Master Run-time

White Paper

Revision date: 2023-03-28

Contents

1. Introduction

This white paper covers the issues related to the following points:

- EtherCAT technology benefits and the EtherCAT protocol advantages
- KPA EtherCAT Master Stack architectural, functional and implementation details
- Explanation of common appliances
- KPA EtherCAT Master Stack additional features

This document is also intended to simplify the process of getting started with KPA EtherCAT Master.

2. EtherCAT at a glance

Nowadays field bus systems have become an integral part of real-time distributed control. It is an effective way to increase the safety aspect of controlling and monitoring the production process.

2.1. EtherCAT - Ethernet Control Automation Technology

Although conventional fieldbus systems (PROFIBUS, CANopen, DeviceNet, SERCOS, etc.) provide more or less fast and safe data transfer, they are inferior to the EtherCAT technology in special considerations such as speed, overall productivity, reliability and real-time transmission.

EtherCAT is a good chance to eliminate the bottlenecks of conventional fieldbus systems allowing direct data exchange with the application by using shared memory without additional hardware since standard network adapters are enough.

EtherCAT (Ethernet for Control Automation technology) is a hard-real-time technology that realizes the specific transfer of data. It offers real-time performance and is aimed to maximize the utilization of high-speed full-duplex Ethernet data transfer through twisted pair or fiber optic cable for industrial process control needs.

EtherCAT based on the Ethernet technology possesses such advantages as the ease of implementation, cost of ownership and standardization, which makes it a perfect solution for industrial applications intended to maximize the performance of control systems.

Medium access control employs the Master/Slave principle, where the Master node (typically the control system) sends the Ethernet frames to the slave nodes, which extract data from and insert data into these frames on the fly.

2.2. Operating principle

An EtherCAT segment is a single Ethernet device from an Ethernet point of view, which receives and sends standard ISO/IEC 802-3 Ethernet frames. This Ethernet device may consist of a large number of EtherCAT slave devices, which process the incoming frames directly and extract the relevant user data, or insert data and transfer the frame to the next EtherCAT slave device. The last EtherCAT slave device within the segment sends the fully processed frame back, so that it is returned by the first slave device to the Master as a response frame.

This procedure utilizes the full duplex mode of Ethernet, which allows for independent communication in both directions. Direct communication without a switch between a Master device and an EtherCAT segment consisting of one or several slave devices may be established. This demonstrates the flexibility of the EtherCAT operating principle.

2.3. Core features of EtherCAT

The **core features** of EtherCAT should be emphasized:

- **EtherCAT** is considered to be the fastest industrial Ethernet technology considering the way the protocol is designed in order to fully utilize the advantages of industrial Ethernet
- **High performance** Obviously, it is a huge boost to your productivity as EtherCAT is designed to achieve high performance due to the flexibility of its topology, EtherCAT protocol specific features and the way of data mapping. For example, with direct memory access (DMA) data can be transferred between the network card and the Master processor or slave I/O with minimal CPU

usage. Slaves write and read data themselves and there is only one telegram passing through all the slaves, which is returned to the Master processed. This reduces the complexity of the Master and frees its resources

- **Low cost technology** EtherCAT is a real-time industrial Ethernet technology that does not demand a special plug-in card for the Master, co-processor or a lot of processing power at all. A standard Ethernet connection without hubs and often even switches is the only necessity
- Very low-cost slave controller chips are in the price segment of standard Ethernet chips. It assures the lowest hardware costs on Master side and the smallest expenditures for installation
- **Flexible Ethernet topologies** EtherCAT network can support up to 65,535 devices without placing restrictions on their topology:
  - Star
  - Line
  - Tree
  - Droplines
  - Ring
- **Integration**
  - with fieldbuses and networks through gateways:
    - CAN/CANopen
    - Ethernet
    - PROFIBUS
    - SERCOS
  - with third-party tools:
    - Mailbox over UDP
    - Mailbox over TCP
- **Openness of technology**
  - open EtherCAT Master specifications
  - hundreds of members worldwide with expertise in manufacturing of slaves and chips, development of software and firmware

To realize solutions based on EtherCAT, we have developed such products as KPA EtherCAT Master and KPA EtherCAT Studio:

- KPA EtherCAT Master to provide interaction of network devices with IPC systems
- KPA EtherCAT Studio to create and modify EtherCAT network configurations

3. Why KPA EtherCAT Master?

KPA EtherCAT Master Stack ensures all the advantages of the EtherCAT technology:

- minimum cycle time
- high performance
- minimum expenses

Its modular architecture provides portability to different operating systems and scalability by Basic, Standard and Feature/Extension Packages. On top of that the Master can be adapted to various hardware platforms.

The KPA EtherCAT Master Stack was designed to fulfill the following requirements:

- Specific optimization to run on embedded real-time operating systems
- Implementation in ANSI “C” language. The source code of the EtherCAT Master Stack can be easily ported to other operating systems by means of “C”
- Modular architecture
- Development by using the EtherCAT specification only
- Operating system independence as far as possible
- Small footprint
- High performance, minimum CPU time
- Intel, Motorola (Power PC) and ARM platforms support

The KPA EtherCAT Master Stack core is implemented in ANSI “C”; therefore, it can be easily ported to any platform which has a “C” compiler. It may run even in Linux kernel space as a module (in contrast to C implementations, which may encounter difficulties while loading the kernel module if the C code uses any of such features as virtual functions, templates or exceptions).

4. KPA EtherCAT Master stack functions and classes

4.1. KPA EtherCAT Master stack functions

- Fast process and diagnostic data exchange from EtherCAT Master stack to application
- Central parameterization of intelligent slaves via UDP
- Adaptation to operating system by special interface
- API for interaction with runtime or/and configuration tools
- Servers for multiple TCP/IP or/and UDP connections

4.2. KPA EtherCAT Master stack classes

The following KPA EtherCAT Master Stack software packages are available:

1. Basic - Class B according to ETG.1500: “EtherCAT Master Classes” specification
2. Standard - Class A according to ETG.1500: “EtherCAT Master Classes” specification
3. Premium
4. Extension Packages

4.2.1. Basic class

Requested by ETG Basic Master Definition ETG.1500

- Support of all commands and EtherCAT frames
- Support of EtherCAT State Machine (ESM)
- Checking of network or slaves errors
- Cyclic process data exchange
- Online scanning
- Network configuration taken from ENI
- Polling of Mailbox State of slave
- Explicit device ID
- Write access to EEPROM
- Statistics gathered from NIC and Master
- CoE SDO Info Service
- FoE (File Access over EtherCAT)
- Distributed clocks (DC)
  - DC support
  - Time distribution (Slaves synchronization)
- Slave-to-slave Communication

4.2.2. Standard class

In additional to Basic class features this class includes:

- Distributed clocks (DC)
  - Synchronization of Master and slaves with continuous delay compensation
  - Sync window monitoring
- Servo drive profile over EtherCAT (SoE)
- Ethernet over EtherCAT (EoE)
- Vendor specific protocol over EtherCAT (VoE)
- ADS over EtherCAT (AoE)

4.2.3. Premium class

In additional to Standard class features this class includes:

- Hot-Connect
- Cable Redundancy (ensures access to all slaves even in case of one-line break or one damaged device)
- TCP or UDP/ IP Mailbox Gateway for configuring devices via EtherCAT

All these features can be included into other classes additionally.

4.2.4. Extension Packs

- External synchronization for two or more EtherCAT buses with an external reference clock via Master
- Data- and Frame-Logger for monitoring traffic with triggering without external tools (included in Premium class)
- Access Rights for different users of Master (included in Premium class)
- CAN DBC driver
- Event Handler (included in Premium class)
- Multimaster
- VCOM driver
- PI snapshot (included in Premium class)
- Optimized drivers and HW Extensions
- PI logger
- Online configuration
- Master Redundancy
- Hardware timed send
- Any future extensions

4.3. Trial mode limitation

KPA EtherCAT Master in the trial mode is full-featured KPA EtherCAT Master with 60 minutes (1 hour) time limitation for operating and a trial license applied. When the trial time is expiered, the Master will be switched to INIT state and all I/O operations will be interrupted. To continue evaluating the Master, restart your application.

5. Master architecture

The core of the KPA EtherCAT Master can be divided into the following logical parts:

- User Application Programming Interface (API) which allows to configure and manage EtherCAT bus
- Master initialization and configuration
  - Process image handling
  - Mailbox protocols handling
  - Master/slave state control
  - Network state (slaves scanning)
  - Master statistics, diagnostics
  - Traces/error handling
- Process Image
- Mailbox implementation
- Master threads (PI update cycle (HItime-critical cycle), Mailbox update cycle (NR-mailbox processing cycle), Autorecovery and diagnostics cycle (LO-auto-recovery, diagnostics cycle))
- Frame scheduler
- OS abstraction layer
  - EtherCAT network driver (API functions: ecat_open, ecat_close, ecat_is_opened, ecat_send_frame, ecat_recv_frame, ecat_get_statistics, ecat_reset_statistics, ecat_get_link_state, ecat_set_callback, ecat_is_available, ecat_get_adapterslist. Driver is closely bonded to OS dependent network stack or network card driver, i.e. underlying network layer)
  - Wrappers for thread handling functions, process synchronization functions (mutexes, semaphores, etc.), time handling functions

The KPA EtherCAT Master architecture with the bundled modules is represented as depicted below.

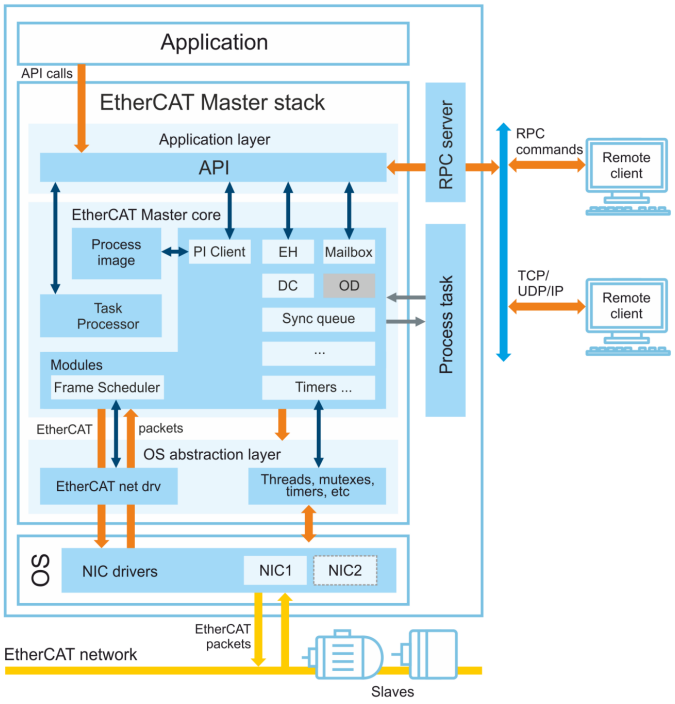

Figure 1. KPA EtherCAT Master stack architecture

Items description:

Application - this is a separate process which invokes Master API functions. The application performs almost full control over Master by means of API: starts/stops the Master, configures it, updates PI, etc.

API - user Application Programming Interface (API) which allows to configure and manage EtherCAT bus.

RPC-server - part of KPA EtherCAT Master Stack. The RPC-server is responsible for establishing connection with a remote client (for example, with KPA EtherCAT Studio) and forwarding requests to the EtherCAT Master Stack core. It supports user-to-user and user-to-kernel modes (through IOCtl calls) modes.

Process Image - Process Image consists of Shadow buffer (i.e. data transferred directly to/from the network) and Active buffer (i.e. buffer which contains data received/sent from/to clients of KPA EtherCAT Master).

Process task (external task) - callback function which implements application-specific control algorithm. This callback function is called at every repetition of PI update (high-priority) cycle.

Frame scheduler - module which assembles EtherCAT frames and forwards them to the EtherCAT network driver according to their priorities.

EtherCAT net dry (EtherCAT network driver) - module which abstracts the EtherCAT Master Stack core from the underlying network implementation.

Threads, mutexes, timers, etc. - wrappers for OS-dependent functions that deal with threads, timers, mutexes.

NIC drivers - network interface card drivers.

NIC1, NIC2 - network interface card is a physical device which sends/receives frames to/from the EtherCAT network.

Remote Client - KPA EtherCAT Studio or another compatible application.

5.1. Common operation modes

There are tree common modes supported by the Master:

1. **Asynchronous mode**

Application task (cycle), which implements control algorithms, and a Master cycle, which updates Process Image, run independently. The whole data exchange is done through Process Image.

HI = PI update cycle

NR = Mailbox cycle

LO = Diagnostics cycle

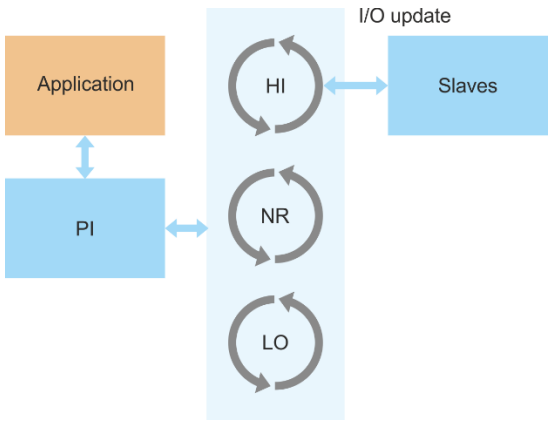

Figure 2. Asynchronous mode

2. **Synchronous 1 mode**

There is no separate application cycle in contrast to Asynchronous mode. All control algorithms are implemented in a Process Task callback function which is called at every Master cycle.

HI = PI update cycle

NR = Mailbox cycle

LO = Diagnostics cycle

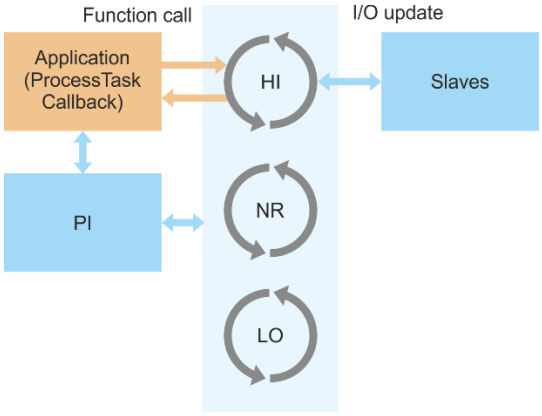

Figure 3. Synchronous 1 mode

3. **Synchronous 2 mode**

There is no separate Master cycle for cyclic data exchange. An application initiates data exchange with EtherCAT slaves.

HI = PI update cycle

NR = Mailbox cycle

LO = Diagnostics cycle

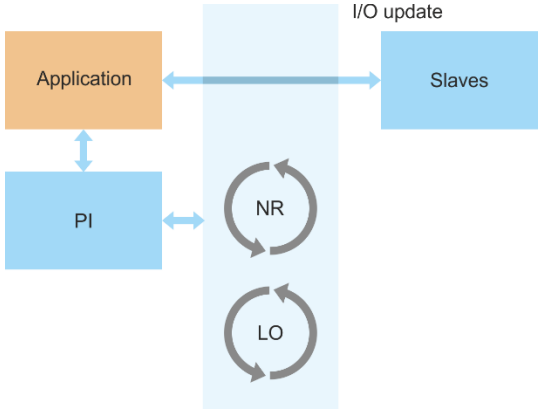

Figure 4. Synchronous 2 mode

6. KPA EtherCAT Master getting started

6.1. Before starting

It is strongly recommended to **use only dynamic library linkage** to avoid library initialization problems. In other cases, **user application has to call EcatMkpaInit (NULL) routine** before any Master API call and `EcatMkpaDestroy()` at final stage.

This way, it is necessary to load Master Library (ecatmkpa.dll for Windows, ecatmkpa.rsl for INtime, RTXEcatKPAMaster.rtdll for RTX, etc.) before using Master API. For example, for Windows you may load the Library by calling LoadLibrary function.

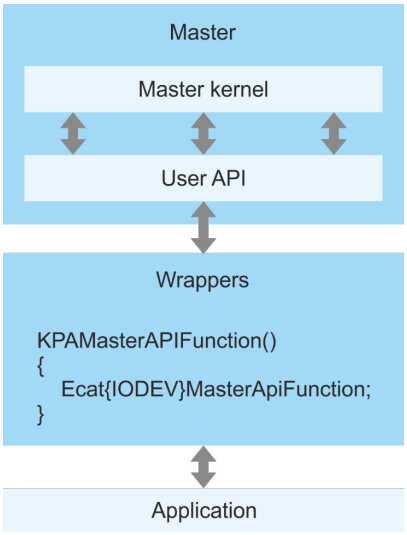

Figure 5. LoadLibrary Function

On the other hand, the usage of wrappers makes your application more platform-independent and allows simplifying the process of adapting your application to different OS.

6.2. Master initialization

It requires configuring Master once during user application start. Master initialization procedure includes the following steps:

1. EcatMkpaInit(NULL)
2. Get a list of available Network interfaces by calling EcatGetAdapterList function
3. Create unique identifier for Master that will be used to reference Master later (EcatCreateMaster)
4. Connect chosen Network Interface to created Master, preliminary specifying expected operating mode (EcatConnectMaster)
5. Operating mode without redundancy (using only one Network interface)
6. Operating mode with redundancy (using two Network interfaces)
7. Load ENI file (EcatLoadConfigFromString)
8. Initiate cyclic data exchange by calling EcatStartCyclicOperation
9. Set time for auxiliary tasks (scanning of number of slaves on the bus and others) by calling EcatSetAutoRecoveryTimeout

6.3. Process Image

Master Process Image may be conditionally divided into two buffers: active and shadow. This scheme is realized for additional data protection to avoid different kinds of collisions while simultaneous access to data by Master and user application.

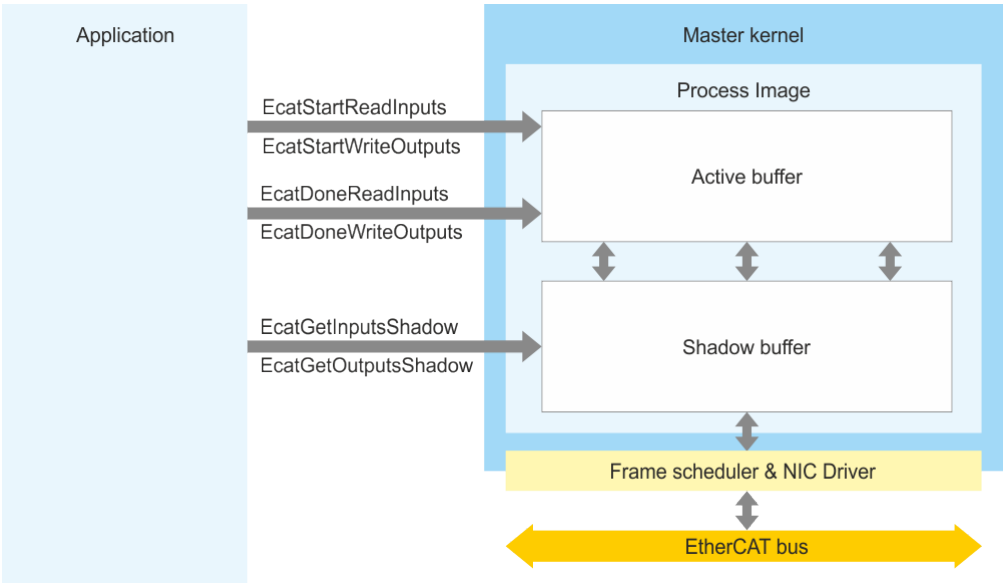

Figure 6. Scheme of data protection

Active buffer serves for work with the user (e.g. preparation of data for sending to the bus).

Shadow buffer is a buffer which is used by Master directly to send data to the bus in case of outputs and to receive actual values for inputs.

Work with shadow buffer is a little bit faster than with active buffer. But direct work with shadow buffer is possible only in synchronous master's operation modes (when it is granted that only user application accesses Process Image). Work with shadow buffer imposes additional obligations on the end-user in case of possible use of a few clients (a few clients try to get the data in the same time). This way, shadow buffer may be used only in synchronous mode and when only one client instance is realized.

In case of using different HMI (for example any SCADA, PLC Configurators, even KPA EtherCAT Studio) as additional component of user application, pay attention that within HMI the request to PI (call of `EcatStartReadInputs/EcatStartWriteOutputs`) is not implemented. Otherwise, it may lead to uncertainty during a call to shadow buffer. For example, if controlling application (kernel of user application: Control Algorithm) calls shadow buffer directly, and in the same time HMI calls active buffer, shadow buffer may be rewritten (from HMI side or from the application side). This way it becomes incomprehensible which data is sent to shadow buffer and then to the bus (HMI or controlling kernel).

Generally, **it is recommended to use active buffer to work with Process Image** (PI).

6.4. Implementation of Master's modes

Asynchronous mode

In Asynchronous operating mode Master (not user application) handles the Master cycle. In other words, internal cycle of user's application and Master's cycle are not synchronized. Master initiates data update in accord with set frequency (during Master initialization). User's application works with its frequency.

Final user's application only operates with active buffer of PI whose obtaining is initiated by call of

`EcatStartReadInputs` / `EcatStartWriteOutputs` functions. Further operating with the obtained buffer may be realized with help of `EcatGetVariable` and/or `EcatSetVariable` functions.

General scheme of Asynchronous mode may be represented as depicted below.

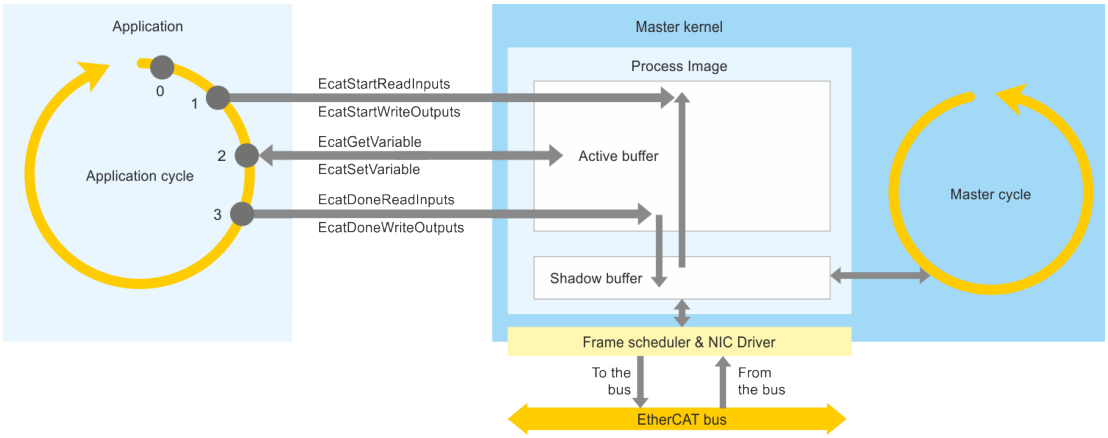

Figure 7. Asynchronous operating mod

0 point: start of user application.

1 point: user application initiates obtaining input/output PI of Master (call of `EcatStartReadInputs` / `EcatStartWriteOutputs` respectively); Master copies shadow buffer's content to active buffer and passes it to user application.

2 point: user application takes values of certain inputs and outputs from the obtained buffer (`EcatGetVariable`), analyses them and generates a new action to control the system (set certain output signals by calling `EcatSetVariable`).

3 point: user application acknowledges completion of operating with data (call of `EcatDoneReadInputs` / `EcatDoneWriteOutputs` respectively); Master copies the active buffer to the shadow buffer (see description in Section 6.3 chapter). This data will be sent to the line during the next Master cycle because cycles of Master and user application are not synchronized.

Timing chart of Asynchronous mode of Master's operate:

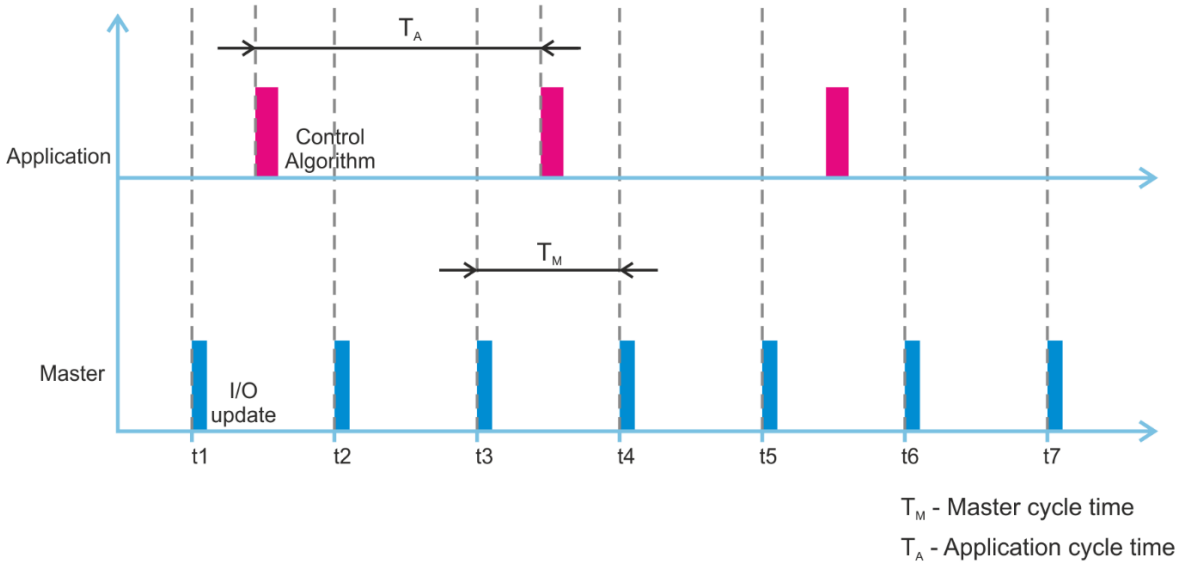

Figure 8. Timing chart of Asynchronous mode

As you can see in the picture, data updating time of Master and user application are not synchronized. Master initiates data receiving every t_i moment (where $t_i = t_{i-1} + T_M$ for $i = \overline{2,7}$). But the application executes control algorithm with its cycle T_A not connected to Master cycle.

Should be noted that Master sends commands to update input/output Process Data according to structure described in KPA EtherCAT Studio within a frame (if ProcessData + all headers and checksums ≤ 1514byte) or a few frames generally.

To initialize Asynchronous mode, it is necessary to pass nonzero value of PDO cycle during initialization of cyclic updated.

Synchronous 1 mode

As well as in Asynchronous mode, in Synchronous 1 mode, Master itself initiates data updating (with a frequency set on start-up). But in this mode, there is no particular cycle for the application. All actions of Control Algorithm are executed according to Process Task (callback function), that is called every Master cycle. In other words, user application works synchronously with Master cycle and Master frequency.

A general scheme of Synchronous 1 mode may be represented as depicted below.

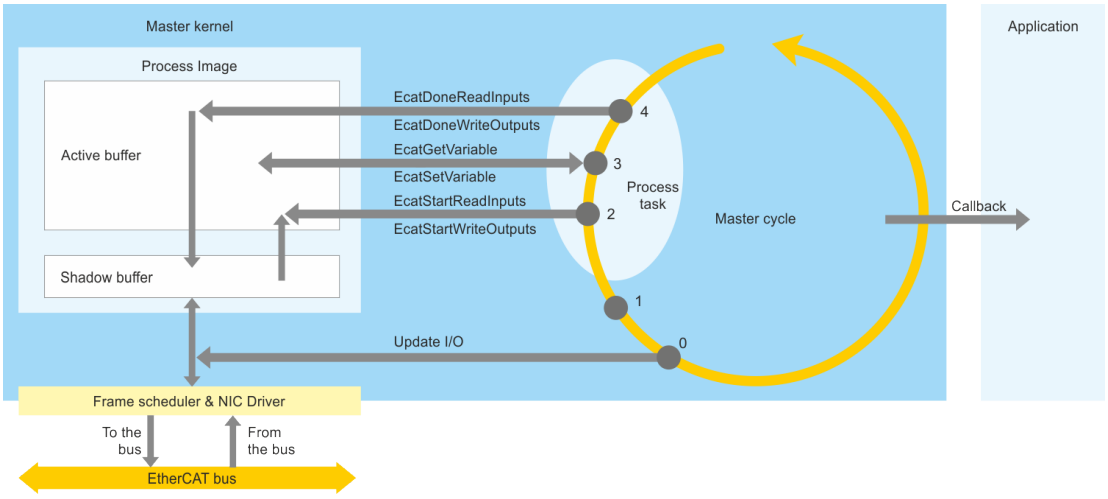

Figure 9. Synchronous mode

0 point: Master initiates data updating (sends commands to the bus in order to get actual inputs and set actual outputs).

1 point: Master receives data from the bus. Since this moment, the latest actual values of Process Data are stored in the shadow buffer.

2 point: Process Task initiates getting input/output active PI buffer from Master (call of `EcatStartReadInputs / EcatStartWriteOutputs` respectively); Master copies shadow buffer's content to active buffer and passes it to user application.

3 point: process task takes values of certain inputs and outputs (`EcatGetVariable`) from obtained buffer, analyses these values and generates a new action to control system (set certain output signals by calling `EcatSetVariable`).

4 point: process task acknowledge completion of operating with data (call of `EcatDoneReadInputs / EcatDoneWriteOutputs` respectively); Master copies the active buffer to the shadow buffer (see description in Section 6.3 chapter). This data will be sent to the line during the next Master cycle, because cycles of Master and user application are not synchronized.

Timing chart of Synchronous 1 mode of Master's operation:

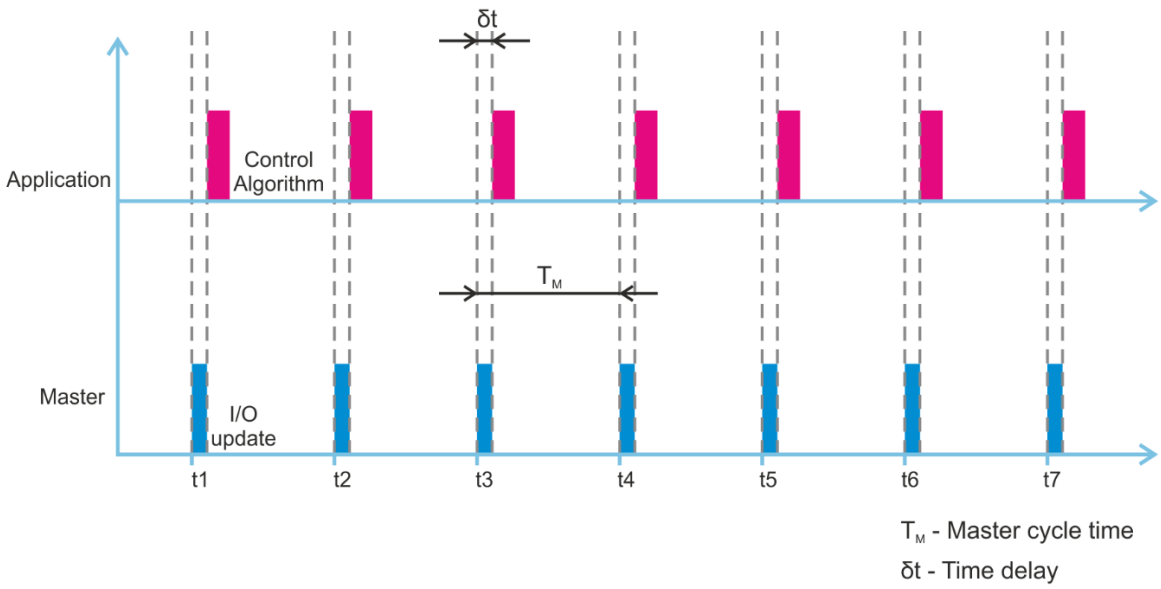

Figure 10. Timing chart of Synchronous 1 mode

Master initiates Process Data updating in t_i moment (where $t_i = t_{i-1} + T_M$ for $i = \overline{2,7}$). Process task (control algorithm of application) execution will be delayed for *δt* (its size depends on physical characteristics of EtherCAT bus and performance of target system where Master is running).

It should be noted that Master sends commands to update input/output Process Data according to the structure described in KPA EtherCAT Studio within a frame (if ProcessData + all headers and checksums ≤ 1514byte) or a few frames generally.

To initialize this Synchronous 1 mode it is necessary to pass nonzero value of PDO cycle during initialization of cyclic update (call `EcatStartCyclicUpadte`) and set handler of user task (Process Task) that should be executed upon each completion of Process Image data update. Thus, function `EcatSetExtCtrlTaskHandler` should be called.

Synchronous 2 mode

In Synchronous 2 mode user application controls Master cycle (Process Data update). Master is represented as I/O driver between user application and EtherCAT slaves.

In this mode, Master's operation may be realized in two ways:

1. Synchronous 2a mode: inputs and outputs are updated in the same time by calling EcatUpdateProcessImage
2. Synchronous 2b mode: inputs and outputs are updated separately (in separate frames) by calling EcatUpdateProcessImageInputs and EcatUpdateProcessImageOutputs

Should be noted that in Synchronous 2 mode Master cycle is a timing difference between two sequential calls of EcatUpdateProcessImage function (in case of simultaneous updating of inputs and outputs) or EcatUpdateProcessImageInputs function (in case of separate updating of inputs and outputs). Please, keep that in mind while using Distributed Clock functionality in which calling of the mentioned functions is used as a synchronous event for Master (synchronization of master and slaves with continuous delay compensation).

NOTE Also, note that in case of using the line, which does not have inputs (only outputs are available), to update data, call EcatUpdateProcessImageInputs function cyclically. Otherwise, Master will not send any commands to the bus.

Consider timing charts of a few possible variants of Synchronous 2 mode realization in case of simultaneous updating of inputs and outputs by calling EcatUpdateProcessImage (generally, more variants of realization are possible because end realization of control program logic depends on physical peculiarities of controlled line):

1. Data updating is executed at the end of Control Algorithm execution.

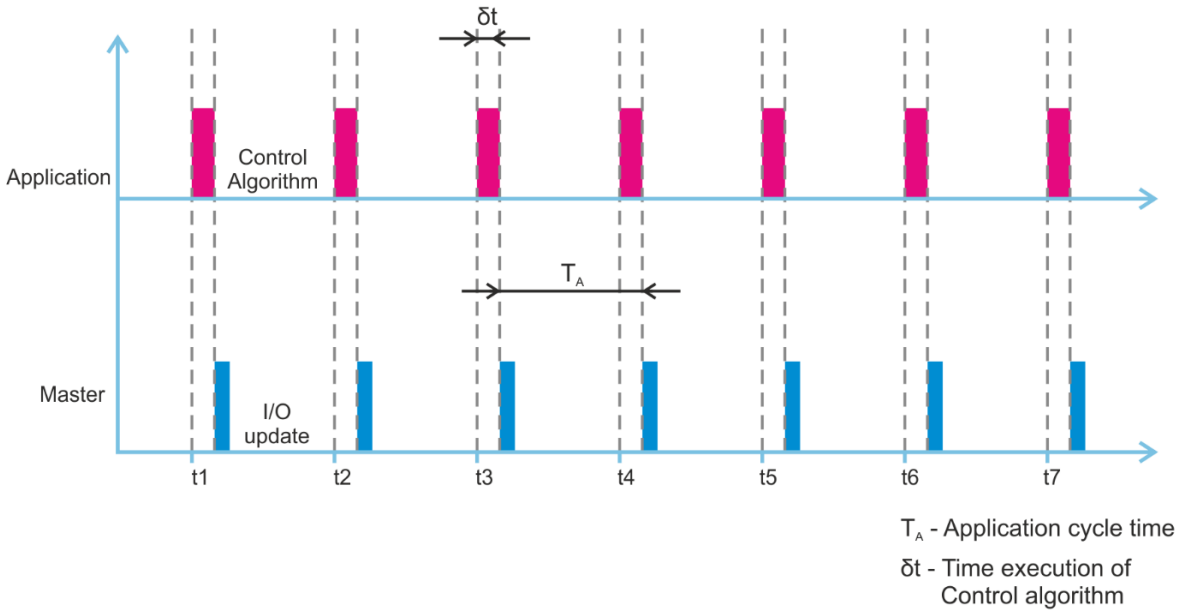

Figure 11. Data updating at the end of Control Algorithm execution

In this chart Master cycle is connected to time of Control Algorithm execution. This cycle is unstable because the time of Control Algorithm execution may fluctuate ($\delta t \pm dt$, where *dt* - some timing delay).

2. Data updating is executed at the beginning of Control Algorithm execution.

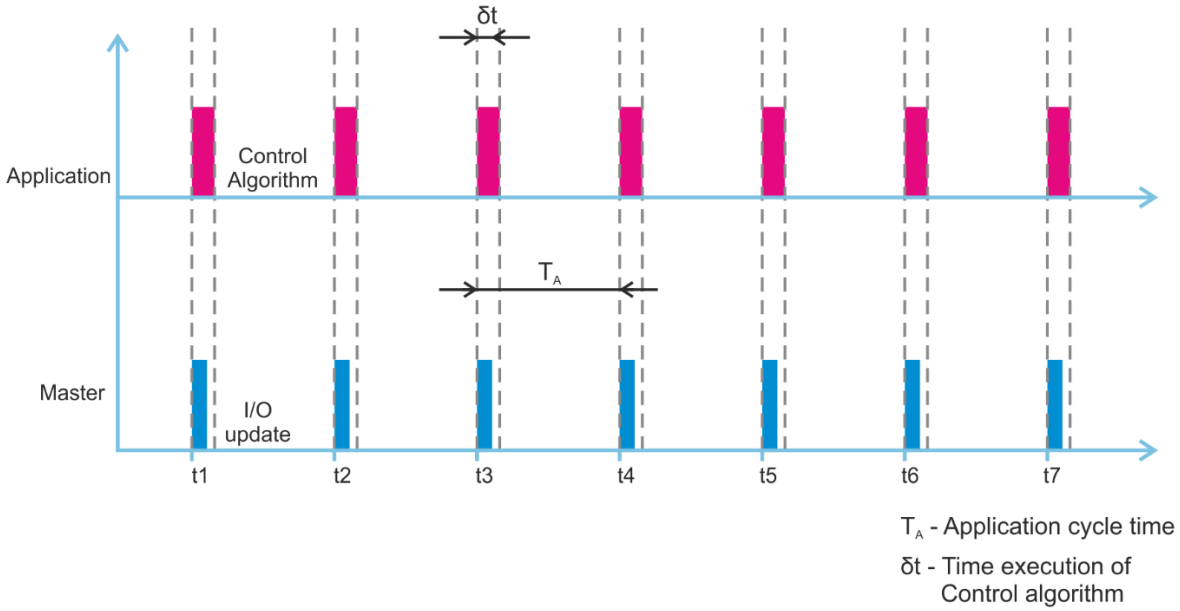

Figure 12. Data updating at the beginning of Control Algorithm execution

This chart is more appropriate, because the control program will always work with actual data that Master just has taken from the bus, and the cycle of user application is more independent of time of Control Algorithm execution.

Keep in mind the considered peculiarities during development of control program in Synchronous 2 mode, because any different kinds of time fluctuating may negatively affect system behavior as in case of using Distributed Clock so and without it.

Scheme of Synchronous 2a mode

A general scheme of Synchronous 2a mode may be represented as depicted below.

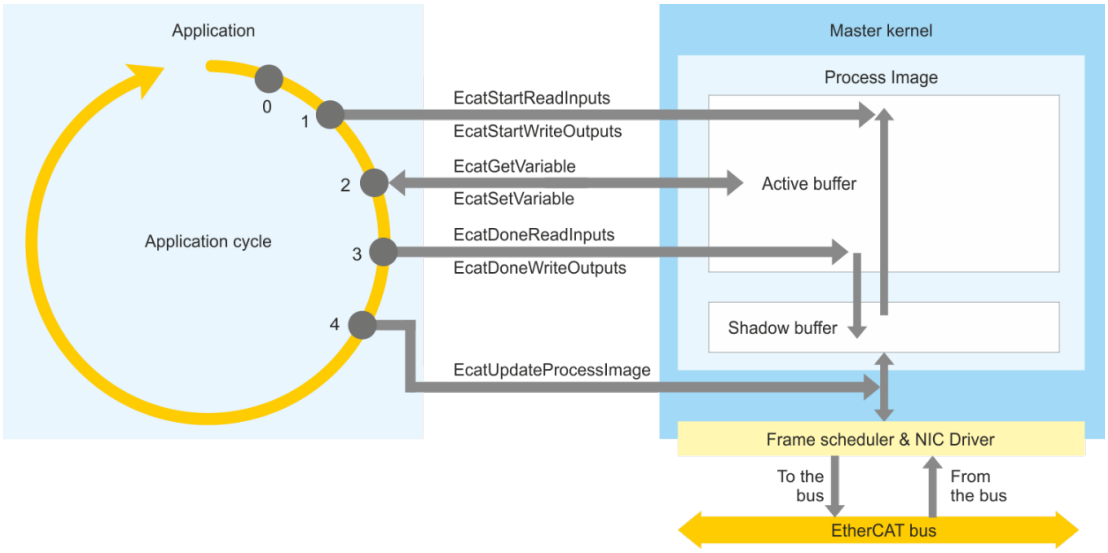

Figure 13. Scheme of Synchronous 2a mode

0 point: start of user application.

1 point: user application initiates obtaining input/output PI of Master (call of `EcatStartReadInputs` / `EcatStartWriteOutputs` respectively); Master copies shadow buffer's content to active buffer and passes it to user application.

2 point: user application takes values of certain inputs and outputs from the obtained buffer (`EcatGetVariable`), analyzes them and generates a new action to control the system (set certain output signals by calling `EcatSetVariable`).

3 point: user application acknowledges completion of operating with data (call of `EcatDoneReadInputs` / `EcatDoneWriteOutputs` respectively); Master copies the active buffer to the shadow buffer (see description of work with active PI in Section 6.3 chapter).

4 point: user application initiates Process Data updating by calling `EcatUpdateProcessImage`.` Master generates cyclic commands on the basis of the shadow buffer and sends it to the line.

Scheme of Synchronous 2b mode

A general scheme of Synchronous 2b mode may be represented as depicted below.

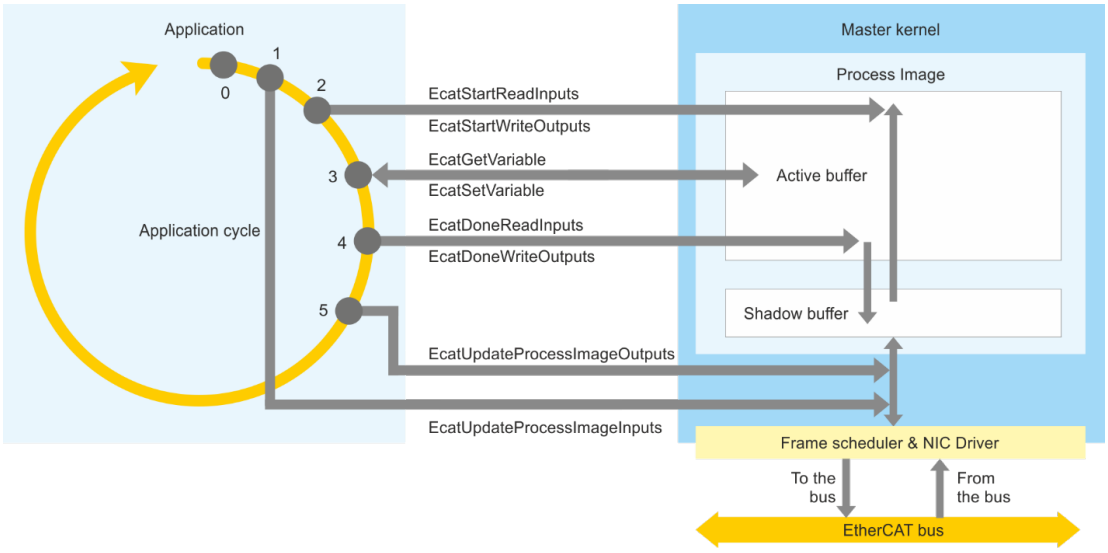

Figure 14. Scheme of Synchronous 2b mode

0 point: start of user application.

1 point: user application initiates update of input Process Data (call of EcatUpdateProcessImageInputs); Master generates the corresponding frame and updates the shadow buffer with new input data.

2 point: user application initiates obtaining input/output PI of Master (call of` EcatStartReadInputs / EcatStartWriteOutputs` respectively); Master copies shadow buffer's content to active buffer and passes it to user application.

3 point: user application takes values of certain inputs and outputs from the obtained buffer (`EcatGetVariable`), analyzes them and generates a new action to control the system (set certain output signals by calling `EcatSetVariable`).

4 point: user application acknowledges completion of operating with data (call of `EcatDoneReadInputs / EcatDoneWriteOutputs` respectively); Master copies the active buffer to the shadow buffer (see description in Section 6.3 chapter).

5 point: user application initiates output Process Data updating by calling `EcatUpdateProcessImageOutputs`. Master generates cyclic commands on the basis of the shadow buffer and sends it to the line.

Initialization of Synchronous 2 mode

To initialize this operating mode, it is necessary to pass **zero value** of PDO cycle during initialization of cyclic updated.

Also, in case of Distributed Clock functionality usage, time of PDO update by user application (e.g. by PLC) should be passed to Master by calling `ExatSetExternalCycleTime` function.

6.5. Shutting down Master

To complete Master's operation, do the following:

1. Stop cyclic data update by calling EcatStopCyclicOperation function
2. Release used Network interfaces by calling EcatDisconnectMaster
3. Free Master resource by calling EcatFreeMaster
4. Release used Master's resources (used Master's libraries)

7. Software packages

Due to support of wide range of platforms different software packages are available.

At present the set of platforms supported by the KPA EtherCAT Master Stack includes:

- Windows
- INtime
- RTX64
- Linux
- Xenomai
- QNX 6.x
- QNX 7.x
- VxWorks
- FreeRTOS
- ITRON
- Intewell

Any other operating system can be supported upon customer's request. In addition, we have ported KPA EtherCAT Master to Windows XP, CE6/7, OnTime RTOS-32, PikeOS, RTAI, etc. but we do not actively support these operating systems.

8. KPA EtherCAT Master is proven

This paper has discussed the KPA EtherCAT Master's architecture, distinctive features and core functions and revealed the advantages of the KPA EtherCAT Master implementation.

With the KPA EtherCAT Master you are able to easily make the process of data exchange fast, secure, and effective. You will considerably reduce expenses as no special Master plug-in cards are required but a standard Ethernet card can be used. The platform independent source code allows making the Master portable to any operating system with fewer efforts. Another crucial factor is the adaptation of Master to any hardware platform implementing only the network adapter driver logic. The architecture provides the Master scalability to fit the size of application, development and customization of each part keeping the functionality of the others untouched.

The KPA EtherCAT Master provides you with the latest approach to enhancing the whole process of data exchange with the network.

For more detailed information on KPE EtherCAT Master, please, refer to documentation that is provided within the Master Development Kit package.

9. KPA EtherCAT Master API usage

9.1. Initializing

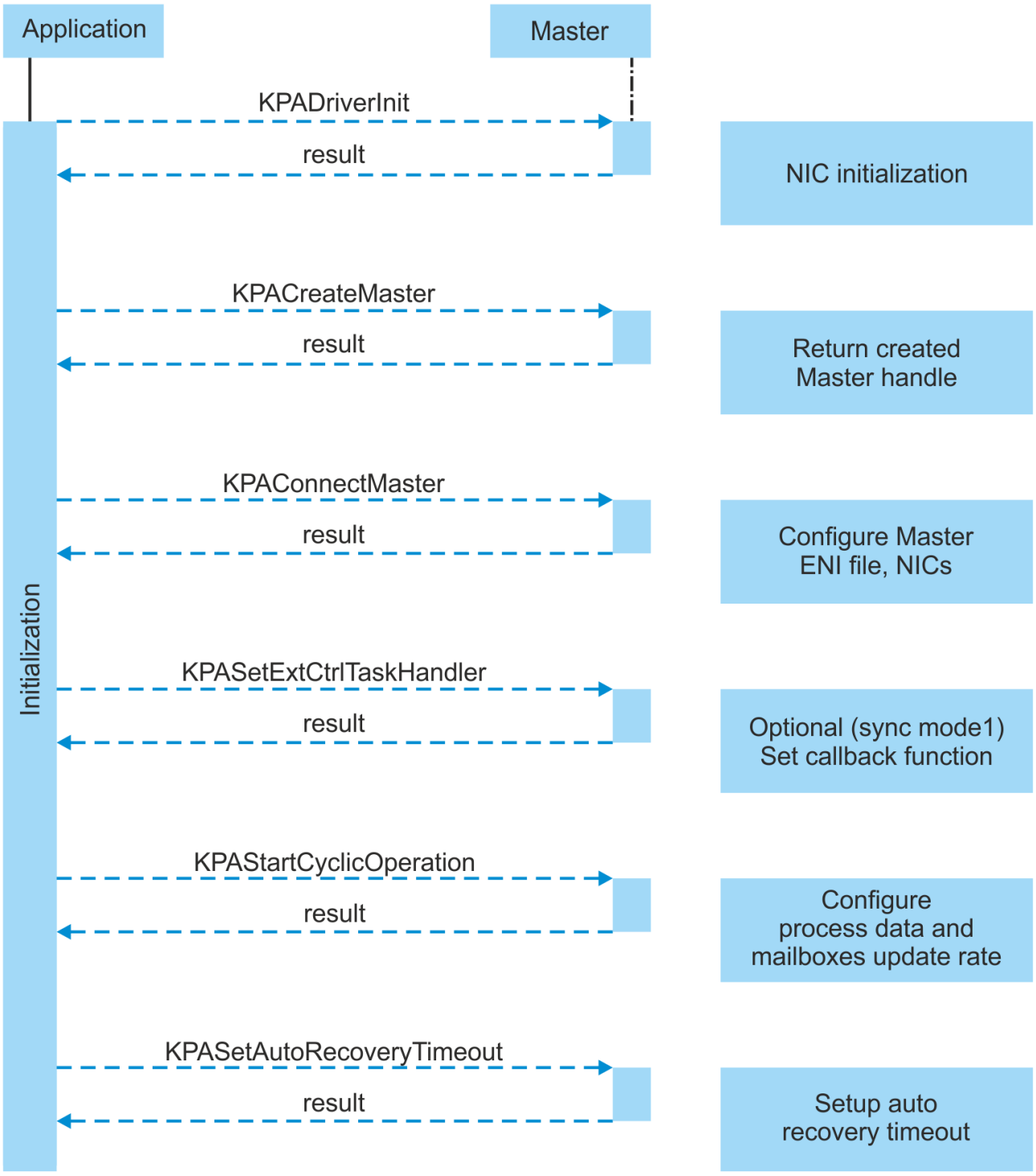

Figure 15. Initializing

9.2. Releasing

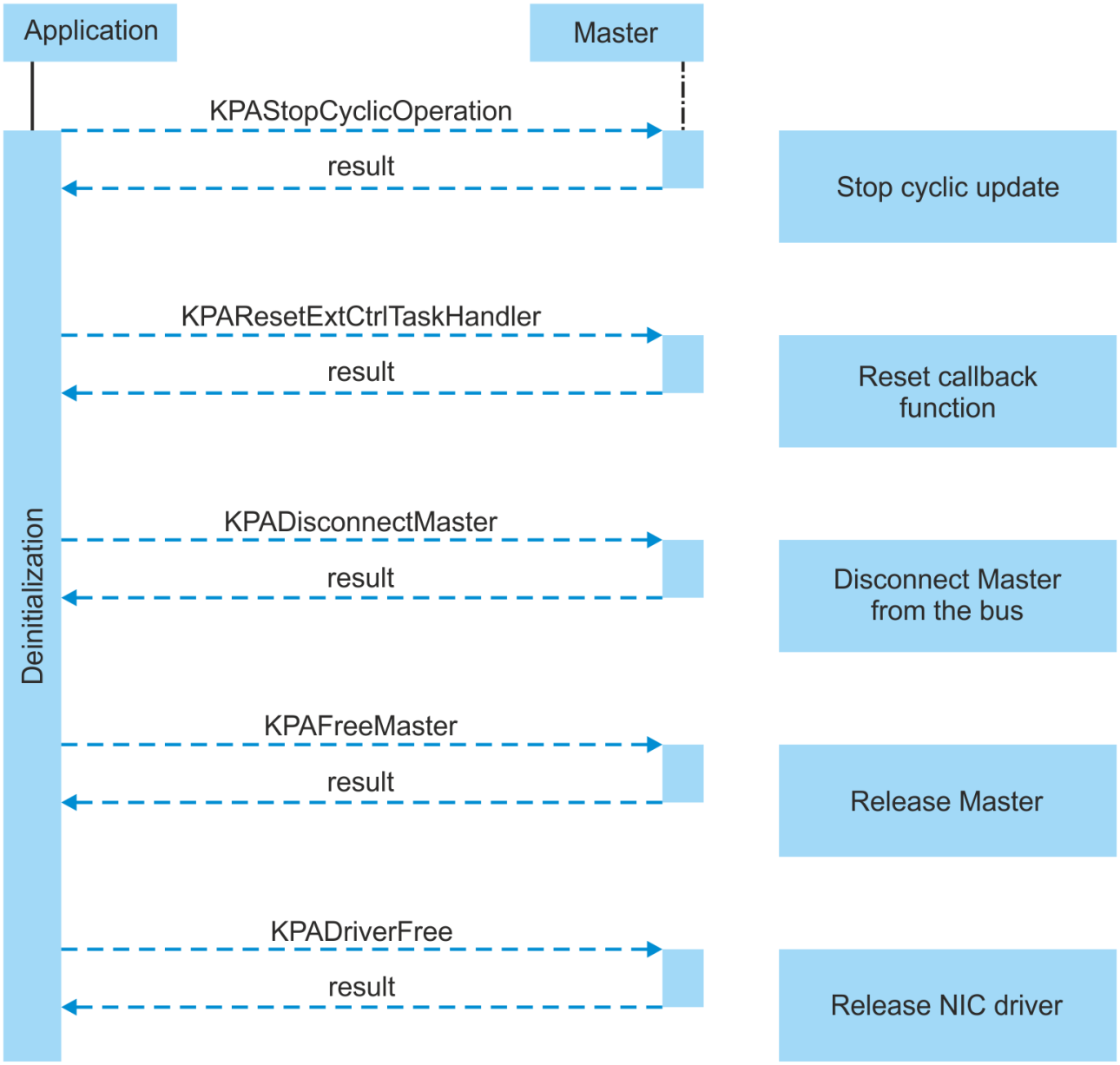

Figure 16. Releasing

10. Imprint and disclaimer

10.1. Trademarks

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

straton® is a registered trademark of Ing. Punzenberger COPA-DATA GmbH, Austria.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

10.2. Disclaimer

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event, shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

10.3. Quality Management

koenig-pa GmbH Quality Management System
is certified according to DIN EN ISO 9001:2015.
Initial certification in 2008.

10.4. Contacts

For more information about products and services, please visit company website: www.koenig-pa.de.
For getting technical support or solving issues arouse from the use of our products, there is a ticketing system in the Customer Portal where you can apply for assistance. Note that the Customer Portal is available for registered users only.
In urgent cases, you have the following options:

- Contact resellers in your country or region.
- Get assistance by phone: +49 9128 725 614, +49 9128 725 330.
- Contact our Support Team at support@koenig-pa.de.

10.5. Mailing address

koenig-pa GmbH
Im Talesgrund 9a
91207 Lauf a.d. Pegnitz, Germany

10.6. Copyright