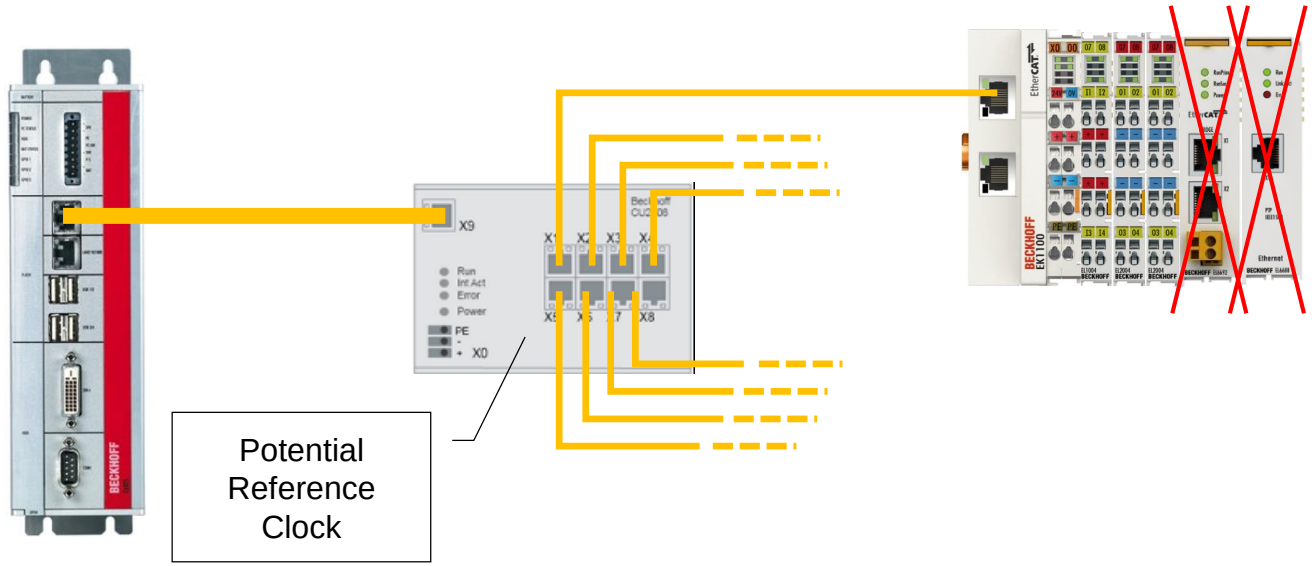
BECKHOFF

When an EL6688 is inserted in a TwinCAT project, the DC operation is **automatically** configured in **External Mode** (no manual setting needed). The local Reference Clock will be cyclically adjusted to the external IEEE 1588 time source (offset between the two time bases will be reported in the DcToExtTimeOffset variable).



Current Limitations in External Synchronization

It is currently **not** possible to externally synchronize (via EL6692/EL6695 or via EL6688) the DCs in EtherCAT networks/systems which are in turn locally synchronized via a CU2508:



In case of EL6692/EL6695, the terminal cannot be used to synchronize the DCs to an external system but it can still be used to exchange process data. As an alternative, CU2508s can be synchronized directly.

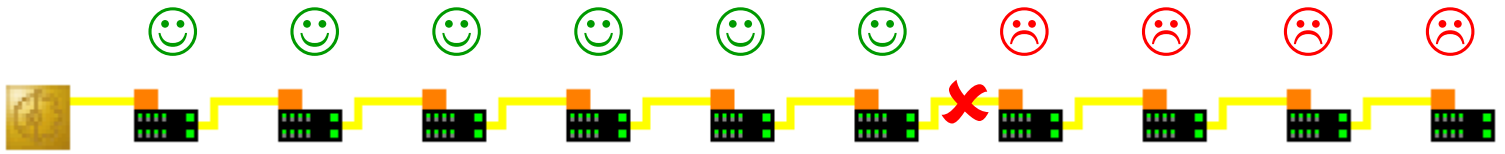
Redundancy

What is Cable Redundancy?

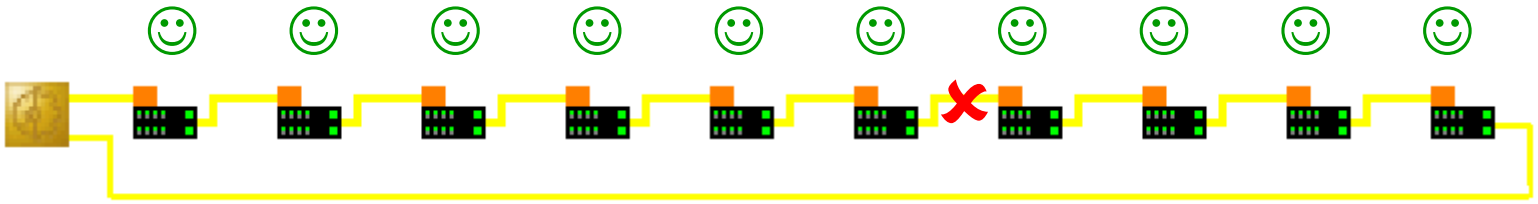
BECKHOFF

EtherCAT, like all Ethernet-based communication technologies, consists of point-to-point connections between devices.

- If one point-to-point connection is interrupted, communication is lost with all other downstream slaves.



Cable Redundancy is an optional EtherCAT functionality enabling to maintain communication and data exchange with all slaves in a ring topology also in case of a network interruption.

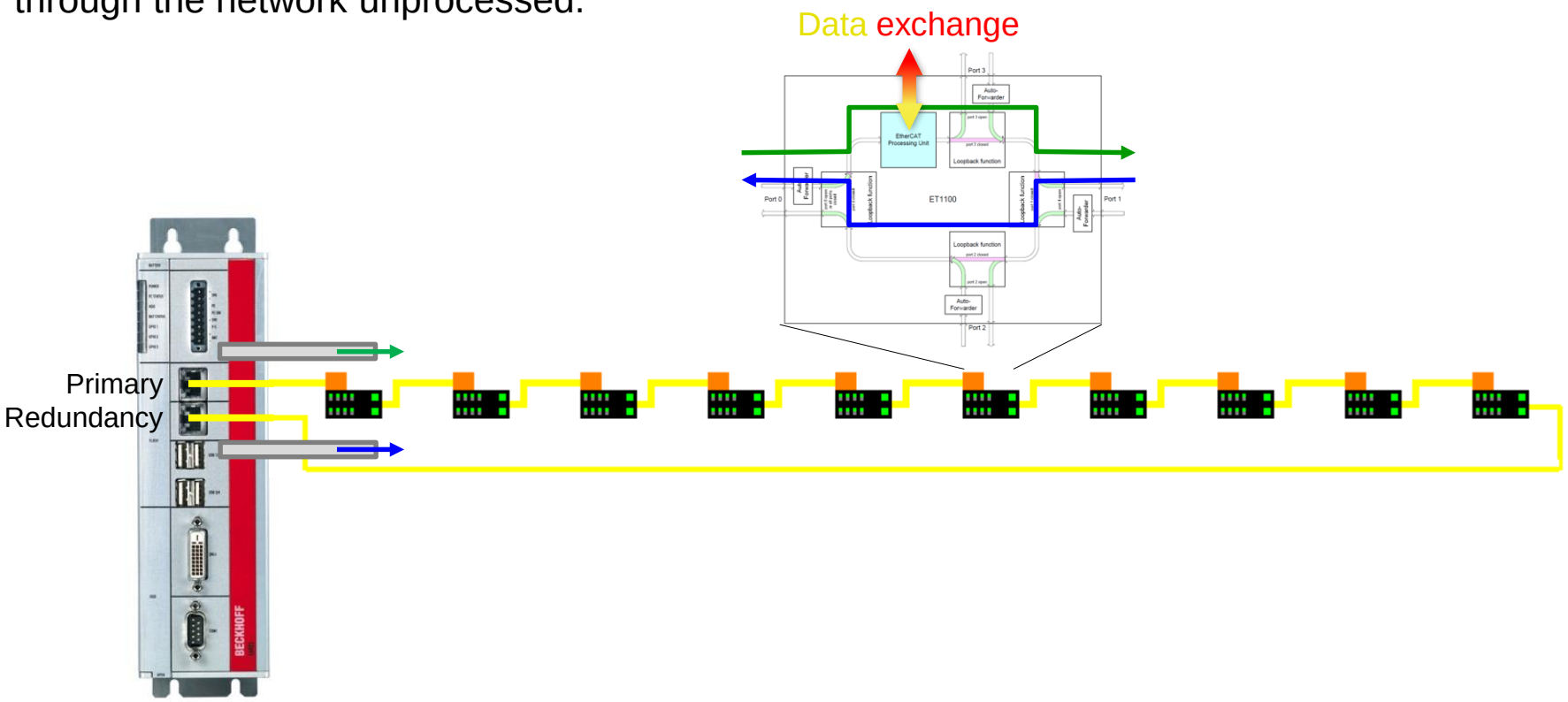


How does Cable Redundancy Work?

BECKHOFF

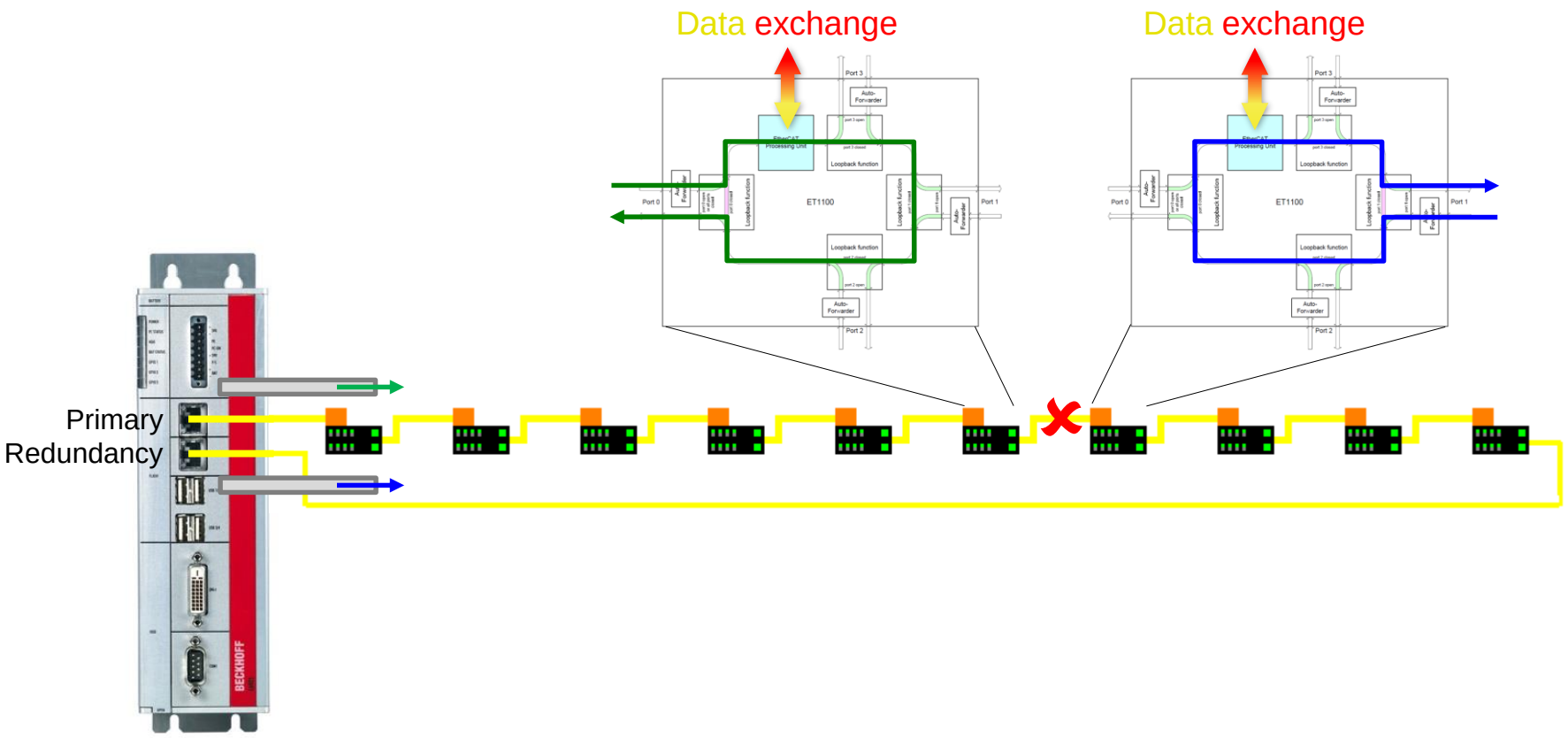
The master always sends each EtherCAT frame through both ports (in case mirroring addresses).

Under normal operating conditions (i.e. when the network is not interrupted) only frames sent by the primary port are processed by slaves, while frames sent by the redundancy port go through the network unprocessed.



How does Cable Redundancy Work?

In case of network interruption, the slaves neighbouring the break close the corresponding ports preventing from frame loss (max. 1 lost frame could occur and is tolerated).

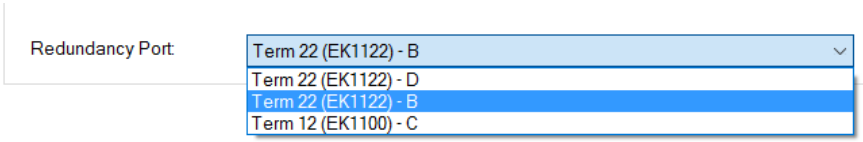


No specific support for Cable Redundancy is necessary on slave side, the standard behaviour of EtherCAT chips ensures proper operation.

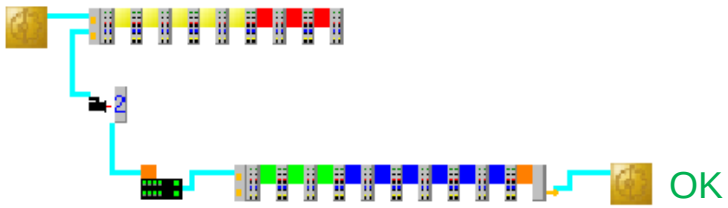
Where can the Redundancy Ring be Closed?

BECKHOFF

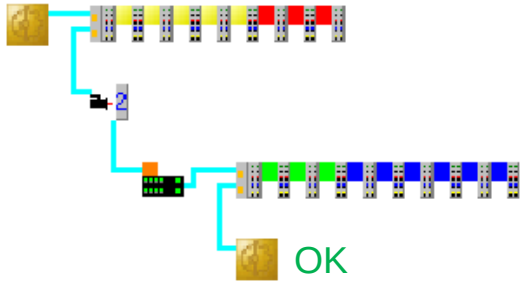
The redundancy ring can be closed at one of the following points:



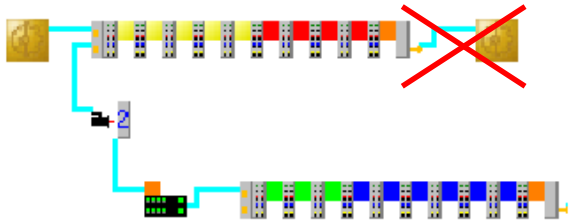
Port B of last 2-port device

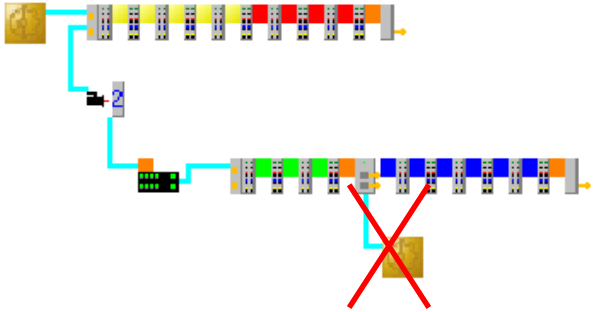


Port C of last EK110x coupler



It is not possible to close the redundancy ring at an **intermediate** point in the network





Which is the Best Topology for Cable Redundancy?

BECKHOFF

Cable redundancy is most effective in connection with a **line** topology

- All slaves can be reached independently from which cable is interrupted

In case of star topologies, not all slaves can be reached if a cable departing from a branch is interrupted

When Cable Redundancy is active, TC3 shows in **light blue** colour the cables which can be interrupted keeping communication with all slaves.

Multiple Interruptions

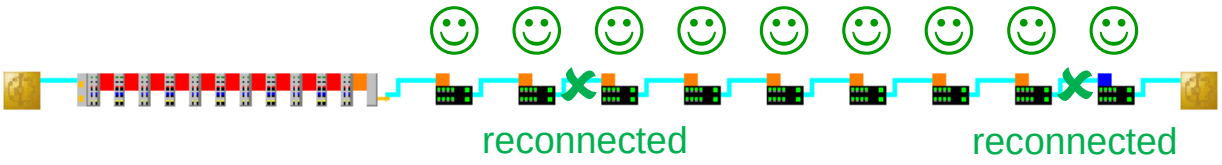
BECKHOFF

Cable redundancy is tolerant to one **single interruption** in the redundancy ring.

In case of (contemporary or consecutive) multiple interruptions:

1. Slave devices included between the two most external interruptions will cannot be reached by frames

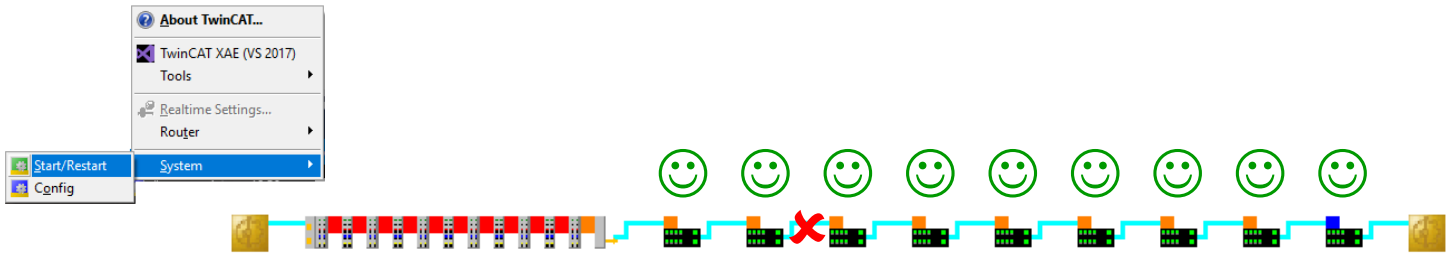
2. The master will restore communication with all nodes only when all interruptions have been removed



Start when Network Interrupted

BECKHOFF

The TwinCAT master is able to properly start the network also when the network is interrupted at one point.



General | Adapter | EtherCAT | Online | CoE - Online

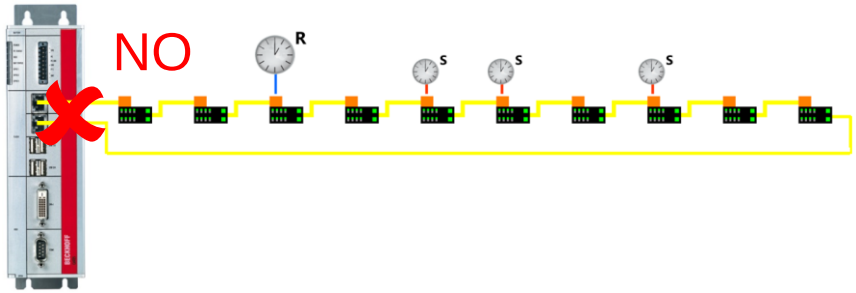
No	Addr	Name	State		CRC
1	1001	Box 1 (EP2308-0002)	OP		0, 0
2	1002	Box 2 (EP2308-0002)	OP	LNK_MIS B	0, 0
3	1003	Box 3 (EP2308-0002)	OP	LNK_MIS A	0, 0
4	1004	Box 4 (EP2308-0002)	OP		0, 0
5	1005	Box 5 (EP2308-0002)	OP		0, 0
6	1006	Box 6 (EP2308-0002)	OP		0, 0
7	1007	Box 7 (EP2308-0002)	OP		0, 0
8	1008	Box 8 (EP2308-0002)	OP		0, 0
9	1009	Box 9 (EP4174-0002)	OP		0, 0

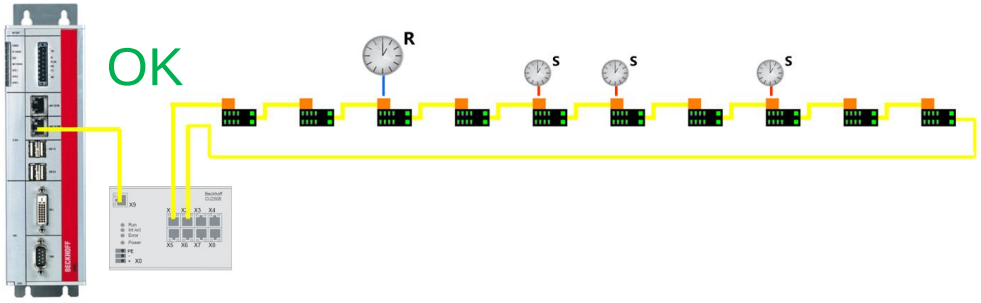
However, an interruption in the network at start-up shall be considered an **anomalous** situation which shall be solved as soon as possible and not accepted as normal operating condition (moreover, in this case the system would not be able to tolerate a further break).

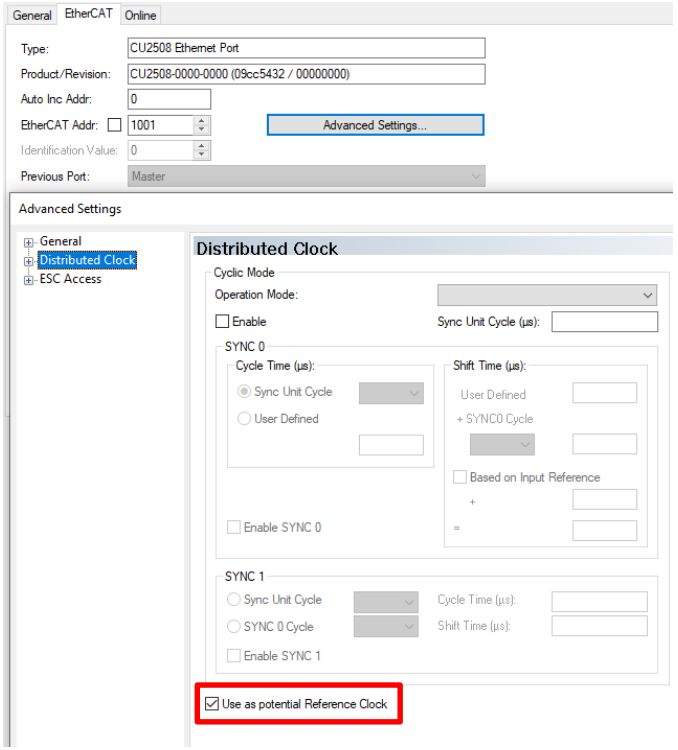
Cable Redundancy with DCs

BECKHOFF

Cable redundancy can be used in combination with Distributed Clocks synchronization only by using a **CU2508** port multiplier.







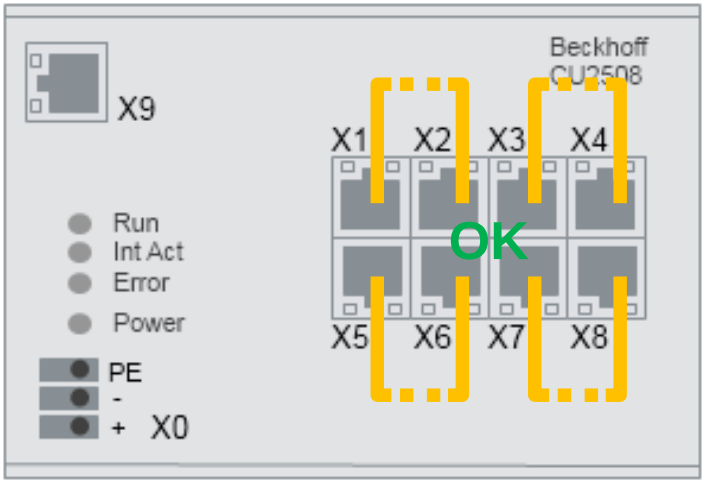
The CU2508 shall be always configured as DC **Reference Clock** in the network.

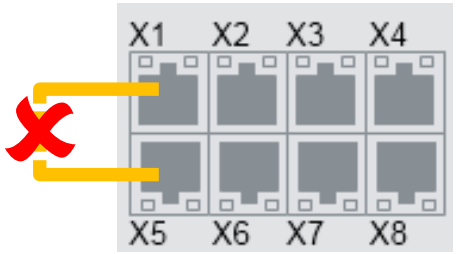
CU2508 Port Assignment in Cable Redundancy with DCs

BECKHOFF

Only some port **combinations** of CU2508 are allowed in order to implement cable redundancy:

- **X1** (primary) + **X2** (redundancy)
- **X3** (primary) + **X4** (redundancy)
- **X5** (primary) + **X6** (redundancy)
- **X7** (primary) + **X8** (redundancy)

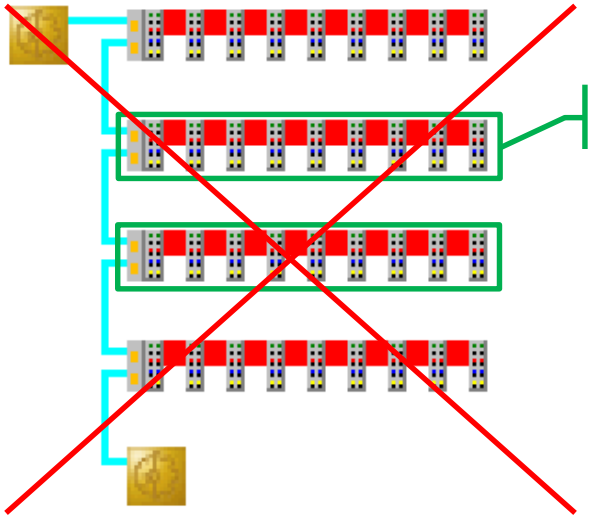




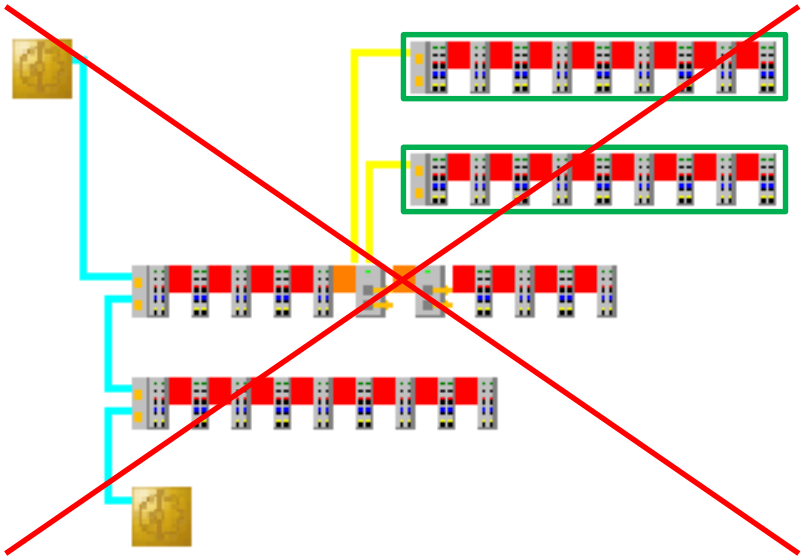
Cable Redundancy with Hot Connect

BECKHOFF

At current state, the contemporary use of Cable Redundancy and Hot Connect is **not** supported.



HC group



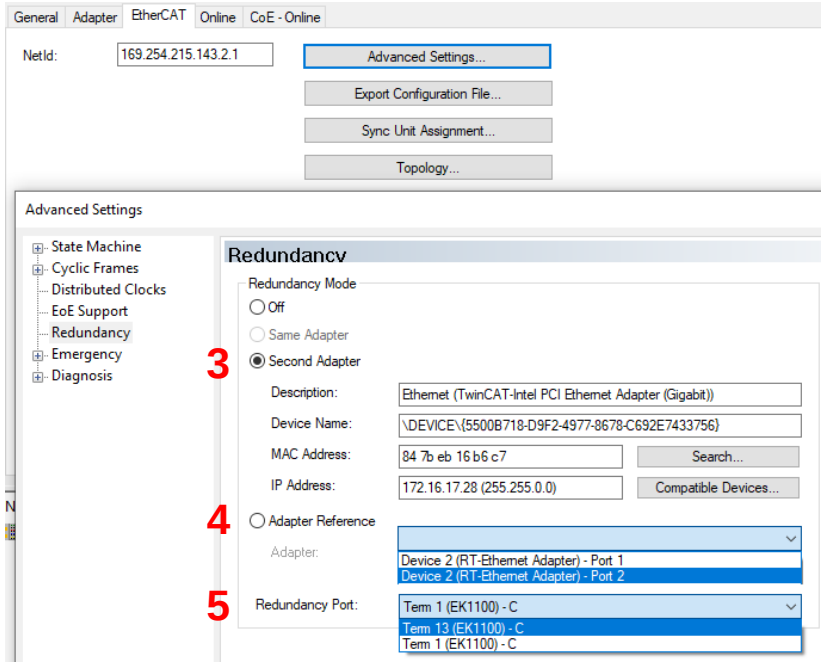
Several use cases can be successfully targeted without need for Hot Connect, simply by defining separate Sync Units for the removable segments…

Configuration of Cable Redundancy in TwinCAT

BECKHOFF

In order to activate the Cable Redundancy in TwinCAT:

1. Install the necessary TwinCAT supplement (TF622x/TS622x)
2. Manually append or automatically scan the network (in case of automatic scan, the cable leading to the redundancy adapter shall be temporarily left open)
3. In case the redundancy ring is closed on a second network card in the controller, select the redundancy adapter (in case of automatic scan, the cable leading to the redundancy adapter shall be reconnected)
4. In case the redundancy ring is closed on a CU2508, select the adapter reference (in case of automatic scan, the cable leading to the redundancy adapter shall be reconnected)
5. Select the end point of the redundancy ring
6. Activate the configuration

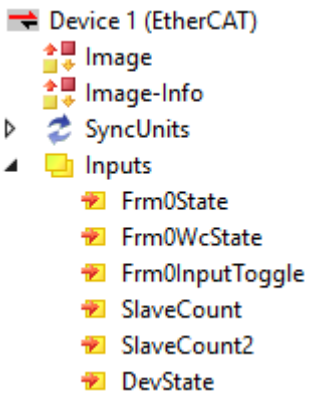


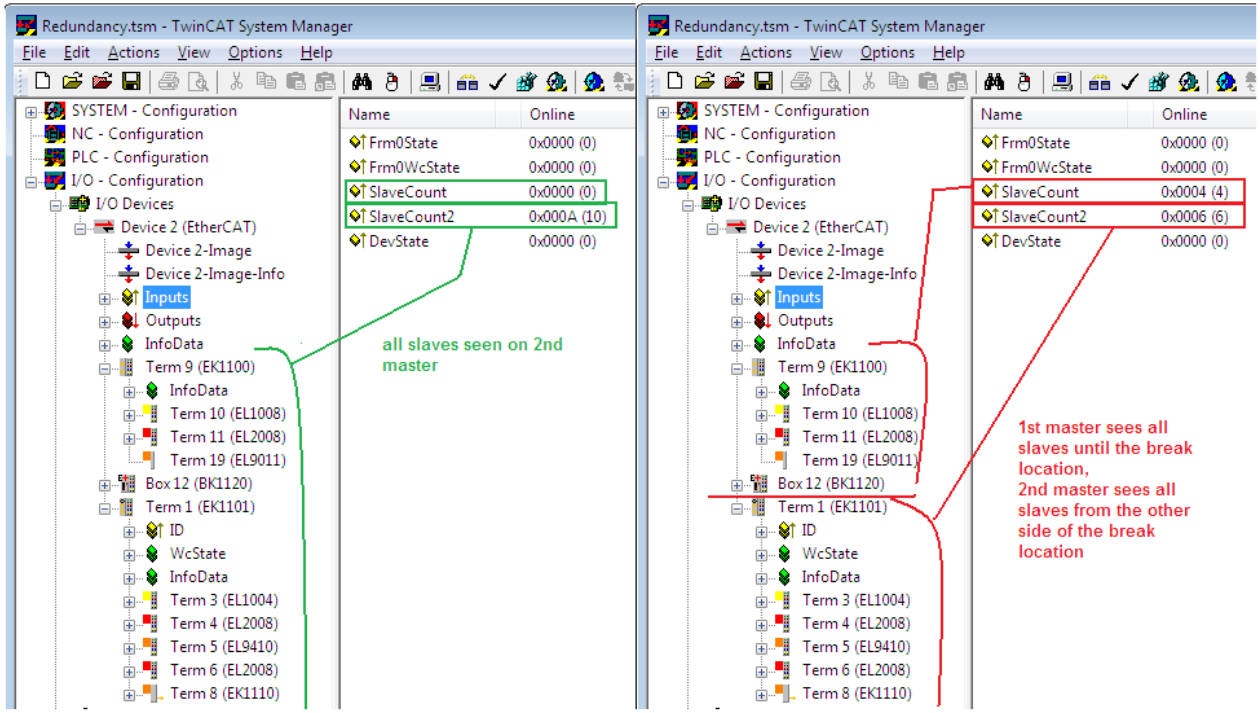
Diagnosis of Cable Redundancy in TwinCAT

BECKHOFF

When Redundancy is active, TwinCAT provides the following two variables:

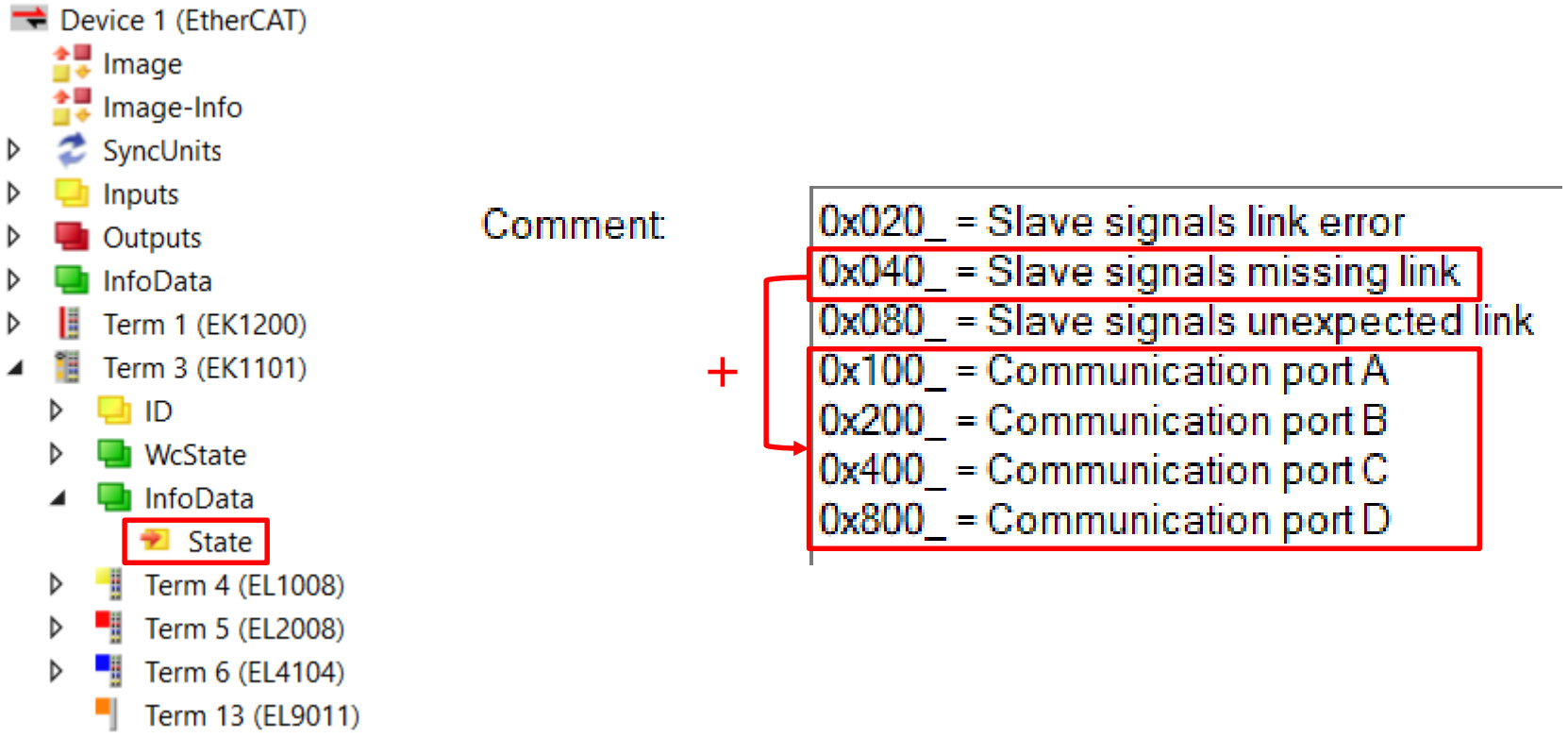
- **SlaveCount** : number of slaves processing the frames received from primary adapter.
- **SlaveCount2** : number of slaves processing the frames received by redundancy adapter





In case no interruption has occurred, all slaves will be counted by SlaveCount2.

Variable **State** provides additional information about the slave and port where the redundancy loop was interrupted.



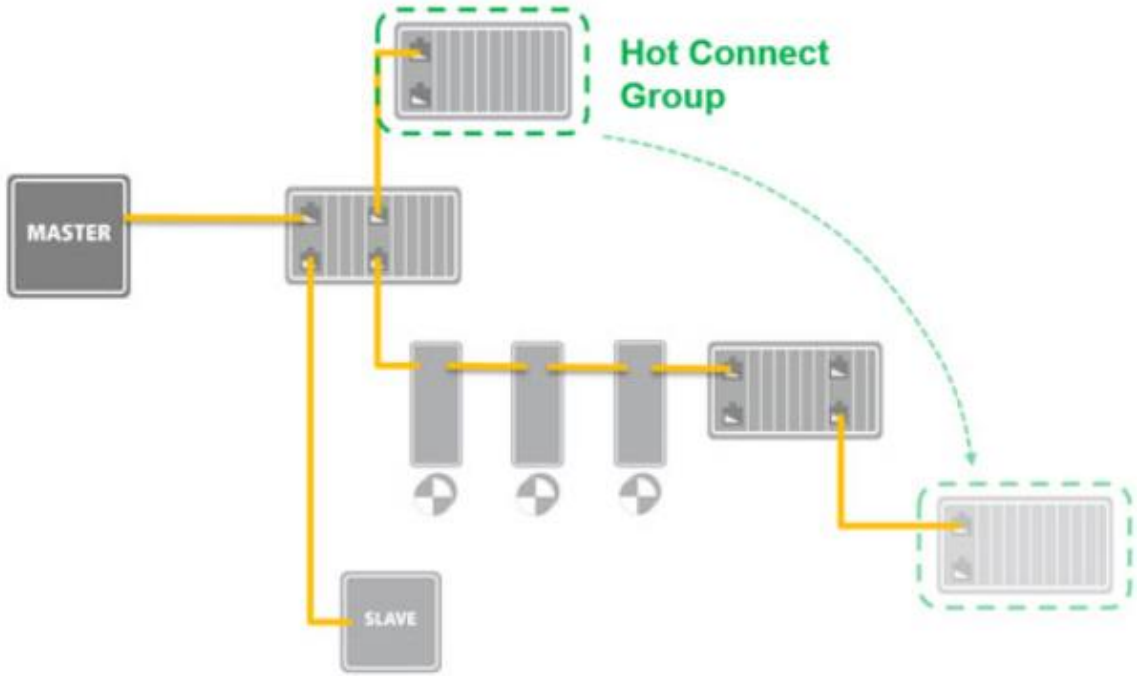
Hot Connect and Swap Prevention

What is Hot Connect?

BECKHOFF

Hot Connect:

- a slave (or an entire group of slaves, like e.g. I/O station) can be freely connected to and disconnected from the network without affecting other slaves.
- the slave (or group of slaves) can be connected at any point where a previous slave provides an unused output port.

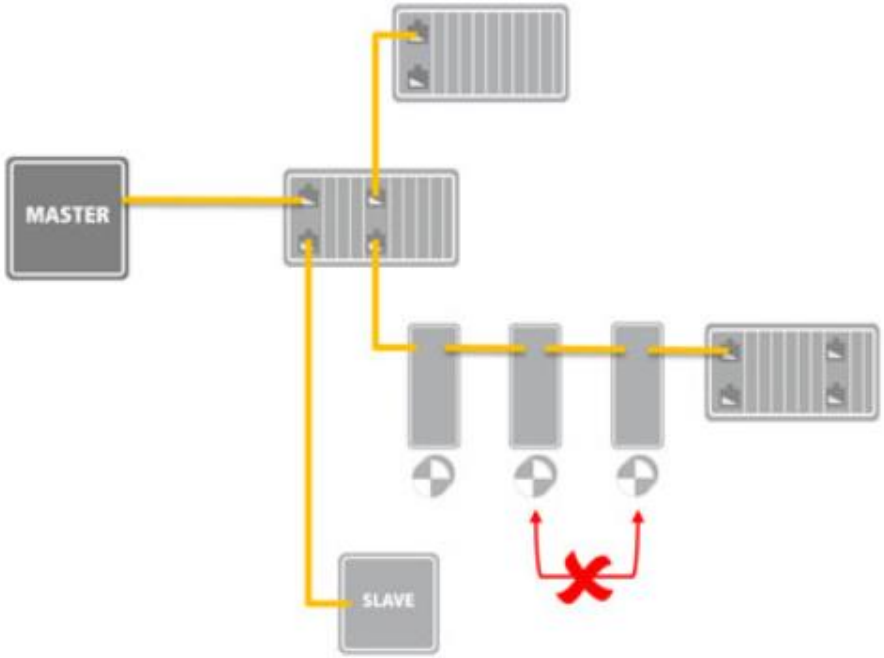


What is Swap Prevention?

BECKHOFF

Swap Prevention:

- during start-up, TwinCAT checks the identity of each slave (Vendor ID, Product Code, optionally Revision Number) according to its position and returns a VPRS error in case.
- identical slaves (e.g. batteries of drives) cannot be distinguished based on their identity, therefore an additional mechanism is needed in order to prevent unintended swap.



Explicit Device Identification

BECKHOFF

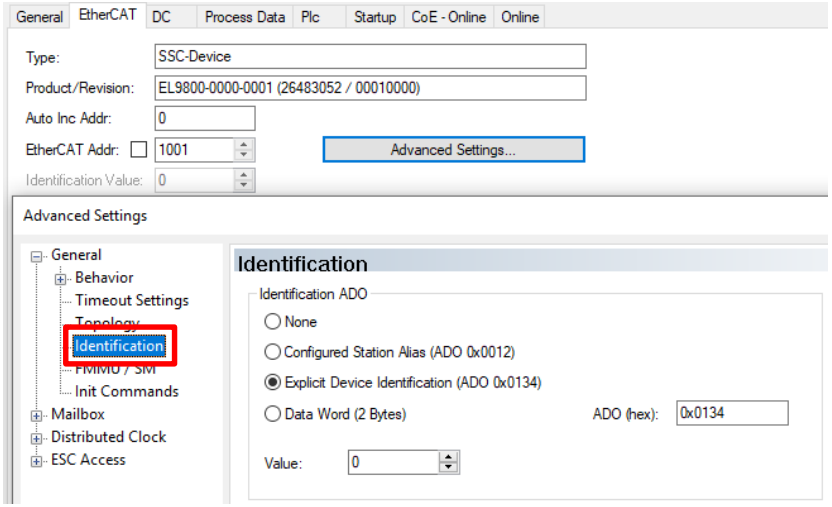
Common prerequisite for both Hot Connect and Swap Prevention user cases:

- possibility to uniquely identify an EtherCAT slave in the network independently from:
  - Its EtherCAT address (this is assigned by the master during start-up)
  - Its position in the network (this can change on purpose or by chance)

Users shall be able to store a **non-volatile** identifier, called **Device ID** or simply ID, in a slave device which will be involved in Hot Connect or Swap Prevention applications.

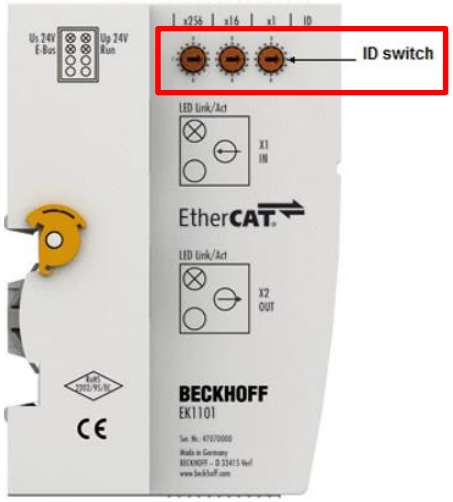
⇒ Explicit Device Identification

(in TwinCAT it is configured in the Advanced Settings of slave device)



How can Device ID be set on Slave?

If a slave supports this possibility, the Device ID can be manually set by means of an **ID Selector** like dip-switch, rotary switch or display…

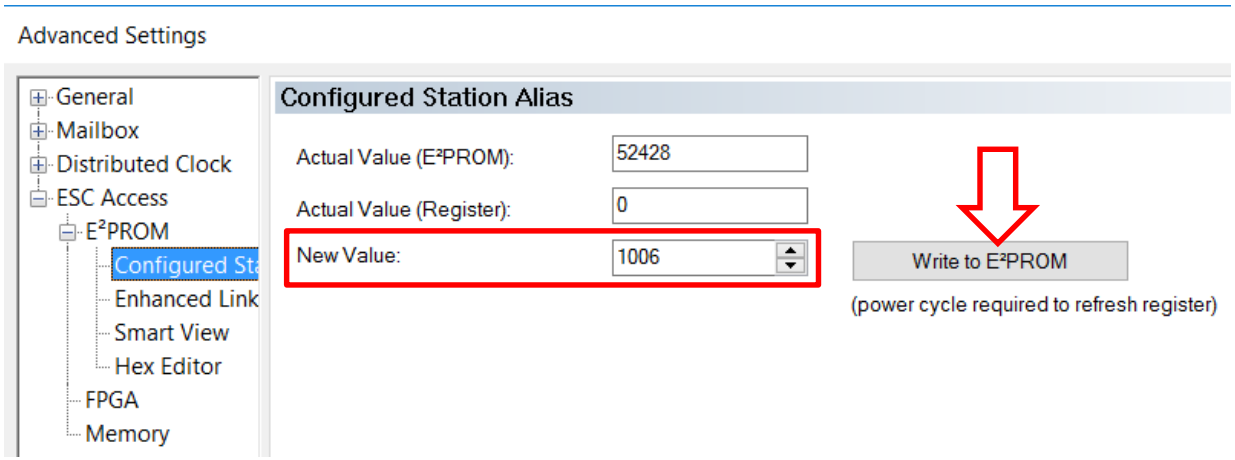


How can Device ID be set on Slave?

BECKHOFF

… for those slaves not providing an ID Selector for explicit device identification, back-up solution can be to store a **Station Alias** in EEPROM memory of slave from master side and use the Station Alias as identifier.

In TwinCAT, Station Alias can be written into the EEPROM from the Advanced Settings of slave device:



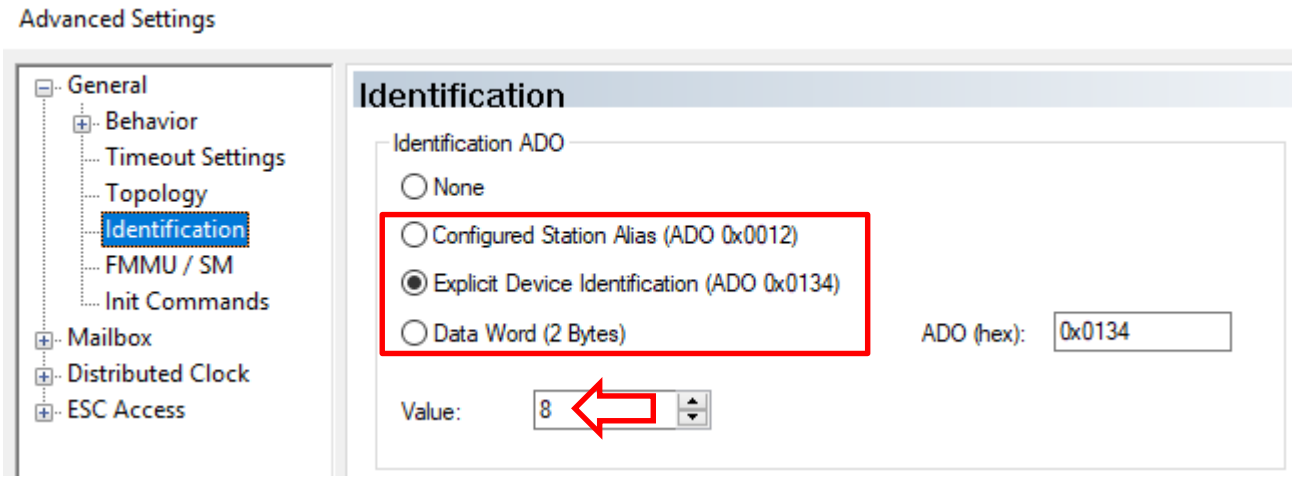
(compared to mechanical ID selectors, the solution with Station Alias in EEPROM is less comfortable in case of device replacement)

Does a Slave support ID Selector…

BECKHOFF

If a slave supports an ID Selector to set the Device ID, the information is coded into the corresponding XML file.

For such slave, one of the following three modes will be selected by default when the slave is scanned/appended into the TwinCAT configuration (the three modes differ by implementation details which are not relevant for the user, the default selection shall never be changed):



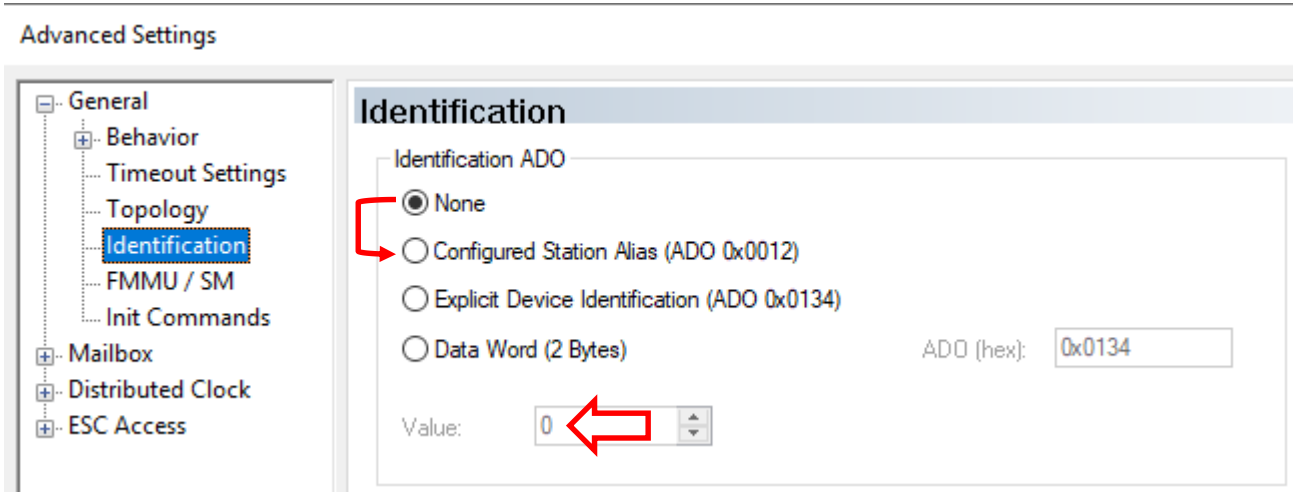
In text box “Value” the user shall set/check the identification value configured on ID Selector.

… or only the Station Alias?

BECKHOFF

If instead a slave only supports the back-up solution of Station Alias, the default selection when appending/scanning the slave into the TwinCAT configuration is “None”.

If the slave needs to be explicitly identified for Hot Connect or Swap Prevention application, the selection shall be changed to “Configured Station Alias (ADO 0x0012)”

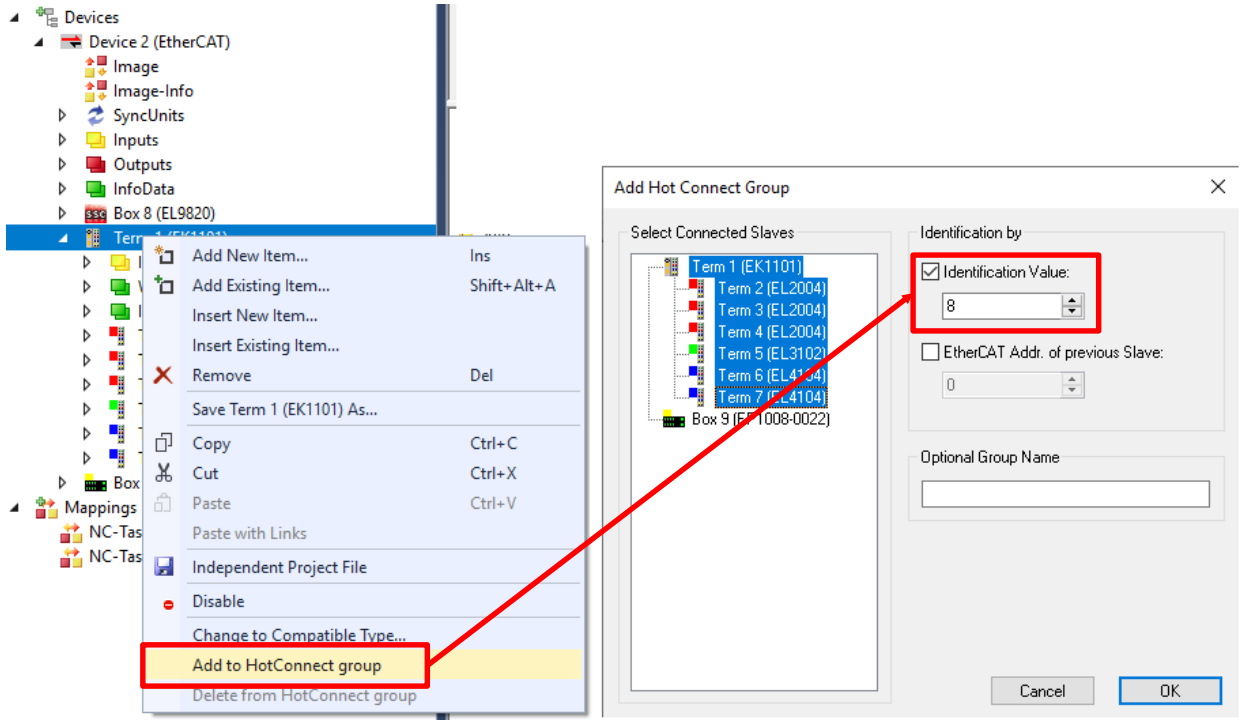


After the selection has been changed, in text box “Value” the user shall set/check the identification value configured on ID Selector.

How to Create Hot Connect Group

BECKHOFF

Click on the first slave of the HC group, optionally change the selection of which slaves shall belong to the group, set the identification value previously assigned to the first slave of the HC group, and click “OK”.



(TwinCAT should already pre-set the “Identification Value” to the one previously configured under Advanced Settings)

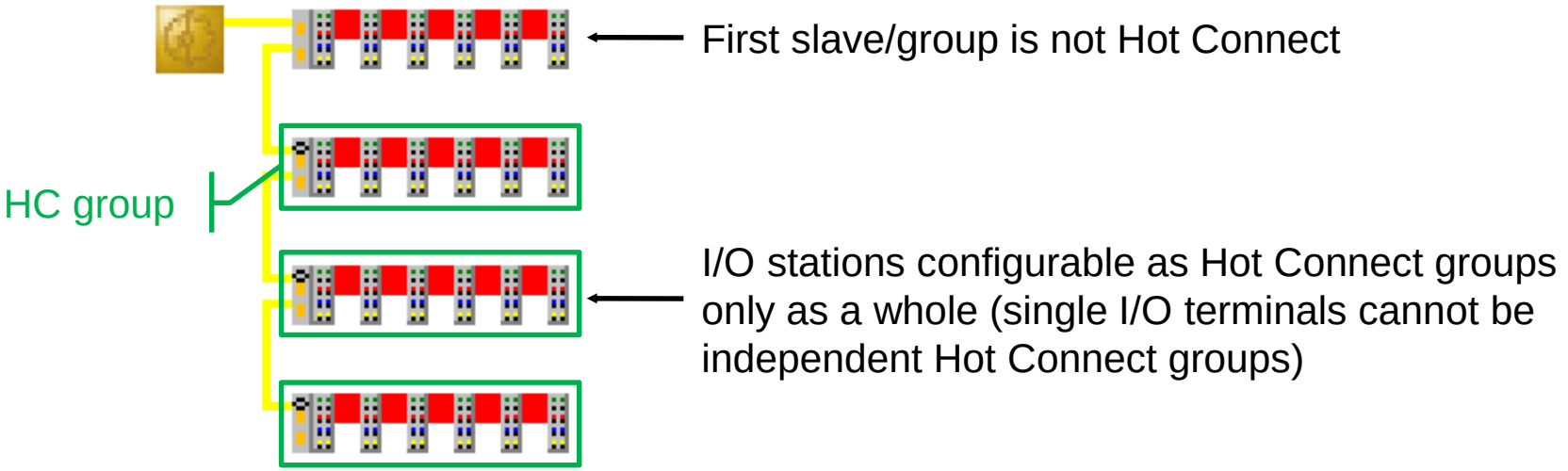
Notes to Hot Connect Implementation in TwinCAT

BECKHOFF

1. What can be configured as HC groups in TwinCAT are:
    1. Stand-alone slaves (e.g. IP67 boxes, servodrives, servovalves, …)
    2. IP20 I/O groups (EK)

    It is not possible to configure single IP20 terminals as independent Hot Connect groups

2. The first slave/group of slaves in a network cannot be configured as Hot Connect group.

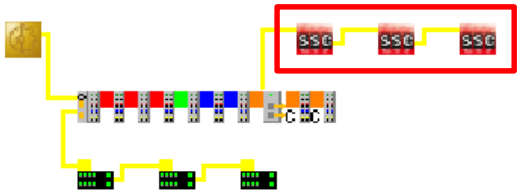


Sync Units instead of Hot Connect

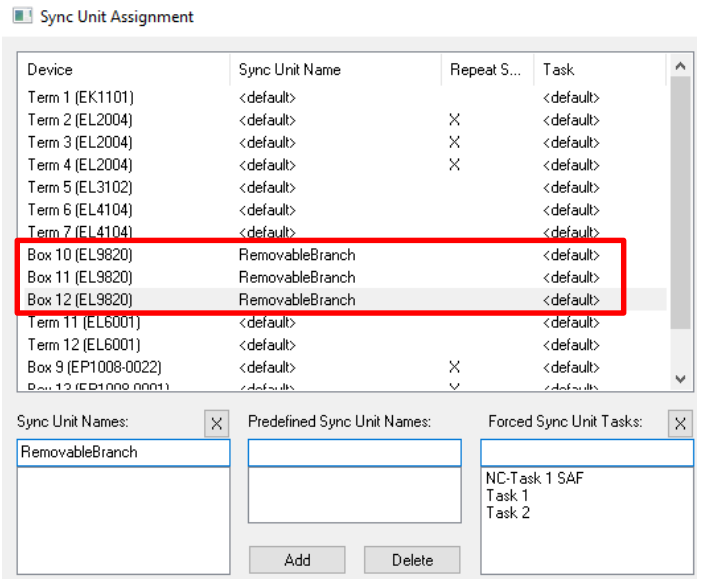
BECKHOFF

One of main scopes of Hot Connect is to make the position of a HC group within the network topology flexible.

If instead the goal is simply that the presence or absence of a slave/group of slaves does not affect the operation of the rest of the network, but the position of the slave/group of slaves never changes, it could be enough to configure a **dedicated Sync Unit** for this slave or group of slaves.



Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)	Utilization (%)	Size / Duration (µs)
0	LRD	0x09000000	1			2.000		
0	LRW	0x01000000	48	6	<default>	2.000		
0	LWR	0x01000800	18	5	<default>	2.000		
0	LRD	0x01001000	13	5	<default>	2.000		
0	LRW	0x01001800	282	9	RemovableBranch	2.000		
0	BRD	0x0000 0x0130	2	16		2.000	1.90 1.90	452 / 38.08

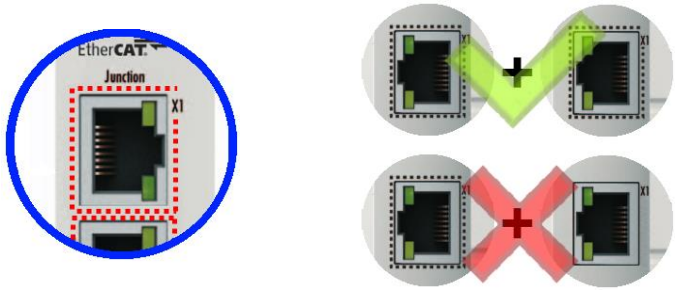


Fast Hot Connect

When a physical link is established, the two interfaces need to negotiate the communication parameters like speed, half-/full-duplex, … This takes time (several hundreds ms to some s).

With Fast Hot Connect, communication parameters are hard-coded in the two interfaces and therefore slaves can almost immediately start exchanging data (only few ms are needed).

- Beckhoff solution (not officially specified by EtherCAT standard)
- Intended for Hot Connect applications requiring very fast re-connection times
- Junction EK1122**-0080** and Coupler EK1101**-0080**
- Fast Hot Connect ports can be connected to other Fast Hot Connect ports **only**

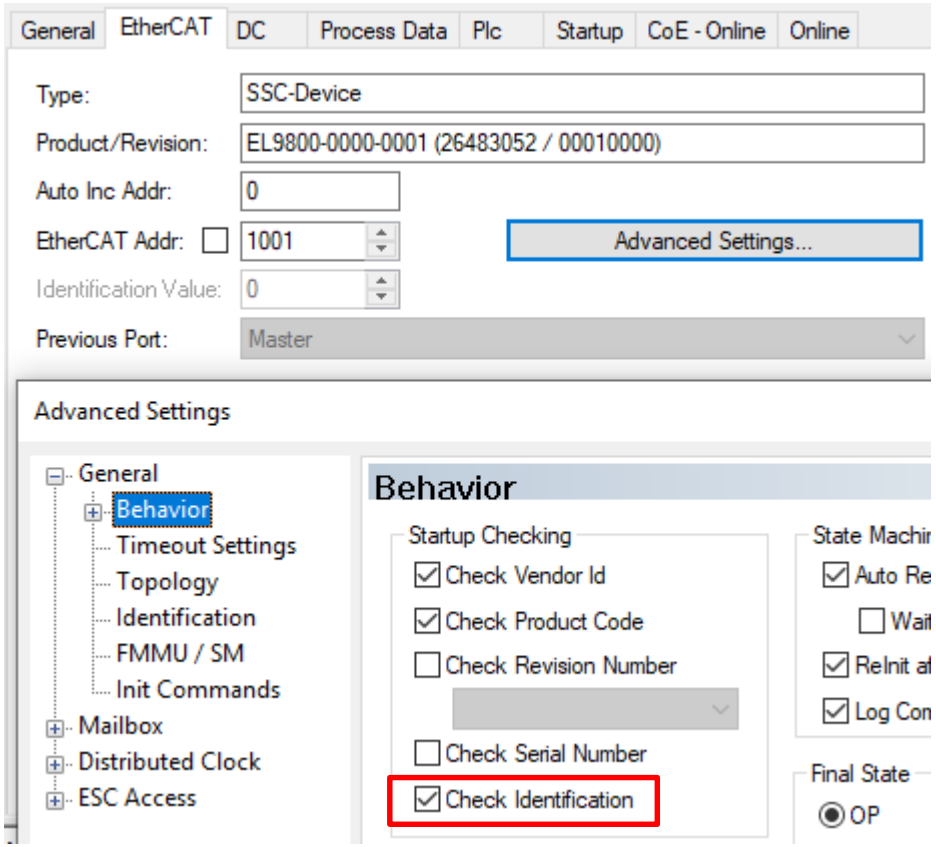


- The first slave in the network cannot have a Fast Hot Connect input port

How to force Identification Check for Swap Prevention

BECKHOFF

In order to force TwinCAT to check the Device ID of a slave device and to detect thereby unintentional swaps in the topology, after one of the identification mechanisms has been activated, the following flag shall be set in the Advanced Settings of the slave:



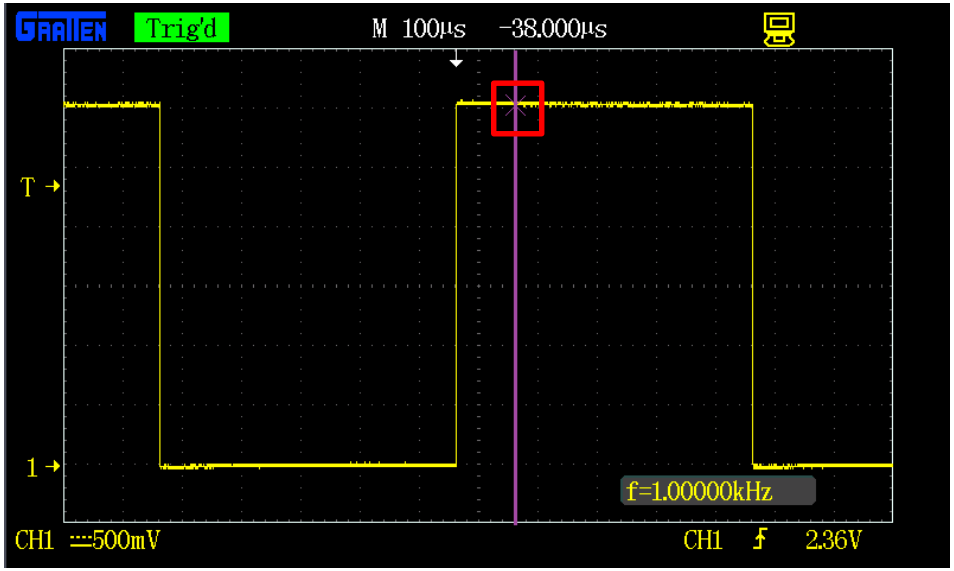
Diagnostics

How are EtherCAT errors typically detected?

BECKHOFF

Issues concerning the EtherCAT communication are typically detected because:

- The machine stops working, or the control application (PLC program, NC task…) receives data values clearly in disagreement with measured physical signals:



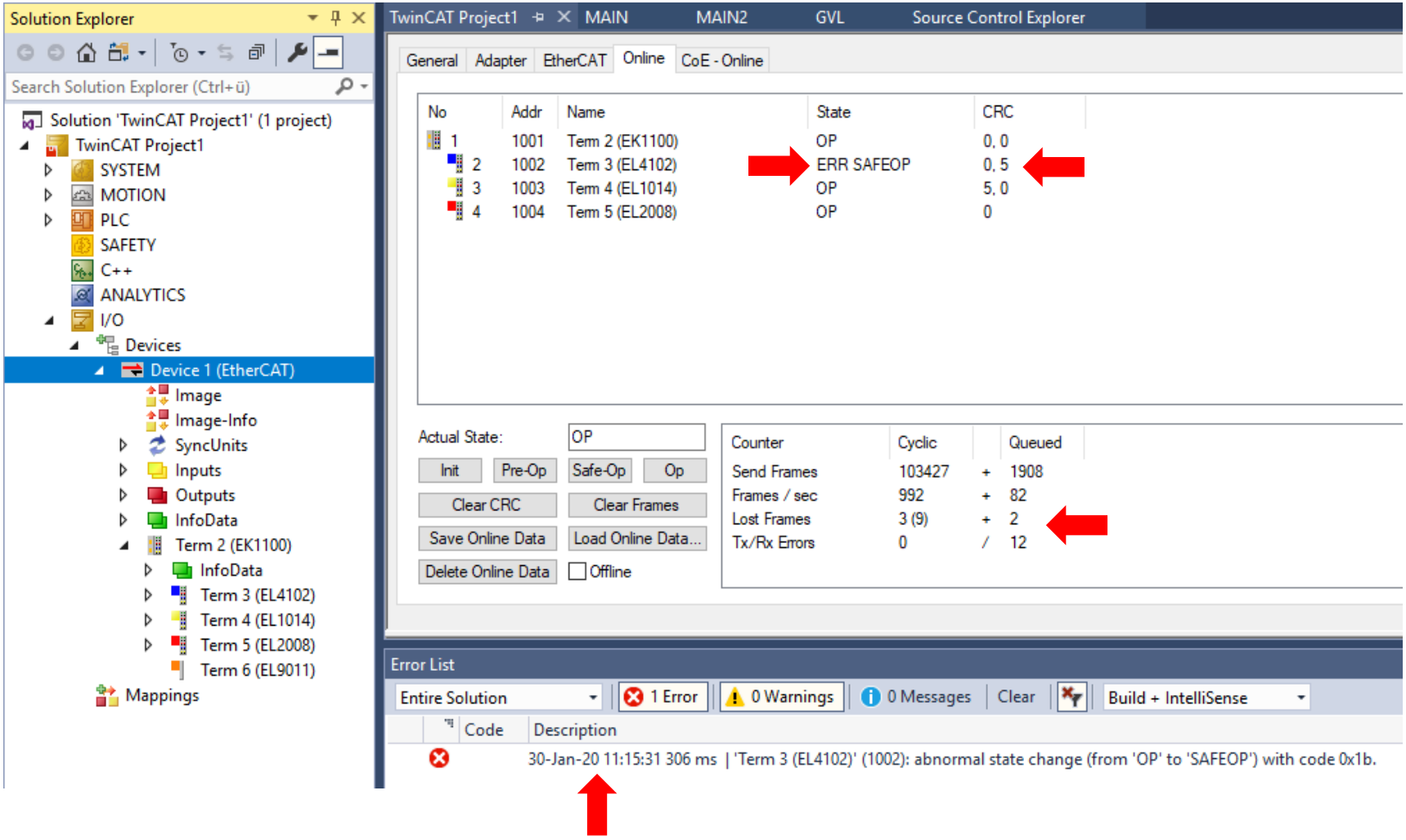
TwinCAT_Project1.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address
DigitalInput	BOOL	FALSE		%I*

How are EtherCAT errors typically detected?

BECKHOFF

- Errors are shown in the TwinCAT project (typically in master Online tab or TwinCAT Logger):



Error Types in an EtherCAT Network

BECKHOFF

Errors which could affect an EtherCAT network can be grouped in **2 categories**:

1. Hardware errors

- The physical medium is permanently or temporarily interrupted, and frames do not reach all network devices.
- Frames reach all the network devices, yet the information contained in them is corrupted.

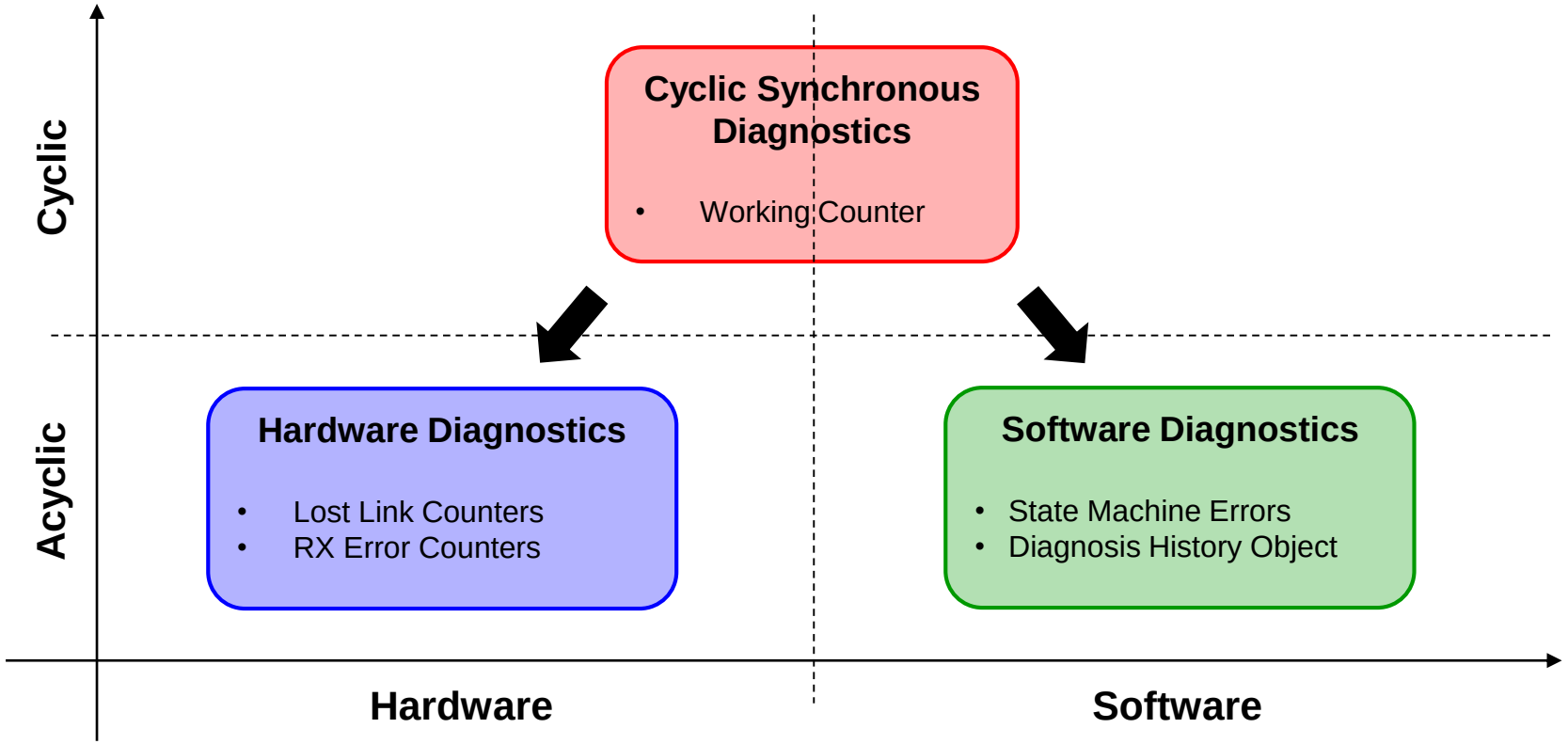
2. Software errors

- The slave cannot reach the Operational state during the initialization phase.
- The slave previously working error-free leaves the Operational state during operation

Error Types in an EtherCAT Network

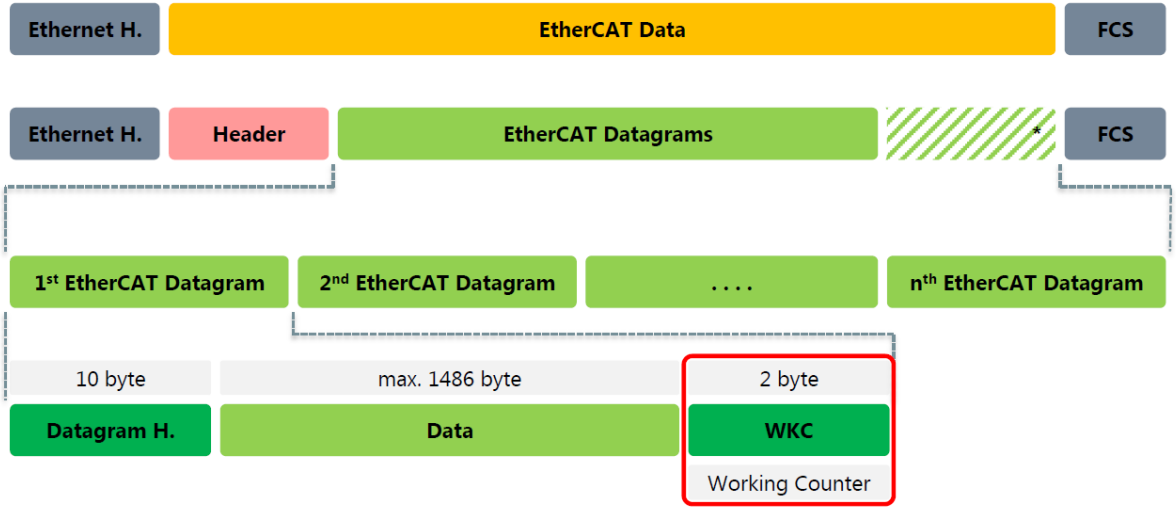
BECKHOFF

EtherCAT provides extensive **diagnostic information** both at hardware and at software level. This diagnostic information can be classified according to the following scheme:



Cyclic Synchronous Diagnostics

The Cyclic Synchronous Diagnosis – Working Counter



Each EtherCAT datagram ends with a 16-bit **Working Counter** (WKC), which is incremented by each addressed slave according to a simple rule:

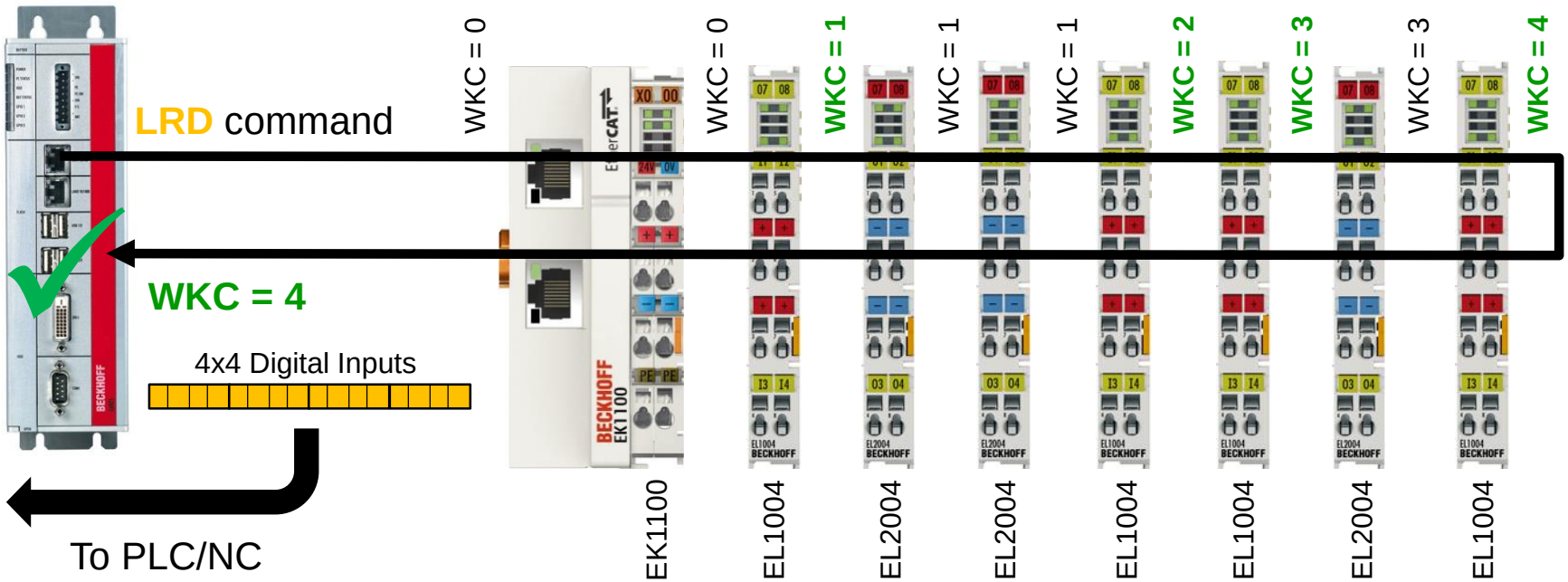
- Read-only commands (xRD): WKC+1 if data could be read from the slave.
- Write-only commands (xWR): WKC+1 if data could be written to the slave.
- Read+Write commands (xRW): WKC+1 if data could be read and WKC+2 if data could be written (i.e. WKC+3 if both reading and writing successful).

The Cyclic Synchronous Diagnosis – Working Counter

BECKHOFF

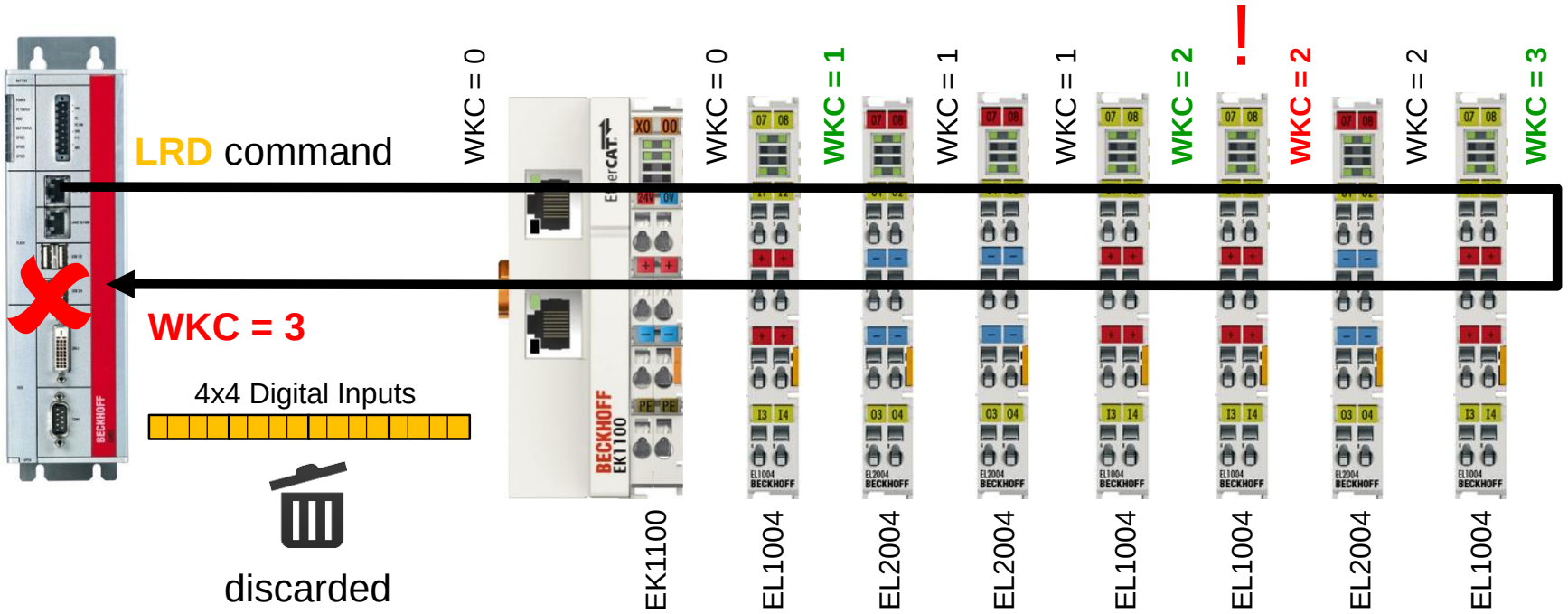
Master checks WKC value for each returning datagram:

- Value in datagram returning to master = expected value → **WKC valid**
  - ➢ input data in the datagram are forwarded to the control application (PLC, NC, …)



The Cyclic Synchronous Diagnosis – Working Counter

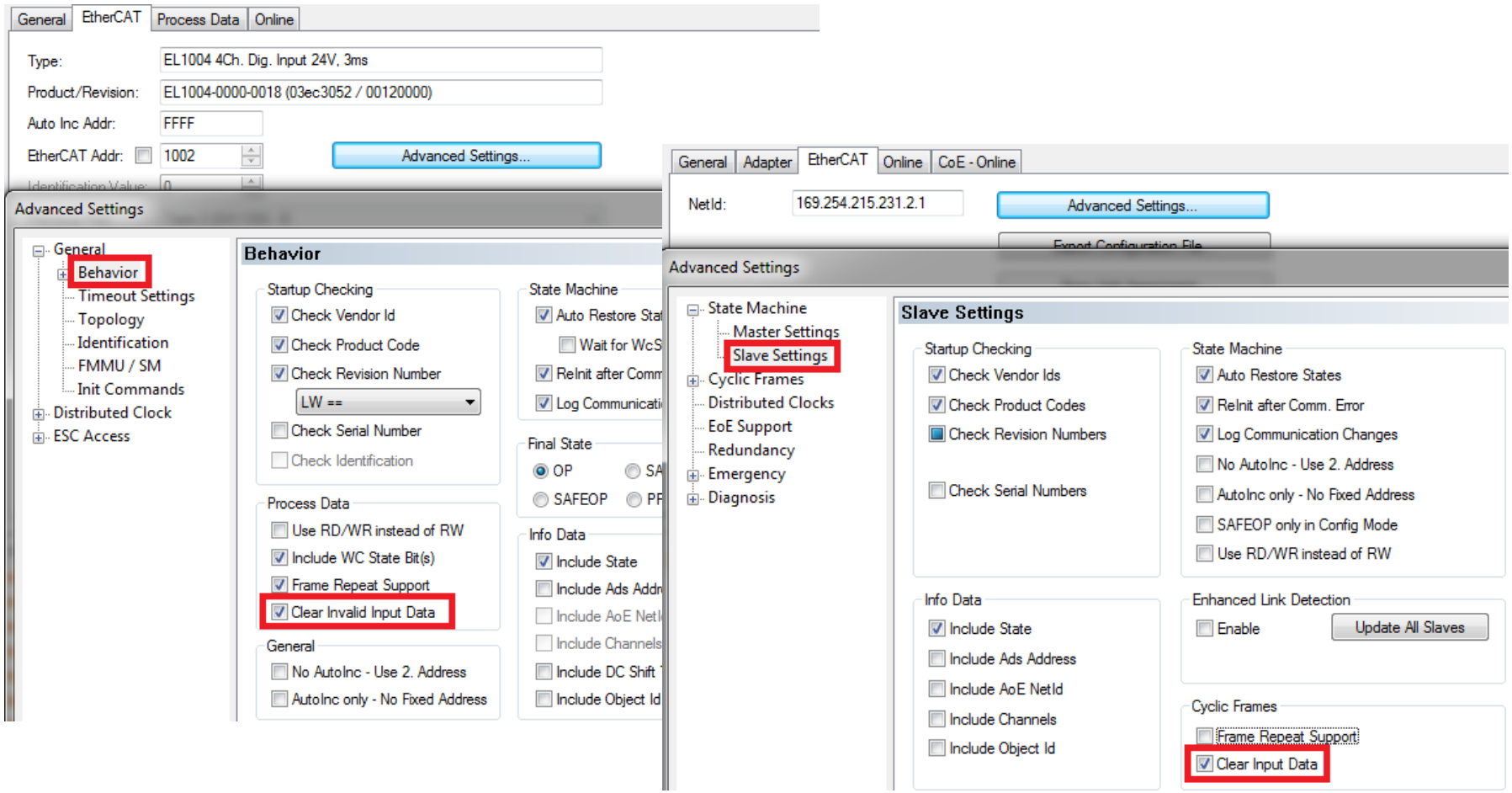
- Value in datagram returning to master ≠ expected value → **WKC invalid**
  - ➢ input data in the datagram are discarded (PLC/NC works with old data)



The Cyclic Synchronous Diagnosis – Working Counter

BECKHOFF

In latest TwinCAT 3.1 builds it is possible to configure single slaves (left) or all network slaves (right) so that the EtherCAT master sets invalid input data to zero instead of discarding them.



The Cyclic Synchronous Diagnosis – Working Counter

BECKHOFF

The Working Counter:

☹ Does not contain the information about which slave is suffering issues (any slave addressed by the datagram could have failed to increment its WKC).

☹ Does not contain the information about the error reason (e.g. a slave is damaged, or is not physically reached by frames, or simply is not in OP state).

☺ Comes **cyclically** with the frames, enabling therefore an immediate error reaction in the master.

☺ Is completed by **additional acyclic** hardware or software **diagnostic information** which helps to determine error location and cause.

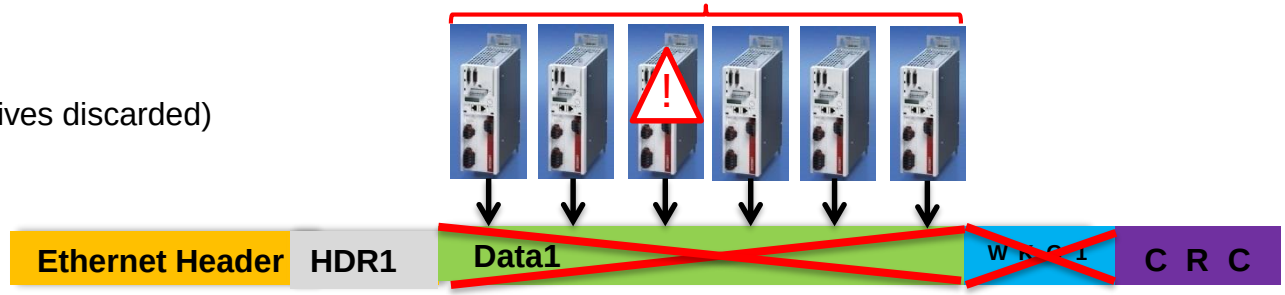
The Working Counter represents the front-end of EtherCAT diagnostic information, and can always be completed with deeper acyclic information.

BECKHOFF

Sync Units enable to group slaves into subsets served by separate datagrams, influencing therefore also the WKC diagnostic.

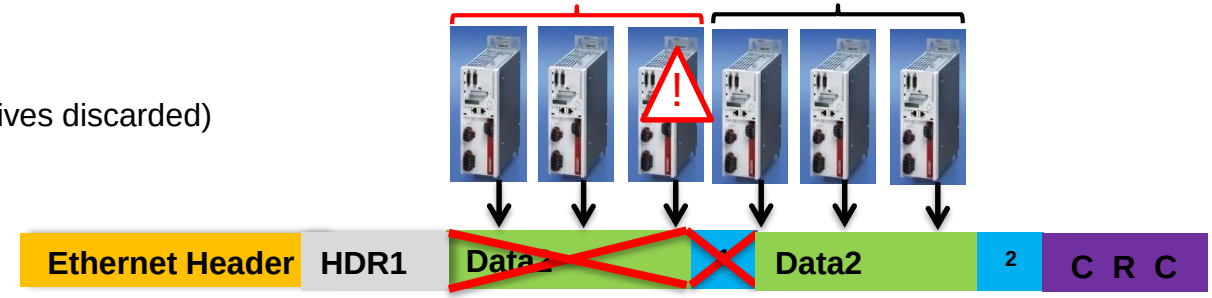
a) 1 Sync Unit

(wrong WKC → data of 6 drives discarded)



b) 2 Sync Units

(wrong WKC → data of 3 drives discarded)

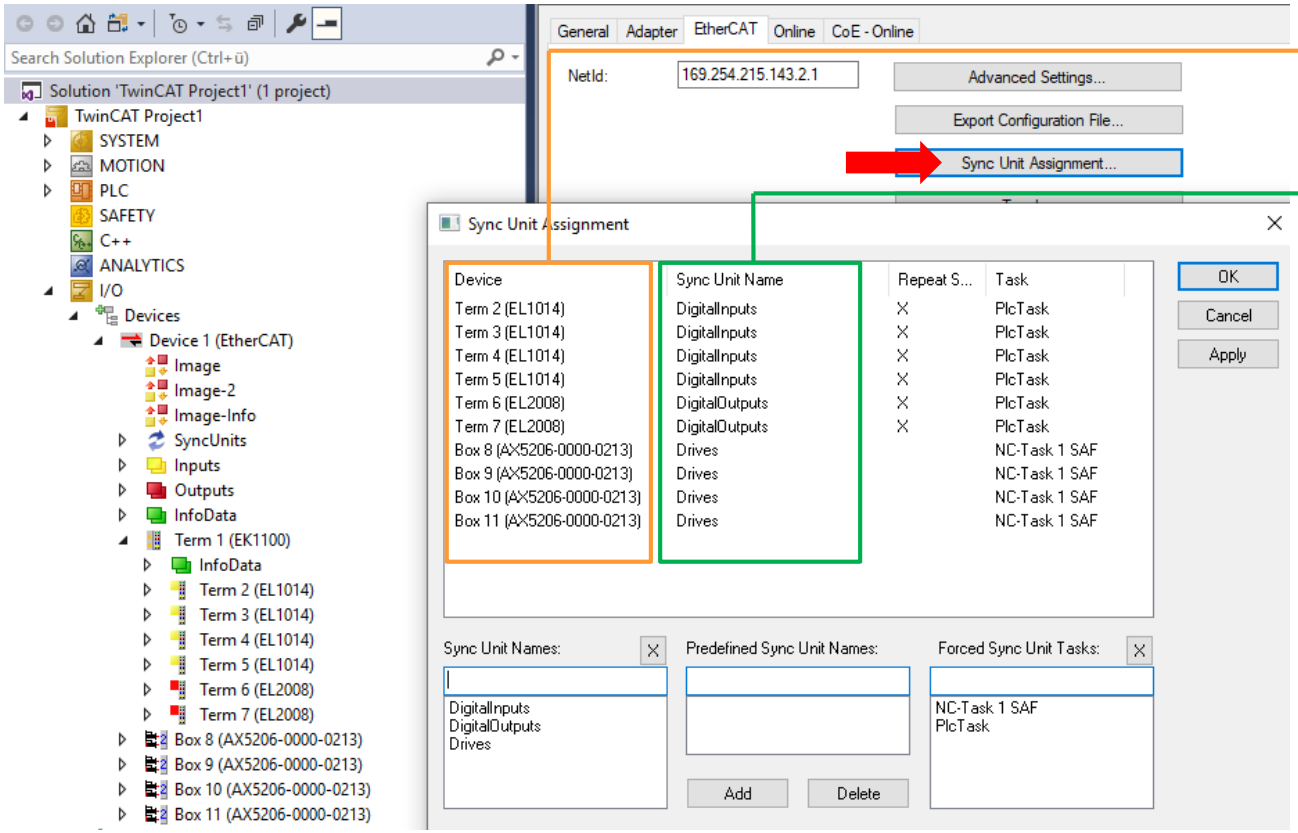


The Sync Unit configuration is an **optional** step intended for **optimization** purposes.

Sync Unit assignment is application-dependent, no golden rule is available.

Sync Unit Assignment in TwinCAT

BECKHOFF



List of slaves

Sync Unit assignment (configurable by user)

For different Tasks, separate frames

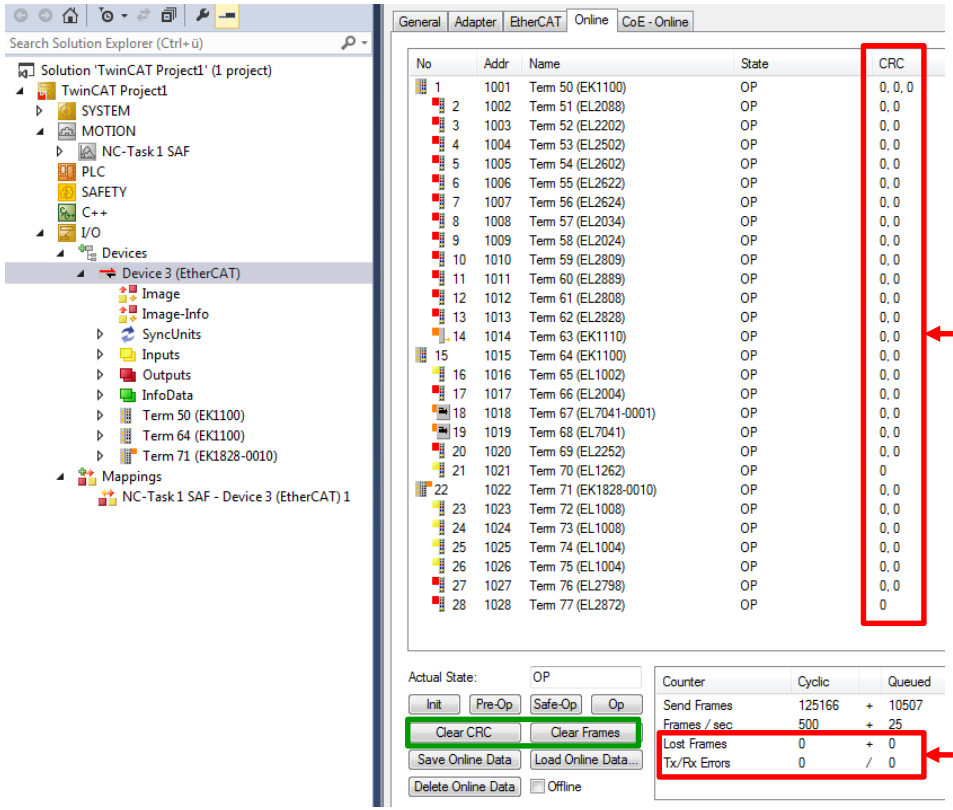
For different Sync Units within a task, separate datagrams

Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)	Utilization (%)	Size / Duration (µs)	Map Id	Flags
0 *	NOP	0x0000 0x0900	4			2.000				
0	ARMW	0xfff9 0x0910	4			2.000				
0	LRD	0x09000000	1			2.000				
0	LRW	0x01000000	80	12	Drives	2.000	0.70	153 / 14.16	0	
1 *	LRD	0x02000000	2	4	DigitalInputs	2.000				
1	LWR	0x02000800	2	2	DigitalOutputs	2.000				
1	BRD	0x0000 0x0130	2	11		2.000	0.67 1.38	2 * 58 / 13.44	0	

Hardware Diagnostics

Quick Overview of Hardware Status

BECKHOFF



The best overview of possible hardware issues on the network is provided by the master **Online** tab.

Slave hardware errors

Master hardware errors

Hardware errors could occur (and be counted by TwinCAT) at power-on, yet typically these errors do not represent a problem. Therefore, when checking possible hardware issues, it is suggested to clear all counters after the application has started, and to monitor if errors are counted during operation.

Preliminary Check - Topology Errors

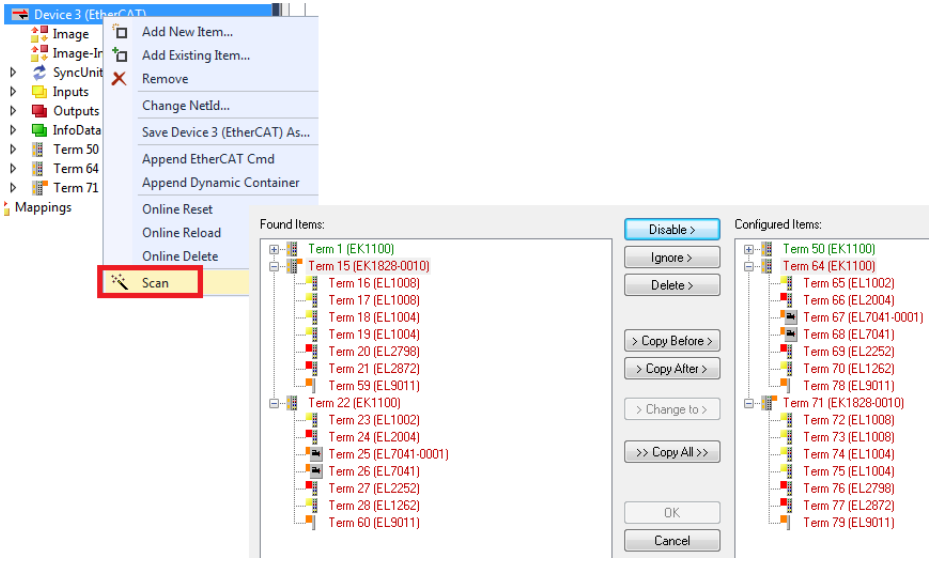
BECKHOFF

VPRS = **V**endor ID, **P**roduct Code, **R**evision Number, **S**erial Number: the topology discovered at network start-up does not correspond to the topology defined in the TwinCAT project: these errors could simply be due to a wrong cabling order!

No	Addr	Name	State
1	1001	Term 50 (EK1100)	OP LNK_MIS C
2	1002	Term 51 (EL2088)	OP
3	1003	Term 52 (EL2202)	OP
4	1004	Term 53 (EL2502)	OP
5	1005	Term 54 (EL2602)	OP
6	1006	Term 55 (EL2622)	OP
7	1007	Term 56 (EL2624)	OP
8	1008	Term 57 (EL2034)	OP
9	1009	Term 58 (EL2024)	OP
10	1010	Term 59 (EL2809)	OP
11	1011	Term 60 (EL2889)	OP
12	1012	Term 61 (EL2808)	OP
13	1013	Term 62 (EL2828)	OP
14	1014	Term 63 (EK1110)	OP
15	1015	Term 64 (EK1100)	INIT VPRS INIT_ERR LNK_ADD C
16	1016	Term 65 (EL1002)	INIT VPRS INIT_ERR
17	1017	Term 66 (EL2004)	INIT VPRS INIT_ERR
18	1018	Term 67 (EL7041-0001)	ERR INIT VPRS INIT_ERR
19	1019	Term 68 (EL7041)	ERR INIT VPRS INIT_ERR
20	1020	Term 69 (EL2252)	INIT VPRS INIT_ERR
21	1021	Term 70 (EL1262)	INIT VPRS INIT_ERR
22	1022	Term 71 (EK1828-0010)	INIT NO_COMM
23	1023	Term 72 (EL1008)	INIT NO_COMM
24	1024	Term 73 (EL1008)	INIT NO_COMM
25	1025	Term 74 (EL1004)	INIT NO_COMM
26	1026	Term 75 (EL1004)	INIT NO_COMM
27	1027	Term 76 (EL2798)	INIT NO_COMM
28	1028	Term 77 (EL2872)	INIT NO_COMM

Possible reasons are missing network devices or wrong cabling order.

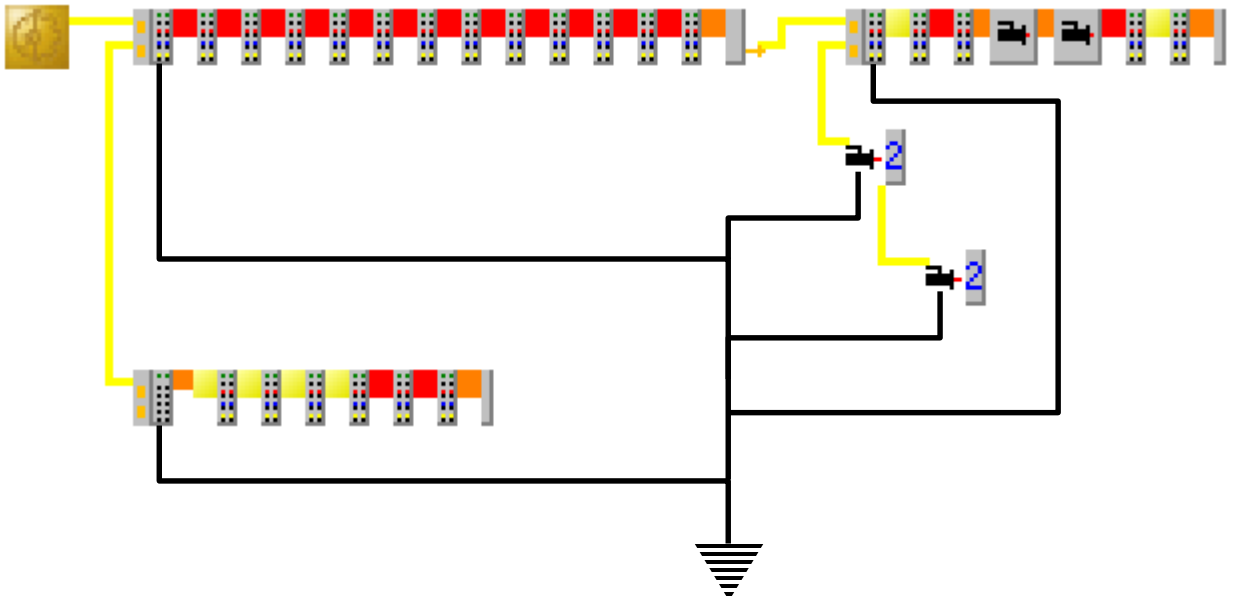
Start a new network scan (in Config Mode) in order to compare detected (left) with expected (right) topology:



Preliminary Check – Grounding and Shielding

BECKHOFF

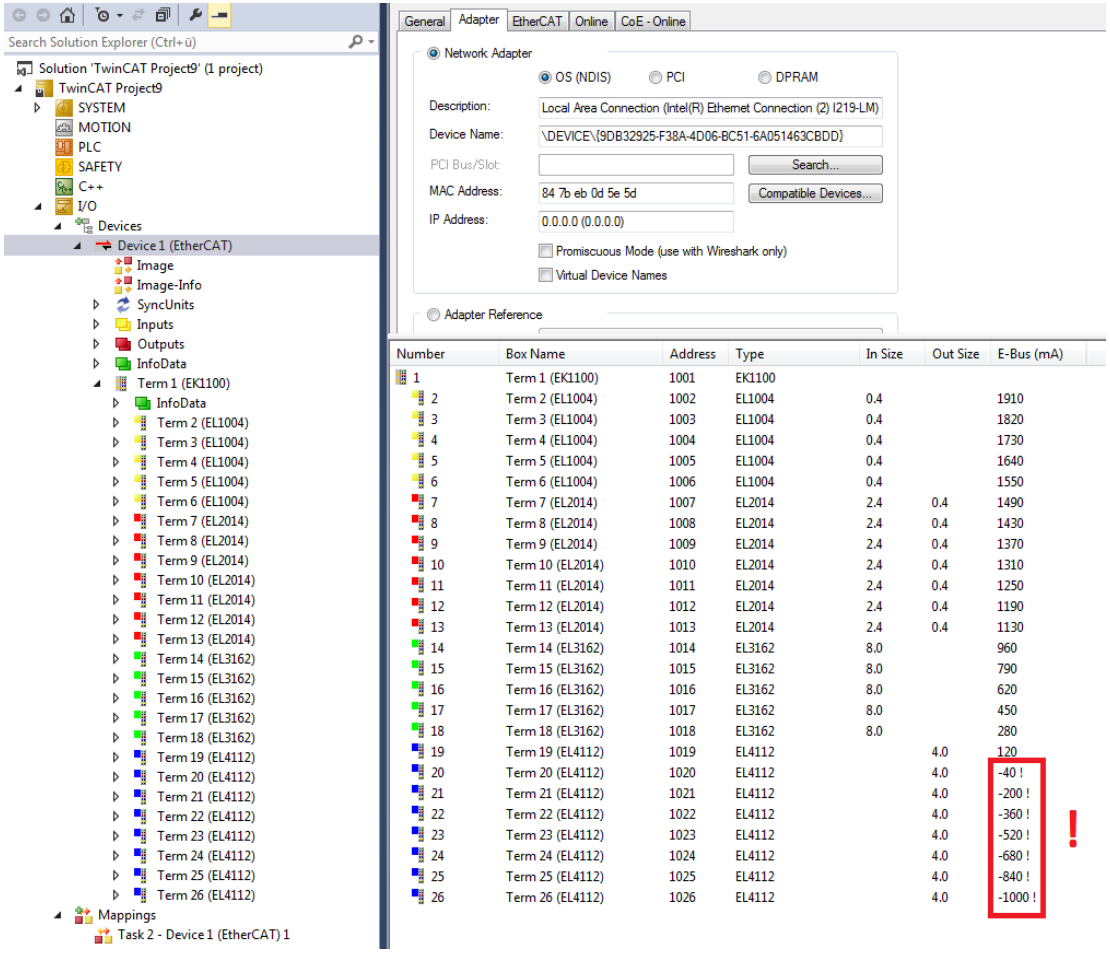
In case hardware errors occur, it is always suggested to check if all network devices share the same **ground potential** (as current loops caused by potential differences could cause data corruption):



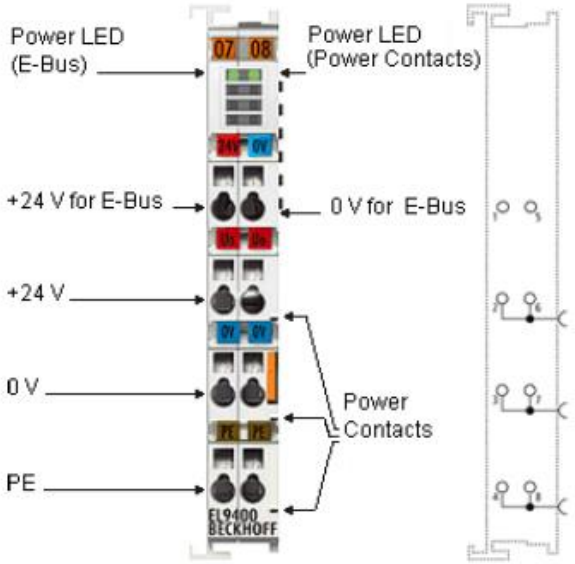
A more complete description of relevant aspects concerning the installation of an EtherCAT network is available in the **ETG.1600** “EtherCAT Installation Guidelines” document, available in the download area of www.ethercat.org.

Preliminary Check - EBUS Current

Each EL terminal sinks a certain amount of **EBUS current**. In case hardware errors occur, it is always suggested to check the available EBUS current for each I/O group:

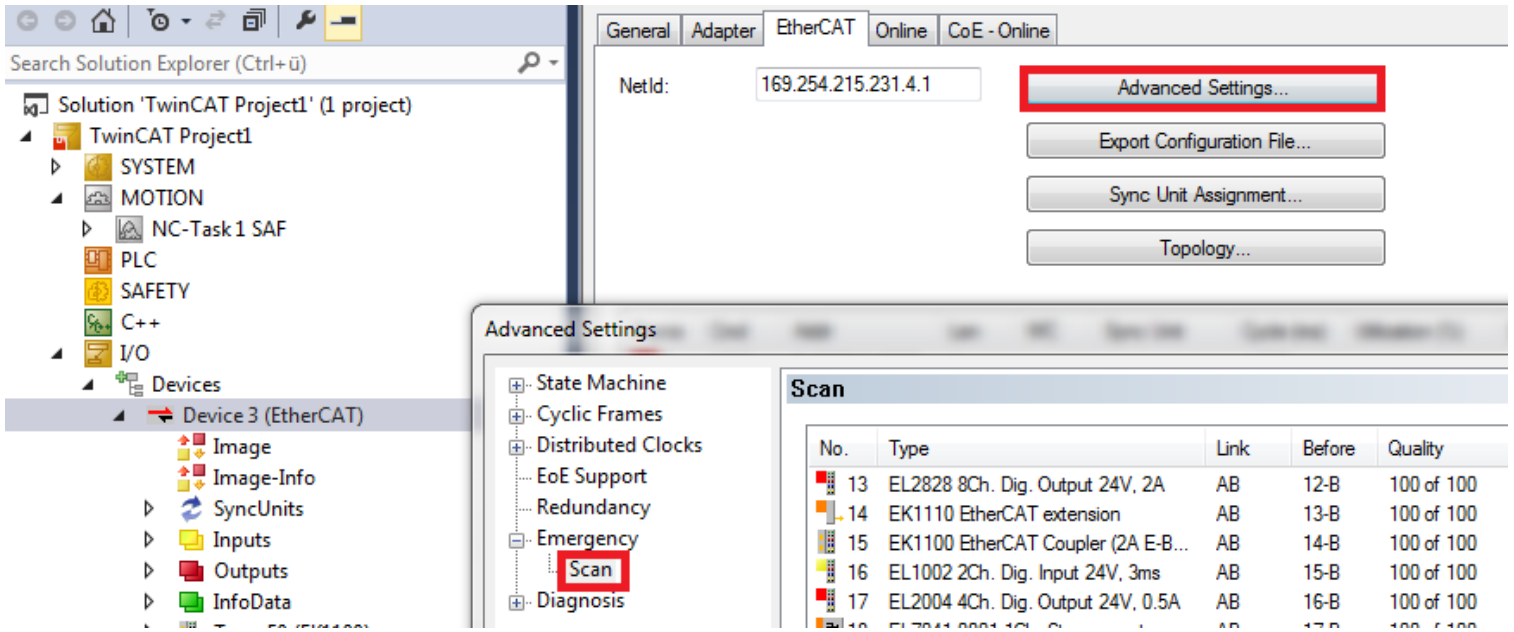


In case the current consumption limit is reached, an **EL94xx** terminal shall be inserted



Emergency Scan

The **Emergency Scan** procedure can be used to quickly test the physical link by sending a predefined amount of probe frames (TwinCAT shall be in **Config Mode**):



The Emergency Scan enables a quick check of permanent hardware issues on the network (damaged devices, cables or connectors).

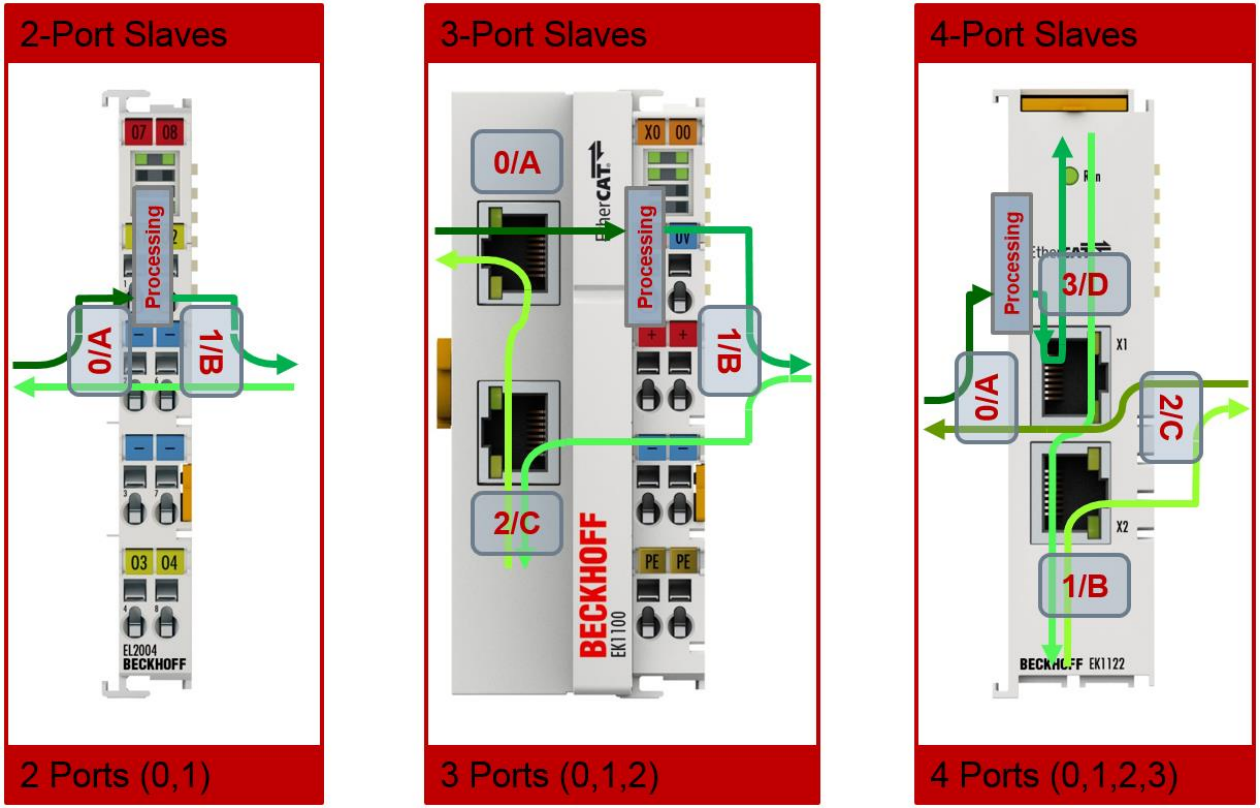
Random/sporadic disturbances are instead more difficult to detect by means of an Emergency Scan: in this case, a complete error counter analysis shall be performed.

Hardware Diagnostics and EtherCAT Ports

BECKHOFF

For a more detailed analysis, slaves provide **port-specific** hardware error counters.

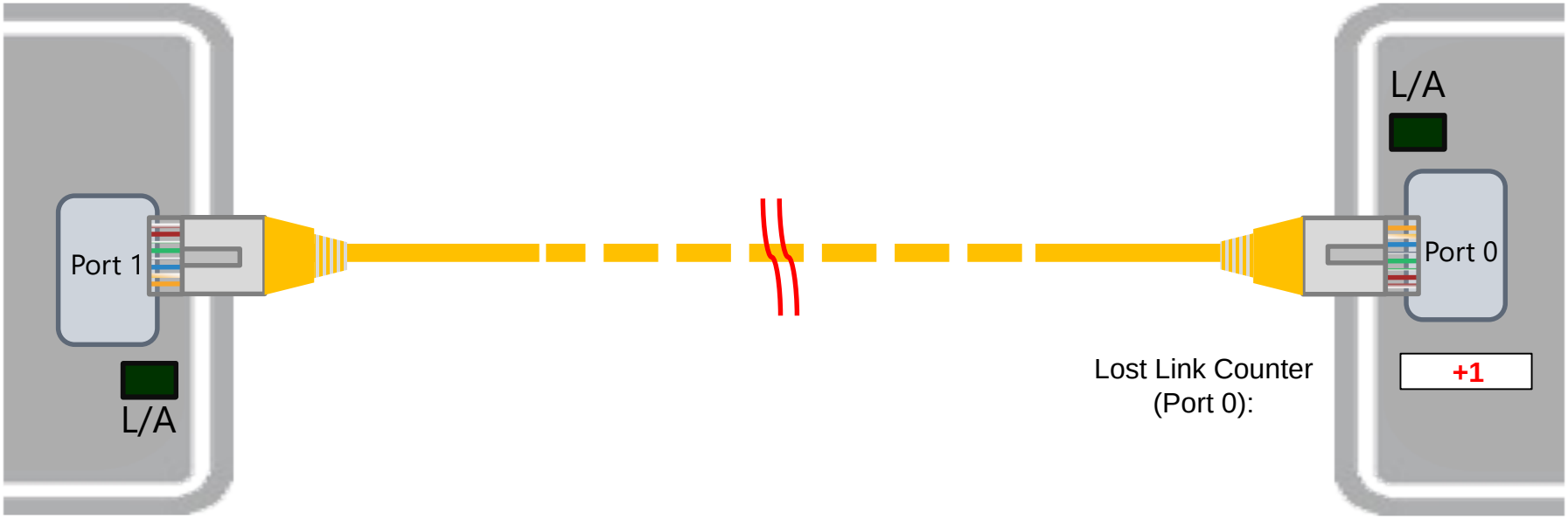
The EtherCAT standard numbers ports 0 to 3 (port 0 being always the IN port), while TwinCAT often names them A to D. The two naming rules are completely equivalent:



Lost Link Error Counters

BECKHOFF

Sometimes the physical link between two network devices can be completely interrupted: signals do not reach the neighbouring device at all.



Most likely reasons for link loss are:

- Damaged cables or connectors (for cabled connections) or poor/oxidized contacts (for EBUS connections).
- Loss of power supply for one or more slave devices.

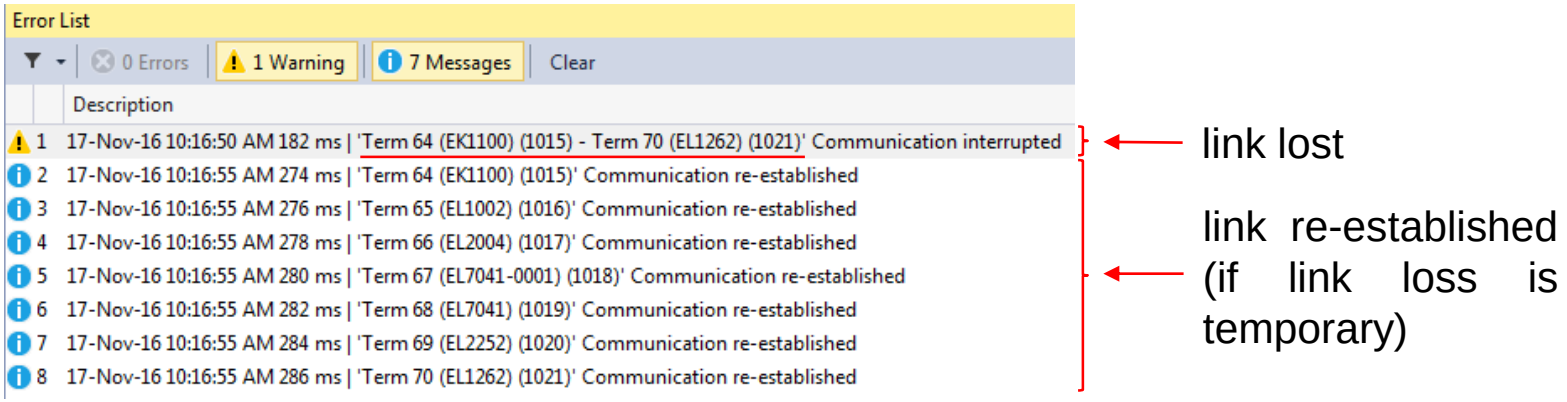
Lost Link Error Counters

BECKHOFF

When the physical link on a port is interrupted, the corresponding **Lost Link Counter** is incremented by the slave.

Register	Length	Meaning
0x0310	1 word	Lost Link Counter port 0 + Lost Link Counter port 1
0x0312	1 word	Lost Link Counter port 2 + Lost Link Counter port 3

Permanent or temporary changes in the phisical link status are reported in TwinCAT logger (and saved in the Windows Log as well):

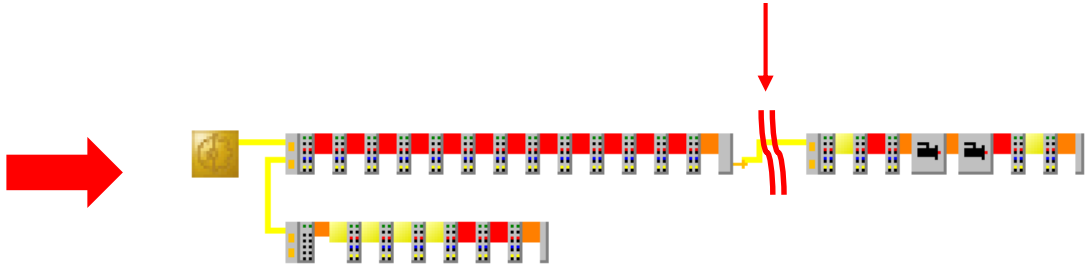


Link/Activity LEDs

BECKHOFF

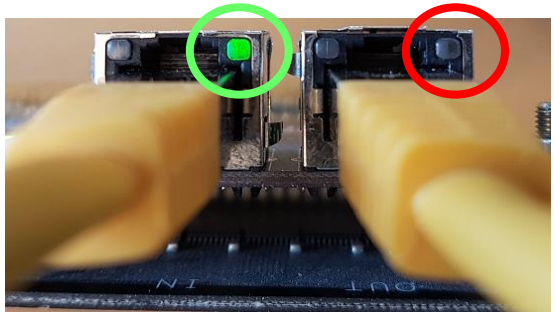
Permanent link interruptions can be easily detected also without analysis of the Lost Link Counters, simply by checking the master Online tab:

No	Addr	Name	State
3	1003	Term 52 (EL2202)	OP
4	1004	Term 53 (EL2502)	OP
5	1005	Term 54 (EL2602)	OP
6	1006	Term 55 (EL2622)	OP
7	1007	Term 56 (EL2624)	OP
8	1008	Term 57 (EL2034)	OP
9	1009	Term 58 (EL2024)	OP
10	1010	Term 59 (EL2809)	OP
11	1011	Term 60 (EL2889)	OP
12	1012	Term 61 (EL2808)	OP
13	1013	Term 62 (EL2828)	OP
14	1014	Term 63 (EK1110)	OP LNK MIS B
15	1015	Term 64 (EK1100)	INIT NO_COMM
16	1016	Term 65 (EL1002)	INIT NO_COMM
17	1017	Term 66 (EL2004)	INIT NO_COMM
18	1018	Term 67 (EL7041-0001)	INIT NO_COMM
19	1019	Term 68 (EL7041)	INIT NO_COMM
20	1020	Term 69 (EL2252)	INIT NO_COMM
21	1021	Term 70 (EL1262)	INIT NO_COMM
22	1022	Term 71 (EK1828-0010)	OP
23	1023	Term 72 (EL1008)	OP
24	1024	Term 73 (EL1008)	OP
25	1025	Term 74 (EL1004)	OP
26	1026	Term 75 (EL1004)	OP
27	1027	Term 76 (EL2798)	OP
28	1028	Term 77 (EL2872)	OP



Possible causes for a permanent link loss are unplugged or badly damaged cables, or powered-off devices.

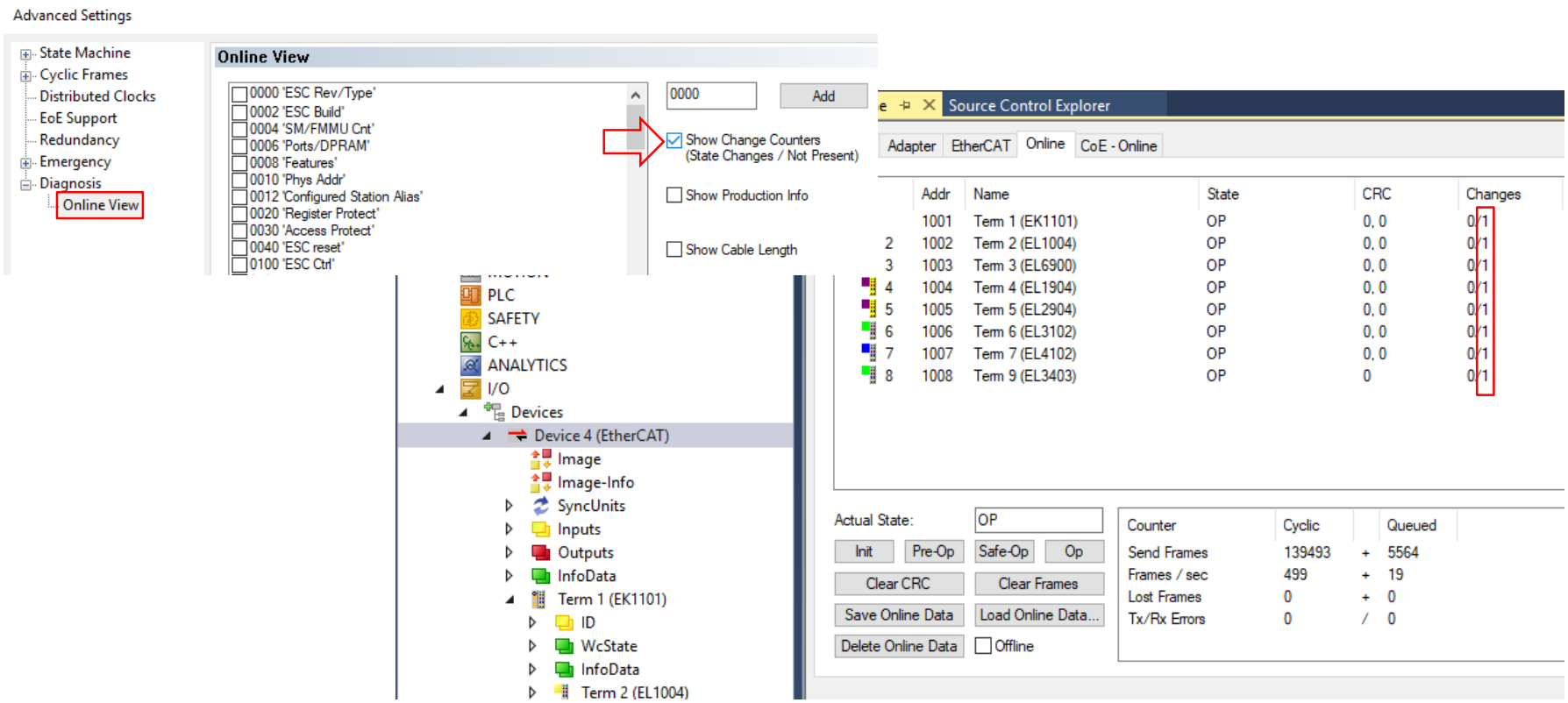
A visual inspection of Link/Activity LEDs (mandatory for all ports with removable connectors) can also help detecting the interruption in this case:



Simplified Monitoring of Lost Link Errors

BECKHOFF

Link losses can be monitored in the Online tab of the EtherCAT master by means of the following incremental counter after enabling them in the Advanced Settings of the master:

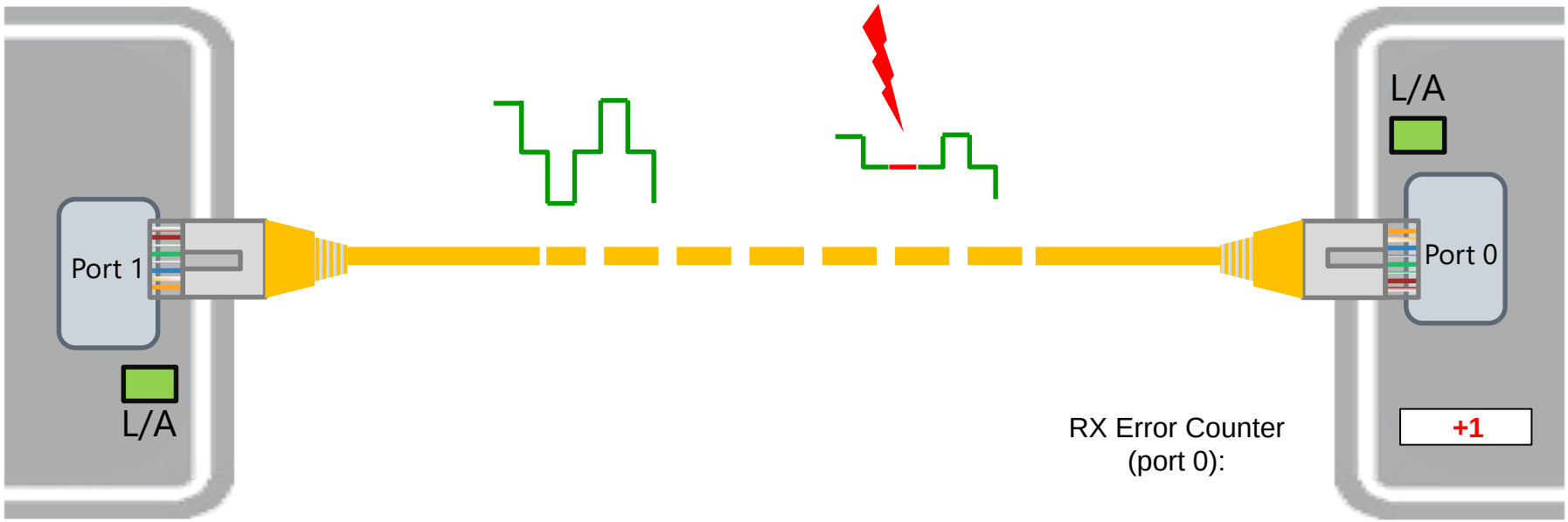


These counters report the number of times the physical connection with the slave was lost (1 is start value, > 1 means that the physical link was lost at least once during operation).

RX Error Counters

BECKHOFF

Sometimes, although hardware signals reach the neighbouring device, the received message does not correspond to the originally transmitted one:



Most likely reasons for frame corruption are:

- External EMC disturbances (especially in case of sporadic counter increment)
- Damaged devices (especially in case of fast and systematic counter increment)

RX Error Counters

BECKHOFF

When the hardware signal is corrupted, the corresponding **RX Error Counter** is incremented by the slave:

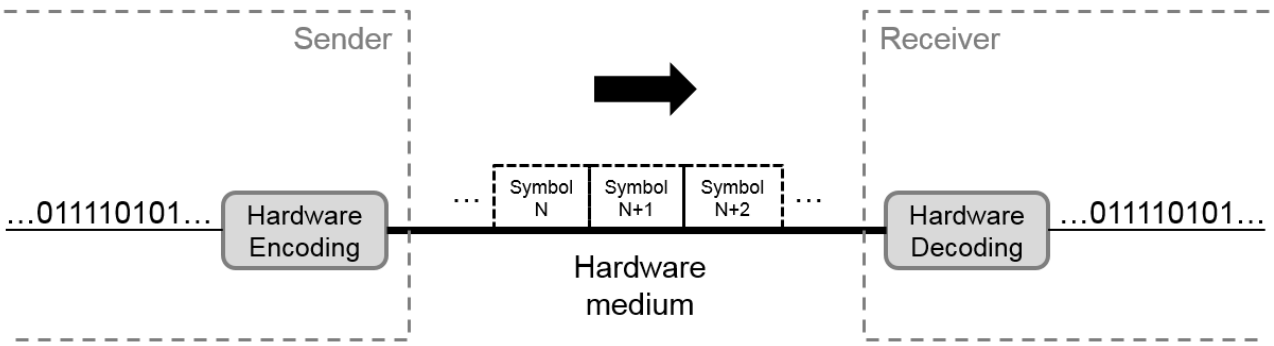
Register	Length	Meaning
0x0300	1 word	RX Error Counter (Physical Layer + Frame Error Counter) port 0
0x0302	1 word	RX Error Counter (Physical Layer + Frame Error Counter) port 1
0x0304	1 word	RX Error Counter (Physical Layer + Frame Error Counter) port 2
0x0306	1 word	RX Error Counter (Physical Layer + Frame Error Counter) port 3

RX Error Counters are formed in turn by an Frame Counter and a Physical Layer Error Counter. Frame errors and physical layer errors both refer to signal corruption, yet they are detected at two different levels in the slave architecture and therefore have slightly different meanings.

Signal Transmission on Physical Medium

BECKHOFF

In order to be transmitted on a physical medium, the logical sequences of 0s and 1s need to be encoded into predefined current/voltage levels (or level transitions):

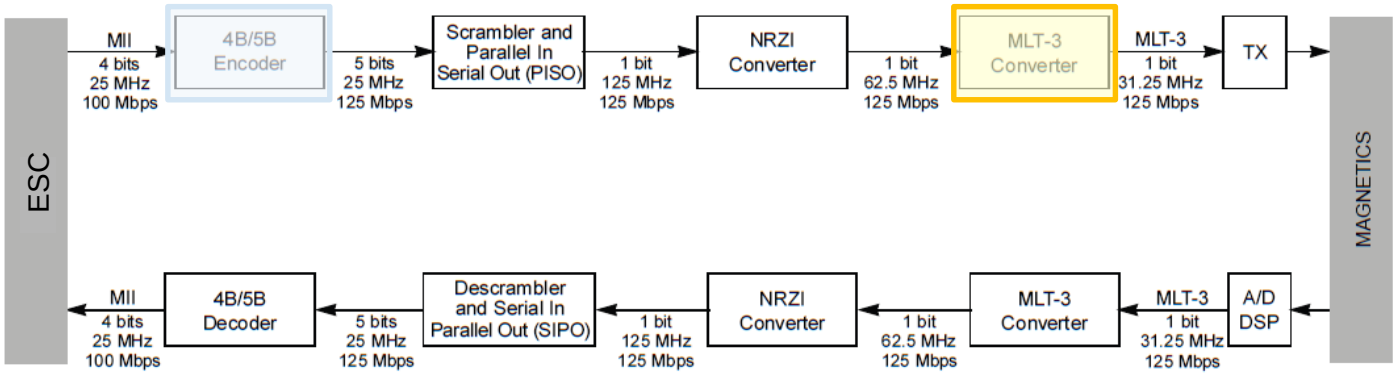


- (Sequences of) current/voltage values are called **symbols**.
  - Not all possible (sequences of) values have a meaning by a specific hardware encoding standard: therefore, there are **valid** and **invalid** symbols.
  - Symbols are continuously transmitted on the physical medium, both **within** and **outside** Ethernet frames (in order to allow the receiver to detect a link loss).

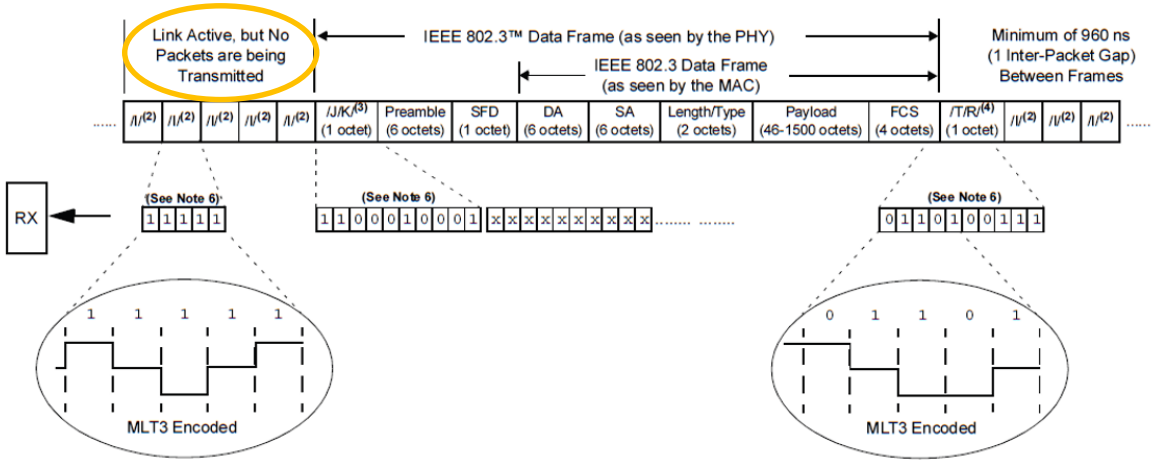
- Communication consists in sequences of symbols.
  - Sequences carrying meaningful information correspond to Ethernet **frames**.
  - Sequences transmitted between meaningful data correspond to **frame intergaps**.

In Depth – 100BASE-TX Hardware Coding Standard

BECKHOFF



Control Character	5b symbols	Purpose
JK	11000 10001	Sync, Start delimiter
II	11111 11111	Not Used
TT	01101 01101	FDDI end delimiter
TS	01101 11001	Not Used
IH	11111 00100	SAL
TR	01101 00111	100BASE-TX end delimiter
SR	11001 00111	Not Used
SS	11001 11001	Not Used
HH	00100 00100	HDLC0
HI	00100 11111	HDLC1
HQ	00100 00000	HDLC2
RR	00111 00111	HDLC3
RS	00111 11001	HDLC4
QH	00000 00100	HDLC5
QI	00000 11111	HDLC6
QQ	00000 00000	HDLC7

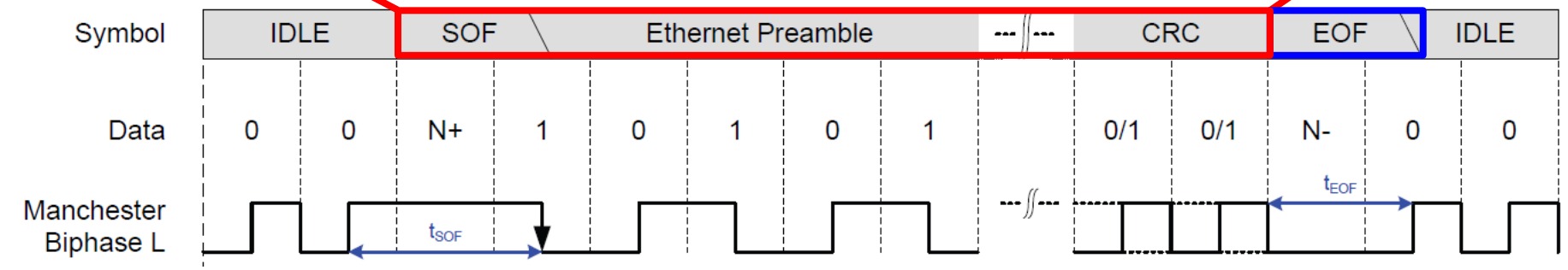


Multi-Layer Transmit 3 Code:

- Logic 0 : LO → HI transition
- Logic 1 : HI → LO transition

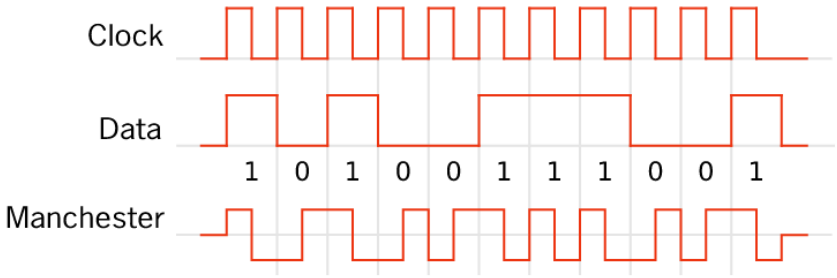
EBUS is a LVDS-based signal compiant to ANSI/TIA/EIA-644:

Layer	Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interpacket gap
	7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46(42)[b]–1500 octets	4 octets	12 octets
Layer 2 Ethernet frame			← 64–1518(1522) octets →						
Layer 1 Ethernet packet	← 72–1526(1530) octets →								



Biphase-L Mancherster Code:

- Logic 0 : LO → HI transition
- Logic 1 : HI → LO transition



RX Error Counters

BECKHOFF

Hardware errors can be distinguished into two types, both counted by RX Error Counters:

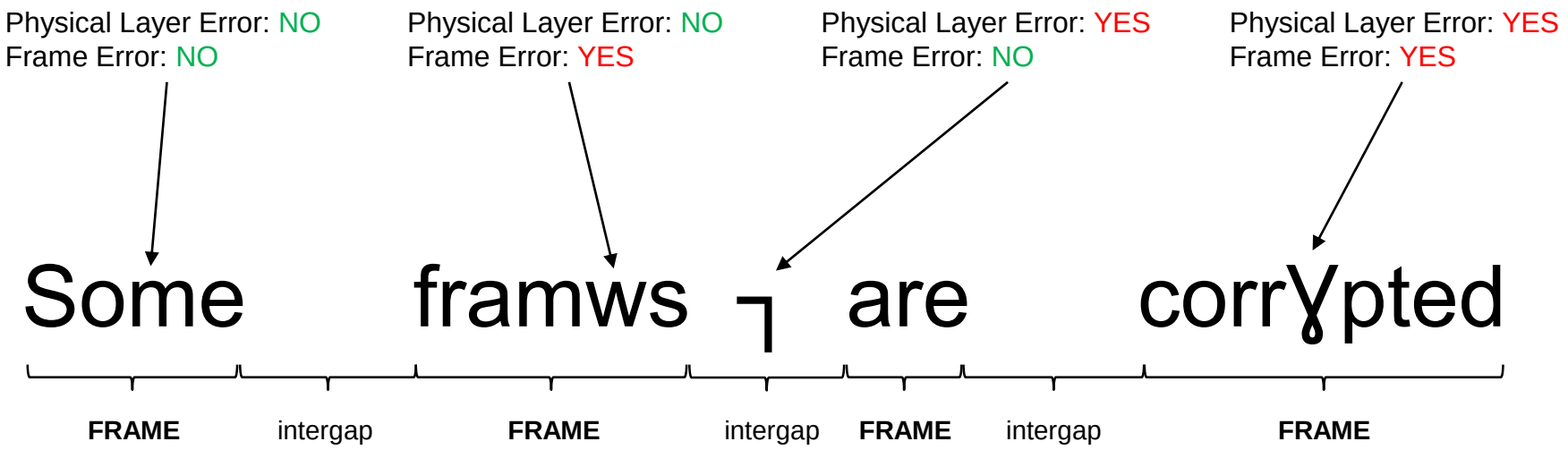
- **Physical Layer Errors**:
  - Individual symbol errors (detected as invalid by the specific hardware coding).
  - Can occur both within and outside frames (each physical interface transmits idle symbols even when no frames are being transmitted)
  - → **Physical Layer Error Counters** (high-byte of RX Error Counters)

- **Frame Errors**:
  - Cyclic Redundancy Check errors on whole frames.
  - Can occur only within frames (the checksum is verified only for Ethernet frames).
  - → **Frame Error Counters** (low-byte of RX Error Counters)

Analogy Physical Layer and Invalid Frame Errors

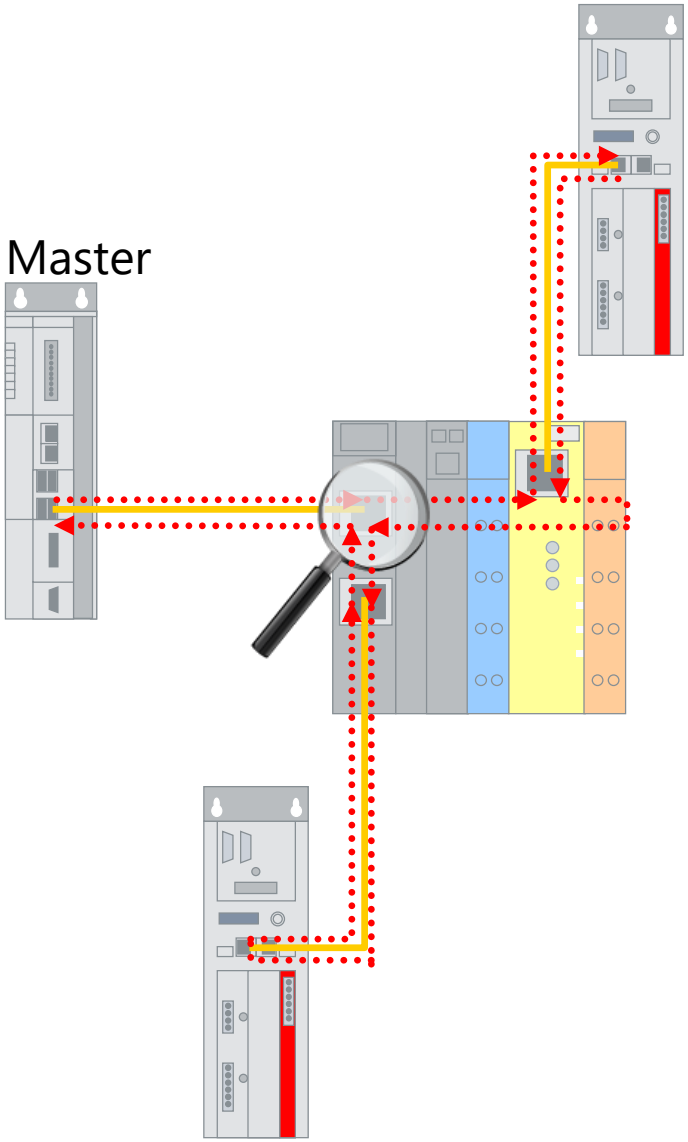
The transmission of data on a hardware communication medium can be compared to a standard written language.

Physical Layer and Frame Errors have slightly different interpretation, as shown by the example:

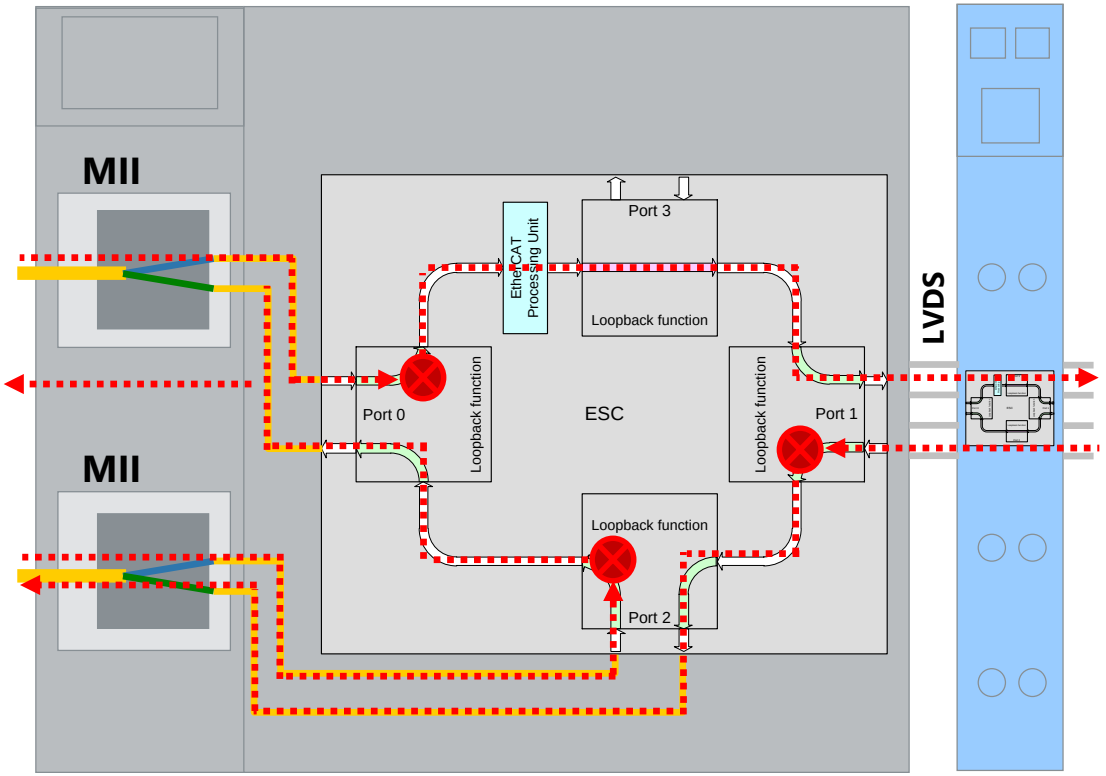


When Frame Errors Are Checked

BECKHOFF



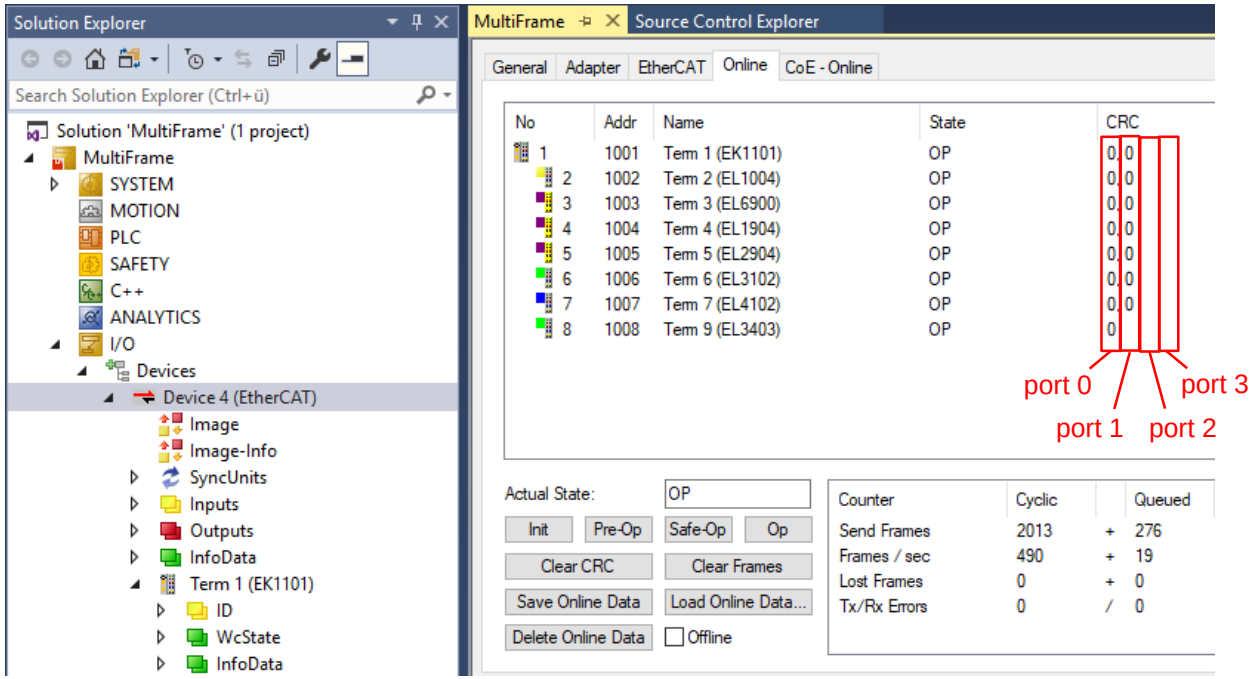
Concerning in particular frame errors, slaves perform the cyclic redundancy check when frames **are received** from outside (⊗): port 0 checks therefore invalid frames in the forward propagation direction, while ports 1, 2 and 3 perform the check in the backward direction.



Simplified Monitoring of Frame Errors

BECKHOFF

Frame errors (low bytes of RX Error Counter registers) can be monitored in the Online tab of the EtherCAT master by means of the following incremental counters:



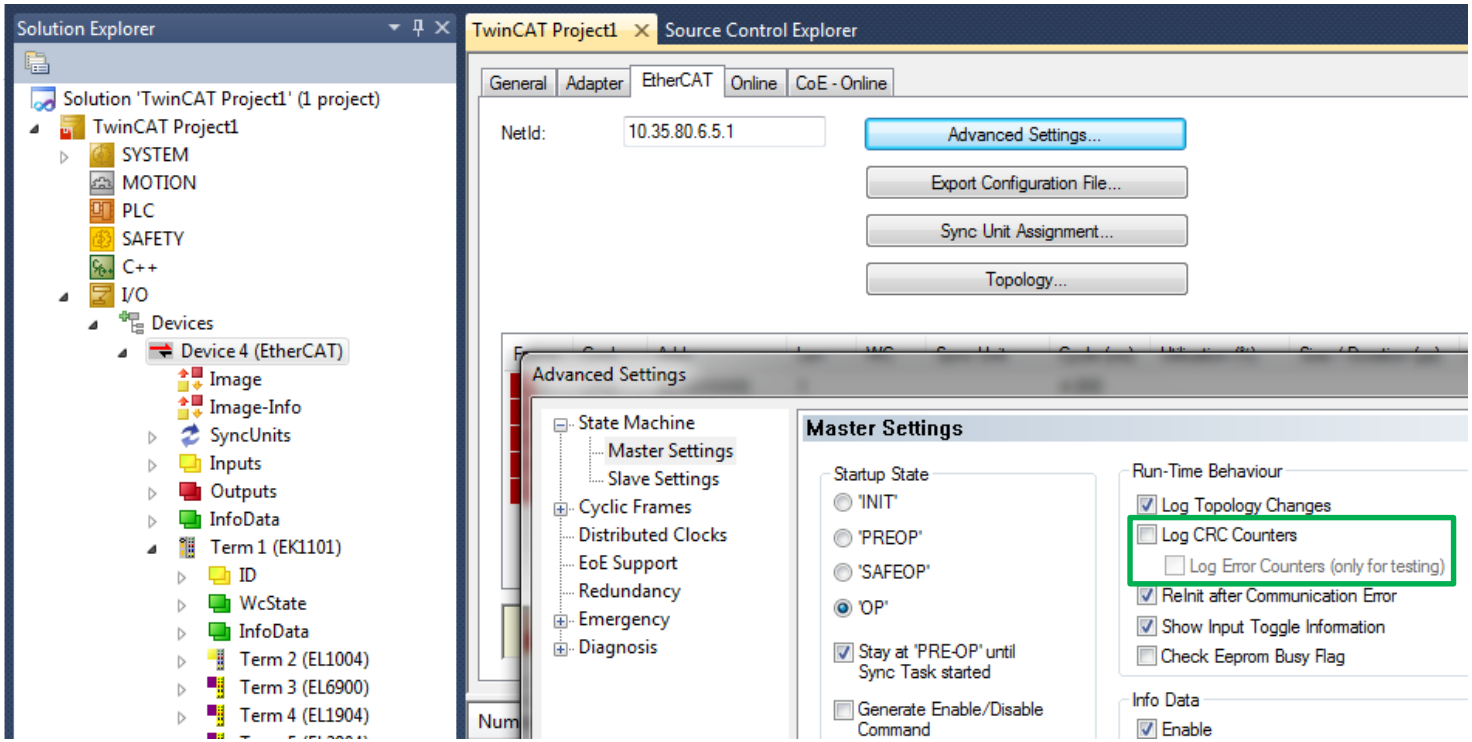
These counters keep trace of all increments in the Frame Error Counters in the slaves since the start of TwinCAT system, and can be reset by clicking on “Clear CRC”

How to Perform Detailed Trace of Hardware Error Counter?

BECKHOFF

In order to trace hardware error counters of individual slave devices directly, it is necessary to apply the following settings:

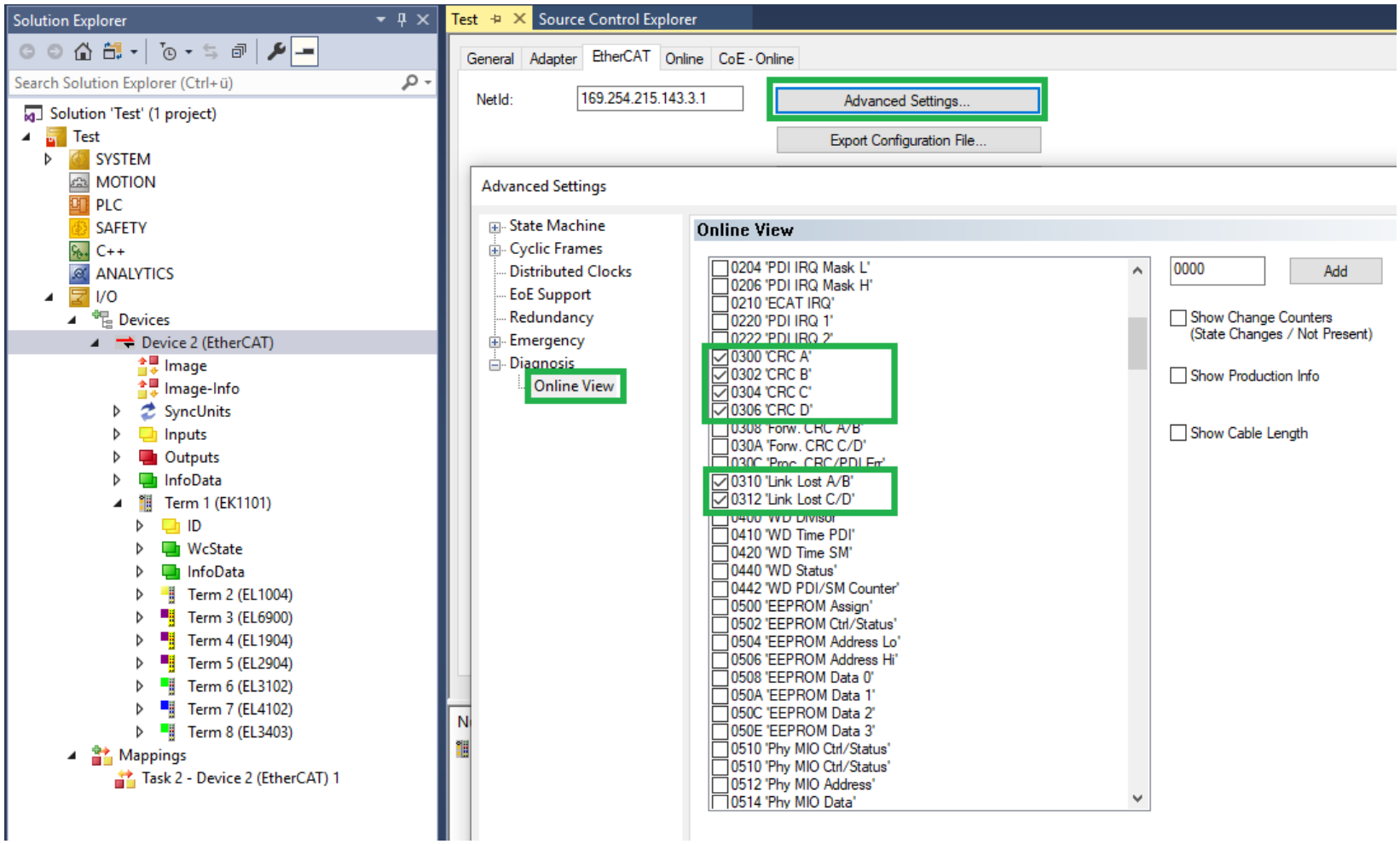
1. Deactivate “Log CRC Counters” flag in the master Advanced Settings:



How to Perform Detailed Trace of Hardware Error Counter?

BECKHOFF

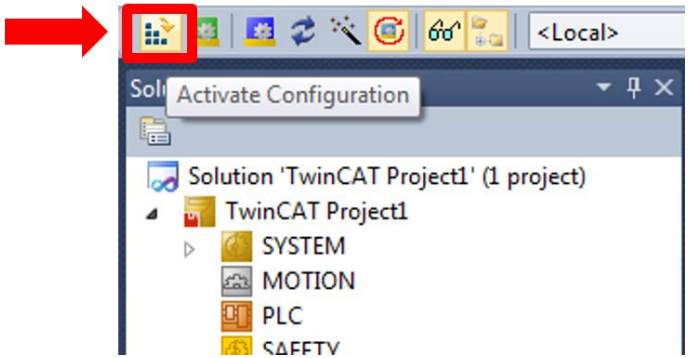
2. Add Registers 0x0300÷0x0306 and 0x0310÷0x0312 to the Online View:



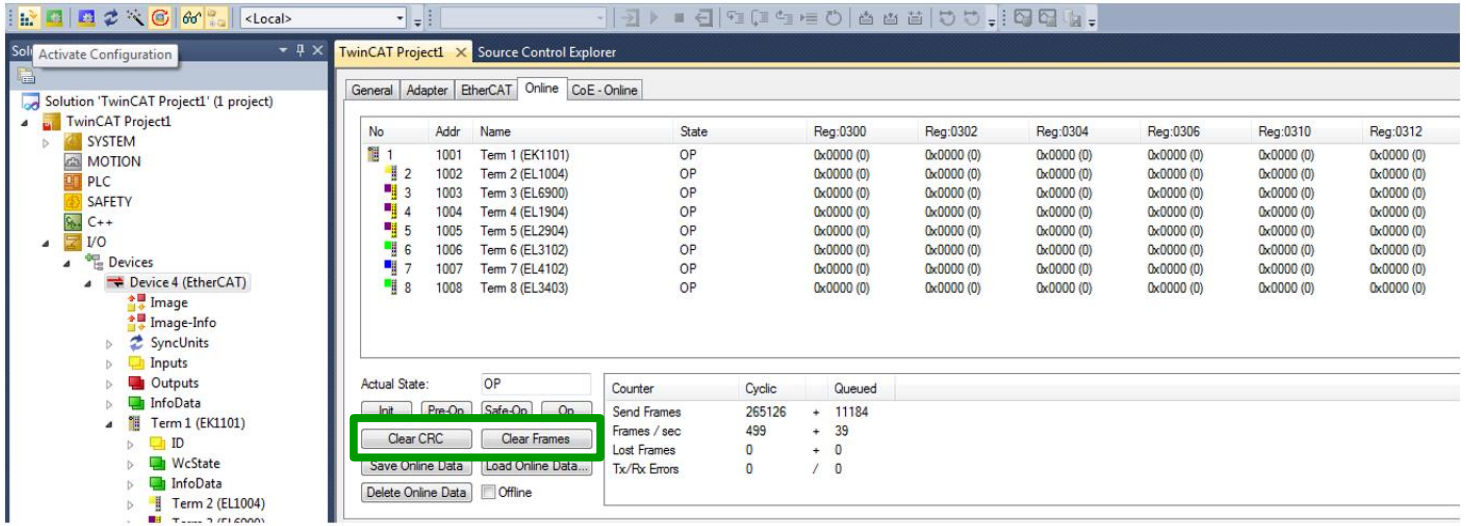
How to Perform Detailed Trace of Hardware Error Counter?

BECKHOFF

3. Activate configuration and restart TwinCAT:



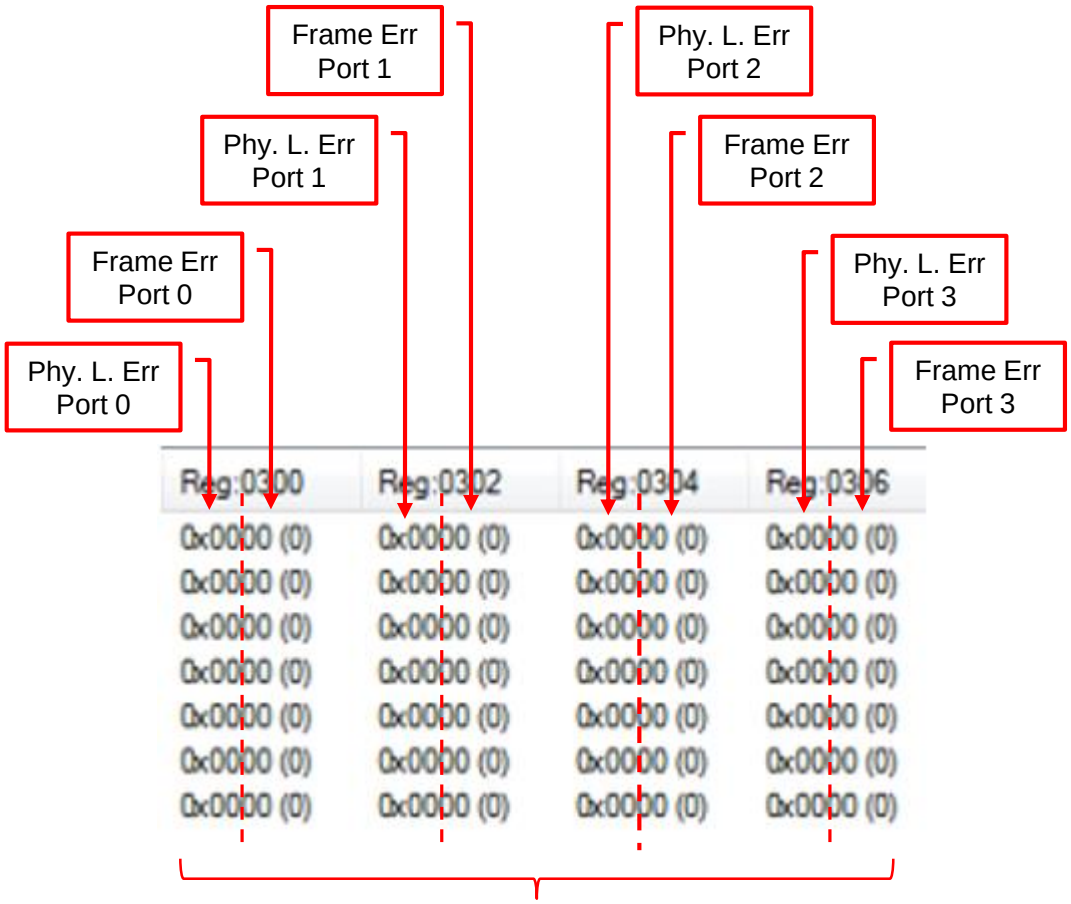
4. After TwinCAT restart, clear initial values of error counters:



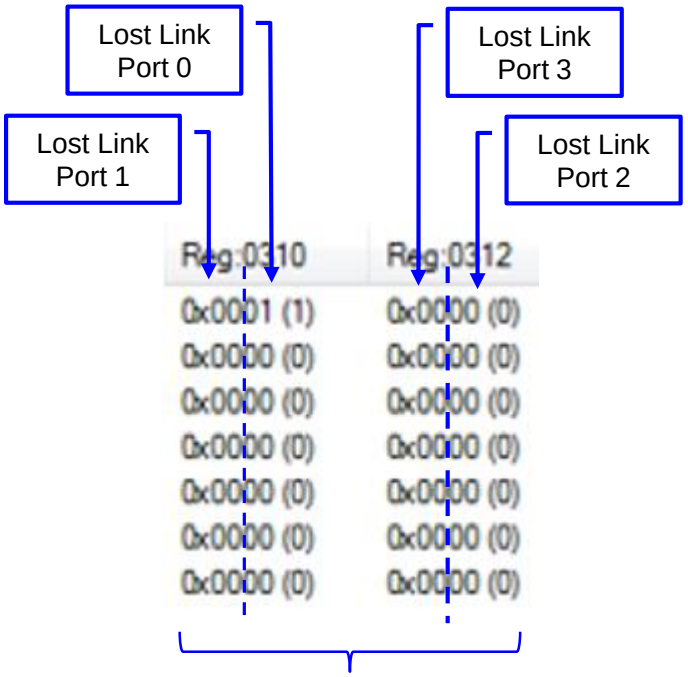
How to Perform Detailed Trace of Hardware Error Counter?

BECKHOFF

Counters are always displayed in TwinCAT in a **word-oriented** way:

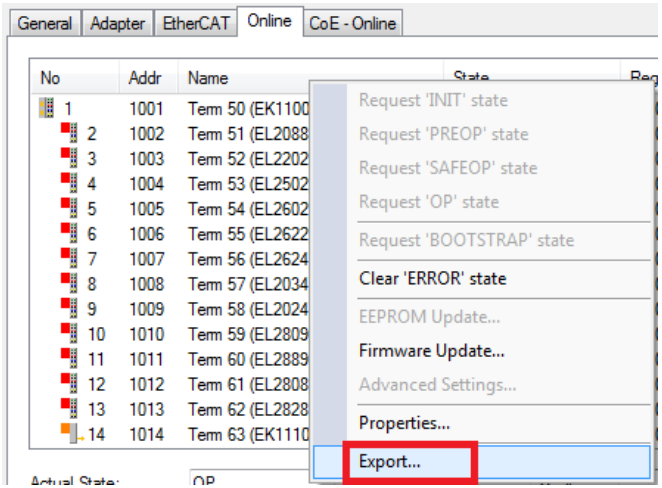


RX Error Counters

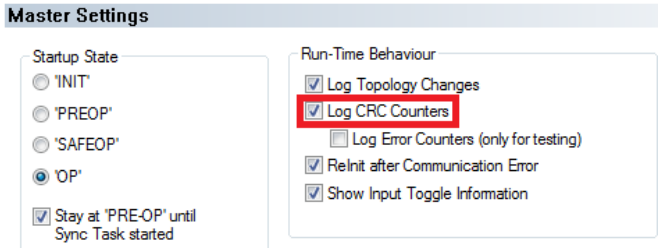


Lost Link Counters

5. Wait until errors are detected: the higher the number of errors captured, the better it is (only closing the TwinCAT project or restarting the TwinCAT system will reset counters: by changing tab or minimizing the project counter values will not be reset). Recorded counter values can optionally be exported for further analysis:



Once the counter acquisition has been completed, “Log CRC Counters” should be set again (TwinCAT restart necessary).

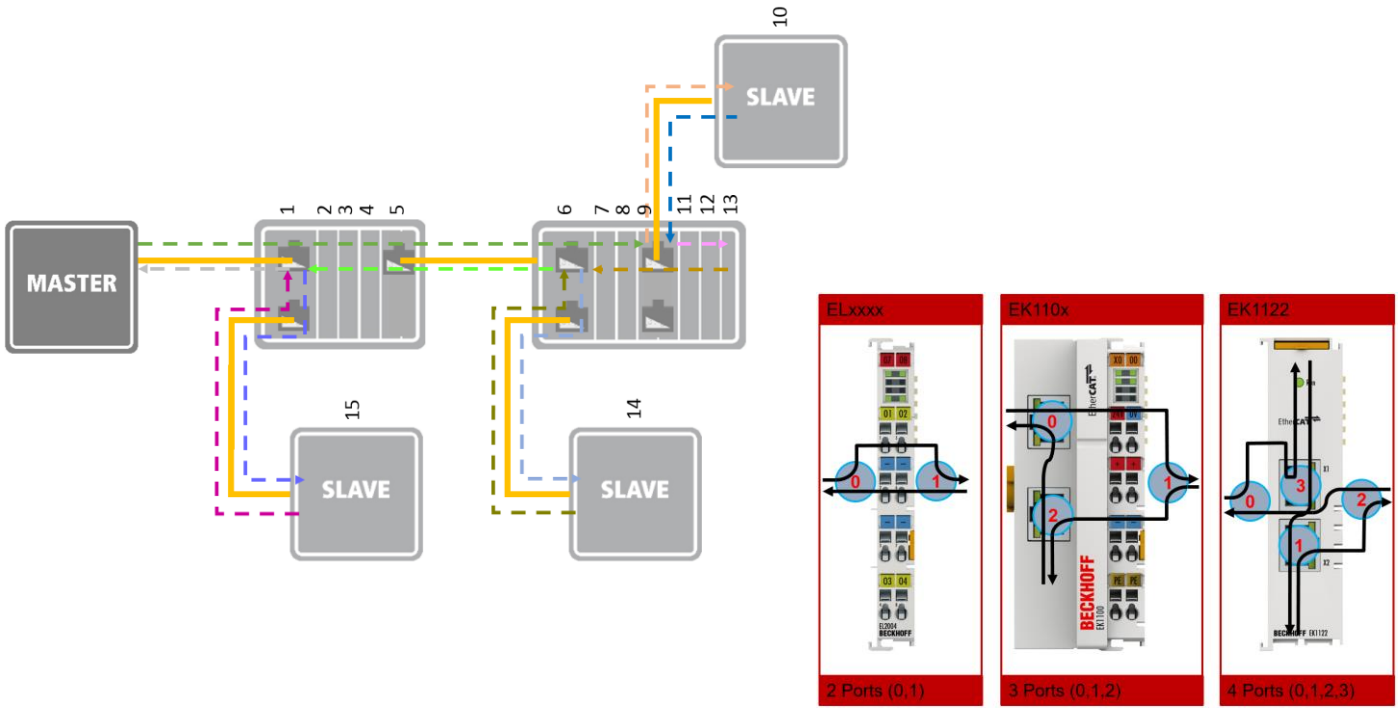


How to Find the Error Location?

BECKHOFF

In order to interpret the error counter map, it is suggested to:

1. Logically follow the frame propagation on the network and determine in which sequence slave ports perform the cyclic redundancy check (according to previous information):



CRC checked by
port 0, slave 1
port 0, slave 2
port 0, slave 3
port 0, slave 4
port 0, slave 5
port 0, slave 6
port 0, slave 7
port 0, slave 8
port 0, slave 9
port 0, slave 10
port 3, slave 9
port 0, slave 11
port 0, slave 12
port 0, slave 13
port 1, slave 12
port 1, slave 11
port 2, slave 9
port 1, slave 8
port 1, slave 7
port 1, slave 6
port 0, slave 14
port 2, slave 6
port 1, slave 5
port 1, slave 4
port 1, slave 3
port 1, slave 2
port 1, slave 1
port 0, slave 15
port 2, slave 1

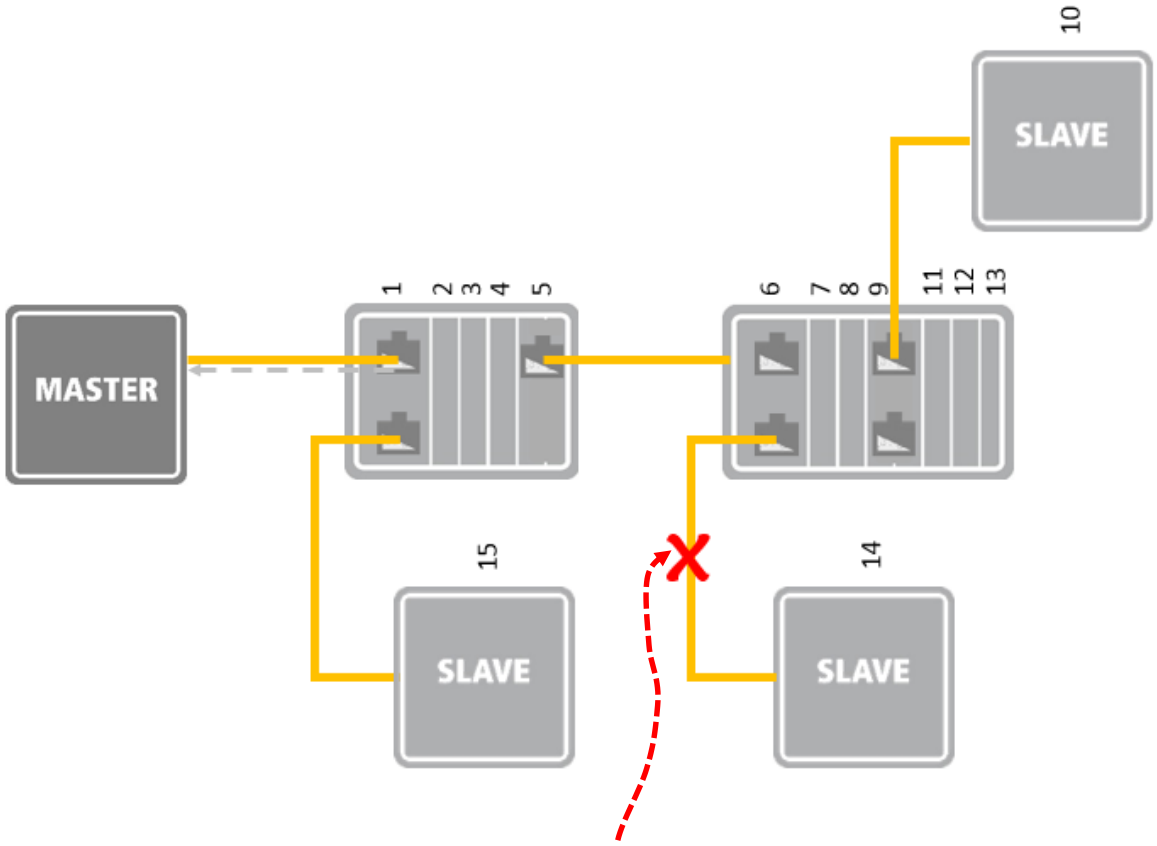
2. Detect the first port reporting an RX Error Counter > 0 according to the previously determined sequence:

CRC checked by
port 0, slave 1
port 0, slave 2
port 0, slave 3
port 0, slave 4
port 0, slave 5
port 0, slave 6
port 0, slave 7
port 0, slave 8
port 0, slave 9
port 0, slave 10
port 3, slave 9
port 0, slave 11
port 0, slave 12
port 0, slave 13
port 1, slave 12
port 1, slave 11
port 2, slave 9
port 1, slave 8
port 1, slave 7
port 1, slave 6
port 0, slave 14
port 2, slave 6
port 1, slave 5
port 1, slave 4
port 1, slave 3
port 1, slave 2
port 1, slave 1
port 0, slave 15
port 2, slave 1

	Reg: 0300	Reg: 0302	Reg: 0304	Reg: 0306
slave 1	√ 0x0000	0x0000	0x0000	0x0000
slave 2	√ 0x0000	0x0012	0x0000	0x0000
slave 3	√ 0x0000	0x0000	0x0000	0x0000
slave 4	√ 0x0000	0x0000	0x0000	0x0000
slave 5	√ 0x0000	0x0012	0x0000	0x0000
slave 6	√ 0x0000	√ 0x0000	0x0412	0x0000
slave 7	√ 0x0000	√ 0x0000	0x0000	0x0000
slave 8	√ 0x0000	√ 0x0000	0x0000	0x0000
slave 9	√ 0x0000	0x0000	√ 0x0000	√ 0x0000
slave 10	√ 0x0000	0x0000	0x0000	0x0000
slave 11	√ 0x0000	√ 0x0000	0x0000	0x0000
slave 12	√ 0x0000	√ 0x0000	0x0000	0x0000
slave 13	√ 0x0000	0x0000	0x0000	0x0000
slave 14	0x0A05	0x0000	0x0000	0x0000
slave 15	0x0012	0x0000	0x0000	0x0000

TwinCAT register Online view (compare with unidimentional list in previous slide)

3. Locate the point in the network at which the first RX errors are reported:



First port reporting RX Error Counter > 0 → likely problem location.

4. At the previously located point, the following should be done:
   - Check cable between detected and previous slave:
     - Is EtherCAT cable routed near to power cables or noise sources?
     - Have self-made cable connectors been badly implemented?
     - Is the cable not properly shielded?
   - Check detected and previous device:
     - Lacking power-supply (current provided too low, for example EBUS current)?
     - Potential difference between ground connection of the two devices?
   - Try to replace detected and previous device or to swap them, in order to see if errors depend on a specific device part, move with the devices or remain in same location.

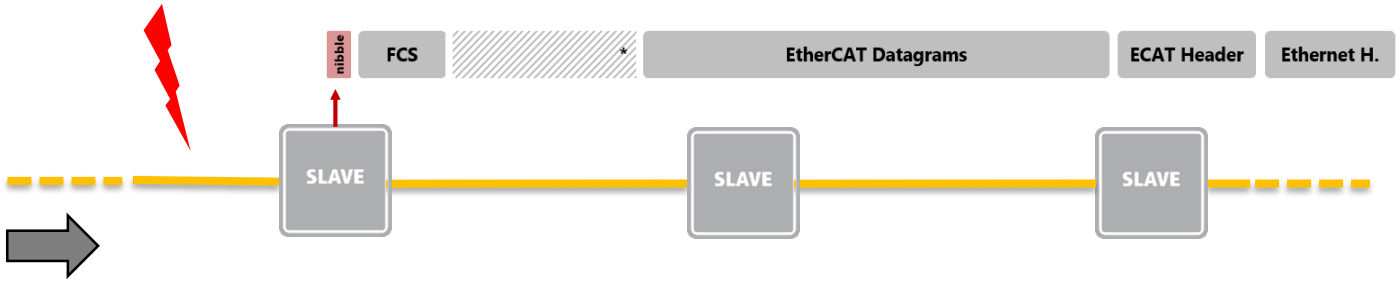
In case of EMC disturbances, both Physical Layer and Frame Errors are expected to be counted (even if their ratio can vary), as external disturbances are asynchronous with respect to the communication and therefore can strike both within and outside frames.

Completely unbalanced counter values (many Physical Layer with no Frame errors, or many Frame with no Physical Layer errors) could instead indicate an internal hardware problem in one device.

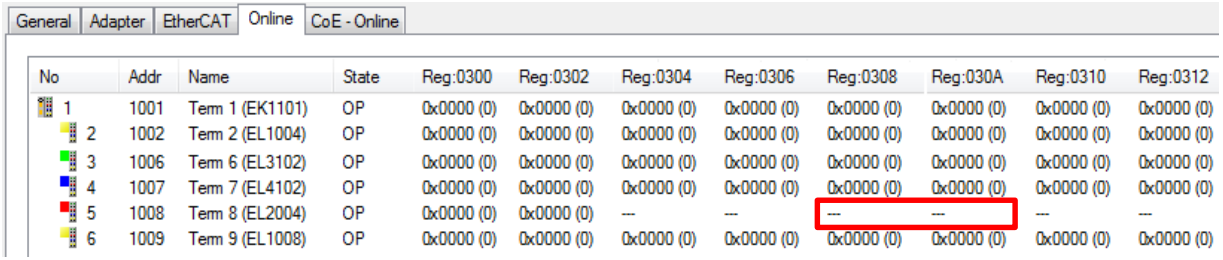
In Depth - CRC Error Forwarding Mechanism

BECKHOFF

Some slaves additionally support **Forwarded Frame Error Counters**: the Frame Error Counters are incremented only if the Frame Error is detected for the first time, otherwise the Forwarded Frame Error Counters are incremented .



Forwarded Frame Error Counters are not supported by all slaves (the feature is optional) and shall be considered as a complementary information: they are helpful but not crucial in order to locate the first error appearance.



No	Addr	Name	State	Reg:0300	Reg:0302	Reg:0304	Reg:0306	Reg:0308	Reg:030A	Reg:0310	Reg:0312
1	1001	Term 1 (EK1101)	OP	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)
2	1002	Term 2 (EL1004)	OP	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)
3	1006	Term 6 (EL3102)	OP	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)
4	1007	Term 7 (EL4102)	OP	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)
5	1008	Term 8 (EL2004)	OP	0x0000 (0)	0x0000 (0)	---	---	---	---	---	---
6	1009	Term 9 (EL1008)	OP	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)	0x0000 (0)

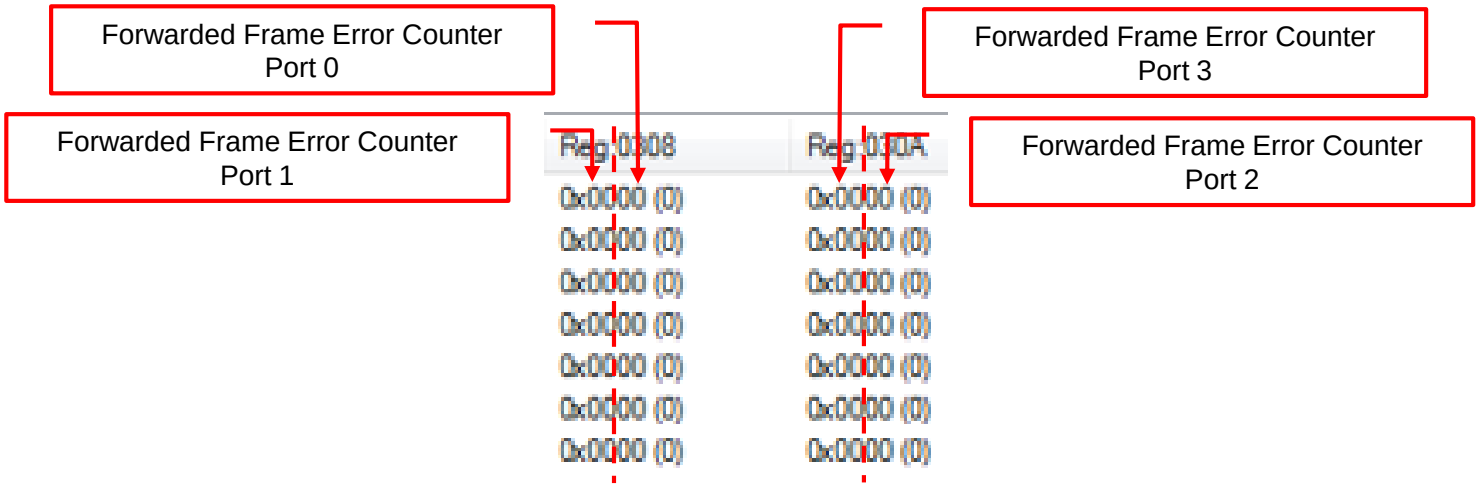
When Forwarded Frame Error Counters are not supported, this can be seen in TwinCAT.

In Depth – Forwarded Error Counters

BECKHOFF

In the master Online tab, registers are always displayed in a word-oriented way. Therefore, the interpretation of the Forwarded Frame Error Counters is the following:

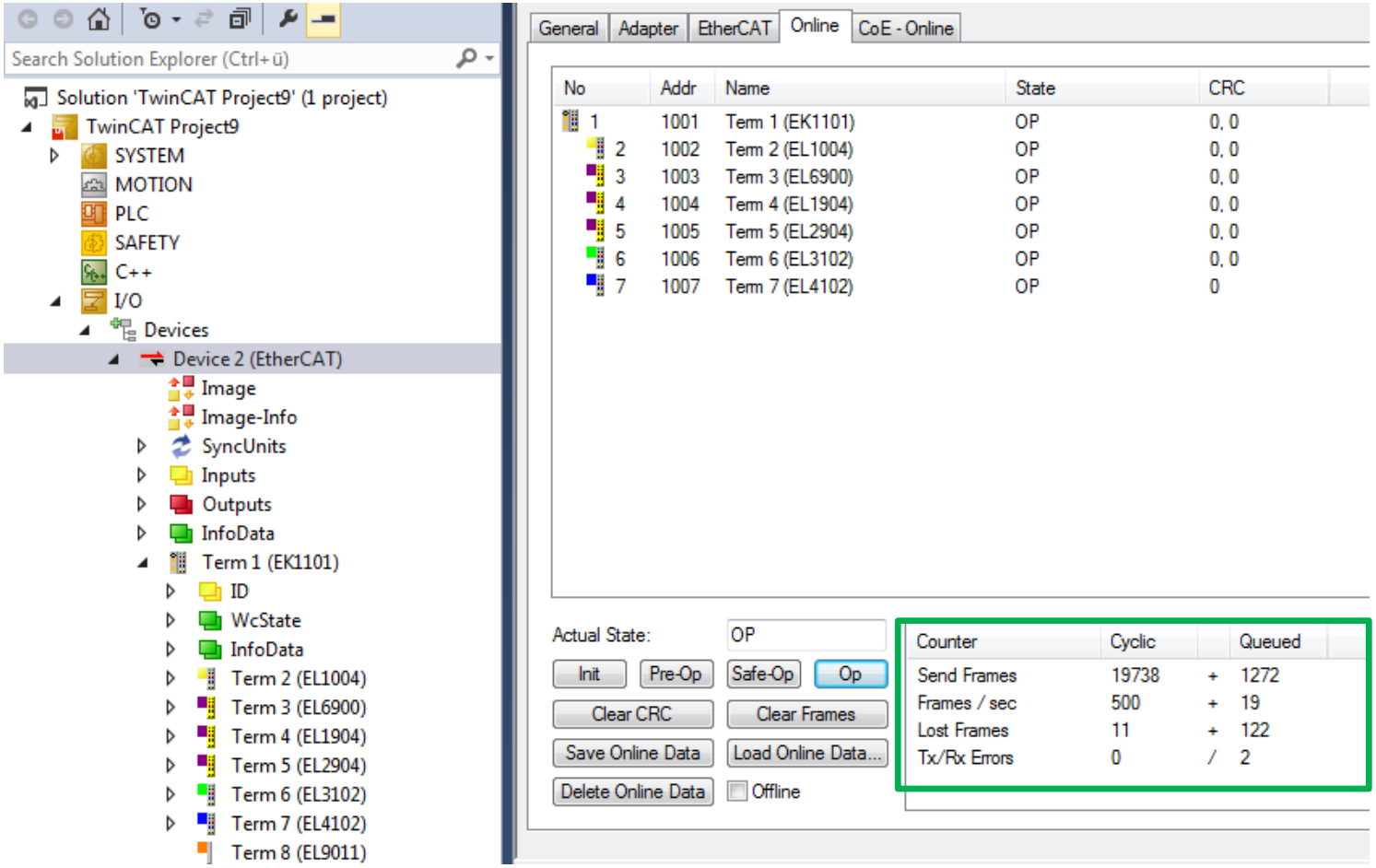
Register	Length	Meaning
0x0308	1 word	Forwarded Frame Error Counter port 0 + port 1
0x030A	1 word	Forwarded Frame Error Counter port 2 + port 3



Master Error Counters

BECKHOFF

Also the EtherCAT **master** provides hardware error **counters**:



Master Error Counters

BECKHOFF

The EtherCAT master device supports two different error counters:

- **Rx Errors**: it is similar to Physical Layer Error Counters on slave side, and counts symbol errors reported by the network card (can occur both within and without frames).
- **Lost Frames**: TwinCAT considers as „lost“ both frames not coming back to the master at all (because the EtherCAT loop is somewhere interrupted) or frames coming back with a wrong CRC. A Lost Frame will always be reported by TwinCAT as Working Counter error for all the datagrams contained in the lost frame!

Concerning the frame type:

- **Cyclic**: cyclic frames sent for Process Data exchange:

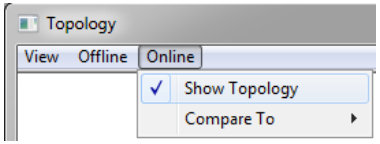
Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)	Utilization (%)	Size / Duration (µs)	Map Id	Flags
0	LRD	0x09000000	1			4.000				
0	LRW	0x01000000	16	9	<default>	4.000				
0	LWR	0x01000800	4	1	<default>	4.000				
0	LRD	0x01001000	11	3	<default>	4.000				
0	BRD	0x0000 0x0130	2	7		4.000	0.27 0.27	110 / 10.72	0	

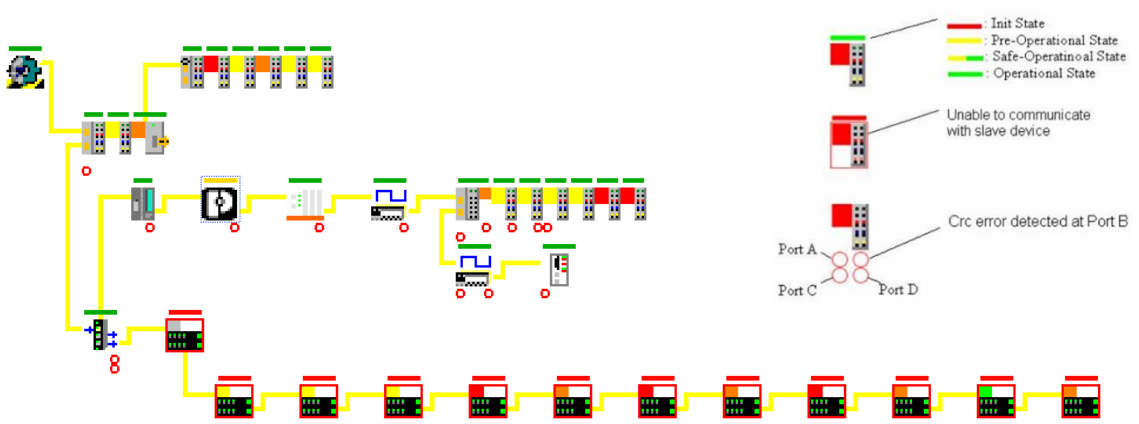
- **Queued**: all acyclic frames (Mailbox, State Machine, Register access…)

Online Topology View

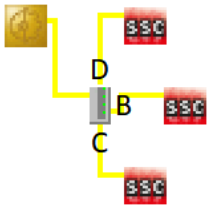
BECKHOFF

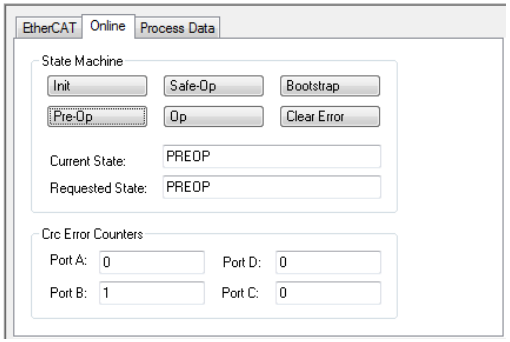
Most of the hardware diagnostic information can be monitored in the **Topology View**:





From the position (above, right, below) with respect to previous slave, it is possible to easily detect to which port a device is connected to:





The Topology View is an ActiveX component which can be imported into any Windows application independently on the TwinCAT development environment.

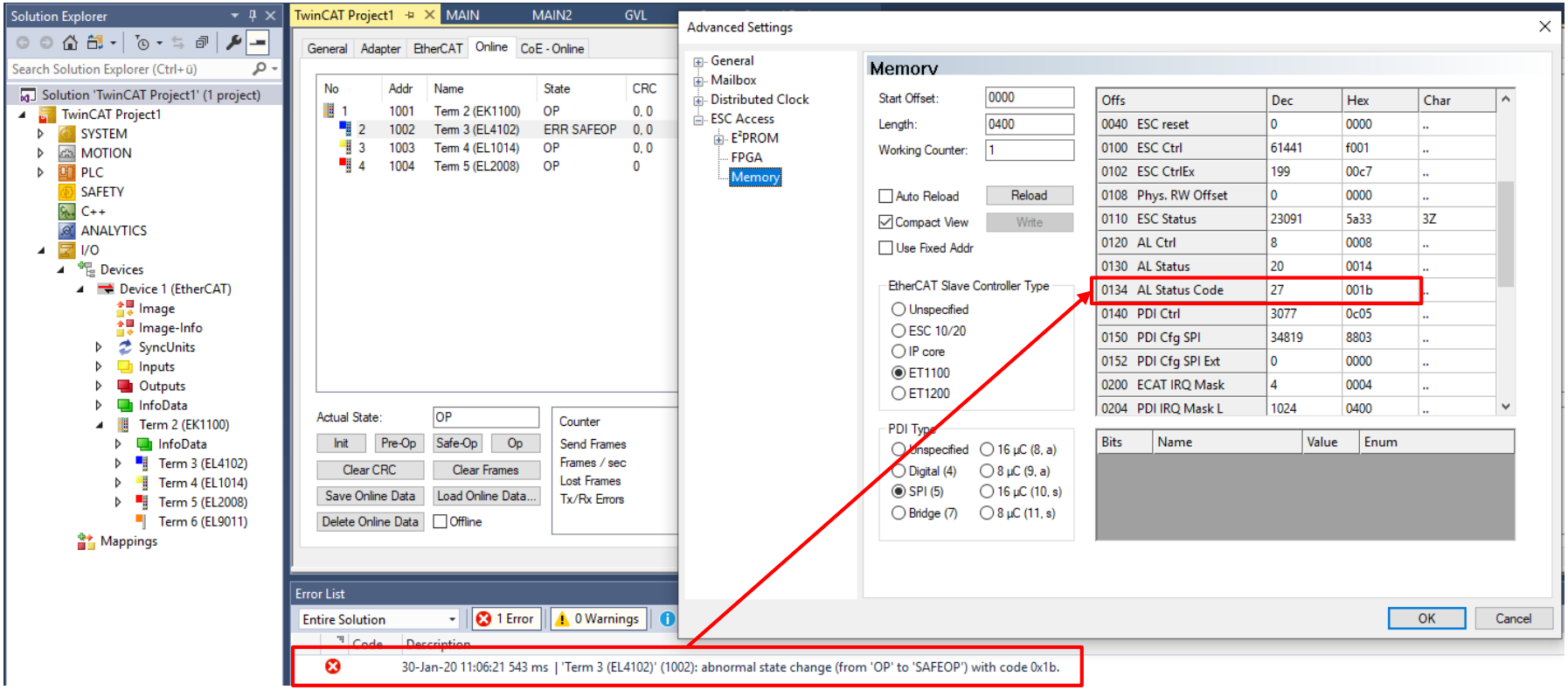
Software Diagnostics

Errors at Software Level – the EtherCAT State Machine

BECKHOFF

The diagnostic information at software level is typically related to the EtherCAT State Machine operation: whenever the slave cannot be in the state expected by the master, it will return an error code in **AL Status Code** Register 0x0134.

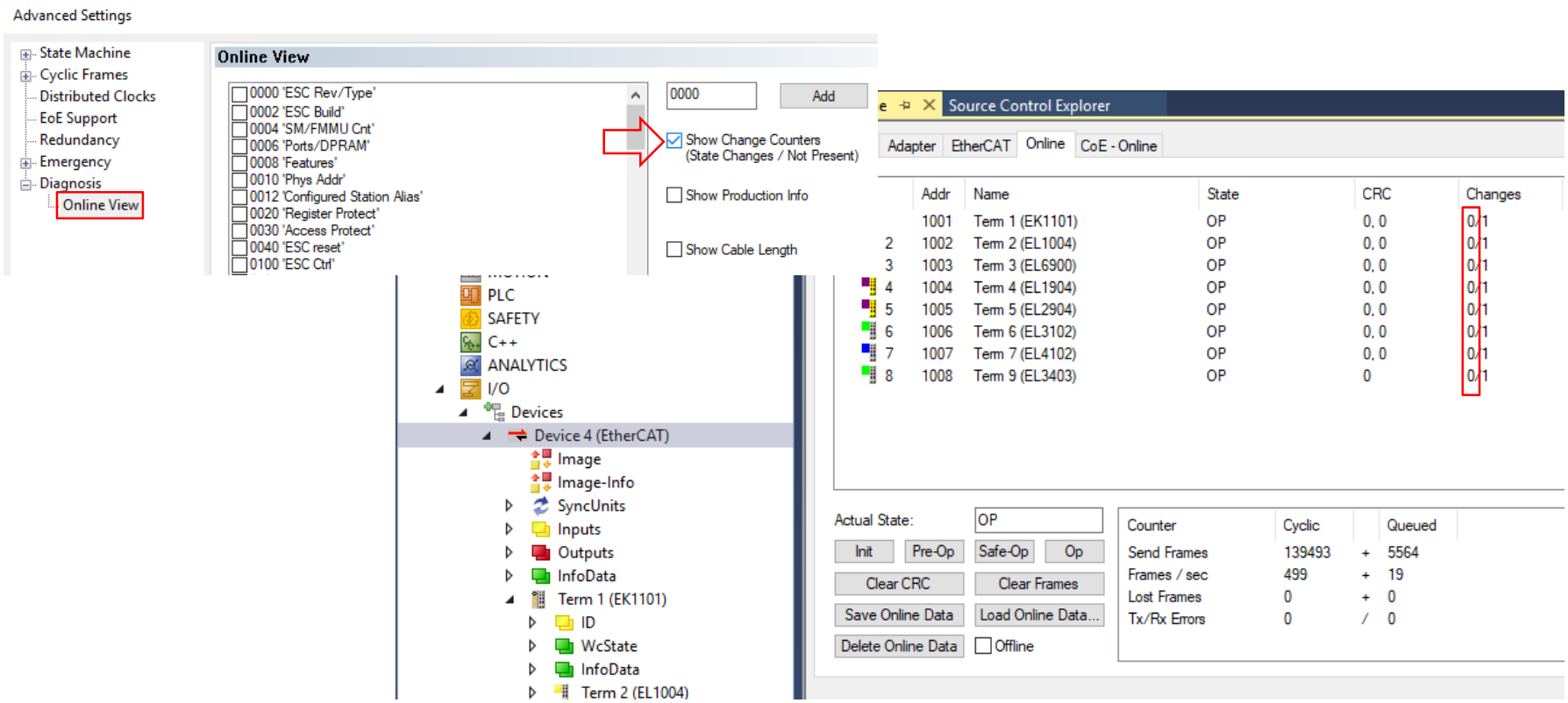
State Machine errors are also reported in the TwinCAT Logger:



Simplified Monitoring of Unexpected State Transitions

BECKHOFF

State transitions can be monitored in the Online tab of the EtherCAT master by means of the following incremental counter after enabling them in the Advanced Settings of the master:



These counters report the number of unexpected state transitions (1o is start value, > 0 means that the slave suffered at least an unexpected state transition).

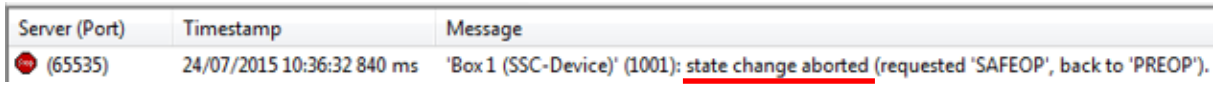
EtherCAT State Machine Error Types

BECKHOFF

EtherCAT State Machine errors can be classified into **2 types**:

- **Initialization errors** (slave does not reach Operational state during start-up): during state transitions, the master sends Init Commands to the slave based on the ESI file content. When the slave detects one or more start-up parameters as invalid, it will refuse the corresponding state transition.

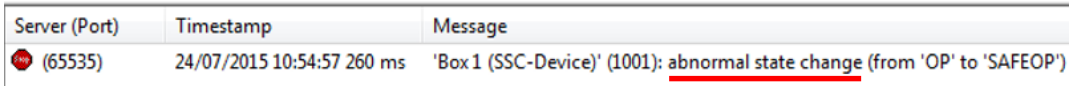
Typical initialization errors:



  - 0x0003 : Invalid Device Setup (the configured sequence of KL modules of BK1xxx is not correct)
  - 0x001D : Invalid Output Configuration (configuration of output process data is invalid)
  - 0x001E : Invalid Input Configuration (configuration of input process data is invalid)
  - 0x0035 : Invalid Sync Cycle Time (the cycle time configured in DC mode is not supported)

- **Runtime errors** (slave steps back from Operational to a lower state): a slave which was properly configured and properly reached Operational state can thereafter detect an error during operation and spontaneously perform a state transition to a lower state.

Typical runtime errors:



  - 0x001A : Synchronization error (network jitter caused the slave to lose synchronization)
  - 0x001B : Sync manager watchdog (slave didn´t receive cyclic frames for > watchdog time)
  - 0x002C : Fatal SYNC error (no SYNC hardware interrupt is received by ESC anymore)

Initialization Errors – What to do?

BECKHOFF

The ESI file of a slave shall contain all the information which is needed by TwinCAT in order to properly configure that slave within the network. If a configuration is activated with the default slave settings as read from the ESI file (no manual modification), the slave should go to Operational without errors.

If an initialization error occurs:

1. Make sure that the ESI file copied in the standard TwinCAT folder contains the description of the hardware device connected to the network (Product Code and Revision Number in the ESI file correspond to the information reported in CoE Object 0x1018).

2. Check if the default settings of the slave were changed, and in case delete and manually import it in the TwinCAT configuration again (the default settings will be restored).

3. (For modular slaves) Check if the modules configured in the tab “Slots” exactly correspond to the hardware modules physically connected to the slave.

4. (For DC-Synchronous devices) Check if the jitter of the master device or the SYNC Shift Time settings could prevent a proper DC synchronization of the slave.

In case these checks do not lead to a solution, contact the slave manufacturer.

Runtime Errors – What to do?

BECKHOFF

Once a slave entered the Operational state successfully, there should be no reason why it performs a spontaneous transition to a lower state in runtime.

If a runtime error occurs:

1. In case of watchdog error from the slave, check if the TwinCAT software Task (PLC Program, NC Task, …) is running properly (in TwinCAT, the sending and receiving of cyclic frames is always triggered by the above laying software tasks).

2. Check if the jitter performances of the master device could justify a loss of synchronization (synchronization errors could easily occur if maximum jitter > 20÷30% of the communication cycle time).

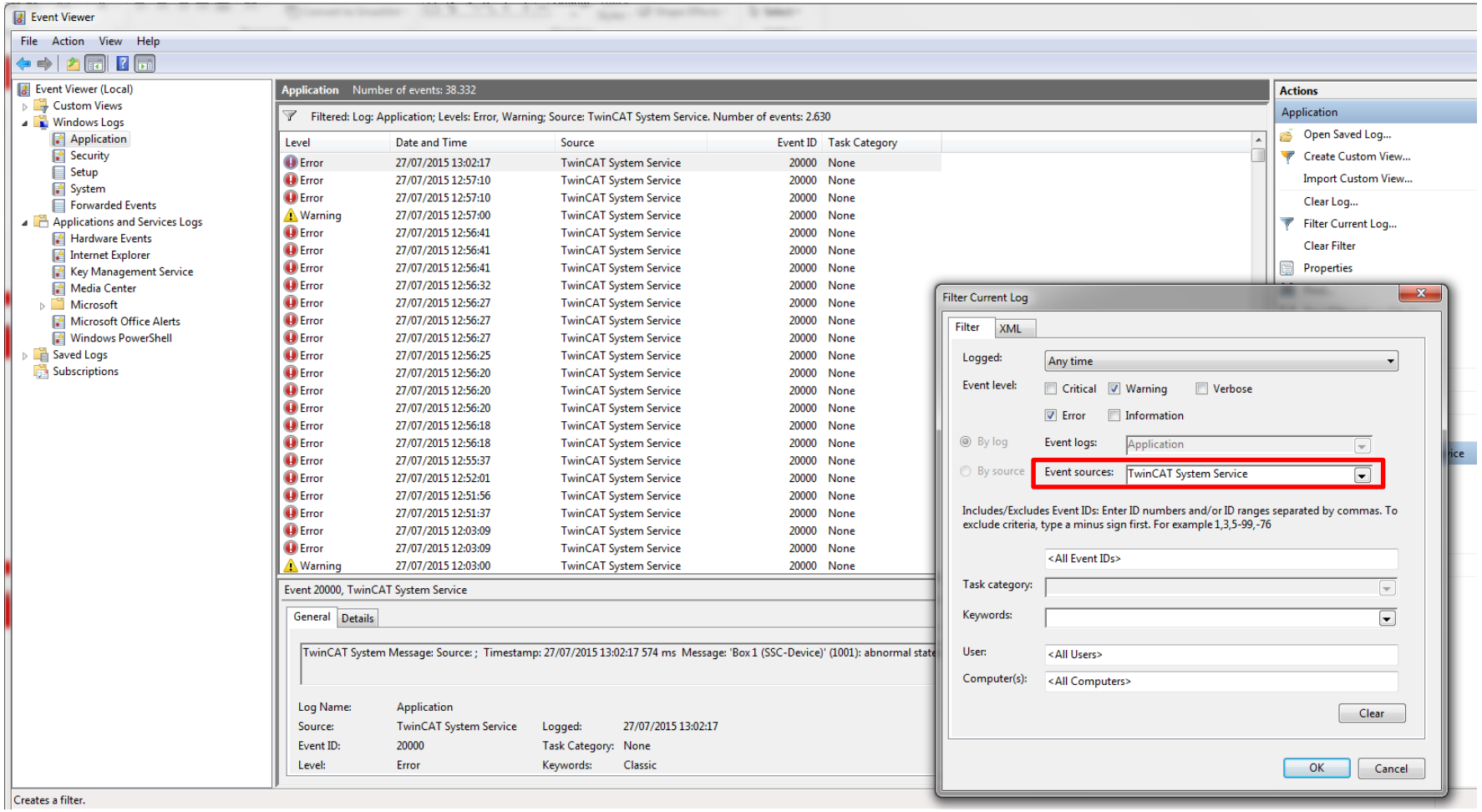
3. Check if hardware errors - like loss of physical link - occur, which could indirectly cause a watchdog reaction or a loss of synchronization (see hardware diagnostics for problems at physical layer).

In case these checks do not lead to a solution, a Wireshark trace can be helpful.

Saving and Exporting Software Errors

BECKHOFF

State Machine errors are saved in the **Windows Log** and can therefore be exported for a further analysis even if the TwinCAT Logger was closed or is not available:

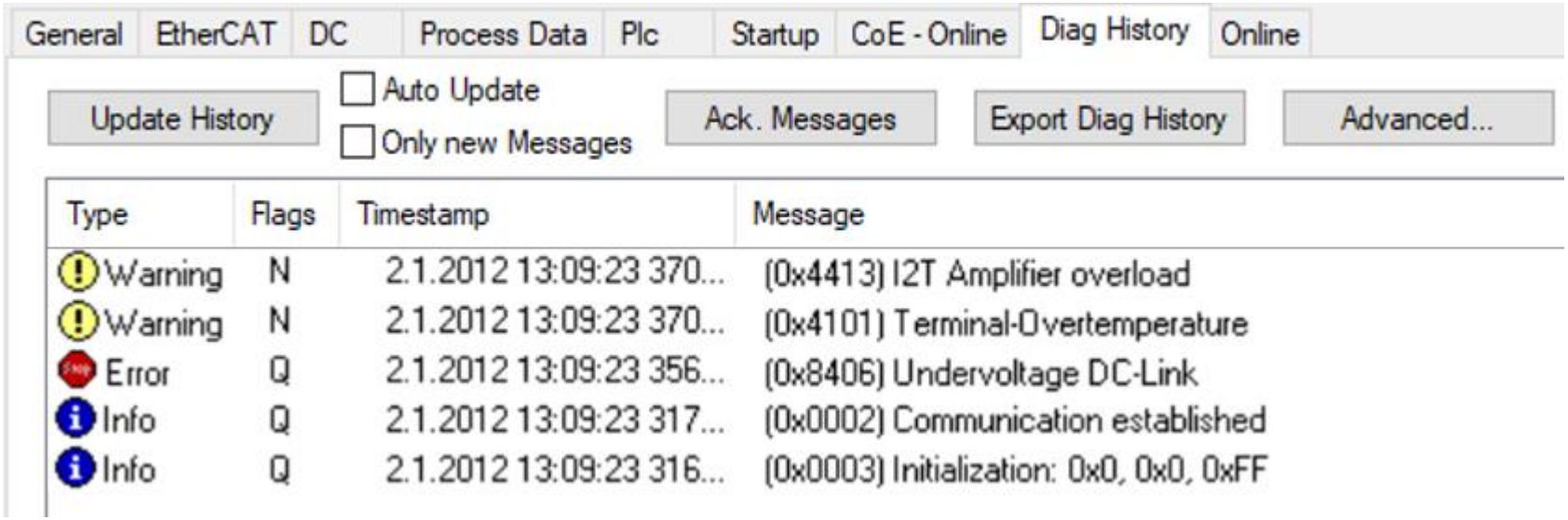


Diagnosis History Object

BECKHOFF

For all the application-specific errors, CoE slave devices can optionally support the Diagnosis History Object **0x10F3**.

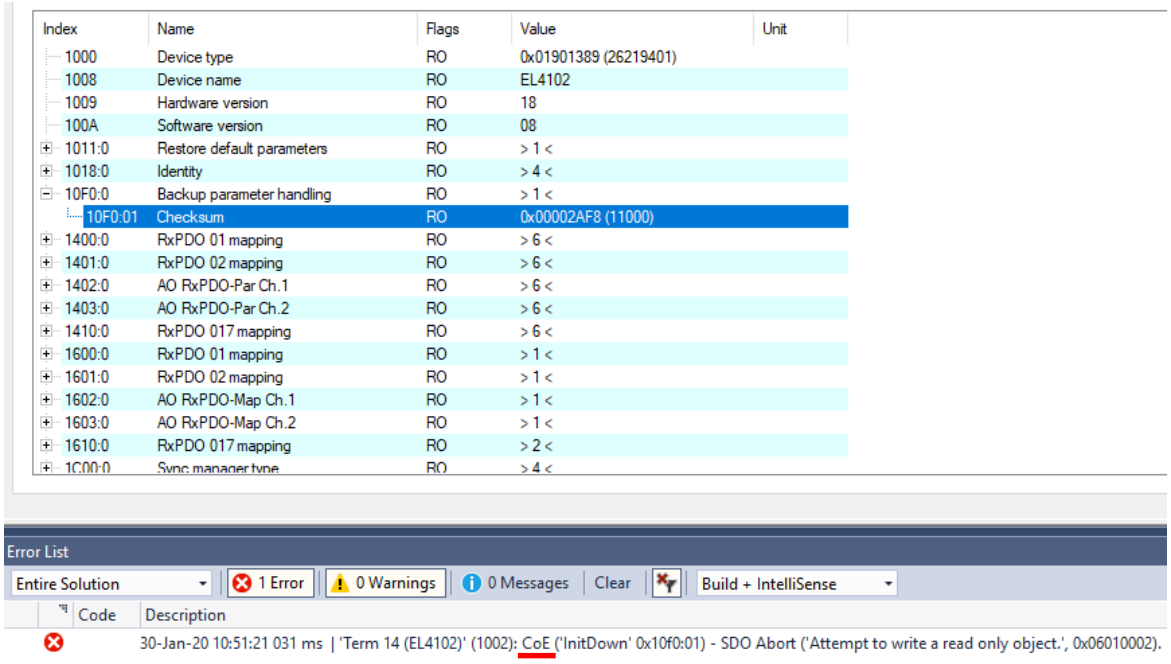
In case this Object is supported by the slave, TwinCAT shows an additional “Diag History” tab:



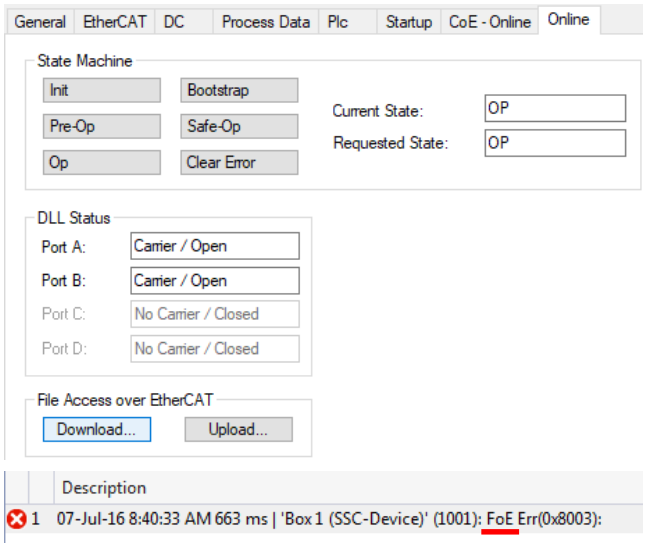
Mailbox Protocol Errors

BECKHOFF

Mailbox protocol errors are a specific class of generic (i.e. not manufacturer-specific) software errors, which do not affect the EtherCAT State Machine (i.e. they do not prevent or determine unexpected state transitions): they are issued whenever some action forbidden by a particular Mailbox protocol is performed.



Index	Name	Flags	Value	Unit
1000	Device type	RO	0x01901389 (26219401)	
1008	Device name	RO	EL4102	
1009	Hardware version	RO	18	
100A	Software version	RO	08	
1011:0	Restore default parameters	RO	> 1 <	
1018:0	Identity	RO	> 4 <	
10F0:0	Backup parameter handling	RO	> 1 <	
10F0:01	Checksum	RO	0x00002AF8 (11000)	
1400:0	RxPDO 01 mapping	RO	> 6 <	
1401:0	RxPDO 02 mapping	RO	> 6 <	
1402:0	AO RxPDO-Par Ch.1	RO	> 6 <	
1403:0	AO RxPDO-Par Ch.2	RO	> 6 <	
1410:0	RxPDO 017 mapping	RO	> 6 <	
1600:0	RxPDO 01 mapping	RO	> 1 <	
1601:0	RxPDO 02 mapping	RO	> 1 <	
1602:0	AO RxPDO-Map Ch.1	RO	> 1 <	
1603:0	AO RxPDO-Map Ch.2	RO	> 1 <	
1610:0	RxPDO 017 mapping	RO	> 2 <	
1C00:0	Sync manager type	RO	> 4 <	



Mailbox Protocol Errors – What to do?

BECKHOFF

According to the specific Mailbox protocol, possible error reasons are:

- **CoE**
  - The master tries to read/write an Object which does not exist in Object Dictionary.
  - The master tries to write a read-only Object.
  - The master tries to access an Object with Complete Access but this feature is not supported by the slave.

- **FoE**
  - The file name is not correct (for example, the extension *.xxx is missing).
  - The slave expects a password, which is not set correctly
  - The file size is too large compared to what the slave expects
  - The slave is not in Bootstrap state

- **EoE**
  - The master tries to configure to the slave an IP-Address which ends with “x.y.z.0”, and the Tcp/Ip stack of the EoE slave refuses this setting

BECKHOFF

EtherCAT Diagnostics in TwinCAT PLC Program

Systematic EtherCAT Diagnostics in the PLC

BECKHOFF

Regarding the EtherCAT network, TwinCAT **automatically** provides a lot of **diagnostic information**, which can be used in the PLC program in order to detect error conditions in the fieldbus communication, to automatically react and to report them to the user.

Users are always encouraged to implement a minimal EtherCAT diagnosis handling in their PLC programs, as this helps saving a lot of time during operation in case communication errors appear.

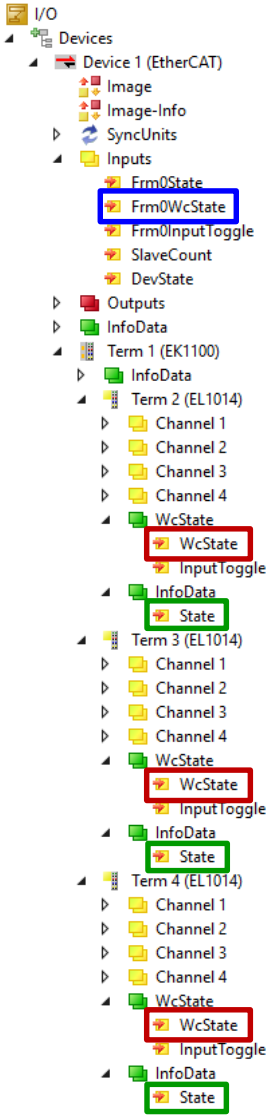
The TwinCAT PLC programs can make combined use of 2 different diagnostic information:

- **Cyclic information**: input data which are included by default in the cyclic process image of the EtherCAT network, and which can directly be linked to corresponding PLC input variables (AT %I*).

- **Acyclic information**: information which can be retrieved by the PLC calling dedicated function-blocks provided by the default libraries (TcEtherCAT.lib).

Cyclic Diagnostic Information in TwinCAT

BECKHOFF



EtherCAT-related diagnostic information provided cyclically by TwinCAT:

- FrmXWcState (WORD, 1 variable per frame)
- WcState (BOOL, 1 variable per slave)
- State (WORD, 1 variable per slave)

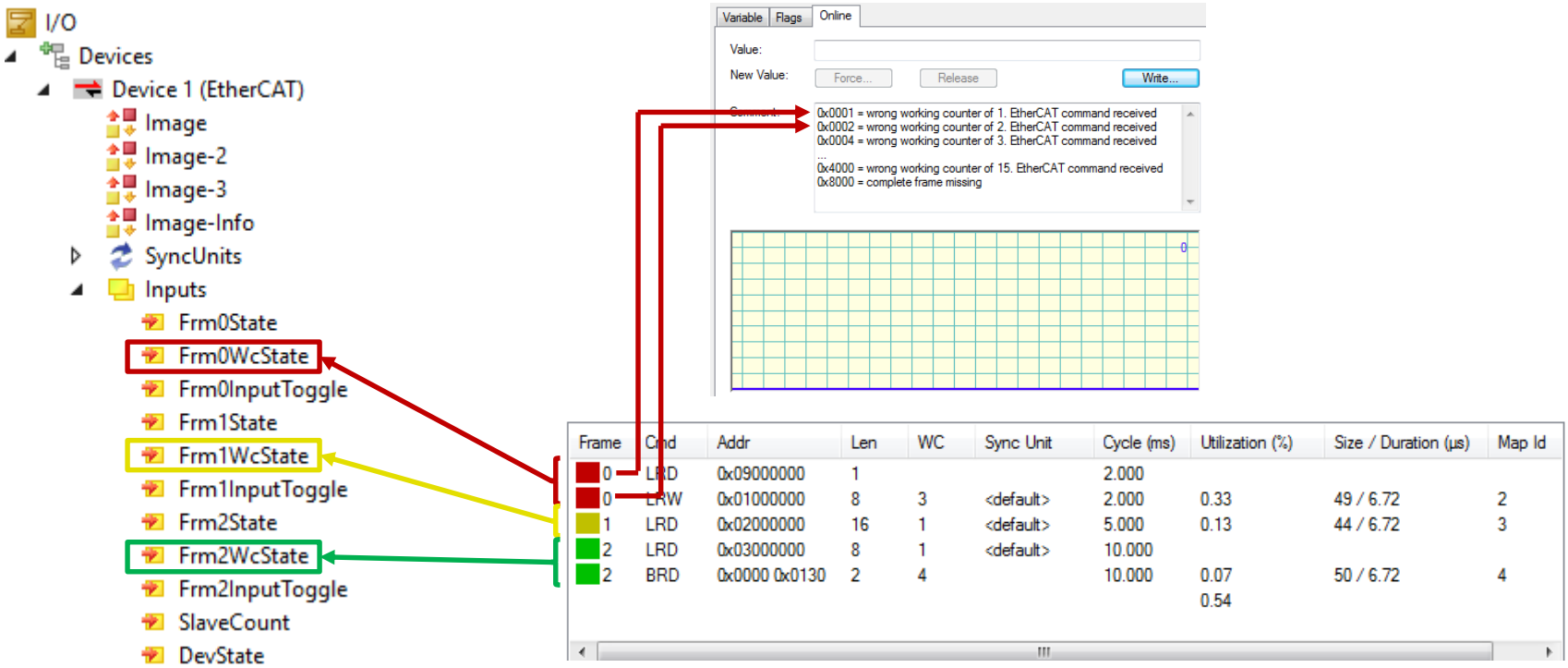
At least the WcState and State variables of all slaves should be linked to the PLC program in order to enable minimal diagnostics in the control application.

Cyclic Information - FrmXWcState variable

BECKHOFF

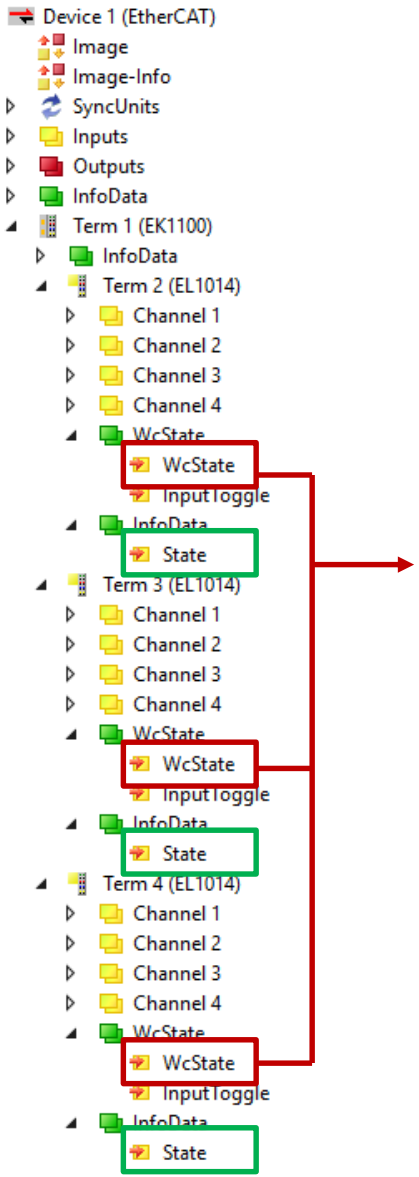
Enables the PLC to check the Working Counter information for each Datagram:

- Each configured Frame has its own 16-bit **FrmXWcState** variable
- Each **bit** in the variable corresponds to a specific Datagram within the Frame
- The bit is set if the corresponding Datagram has a wrong Working Counter



Cyclic Information - WcState variable

BECKHOFF



- Boolean variable in the process image of each slave.
- If a datagram has a wrong Working Counter value, WcState variable is set for **all** the slaves addressed by that datagram.
- A slave with WcState = 1, therefore, is not necessarily the cause of the problem. Additional information will be provided by the 16-bit **State** variable of slaves with WcState = 1.

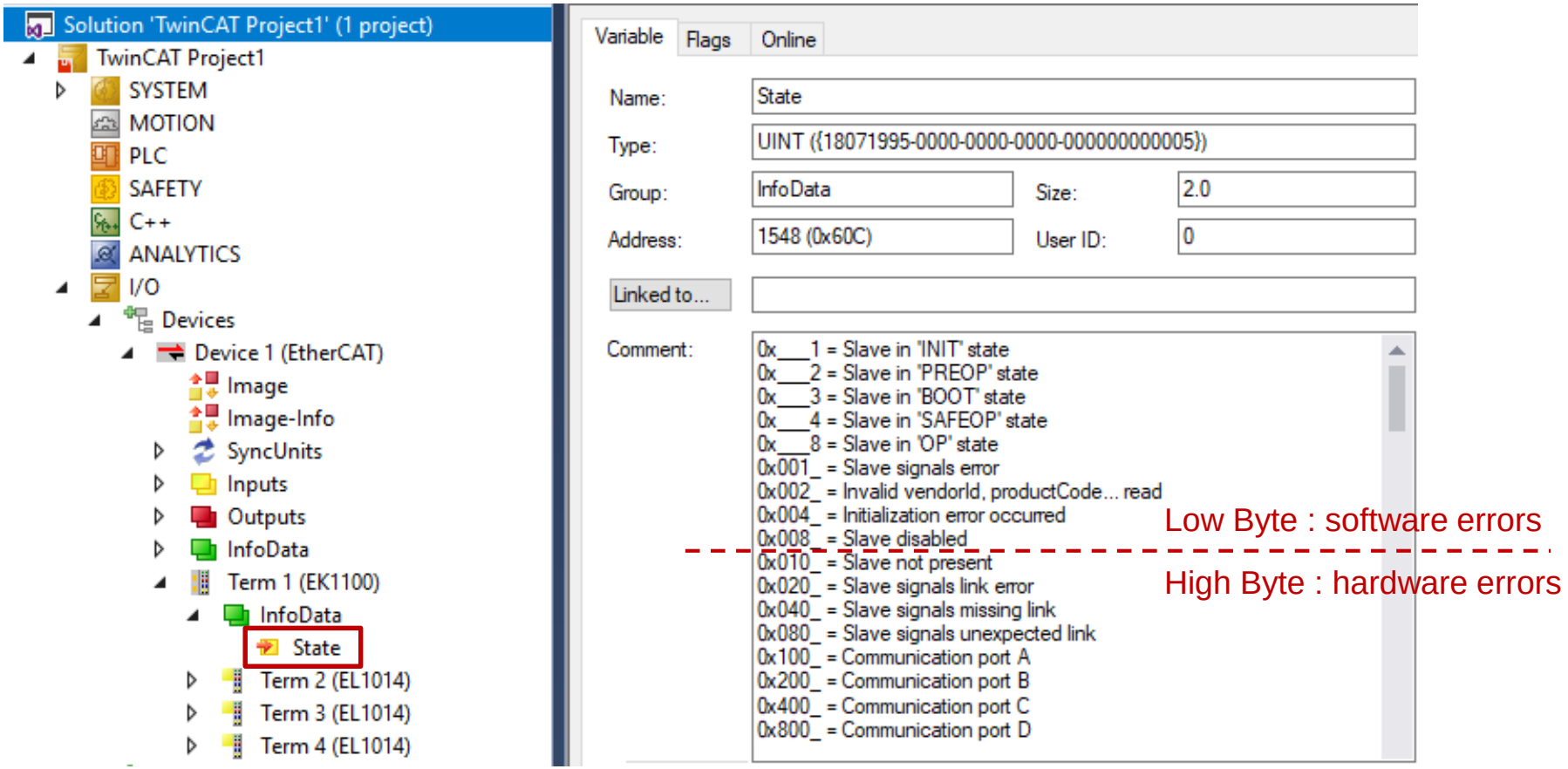
Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)	Utilization (%)	Size / Duration (µs)	Map Id
0	LRD	0x01000000	2	3	<default>	2.000			
0	BRD	0x0000 0x0130	2	4		2.000	0.33 0.34	44 / 6.72	0

(Example: in case of wrong Working Counter for the LRD datagram, all Slaves addressed by this datagram will set WcState = 1. Reading the State variable of these slaves the master application can investigate which slave is responsible for the error).

Cyclic Information - State variable

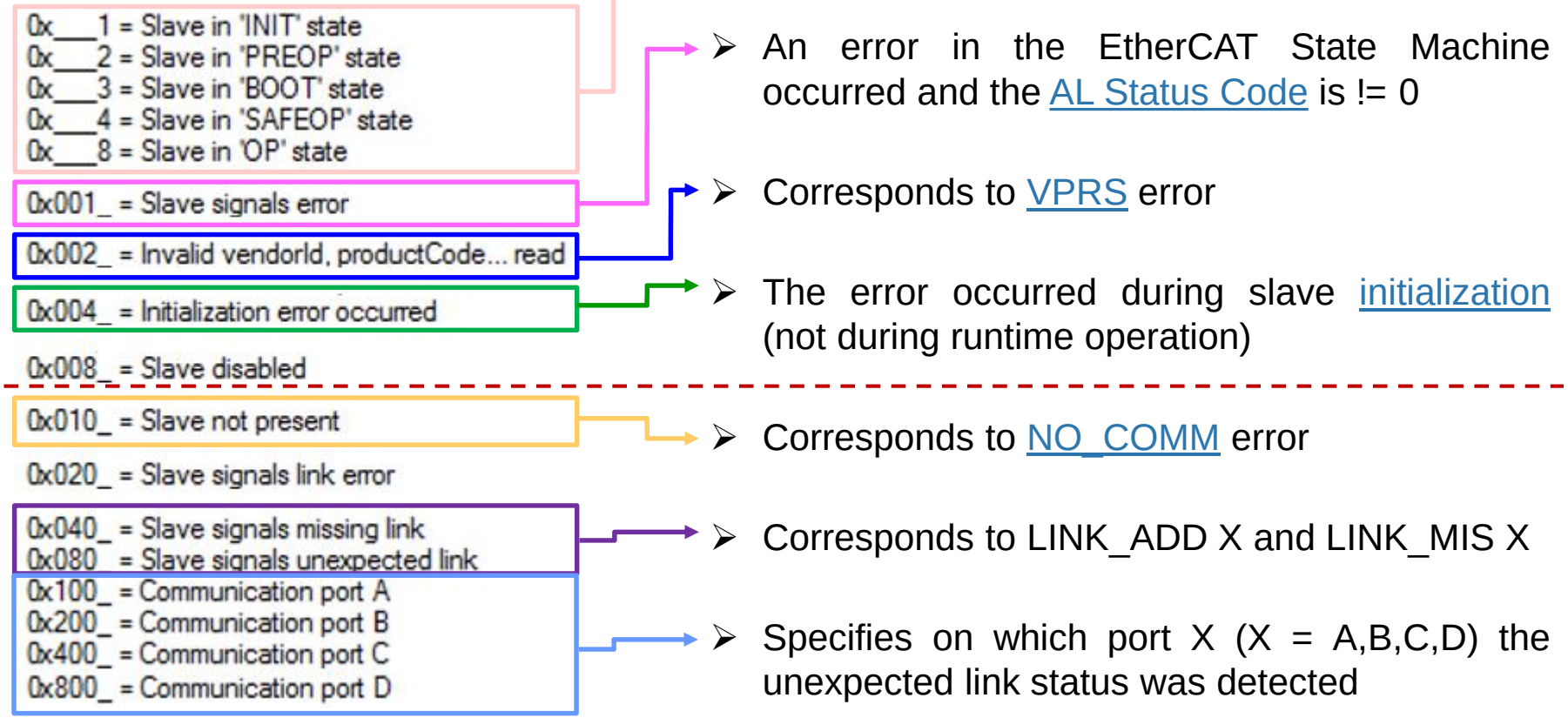
BECKHOFF

Through this variable the EtherCAT Master summarizes the diagnostic information collected from the network, cyclically reports several error conditions:



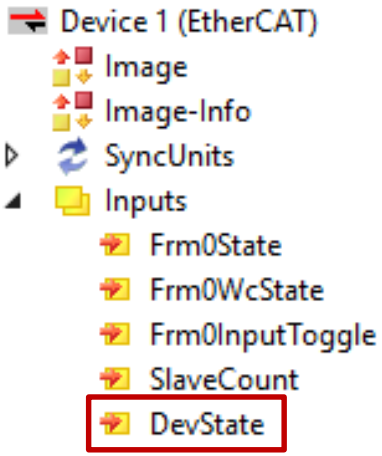
BECKHOFF

Examples of error diagnostic information reported by the State variable are:



BECKHOFF

The EtherCAT master provides an own additional cyclic diagnostic variable:

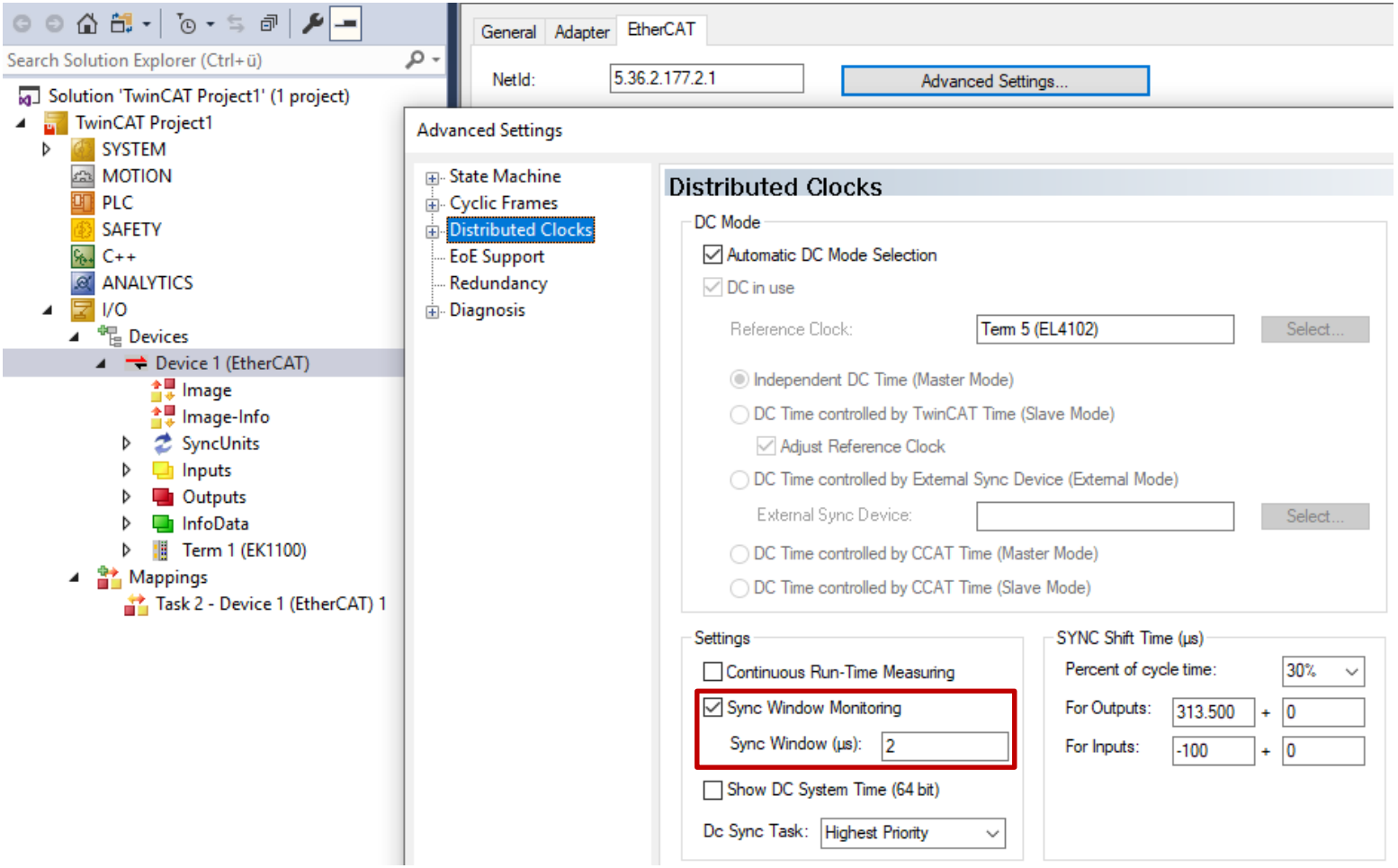


0x0001 = Link error
0x0002 = I/O locked after link error (I/O reset required)
0x0004 = Link error (redundancy adapter)
0x0008 = Missing one frame (redundancy mode)
0x0010 = Out of send resources (I/O reset required)
0x0020 = Watchdog triggered
0x0040 = Ethernet driver (miniport) not found
0x0080 = I/O reset active
0x0100 = At least one device in "INIT" state
0x0200 = At least one device in "PRE-OP" state
0x0400 = At least one device in "SAFE-OP" state
0x0800 = At least one device indicates an error state
0x1000 = DC not in sync

- **Link error**, **Link error (redundancy mode)**: the physical link between the (redundancy) network adapter in the master and the first slave connected to it is missing.
- **Missing one frame (redundancy mode)**: one frame was lost when the network was interrupted and the redundancy took action.
- **At least one device…** : global monitoring of the EtherCAT State Machine in connected slaves.

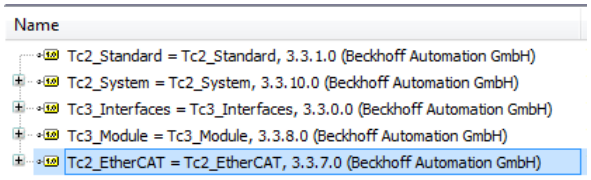
BECKHOFF

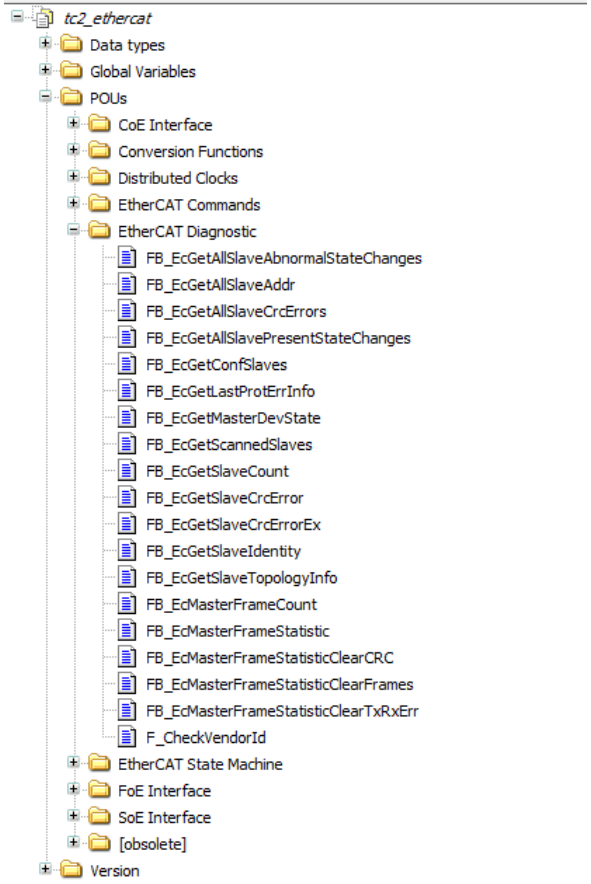
- **DC not in sync**: the maximum deviation among DC clocks exceeds the configured threshold, if “Sync Window Monitoring” is active:



Acyclic Information – TcEtherCAT PLC Library

BECKHOFF



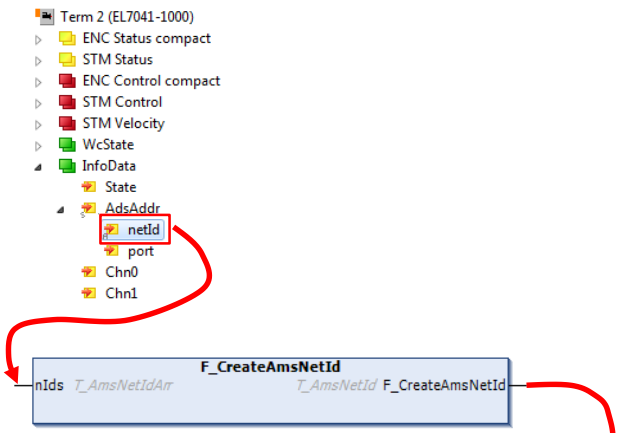


The **TcEtherCAT.lib** library (available free-of-charge with both the default installation of TC2 and TC3) provides function-blocks which enable to acyclically diagnose the EtherCAT network

- Frame statistics
- CRC statistics
- Slave Identity
- Number and list of configured Slaves
- Number and list of found Slaves

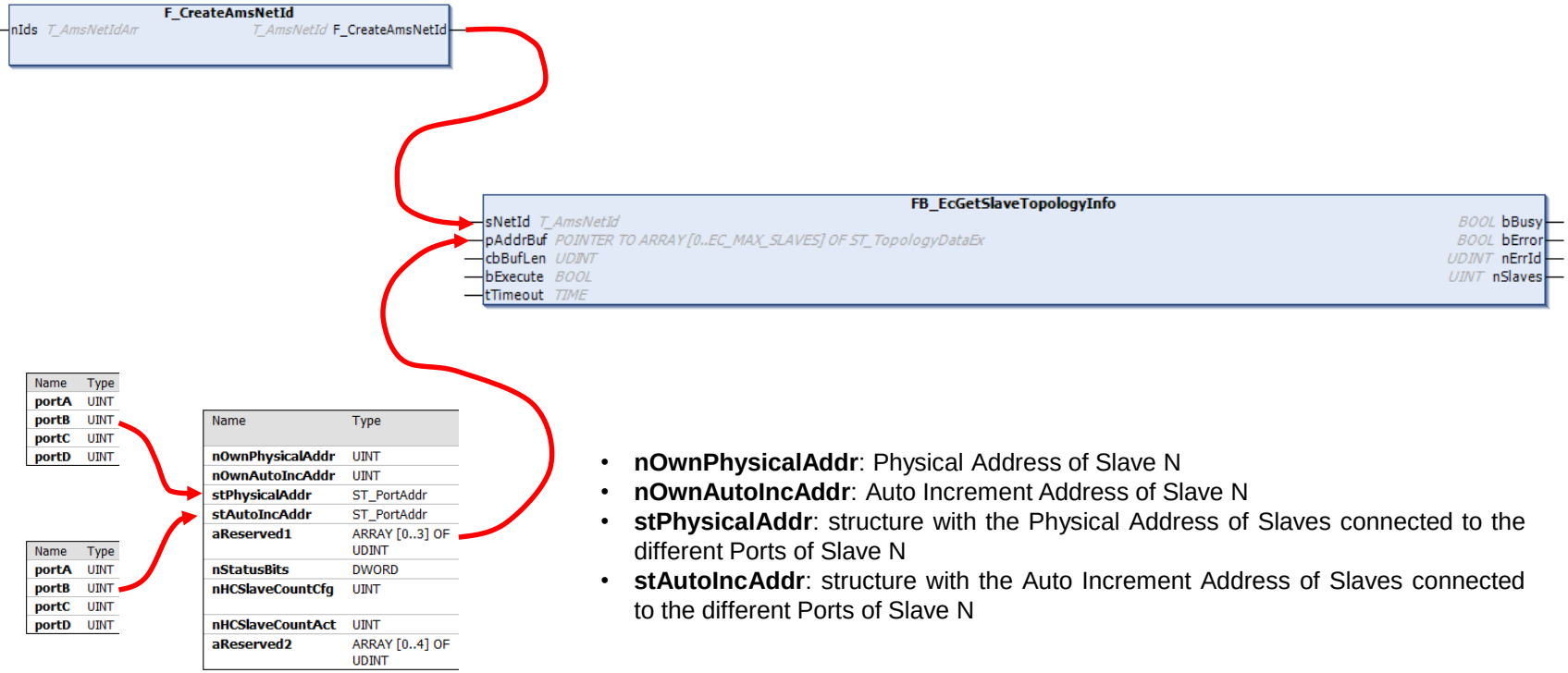
The library basically allows to automatically read from the PLC program all the information which is online visible in the TwinCAT project.

BECKHOFF



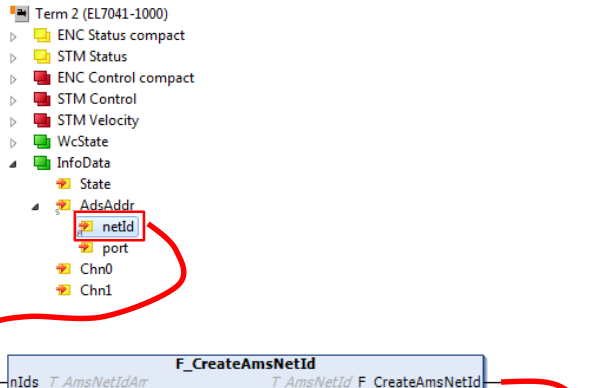
The Function Block **FB_EcGetSlaveTopologyInfo** returns information on how the ports of different slaves are connected to each other, and therefore the structure of the (configured) network topology.

(If Hot Connect groups are present in the configuration, they will be listed at the and of the ARRAY).

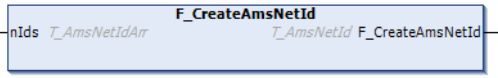


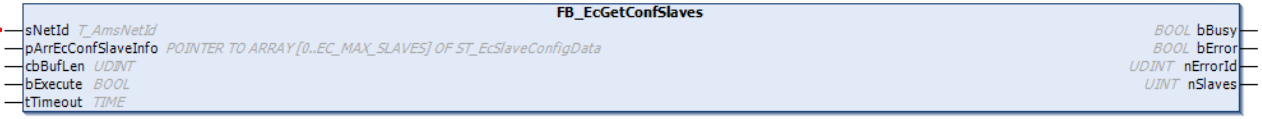
- **nOwnPhysicalAddr**: Physical Address of Slave N
- **nOwnAutoIncAddr**: Auto Increment Address of Slave N
- **stPhysicalAddr**: structure with the Physical Address of Slaves connected to the different Ports of Slave N
- **stAutoIncAddr**: structure with the Auto Increment Address of Slaves connected to the different Ports of Slave N

BECKHOFF



The Function Block **FB_EcGetConfSlaves** allows to retrieve the information on which Slaves should form the EtherCAT network according to the “offline” TwinCAT configuration.



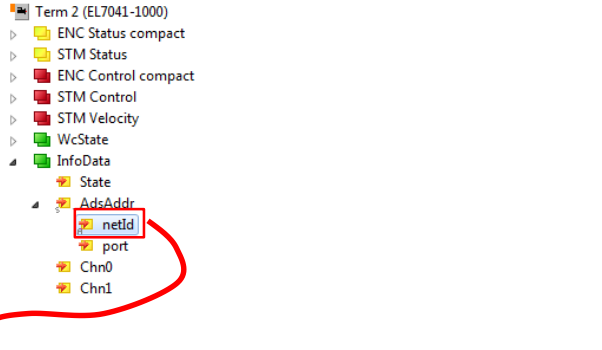


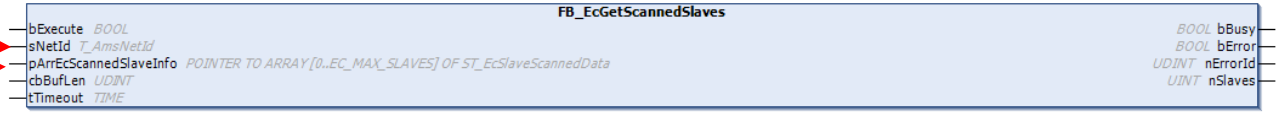
Name	Type
nEntries	WORD
nAddr	WORD
sType	STRING(15)
sName	STRING(31)
nDevType	DWORD
stSlaveIdentity	ST_EcSlaveIdentity
nMailboxOutSize	WORD
nMailboxInSize	WORD
nLinkStatus	BYTE

Name	Type
vendorId	UDINT
productCode	UDINT
revisionNo	UDINT
serialNo	UDINT

BECKHOFF

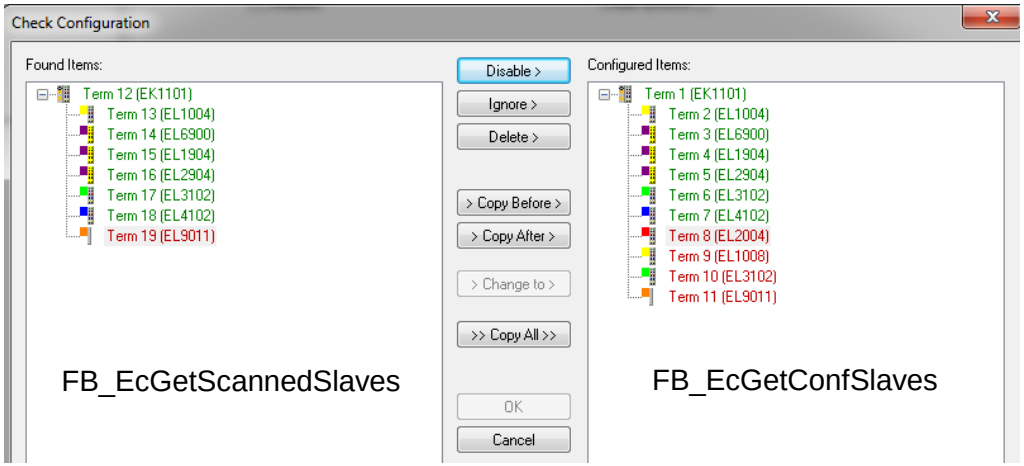
The Function Block **FB_EcGetScannedSlaves** allows to retrieve the information on which Slaves are actually detected “online” on the network. This information can be compared to the “offline” expected configuration retrieved with *FB_EcGetConfSlaves*.





Name	Type
nEntries	WORD
nAddr	WORD
stSlaveIdentity	ST_EcSlaveIdentity
ndlStatusReg	WORD

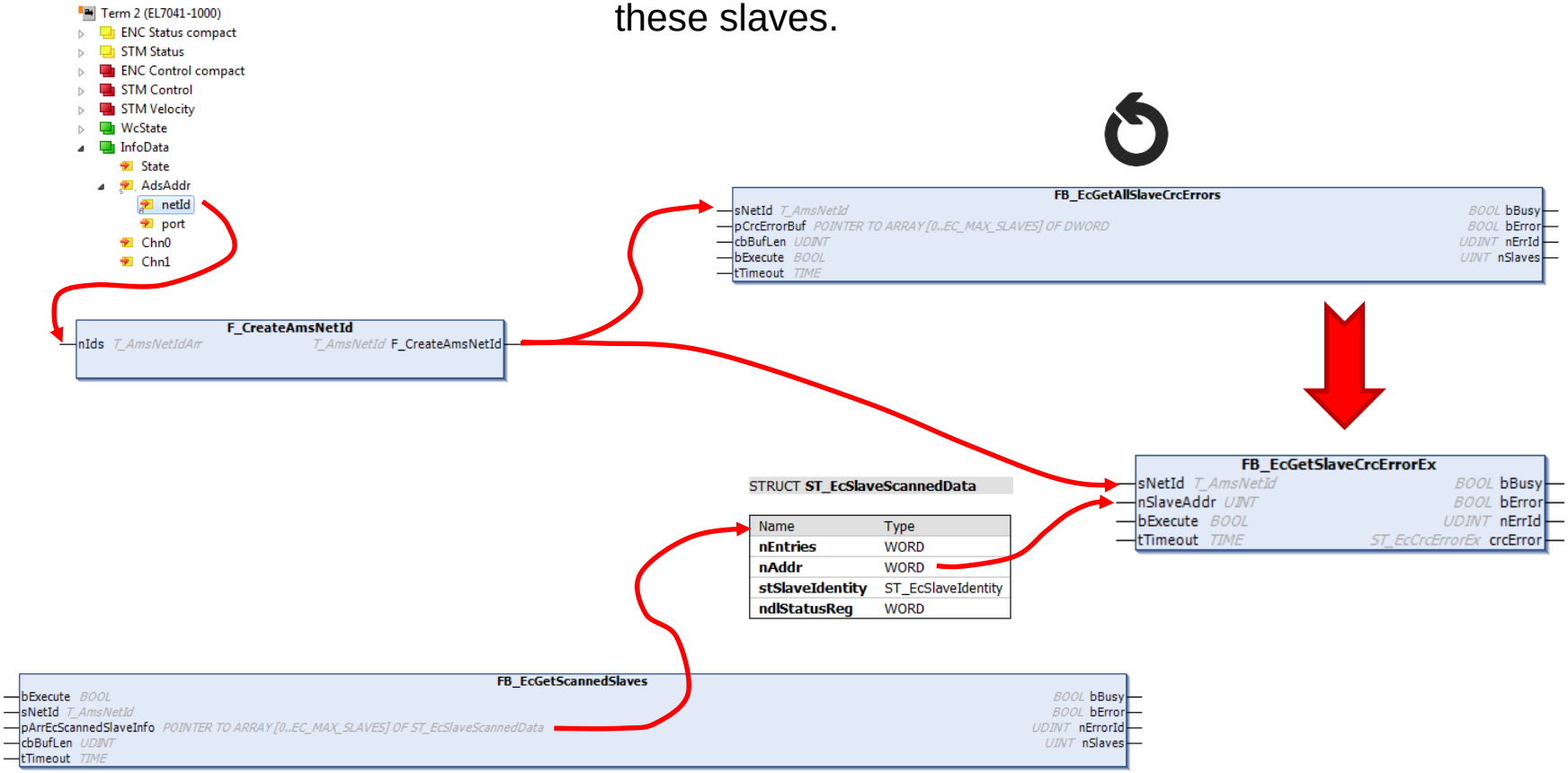
Name	Type
vendorId	UDINT
productCode	UDINT
revisionNo	UDINT
serialNo	UDINT



FB_EcGetScannedSlaves

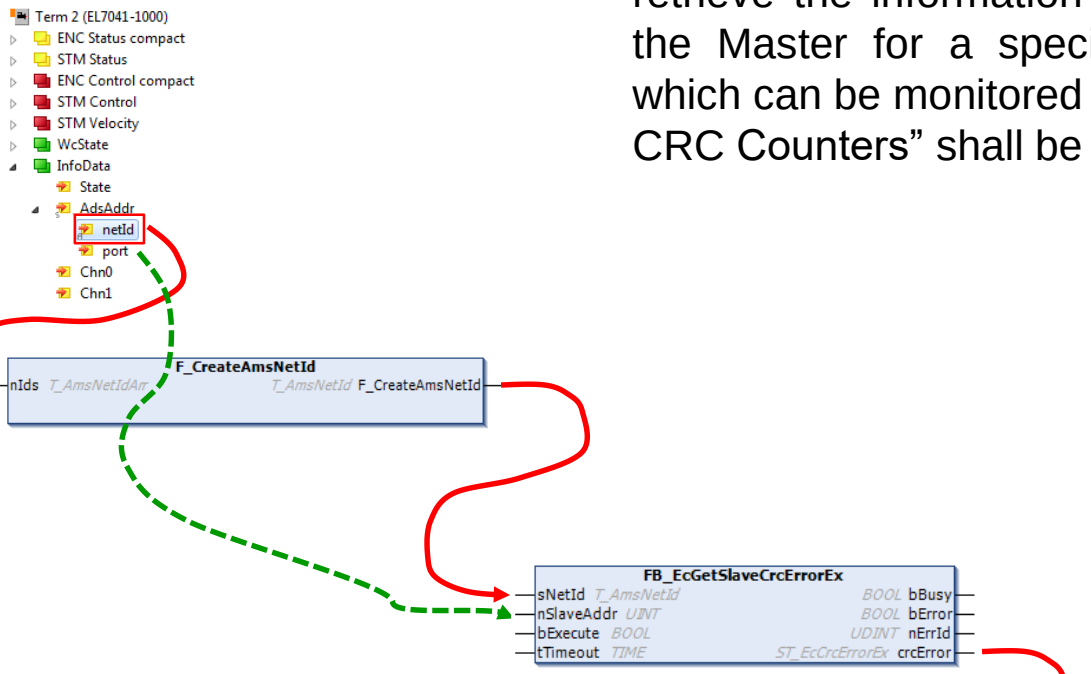
FB_EcGetConfSlaves

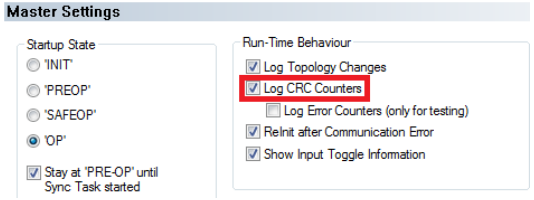
The Function Block **FB_EcGetAllSlaveCrcErrors** can be periodically called in order to determine if one or more slaves report CRC errors. In this case, the Function Block **FB_EcGetSlaveCrcErrorEx** can be called only for these slaves.

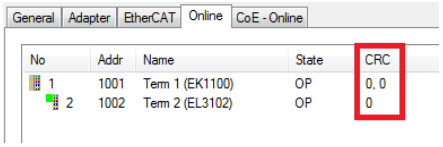


BECKHOFF

The Function Block **FB_EcGetSlaveCrcErrorEx** allows to retrieve the information about the CRC errors logged by the Master for a specific Slave (the same information which can be monitored in the TwinCAT project). Flag “Log CRC Counters” shall be active!



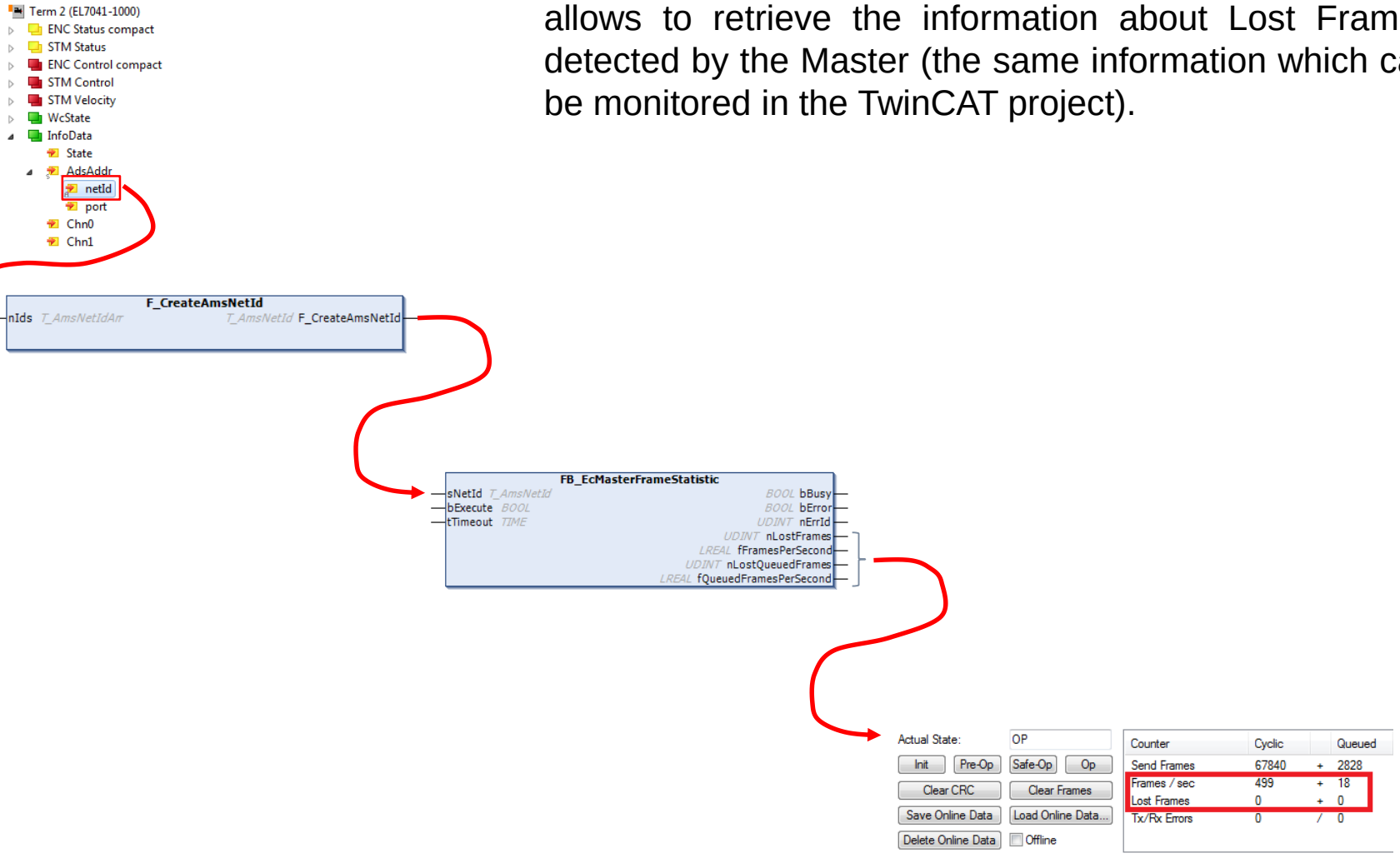




Acyclic Information – TcEtherCAT PLC Library

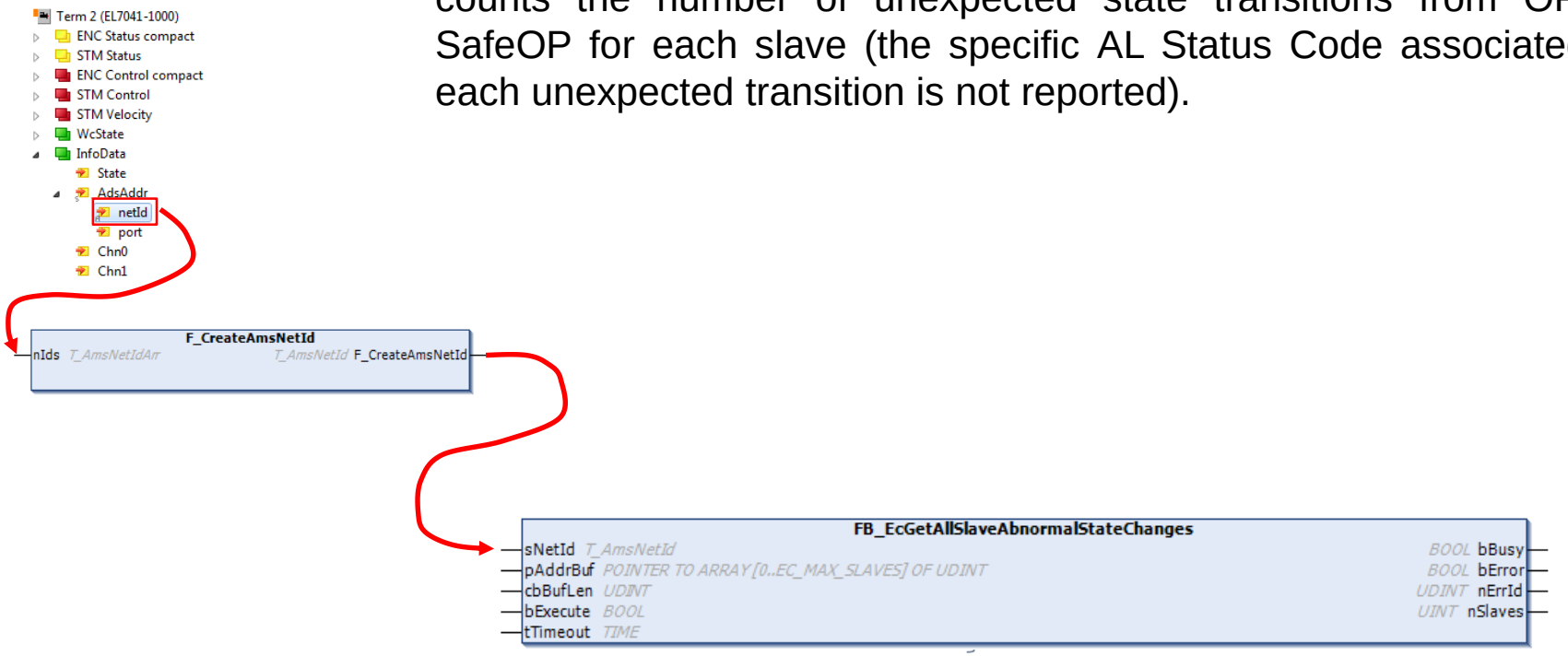
BECKHOFF

The Function Block **FB_EcMasterFrameStatistics** allows to retrieve the information about Lost Frames detected by the Master (the same information which can be monitored in the TwinCAT project).

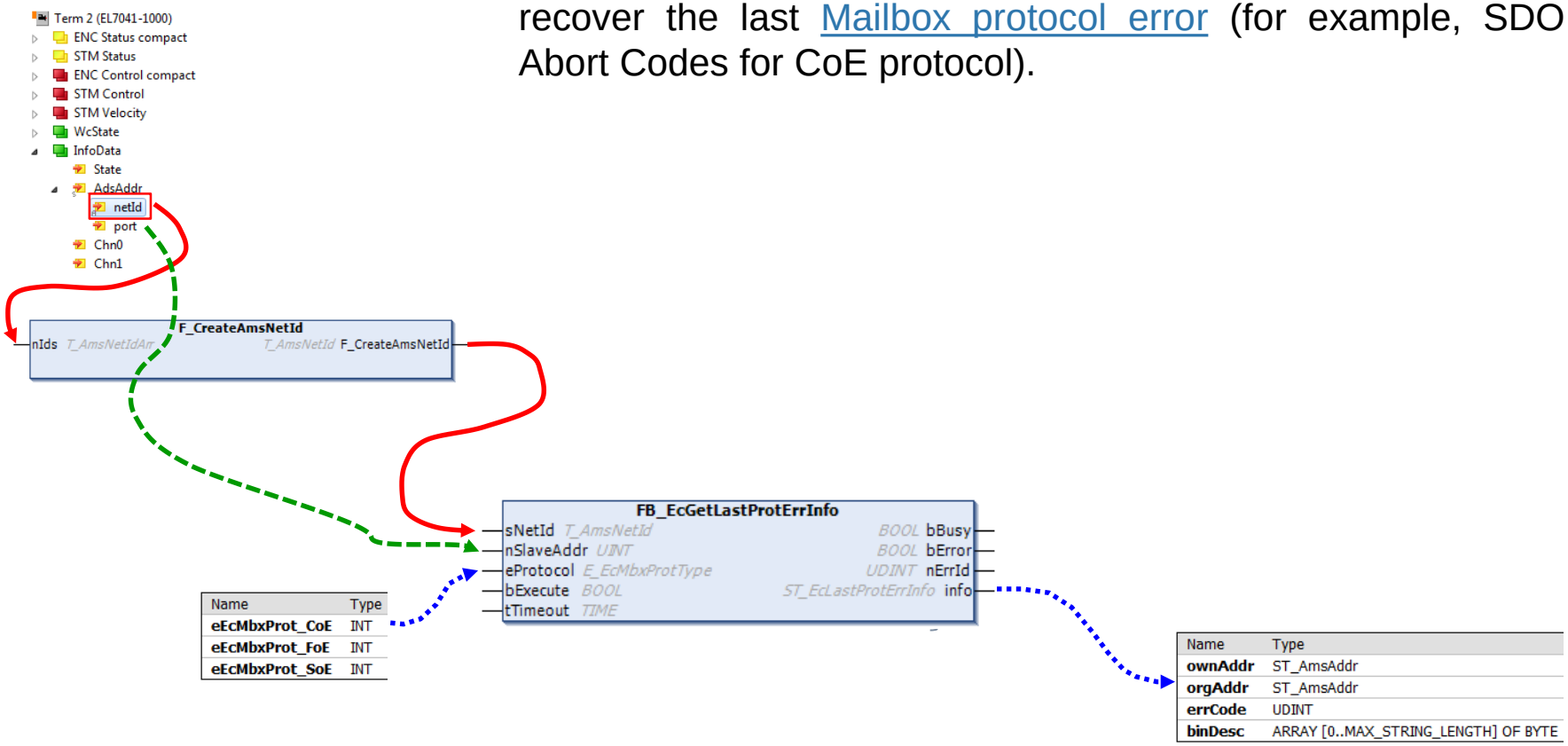


BECKHOFF

The Function Block **FB_EcGetAllSlaveAbnormalStateChanges** counts the number of unexpected state transitions from OP to SafeOP for each slave (the specific AL Status Code associated to each unexpected transition is not reported).



The Function Block **FB_EcGetLastProtErrInfo** enables to recover the last Mailbox protocol error (for example, SDO Abort Codes for CoE protocol).



The next error-free Mailbox access to the same slave deletes the error memory!

BECKHOFF

Diagnostics with Wireshark

When the Software Diagnosis in not Enough - Wireshark

BECKHOFF

The diagnostic via Wireshark does not replace the diagnostic in the PLC, and for users should be the **last option** when all other diagnostic functionalities didn´t help to locate the problem.

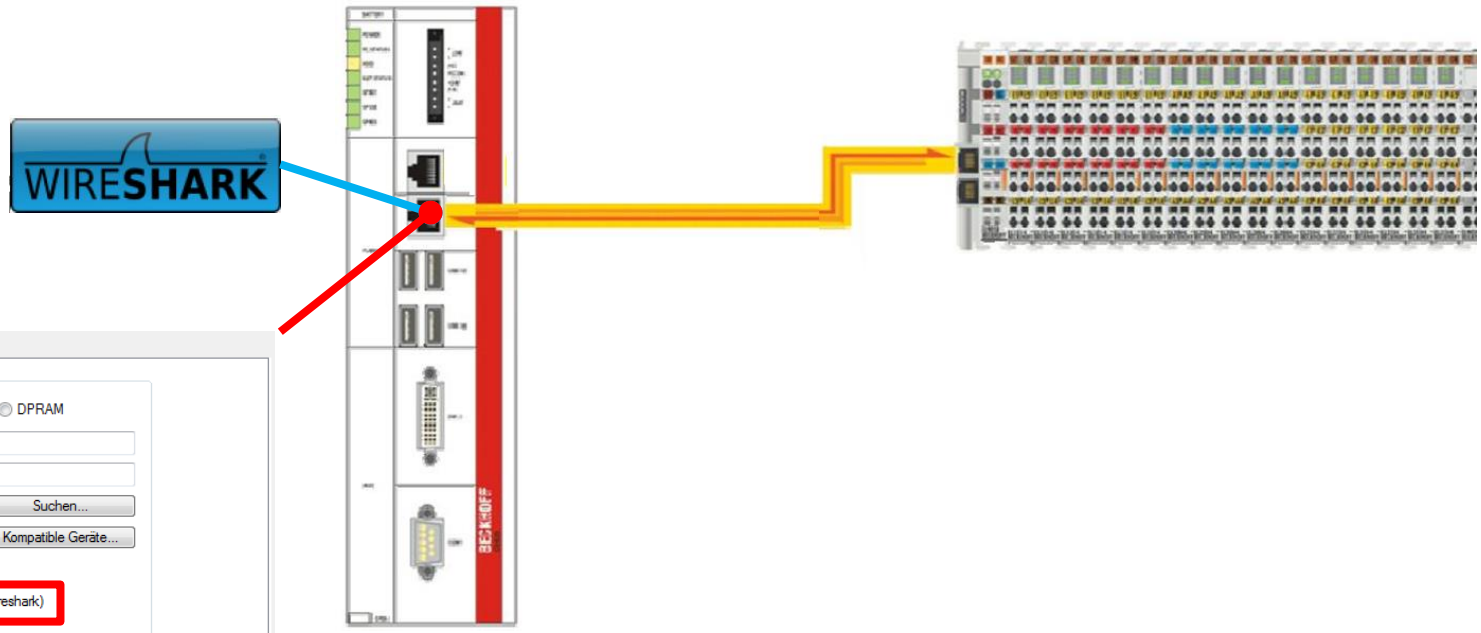
Technical data

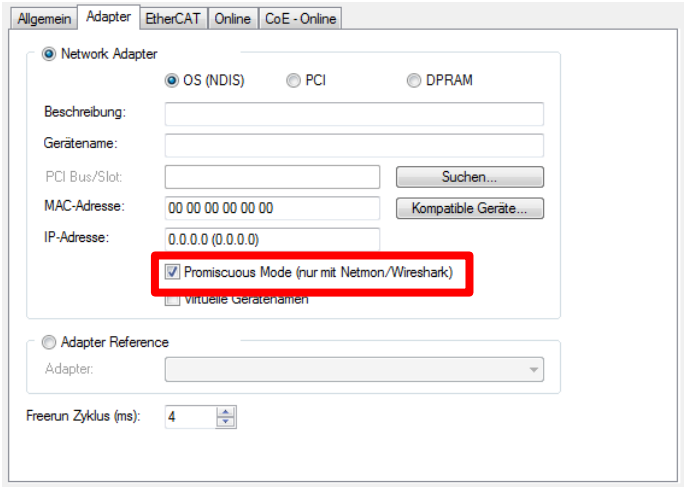
Technical data	ET2000
Bus system	Ethernet (all Ethernet based protocols (IEEE 802.3)
Number of ports/channels	8/4
Ethernet interface	100BASE-TX Ethernet with RJ45
Uplink port	1 GBit/s
Baud rate	Probe ports: 100 MBit/s, Uplink port: 1 GBit/s
Delay	< 1 µs
Resolution time stamp	1 ns
Accuracy time stamp	40 ns
Diagnosis	2 LEDs per channel - Link/Activity 8 Status LED for future diagnostics
Supply voltage	24 V_{DC} (18 V_{DC} ... 30 V_{DC})
Software interface	"Wireshark" extension currently required "Wireshark" version

In order to take a Wireshark trace, **different hardware setups** are possible.

Wireshark Capture Configuration 1

BECKHOFF

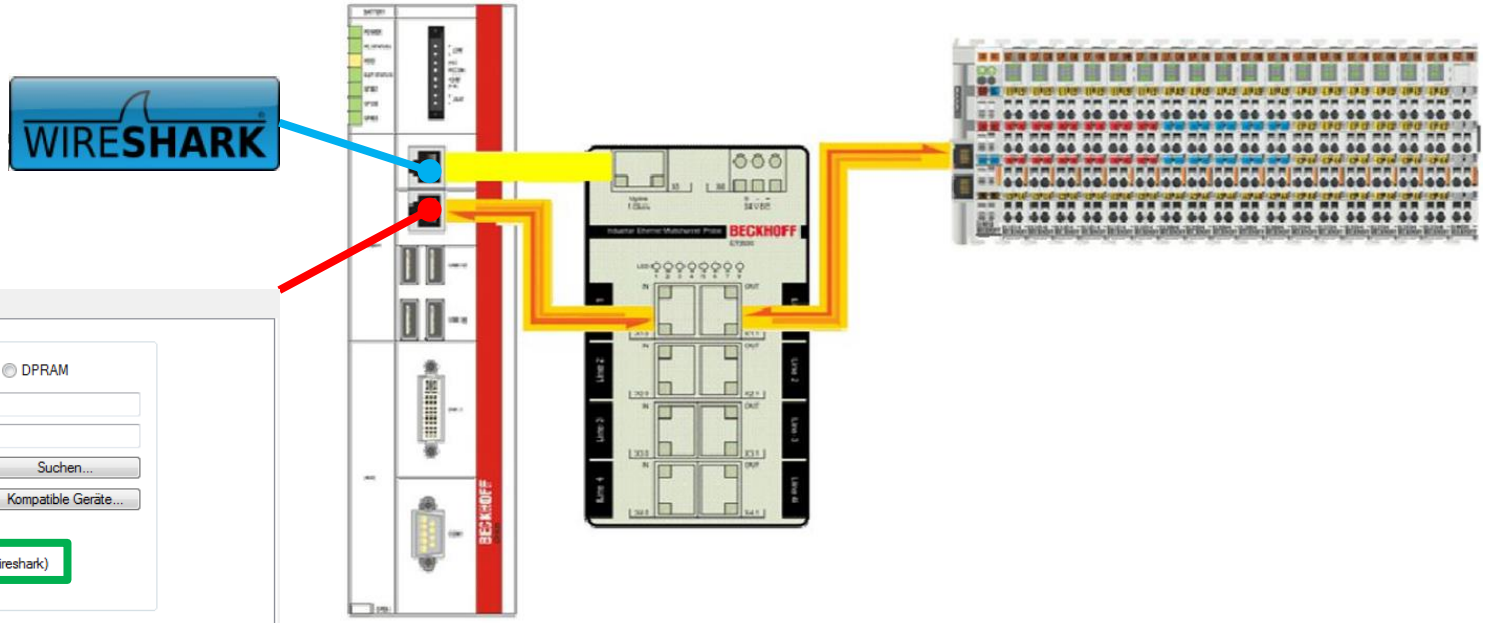


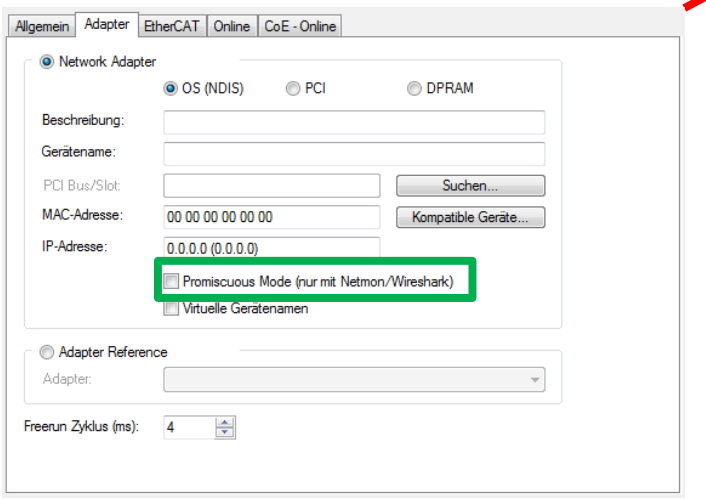


- No Windows CE
- No precise time-stamp

Wireshark Capture Configuration 2

BECKHOFF

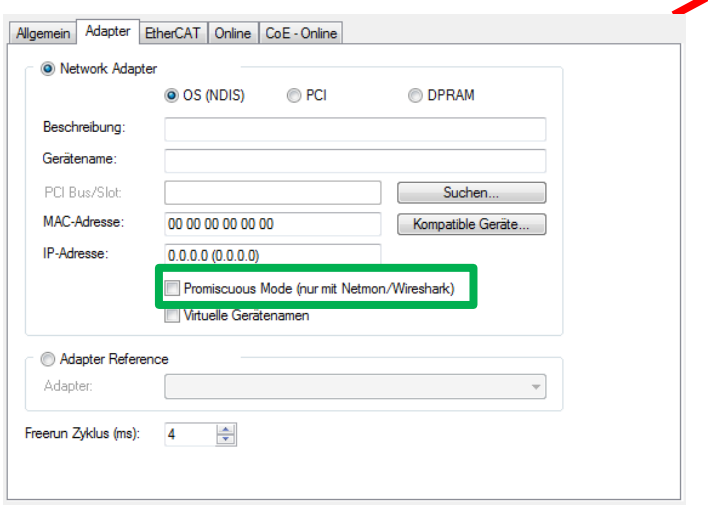


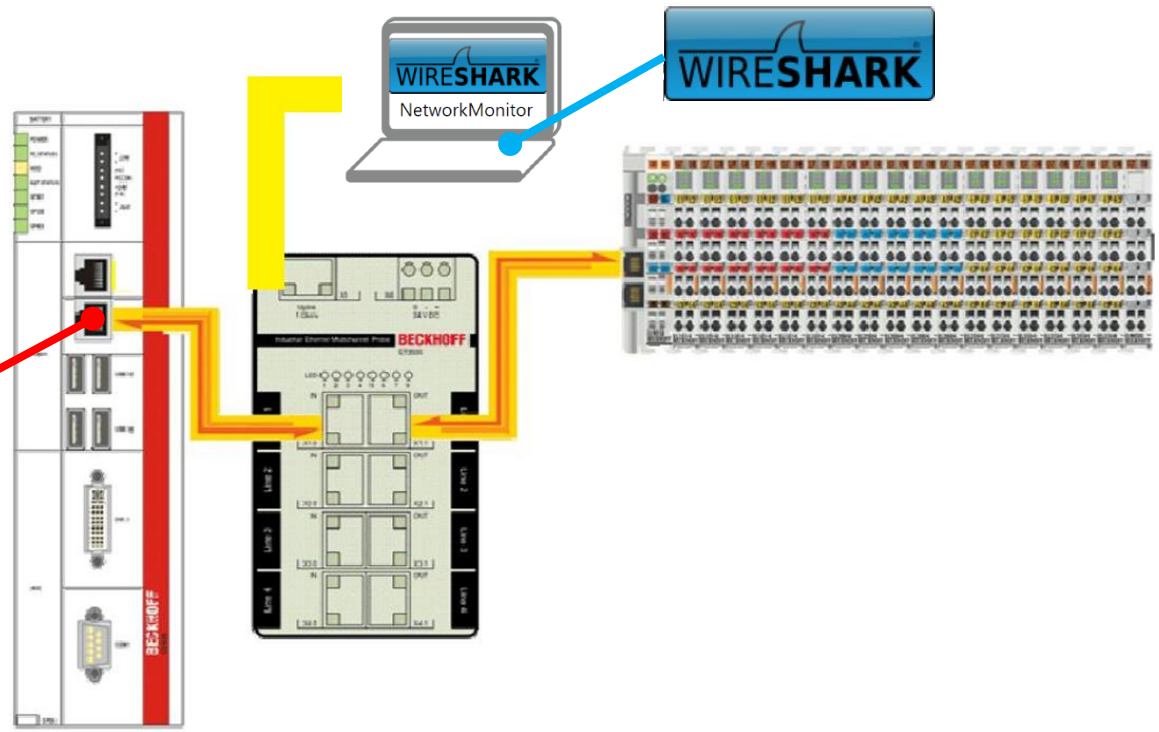


- No Windows CE
- Precise time-stamp

Wireshark Capture Configuration 3

BECKHOFF





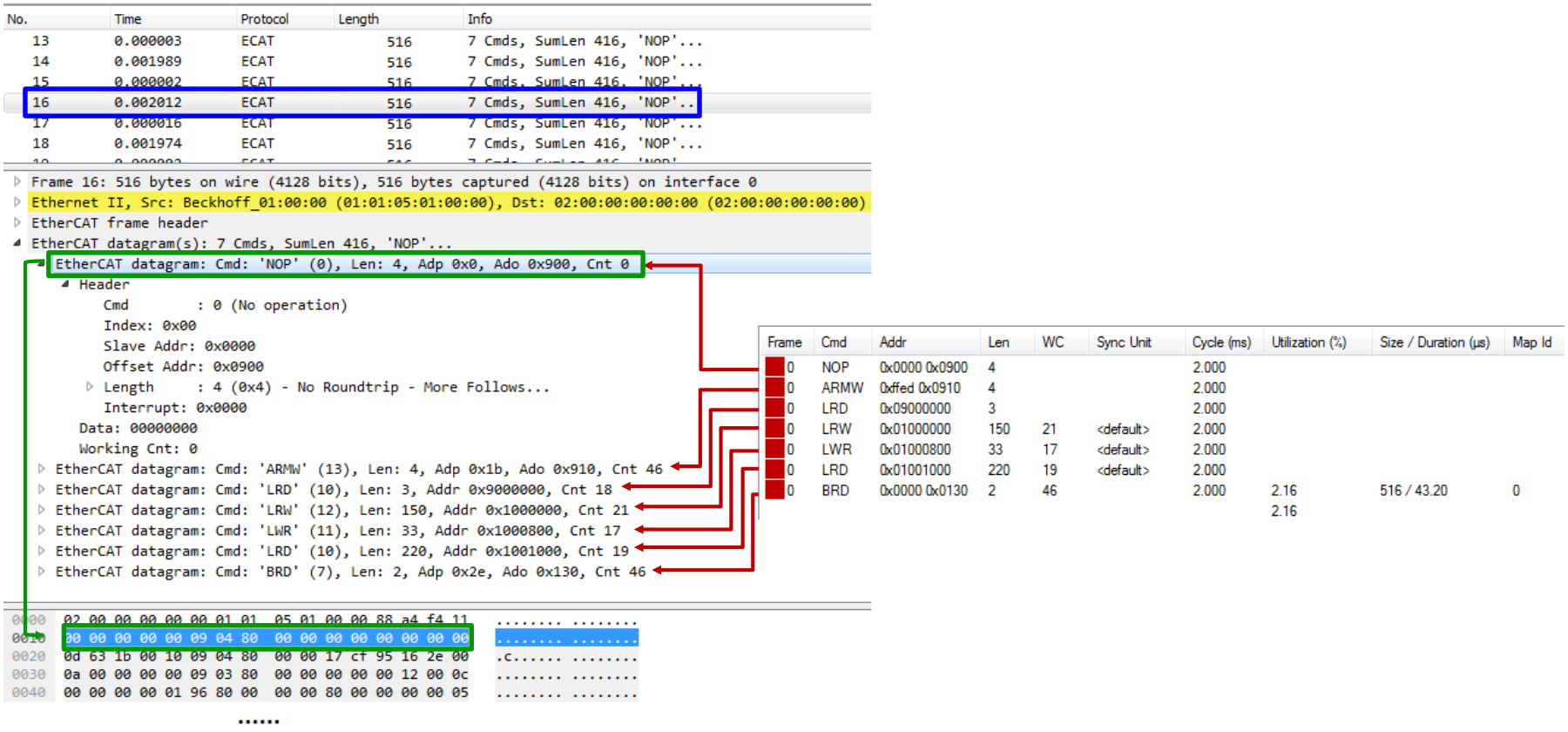
- Also Windows CE
- Precise time-stamp

Wireshark Example

BECKHOFF

Wireshark allows to trace **list**, **structure** and **content** of each EtherCAT frame (each frame will be traced twice in each capture!).

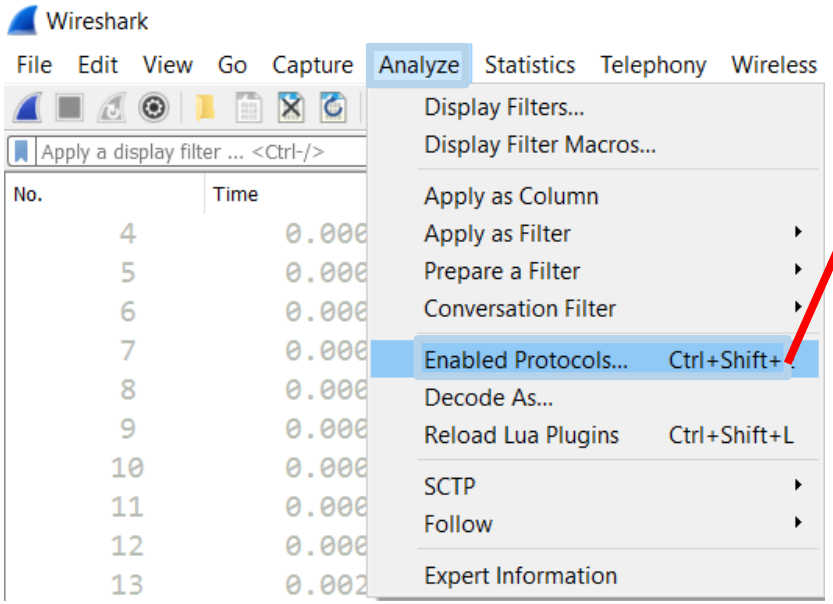
The following example shows the capture of a cyclic frame:

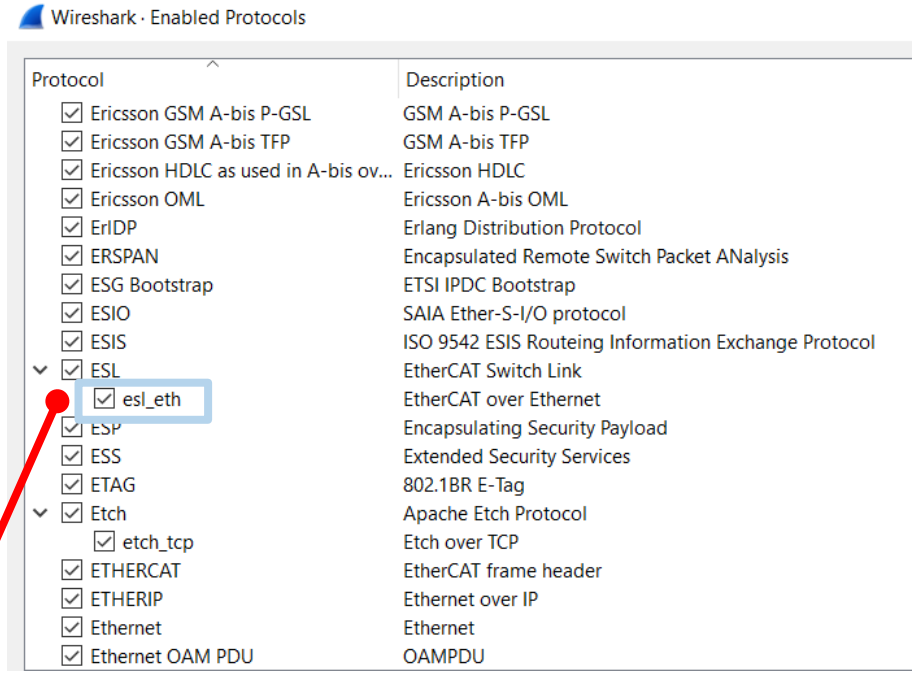


Wireshark - Time-Stamping Protocol (only with ET2000)

BECKHOFF

In order to retrieve extremely **precise time-stamping information** in Wireshark when using ET2000, the dissector for ESL protocol shall be enabled. Enabling in Wireshark **Version 2**:

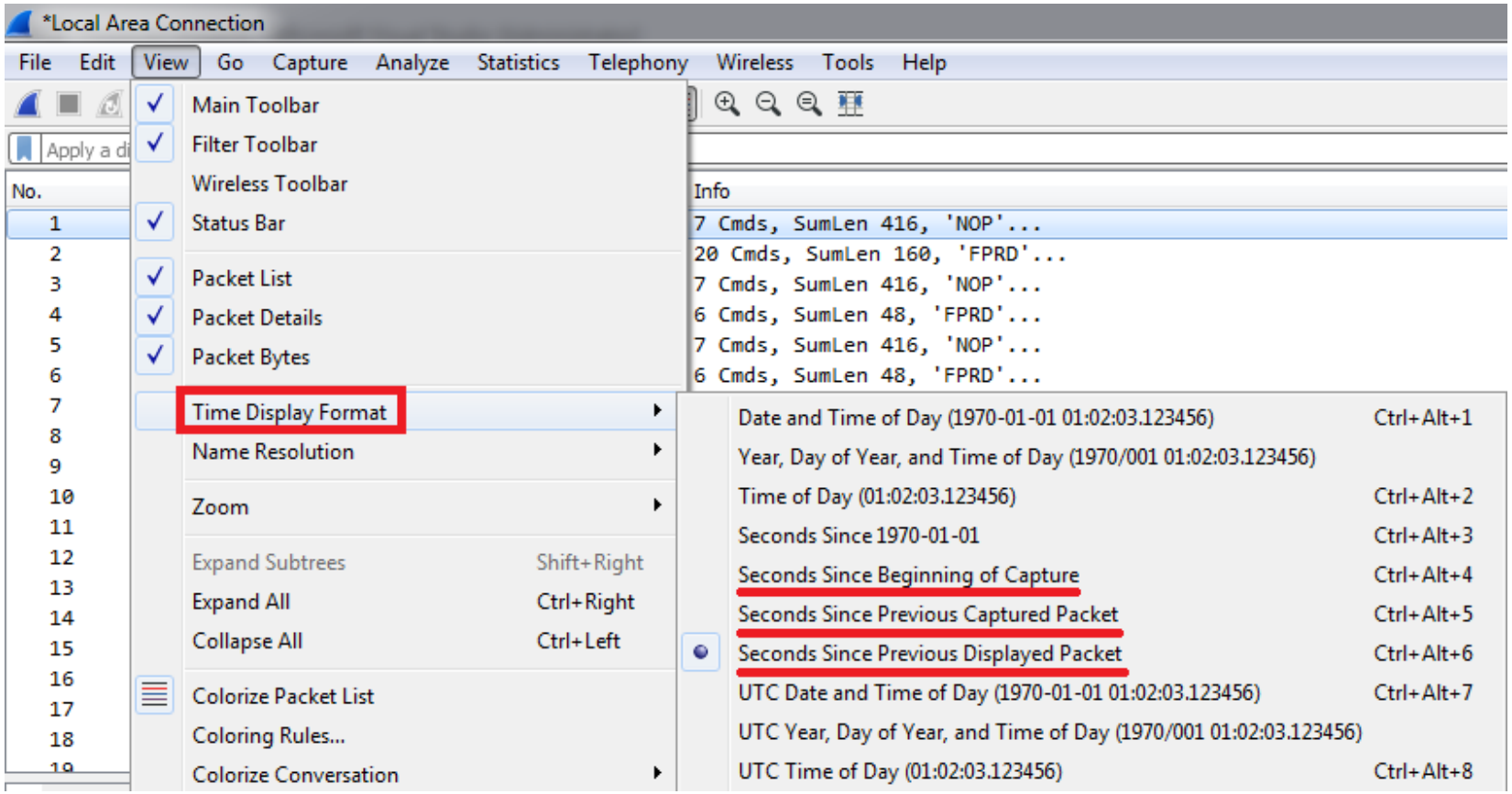




Wireshark - Display Time Format

BECKHOFF

The user can select the most suitable format when displaying **time information** in a Wireshark trace:



Wireshark – Some Filter Options

BECKHOFF

Captured frames can be filtered. Some possible **filters** are:

- **Only cyclic frames**

  Filter: (ecat.cmd == 0x0a) || (ecat.cmd == 0x0b) || (ecat.cmd == 0x0c)

- **Only mailbox communication**

  Filter: ecat_mailbox

  Filter: ecat_mailbox.coe

  Filter: ecat_mailbox.soe

  Filter: ecat_mailbox.foe

  Filter: ecat_mailbox.eoe

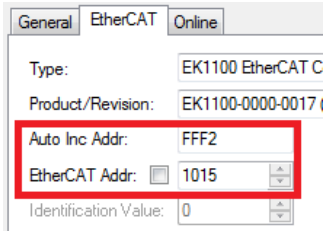
- **Only access to a specific register or register range**

  Filter: ecat.ado == <*register_address*>

  Filter: (ecat.ado >= <*lower_address*>) && !(ecat.ado >= <*upper_address*>)

- **Only datagrams for a slave with specific fixed or auto-increment address**

  Filter: ecat.adp == <*slave_address*>



Filter options can be freely **combined** into logical expressions:

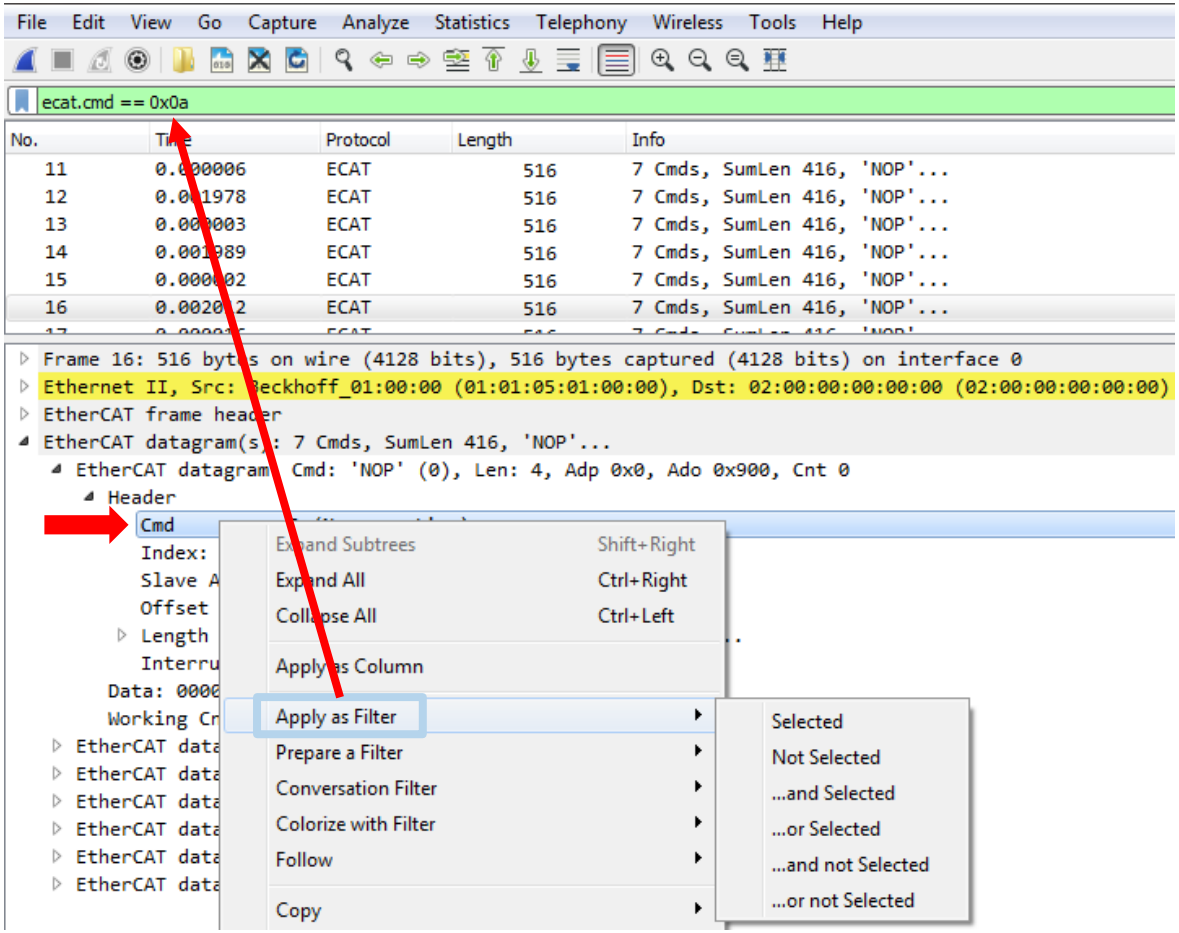
Operator	Meaning
&&	AND
||	OR
!	NOT
==	equal
>=	greater or equal
<=	smaller or equal
>	greater
<	smaller

Other filter options are available on https://www.wireshark.org/docs/dfref/e/ecat.html

Wireshark – Some Filter Options

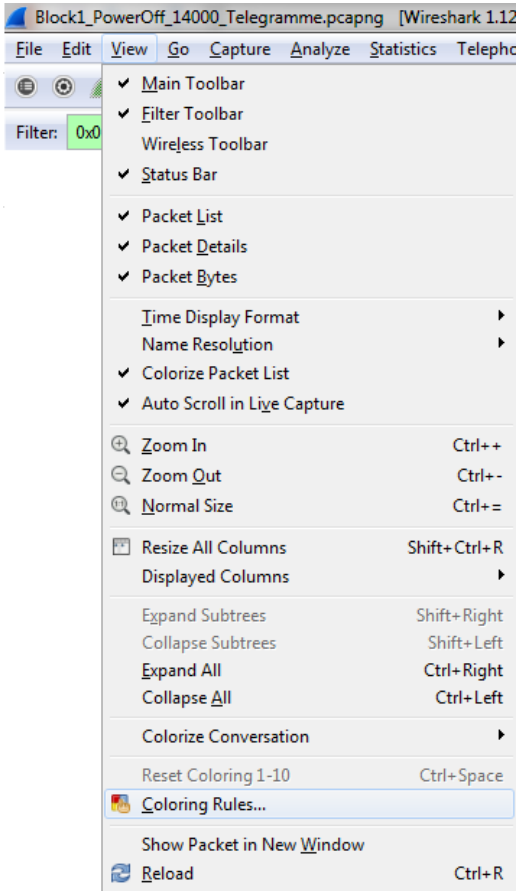
BECKHOFF

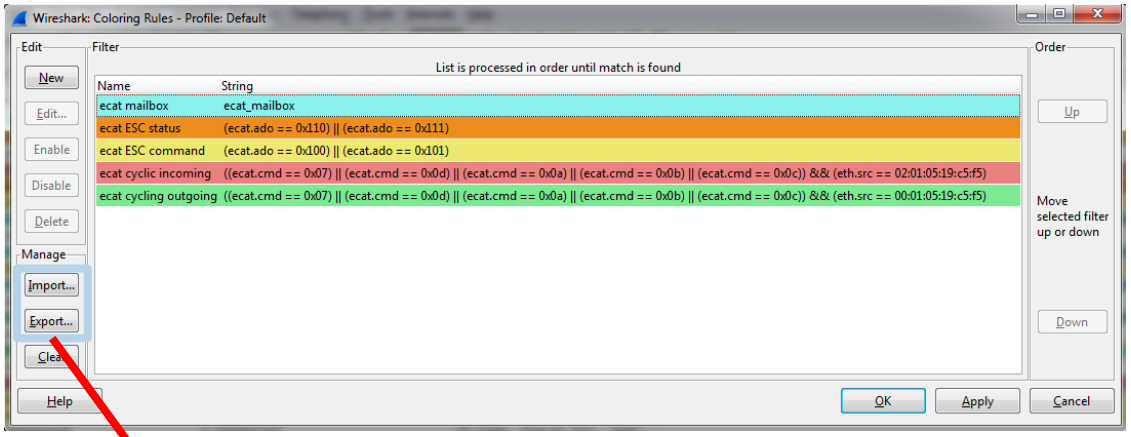
Filter options can be picked directly from the trace by clicking on „**Apply as Filter**“:



Wireshark – Coloring Rules

In order to better display the different types of information, **colouring rules** can be associated to specific filtering options:



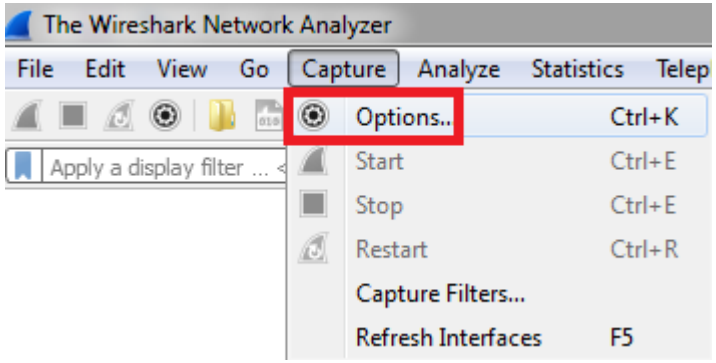


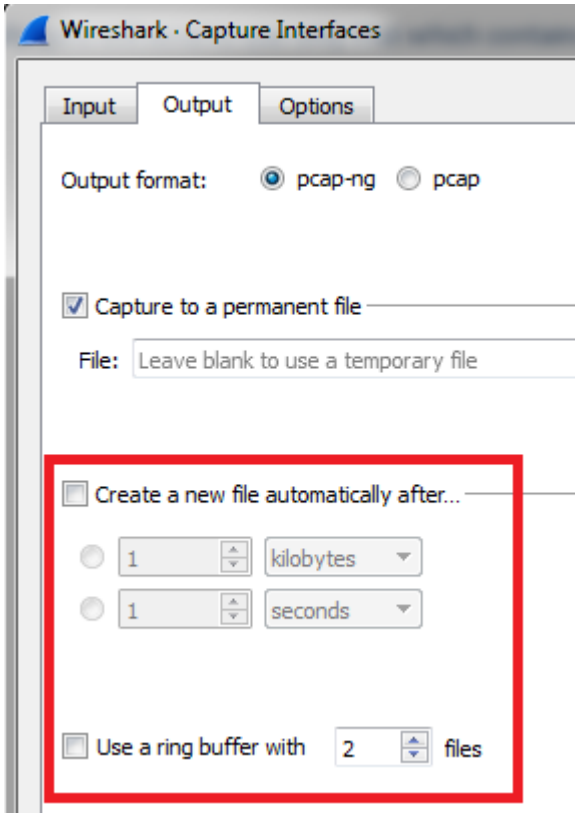
Once defined, coloring rules are applied to all Wireshark traces taken on the same PC. In order to transfer the same coloring rules to another PC, they can be exported and re-imported.

Wireshark – Reducing the Size of the Saved Capture

BECKHOFF

Before starting the capture, it is possible to configure Wireshark in order to save separate files automatically instead of saving all frames within one single file:

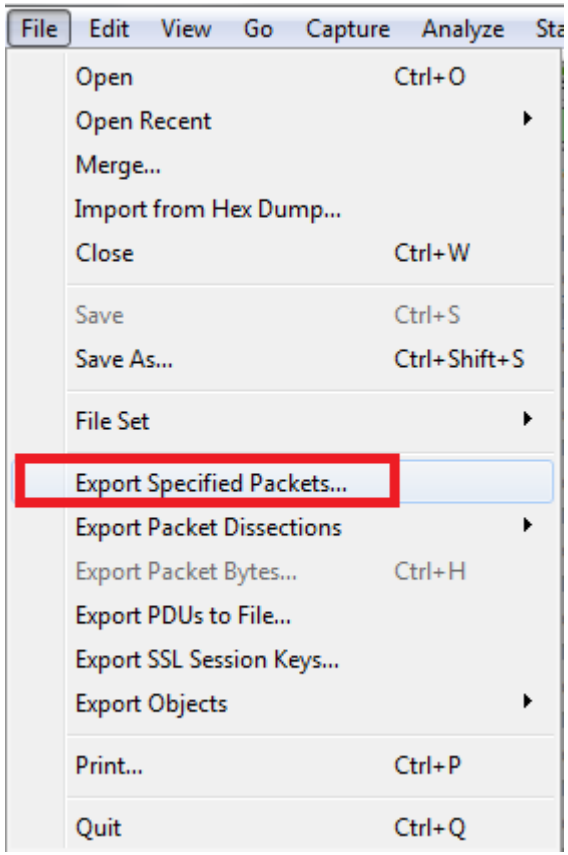


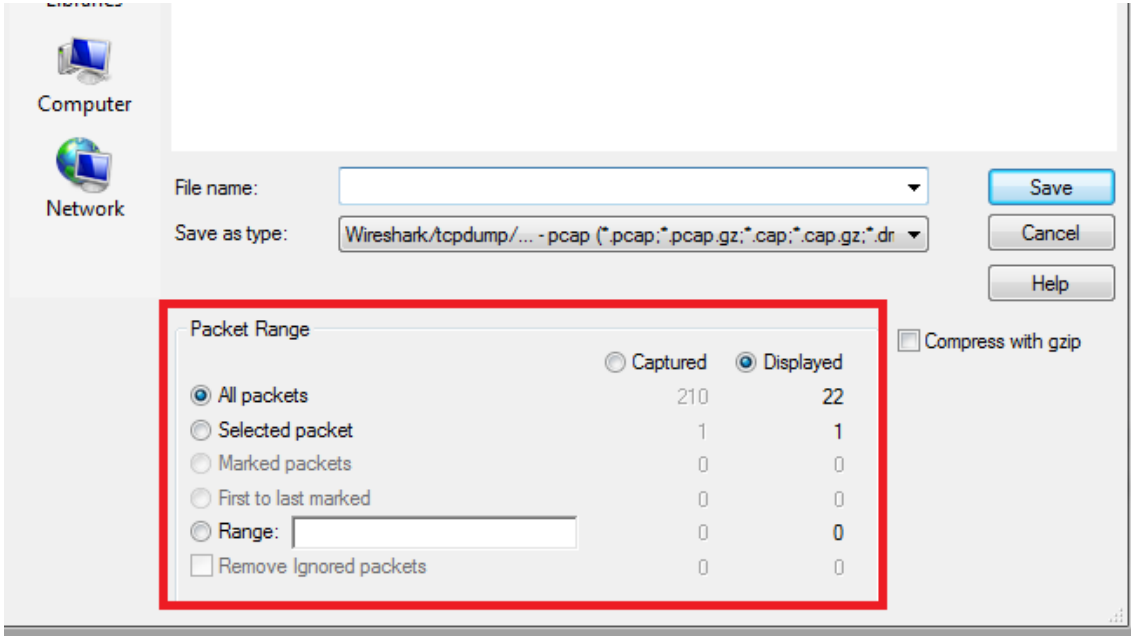


Wireshark – Reducing the Size of the Saved Capture

BECKHOFF

After taking the capture, it is possible to save only a selection of the captured frames:

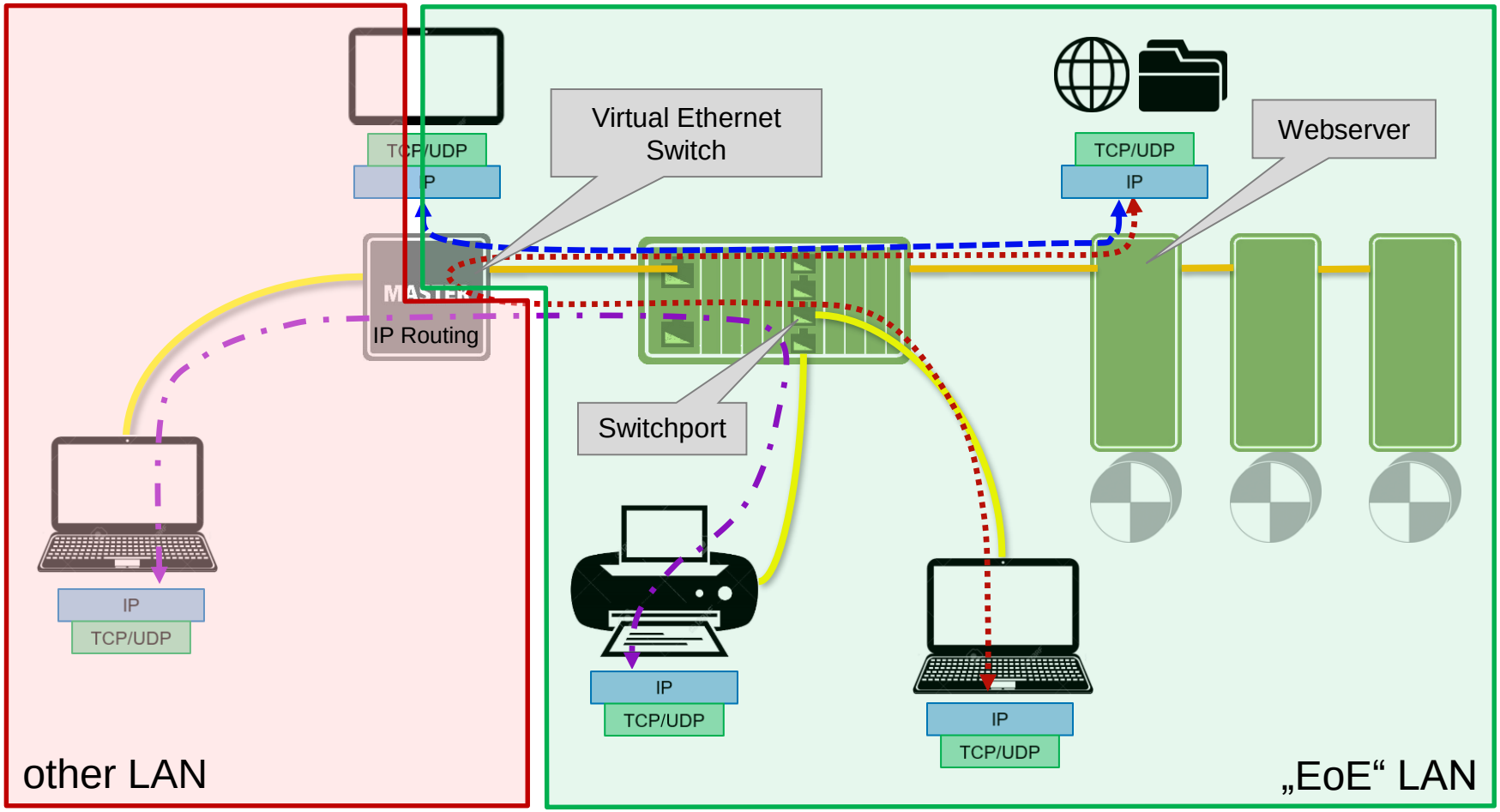




Ethernet over EtherCAT

BECKHOFF

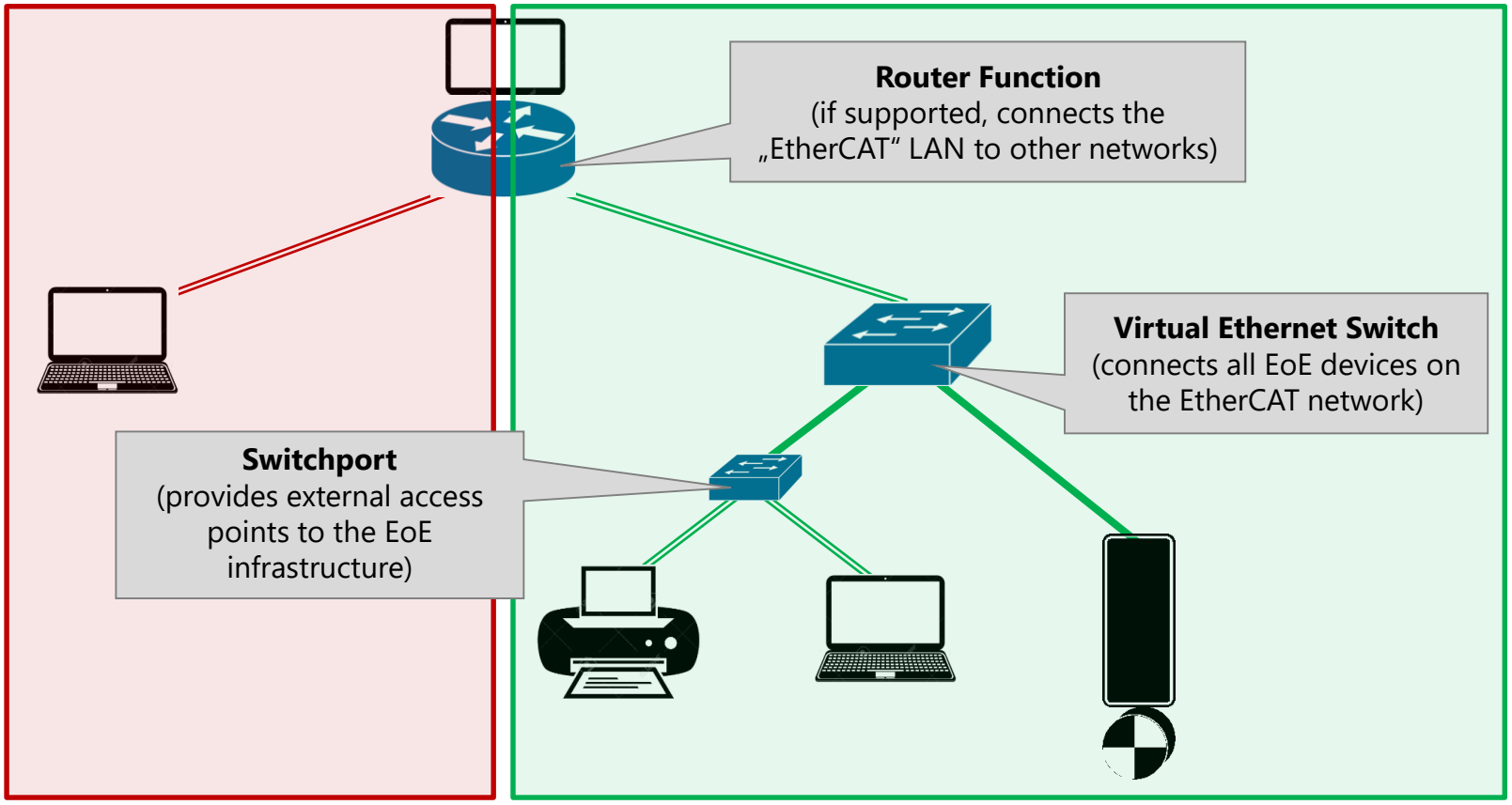
Ethernet over EtherCAT (EoE) enables to transfer TCP/IP- or UDP/IP-based Ethernet traffic (at least partly) using an EtherCAT network as hardware infrastructure.



Equivalent Network Architecture

BECKHOFF

The logical connections correspond to those of a standard **IT infrastructure**.



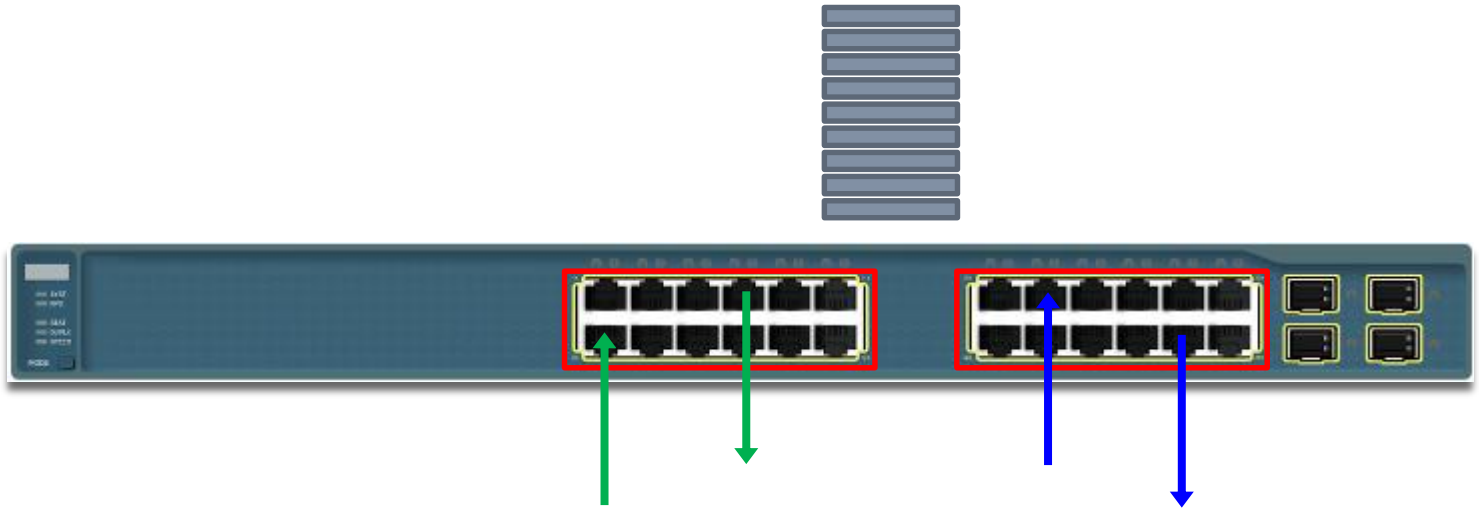
EoE LAN, propagation within EtherCAT segment

EoE LAN, propagation outside EtherCAT segment

other LAN

The **Virtual Ethernet Switch** functionality in the EtherCAT master can be considered as the software equivalent a hardware Ethernet switch:

1. It provides a specific number of connection ports
2. Through these ports, it can interconnect a number of end devices by storing their MAC Ids in an internal look-up table
3. It can store a maximum number of Ethernet frames before forwarding them

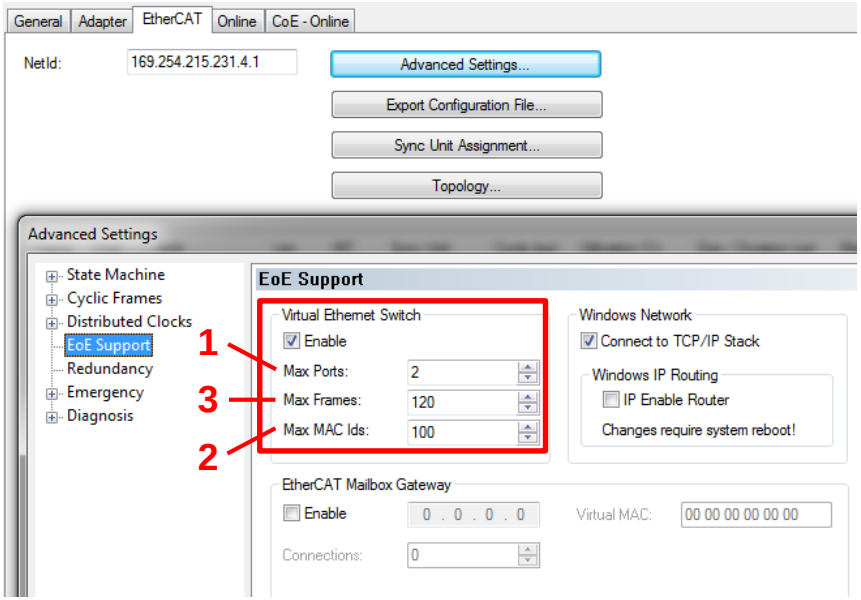


The Virtual Ethernet Switch is automatically activated in TwinCAT whenever the network configuration contains at least one slave supporting EoE.

EoE Master Settings

BECKHOFF

The features of Virtual Ethernet Switch can be **configured** in the master settings:



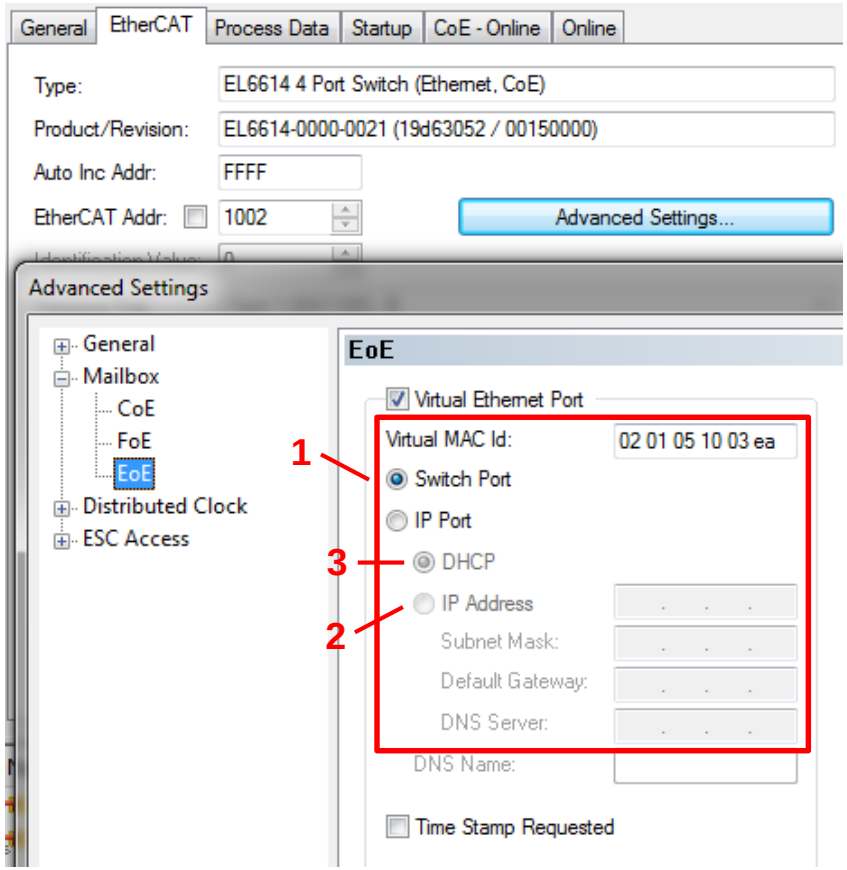
1. Shall be ≥ to the number of EoE slaves connected to the EtherCAT network.
2. Shall be ≥ to the maximum number of different Ethernet end points connected to EoE during operation (multiple connections like over EL6614 or temporarily connected devices should be taken into account).
3. Max. number of Ethernet frames which can temporarily be stored (can be increased in case of large data throughputs or if a slow communication over EoE is experienced).

EoE Slave Settings

BECKHOFF

EtherCAT slaves supporting EoE can be divided into 2 groups:

- Devices not supporting a local TCP/IP or UDP/IP based application (like *switchports*).
- Devices supporting a local TCP/IP (or UDP/IP) based application (like *webservers*).

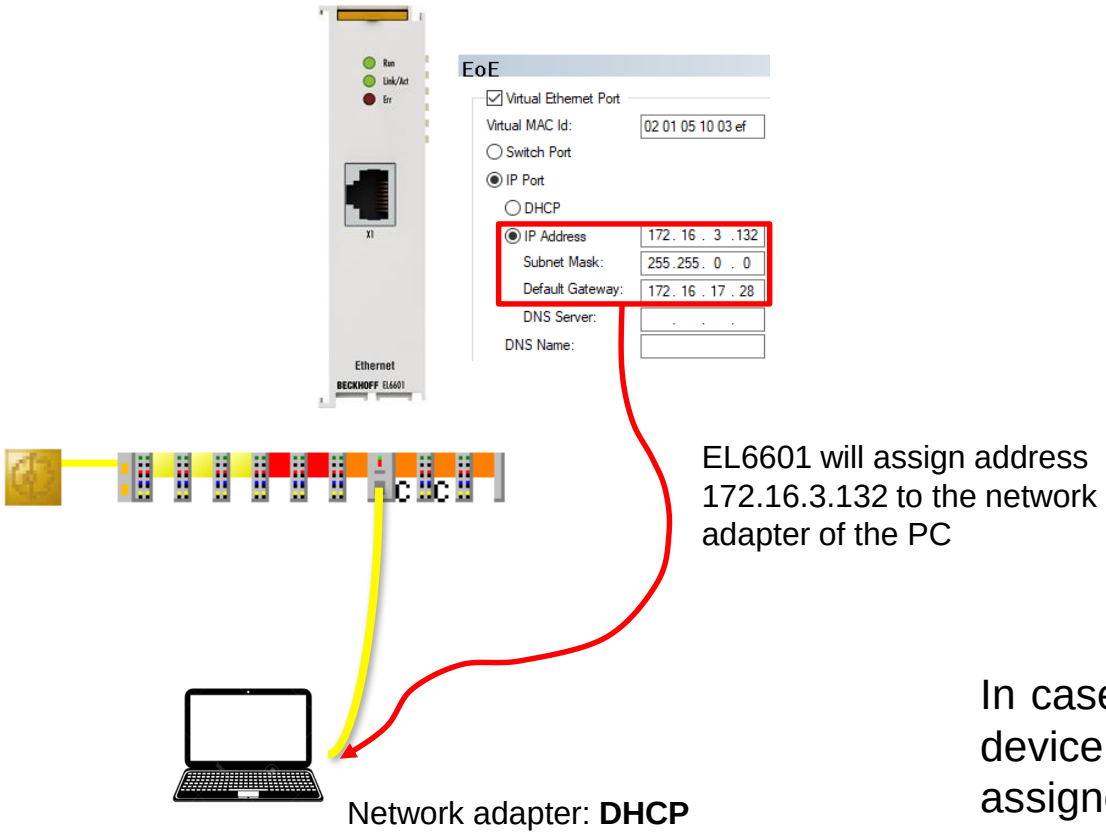


1. No IP address is set by the EtherCAT master during start-up (either the EoE device needs no IP address at all like it is the case for EL6601/EL6614, or the IP address is assigned to the EoE device through a separate interface).
2. The master sets the IP parameters of the Ethernet interface via EoE during start-up (in case this option is set for EL6601/EL6614, the terminal will act as DHCP server for an external device)
3. The IP interface of the slave is configured by the master as DHCP client during start-up, and expects to receive an IP address from a DHCP server connected to the subnet.

IP Address with EL6601/EL6614

BECKHOFF

Switchport terminals EL6601/EL6614 do not need a local IP address, as they work as switches. If an IP address is configured in their EoE settings, this will not be assigned to the EL6601/EL6614 itself: the EL6601/EL6614 will instead assign the IP address to an external Ethernet interface configured in DHCP mode.



EL6601 will assign address 172.16.3.132 to the network adapter of the PC

Network adapter: **DHCP**

In case of EL6614, only the first external device DHCP device connected will be assigned with the configured IP address.

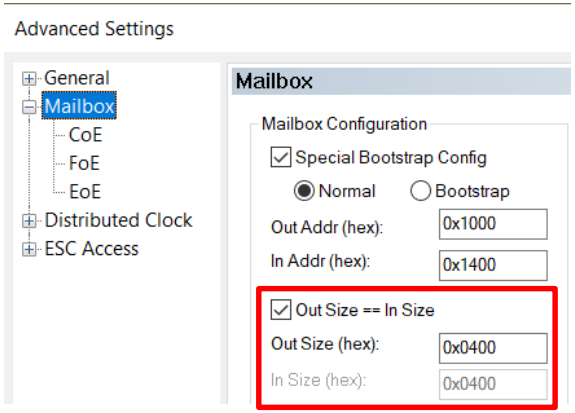
EoE Performances

BECKHOFF

Which factors have an influence on EoE performances?

1. Size of Mailbox buffers

   (the larger the better, compatibly with max. size supported by slave)



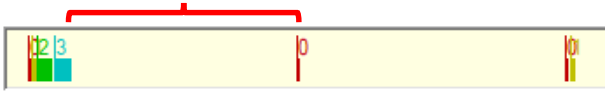
1. Cycle time of the fastest cyclic frame

   (the shorter the better)

Frame	Cmd	Addr	Len	WC	Sync Unit	Cycle (ms)
0	LRD	0x09000000	3			2.000
0	LWR	0x01000000	1	1	<default>	2.000
0	LRD	0x01000800	1	1	<default>	2.000
1	LRW	0x02000000	48	24	<default>	5.000
1	LWR	0x02000800	33	8	<default>	5.000
1	LRD	0x02001000	192	38	<default>	5.000
1	BRD	0x0000 0x0130	2	56		5.000

2. Free bandwidth between cyclic frames

   (the larger the better)



Typical **effective bandwidth** for EoE is **several hundreds kbyte/s up to 2-3 Mbyte/s**.

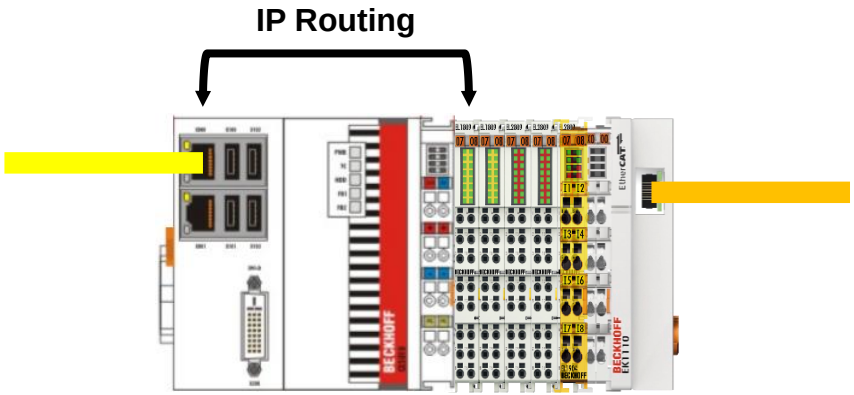
IP Routing

BECKHOFF

Only Ethernet end points belonging to the same subnet can exchange data directly.

- In case of Ethernet devices belonging to two or more different subnets (e.g. the EtherCAT segment and the plant LAN), communication will be possible only via a **router** device connected to both subnets and translating IP addresses in both directions (**IP Routing**).

In EoE applications, the router is typically the EtherCAT controller device provided with two or more independent network interfaces (one of them being the master device).

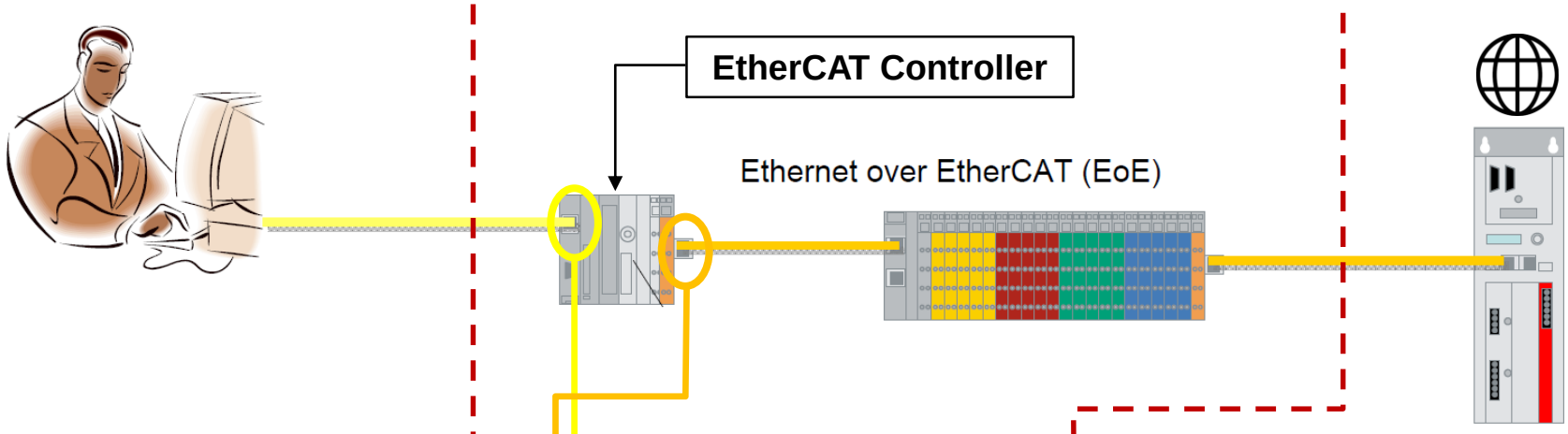


In order to work properly, IP routing shall be:

- Activated in the router device
- Configured in one or both Ethernet end points

How to Configure IP Routing?

BECKHOFF



- **Ethernet Port**

  IP Address : 172.16.8.31
  Mask : 255.255.0.0

- **Settings needed**

  Route Add
  ADD 192.168.1.0
  MASK 255.255.255.0
  172.16.4.205

- **Ethernet Port 1** (LAN)

  IP Address : 172.16.4.205
  Mask : 255.255.0.0

- **Ethernet Port 2** (ECAT)

  IP Address : 192.168.1.100
  Mask : 255.255.255.0

- **Settings needed**

  Enable IP Routing

- **EoE Address**

  IP Address : 192.168.1.11
  Mask : 255.255.255.0

- **Settings needed**

  (only needed if the end point is an Operating System; non necessary for EoE devices)

  Route Add
  ADD 172.16.0.0
  MASK 255.255.0.0
  192.168.1.100

The use of default Windows subnet **169.254.x.y** shall be **avoided**, as this class of IP addresses is not routed by the Operating System!

How to Configure IP Routing?

BECKHOFF

- **Enabling IP Routing in Windows (manually)**

  In system registry, select the following directory:

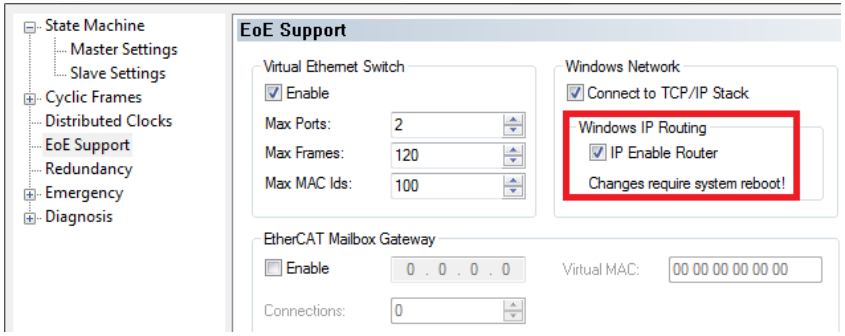
  [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters]

  Set the following key to 1:

  And then reboot the PC.

- **Enabling IP Routing in Windows (automatically).**

  Specific EtherCAT master platforms could provide higher-level settings to enable IP Routing (the Windows registry key will be set in background automatically).



How to Configure IP Routing?

BECKHOFF

- **Adding a route (Windows)**

  In command prompt, digit : ***ROUTE ADD <destination subnet> MASK <Destination subnet mask> <Gateway> -p***

```
C:\Windows\System32>ROUTE ADD 192.168.1.0 MASK 255.255.255.0 172.16.4.205 -p
```

- **Deleting a route (Windows)**

  In command prompt, digit : ***ROUTE DELETE <destination subnet> -p***

```
C:\Windows\System32>ROUTE DELETE 192.168.1.0 -p
```

- **Show active routes (Windows)**

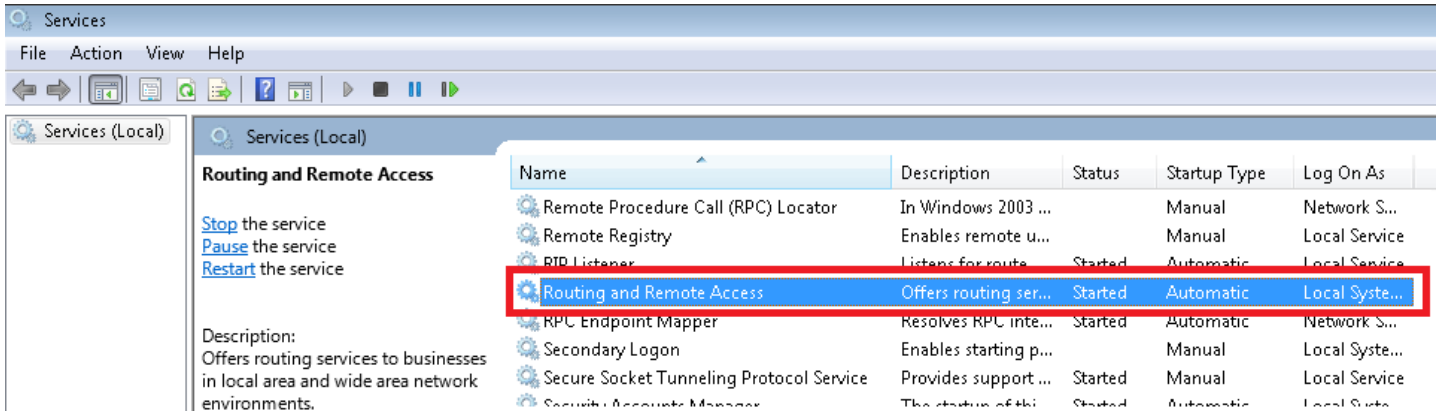
  In command prompt, digit : ***ROUTE PRINT***

```
C:\Windows\System32>ROUTE PRINT
```

In case of ≥ Windows 7, command prompt must be run as Administrator.

Other settings could be relevant in devices provided with Operating System (depending on the specific Operating System itself)…

1. In Control Panel → Administrative Tools → Services, set “Routing and Remote Access” on Automatic and Started:

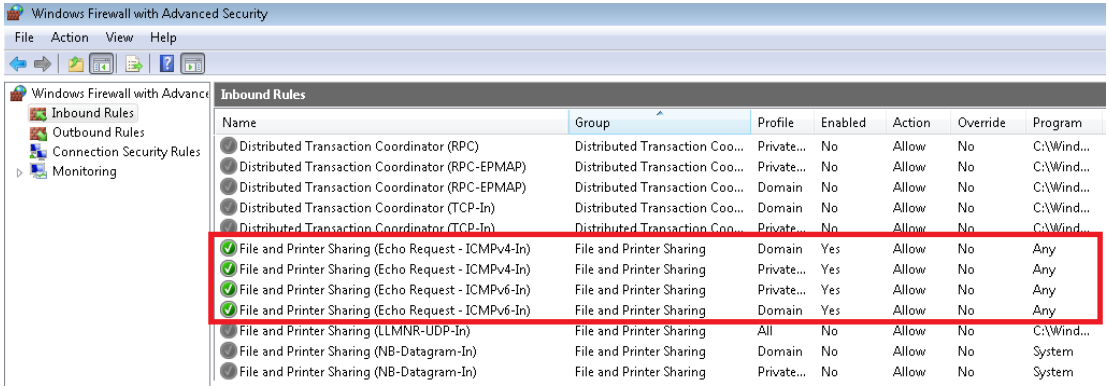


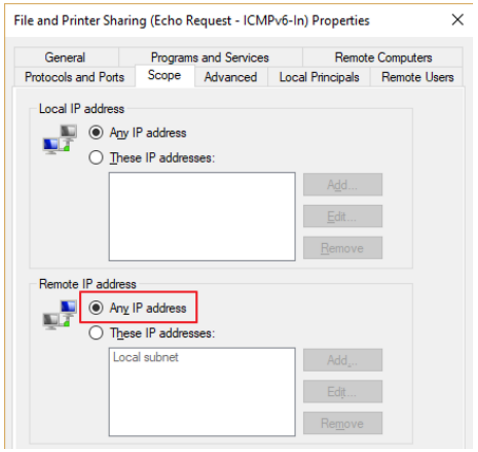
(This setting is required independently from the specific protocol used to access the end device)

How to Configure IP Routing?

BECKHOFF

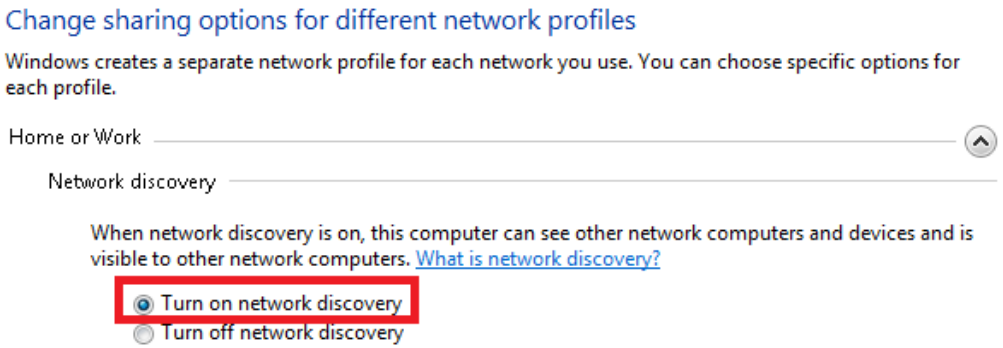
2. In Control Panel → Windows Firewall → Advanced Settings → Inbound Rules, enable Echo Request:





(In particular, this setting could be required if you try to **ping** the end device)

3. In Control Panel → Network and Sharing Center → Advanced sharing settings, enable network discovery:



(In particular, this setting could be required if you try to **ping** the end device)

Mailbox Gateway

BECKHOFF

The **Mailbox Gateway** is an optional functionality enabling to route Mailbox protocols from a client application (e.g. configuration or diagnostic tool) located in an external subnet to slave devices within an EtherCAT network over the master device.

- All Mailbox protocols can be routed : CoE, SoE, FoE, VoE (routing EoE via Mailbox Gateway makes no sense, as is already routed by the TCP/IP stack of the master.
- There Mailbox Gateway does not define error handling: requests to a non-existing slave device leads to no response, and automatic fragmentation is not supported.
- The Mailbox Gateway operates via UDP/IP and optionally via TCP/IP (TwinCAT supports only UDP/IP).