

EtherCAT as a drive bus



→ The option of being able to use EtherCAT for both drive applications and fast I/O signals was one of its main development aims right from the start. This article by Dr. Dirk Janssen, who manages software developments at Beckhoff, explains the associated features of the EtherCAT Slave Controller and the protocol profile implementations that are relevant for drive technology.

For fieldbus systems, the special requirements of drive technology are regarded as particularly difficult to deal with. In existing systems, short cycle times and high synchronicity (as required for control loops that are closed via the bus) could only be realized with special "drive buses" such as the SERCOS interface. However, these drive buses are only partly suitable for supposedly less critical tasks, such as exchanging normal I/O signals in the PLC cycle. This resulted in the development of separate fieldbuses for drives and I/O signals. If the control system is to be networked, an additional Ethernet connection is required. The user therefore ends up with three incompatible networks that are wired and parameterized differently, although understandably he would prefer a single system. The currently emerging Ethernet-based fieldbuses are set to improve the situation. All variants claim to be suitable for drive technology, able to process PLC signals, and, naturally, to support global networking – after all they are Ethernet-based. However, this consolidation on its own is not quite sufficient. A new fieldbus system should also offer significant advantages in terms of the details of the different requirements.

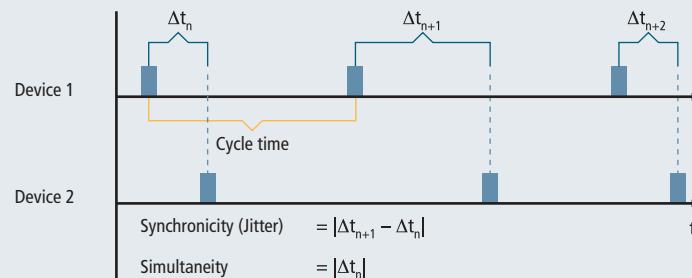
Special drive technology requirements

The supposedly demanding drive technology requirements for fieldbus systems can be listed relatively quickly: cycle time, synchronicity and simultaneity. Typical values for required cycle times range between a moderate 4 ms (cyclic position specification with position control in the drive) to 62.5 μ s in extreme cases (current control loop closed via the bus).

One microsecond is often quoted as an adequate value for synchronicity in drive technology. Since the term synchronicity is not clearly defined in this context, simultaneity should be considered as an additional factor. While synchronicity describes the temporal jitter during processing of the functions in the device involved (drives and controllers), simultaneity defines the measure of temporal offset of these functions.

Synchronicity is important for the individual devices, so that their own subordinate control loops can synchronize with the cyclic signal with the required precision. Simultaneity moreover enables distributed devices to work on a common task with absolutely identical time bases.

Figure 1: Cycle time, synchronicity and simultaneity



Compared with classic PLC technology involving cycle times of 20–30 ms with free-running process data communication, the requirements described are already significantly higher. Meanwhile, advanced PLC systems (e.g. PC-based soft PLC) reach at least the same cycle times and also provide synchronized exchange of I/O signals. Apart from actual control tasks, a PLC tends to deal with more and more monitoring and measurement functions, and requirements are therefore expected to become even more demanding in future. Machine and tool protection systems, which today are realized with special hardware, can thus easily be integrated as a software solution in the PLC.

While for drive technology the required cycle times and synchronicity are limited by the inertia of the mechanical systems to be controlled, measurement technology requirements will determine the range of future applications for fieldbus systems.

Distributed clocks – EtherCAT slave controller features

Synchronicity and simultaneity between distributed systems can be achieved – at least to some extent – in different ways. One frequently used method is based on cyclic transmission of a synchronization signal, which is sent by one of the devices – usually the master – and received quasi simultaneously by all other devices. The procedure requires the bus system to be free whenever the synchronization signal is sent and assumes that the devices receive it without delay. However, non-zero run times in the devices and in the infrastructure components (cables, switches, network controllers etc.), simultaneity cannot be taken for granted in practice. A 100 m long cable, for example, leads to a delay of approximately half a microsecond. Relaying of the signal via a switch can take several microseconds. If the bus system is used for several parallel communications (i.e. drive control, I/O signals and higher-level control systems), even precise transfer of the synchronization signal is difficult to ensure.

Synchronization control: EtherCAT uses a different approach based on so-called “distributed clocks”: All devices have an independent clock as a basis for running local cycles and events. The crucial factor is that all clocks run at the same speed and have the same base time. A special control integrated in the EtherCAT Slave Controller (ESC) ensures that all clocks are guided by a reference clock and are synchronized irrespective of temperature and production tolerances.

Simultaneity – system time: For providing simultaneity, all clocks should have the same base time. To this end, EtherCAT defines a system time with a resolution of 1 ns. The system time (64-bit integer) is managed directly by the ESC and provides an absolute temporal reference (based on Jan. 1, 2000) for events in the complete network at any time. Since the control ensures that all distributed clocks are synchronized, the system time has to be initialized only once.

Run-time measurement: In order to be able to initialize the system time with the required precision in all devices, an ESC-supported run-time measurement is carried out that determines the signal run times between all EtherCAT devices, which are then taken into account for the distribution of the system time. This requires precise knowledge of the installed topology, which for EtherCAT can be very flexible (line, star, tree). The topology is fully readable – also with support from the ESC. A statistical mean value calculation helps to compensate jitter in the communication system. The use of distributed clocks and their initialization requires no support from the application processor in the device, since this is fully dealt with by the EtherCAT Slave Controllers.

Cyclic signals: Based on distributed clocks, ESC offers a number of functions and signals that provide a high-precision time base for the respective application (e.g. a drive controller). The system always uses absolute system time, which automatically ensures the temporal link with distributed devices. Synchronization signals that are connected with capture/compare units of the application processor, for example, provide a high-precision time base for the application. The ESC also provides synchronization inputs for latching the system time for external events. The use of absolute system time also enables analysis of remote events.

Achievable precision: The quality of the control and the precision with which the system times of the individual device can be initialized are crucial factors for the applicability of distributed clocks. The synchronicity value of 1 ms mentioned above can already almost be achieved with classic fieldbuses that are specialized for drive technology. However, depending on the number of devices, the simultaneity error is often significantly higher. Even the SERCOS interface communication system ignores the simultaneity requirement and assumes that all devices receive the associated synchronization signal at the same time. However, in reality the signal is relayed from one device to the next, involving a delay of

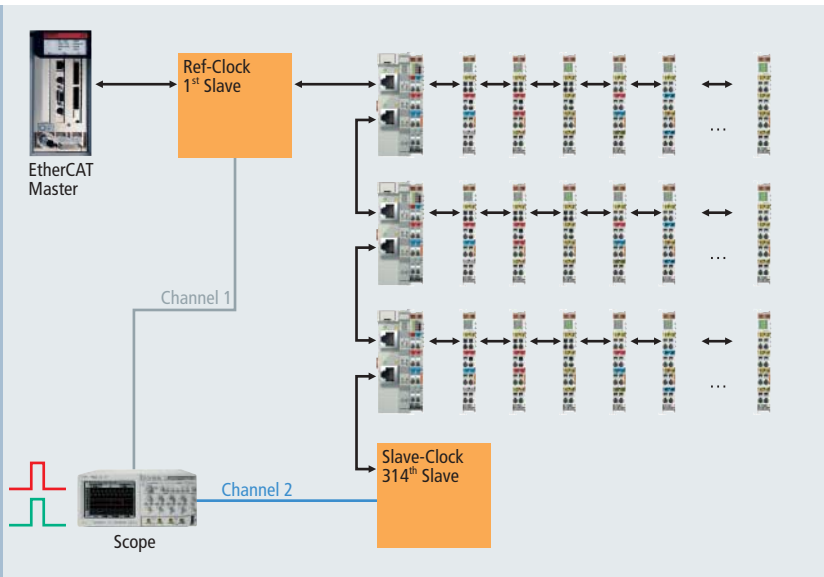


Figure 2: Test configuration

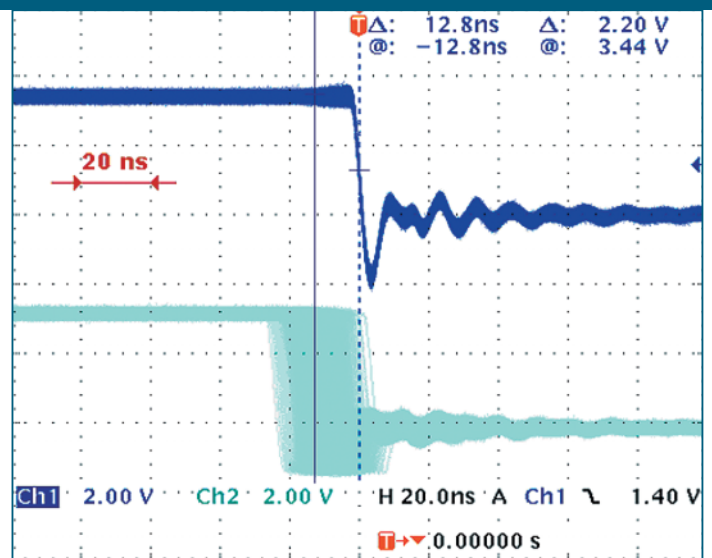
approx. 1 μ s in each case. Even in a system involving only 20 devices, the associated error is therefore approx. 20 μ s. This error cannot be compensated automatically, because firstly the topology is not readable, secondly the run time is not measurable, and thirdly no associated measures are implemented in the associated protocol.

EtherCAT is designed for supporting not just 20 devices, but several hundred or thousand devices. Despite this, the mechanisms described above provide results that are several orders of magnitude better. Figure 2 shows a test configuration for a long-term scope recording (Figure 3). The synchronization signals of two EtherCAT devices are shown. Between these two devices, the test configuration features more than 300 further devices and overall approximately 120 meters of cable, representing a realistic application. The scope is triggered by the signal of the first device and displays the signal of the second at the bottom. The width of the signal edge (persist time set to infinite) corresponds to the synchronization error and is approx. \pm 20 ns! The deviation of the mean value from the trigger point of the first device corresponds to the simultaneity error, in this case approx. 12 ns! Comprehensive measurements showed that both the synchronization error and the simultaneity error are significantly less than 100 ns, even for large networks.

Multi-protocol capability

Other important criteria for a fieldbus system that is to support drive technology are the communications protocol and profile used, which are responsible for compatibility and efficient data exchange between the controller and the drive. Instead of re-inventing the wheel, EtherCAT uses proven technology for this purpose.

Figure 3: Synchronicity and simultaneity in an EtherCAT network



None of the available protocols on their own support all communication requirements of modern fieldbuses (process data, parameter data, parallel TCP/IP, firmware updates, routing to subordinate bus systems, etc.). EtherCAT therefore introduces multi-protocol capability, consolidating the different protocols in a standardized mailbox. This enables quick and full conversion of existing devices to EtherCAT. The protocols relevant for drive technology are "CANopen over EtherCAT" (CoE) and "SERCOS over EtherCAT" (SoE). They enable the advantages of EtherCAT in terms of transfer characteristics to be combined with proven, profile-specific drive functions. The "Ethernet over EtherCAT" (EoE) and "File Access over EtherCAT" (FoE) protocols provide options for integrating a web server in

Figure 4: Multi-protocol capability of the EtherCAT mailbox

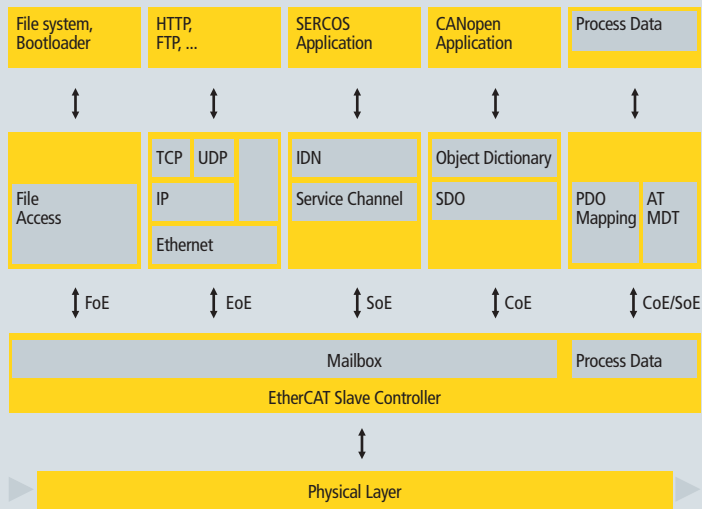
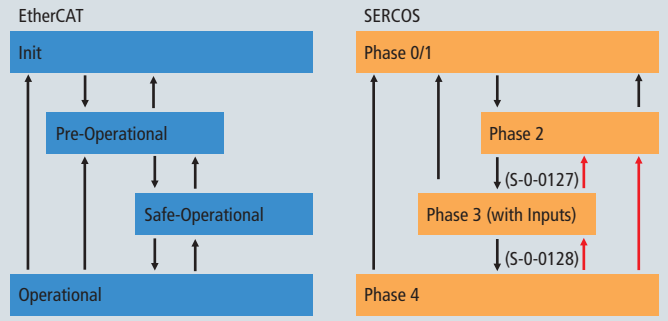


Figure 5: EtherCAT slave state machine and its mapping to the SERCOS phases



the drive, for example, or for efficiently exchanging firmware or cam plate tables via the bus (Figure 4).

CANopen over EtherCAT: The “CANopen over EtherCAT” (CoE) protocol enables the complete CANopen profile family to be utilized via EtherCAT. The SDO protocol is used directly, so that existing CANopen stacks can be used practically unchanged. Optional extensions are defined that lift the 8-byte limit and enable complete readability of the object list.

The process data are organized in process data objects (PDO), which are transferred using the efficient means of EtherCAT – naturally without 8-byte limit. All CANopen profiles – including the drive profile (DS 402) – are fully usable, and devices based on it can be transferred to EtherCAT very easily.

Apart from a few details, the EtherCAT slave state machine corresponds to the CANopen state machine, so that only a limited number of changes are required. In order to enable less ambiguous start-up behavior, a further state called “safe operational” is defined, to which inputs that are already valid are transferred, while the outputs remain in safe state (see Figure 5).

SERCOS over EtherCAT: The protocol “SERCOS over EtherCAT” (SoE) enables the proven SERCOS profile, which is specialized for demanding drive technology, to be used. The SERCOS service channel, and therefore access to all parameters and functions residing in the drive, is mapped to the EtherCAT mailbox. Here too, the focus is on compatibility with the existing protocol (access to value, attribute, name, units etc. of the SERCOS identifiers) and expandability with regard to data length limitation. The process data (for SERCOS in the form of AT and MDT data) are transferred via the EtherCAT Slave Controller. The associated mapping is done SERCOS-compliant via the identifiers S-0-0015, S-0-0016 and S-0-0024.

For synchronization – like for the CoE protocol – the synchronization features of the EtherCAT Slave Controller are utilized as described above. This also includes – with associated quality improvements – those of standard SERCOS, so that implementation is correspondingly easy.

The EtherCAT slave state machine explained above can also be mapped easily to the phases of the SERCOS protocol. “Pre-operational” corresponds to SERCOS, phase 2, and enables service channel communication without process data exchange. “Safe operational” is comparable with phase 3. Synchronization is carried out as required, although for EtherCAT inputs that are already valid have to be transferred. “Operational” exactly corresponds to phase 4 for normal, cyclic data exchange.

As the name suggests, the EtherCAT slave state machine refers to a slave and – unlike SERCOS – therefore enables individual drives to be parameterized and started up independent of other devices.

Conclusions

While EtherCAT is not a pure drive bus, it meets the associated requirements at least one order of magnitude better than familiar, specialized systems. Drive, I/O and communication buses therefore no longer have to be separated. Even demanding tasks, such as measurement technology applications, can be integrated and enable new functionalities to be utilized with “classic” control technology. Because proven communication profiles are used, migration of existing devices and applications can easily be accomplished. For drive technology in particular, a small number of profiles have been developed and tried and tested over years. It also means that the complete tool chain and existing experience with the parameterization of associated drives is maintained.