HOCHSCHULE für Angewandte Wissenschaften LANDSHUT
UNIVERSITY of Applied Sciences

FAKULTÄT Informatik

HOCHSCHULE LANDSHUT

FAKULTÄT INFORMATIK

Aufbau einer Master-Slave-Kommunikation mit EtherCAT

Bachelorarbeit

vorgelegt von

Sebastian Bilda

aus Landshut

eingereicht am: .....

Betreuer: Thomas Franzke

Erklärung zur Bachelorarbeit

Name, Vorname
der/des Studierenden: …….

Hiermit erkläre ich, dass ich die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Landshut, den ……. …..…….
(Unterschrift der/des Studierenden)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

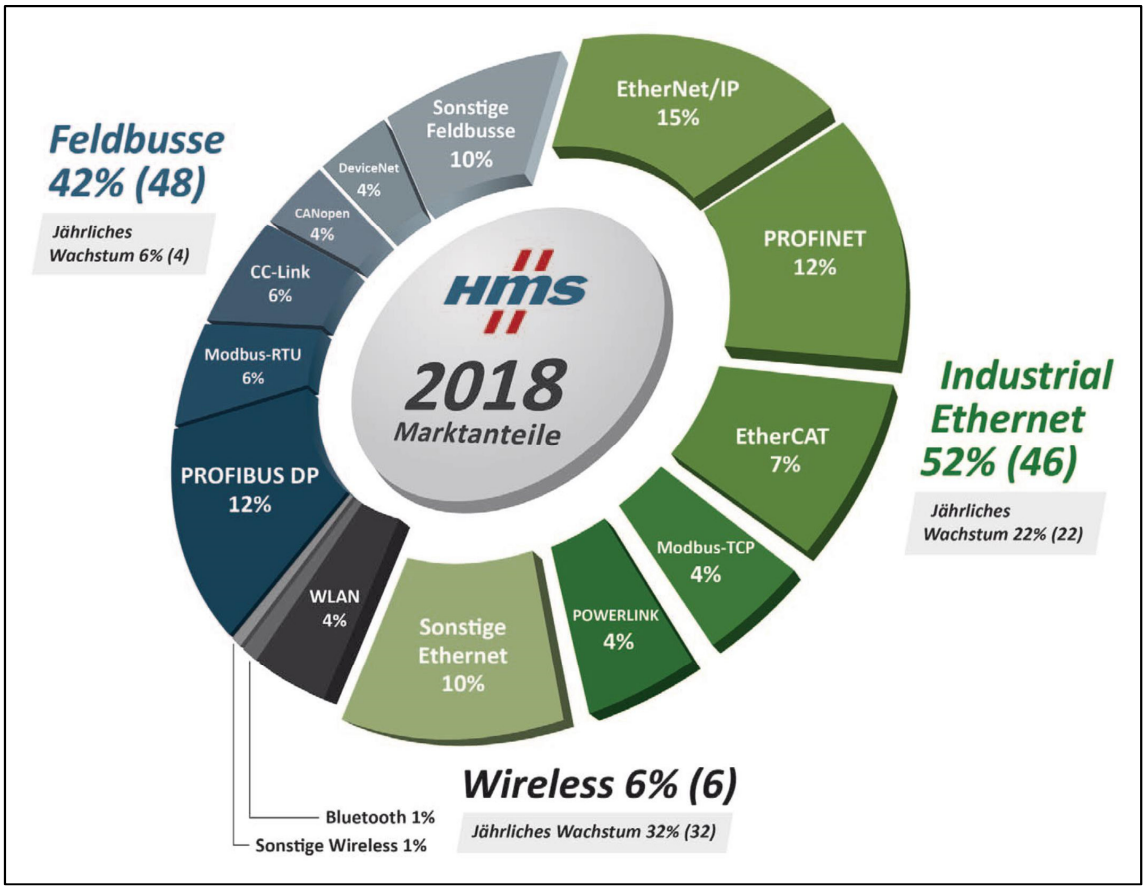
Listingverzeichnis

Abkürzungsverzeichnis

CSMA/CD	Carriere Sense Multiple Access / Collision Detection
SFD	Start Frame Delimiter
FCS	Frame Check Sequence
CRC	Cyclic Redundancy Check
SOEM	Simple Open EtherCAT Master
SDO	Servicedatenobjekte
DC	Distributed Clocks
FoE	File over EtherCAT
CoE	Can over EtherCAT
SII	Slave Information Interface
SoE	Servo over EtherCAT
ETG	EtherCAT Technologie Group
EDP	EtherCAT Device Protocol
EAP	EtherCAT Automation Protocol
MAC	Medium Access Control
ESC	EtherCAT Slave Controller
FPGA	Field Programmable Gate Array
ASIC	Application-Specific Integrated Circuit
WKC	Working Counter
ESM	EtherCAT State Maschine
FMMU	Fieldbus Memmory Management Unit
MMU	Memmory Management Unit
SM	Sync Manager
PDO	Prozessdatenobjekte

1 Einleitung

Die noch weit verbreiteten traditionellen Feldbusse, die zum Einsatz in der Automatisierungs- und Regelungstechnik kommen, werden mehr und mehr von Industrial Ethernet Systemen abgelöst. Laut einem Bericht von Thomas Carlsson [Thomas Carlsson 2018] liegt der Anteil an Industrial Ethernet im Einsatz in der Automatisierung bei mehr als der Hälfte, wenn die Anzahl der neu installierte Systeme betrachtet werden. Es gibt mehrere Industrial Ethernet Lösungen, wie in der Abbildung 1-1 zu sehen ist.



Quelle: [Buchholz 2018, S. 2]

Abbildung 1-1: Marktanteile Feldbus vs. Industrial Ethernet und Wireless

Es wird ebenfalls ein weiterer Rückgang der traditionellen Feldbusse erwartet. Dies liegt an den Faktoren, die unter anderem auch für den Wachstum des Industrial Ethernet ausschlaggebend sind. Zum einen schnelle Zykluszeiten von 1 ms und zum anderen die hohe Bandbreite mit 1 Gbit/s für die immer wachsende Datenmenge, die über das Netz übertragen werden soll. Außerdem sind das Industrial Internet of Things und die Integration von Fabrikinstallationen und IT/IoT-Systemen wesentliche Treiber für Industrial Ethernet. [Buchholz 2018, S. 3]

Das Ziel dieser Arbeit ist der Aufbau einer Master-Slave-Kommunikation zur Bestätigung, dass eine Hausautomatisierung über EtherCAT realisierbar ist. Dazu werden von der Firma WAGO, einem Hersteller für Automatisierungstechnik, analoge und digitale Ein- und Ausgangsklemmen und der WAGO Feldbuskoppler als EtherCAT Slave Controller verwendet. Diese Geräte können über die Industrial Ethernet Lösung EtherCAT kommunizieren. EtherCAT ist für sogenannte Motion-Control-Anwendungen ausgelegt, das heißt Anwendungen mit sehr kurzen Zykluszeiten, bei denen Synchronität, Gleichzeitigkeit und Zyklustreue entscheidend sind [EtherCAT Technology Group 2018a].

Diese Arbeit ist in sieben Kapitel unterteilt. Im folgenden zweiten Kapitel wird der Ethernet-Standard erläutert, der die Grundlage für EtherCAT ist. Des Weiteren werden Industrial Ethernet und der Begriff der Echtzeitfähigkeit beschrieben. Zudem wird auf den Simple Open EtherCAT Master (SOEM) Software Stack eingegangen und dessen Funktionalität erläutert. Im dritten Kapitel wird genauer auf EtherCAT eingegangen, insbesondere auf dessen Funktionsweise, Protokoll und weiteren Eigenschaften. Der Versuchsaufbau und die Vorbereitungen für die Umsetzung werden im vierten Kapitel beschrieben. Kapitel fünf erläutert die Implementierung der Testanwendung unter Verwendung des SOEM Stacks. Im Zuge dessen wird auf die Verwendung der Funktionen des SOEM eingegangen. Darauf folgt in Kapitel sechs die Darstellung der Ergebnisse. Im letzten Kapitel wird ein Fazit der Arbeit gezogen und ein Ausblick auf die Möglichkeiten zur Hausautomatisierung über EtherCAT.

2 Grundlagen

In diesem Kapitel sollen die wichtigsten Grundlagen definiert werden. Es bietet eine Einführung in das Thema und beschreibt die Funktionalität der Simple Open EtherCAT Master Bibliothek, die in der entwickelten Software verwendet wird.

2.1 Ethernet

Ethernet ist der am weitesten verbreitete Standard für lokale Netze, welcher auf gemeinsame Spezifikationen von DEC, Intel und Xerox (DIX) zurück geht. Ziel war es, die Ethernet-Technologie als LAN-Standard durchzusetzen. Später übernahm die IEEE (Institute of Electrical and Electronical Engineers) die Weiterentwicklung und gründete dafür die Arbeitsgruppe 802.3. Der erste Entwurf wurde als Carrier Sense Multiple Access with Collision Detection (CSMA/CD) vorgelegt. 1983 wurde Ethernet als Standard IEEE 802.3 verabschiedet [IEEE]. Ethernet lehnt sich an das OSI-Schichtenmodell an (siehe Abbildung 2-1). Das OSI-Modell wird in sieben Teilbereiche, die als Schichten oder Layer bezeichnet werden, unterteilt. Jede Schicht stellt ihre Funktion der übergeordneten Schichten zur Verfügung [Rech 2008, S. 31 ff.].

Quelle: [vgl. Rech 2008, S. 79]

Abbildung 2-1: Die 7 Schichten des OSI-Referenzmodul

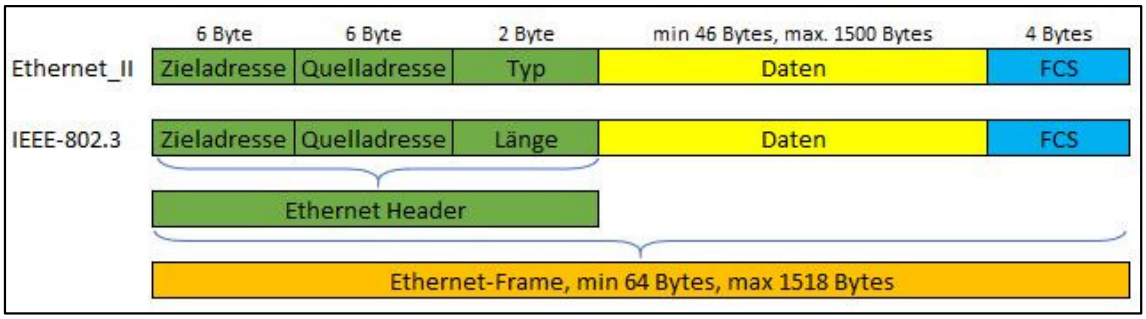
2.1.1 Ethernet-Frameformat

Ethernet-Frames ermöglichen den Datenaustausch zwischen zwei Geräten in einem Netzwerk. Das Ethernet-Interface, welches jedes Gerät besitzt, stellt eine Quelle

und/oder Senke für Nachrichten dar. Dabei kann es sich, je nach Gerät, um ein echtzeitfähiges oder nicht-echtzeitfähiges Gerät handeln. [Dopatka 2008, S. 33]

Bedingt durch die Entstehungsgeschichte von Ethernet sind mehrere Frameformate entwickelt worden. Das Ethernet_II Frame, welches von der DIX-Gruppe veröffentlicht wurde und das neuere IEEE-802.3-Frame, das auch im Ethernet-Standard manifestiert ist. [Rech 2008, 75-77]

In Abbildung 2-2 werden die beiden Frameformate gegenübergestellt. Im weiteren Verlauf dieser Arbeit wird das Ethernet_II Frame verwendet, da EtherCAT einen eigenen Ethertype besitzt.



Quelle: [vgl. Rech 2008, S. 76]

Abbildung 2-2: Ethernet_II Frame und IEEE-802.3-Frame

Das Ethernet_II Frame gliedert sich in drei Segmente. Den Ethernet Header (grün), den Daten (gelb) und der Frame Check Sequence (blau).

Der Ethernet Header beinhaltet die je 6 Byte großen Ziel- und Quell-MAC-Adressen sowie ein 2 Byte großes Typ Feld. Das Typfeld, auch Ethertype genannt, legt den Typ der höheren Protokollschicht fest. Die Typnummer für das EtherCAT Protokoll ist $88A4_{16}$. In diesem dritten Feld des Ethernet-Headers liegt auch der Unterschied der beiden Frameformate. Der IEEE-802.3-Frame verwendet es als Längenfeld. Das bedeutet, wenn der Hexadezimalwert kleiner als $05DC_{16}$ (1500_{10}) ist, entspricht das einer Längenangabe des Datenfelds. Mit dieser Feststellung kann mit Gewissheit gesagt werden, dass es sich um ein IEEE-802.3-Frame handelt. Die Normungen des Ethertypes beginnen bei 0600_{16} (1536_{10}). [Rech 2008, S. 76 f.]

Im Anschluss an das Typenfeld kommt der Datenteil des Ethernet-Frames. Die Mindestlänge wird durch das Zugriffsverfahren CSMA/CD bestimmt. Sollte die Länge der zu übertragenden Daten geringer sein, wird mit Füllzeichen bis auf 46 Bytes aufgefüllt. [Rech 2008, S. 76 f.]

Den Abschluss des Frames bildet die Frame Check Sequence (FCS), die mit einem Algorithmus prüft, ob die Daten fehlerfrei übertragen wurden. Der Empfänger des Frames berechnet über die Ziel- und Quelladresse, Typen- oder Längenfeld und Datenteil die FCS und vergleicht sie mit der vom Sender gleichermaßen gebildeten FCS im Frame. Falls diese nicht übereinstimmen wird das Frame verworfen. [Rech 2008, S. 76 ff.]

2.1.2 CSMA/CD

Das CSMA/CD Verfahren regelt den Zugriff auf das Übertragungsmedium und wird nur im Halbduplexmodus verwendet. In diesem Modus kann ein Gerät entweder Senden oder Empfangen, jedoch nicht beides gleichzeitig. [Dopatka 2008, S. 37 f.]

Dieses Verfahren ist unabhängig davon, welches Übertragungsmedium genutzt wird. Durch den Algorithmus besitzen alle Stationen die Chance auf gleichberechtigten Zugriff auf das Netzwerk. Die grundsätzliche Funktionsweise ist in mehrere Phasen unterteilt. Das Überwachen von Aktivitäten auf dem Kabel überprüft zu jeder Zeit ob eine andere Station sendet. Ist die Leitung frei kann der Sendevorgang eingeleitet werden, andernfalls wird gewartet bis die Leitung frei ist. Erkennt eine Station eine Kollision, wird der Sendevorgang abgebrochen und nach einer zufällig berechneten Zeit die Übertragung wiederholt. [Riggert et al. 2014, S. 77 f.]

2.1.3 Kenngrößen der Übertragungsstandards

Im Nachfolgenden werden Spezifikationen mit einer Kennzeichnung, wie zum Beispiel 10BASE5 verwendet. Diese werden wie folgt interpretiert:

{Datenrate}{Übertragungsverfahren}{Segmentlänge/Kabeltyp}

2.1.3.1 Der 10Mbit/s-Standard

Das „Ursprungs-Ethernet“ mit einer Übertragungsgeschwindigkeit von 10 Mbit/s nach IEEE802.3 [IEEE] wurde durch verschiedene Übertragungsmedien spezifiziert. Identisch ist die Kodierung in der Bitübertragungsschicht. Bei dem 10Mbit/s-Standard wird die Manchester-Kodierung eingesetzt.

Die Variante **10BASE5** wird mittlerweile nicht mehr neu installiert, außerdem ist sie, wenn überhaupt nur noch selten in bereits installierten Netzwerken zu finden. Dennoch ist sie die Basis aller Erweiterungen und gehört deshalb der Vollständigkeit halber erwähnt. 10BASE5 wird als Bus-Topologie realisiert und nutzt als Übertragungsmedium ein im Durchschnitt 1 cm dickes Koaxialkabel („Yellow Cable“). Die Endstationen

werden alle mittels Transceiver am Übertragungsmedium angeschlossen. Es ist eine maximale Begrenzung von 100 Stationen im Segment festgesetzt, bei der die Netzwerkausdehnung 500 Meter beträgt. Mit Repeatern kann sie auf 2500 Meter erweitert werden. 10BASE5 arbeitet ausschließlich im Halbduplexmodus. [Rech 2008, 43f]

10BASE-T (Twisted Pair) / F (Fiber-Optic) bleibt logisch durch eine Bus-Topologie realisierbar, wird jedoch physikalisch als Stern umgesetzt. Hierbei wird als Übertragungsmedium ein symmetrisches Kupferkabel (Twisted-Pair Kabel) anstelle des Koaxialkabels verwendet, welches die Fehleranfälligkeit senkt. Dieses Kabel kann genau zwei Komponenten miteinander verbinden. Sollen also mehrere Teilnehmer angeschlossen werden, müssen Verteiler (Hubs/Switches) verwendet werden, um eine sternförmige Verkabelungsstruktur zu erreichen. Mit der Einführung von 10BASE-T wurde der Einsatz von RJ45-Stecker und -Buchsen und die Vollduplexeigenschaft etabliert, die bis heute eingesetzt werden. Die Vollduplexeigenschaft besagt, dass jedes Gerät in einem Netzwerk gleichzeitig Datenpakete senden und empfangen kann. Die Twisted Pair (TP)-Kabel sind in verschiedene Kategorien eingeteilt, in denen die elektrischen Mindestanforderungen und Übertragungsfrequenzen spezifiziert sind. Die Kabel sind abwärtskompatibel. 10BASE-F ist eine Alternative zur Datenübertragung über Lichtwellenleiter. So können Strecken von bis zu einem Kilometer zwischen zwei Repeatern sein, und damit die Fabrikhalle und das Bürogebäude miteinander verbinden. [Rech 2008, S. 49 ff.]

2.1.3.2 100 Mbit/s – Fast-Ethernet

100BASE-T wurde von der IEEE-802.3u-Gruppe erarbeitet und in Form einer Erweiterung des Ursprungs-Ethernet 1995 verabschiedet. Mit Fast Ethernet wurde die Datenrate um den Faktor 10 gesteigert. Bei dieser Erweiterung kam ebenfalls das Zugriffsverfahren CSMA/CD zum Einsatz, wodurch eine Rückwärtskompatibilität gewährleistet ist. In der IEEE 802.3u wurden zwei Interfacevarianten definiert, die auf unterschiedlichen Übertragungsmedien aufsetzten. 100BASE-TX verwendet ein TP-Kabel der Kategorie 5 und 100BASE-FX basiert auf Glasfaserkabel. Die gesteigerte Datenrate hatte eine Veränderung der Bitübertragungsschicht zufolge. Durch ein anderes Kodierungsverfahren, namens 4B/5B, konnte die Datenübertragung auf Medien wie dem TP-Kabel ermöglicht werden. Die Erweiterung ist für eine Halb- und Vollduplexübertragung geeignet. [Rech 2008, S. 85]

Das TP-Kabel, bei dem ein Adernpaar für das Senden und eines für den Empfang zuständig ist, hat wie der 10 Mbit/s Standard, RJ-45-Stecker und -Buchsen. Wenn mehrere

Stationen vernetzt werden sollen, sind diese sternförmig an einen Hub oder Switch anzuschließen. [Rech 2008, S. 93 f.]

2.1.3.3 Gigabit-Ethernet

Gigabit Ethernet ist nach dem gleichen Prinzip aufgebaut wie die vorangegangenen Standards. Es verwendet die gleichen Frametypen, Längen und Formate und setzt nach wie vor das CSMA/CD-Verfahren zur kollisionsfreien Übertragung ein [Rech 2008, S. 116]. Die Sicherungsschicht (Data Link Layer) wird nicht verändert. Durch Veränderungen in der Bitübertragungsschicht konnte erneut die Übertragungsgeschwindigkeit, um den Faktor 10, auf 1000 Mbits/s erhöht werden. Die Erweiterung des Kodierungsverfahren auf 8B/10B und die Einführung eines Serializer/Deserializer ermöglichten diese Erhöhung. Auch der Gigabit Standard hat die Voll-Duplex-Fähigkeit. [Airnet Technologie- und Bildungszentrum GmbH 2009]

2.2 Industrial Ethernet

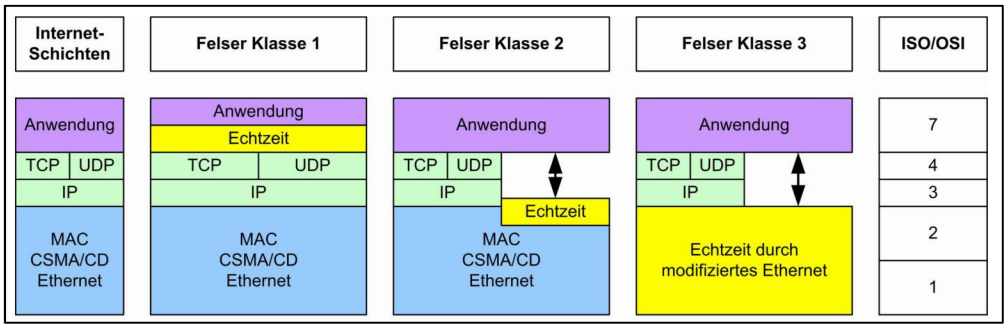
Michael Volz beschreibt in dem Sammelwerk „Industrielle Kommunikation mit Feldbus und Ethernet“ den Begriff Industrial Ethernet als „Oberbegriff für alle Bestrebungen den Ethernet-Standard gemäß der internationalen Normenreihe IEE 802.xx für die Vernetzung von Geräten in der Automatisierungstechnik nutzbar zu machen“ [Klasen et al. 2010, S. 129]. Für Industrial Ethernet wird auch oft der Begriff „Echtzeit-Ethernet“ verwendet.

Da der 10 Mbit/s Ethernet-Standard nicht ausreichend schnell und sehr kollisionsbehaftet ist, wurde durch die Einführung des Fast Ethernets mit den Strategien der Switching-Technologie und einer Deaktivierung des CSMA/CD für eine Vollduplexübertragung die Grundlage für schnelle Echtzeitübertragung geschaffen. Mit Industrial Ethernet soll ein einheitliches Kommunikationssystem mit guter Anbindung an die IT-Systeme in der Leitebene und gute Echtzeiteigenschaften für die Kommunikation der dezentralen Feldgeräte mit den Steuerungen definiert werden. Es entstanden viele miteinander konkurrierende Protokollstandards, die als einzige Gemeinsamkeit nur die Übertragungstechnik und das Buszugriffsprotokoll haben [Klasen et al. 2010, S. 129 ff.]. Mit Industrial Ethernet ist es möglich Echtzeitdaten und IT-Daten zeitgleich und über das gleiche Medium zu übertragen. Ebenso steht eine fast unbegrenzte Anzahl an Teilnehmern durch den großen Adressbereich zur Verfügung. Eine große Netzwerkausdehnung ist durch die Kaskadierung von Switches realisierbar. Auch große Datenmengen können effizient übertragen werden. Es besteht ein gleichberechtigter Buszugriff aller Netzwerkteilnehmer. Das Übertragungsmedium ist nicht eingeschränkt auf Kabel, sondern kann auch in

Kombination mit Funk oder Lichtwellenleiter eingesetzt werden. [Schnell et al. 2012, S. 274 ff.]

2.2.1 Klassen der Industrial Ethernet Systeme

In der Dissertation von Dopatka [Dopatka 2008] wird die Echtzeitfähigkeit in das 4-Schichten-Modell der Internet-Architektur und das OSI-Modell in 3 Klassen kategorisiert. Die Abbildung 2-3 zeigt die drei Klassen, in die Industrial Ethernet Systeme eingegliedert werden können.



Quelle: [Dopatka 2008, S. 70]

Abbildung 2-3: Übersicht der Klassen nach Felser

Jede Felser[1] Klasse gewährleistet eine deterministische, auf Ethernet basierende Infrastruktur. Sie unterscheiden sich hinsichtlich der Einbettung der Echtzeitfähigkeit.

Die erste Klasse setzt die Echtzeitfähigkeit oberhalb der Transportschicht ein, sodass der gewohnte Protokollstapel erhalten bleibt. Diese Lösung hat eine limitierte Performance. [KINGSTAR 2015, S. 6] Ein Vertreter dieser Klasse ist Modbus/TCP.

Bei der zweiten Klasse werden neue Protokolle auf den Ethernet-Schichten implementiert. Hier wird von der Ethernet-Evolution ohne weiteren Aufwand profitiert. Jedoch muss ein Software-Controller (Switch/Hub) auf OSI-Schicht 3 und 4 enthalten sein um Latenzen zu vermeiden. [KINGSTAR 2015, S. 6] Zu der zweiten Klasse gehört zum Beispiel PROFINET.

Der Ansatz der dritten Klasse liegt in der Implementierung eines neuen Standards, der die vorhandene Ethernet-Hardware nutzt und Determinismus garantiert [KINGSTAR 2015, S. 6]. Diese Modifikation hat den Zweck, "für die härtesten Echtzeitklassen optimierte Hardware bereit zu stellen" [Dopatka 2008, S. 70]. Der in Kapitel 3 behandelnde EtherCAT Standard ist ein Vertreter dieser Klasse.

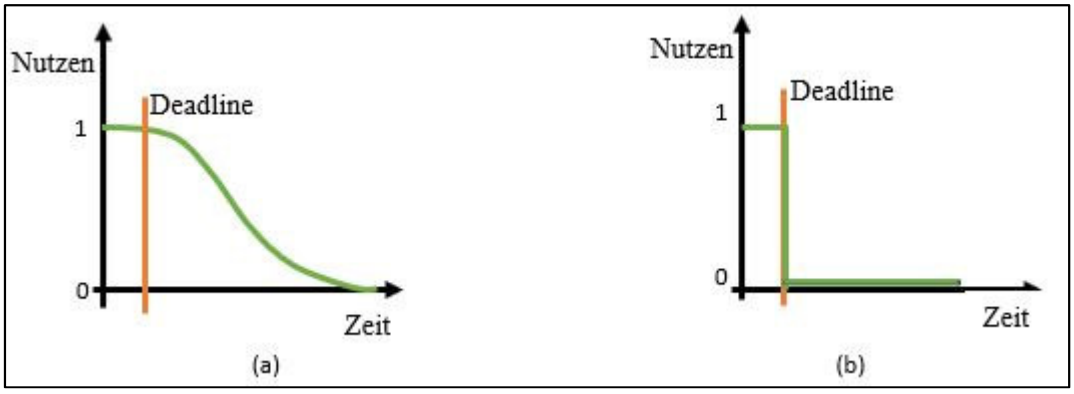
[1] Max Felser, Dipl. El. Ing. ETH kategorisierte die Industrial Ethernet Systeme

2.3 Der Begriff „Echtzeitfähigkeit“

Ein System gilt dann als „echtzeitfähig“, wenn ein streng definiertes Zeitverhalten vorliegt. Nicht die Geschwindigkeit der Bearbeitung der Daten, sondern die Verlässlichkeit der Bearbeitungszeit sind relevant. Dopatka [Dopatka 2008, S. 24 ff] definiert folgende Begriffe als die Hauptanforderungen an Echtzeitsysteme:

- Rechtzeitigkeit
- Gleichzeitigkeit
- Vorhersehbarkeit
- Verlässlichkeit

Die Rechtzeitigkeit besagt, dass Daten bis zu einer vorgegebenen Zeitschranke (Deadline) erfasst und ausgewertet werden. Es gibt unterschiedliche Arten von Zeitschranken, die meist in Zeit/Nutzen-Diagrammen dargestellt werden. Zum einen die weiche Echtzeit (Abbildung 2-4a) und zum anderen die harte Echtzeit (Abbildung 2-4b).



Quelle: [vgl. Prof. Dr. M. Pellkofer 2017, S. 15 f.]

Abbildung 2-4: Zeit/Nutzen-Diagramme der Echtzeit

Bei weicher Echtzeit ist die Einhaltung der Deadline zwar wichtig, wobei gelegentliche Überschreitungen der Deadline tolerierbar sind. Bei harter Echtzeit hingegen kann ein Verpassen der Deadline katastrophale Folgen haben, sodass eine Fehlfunktion des Systems vorliegt. Die Einhaltung der Deadline ist zu allen Zeitpunkten zwingend erforderlich. [Prof. Dr. M. Pellkofer 2017, S. 15 f.]

Bei der Gleichzeitigkeit geht es darum, auf mehrere, gleichzeitig ablaufende Prozesse, insbesondere unter der Berücksichtigung der jeweiligen Zeitschranken reagieren zu können [Dopatka 2008, S. 25].

Das Ziel der Vorhersehbarkeit ist es "Verletzungen von Zeitschranken und entsprechenden Schäden quantifizierbar und so unwahrscheinlich wie möglich zu machen." [Prof. Dr. M. Pellkofer 2017, S. 161], daher müssen die von einem "Rechner auszuführenden Reaktionen auf Zustandsänderungen […] immer genau geplant und vorhersehbar sein". [Prof. Dr. M. Pellkofer 2017, S. 161]

Die drei Eigenschaften Zuverlässigkeit, Verfügbarkeit, und Sicherheit müssen erfüllt sein, um die Verlässlichkeit in einem Echtzeitsystem zu gewährleisten, d. h. geeignete Schutzmaßnahmen gegen Störeinflüsse sind zu implementieren [Prof. Dr. M. Pellkofer 2017, S. 159 f.].

2.4 Simple Open EtherCAT Master

Die Simple Open EtherCAT Society ist eine Initiative der Firmen rt-labs und Speciaal Machinefabriek Ketels. Diese Vereinigung hat die EtherCAT Master Bibliothek erstellt, mit der ein EtherCAT Master implementiert und mit EtherCAT Slaves kommuniziert werden kann. Der Simple Open EtherCAT Master (SOEM) ist in der Programmiersprache C geschrieben und wird in dieser Bachelorarbeit verwendet. Die Bibliothek kann sowohl unter Windows als auch unter Linux verwendet werden. Sie stellt der Benutzeranwendung folgende Funktionalitäten bereit [Open EtherCAT Society 2017]:

- Datagramme in Ethernet-Frames einzubetten und die grundlegende Basis EtherCAT-Frames über verschiedenen Adressierungsmöglichkeiten zu senden und zu empfangen
- Automatische Konfiguration von Fieldbus Memory Management Unit, SyncManager und Verteilte Uhren der Slaves
- Lesen und Verändern der Slavezustände
- Automatische Generierung des Prozessdatenmappings
- Lesen und Schreiben des EEPROM der Slaves
- Zeit-, Timer- und Threadfunktionen

3 EtherCAT

Ethernet for Control Automation Technology, kurz EtherCAT, ist ein Ethernet-basiertes Feldbussystem, das von Beckhoff und der ETG entwickelt wurde. EtherCAT wird in der ETG-Broschüre „durch herausragende Performance, niedrige Kosten, flexible Topologie und einfache Handhabung gekennzeichnet“ [Martin Rostan, EtherCAT Technology Group 2018, S. 4] und bietet somit schon viele Eigenschaften die ein Feldbussystem in der Automatisierungs- und Steuerungstechnik mit sich bringen soll. Der nach Rostan „schnellste Industrial-Ethernet-Standard“ [Martin Rostan, EtherCAT Technology Group 2018, S. 3] wurde 2003 vorgestellt und gilt seit 2007 als internationaler Standard. Auf der Grundlage von Industrial-Ethernet nutzt EtherCAT Standard-Frames sowie die physikalischen Schichten aus dem Ethernet Standard IEEE 802.3.

EtherCAT wird aufgeteilt in das EtherCAT Device Protocol (EtherCAT Feldbus) und das EtherCAT Automation Protocol (EAP). Die Feldbusebene basiert auf einer Master-Slave-Kommunikation zwischen einem Echtzeitcontroller (EtherCAT-Master) und den Feldgeräten (Sensoren, Aktuatoren, I/O). Das EAP ist übergeordnet und ermöglicht die Kommunikation von EtherCAT-Master und ERP Systemen. [EtherCAT Technology Group 2018b]

3.1 Funktionsprinzip

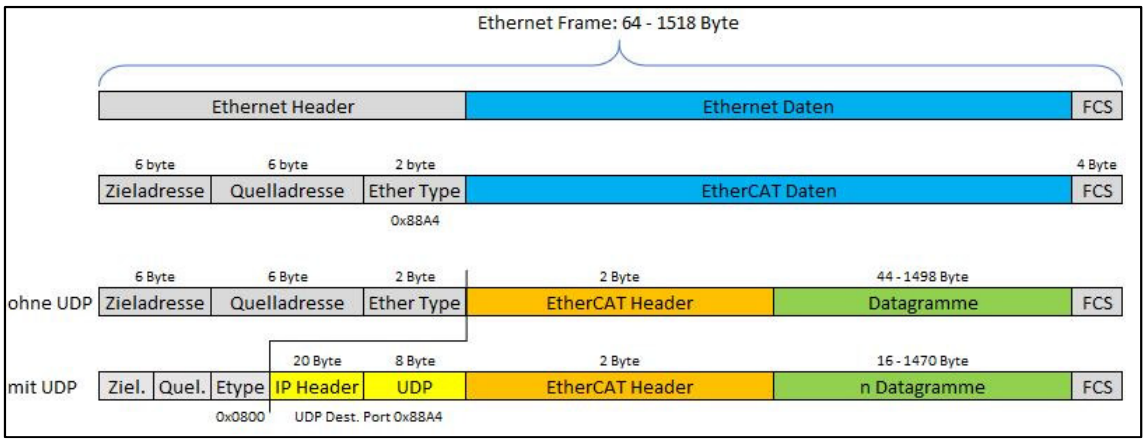
Der EtherCAT-Master sendet zyklisch ein Telegramm aus, dass alle Teilnehmer durchläuft. Er ist dabei der einzige, der aktiv einen EtherCAT-Frame senden darf. Die EtherCAT-Slaves lesen aus dem Paket "on the fly" die jeweilig an sie adressierten Ausgangsdaten, fügen ihre Eingangsdaten ein und leiten den Frame anschließend weiter. Der letzte EtherCAT-Slave sendet das vollständig bearbeitet Telegramm als Antwort an den Master zurück. Hierbei wird die Voll-Duplex-Eigenschaft der Ethernet-Physik und die getrennten Übertragungsleitungen (Tx- und Rx-Leitung) ausgenutzt. [Dr.-Ing. Dirk Janssen et al. 2003, S. 2] Dadurch liegt die effektive Datenrate bei über 100 MBits/s [Dipl.-Ing. Martin Rostan 2003, S. 2].

Als Hardware für den Master ist eine Standard-Ethernet-MAC ohne zusätzlichen Kommunikationsprozessor ausreichend um mit EtherCAT Echtzeitanwendungen zu steuern [BECKHOFF New Automation Technology].

Die Slaves verwenden einen EtherCAT Slave Controller (ESC), welche als FPGA oder ASIC erhältlich sind. Die Verarbeitung der Telegramme wird komplett über die Hardware umgesetzt. [EtherCAT Technology Group 2018a]

3.2 Das EtherCAT Protokoll

In diesem Abschnitt wird ein EtherCAT-Frame beschrieben und dessen Einbettung in ein Ethernet-Frame. Die Abbildung 3-1 zeigt diese Einbettung eines EtherCAT-Frames.

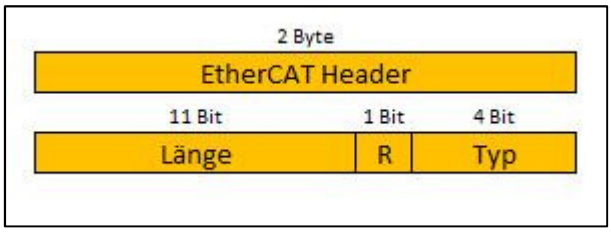


Quelle: [vgl. BECKHOFF New Automation Technology 2014, S. 4]

Abbildung 3-1: EtherCAT-Frame Struktur

Wie bereits erwähnt, werden Standard-Ethernet-Frames genutzt und das EtherCAT-Frame darin eingebettet. EtherCAT besitzt einen reservierten EtherType, der eine eindeutige Identifizierung von EtherCAT-Frames festlegt. Somit können parallel auch andere Ethernet Protokolle übertragen werden. [BECKHOFF New Automation Technology 2014]

Der EtherCAT-Header ist in drei Felder aufgeteilt, wie in Abbildung 3-2 zu sehen ist.



Quelle: [vgl. BECKHOFF New Automation Technology 2014, S. 4]

Abbildung 3-2: EtherCAT-Header

Das Längenfeld bezieht sich auf die gesamte Länge der EtherCAT-Datagramme ohne der FCS. Danach folgt ein reserviertes Bit. Abgeschlossen wird der EtherCAT-Header mit einem 4 Bit großem Typfeld. Der Typ legt fest, ob eine Kommunikation auf Feldebene oder zur Leitebene vorliegt und ist nicht zu verwechseln mit dem Ethertype im Ethernet-Header. Die EtherCAT Slave Controller werten nur den Typ „1“ aus. Die Typen 4 und 5 werden beispielsweise für die Master-Master Kommunikation verwendet.

Tabelle 1 zeigt die unterschiedlichen Typen und deren Bedeutung. [EtherCAT Technology Group 2018b, S. 22]

Tabelle 1: EtherCAT-Header Protokolltypen

Typ	Bedeutung
0	Reserviert
1	EtherCAT Geräte Kommunikation. Nur dieser Typ wird vom ESC ausgewertet
2/3	Reserviert
4	EAP Prozessdaten Kommunikation
5	EAP Mailbox Kommunikation
6-15	Reserviert

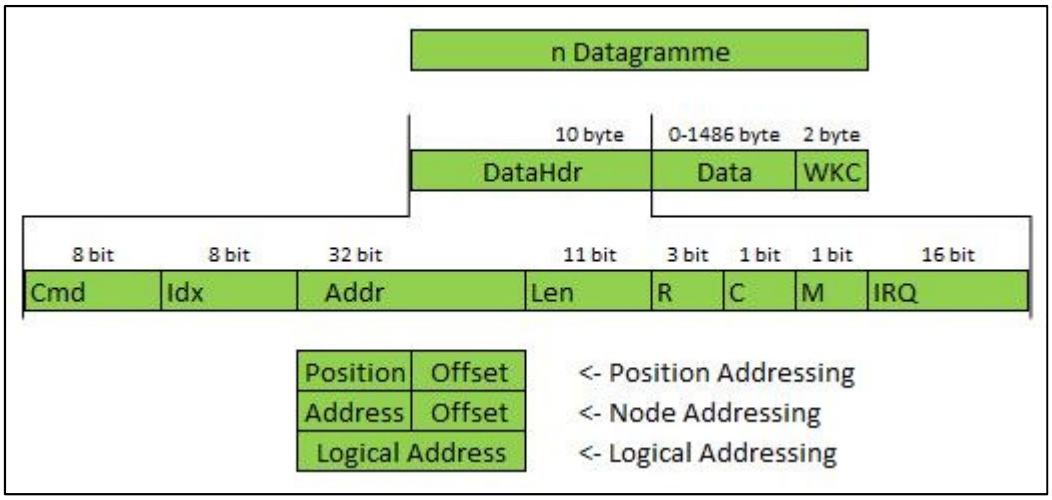
Quelle: [vgl. EtherCAT Technology Group 2018b, S. 22]

Die EtherCAT-Datagramme unterteilen sich in 3 Felder den EtherCAT-Datagram-Header, das Datenfeld und den Working Counter (WKC). Abbildung 3-3 stellt ein EtherCAT-Datagramm dar. Während Daten und Working Counter nicht weiter unterteilt werden, besteht der EtherCAT-Datagramm-Header aus acht weiteren Feldern [EtherCAT Technology Group 2018b, S. 24]:

- Cmd: Das Cmd Feld beinhaltet das EtherCAT Kommando (siehe Kapitel 3.4).
- Idx: Ein vom Master numerischer Index zur Identifizierung von doppelten oder verlorenen Datagrammen.
- Addr: Ein 4 Byte großes Adressfeld (siehe Kapitel 3.3).
- Len: Entspricht der Länge der Daten in dem jeweiligen Datagramm.
- R: Das Feld R ist ein reserviertes Feld mit dem Wert 0.
- C: Ein Bit, das angibt, ob das Frame zyklisch ist.

- M: Gibt an, ob nach diesem Datagramm noch weitere Datagramme kommen. Der Wert 0 steht für das letzte Datagramm.

- IRQ: Die EtherCAT Event Request Register von allen Slaves, die mit einem logischen ODER verknüpft sind.



Quelle: [vgl. BECKHOFF New Automation Technology 2014, S. 5]

Abbildung 3-3: EtherCAT-Datagramm

3.3 EtherCAT Adressierung

Die Geräte in einem EtherCAT Netzwerk können auf zwei Arten adressiert werden:

- Physikalische Adressierung (P)
- Logische Adressierung (L)

Bei der physikalischen Adressierung können die Slaves wiederum auf drei verschiedene Wege angesprochen werden:

- Position Address / Auto Increment Address (AP)
  Die Adresse des adressierten Slaves ist als Negativwert im Datagramm enthalten. Jeder Slave erhöht nun den Wert um eins. Der Slave, bei dem der Wert 0 ist, ist der adressierte Slave und führt den empfangen Befehl aus.
- Node Address / Configured Station Address and Configured Station Alias (FP)
  Der Slave wird entweder über die vom Master konfigurierte Stationsadresse oder über die im EEPROM gesetzte Station Alias adressiert.
- Broadcast (B)
  Es werden alle Slaves adressiert. Bei der Initialisierung und bei Zustandsabfragen wird die Broadcast Adressierung verwendet.

EtherCAT Geräte können bis zu zwei konfigurierte Geräteadressen haben. Eine davon wird bei der Initialisierung, vom Master zugewiesen, die andere ist im Slave Information Interface (SII) gespeichert und wird beim Start des Gerätes abgerufen. Die konfigurierte Adresse im SII kann geändert werden und wird auch konfigurierte Alias Adresse genannt. [BECKHOFF New Automation Technology 2014, S. 6]

Für die Logische Adressierung (L) steht in einem EtherCAT Netzwerk ein 4 GByte großes Prozessdatenabbild zur Verfügung, das von allen Geräten genutzt wird. Der Zugriff auf dieses Abbild erfolgt über die Fieldbus Memory Management Unit (FMMU). Weitere Erläuterungen zur FMMU in Kapitel 3.8. Logische Adressierung wird in der Regel bei zyklischer Prozessdatenkommunikation verwendet, da so der Overhead sehr geringgehalten wird. [BECKHOFF New Automation Technology 2014, S. 7]

3.4 EtherCAT Kommandos

Wie bei den Adressierungsmethoden gibt es vier Kommandotypen: Read (RD), Write (WR), ReadWrite (RW) und ReadMultipleWrite (RMW). Die Befehle werden mit den Adressierungen kombiniert. Somit ergeben sich folgende EtherCAT Kommandos, die in Tabelle 2 aufgelistet sind.

Tabelle 2: EtherCAT Befehlsübersicht

CMD	Abk.	Name
0	NOP	Keine Operation
1	APRD	Auto Increment Read
2	APWR	Auto Increment Write
3	APRW	Auto Increment Read Write
4	FPRD	Configured Address Read
5	FPWR	Configured Address Write
6	FPWR	Configured Address Read Write
7	BRD	Broadcast Read
8	BWR	Broadcast Write

9	BRW	Broadcast Read Write
10	LRD	Logical Memory Read
11	LWR	Logical Memory Write
12	LRW	Logical Read Write
13	ARMW	Auto Increment Read Multiple Write
14	FRMW	Configured Read Multiple Write
15 - 255		reserviert

Quelle: [vgl. BECKHOFF New Automation Technology 2014, S. 10]

3.5 Working Counter

Der Working Counter (WKC) gibt die Anzahl der Slaves wieder, indem jeder Slave bei korrekter Adressierung und verfügbaren Speicherbereich für den Datenzugriff den WKC automatisch um den dafür vorgesehen Wert erhöht. Der Wert setzt sich aus den Befehlsvorgängen zusammen. [Martin Rostan, EtherCAT Technology Group 2018, S. 20] Tabelle 3 zeigt die unterschiedlichen Inkrementierschritte bei verschiedenen Befehlstypen.

Tabelle 3: Working Counter Inkrements

Befehl		**Inkrement**
Lesen	Nicht erfolgreich	Kein Inkrement
	Erfolgreich gelesen	+1
Schreiben	Nicht erfolgreich	Kein Inkrement
	Erfolgreich geschrieben	+1
Lesen/Schreiben	Nicht erfolgreich	Kein Inkrement
	Erfolgreich gelesen	+1
	Erfolgreich geschrieben	+2

	Erfolgreich gelesen und geschrieben	+3

Quelle: [BECKHOFF New Automation Technology 2014, S. 8]

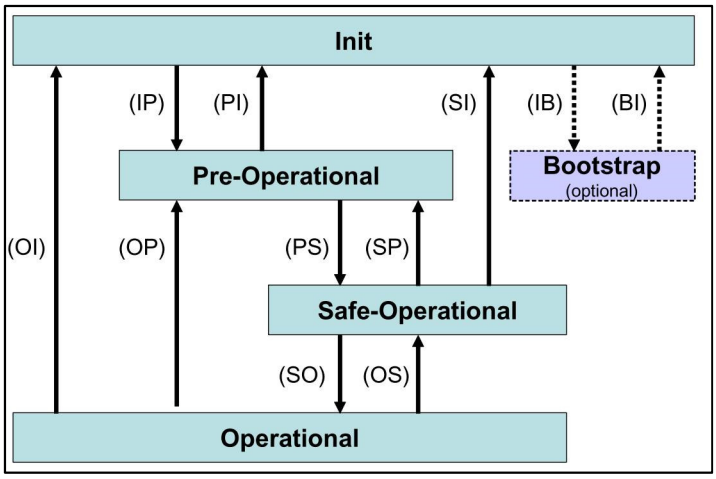
3.6 EtherCAT State Maschine

Die EtherCAT State Maschine (ESM) ist verantwortlich für die Koordination von Master- und Slaveanwendungen beim Start und während der Ausführung. Der Master leitet die Zustandswechsel der Slaves ein. Sobald ein Wechsel eingeleitet wurde, wird dieser nach Abschluss der damit verbundenen Aufgabe, beispielsweise der Konfiguration der Slavegeräte, bestätigt. Es gibt vier Zustände [BECKHOFF New Automation Technology 2014, S. 70]:

- Init (nach der Initialisierung)
- Pre-Operational (nach dem Mappen der PDOs)
- Safe-Operational (nach der Konfiguration von FMMU und SM)
- Operational (nach dem Senden eines gültigen Prozessdatenpakets)

Diese vier Zustände muss jedes EtherCAT Gerät unterstützten. Es gibt noch einen optionalen Zustand, den Bootstrap Zustand. Die erlaubten Zustandswechsel werden in Abbildung 3-4 dargestellt.

Die Übergänge sind in ihrer Reihenfolge klar definiert, weshalb der Übergang von Init zu Operational nur über die Zustände Pre-Operational und Safe-Operational möglich ist, jedoch der Übergang von Operational zu Init direkt möglich ist.



Quelle: [BECKHOFF New Automation Technology 2014, S. 70]

Abbildung 3-4: EtherCAT State Maschine, Anfangsbuchstaben der Zustände stellen die Abkürzungen der Übergänge dar

3.7 Slave Information Interface

Der EEPROM, ein nicht flüchtiger Speicherbaustein, in einem EtherCAT-Slave wird als Slave Information Interface (SII) bezeichnet. Das SII kann direkt vom Master gelesen werden und enthält Informationen zur Konfiguration der Fieldbus Memory Management Units, die SyncManager und dem Prozessdatenmapping. Zudem enthält das SII generelle Informationen wie Hersteller ID, Produkt ID, Seriennummer und Revisionsnummer. [BECKHOFF New Automation Technology 2014, S. 72]

3.8 Fieldbus Memory Management Unit

Eine Fieldbus Memory Management Unit (FMMU) ist im EtherCAT-Slave-Controller-Chip integriert und hat die Aufgabe "den physikalischen Adressen eines Teilnehmers eine logische Adresse zuzuordnen." [Dr.-Ing. Dirk Janssen et al. 2003, S. 3]. Dadurch wird ein individuelles bitorientiertes Adressmapping für jeden Teilnehmer möglich. Bereits in der Initialisierungsphase werden vom Master logische und physikalische Startadressen, Länge und Richtung des Mappings konfiguriert und somit komplette Prozessabbilder zusammengestellt. Damit ist durch eine FMMU möglich, dass mehrere Slaves sich ein Datagramm teilen. [Dr.-Ing. Dirk Janssen et al. 2003, S. 2]

3.9 SyncManager

Die Aufgabe eines SyncManager ist es, einen konsistenten und sicheren Datenaustausch zwischen dem Master und einer lokalen Anwendung (Sensoren, Aktuatoren) zu ermög-

lichen. Sie erzeugen Interrupts um Master- und Slaveanwendung über Änderungen zu informieren. Über einen Puffer, der sich im Speicherbereich befindet, werden die Daten ausgetauscht. Der Zugriff auf diesen Puffer wird von der Hardware kontrolliert. Die SyncManager werden hinsichtlich der Kommunikationsrichtung und des Kommunikationsmodus (Buffered Mode, Mailbox Mode) vom Master konfiguriert. Der Buffered Mode wird für die Übertragung zyklischen Prozessdaten verwendet. Dieser Modus erlaubt dem Master, sowie dem Aktuator zu jeder Zeit Zugriff auf den Kommunikationspuffer. Der Mailbox Mode wird typischerweise für die Protokolle der Anwendungsschicht angewendet, denn hier wird ein Handshake Mechanismus für den Datenaustausch verwendet, sodass keine Daten verloren gehen. Der Zugriff auf den Puffer ist erst dann möglich, wenn Master oder lokale Anwendung ihren Zugriff beendet haben. [BECKHOFF New Automation Technology 2014, S. 41]

4 Aufbau

Für die Realisierung der Kommunikation über EtherCAT sind ein Master, die Slaves und die Software für den Master notwendig. Im Folgenden werden die einzelnen Komponenten erläutert, die für die Umsetzung nötig sind.

Da für die Realisierung des Masters keine zusätzliche Hardware nötig ist, wird ein Laptop verwendet, auf dem ein Windows 10 Betriebssystem installiert ist. Als Entwicklungsumgebung wird Visual Studio 2015 Professional (VS) verwendet und die Programmiersprache C. In VS bilden die drei Projekte „SOEMLib“, „Slaveinfo“ und „ECAT_Test“ die Projektmappe. Das Projekt „Slaveinfo“ ist eine von Simple Open EtherCAT Society [OpenEtherCATsociety 2019] programmierte Testanwendung zum Auslesen von Informationen und Mappings der Slave Geräte. Das Einbinden des SOEM und die notwendigen Einstellungen werden im Kapitel 4.1 erläutert.

Als Slave dienen mehrere Geräte der Firma WAGO. Die WAGO Input und Output Klemmen haben keine eigene MAC-Adresse. Sie werden über den WAGO EtherCAT Buskoppler angesprochen, der den ESC darstellt. Der Zusammenschluss des Feldbuskopplers und der Busklemmen wird auch EtherCAT Segment genannt. Deswegen werden alle Klemmen als ein Slave betrachtet. Der Zugriff bei den WAGO Klemmen geschieht über ein internes Mapping, dass die SOEM Bibliothek in der Initialisierungphase durchführt. Der Laptop (Master) und der WAGO Buskoppler sind mit einem CAT 5e Netzwerkkabel verbunden. Es sind sechs Klemmen an dem Buskoppler angesteckt. Je eine analoge und digitale Input Klemme und eine analoge und digitale Output Klemme. Eine zusätzliche Klemme für die Spannungseinspeisung und die Endklemme. Die genauen Bezeichnungen und die Reihenfolge der Klemmen sind folgendermaßen:

- WAGO EtherCAT Buskoppler 750-354
- WAGO 4-Channel analog input module (750-457)
- WAGO 4-Channel analog output module (750-557)
- Supply module passive DC 24 V (750-602)
- WAGO 8-Channel digital input module DC 24 V 3,0 ms (750-430)
- WAGO 8-Channel digital output module DC 24 V 0,5 A (750-530)
- End module I/O module for testing (750-600)

Ein Oszilloskop dient zur Veranschaulichung der gesendeten Daten. Dabei wird mit dem Oszilloskop eine generierte Kurve an der analoge Eingangsklemme eingelesen und an die analoge Ausgangsklemme gesendet.

In Abbildung 4-1 geht hervor, wie die einzelnen Komponenten verbunden sind und in welche Richtungen die Signale verlaufen.

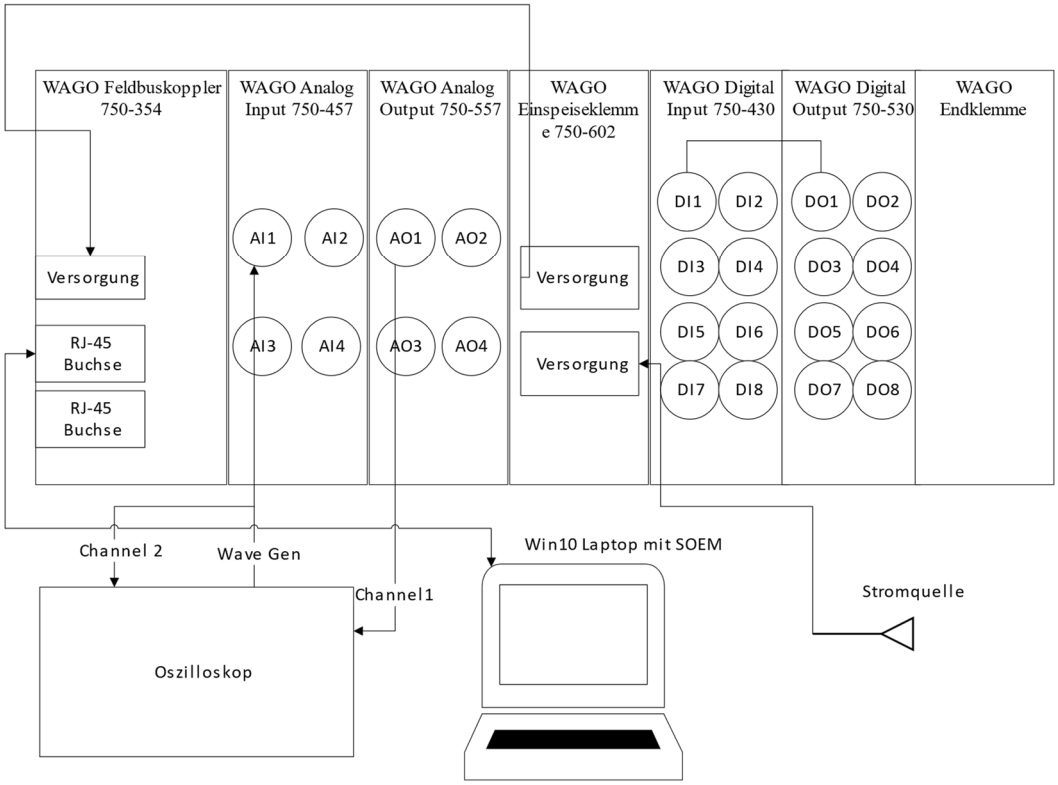


Abbildung 4-1: Vernetzung der Gesamtstruktur

4.1 Vorbereitungen

Um die Funktionen von Simple Open EtherCAT Master zu verwenden, wird eine statische Library in VS erstellt und diese dann in eine Konsolenanwendung eingebunden. Die nötigen Abhängigkeiten, die hier zu beachten sind, und die Vorgehensweise werden im Folgenden kurz beschrieben.

Es wurde ein Win32Projekt erstellt, bei dem der Anwendungstyp eine statische Library ist. Danach wurde die gesamte Ordnerstruktur des Simple Open EtherCAT Masters in dieses Projekt eingefügt. Um alle Unterverzeichnisse in das Projekt einzubinden werden zusätzliche Includeverzeichnisse benötigt, die in VS in den Projekteigenschaften C/C++ unter Allgemein eingetragen werden müssen. Diese Einstellung macht das kom-

pilieren der Bibliothek möglich. Die Abbildung 4-2 zeigt diesen Schritt und die einzutragenden Verzeichnisse. Im Anschluss kann die statische Library erstellt werden.

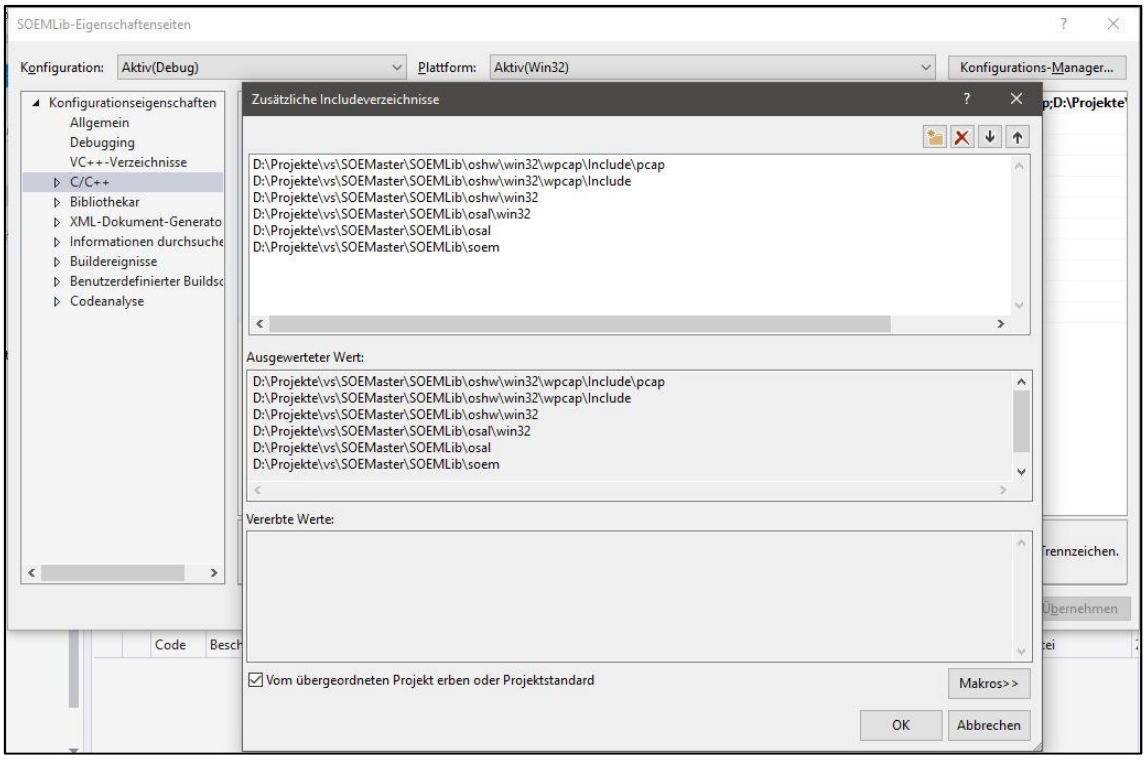


Abbildung 4-2: Zusätzliche Includes für die SOEM Library

Nachdem die SOEMLib erstellt ist und somit als Bibliothek zur Verfügung steht, können die Einstellungen der Testanwendung erläutert werden, um das Kompilieren und Linken zu ermöglichen.

Für die „ECAT_Test“- und „Slaveinfo“-Anwendung müssen, wie auch bei dem Projekt „SOEMLib“, die gleichen zusätzlichen Includeverzeichnisse eingetragen werden, damit das Kompilieren der Anwendung möglich ist. Die Ausführung und das Linken benötigt jedoch zusätzliche Abhängigkeiten und Bibliotheksverzeichnisse. In den Linkereinstellungen der Projekteigenschaften unter dem Reiter Eingabe müssen die Abhängigkeiten eingetragen werden, die in Abbildung 4-3 zu sehen sind. Unter ihnen ist auch das Endprodukt der erstellten „SOEMLib“-Anwendung, die SOEMLib.lib.

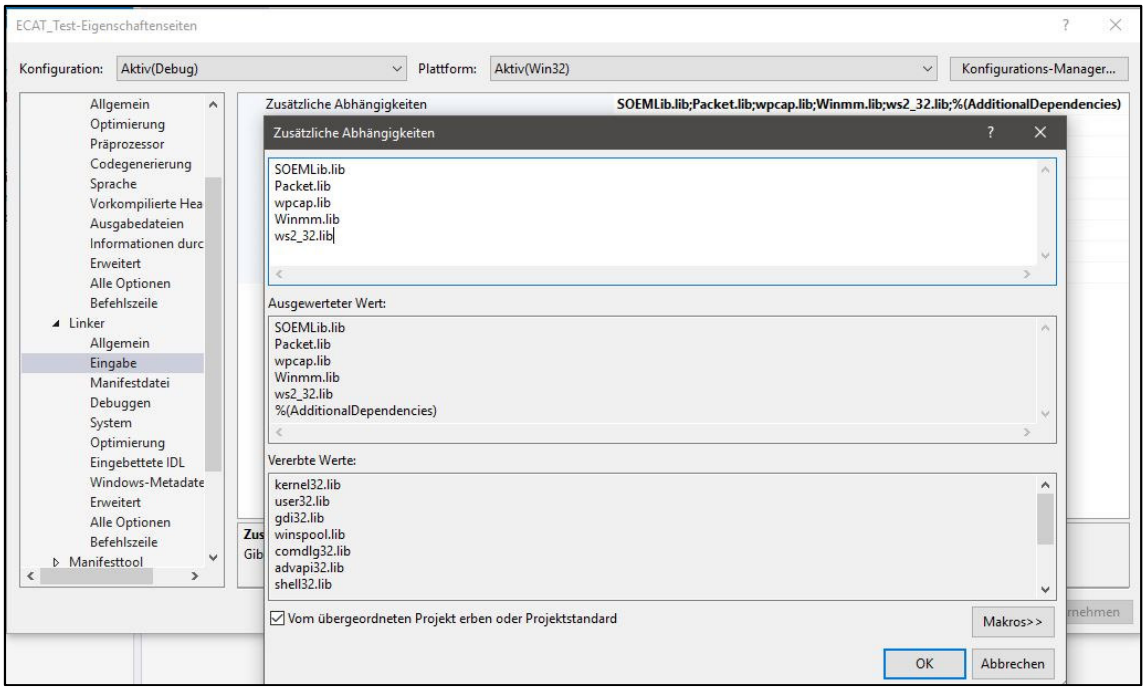


Abbildung 4-3: EtherCAT Programm Abhängigkeiten

Als nächstes werden Bibliotheksverzeichnisse in den Projekteigenschaften Linker unter Eingabe hinterlegt (siehe Abbildung 4-4). Die SOEMLib.lib und zwei zusätzliche Libraries (packet.lib, wpcap.lib aus dem Verzeichnis SOEMLib\oshw\win32\wpcap\Lib) müssen eingetragen werden, damit das Linking ordnungsgemäß ausgeführt werden kann.

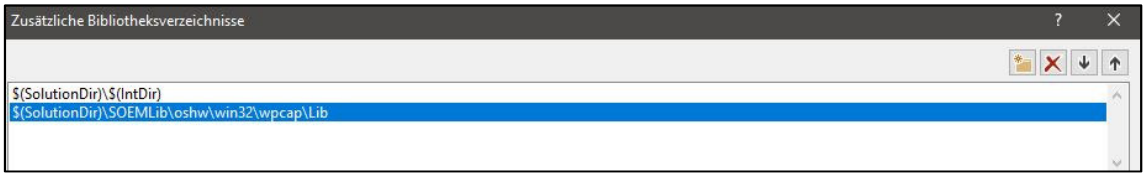


Abbildung 4-4: Zusätzliche Bibliotheksverzeichnisse

Für die Verwendung des SOEM muss unter Windows WinPcap installiert sein. WinPcap ist eine als Freeware vertriebene Programmbibliothek, bestehend aus einem Treiber, der hardwarenahen Zugriff auf die Netzwerkkarte ermöglicht, und einer Sammlung von Programmen, die den Zugriff auf die einzelnen für Netzwerke relevanten Schichten des OSI-Modells bieten.

5 Implementierung und Verwendung der SOEM Bibliothek

In diesem Kapitel wird auf die Verwendung des SOEM eingegangen und wie dessen Funktionen genutzt werden können, beginnend mit der Initialisierung des SOEM und endend mit dem Schließen des von SOEM erstellten Sockets. Der erste Ausschnitt des Programmcodes ist in Listing 1 dargestellt.

```
168     /* SOEM initialisieren, Socket an ifname binden */
169     if (ec_init(ifname))
170     {
171         /* Konfiguration der Slave-Geräte */
172         if (ec_config_init(FALSE) > 0)
173         {
174             /* Slaves befinden sich in Pre-Operational Zustand */
175             /* Konfiguration der IOMap und der Verteilten Uhren */
176             ec_config_map(&IOmap);
177             ec_configdc();
178 
179             /* Prüfe Slaves auf Safe-Operational Zustand */
180             ec_statecheck(ECAT_MASTER, EC_STATE_SAFE_OP, EC_TIMEOUTSTATE * 4);
181 
182             /*
183                 ExpectedWKC ist für die Fehlerbehandlung der Slaves erfolderlich,
184                 dieser muss immer dem WKC entsprechen.
185             */
186             expectedWKC = (ec_group[0].outputsWKC * 2) + ec_group[0].inputsWKC;
187 
188             /* Senden eines gültigen Prozessdatenpaket zum Initialisieren der SM's */
189             ec_send_processdata();
190             wkc = ec_receive_processdata(EC_TIMEOUTRET);
191 
192             /* Leite OP Zustand für alle Slaves ein */
193             ec_slave[ECAT_MASTER].state = EC_STATE_OPERATIONAL;
194             ec_writestate(ECAT_MASTER);
195 
196             /* Prüfe Slaves auf Operational Zustand */
197             ec_statecheck(ECAT_MASTER, EC_STATE_OPERATIONAL, EC_TIMEOUTSTATE);
198 
199             /* Zuordnung des Mappings und der Strukturen */
200             outs = (wagostruc_in_out *)ec_slave[WAGO_750_354].outputs;
201             ins = (wagostruc_in_out*)ec_slave[WAGO_750_354].inputs;
```

Listing 1: Initialisierung und Konfiguration

Um die Netzwerkkarte des Windowslaptops zu konfigurieren, sodass eine Kommunikation über EtherCAT möglich ist, wird die Funktion *ec_init(ifname)*, die in dem SOEM zu finden ist, aufgerufen (Listing 1, Z. 169). Die Namensgebung der Funktionen des SOEM beginnt mit ec was für ethercat steht. Sie übernimmt die Initialisierung des SOEM und erstellt einen Socket, über den die Datenpakete gesendet werden (als Argument wird der Gerätename der Netzwerkkarte übergeben).

Nachdem das Ethernet-Interface des Masters nun die Kommunikation über EtherCAT unterstützt, werden mit der Funktion *ec_config_init(FALSE)* alle Slaves durch einen Broadcast Read (BRD) angesprochen. Slaves, die voll funktionsfähig sind und sich im Netzwerk befinden antworten auf den BRD, und der Master registriert die Anzahl der

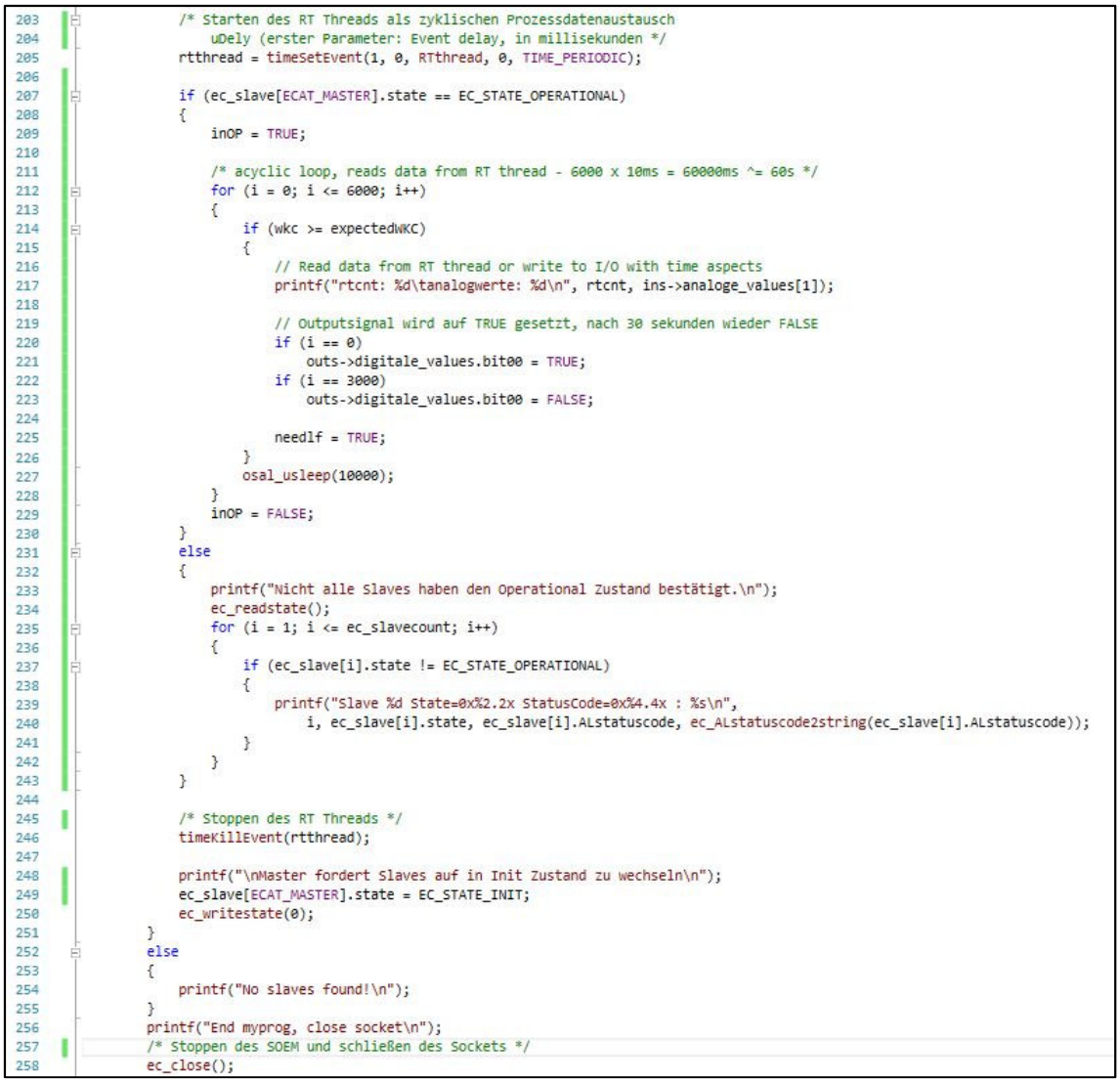
Slaves. Alle gelesenen und konfigurierten Daten werden in einer globalen Struktur gespeichert, dem *ec_slave*. Die Struktur *ec_slave* beinhaltet die Hauptinformationen der Slaves. Der Master kennt nun alle EtherCAT-Slave-Geräte, die im selben Netzwerk sind und hat diese konfiguriert.

Als nächstes wird über die Funktion *ec_config_map(&IOmap)* (Listing 1, Z. 176) ein Mapping der Inputs und Outputs erstellt sowie die FMMU's konfiguriert, um den EtherCAT-Master und die Slaves zu verknüpfen. Sobald das Mapping durchgeführt wurde fordert der Master die Slaves auf in den SAFE_OP Zustand zu wechseln, der daraufhin mit der Funktion *ec_statecheck()* überprüft wird (Listing 1, Z. 197).

Um das System in Betrieb nehmen zu können, müssen alle Slaves in den OP Zustand wechseln. Dies erfolgt mit dem Senden eines gültigen EtherCAT-Frames. Die Funktionen *ec_send_processdata()* und *ec_receive_processdata()* senden ein EtherCAT Frame aus und empfangen das gleiche wieder.

Am Ende von Listing 1 werden den I/Os Strukturen für den Zugriff zugeordnet (Listing1, Z. 200-201). Die Strukturen werden in Listing 3 definiert.

Die eigentliche Echtzeitausführung wird ebenfalls in einem zusätzlichen periodischen Thread ausgelagert. In Listing 2, Zeile 205, wird dieser Thread gestartet. Durch eine Schleife zwischen dem Starten und Stoppen des Realtime Threads wird das Programm nicht beendet und es können I/O Daten aus dem RT-Thread gelesen und geschrieben werden. Es wird zu Beginn der azyklischen Schleife der erste digitale Ausgang auf TRUE gesetzt und nach 30 Sekunden wieder zurück auf FALSE. Mit der Abfrage in Listing 2, Zeile 207 wird sichergestellt, dass sich alle Slaves im OP Zustand befinden. Ist ein Slave nicht im OP Zustand wird ein Status des Slaves ausgegeben. Die Letzte Zeile schließt den Socket und beendet den SOEM.



```
203             /* Starten des RT Threads als zyklischen Prozessdatenaustausch
204                 uDely (erster Parameter: Event delay, in millisekunden */
205             rtthread = timeSetEvent(1, 0, RTthread, 0, TIME_PERIODIC);
206
207             if (ec_slave[ECAT_MASTER].state == EC_STATE_OPERATIONAL)
208             {
209                 inOP = TRUE;
210
211                 /* acyclic loop, reads data from RT thread - 6000 x 10ms = 60000ms ^= 60s */
212                 for (i = 0; i <= 6000; i++)
213                 {
214                     if (wkc >= expectedWKC)
215                     {
216                         // Read data from RT thread or write to I/O with time aspects
217                         printf("rtcnt: %d\tanalogwerte: %d\n", rtcnt, ins->analoge_values[1]);
218
219                         // Outputsignal wird auf TRUE gesetzt, nach 30 sekunden wieder FALSE
220                         if (i == 0)
221                             outs->digitale_values.bit00 = TRUE;
222                         if (i == 3000)
223                             outs->digitale_values.bit00 = FALSE;
224
225                         needlf = TRUE;
226                     }
227                     osal_usleep(10000);
228                 }
229                 inOP = FALSE;
230             }
231             else
232             {
233                 printf("Nicht alle Slaves haben den Operational Zustand bestätigt.\n");
234                 ec_readstate();
235                 for (i = 1; i <= ec_slavecount; i++)
236                 {
237                     if (ec_slave[i].state != EC_STATE_OPERATIONAL)
238                     {
239                         printf("Slave %d State=0x%2.2x StatusCode=0x%4.4x : %s\n",
240                             i, ec_slave[i].state, ec_slave[i].ALstatuscode, ec_ALstatuscode2string(ec_slave[i].ALstatuscode));
241                     }
242                 }
243             }
244
245             /* Stoppen des RT Threads */
246             timeKillEvent(rtthread);
247
248             printf("\nMaster fordert Slaves auf in Init Zustand zu wechseln\n");
249             ec_slave[ECAT_MASTER].state = EC_STATE_INIT;
250             ec_writestate(0);
251         }
252         else
253         {
254             printf("No slaves found!\n");
255         }
256         printf("End myprog, close socket\n");
257         /* Stoppen des SOEM und schließen des Sockets */
258         ec_close();
```

Listing 2: Starten und Stoppen des Realtime Threads

Wie bereits erwähnt, wird eine Struktur definiert, die den analogen und digitalen Output- und Inputklemmen entsprechen. In diesem Fall ist eine Struktur für Outputs und Inputs ausreichend, weil die Ein- und Ausgänge gleichermaßen verteilt sind. Wie in Listing 3 zu sehen, gibt es für jeden Ausgang/Eingang der Klemme eine entsprechende Variable. Bei den Analog Klemmen handelt es sich um vier Kanäle, die angesprochen werden können. Die digitalen Aus- und Eingänge werden durch ein Bitfeld repräsentiert. In Zeile 44 werden zwei Zeiger für den Zugriff definiert, die bereits in Listing 1 zugewiesen wurden.

```
24  typedef struct PACKED
25  {
26      /* Reserviert 4 Byte für den WAGO Feldbuskoppler */
27      uint32 reserved;
28      /* Analogklemmen mit 4 Kanälen */
29      uint16 analoge_values[4];
30      /* Digitalklemmen mit 8 Kanälen */
31      struct bitfeld_8
32      {
33          BOOLEAN bit00 : 1;
34          BOOLEAN bit01 : 1;
35          BOOLEAN bit02 : 1;
36          BOOLEAN bit03 : 1;
37          BOOLEAN bit04 : 1;
38          BOOLEAN bit05 : 1;
39          BOOLEAN bit06 : 1;
40          BOOLEAN bit07 : 1;
41      } digitale_values;
42  } wagostruc_in_out;
43
44  wagostruc_in_out *outs, *ins;
```

Listing 3: Struktur der WAGO-Klemmen Analog Input/Output

Der Kern dieses Programms ist der Realtime (RT)-Thread, bei dem die Prozessdaten mit einer Zykluszeit von einer Millisekunde ausgetauscht werden. In Listing 6 wird der RT-Thread dargestellt.

```
46  /* Realtime Thread für die Prozessdatenkommunikation */
47  void CALLBACK RTthread(PVOID lpParam)
48  {
49      ec_send_processdata();
50      wkc = ec_receive_processdata(EC_TIMEOUTRET);
51
52      /* Analoge Signale (Sinuskurve) */
53      /* Beide analogen Klemmen auf Kanal 1 */
54      outs->analoge_values[1] = ins->analoge_values[1];
55
56      /* Digitale Signale */
57      // nach 30 sekunden wird das outputsignal auf Kanal1 wieder auf FALSE gesetzt in der azyklischen Schleife
58      // wenn digital channel 1 TRUE, schalte 4 - 8 auch TRUE, sonst 4 - 8 FALSE
59      if (ins->digitale_values.bit00)
60      {
61          outs->digitale_values.bit04 = TRUE;
62          outs->digitale_values.bit05 = TRUE;
63          outs->digitale_values.bit06 = TRUE;
64          outs->digitale_values.bit07 = TRUE;
65      }
66      else
67      {
68          outs->digitale_values.bit04 = FALSE;
69          outs->digitale_values.bit05 = FALSE;
70          outs->digitale_values.bit06 = FALSE;
71          outs->digitale_values.bit07 = FALSE;
72      }
73
74      /* Syncronisieren des WAGO Buskopplers */
75      ec_dcsync0(WAGO_750_354, TRUE, 1000000U, 0);
76
77      /* rtcnt: Zählt die Prozesszyklen */
78      rtcnt++;
79  }
```

Listing 4: Der Realtime Thread

In dem RT-Thread werden die Funktionen des SOEMs zum Senden und Empfangen der Prozessdaten aufgerufen. Die Zeile 54 zeigt die Zuweisung des analogen Wertes der

Inputklemme und schreibt diesen auf den analogen Ausgang. Bei beiden Klemmen wird der Kanal 1 verwendet. Durch die Struktur kann mithilfe eines Zeigers auf die Werte zugegriffen werden. Des Weiteren wurde ein Test mit den digitalen Signalen durchgeführt. Wenn ein Signal auf Kanal 1 der digitalen Eingangsklemme registriert wird, werden Signale auf den Kanälen 4 bis 8 der digitalen Ausgangsklemme gesetzt.

6 Ergebnisse

Die Darstellung der Ergebnisse zur Kommunikation über EtherCAT wird mit einem Oszilloskop und mit dem Paketsniffer Wireshark dargestellt.

6.1 Oszilloskop

In dem Oszilloskop kann eine Sinuskurve über eine Funktion generiert werden und als Output in die Analoge Inputklemme eingespeist werden. Dieses Signal wird abgefangen und geht in Kanal 1 des Oszilloskops zurück (siehe Abbildung 4-1). Des Weiteren wird das über EtherCAT verarbeitete Output-Signal auf Kanal 2 am Oszilloskop ausgegeben. Die Abbildung 6-1 zeigt die generierte Sinuskurve (grün) und die mit EtherCAT verarbeitete Sinuskurve (gelb).

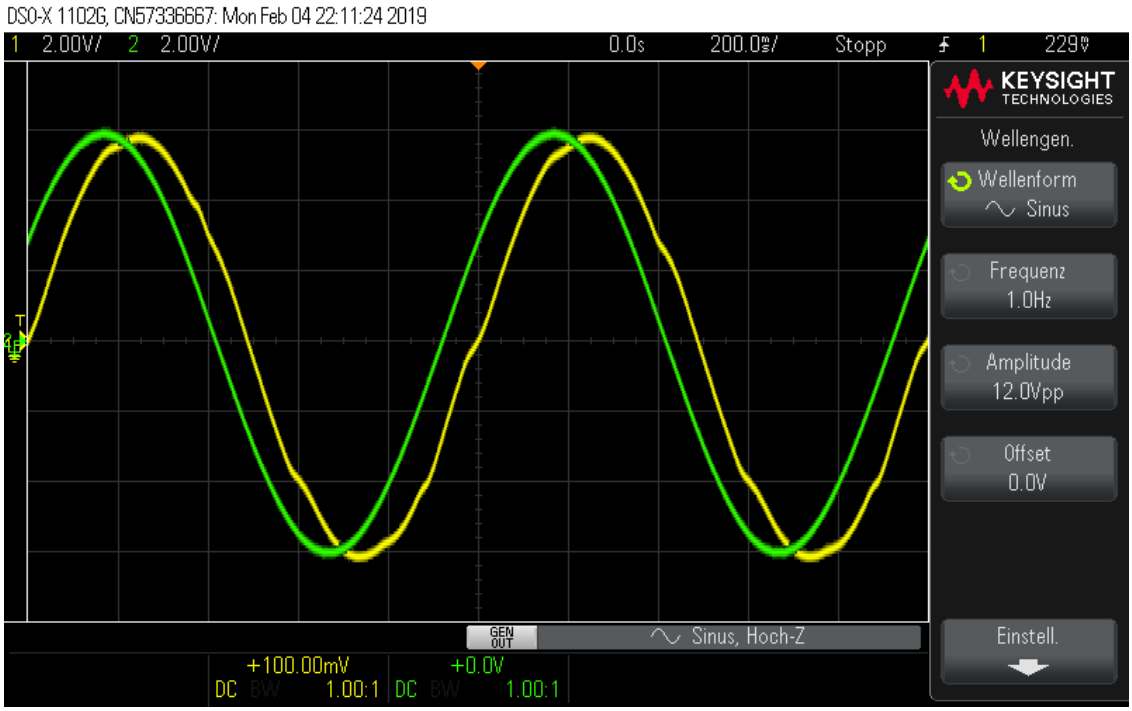


Abbildung 6-1: Am Oszilloskop generiert Sinuskurve und über EtherCAT zurück ans Oszilloskop gesendete Sinuskurve

Es sind kurze Ausschweifungen in der gelben Sinuskurve zu sehen. Diese sind durch die in den analogen Eingangs- und Ausgangsklemmen bedingte Wandlungszeit von 10 ms zu erklären.

6.2 Wireshark

Eine Analyse der Netzwerkdaten mit Wireshark bestätigt die Kommunikation über EtherCAT. Es werden drei Pakete gezeigt:

1. Das vom Master ausgesendete Paket ohne Daten (Abbildung 6-2)
2. Das Zurückkehrende Paket aus 1., das Daten der Inputklemme enthält (Abbildung 6-3)
3. Das nächste vom Master ausgesendete Paket, das Inputdaten und Outputdaten enthält (Abbildung 6-5)

Die Abbildung 6-2 zeigt das erste Paket. Wie in Kapitel 3.2 beschrieben, zeigt auch der Paketsniffer Wireshark die Struktur eines EtherCAT Frames. Ein Ethernet_II-Frame und das darin eingebettete EtherCAT-Frame, welches aus EtherCAT Header und EtherCAT Datagrammen besteht. Es befinden sich zwei Datagramme in dem Paket. Beide Datagramme weisen einen Working Counter von 0 auf, sodass festzuhalten ist, dass dieses Paket bisher von keinem Slave bearbeitet wurde. Betrachtet man das erste Datagramm ist ein Logical Read Write Befehl zu erkennen. Logische Adressierung ist kennzeichnend für den zyklischen Prozessdatenaustausch. Außerdem enthält dieses Paket keine Daten.

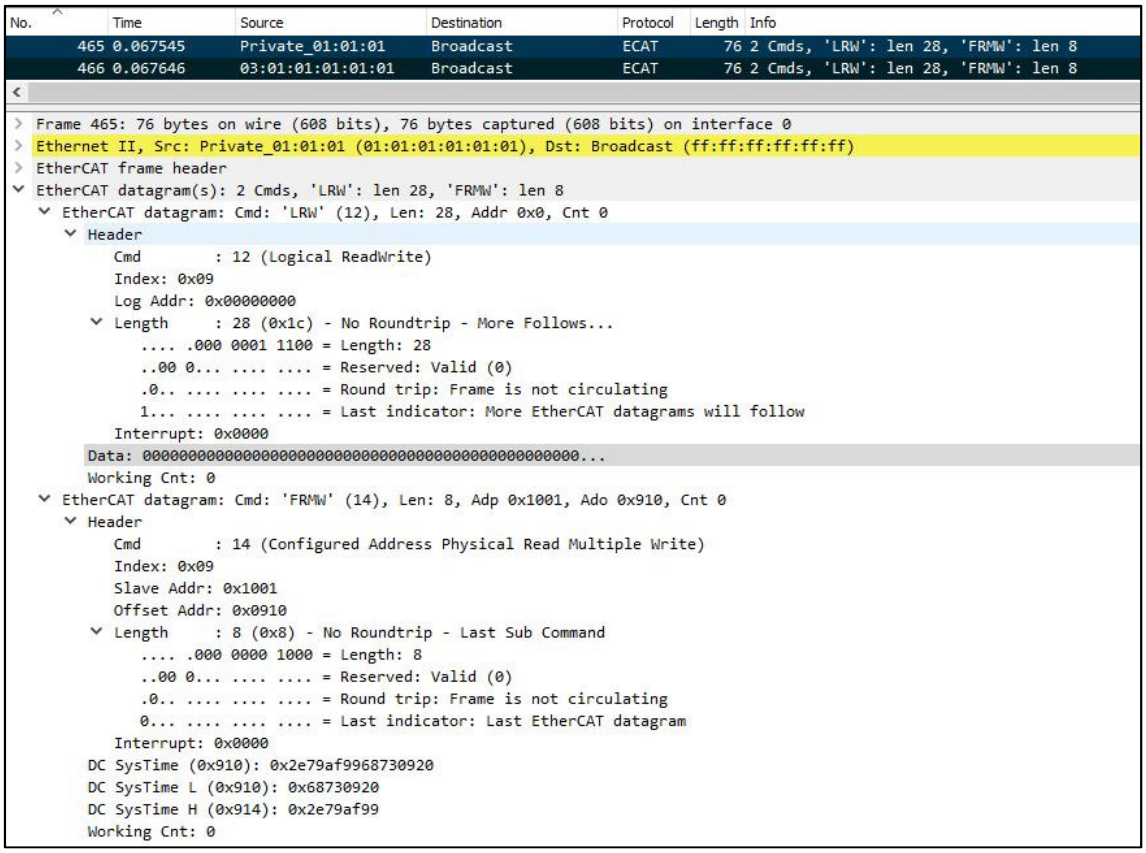


Abbildung 6-2: Wireshark EtherCAT Paket 1

Das Paket in Abbildung 6-3 ist das von den Slaves modifizierte Paket. Durch die Bearbeitung der Slaves wird der WKC inkrementiert, sodass dieser hier „3“ ist. Der angegebene Befehl ist ein LRW, der bei erfolgreichen lesen und schreiben den WKC um zwei erhöht. Der bearbeitende Slave hat Eingangsdaten registriert und sie in das Datagramm eingefügt. Die Zeichenfolge „a8b8“ ist der Hexadezimalwert, der von dem Oszilloskop gesendeten Sinuskurve.

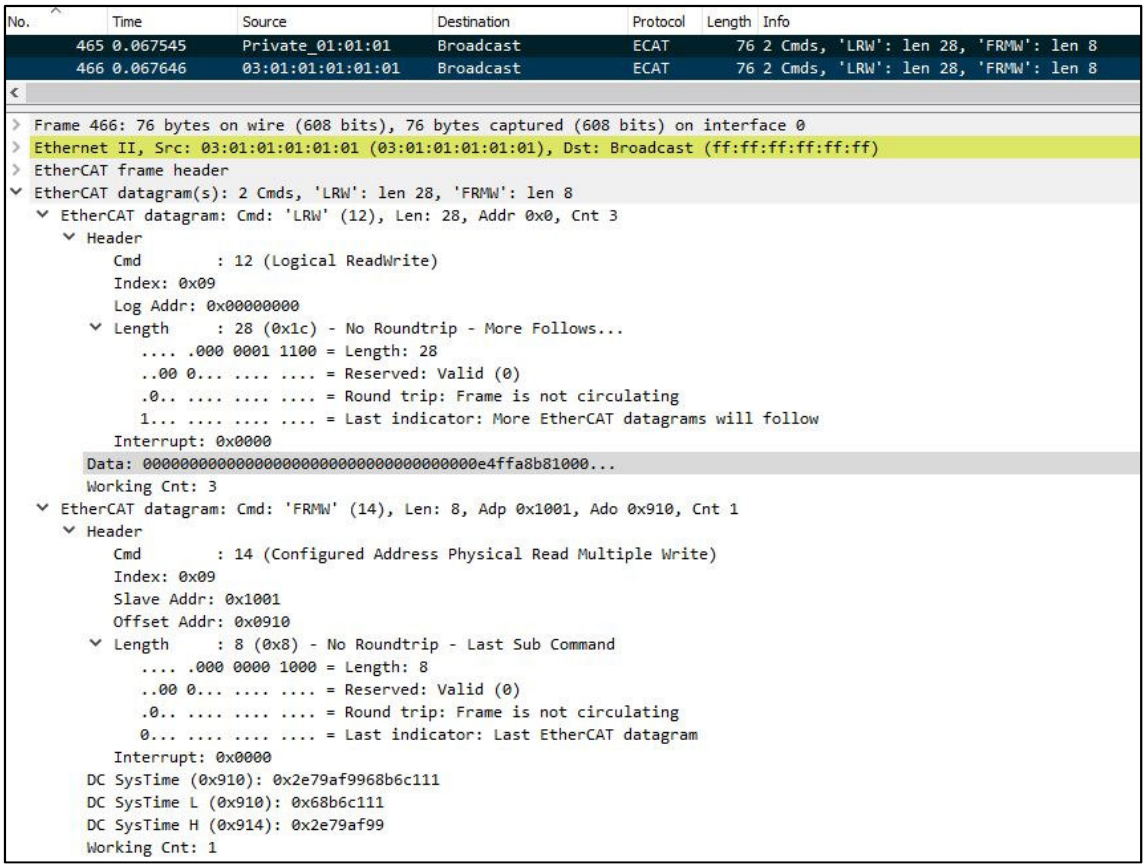


Abbildung 6-3: Wireshark EtherCAT Packet 2

Der Ausschnitt des laufenden Programms (Abbildung 6-4) zeigt den analogen Wert „b8a8“. Durch unterschiedliche Darstellungsmöglichkeiten von Hexadezimalwerten, muss einer der Werte umgestellt werden, um sie miteinander vergleichen zu können.

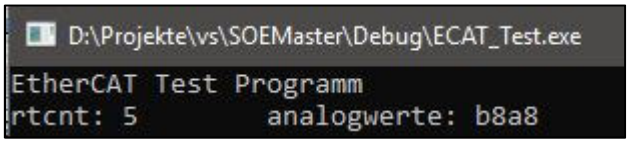


Abbildung 6-4: Laufendes Programm mit Ausgabe des übermittelten Analogwertes

Der Analoge Wert soll über die Inputklemme zurück an das Oszilloskop gesendet werden. Die Abbildung 6-5 zeigt den vorangegangen Input und Output des dritten Pakets.

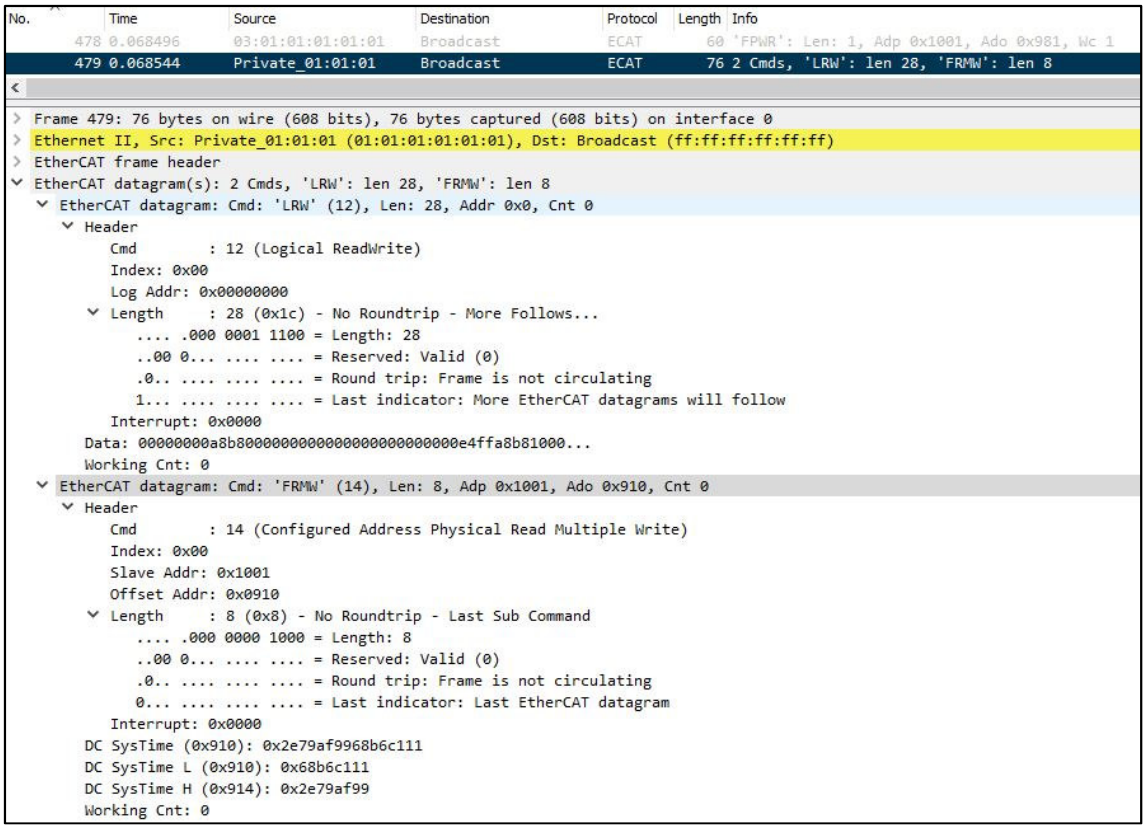


Abbildung 6-5: Wireshark EtherCAT Packet 3

Zu den hexadezimal dargestellten Daten ist festzuhalten, dass diese aus 28 Bytes bestehen. Das sind umgerechnet 224 Bits, welche die Anzahl der Inputs und Outputs der WAGO Klemmen wiederspiegeln.

7 Zusammenfassung und Ausblick

Zusammenfassend kann festgehalten werden, dass EtherCAT eine revolutionäre Entwicklung für die Automatisierungstechnik ist. Sie ist mit der einfachen Verkabelung durch Twisted Pair Kabel und der Tatsache, dass für den Master eine einfache Netzwerkkarte ausreicht, schnell und kostengünstig umzusetzen. Es ist keine aufwendige manuelle Konfiguration der Adresseinstellungen nötig. EtherCAT unterstützt sämtliche Topologien, wodurch die Flexibilität gewährleistet ist. EtherCAT ist außerdem eine offene Technologie und ihre Echtzeitfähigkeit geht über die Möglichkeiten anderen Industrial Ethernet Lösungen und den traditionellen Feldbussen hinaus.

Dies spricht nicht nur für die Automatisierungstechnik in Fabriken, sondern auch für „Hobby-Entwickler“, die Hausautomatisierung auf privater Ebene betreiben wollen. EtherCAT ermöglicht die Übertragung der zyklischen Prozessdaten und eine azyklische Kommunikation über höhere Protokolle.

Mit dem SOEM und der Voraussetzung, dass für den EtherCAT Master eine einfache Netzwerkkarte ausreicht, besteht die Möglichkeit auch auf Linux Basis einen EtherCAT Master zu entwickeln und diesen auf einem Raspberry PI laufen zu lassen. Dieser kann die Steuerung der Hausautomatisierung übernehmen. Für den Raspberry gibt es einen Aufsatz von Hilscher (netX), der kostengünstig zu erwerben ist und die EtherCAT Kommunikation unterstützt.

Ein Interessanter Ansatz wäre EtherCAT-Geräte über Wireless LAN kommunizieren zu lassen und den Determinismus in Frage zu stellen. Beispielsweise welche Zykluszeiten für eine wireless Übertragung realistisch wären. Ein weiterer Ansatz wäre, das 5G Netz mit einzubinden, um niedrige Reaktionszeiten zu erhalten.

Literaturverzeichnis

[Airnet Technologie- und Bildungszentrum GmbH 2009]
Airnet Technologie- und Bildungszentrum GmbH; *Basic Internetworking,* 2009, https://www.airnet.de/basic_internetworking1/de/html/index.xml. Abgerufen am 12.02.2019.

[BECKHOFF New Automation Technology]
BECKHOFF New Automation Technology; *BECKHOFF New Automation Technology,* https://www.beckhoff.de/default.asp?ethercat/default.htm. Abgerufen am 04.12.2018.

[BECKHOFF New Automation Technology 2014]
BECKHOFF New Automation Technology; *EtherCAT Slave Controller: Technology EtherCAT Protocol, Physical Layer, EtherCAT Processing Unit, FMMU, SyncManager, SII EEPROM, Distributed Clocks,* 2014.

[Buchholz 2018]
Buchholz, Andree; *Industrial Ethernet überholt Feldbusse*, in: PC & Industrie, 2018.

[Dipl.-Ing. Martin Rostan 2003]
Dipl.-Ing. Martin Rostan; *Ethernet bis in die Reihenklemme,* 2003.

[Dopatka 2008]
Dopatka, Frank; *Ein Framework für echtzeitfähige Ethernet-Netzwerke in der Automatisierungstechnik mit variabler Kompatibilität zu Standard-Ethernet,* 2008.

[Dr.-Ing. Dirk Janssen et al. 2003]
Dr.-Ing. Dirk Janssen; Holger Büttner; *EtherCAT,* 2003.

[EtherCAT Technology Group 2018a]
EtherCAT Technology Group; *EtherCAT Technology Group | EtherCAT,* 02.11.2018, https://www.ethercat.org/de/. Abgerufen am 04.12.2018.

[EtherCAT Technology Group 2018b]
EtherCAT Technology Group; *EtherCAT Communication,* 2018.

[IEEE]
IEEE; *IEEE Standard for Ethernet,* Piscataway, NJ, USA.

[KINGSTAR 2015]
KINGSTAR; *Best_Ethernet_Protocol_for_Your_PLC: 5 Fieldbuses Compared For Industry 4.0,* 2015/2016.

[Klasen et al. 2010]
Klasen, Frithjof; Oestreich, Volker; Volz, Michael (Hrsg.); *Industrielle Kommunikation mit Feldbus und Ethernet,* Berlin. VDE-Verl., 2010.

[Martin Rostan, EtherCAT Technology Group 2018]
Martin Rostan, EtherCAT Technology Group (ETG); *EtherCAT - Der Ethernet Feldbus,* 2018.

[Open EtherCAT Society 2017]
Open EtherCAT Society; *Home of SOEM and SOES,* 13.12.2017, https://openethercatsociety.github.io/. Abgerufen am 02.01.2019.

[OpenEtherCATsociety 2019]
OpenEtherCATsociety; *OpenEtherCATsociety/SOEM,* 01.02.2019, https://github.com/OpenEtherCATsociety/SOEM/. Abgerufen am 14.02.2019.

[Prof. Dr. M. Pellkofer 2017]
Prof. Dr. M. Pellkofer; *Prozessrechentechnik,* 2017/18.

[Rech 2008]
Rech, Jörg; *Ethernet: Technologien und Protokolle für die Computervernetzung ; [Standard-Ethernet ; Fast Ethernet ; Gigabit-Ethernet ; 10Gigabit-Ethernet ; Power over Ethernet,* 2. Auflage, Hannover. Heise, 2008.

[Riggert et al. 2014]
Riggert, Wolfgang; Märtin, Christian; Lutz, Michael; *Rechnernetze: Grundlagen ; Ethernet ; Internet,* 5. Auflage, s.l. Carl Hanser Fachbuchverlag, 2014.

[Schnell et al. 2012]
Schnell, Gerhard; Wiedemann, Bernhard (Hrsg.); *Bussysteme in der Automatisierungs- und Prozess-*

technik: Grundlagen, Systeme und Anwendungen der industriellen Kommunikation, 8. Auflage, Wiesbaden. Springer Vieweg, 2012.

[Thomas Carlsson 2018]
Thomas Carlsson; *Industrial Ethernet is now bigger than fieldbuses,* 2018, https://www.anybus.com/about-us/news/2018/02/16/industrial-ethernet-is-now-bigger-than-fieldbuses.
Abgerufen am 19.02.2019.