Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Diplomarbeit

Oliver Gräfe

Entwicklung einer EtherCAT Verbindung zwischen einem 32-Bit PIC Mikrokontroller und einer Soft-SPS

*Fakultät Technik und Informatik*
*Department Informations- und Elektrotechnik*

*Faculty of Engineering and Computer Science*
*Department of Information and Electrical Engineering*

Oliver Gräfe

# Entwicklung einer EtherCAT Verbindung zwischen einem 32-Bit PIC Mikrokontroller und einer Soft-SPS

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Kommunikationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Henning Hasemann
Zweitgutachter : Prof. Dr.-Ing. Ulfert Meiners

Abgegeben am 24. August 2009

**Oliver Gräfe**

**Thema der Diplomarbeit**

Entwicklung einer EtherCAT Verbindung zwischen einem 32-Bit PIC Mikrokontroller und einer Soft-SPS

**Stichworte**

EtherCAT, ASIC, PIC32, ESI-Datei

**Kurzzusammenfassung**

Ziel der Arbeit ist die Entwicklung einer E/A-Karte, die analoge und digitale Signale mit einem Mikrokontroller verarbeitet. Über einen EtherCAT Slave Controller werden die Daten mit einer Soft-SPS ausgetauscht. Eine ESI-Datei ist zu erstellen, um die Daten der entwickelten E/A-Karte in der Soft-SPS mit Variablen zu verknüpfen.

**Oliver Gräfe**

**Title of the paper**

Development of an EtherCAT connection between a 32-bit PIC microcontroller and a Soft-PLC

**Keywords**

EtherCAT, ASIC, PIC32, ESI-File

**Abstract**

The goal of this paper is the development of an I/O-Card, which processes analog and digital signals. Using an EtherCAT Slave Controller data has to be exchanged with a Soft-PLC. The creation of an ESI-File has to be done in order to link the data of the microcontroller to the variables of the Soft-PLC.

# Inhaltsverzeichnis

# Abkürzungen

| | |
|---|---|
| $\mu$C | Mikrokontroller |
| ADS | Automation Device Specification (Beckhoff) |
| ADU | Analog-Digital Umsetzer |
| AL | Application Layer |
| AoE | ADS over EtherCAT |
| ASIC | Application Specific Integrated Chip |
| CAN | Controller Area Network |
| CANopen | Ein auf CAN basierendes Kommunikationsprotokoll |
| CoE | CANopen over EtherCAT |
| CPU | Central Processing Unit |
| CRC | Cyclic Redundancy Check |
| CSMA/CD | Carrier Sense Multiple Access/Collision Detection |
| DC | Distributed Clocks |
| DL | Data Link Layer |
| DMA | Direct Memory Access |
| DPRAM | Dual Port Random Access Memory |
| EtherCAT | Ethernet for Control Automation Technology |
| $E^2$PROM | Electrically Erasable Programmable Read Only Memory |
| EOF | End of Frame |
| EPU | EtherCAT Processing Unit |
| ESC | EtherCAT Slave Controller |
| ESI | EtherCAT Slave Information |
| ETG | EtherCAT Technology Group |
| FCS | Frame Check Sequence |
| FMMU | Fieldbus Memory Management Unit |
| FPGA | Field Programmable Gate Array |
| $I^2C$ | Inter-Integrated Circuit |
| IEEE | Institute of Electrical and Electronics Engineers |
| IP | Internet Protocol |
| IRQ | Interrupt Request |
| LcID | Locale ID |
| LED | Light Emitting Diode |

| | |
|---|---|
| LVDS | Low Voltage Differential Signalling |
| MAC | Media Access Control |
| MI | (PHY) Management Interface |
| MII | Media Independent Interface |
| NC | Numerical Control |
| PDI | Process Data Interface or Physical Device Interface |
| PDO | Process Data Object |
| PDU | Protocol Data Unit |
| PHY | Physical layer Device |
| PLC | Programmable Logic Controller (=SPS) |
| PLL | Phase Locked Loop |
| RAM | Random Access Memory |
| SDO | Service Data Object |
| SII | Slave Information Interface |
| SM | SyncManager |
| SOF | Start of Frame |
| SPI | Serial Peripheral Interface |
| SPS | Speicherprogrammierbare Steuerung (=PLC) |
| SSC | Slave Sample Code |
| WD | Watchdog |
| WKC | Working Counter |
| XML | Extensible Markup Language |

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

Die Programme in der Steuerungstechnik werden immer komplexer und damit auch größer. An vielen Stellen wird deswegen eine Soft-SPS verwendet, welche auf jedem PC laufen kann. Im Vergleich zu einer gewöhnlichen SPS stehen einer Soft-SPS nahezu unbegrenzter Speicherplatz für Steuerungsprogramme zur Verfügung. Damit können auch zukünftige, umfangreichere Steuerungen realisiert werden. Um die Zykluszeiten bei zeitkritischen Prozessen niedrig zu halten, wird ein Teil der Signalverarbeitung ausgelagert. Mikroprozessoren übernehmen einen Teil der Aufgaben und die Daten werden der SPS über ein Bus-System zur Verfügung gestellt. Die Verbindung zwischen Steuerungstechnik und Mikroprozessortechnik ist daher sehr wichtig. Da die zu übertragende Datenmenge steigt, ist ein Bussystem mit hohen Übertragungsgeschwindigkeiten unumgänglich. EtherCAT ist derzeit eines der schnellsten Bus-Systeme, mit welchem einerseits sehr große aber auch sehr kleine Datenmengen schnell übertragen werden können. Durch EtherCAT können Daten zyklisch und asynchron übertragen werden. Bei der Erstellung einer E/A-Karte soll eine Soft-SPS mit einem PIC32 Mikrokontroller über EtherCAT verbunden werden. Die für dieses Projekt zu erstellende E/A-Karte wird den Namen BIN tragen und sie ist Teil des Projekts Win2.1 zur Nachrüstung von Sensoren in diversen Maschinen. In Abbildung 1 auf Seite 9 ist ein 3D Modell von Win2.1 zu sehen. Die E/A-Karte ist zweimal in dem oberen Einschub vorgesehen.

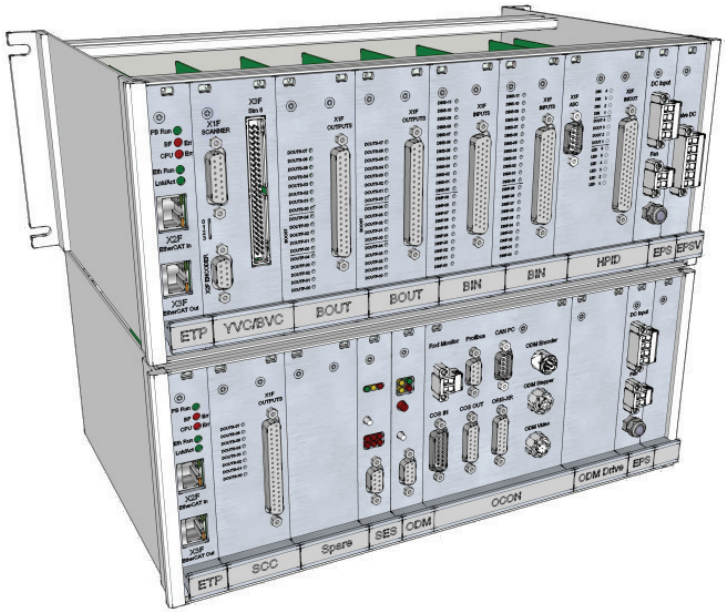



Abbildung 1: 3D Modell von Win2.1

Die Karte hat diverse digitale und analoge Eingänge, deren Zustände dem PIC32 Mikrokontroller zugeführt werden. Dort werden die Daten verarbeitet und über EtherCAT an die Soft-SPS übertragen. Damit ein Mikrokontroller mit einer SPS kommunizieren kann, ist ein EtherCAT Slave Controller (ESC) notwendig. Eine Skizze des Systems ist in Abbildung 2 auf Seite 10 dargestellt. Mit dem Mikrokontroller wird über den parallelen Port kommuniziert und mit der Soft-SPS über den EBUS. Zusätzlich werden in dem Mikrokontroller der BIN-Karte Zählerstände aus digitalen Eingängen ermittelt. Diese werden ebenfalls ein Teil der EtherCAT-Daten sein. Die zu entwickelnde ESI-Datei verbindet das EtherCAT Datagramm mit Variablen, damit diese mit dem SPS-Programm verknüpft werden können. Außerdem werden dort Konfigurationsdaten für den ESC gespeichert. Die Informationen der ESI-Datei werden auch in einem $E^2$PROM auf der E/A-Karte abgelegt.

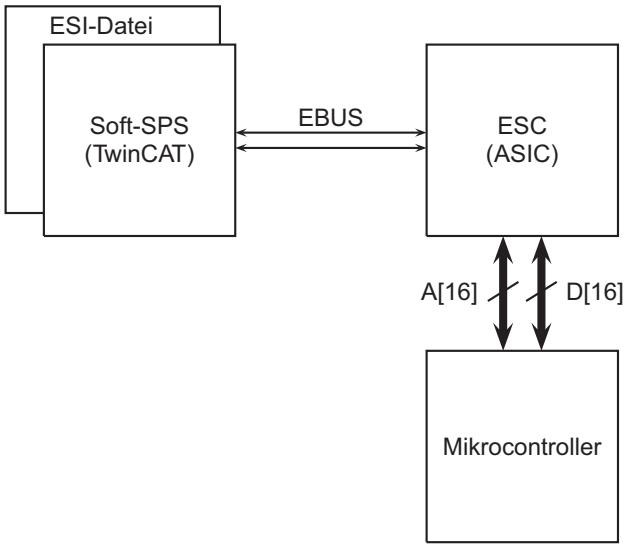



Abbildung 2: Blockdiagramm für die Mikrokontroller - Soft-SPS Kommunikation

## 1.1 Lastenheft

**Spezifikation** Eine E/A-Karte soll erstellt werden, welche über digitale und analoge Eingänge verfügt. Dafür sind in einem bestehendem Projekt Änderungen vorzunehmen. Es soll ein EtherCAT Slave Controller (ESC) verwendet werden, um die Kommunikation zwischen dem PIC Mikrokontroller und der Soft-SPS über EtherCAT herzustellen. Die Verbindung des Mikrokontrollers zum ESC soll über den parallelen Port hergestellt werden, um den Datenaustausch zu beschleunigen. Die E/A-Karte wird über 27 digitale Eingänge, fünf analoge Eingänge und einen analogen Ausgang verfügen. Zehn digitale Eingänge werden dafür benutzt, Zählerstände zu erzeugen, die im weiteren Verlauf an die Soft-SPS übermittelt werden sollen. Des Weiteren soll es möglich sein, dass von der Soft-SPS aus, die Zählerstände im PIC Mikrokontroller zurück gesetzt werden können. Diese Funktion soll nach Möglichkeit mit Hilfe der Mailboxkommunikation implementiert werden. Der ASIC wird durch einen Koppler mit der Soft-SPS verbunden. Die Verbindung zwischen dem ASIC und dem Koppler wird über den EBUS realisiert. Als Soft-SPS soll TwinCAT von Beckhoff verwendet werden.
Die Erstellung einer Hauni LNI EtherCAT Slave Informations-Datei (ESI-Datei) ist ebenfalls durchzuführen. In dieser Datei soll für den TwinCAT System Manager ein neuer Zweig für LNI Baugruppen entstehen und die E/A-Karte BIN soll mit ihren Ein- und Ausgängen hinterlegt werden, damit eine automatische Erkennung durchgeführt werden kann.

**Umsetzung der Hardware** Der Großteil der Hardware wird aus einem vorherigen Projekt übernommen, da hierfür keine Änderungen vorgenommen werden müssen. Geändert wird die Verbindung zur Steuerung. Zuvor ist über $I^2C$ kommuniziert worden. Nun wird die Anbindung über EtherCAT realisiert. Als EtherCAT Slave Controller soll ein Application Specific Integrated Chip (ASIC) verwendet werden. Auf die Auswahl wird im Kapitel “Entwurf” näher eingegangen. Auch ein PIC Mikrokontroller wird neu hinzugefügt, an welchen alle digitalen und analogen Eingänge, der ASIC, ein Temperatur-Sensor und ein Digital/Analog-Umsetzer angebunden werden. Die letzten beiden Module werden über den $I^2C$-Bus mit den PIC kommunizieren. Des Weiteren soll noch die Verwendung der seriellen Schnittstelle implementiert werden, um Fehlermeldungen anzeigen zu können.

**Vorgehensweise** Zuerst wird der neue Blockschaltplan mit den geänderten Spezifikationen entworfen. Anschließend wird der Schaltplan den Vorgaben entsprechend angepasst. Da die neu entwickelte E/A-Karte für Tests nicht zur Verfügung stehen wird, muss eine Test-Hardware erstellt werden, welche die gleichen Anbindungen hat, wie die spätere BIN-Karte.

Mit diesem Testaufbau werden dann die einzelnen Mikrokontrollerfunktionen und die Kommunikation mit TwinCAT getestet. Zu den Grundfunktionen gehören das Auslesen von digitalen Eingängen, der Analog/Digital-Umsetzer, die Timer, die serielle Schnittstelle und die Kommunikation über den parallelen Port. Nachdem die Grundfunktionen einzeln und zusammen einwandfrei arbeiten, wird der Programmcode für die Kommunikation zwischen Mikrokontroller und ASIC geladen. Dieser Quellcode wird ebenfalls als Teil des Gesamtprogramms in den PIC $\mu$C übertragen. In diesem Programm sind Änderungen durchzuführen, welche speziell auf den PIC abgestimmt sind. Danach soll die Verbindung zwischen dem ASIC und der Soft-SPS getestet werden.
Die ESI-Datei wird im Anschluss eingebunden, damit diese im TwinCAT System Manager leicht hinzugefügt werden kann. Die ESI-Datei ist eine XML-Konfigurationsdatei und muss nach bestimmten Richtlinien erstellt werden. Es müssen alle digitalen und analogen Ein- und Ausgänge in der Datei gespeichert werden, damit sie später in TwinCAT mit dem SPS Programm verknüpft werden können. Außerdem müssen in der Datei die Ressourcen-Manager eingerichtet und die Konfigurationsdaten für das $E^2$PROM hinterlegt werden.

## 1.2 Motiv

Durch die Änderung der Verbindung von $I^2C$ auf EtherCAT wird eine höhere Flexibilität erwartet. Wünsche des Kunden können über EtherCAT schneller realisiert werden als über proprietäre Lösungen. Herstellerspezifische Neuentwicklungen benötigen eine deutlich größere Zeitspanne in der Entwicklung. Viele Lösungen für EtherCAT sind bereits am Markt verfügbar, so dass Eigenentwicklungen häufig nicht mehr nötig sind. Dadurch können Entwicklungskosten eingespart werden. Ein weiteres Ziel dieser Umstellung ist die Anpassung an die Steuerung der Host-Maschine, die ebenfalls über EtherCAT kommuniziert. Dadurch kann bei zukünftigen Entwicklungen auf das technische Know-how zurückgegriffen werden.

## 1.3 Zeitplan

Zur Erstellung eines Zeitplans wurde das Projekt in verschiedene Themen unterteilt. Jedem dieser Themen wird eine geplante Bearbeitungsdauer zugewiesen. Die Zeiten wurden in Absprache mit dem Projektleiter festgelegt. Tabelle 1 auf Seite 13 zeigt die Auflistung der einzelnen Themen mit der geplanten Bearbeitungsdauer. Eine Aufstellung mit einem Zeitdiagramm wurde nicht erstellt, da die einzelnen Themen zum Teil sehr stark ineinander übergehen und dadurch die Themen nicht nacheinander abgearbeitet werden. Es ist auch zu erwarten, dass häufig zwischen den Themen hin und her gesprungen wird.

Tabelle 1: Zeitplan

| Thema | Bearbeitungsdauer |
|---|---|
| Einarbeitung EtherCAT Protokoll | 7 Tage |
| Einarbeitung ASIC | 7 Tage |
| Einarbeitung TwinCAT (System Manager) | 5 Tage |
| Einarbeitung Mikrokontroller | 5 Tage |
| Kommunikations Interface ASIC <> Mikrokontroller | 2 Tage |
| Umsetzung des ASIC <> Mikrokontroller Interface | 2 Tage |
| Implementierung der $I^2C$ Kommunikation | 5 Tage |
| Analyse des Slave Sample Code | 10 Tage |
| Adaption des SSC an den $\mu$C für zyklische Datenübertragung | 15 Tage |
| Adaption des SSC an den $\mu$C für Mailboxkommunikation | 10 Tage |
| Erstellung der ESI-Datei | 2 Tage |
| Testphase | 10 Tage |
| Schreiben der Diplomarbeit | 20 Tage |
| Gesamtdauer | 100 Tage |

## 1.4 Struktur der Arbeit

Zu Beginn werden einige Grundlagen erläutert, um die im Verlauf der Arbeit benutzten Begriffe sowie die Technik zu erläutern. Die Grundlagen sind in zwei Gebiete aufgeteilt: die EtherCAT Kommunikation und der PIC Mikrokontroller. Danach wird der Entwurf der Arbeit vorgestellt. In diesem Kapitel wird dargestellt, wie an das Projekt herangegangen wird und welche Überlegungen vorab gemacht worden sind. In der Implementierung wird die Vorgehensweise erklärt und es wird auf Besonderheiten eingegangen. Dies wird gemacht, um ein besseres Bild zu bekommen, welche Arbeiten durchgeführt worden sind und wo Schwierigkeiten gelegen haben. Hier wird auch auf die Erstellung der ESI-Datei im Einzelnen eingegangen. Das Kapitel Leistungsbewertung wird dafür verwendet, zwei Tests vorzustellen und diese anhand von Messergebnissen zu bewerten. Außerdem wird in diesem Kapitel die geplante und tatsächlich verwendete Zeit verglichen. Zum Schluss wird die Arbeit zusammengefasst und es wird ein Ausblick auf kommende Aufgaben gegeben.

# 2 EtherCAT Grundlagen

## 2.1 Einführung

EtherCAT steht für Ethernet Control Automation Technology und ist ein auf Ethernet basierender Feldbus. Das Bus-System ist 2003 erstmals vorgestellt und ursprünglich von der Firma Beckhoff Automation GmbH entwickelt worden. Derzeit ist EtherCAT ein öffentliches System, welches von der EtherCAT Technology Group ständig weiterentwickelt wird. Die Organisation ist ein Zusammenschluss von Interessenten, Herstellern und Anwendern mit über 900 Mitgliedern aus 45 Ländern (Stand Januar 2009).
EtherCAT besitzt keine untergelagerten Sub-Busse mehr und hat auch keine Verzögerung in Gateways. Die Übertragungsrate beträgt 2 x 100 Mbaud im Voll-Duplex[1] Mode. Dadurch können 1000 digitale IOs verteilt über 100 Busteilnehmern in nur 30$\mu$s aktualisiert werden. Im Vergleich zu anderen Ethernet-Lösungen ist die Bandbreitenausnutzung wesentlich effizienter. Bei vier Byte Nutzdaten pro Busteilnehmer erreicht Polling[2] oder Timeslicing[3] nur jeweils 2-2,5% Bandbreitenausnutzung. EtherCAT erreicht ab zwei Bit Nutzdaten bereits 80 bis 97% Auslastung der Bandbreite.
EtherCAT wird vor allem für sehr schnelle Echtzeitsysteme verwendet. Durch die niedrigen Updatezeiten ist es auch für schnelle Antriebsregelung geeignet. Es kann gesagt werden, dass EtherCAT vor allem für den industriellen Einsatz vorgesehen ist. Im Office-Bereich wäre das Bus-System nicht tauglich.
Der Feldbus besteht aus einem Master und bis zu 65535 Slaves. Die Topologie kann beliebig sein. Die Frames laufen aber immer zweimal durch den Slave, so dass letztendlich immer ein Ring entsteht. Es sind keine speziellen Bus-Kabel notwendig, da zur Verbindung Standard CAT5[4] Kabel zu benutzen sind.
Der EtherCAT-Master ist zuständig für das Mapping der Prozessdaten, die Initialisierung des Bus-Systems, sowie das Senden der Mailbox-Befehle. Zu weiteren Aufgaben gehören die State Machine, welche im Master läuft, das Senden der zyklischen Prozessdaten-Befehle und das Bearbeiten diverser Protokolle. Außerdem befindet sich im Master die Schnittstelle zum Netzwerktreiber und zur Applikation. Für einen Master wird zum einen eine einfache

[1] Übertragung von Informationen in beiden Richtungen zur gleichen Zeit
[2] Zyklische Statusabfrage
[3] Eine Technik, welche einzelnen Prozessen Rechenzeit zuweist.
[4] Ein typisches Ethernet Netzwerkkabel

Netzwerkkarte benötigt und zum anderen muss die Software ein Realtime Kernel[5] zur Verfügung stellen. Als Konfigurations-Tool kann zum Beispiel TwinCAT verwendet werden.
Ein EtherCAT Slave hat eine eigene Applikation, welche von diesem ausgeführt wird und unabhängig vom Master läuft. Die Kommunikation der Prozessdaten wird über den EtherCAT Slave Controller (ESC) geregelt.

## 2.2 Allgemeine Informationen

**Data Link Layer (DL)** Der Data Link Layer verbindet den Physical und den Application Layer. Dabei wird die grundlegende Kommunikations Infrastruktur bereit gestellt. Der Zugriff auf das Sende-/Empfangsgerät des PHY wird verwaltet. Das Adressieren im globalen und lokalen Bereich, sowie Zugang zum $E^2$PROM werden ebenfalls kontrolliert. Außerdem liegt der SyncManager und die Fieldbus Memory Management Unit in diesem Bereich. Auch die Distributed Clocks werden hier verwaltet. Das Zusammenspiel mit der State Machine im Application Layer wird im Data Link Layer implementiert. Die Funktionen der Distributed Clocks, der FMMU und des SyncManager werden in weiteren Abschnitten beschrieben.

**Application Layer (AL)** Im Application Layer von EtherCAT befindet sich die EtherCAT State Machine. Außerdem gibt es noch das Mailbox-Interface, diverse Protokolle und das Slave Information Interface (SII). Über das SII werden die Informationen der EtherCAT Baugruppe mit dem Master abgeglichen. Die EtherCAT State Machine fährt die Baugruppen, sowie das Netzwerk hoch.

**State Machine** Die State Machine beschreibt die verschiedenen Kommunikationszustände der EtherCAT Slave Geräte. Es werden in der State Machine auch die Initialisierung und das Verhalten im Fehlerfall der Slaves spezifiziert. Fünf Zustände können von den Baugruppen eingenommen werden. Zu ihnen gehören Init, Pre-Operational, Safe-Operational, Operational and Bootstrap. In Tabelle 2 auf Seite 17 werden die Eigenschaften der einzelnen States beschrieben. Abbildung 3 (Seite 17) zeigt, wie sich die Zustände der State Machine verhalten.

**Mailbox** Die Mailbox ist der übliche Weg, um Parameter auszutauschen. Sie wird hauptsächlich für nicht zyklische Anwendungen benutzt. Die Daten können im Voll-Duplex Mo-

[5] Ein Kernel ist der Systemkern des Betriebssystem. Bei einer Soft-SPS wird ein Realtime Kernel noch über den Kernel des Betriebssystem gelegt und der Realtime Kernel führt diesen mit niedrigster Priorität aus. Alle Echtzeitprozesse werden dann mit höchster Priorität aufgerufen.

Tabelle 2: Zustände der State Machine

| | |
|---|---|
| Init | Es besteht keine Kommunikation im Application Layer, aber der Master hat bereits Zugriff auf die DL-Informations Register. Hier muss der Master wenigstens das DL Adressregister und die Kanäle für die Mailbox des SyncManager konfigurieren. |
| Pre-Operational | Es funktioniert bereits die Kommunikation mit dem Application Layer über die Mailbox, es gibt aber weiterhin keine Kommunikation der Prozessdaten. Nun müssen weitere Parameter konfiguriert werden. Hierzu gehören das Mapping der Prozessdaten und das Einrichten des SyncManager und der FMMU. Danach kann der Safe-Operational Zustand angefragt werden. |
| Safe-Operational | Die Kommunikation der Prozessdaten beginnt, aber zuerst sind nur Eingangswerte gültig. Ausgänge werden in einem sogenannten Safe State belassen. Sobald der Master gültige Ausgangswerte sendet und den Operational Zustand anfordert, wird dies geändert. |
| Operational | Ein- und Ausgänge sind gültig und der finale Zustand ist erreicht. |
| Bootstrap | Ein optionaler Zustand, welcher für Firmware Updates verwendet wird. |

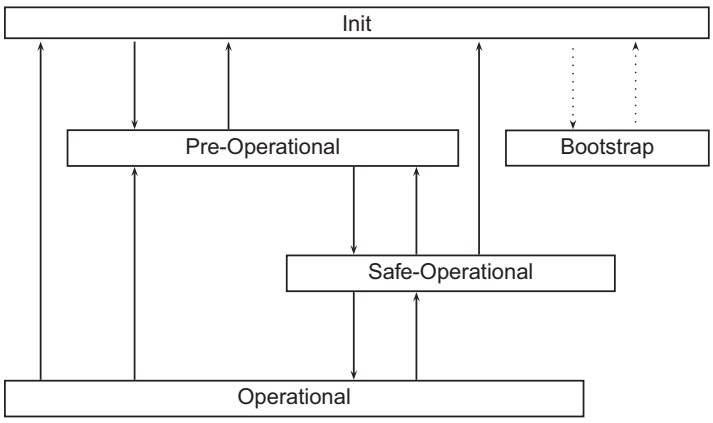



Abbildung 3: Die Zustände der EtherCAT State Machine und ihr Verhalten untereinander [6]

de ausgetauscht werden. Auch der Slave kann hierüber eine Kommunikation initiieren. Die Mailbox ist bereits in einer frühen Phase, dem Pre-Operational State, der Kommunikation vorhanden. Es gibt fünf Mailboxprotokolle. Ethernet over EtherCAT (EoE), CANopen over

EtherCAT (CoE), File Access over EtherCAT (FoE), Servo Drive over EtherCAT (SoE) und Vendor specific Profile over EtherCAT (VoE). Das Mailbox-Interface liegt in einem Datagram des EtherCAT Frames. In dem Fall besteht ein Datagram aus dem Datagram Header, dem Mailboxprotokoll selbst und dem Working Counter (WKC). Das Mailboxprotokoll besteht aus einem Header, einem Command und den den zugehörigen, spezifischen Daten. Im Header stehen die Länge der folgenden Daten, die Adresse des Absenders, der Mailbox-Type und eine Sequenz Nummer, die zur Erkennung von Duplikaten verwendet wird. Für die Zukunft sind noch ein 6 Bit langer Kanal Bereich und eine 2 Bit lange Priorität reserviert. Diese kommen aber noch nicht zum Einsatz.

Wenn die Mailbox auf Ethernet over EtherCAT gesetzt ist, werden Ethernet Frames über EtherCAT getunnelt. EoE wird für Elemente mit TCP/IP Stack verwendet und ermöglicht so den Zugriff auf Webserver einzelner Baugruppen. Der Master fungiert hierbei als virtueller Ethernet Switch.

Mit dem Kommunikationsprotokoll CANopen können in der Automatisierungstechnik Baugruppen miteinander verbunden werden. Dieses Protokoll kann mit der CoE Funktion der Mailbox von Baugruppen zur Steuerungseinheit im Master übertragen werden. Mit Hilfe des CANopen Protokolls werden alle Objekte in einem Objektverzeichnis (Englisch: Object Dictionary) abgelegt. Dadurch kann die Mailbox über das CoE Protokoll auf einzelne Prozessdaten zugreifen. Das ermöglicht einen asynchronen Datenaustausch von zeitkritischen Informationen.

## 2.3 Funktionsweise

Die Arbeitsweise von EtherCAT ist anders als die Standard Arbeitsweise vom Ethernet Protokoll. Bei Ethernet wird das ganze Protokoll von jedem Teilnehmer empfangen, interpretiert und die relevanten Prozessdaten weiterkopiert. Bei EtherCAT durchläuft das Frame den Slave. Während des Durchlaufs können einzelne Bits oder Bytes aus dem Frame entnommen oder hineingeschrieben werden. Diese Telegrammstruktur ist sehr effizient. Die Kommunikation erfolgt komplett in Hardware und erreicht dadurch eine sehr gute Performance, da keine $\mu$C hierfür verwendet werden müssen. Es ist kein Ethernet-Switch erforderlich, wenn nur EtherCAT Teilnehmer an einem Port angeschlossen sind. Dennoch behält das System seine Kompatibilität zu Ethernet. Es kann eine Prozessdatengröße pro Slave von 1 Bit bis 60 kByte vorliegen. Diese können über mehrere Frames verteilt sein. Die Zusammensetzung der Prozessdaten kann sich mit jedem Zyklus ändern. Es ist möglich für einige Slaves, wie Antriebe die kürzeste Zykluszeit zu verwenden, und gleichzeitig für E/A Baugruppen längere Zykluszeiten zu benutzen. Asynchroner und bedarfsgesteuerter Datenaustausch kann jederzeit durchgeführt werden. Es kann theoretisch ein Standard Switch eingesetzt werden, um Ethernet Kommunikation mit anderen Netzen durchzuführen. Dies wird aber nicht empfohlen, da es dadurch zu Instabilitäten des Netzwerkes führen kann. Mit einem solchen Switch wird nur genau eine MAC Adresse für bis zu 65535 Teilnehmer vergeben. Dann werden die Baugruppen segmentweise adressiert. Durch diese Art der Übertragung hat EtherCAT schnellere Update-Zeiten als andere Bus-Systeme ohne die ganze Bandbreite auszunutzen.

In Abbildung 4 auf Seite 20 ist zu sehen, wie sich verschiedene Bus-Systeme unter vergleichbaren Bedingungen verhalten. Hierbei wurden 40 Achsen (je 20 Byte Eingangs- und Ausgangsdaten), 50 E/A Stationen mit 560 EtherCAT Busklemmen und 2000 Digitalen und 200 Analogen IO, sowie 500m Buslänge verwendet. Wie zu sehen ist, hat EtherCAT mit 276$\mu$s die geringste Update Zeit. Außerdem hat EtherCAT auch noch 56% seiner Bandbreite zur Verfügung. Im Gegensatz dazu benötigen die anderen Bus-Systeme die volle Bandbreite, um auf die angegebenen Zeiten zu kommen. Bei EtherCAT wird kein untergelagerter Erweiterungsbus benötigt und auch die sonst typische Master-Anschaltbaugruppe ist nicht weiter vorhanden, dadurch sind keine speziellen Einsteckkarten für den Master mehr erforderlich. Zwischen den PLC Tasks sind einerseits der EtherCAT Zyklus und andererseits

zwei Datentransfers per Direct Memory Access (DMA[6]) vorhanden. Die Zeiten des DMA Zugriffs sind aber zu vernachlässigen. Deshalb wird die Reaktionszeit deutlich verkürzt, wobei die Steuerungsleistung gleich bleibt.

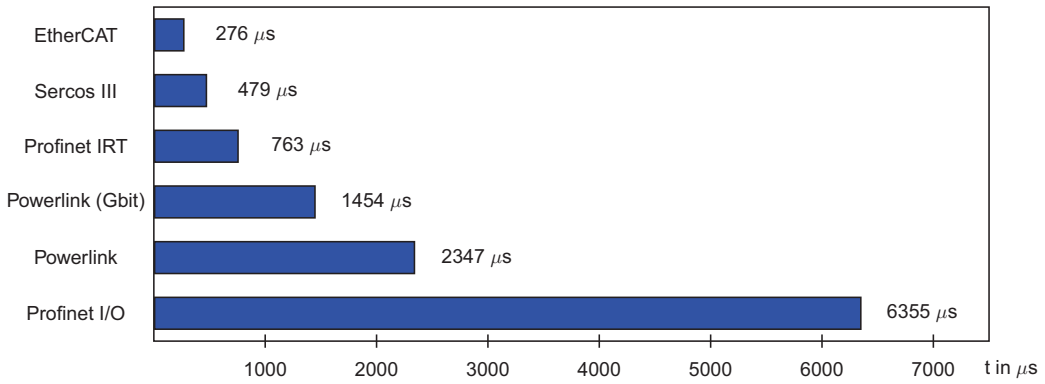



Abbildung 4: Update Zeiten verschiedener Bus-Systeme [9]

Bei herkömmlichen Feldbussystemen ist es üblich, dass aus einem physikalischen Prozessabbild durch das Mapping ein logisches Prozessabbild erzeugt wird. Dieses erfolgt in der Steuerung des Systems. Auch bei Steuerungen, die nur ein Prozessabbild haben, wie eine SPS, ist das Mapping erforderlich. Bei EtherCAT ist ebenfalls eine Abbildung der Daten nötig, aber hier wird dieses in die Slaves verlagert. Durch diese Art des Mappings wird die Steuerung entlastet und die Entwicklung eines Master wird sehr vereinfacht. Die Daten werden je nach Applikationsanforderung übertragen, was schnell, flexibel und effizient ist.

EtherCAT verwendet Standard Frames nach IEEE 802.3 (Ethernet Standard Frame mit CSMA/CD), also keine verkürzten Frames. In Abbildung 5 auf Seite 21 ist der Aufbau des Frames zu sehen. Eingebettet in den Ethernet Daten liegen die EtherCAT Datagramme. Vor den Datagrammen sind noch elf Bit, welche die Gesamtlänge der Datagramme ohne die Frame Check Sequence (FCS) angeben. Des weiteren ist noch ein reserviertes Bit und vier Bit für den Protocol Typ. Der EtherCAT Slave Controller unterstützt nur den Typ 0x1, welches EtherCAT Kommandos sind. Es stehen insgesamt bis zu 1498 Byte an Daten pro Ethernet Block zur Verfügung. Die Daten werden in n Datagramme aufgeteilt. Jedes von diesen

[6]Direct Memory Access erlaubt Baugruppen den direkten Zugriff auf den Speicher. Die CPU wird hierbei nicht beansprucht und der Zugriff erfolgt deutlicher schneller, als bei anderen Kommunikationsarten.

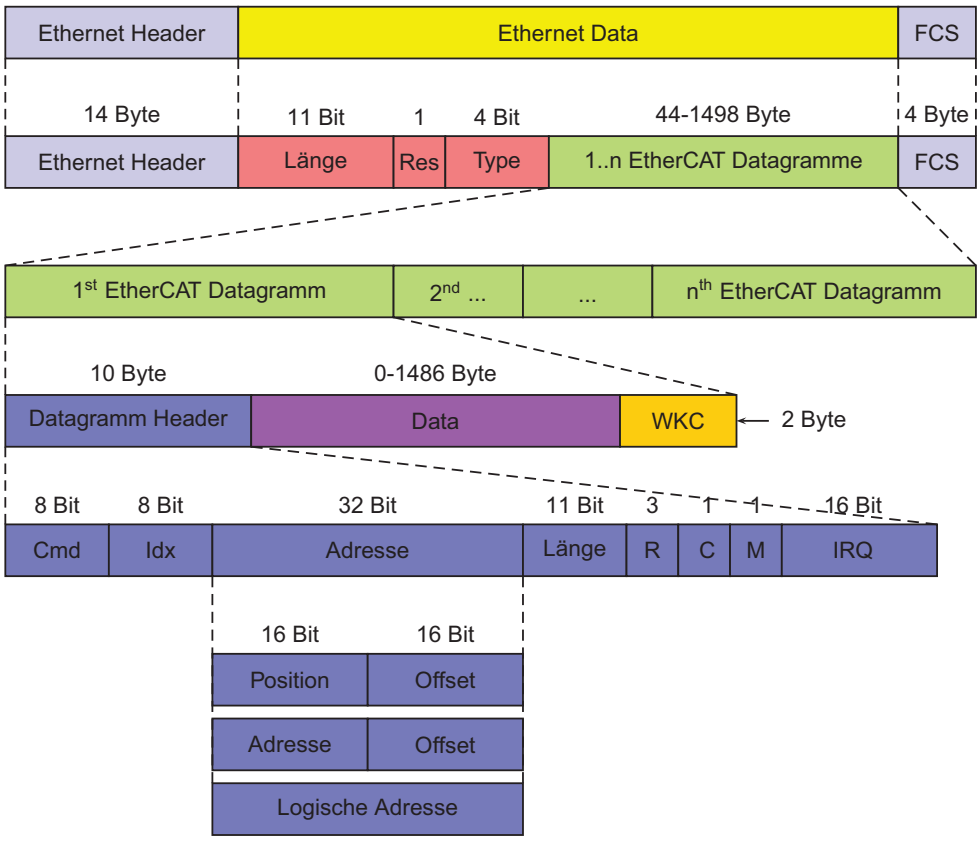



Abbildung 5: Aufbau des Ethernet Frames mit integriertem EtherCAT Frame [6]

hat einen zehn Byte Header bis zu 1486 Byte Daten und 2 Byte für den Working Counter (WKC). Der WKC erhöht sich in jeder Baugruppen, welche erfolgreich mit dem jeweiligen Datagramm adressiert worden ist. Erfolgreich bedeutet in diesem Fall, dass der EtherCAT Slave Controller adressiert worden ist und auf den Speicher zugegriffen werden konnte. Der ESC erhöht den Zähler in Hardware. Jedes Datagramm hat einen erwarteten Working Counter Wert, welcher vom Master zuvor berechnet worden ist. Ein Lese- oder Schreibzugriff erhöht den WKC um Eins und bei einem kombinierten Zugriff erhöht ein erfolgreiches Lesen um Eins, ein Schreiben um zwei und sollten beide Zugriffe erfolgreich sein, dann erhöht sich der WKC in dieser Baugruppe um drei. Dadurch kann der Master kontrollieren, ob das Datagramm richtig verarbeitet wurde. Im Header eines Datagramms wird zuerst eine von 15 Anweisungen übertragen, die die Zugriffsart und den Kommunikationstyp beinhalten. Gefolgt werden diese von einem acht Bit großen Identifier. Danach kommen 32 Bit für die Adresse. Die Adresse kann entweder eine Positionsadressierung, eine Knotenadressierung

oder eine logische Adressierung sein. Der Adresse folgt ein elf Bit langer Wert, welcher die Länge des Datagramms angibt. Nach drei reservierten Bits kommt je ein Bit, welches zum einen anzeigt, ob das Frame bereits in diesem Slave gewesen ist und ein Bit, welches aussagt, ob noch weitere Datagramme folgen. Abgeschlossen wird der Header mit zwei Byte, in denen alle EtherCAT Event Request Register mit einer ODER-Funktion verknüpft werden.

## 2.4 EtherCAT Slave Controller (ESC)

Der EtherCAT Slave Controller ist die Schnittstelle zwischen dem Feldbus EtherCAT und der Anwendung der Baugruppe, beziehungsweise der Slaves. Als EtherCAT Slave Controller werden ASICs oder FPGAs verwendet. Die Abbildung 6 auf Seite 23 stellt den allgemeinen Aufbau eines ESC dar.

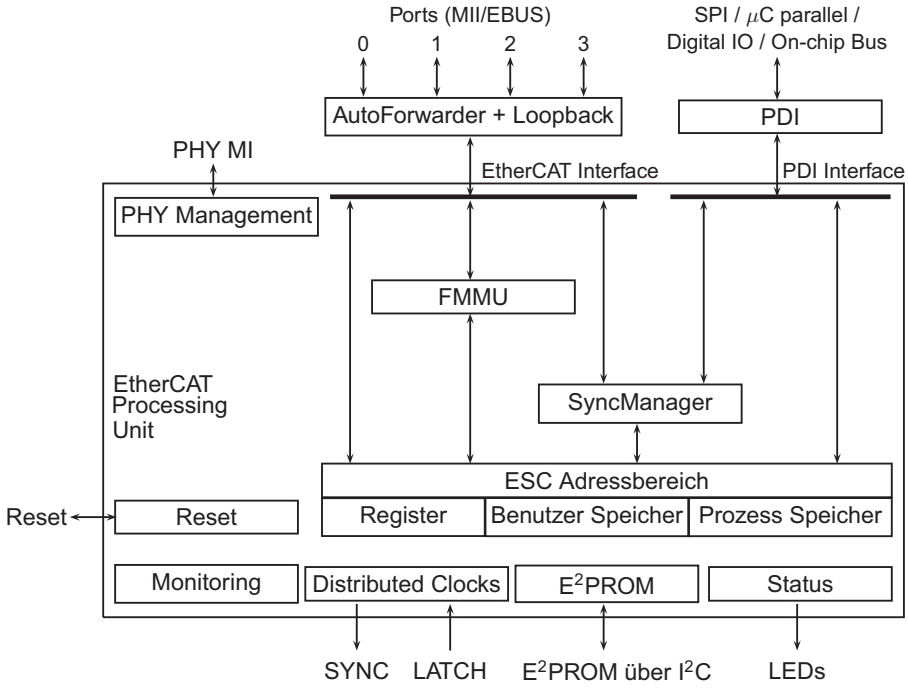



Abbildung 6: Blockdiagramm eines EtherCAT Slave Controllers [6]

**EtherCAT Interface** Diese Schnittstelle verbindet den ESC zu weiteren EtherCAT Slaves und dem Master. Als physikalische Schicht kann Ethernet oder EBUS verwendet werden. EBUS ist komplett in den ASICs integriert. Um Ethernet Ports zu verwenden wird ein Physical Layer Device (PHY) benötigt, um die Daten vom Ethernet in elektrische oder optische Signale zu übertragen. Diese PHYs werden dann mit Hilfe des Media Independent Interface (MII) mit dem Ethernet verbunden. Die EtherCAT Übertragungsrate ist fixiert auf 100 MBit/s im Voll-Duplex Mode. Es werden zwei bis vier Ports unterstützt. Die Wahl des ESC entscheidet, wie viele Ports tatsächlich vorhanden sind.

**EtherCAT Processing Unit (EPU)** Die EPU empfängt, analysiert und verarbeitet den EtherCAT Datenstrom. Sie befindet sich zwischen dem Port 0 und dem Port 3. Die Hauptaufgabe der EPU ist es den Zugriff auf die internen Register und den Speicher des ESC zu ermöglichen und zu koordinieren. Der Zugriff kann zum einen von dem EtherCAT Master her erfolgen oder zum anderen von der lokalen Applikation über das Process Data Interface (PDI). Das PDI ist die Schnittstelle zwischen dem ESC und der Prozessseite. Um den Datenaustausch zwischen Master und Slave im Vergleich zu einem gewöhnlichen zweiseitigem Speicher zu verbessern, werden spezielle Funktionen verwendet. Zu diesen gehören die Überwachung der Datenkonsistenz des SyncManager und das Daten-Mapping der FMMU.

**Auto-Forwarder und Loopback Funktion** Der Auto-Forwarder empfängt die Ethernet Frames, führt einen Frame Check durch und leitet das Frame weiter an die Loop-back Funktion. Die Time Stamps werden vom Auto-Forwarder erzeugt. Eine schematische Darstellung ist in Abbildung 7 auf Seite 25. Die Loopback Funktion leitet die Ethernet Frames weiter zum nächsten Port. Sollte der Port nicht verbunden oder geschlossen sein, wird der Port übergangen (roter Pfeil). Die Loopback Funktion von Port 0 leitet das Ethernet Frame weiter zur EPU. Die Einstellung der Funktion können vom EtherCAT Master kontrolliert werden.

**Fieldbus Memory Management Unit (FMMU)** Die Fieldbus Memory Management Unit ist dafür zuständig Bereiche aus dem lokalen Adressbereich auf den globalen Adressbereich abzubilden und umgekehrt. Ob der Bereich für Lese- oder Schreibzugriff ausgelegt ist, ist dabei klar erkennbar. Selbst eine bitweise Struktur des Speichers ist möglich. Es stehen 16 unabhängige Kanäle für die FMMU und vier Gigabyte globaler Speicher zur Verfügung. Die FMMU bildet diesen Speicherbereich auf den lokalen Speicherbereich ab, indem sie die Adressen miteinander verknüpft. Zu Beginn der Initialisierung des Systems wird die Hardwarekonstellation vom Master ausgelesen. Hierbei werden auch alle Input und Output Datenbereiche der Slaves erkannt. Die FMMU beginnt darauf mit dem Mapping der Prozessdaten. Danach wird vom Master jedem Slave mitgeteilt ab welcher Adresse des Datagramms ihm zugewiesen ist. Nun kann die Kommunikation mit Prozessdaten beginnen.

**SyncManager** Der SyncManager ist ein Ressourcenmanager und dafür verantwortlich, dass der Speicherbereich des DPRAM[7] vor gleichzeitigem Zugriff geschützt ist. Dadurch

[7] Dual Port Random Access Memory ist ein spezieller RAM, welcher den Zugriff mehrerer Lese- oder Schreibzugriffe nahezu Gleichzeitig erlaubt.

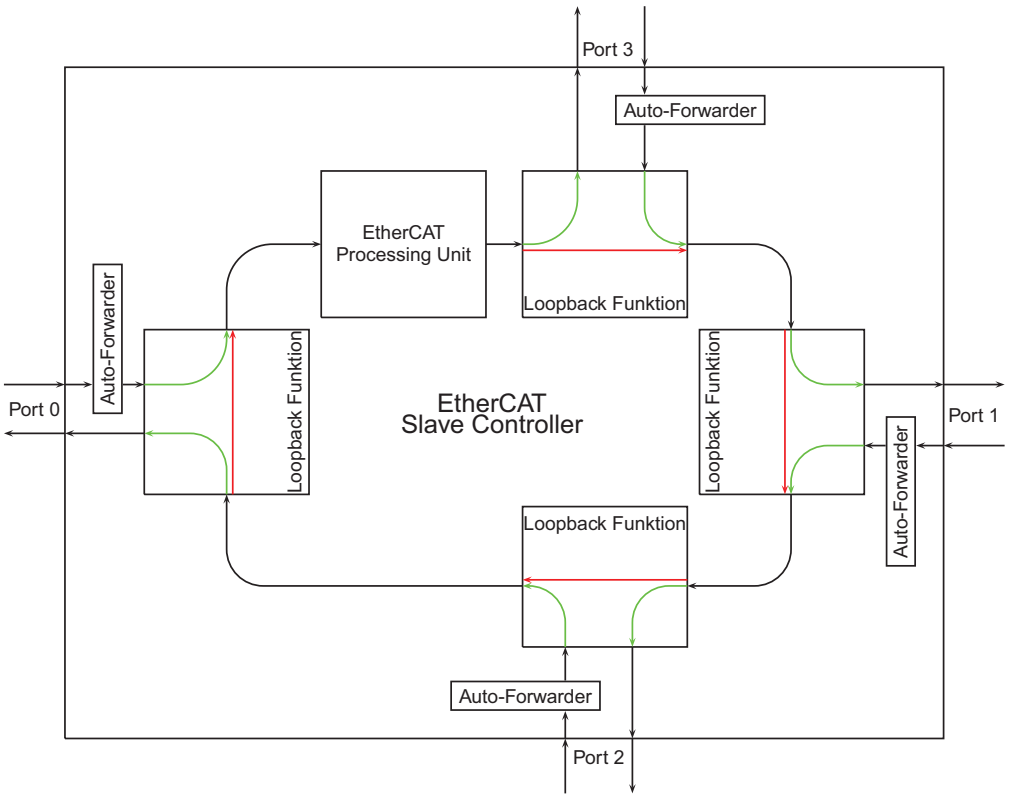



Abbildung 7: Schematische Darstellung der Frame Abarbeitung [6]

soll die Daten Konsistenz gesichert werden. Es können bis zu 16 unabhängige Kanäle benutzt werden, um die Beständigkeit der Daten zu sichern. Der SyncManager kann auf zwei Arten agieren. Zum einen als Mailbox-Type und zum anderen als Buffered-Type.

- Beim Mailbox-Type wird ein Puffer verwendet. Die Daten sind gegen Überlauf geschützt und ein Handshake zwischen EtherCAT und Benutzer wird eingesetzt. Zuerst beginnt die schreibende Seite zu schreiben und hinterher können die Daten gelesen werden. Während des Schreibens beziehungsweise Lesens des Puffer, ist der Puffer für die andere Seite gesperrt. (Abbildung 8, Seite 26)

- Beim Buffered-Type Prinzip gibt es drei Puffer, damit zu jeder Zeit Daten gelesen werden können. Hierbei ist immer ein freier Puffer zum Schreiben und ein Puffer zum Lesen verfügbar. Nur vor dem ersten Schreiben kann noch nicht gelesen werden. Diese Technik wird hauptsächlich bei der Übertragung von Prozessdaten verwendet (Abbildung 9, Seite 26).

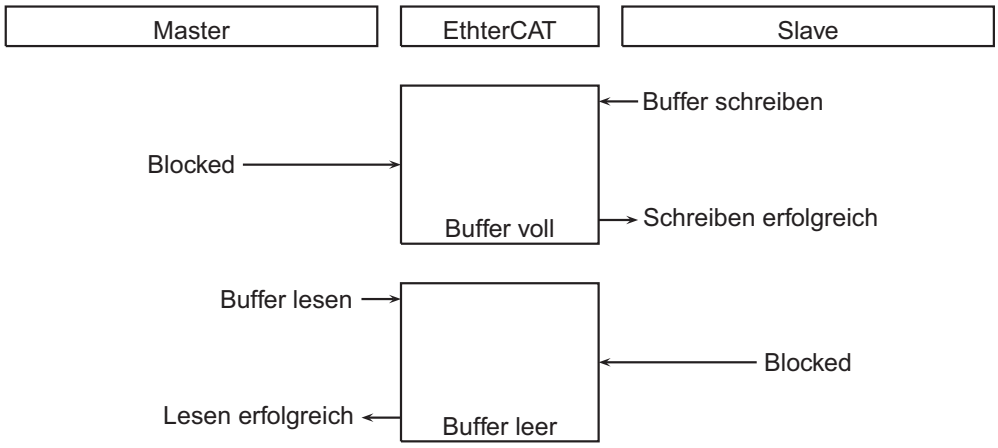



Abbildung 8: Beispiel für einen Lesezugriff der Mailbox

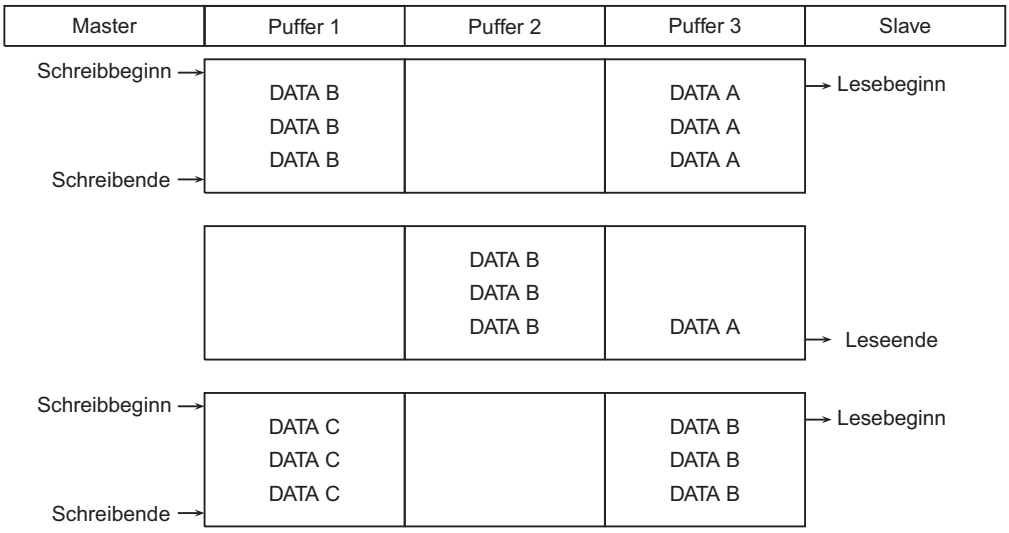



Abbildung 9: Beispiel für einen Schreibzugriff des Buffered-Type

**Monitoring** Diese Einheit enthält Zähler für Fehlermeldungen und Watchdogs. Die Watchdogs werden verwendet, um die Kommunikation zu überwachen und um zu einem sicheren Status zurückkehren zu können, sollte ein Fehler auftreten. Die Zähler werden zur Analyse und Fehlererkennung verwendet.

**Reset** Diese Funktion ist nur in den ASICs vorhanden. Der integrierte Reset Controller überwacht die Versorgungsspannung und kontrolliert die externen und internen Resets.

**PHY Management** Dieses Interface verbindet die physikalische Schicht des Ethernet Busses mit dem ESC. Diese Schnittstelle kann entweder vom EtherCAT Master oder Slave verwendet werden. Sie wird vom ESC selbst benutzt, um eine erneute Kommunikation aufzubauen, nachdem es Fehler beim Empfangen von Daten gegeben hat.

**Distributed Clocks** Die Distributed Clocks sind ein Verbund von Uhren, welche eine Besonderheit des EtherCAT Protokolls sind. Jeder EtherCAT-Teilnehmer, der über Distributed Clocks verfügt, gleicht seine „lokale“ Uhr mit einem vordefinierten Master bzw. Referenz-Baugruppe ab. Jeder dieser Teilnehmer misst die Zeitdifferenz zwischen dem abgehenden und zurückkehrendem EtherCAT Frame. Der Master berechnet darauf basierend die Laufzeiten zwischen verschiedenen Teilnehmern im EtherCAT Netz. Aus diesen Informationen zieht der Master auch Rückschlüsse auf die Topologie des Netzwerks. Der Master schickt jedem Slave seinen persönlichen Offset gegenüber der System Zeit. Eine der Baugruppen wird als Referenz-Baugruppe festgelegt. Typischerweise ist das die Baugruppe, welche über Distributed Clocks verfügt, mit der geringsten Verzögerungszeit zum Master. Distributed Clocks haben es ermöglicht Baugruppen mit den Features Oversampling und Time Stamp auszustatten. Es gibt mehrere verschiedene Arten die Synchronisierung mit den Distributed Clocks zu benutzen. Distributed Clocks werden mit einer Genauigkeit von 1ns angegeben. Der absolute Nullpunkt dieser Zahl ist der 1.1.2000 00:00. Die Zahl hat einen Umfang von 64 Bit und reicht damit für die nächsten 584 Jahre vor. Die unteren 32 Bit repräsentieren etwa 4,2 Sekunden und sind damit meistens ausreichend für die Kommunikation und die Time Stamp Eigenschaft. Einige ESCs besitzen nur 32-Bit Distributed Clocks. Diese sind kompatibel zu den 64 Bit Varianten. In einem Beispiel (Abbildung 10, Seite 28) wurden zwei Baugruppen verbunden, zwischen den 300 Busteilnehmern liegen und 120m Leitung. Laut Ansteuerung sollen beide Geräte gleichzeitig schalten. Zu sehen ist, dass ein Jitter von etwa $\pm$20ns auftritt und die Differenz im Mittel nur ca. 15ns beträgt. Als maximaler Unterschied ergibt sich ca. 40ns.

**Speicher** Ein EtherCAT Slave hat einen Adressbereich von bis zu 64 KByte. Die ersten vier KByte (0x0000-0x0FFF) sind reserviert für Register und Benutzer Speicher. Danach folgt ab der Adresse 0x1000 der Prozessspeicher (bis zu 60 KByte) für die Mailbox und die Prozessdatenobjekte. Die Größe des Speichers ist abhängig von der Baugruppe. Der ESC

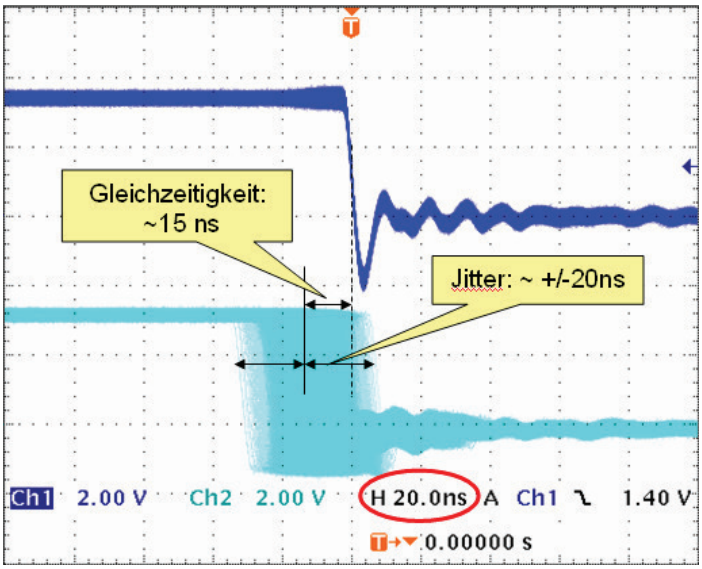



Abbildung 10: Langzeitmessung von zwei Baugruppen [6]

Adressbereich ist direkt vom EtherCAT Master und einem angeschlossenen Mikrokontroller ($\mu$C) adressierbar.

**Process Data Interface (PDI)** Das Prozessdaten Interface wird in der ESI-Datei gespeichert und in das E$^2$PROM geladen. Darüber wird der ESC konfiguriert. Ein Mikrokontroller kann über einen 8 oder 16 Bit breiten parallelen Port angeschlossen werden oder über das Serial Peripheral Interface (SPI). Die Kommunikation über den parallelen Port kann synchron oder asynchron verwendet werden. Außerdem ist es möglich digitale Ein- und Ausgänge direkt anzusprechen.

**ESI E$^2$PROM** Ein nicht flüchtiger Speicher ist notwendig, um die EtherCAT Slave Controller Konfigurationsdaten und die Baugruppen Beschreibung zu speichern. Hierfür wird typischerweise ein E$^2$PROM verwendet, welches über den I$^2$C-Bus angebunden ist. Sollte der ESC in einem FPGA implementiert sein, ist ein weiterer E$^2$PROM notwendig, um den FPGA Konfigurations Code zu speichern.

**Status LEDs** Über LEDs können einzelne Status Signale angezeigt werden. Es gibt eine RUN LED, die je nach Status der State Machine unterschiedliche Eigenschaften hat (Tabelle 3, Seite 29). Des Weiteren gibt es für jeden Port eine LED, die eine Verbindung und Aktivität anzeigt und es gibt eine LED, welche einen Fehler in der Übertragung anzeigt.

Tabelle 3: LED Status bei unterschiedlichen Baugruppen Zuständen

| LED Status | Zustand der Baugruppe |
|---|---|
| Aus | Baugruppe befindet sich im Status INIT |
| Blinkend | Baugruppe befindet sich im PRE-OPERATIONAL |
| Einzelner Puls | Baugruppe befindet sich im Status SAFE-OPERATIONAL |
| An | Baugruppe befindet sich im Status OPERATIONAL |
| Flimmern | Baugruppe befindet sich im Status BOOTSTRAP |

## 2.5 TwinCAT

TwinCAT steht für “The Windows Control and Automation Technology”. TwinCAT ist eine Echtzeit-Softwarelösung unter Windows von der Firma Beckhoff Automation GmbH. Es basiert auf der Entwicklungsumgebung CoDeSys (Control Development System) von 3S (Smart Software Solutions GmbH). Es transformiert jeden kompatiblen PC in eine Echtzeitsteuerung mit Multi-SPS, NC[8]-Achsenregelung, Programmierumgebung und Bedienstation. Zu den Eigenschaften von TwinCAT gehören unter Anderem harte Echtzeit ohne Hardwarezusätze auf jeden kompatiblen PC für SPS- und NC-Anwendungen. Es sind bis zu vier SPS Einheiten pro PC möglich. Es bestehen Anbindungen an alle gängigen Feldbusse und PC - Schnittstellen für E/A Signale. Die Fähigkeiten zur Echtzeitsteuerung werden durch TwinCAT mit der Softwareplattform Microsoft Windows verknüpft. Zu den Vorteilen der Verwendung einer Software-SPS gehören der nahezu unbeschränkte Speicherplatz für Programme und Daten, wodurch auch kommende, aufwendigere SPS-Programme ohne Probleme untergebracht werden können. Durch die niedrigere Anzahl an Komponenten ergibt sich auch eine höhere Systemzuverlässigkeit. Durch die volle Integration in das Betriebssystem kann direkt über das Betriebssystem auf PC-Ressourcen zugegriffen werden ohne Treibersoftware verwenden zu müssen.
Windows selbst ist nicht echtzeitfähig, dadurch muss das Betriebssystem für Automatisierungsaufgaben erweitert werden. Durch eine Kernel Erweiterung verfügt TwinCAT über eine genaue Zeitbasis, die mit höchster Priorität ausgeführt wird.
Es gibt zwei Entwicklungsumgebungen von TwinCAT. Das sind zum einen der TwinCAT System Manager und zum anderen die TwinCAT PLC Control.

**System Manager** Der System Manager (Abbildung 11, Seite 30) ist die Konfigurationszentrale des Systems. Der System Manager verknüpft die Soft-SPS mit der angeschlossenen Hardware. Hier werden die angeschlossenen E/A-Baugruppen geladen und mit dem SPS-

[8]Numerical Control: Eine Numerische Steuerung, mit der Werkzeugmaschinen gesteuert werden können.

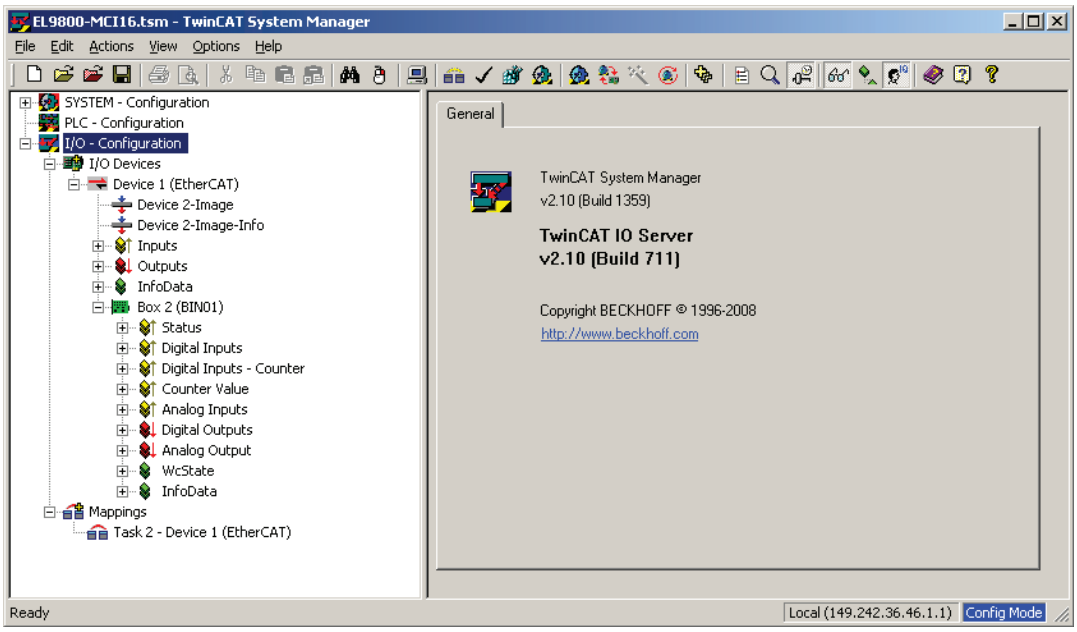



Abbildung 11: TwinCAT System Manager

Programm verlinkt. Die Einstellungen der Baugruppen werden ebenfalls über den System Manager vorgenommen. Die Informationen über die E/A-Baugruppen werden in ESI-Dateien hinterlegt. Das sind XML[9]-Konfigurationsdateien, in denen die Hardware-Eigenschaften der verschiedenen Baugruppen gespeichert werden. In einer ESI-Datei wird der Hersteller (Vendor) sowie eine oder mehrere Baugruppen (Devices) gespeichert. Zu den Informationen, welche hinterlegt werden, gehören unter Anderem die SyncManager (SM) und die Prozessdatenobjekte (PDO). Dabei wird noch zwischen TxPdo und RxPdo unterschieden. TxPdo sind Eingangsdaten, welche von den Slaves zum Master übertragen werden und RxPdo sind Ausgangsdaten, welche vom Master an die Slaves gesendet werden. Durch Verwendung von Indizes kann eine Ordner-Struktur erstellt werden, mit der die Prozessdatenobjekte übersichtlicher dargestellt werden. In der ESI-Datei kann auch ein Hersteller Logo und ein Baugruppen Symbol eingebunden werden. Diese Information kann entweder als Bitmap-Datei vorliegen oder aus Hexadezimal Werten bestehen, wobei die Hex Werte die gleiche Struktur haben müssen wie eine Bitmap-Datei. Es können auch Namen für unterschiedliche Sprachpakete von TwinCAT gespeichert werden. Auch der Product Code und die Revisions Nummer sollte hinterlegt werden, damit zwischen verschiedenen Versionen unterschieden werden kann. Der System Manager kontrolliert beim Laden der Baugruppe, ob die Num-

[9]Extensible Markup Language: Eine Programmiersprache, die Daten in einer hierarchischen Struktur speichert.

mern in der ESI-Datei mit der Nummer aus dem E$^2$PROM der Baugruppe übereinstimmen. Außerdem werden im System Manager auch die Einstellungen für die Distributed Clocks eingerichtet. Das Mapping der FMMU wird von dem System Manager erstellt und in die Slaves geladen. Der PC mit dem laufenden System Manager fungiert dann als Master. Mit dem System Manager können die States der EtherCAT Baugruppen überwacht und die aktuellen Werte an den Baugruppen abgefragt werden. Über den System Manager ist es auch möglich den DPRAM eines ASIC auszulesen oder manuell Daten in diesen zu schreiben. Ein Free-Run Mode ist ebenfalls im System Manager integriert und es können damit EtherCAT Baugruppen ohne laufendes SPS Programm getestet werden.

**PLC Control** Der PLC Control Bereich ist die Entwicklungsumgebung zur Erstellung einer Steuerung (Abbildung 12, Seite 31). Hier werden das Zielsystem und die Tasks definiert.

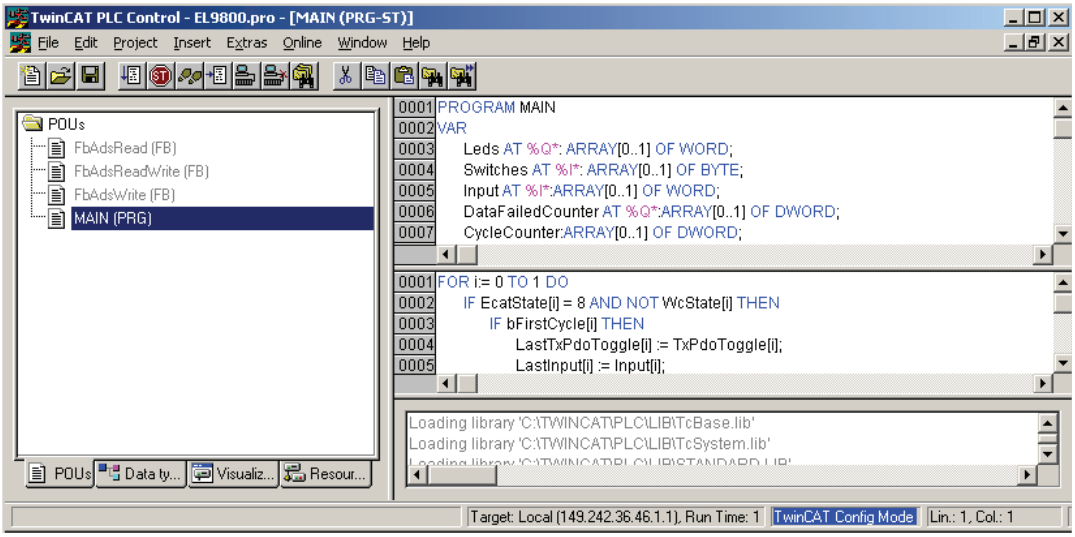



Abbildung 12: TwinCAT PLC Control

Danach kann begonnen werden in verschiedenen Sprachen zu programmieren. Zur Auswahl stehen hier zum einen die textuellen Sprachen, zu diesen gehören die Anweisungsliste (AWL, Englisch: Instruction List - IL) und der Strukturierter Text (ST, Englisch: Structured Text) und zum anderen die grafischen Sprachen wie die Ablaufsparache (AS, Englisch: Sequential Function Chart - SFC), der Funktionsplan (FUP, Englisch: Function Block Diagram - FBD), der freigrafische Funktionsplaneditor (Englisch: Continous Function Chart - CFC) und der Kontaktplan (KOP, Englisch: Ladder Diagram - LD). Nach Abschluss des Programmierens kann das Projekt übersetzt werden, um danach das Projekt in den TwinCAT System

Manager zu laden, wo das Programm mit den E/A-Baugruppen verknüpft wird. Nachdem die SPS in den Run Mode versetzt worden ist, ist es möglich sich von der PLC Control Oberfläche in die SPS einzuloggen, um das SPS-Programm zu starten und Variabelwerte zu verfolgen. Im System Manager können die Werte auch an den Baugruppen, sowie am PLC Programm eingesehen werden.

## 2.6 Nachteile

Eine Problematik von EtherCAT liegt in der Struktur selbst. Wenn ein Slave entnommen wird, muss die gesamte Konfiguration überarbeitet werden. Es können auch nicht einfach Slaves in der Reihenfolge getauscht werden. Die Position ist durch den Aufbau des Protokolls fest vorgegeben. Wenn EtherCAT Slaves direkt in ein Office Netzwerk eingebunden werden, dann wird das Netzwerk mit einer Flut an Frames überschwemmt, weil der ESC jedes Frame, egal ob es für ihn bestimmt ist oder nicht, reflektiert. Deswegen sollten alle EtherCAT Baugruppen und der Master in einem separaten Netzwerk aufgebaut werden, um maximale Funktionalität zu erhalten.

# 3 PIC32 Grundlagen

## 3.1 Technische Spezifikationen

**Allgemeines** Der PIC32MX Mikrokontroller von Microchip besitzt eine 32-Bit CPU mit einer maximalen Taktfrequenz von 80 MHz. Aus der CPU Taktfrequenz wird mit Hilfe eines Teilers die Taktfrequenz für die Peripheriegeräte des PIC32MX ermittelt. Der Mikrokontroller operiert im 3.3V Spannungsbereich und hat einen Flash Speicher von 32 bis 512 KByte. Zusätzlich können 8 bis 32 KByte an externen RAM (Random Access Memory) Speicher adressiert werden. Zur Peripherie gehören unter Anderem zwei $I^2C$ Module und zwei UART[10] Module, welche zur Übertragung von seriellen Daten verwendet werden. Ebenfalls gibt es einen parallelen Port mit 8-Bit oder 16-Bit breiter Übertragung und bis zu 16 Adressleitungen. Der PIC32MX stellt außerdem fünf 16-Bit Timer zur Verfügung, wobei je zwei Timer zu einem 32-Bit Timer zusammengefügt werden können. Auch vorhanden sind je fünf Capture Eingänge, PWM[11] Ausgänge und externe Interrupt Pins. Es können bis zu 16 Kanäle eines 10-Bit A/D Umsetzers verwendet werden. Es gibt eine 64-Pin und eine 100-Pin Variante vom PIC32MX. Die Variante mit 64 Pins kann nur einen 8-Bit breiten parallelen Port verwenden. Des Weiteren sind viele Pins, wie auch bei der Variante mit 100 Pins, mehrfach belegt.

**Interrupts** Im PIC32MX sind bis zu 96 Interrupt Quellen möglich mit insgesamt 64 Interrupt Vektoren. Es sind fünf externe Interrupt Pins verfügbar. Bei der 64-Pin Variante liegen diese auf den gleichen Pins, welche für die Capture Eingänge verwendet werden. Daher ist es bei der Variante mit 64 Pins nicht möglich fünf externe Interrupts und fünf Capture Eingänge gleichzeitig zu benutzen. Für jeden Interrupt wird eine von sieben Prioritäten und eine von vier Subprioritäten ausgewählt. Sollten zwei Interrupt die gleiche Priorität haben und beide zur gleichen Zeit aufgerufen werden, dann entscheidet die Subpriorität darüber, welcher Interrupt zuerst abgearbeitet wird. Ist diese auch gleich, wird der Interrupt mit der niedrigeren Interrupt Vektor Adresse zuerst ausgeführt.
Alle Interrupt Anfragen werden bei der positiven Flanke des Systemtaktes zwischengespeichert und in die jeweiligen Status Register geladen. Eine „1“ in einem Status Register bedeutet, dass dieser Interrupt ausgelöst wurde. Der jeweilige Interrupt wird aber nicht weiter verarbeitet, wenn das zugehörige Enable-Bit nicht aktiv ist.

[10] Universal Asynchronous Receiver Transmitter: Schnittstelle für serielle Datenübertragung
[11] Pulsweitenmodulation: Eine Modulationsart, bei der das Tastverhältnis bei konstanter Frequenz verändert werden kann.

**E/A Ports** Beim PIC32MX stehen 86 Pins (100 Pin Variante) als digitale Ein- oder Ausgänge zur Verfügung. Bei der Variante mit 64 Pins sind nur 53 Pins verfügbar. Fast alle dieser Pins sind jedoch mit anderen Funktionen mehrfach belegt. Von daher ist es wichtig zuerst die Pins, welche durch ihre Funktion benutzt werden müssen, festzulegen und danach die übrigen Pins für die digitalen Ein- und Ausgänge zu verwenden.
Es gibt sieben E/A Ports, welche mit den Buchstaben A bis G benannt sind. Jeder dieser Ports hat die drei Register TRIS, LAT und PORT. In dem Register TRIS wird die Richtung der einzelnen Pins gesetzt. Das Register LAT ist der Signalspeicher für die digitalen Ausgänge. Über das Register PORT kann mit einem Lesezugriff der entsprechende Zustand des Pins abgefragt werden. Durch ein Schreiben wird der Wert in den Signalspeicher geschrieben.

**Timer** Es stehen fünf Timer im PIC32MX zur Verfügung. Alle Timer können als 16-Bit synchron laufende Timer oder Zähler verwendet werden. Der Timer1 kann zusätzlich auch als asynchron laufender Timer mit externem Takt benutzt werden. Dafür können jeweils Timer2 und Timer3, sowie Timer4 und Timer5 zu einem 32-Bit Timer oder Zähler verknüpft werden. Die Timer oder Zähler erzeugen einen Interrupt, wenn der Wert des Zählregisters TMRx (x steht für die Timer Nummer) den gleichen Wert hat wie das zugehörige Perioden-Register PRx. Im nächsten Zyklus wird das Zählregister TMRx wieder auf Null gesetzt. In der Timer-Funktion wird das Zählregister bei jeder positiven Flanke des Peripherietaktes erhöht. Es kann hier auch ein weiterer Teiler vor den Takt gesetzt werden, um die Geschwindigkeit des Timers zu verringern. Möglich ist hier ein Teiler von bis zu 1:256. Das Zählregister wird dann nur bei jeder 256ten positiven Flanke des Peripherietaktes erhöht.

**$I^2C$** Es stehen zwei unabhängig voneinander funktionierende $I^2C$ Module im PIC32MX zur Verwendung bereit. Jedes $I^2C$ Interface besteht aus zwei Pins, dem Clock Pin (SCLx) und dem Daten Pin (SDAx). Es werden 7-Bit und 10-Bit Adressen unterstützt und es kann bidirektional kommuniziert werden. Die Module können für $I^2C$ Slave oder Master Operationen verwendet werden.

**Allgemeines zum $I^2C$-Bus** Der Inter-Integrated-Circuit-Bus ($I^2C$) wurde von Philips entwickelt und ermöglicht die Kommunikation zwischen einzelnen integrierten Schaltungen. Der Bus wird auch 2-Draht-Bus genannt, da er mit nur zwei Leitungen auskommt. Dem Takt und der Datenleitung. Die Daten werden seriell über die Datenleitung übertragen. Die einzelnen Bausteine werden über eine zehn Bit breite Adresse ausgewählt (früher 7 Bit breit). Die Adresse kann für jeden IC separat festgelegt werden, so dass mehrere Bausteine vom

gleichen Typ am selben $I^2C$-Bus sein können. Es sind jedoch nur die letzten drei Bit veränderbar. Nach einer Start Bedingung sendet der Master die Adress-Bits. Daraufhin können Daten zwischen dem Master und dem Baustein mit der angeforderten Adresse ausgetauscht werden. Nur der Master kann die Kommunikation beenden. Die Daten können derzeit mit bis zu 3,4 MBit/s übertragen werden, typisch sind jedoch 100 und 400 KBit/s. Der große Vorteil von $I^2C$ ist, dass nur zwei Pins für die Kommunikation notwendig sind.

**UART** Es sind zwei UART Module im PIC32MX integriert. Diese Module können im Voll-Duplex verfahren über einen asynchronen Kanal mit externer Peripherie oder einem Computer kommunizieren. Typische Kommunikationsprotokolle sind RS-232, RS-485, LIN 1.2 und IrDA. Es werden ebenfalls die Hardware Optionen für die Flußkontrolle unterstützt. Dazu gehören die Pins CTS (Clear To Send) und RTS (Request To Send). 8-Bit oder 9-Bit Datenübertragung ist möglich. Bei einer 8-Bit Übertragung wird eine Parität unterstützt und es sind ein oder zwei Stop Bits verwendbar. Eine Baudrate von 76 bps (Bits pro Sekunde) bis zu einer Baudrate von 20 MBps (Mega Bit per second) können mit dem integrierten Baudratengenerator erstellt werden. Außerdem gibt es separate Interrupts für Senden und Empfangen.

**Parallel Port** Der parallele Port des PIC32MX kann entweder mit einem 8-Bit oder einem 16-Bit Interface arbeiten. Die Variante mit 64 Pins unterstützt nur das 8-Bit Interface. Bis zu 16 Adressleitungen und zwei Chip-Select (CS) Leitungen sind verfügbar. Das Modul besitzt eine Funktion zur automatische Adresserhöhung oder -verringerung. Es ist auch möglich die Adress- und Datensignale auf die gleichen Leitungen zu legen und damit zu multiplexen. Wie in Abbildung 13 auf Seite 36 zu sehen ist, können nicht die vollen 64 KByte adressiert werden, da die Chip-Select Signale gleichzeitig die Adressleitungen A15 (=CS2) und A14 (=CS1) belegen. Wenn zwei Geräte angeschlossen werden sollen, dann können in beiden Geräten je 16 KByte adressiert werden. Wird nur ein Gerät verwendet, dann ist es möglich in diesem bis zu 32 KByte zu adressieren. Die Adresse 0x8000 entspricht dann der Adresse 0x0000 in dem Gerät.

**A/D Umsetzer** Der Pic32MX besitzt einen 10-Bit Analog-Digital Umsetzer (ADU), welcher eine Umsetzgeschwindigkeit von bis zu 500 ksps (kilo sample per second) haben kann. Es können bis zu 16 analoge Eingänge angeschlossen werden, welche über einen Multiplexer mit dem A/D-Umsetzer verbunden werden. Es ist möglich eine externe Referenzspannung zu verwenden. Der ADU hat einen 16 Wort großen Ergebnispuffer, in dem die digitalen Werte

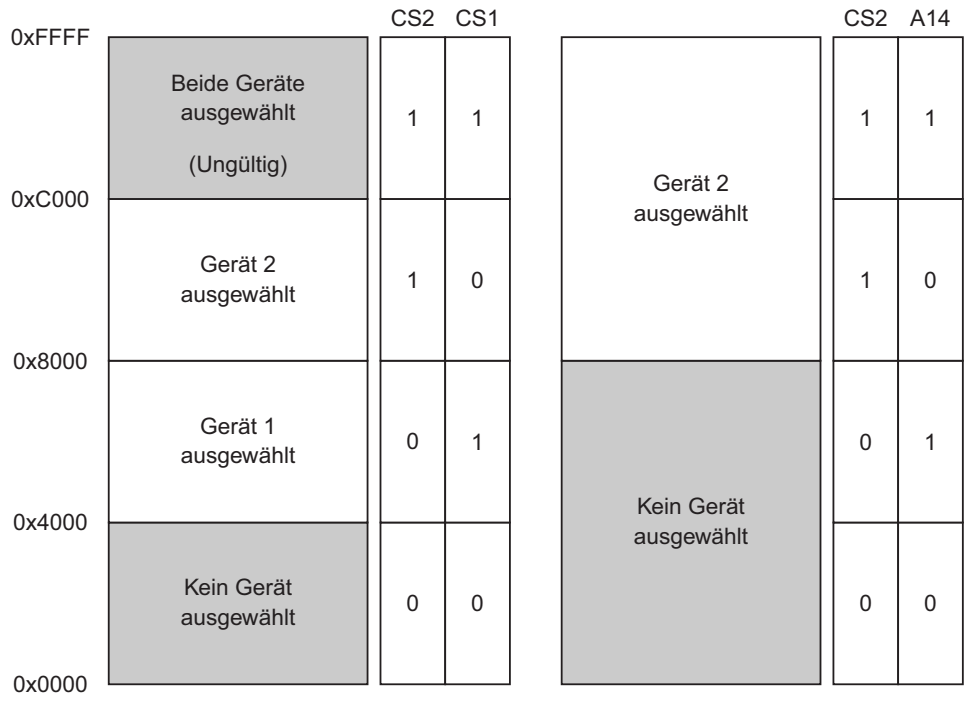



Abbildung 13: Darstellung der Adressverwaltung mit Hilfe der Chip-Select Signale [14]

zwischengespeichert werden können. Dieser Puffer kann in zwei acht Wort große Zwischenspeicher aufgeteilt werden. Dadurch verringert sich die Gefahr, dass die Daten während des Auslesens wieder überschrieben werden, weil die Puffer abwechselnd verwendet werden. Die Analogwertumrechnung besteht aus zwei Vorgängen. Der Erfassung des Wertes, bei der für eine ausreichende Zeitspanne der analoge Eingangspin über den Multiplexer mit dem “Sample und Hold”-Verstärker verbunden ist und der im Anschluss erfolgenden Umsetzung des Analogwertes in einen digitalen Wert. Dafür wird der Pin wieder von dem “Sample und Hold”-Verstärker gelöst, um keine Änderungen am Eingang zu haben. Der ADU hat auch die Möglichkeit in einem Scan Mode mehrere analoge Eingänge hintereinander abzufragen. Die Zeit der Erfassung kann manuell festgelegt oder automatisch durch das Modul vorgenommen werden. Für die Umsetzung werden pro Bit je ein ADU Taktzyklus (TAD) benötigt plus zwei zusätzliche Zyklen. Die Umsetzung nimmt somit zwölf TAD Zyklen in Anspruch. Ein TAD Zyklus kann entweder den internen Oszillator oder den Taktzyklus der Peripherie (PBClock) als Quelle haben. Es gibt zwei Betriebsarten des ADU. Das ist zum einen der

Auto-Sample Mode und zum anderen der Manual Mode. Im Auto-Sample Mode wird der ADU nach jeder Umsetzung eines analogen Wertes automatisch neu gestartet. Im Manual Mode muss die Software die Erfassung beginnen und beenden. Mit Hilfe des ADU Control Register 2 (AD1CON2) kann die Anzahl der Messwerte pro Interrupt festgelegt werden. Mit jedem Interrupt kann das Schreiben der Messwerte neu begonnen werden. Wird nur ein Puffer verwendet, beginnt der ADU wieder in das erste Wort die neuen Werte rein zu schreiben. Werden zwei Zwischenspeicher verwendet, springt der ADU mit jedem Interrupt zwischen diesen beiden Speichern hin und her. Mit Hilfe eines Statusbits ist es möglich zu erkennen, in welchen Puffer gerade geschrieben wird, damit aus dem anderen gelesen werden kann.

## 3.2 MPLAB IDE

MPLAB IDE ist eine kostenlose Entwicklungsumgebung von Microchip Technology Inc. Sie wird zur Erstellung von Anwendungen mit den Microchip Mikrokontrollern PIC und dsPIC verwendet. Das Programm wird als Editor für den C-Quellcode, welcher mit Hilfe eines über die Toolsuite ausgewählten C-Compiler kompiliert werden kann. Gegen eine Lizenzgebühr kann die Quellcode Optimierung des C-Compilers freigeschaltet werden. Es ist ebenfalls möglich

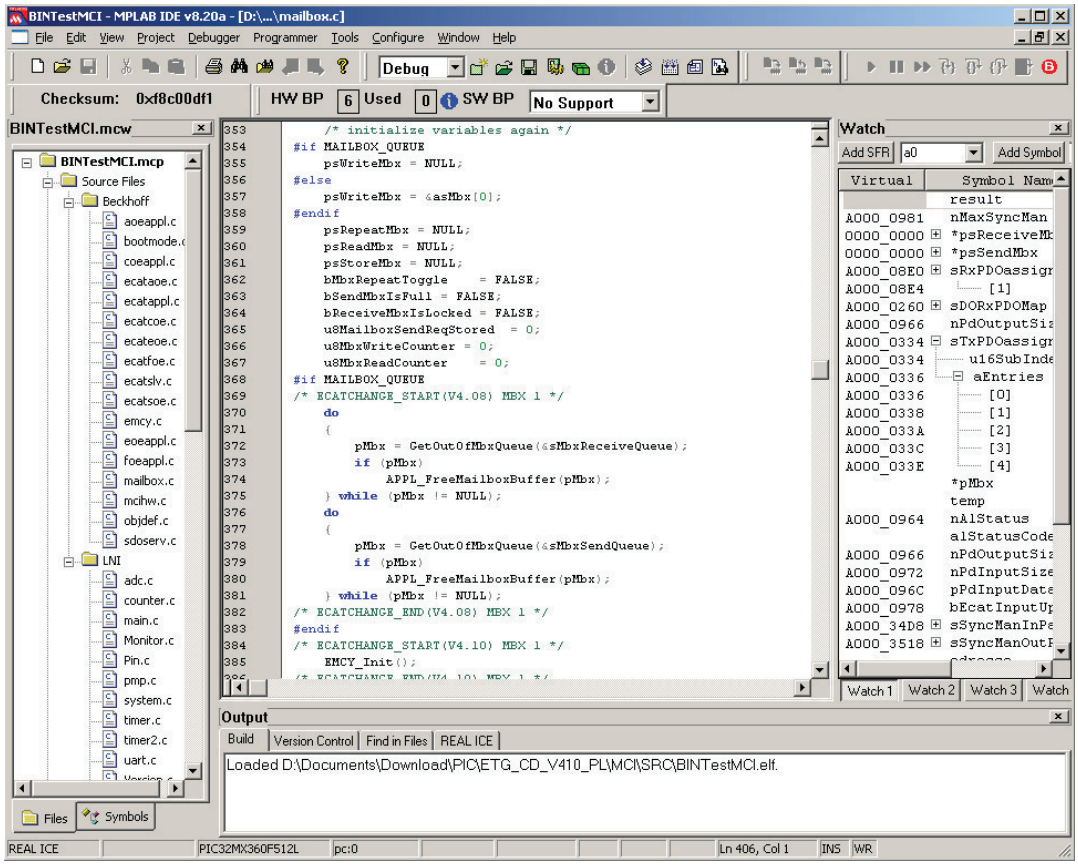



Abbildung 14: MPLAB 8.20a Entwicklungsumgebung

einen PIC oder dsPIC direkt mit MPLAB zu programmieren. Danach gibt es diverse Debug Möglichkeiten, um das Programm direkt in der Hardware zu testen. Zum Debuggen können bis zu sechs Breakpoints gesetzt werden. Es ist jedoch ein Breakpoint nötig, um das Programm zeilenweise auszuführen. In Abbildung 14 auf Seite 38 ist die Standard Oberfläche von MPLAB 8.20a zu sehen. Auf der linken Seite sind die dem Projekt zugeordneten Dateien abgelegt. Diese können für eine bessere Struktur in Unterordner verschoben werden. Auf der rechten Seite können Variablen oder Spezial Funktion Register (SFR) angezeigt werden. In

der Mitte ist der Bereich, in dem Dateien editiert werden können. MPLAB verfügt über Syntax Highlighting, bei dem Kommentare grün, Schlüsselwörter blau und weiterer Text schwarz angezeigt wird. Im unteren Bereich des Fensters werden unter anderem Warnungen und Fehler beim Kompilieren angezeigt, sowie weitere Debugging Hilfen. Um einen Mikrokontroller zu programmieren und zu debuggen können unterschiedliche Tools verwendet werden. Möglichkeiten wären zum Beispiel die in-circuit Debugger MPLAB ICD 2 und MPLAB ICD 3 oder der in-circuit Emulator MPLAB REAL ICE.

# 4 Entwurf

## 4.1 Auswahl des ESC

Um einen Mikrokontroller mit einer Soft-SPS über EtherCAT zu verbinden, ist ein EtherCAT Slave Controller (ESC) nötig. Hierfür kann entweder ein Application Specific Integrated Chip (ASIC) verwendet werden oder ein FPGA. ASICs werden hauptsächlich von zwei Firmen hergestellt. Es gibt den ET1100 und den ET1200 von Beckhoff und die netX Serie von Hilscher. Diese unterscheiden sich in Kriterien wie Größe des DPRAM, Anzahl der SyncManager (SM) und Fieldbus Memory Management Units (FMMU) oder der Anzahl und Beschaffenheit der Ports. Außerdem variieren sie in der Art des $\mu$C Interfaces. In Tabelle 4 auf Seite 40 sind die Eigenschaften im Einzelnen aufgelistet.

Tabelle 4: EtherCAT Slave Controller (ASIC)

| Name | ET1100 | ET1200 | netX 5 | netX 100 |
|---|---|---|---|---|
| Typ | ASIC | ASIC | ASIC | ASIC |
| Hersteller | Beckhoff | Beckhoff | Hilscher | Hilscher |
| $\mu$C Interface | seriell/parallel | seriell | seriell, parallel | $\mu$C-Bus |
| Digitale E/A | 32 | 8-16 | 16 | 16(GPIO) |
| DPRAM | 8 kByte | 1kByte | 6 kByte | 656 Byte |
| SyncManager | 8 | 4 | 8 | 4 |
| FMMU | 8 | 3 | 8 | 3 |
| Anzahl Ports | 2-4 (MII/EBUS) | 2-3 (EBUS/1xMII) | 2 (MII) | 2 (100BASE-TX) |

Da laut Vorgabe die Kommunikation über den parallelen Port durchgeführt werden soll, stehen nur noch der ASIC ET1100 und netX 5 zur Auswahl. Das EtherCAT Protokoll soll über den EBUS an die Soft-SPS übermittelt werden, deswegen wurde der ASIC ET1100 als ESC ausgewählt.
Die Beckhoff Automation GmbH bietet das Piggybackboard FB1111-014x (Abbildung 15, Seite 41) an. Es gibt drei mögliche Varianten. Die Variante 0142 ist für das Testen der digitalen Ein- und Ausgänge vorgesehen. Für serielle Kommunikation über das Serial Peripheral Interface (SPI) ist das Modell 0141 bestimmt. Für dieses Projekt wurde jedoch die Variante 0140 benötigt, welche für eine Kommunikation mit einem Mikrokontroller über den parallelen Port vorkonfiguriert ist. Mit Hilfe der Piggybackboards kann die Verbindung zwischen Mikrokontroller, ASIC und Soft-SPS getestet werden.

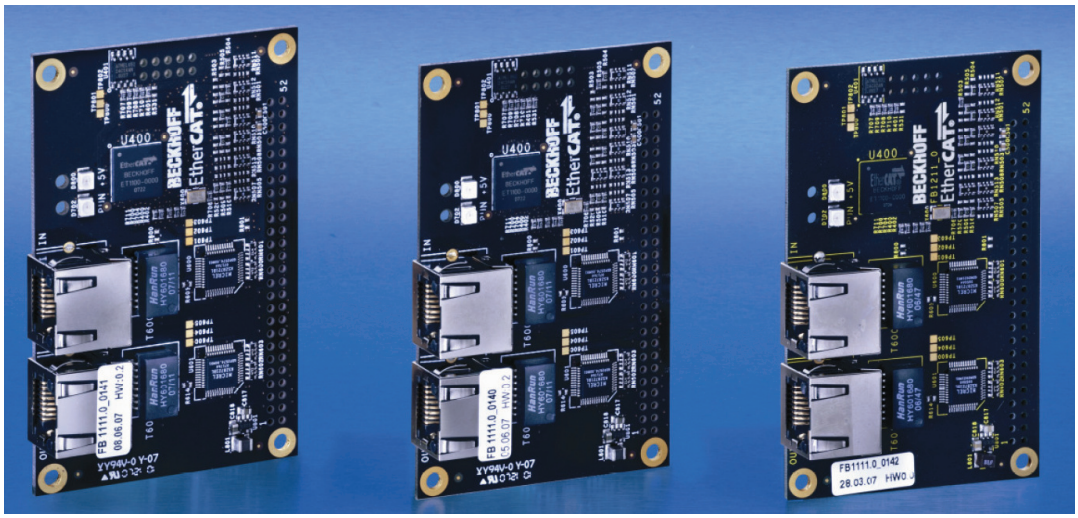



Abbildung 15: Die Piggybackboards der FB1111 Serie [2]

## 4.2 Auswahl des Mikrokontrollers

Die Vorgaben für die Auswahl des Mikrokontrollers waren sehr weit gefasst. Der Mikrokontroller sollte über einen 16- oder 32-Bit Prozessor und einen 16-Bit breiten parallelen Port verfügen. Auf Grund des parallelen Ports können die Daten schneller übertragen werden als bei einer seriellen Schnittstelle. Weitere wichtige Eigenschaften waren ein Analog/Digital-Umsetzer, mehrere Timer, externe Interrupts, eine UART Schnittstelle, zwei $I^2C$ Module und genügend digitale Ein- und Ausgänge für die Applikation. Da es mehrere Mikrokontroller gibt, welche diesen Anforderungen entsprechen, wurde entschieden, dass ein Mikrokontroller gewählt wird, der lange verfügbar ist und mit dem Teile des Quellcodes aus anderen Projekten wiederverwendet werden können. Der PIC32MX wurde im November 2007 in den Markt eingeführt und beruht auf einer ähnlichen Struktur wie andere PIC Mikrokontroller. Da zur PIC-Programmierung bereits Know-How vorhanden ist und der Hersteller Microchip für hohe Verfügbarkeit steht, wurde dieser als Mikrokontroller ausgewählt.

Um zu zeigen, dass der parallele Port die schnellere Lösung im Vergleich zum seriellen Port ist, sind Informationen aus den Datenblättern des PIC32MX [14] und des ASIC ET1100 [3] verwendet worden, um die theoretischen Übertragungszeiten zu prüfen. In der PC-Technik wird häufiger der serielle Port verwendet. Die Gründe dafür sind jedoch durch eine andere Problematik zu erklären. Gründe hierfür sind die hohe Anzahl an benötigten Leitungen für die parallele Kommunikation und die sich dadurch ergebenden höheren Fertigungskosten.

Diese Probleme sind für die BIN-Karte nicht ausschlaggebend, da die Produktionszahlen sehr gering sind.
Die Berechnungen sind auf einem Peripherietakt von 20MHz basiert. Da die meisten Abfragen von Daten im ASIC Wortweise geschehen, wird der Vergleich im Lesen von zwei Byte durchgeführt. Laut Datenblatt des ASIC werden für einen Lesezugriff von zwei Byte eine Zeit von maximal 560ns benötigt. Zwischen zwei Zugriffen müssen mindestens 10ns liegen. Damit wird für das Lesen von zwei Byte 570ns benötigt. Wird die doppelte Menge an Daten gelesen, wird auch die doppelte Zeit benötigt.
Bei dem Zugriff über das Serial Peripheral Interface[12] (SPI) erwartet der ASIC zuerst drei Byte für die Adresse und die Zugriffsart. Nach diesen Adressinformationen wird ein Byte gewartet, bevor die Daten übertragen werden. Bei zwei Datenbytes sind das somit insgesamt sechs Byte, die zu übertragen sind. Aus dem Datenblatt des PIC32 geht hervor, dass der schnellste Takt für die serielle Schnittstelle der halbe Peripherietakt ist. Daraus ergibt sich eine Übertragungsdauer von 800ns pro Byte und da sechs Byte transferiert werden, sind 4800ns nötig. Damit benötigt der serielle Port mehr als die 8-fache Zeit. Umso mehr Byte pro Zugriff übertragen werden, desto geringer wird der Unterschied. Da aber hauptsächlich Byte- und Wortzugriffe durchgeführt werden, ist der parallele Port die deutlich schnellere Variante.

[12] Ein Bus-System, welches eine serielle, synchrone Kommunikation ermöglicht.

## 4.3 BIN-Karte

In dem Blockschaltplan (Abbildung 16, Seite 44) ist der Aufbau der BIN-Karte dargestellt. Der Schaltplan wird größtenteils durch einen Kollegen erstellt. Der Schaltplan der E/A-Karte befindet sich im Anhang A1. Bestandteil dieses Projektes ist die Pin Zuweisung für den PIC32. Die Anbindung der Soft-SPS erfolgt über die Backplane. Das EtherCAT Protokoll wird über einen EBUS übertragen. Des Weiteren ist ein $I^2C$ Bus sowie die SYNC Signale CUT und INC über die Backplane des Racks mit der BIN-Karte verbunden. Die SYNC Signale werden über einen Maschinentakt erzeugt. Zwischen zwei CUT Pulsen sind immer genau 250 INC Pulse. Der EBUS wird auf der BIN-Karte mit dem EtherCAT Slave Controller ET1100 verbunden, welcher über einen parallelen Port mit dem PIC32MX Mikrokontroller kommuniziert. An einem weiteren $I^2C$ Bus sind ein Temperatur-Sensor und ein Digital-Analog Umsetzer angeschlossen. Des weiteren gibt es eine serielle RS232-Schnittstelle, welche von dem UART Modul des Mikrokontrollers gesteuert wird. Der PIC32 ist mit den digitalen Eingängen verbunden. Diese werden jedoch zuerst über ein Interface auf den korrekten Spannungsbereich gebracht. Die Zustände der unterschiedlichen Spannungsniveaus werden mit Hilfe von vier Voltage Status Signalen angezeigt. Außerdem wird die Erstellung der Analogkarte, die aus Platzgründen nötig ist, und des Interfaces für die digitalen Eingänge nicht in diesem Teil des Projekts durchgeführt. Die Hardware ist von einem vorherigem Modell übernommen worden. Ein 3D Modell der BIN-Karte ist in Abbildung 17 auf Seite 45 und in Abbildung 18 auf Seite 45 zu sehen. Nur die Anbindung an den Mikrokontroller und die Auswertung in der Software ist Bestandteil des Projekts.

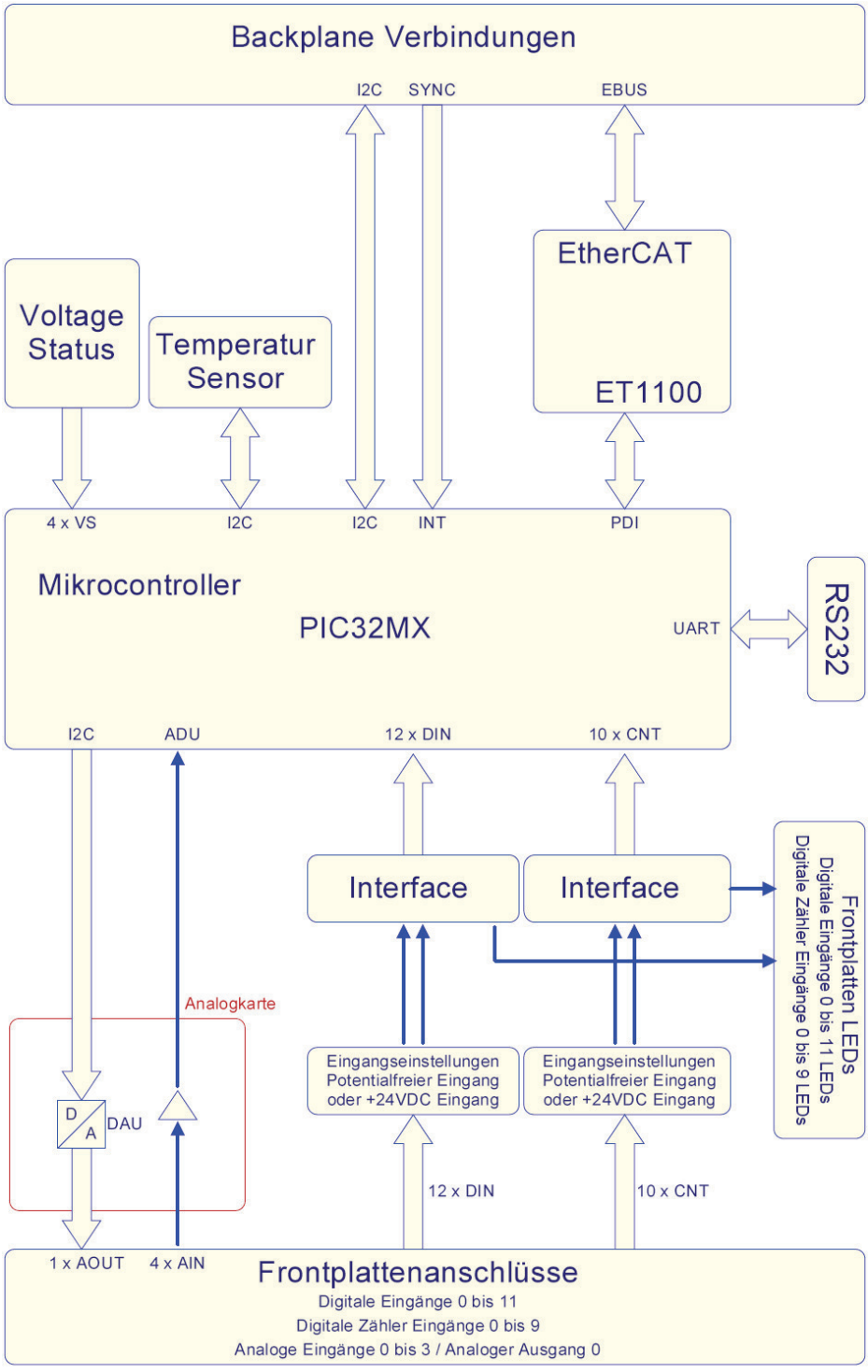



Abbildung 16: Blockschaltplan der E/A-Karte BIN

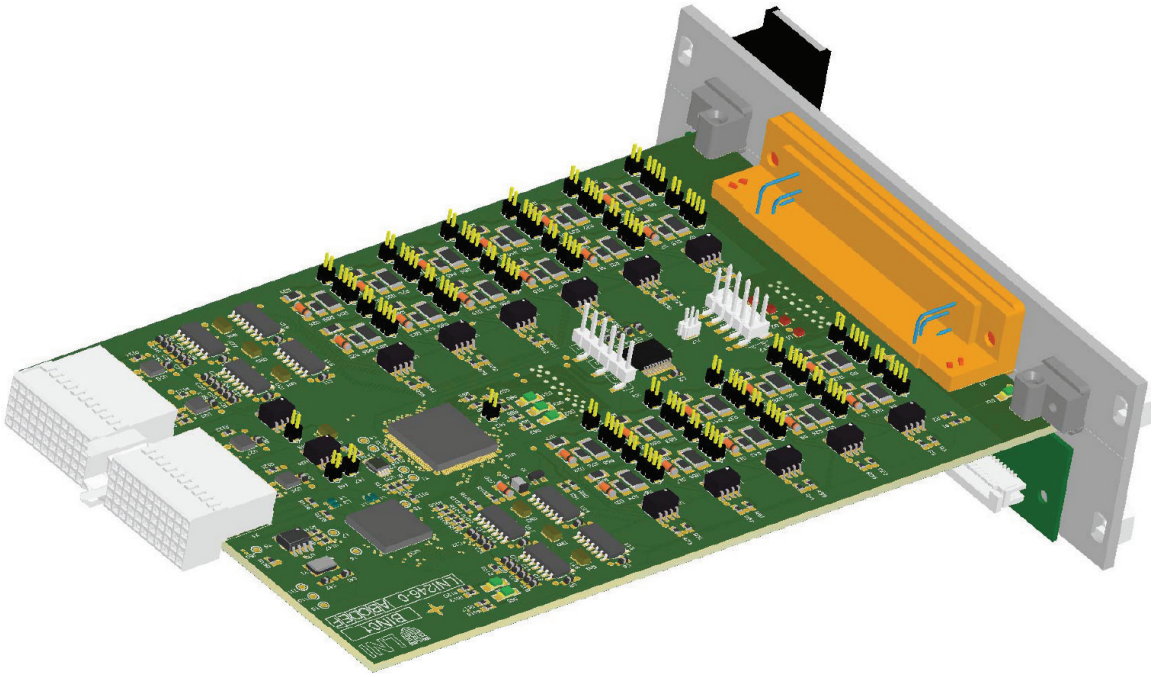


Abbildung 17: Modell der BIN Karte in 3D (Oberseite)

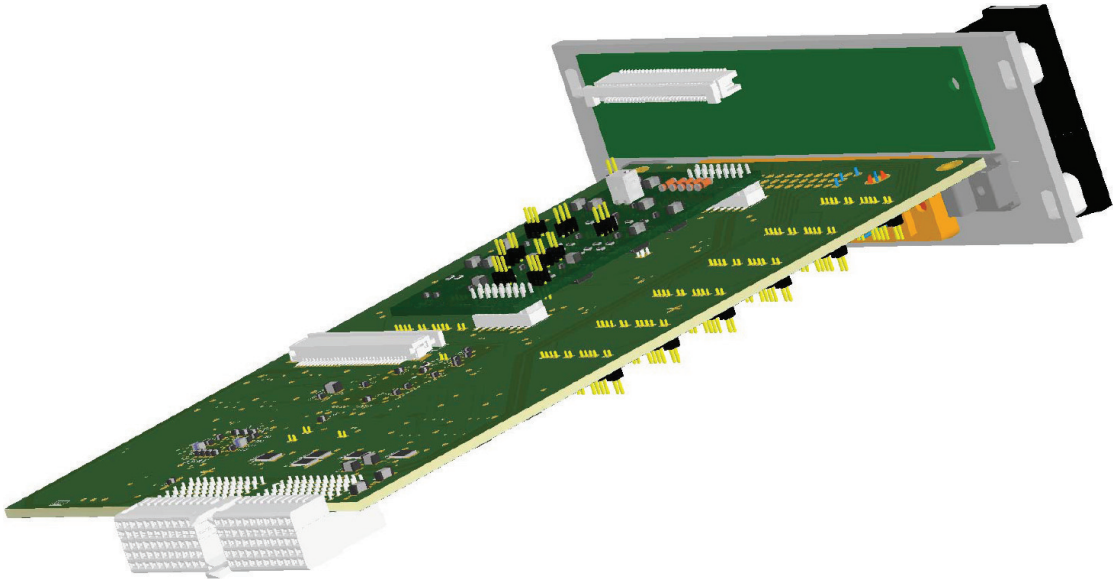


Abbildung 18: Modell der BIN Karte in 3D (Unterseite)

## 4.4 Ansatz

**Hardware** Da zu Beginn der Arbeit die BIN-Karte noch im Entwicklungsstadium gewesen ist, muss eine Test-Hardware entworfen werden, um das Mikrokontrollerprogramm testen zu können und die ESI-Datei zu erstellen. Parallel wurde das Layout entwickelt und die Platine gefertigt. Diese Arbeiten sind jedoch nicht in meinen Aufgabenbereich gefallen. Die Test-Hardware sollte nach Möglichkeit die gleichen Voraussetzungen haben, wie die tatsächliche E/A-Karte später. Es wird das PIC32 Starter Kit mit dem PIC32 I/O Expansion Board verwendet. Auf dieses Expansion Board können bis zu zwei Erweiterungskarten gesteckt werden. Über eine Steckkarte soll die Verbindung zu einem Test-Rechner mit der Soft-SPS realisiert werden, wofür das Piggybackboard FB1111-0140 von Beckhoff verwendet wird. Die parallele Schnittstelle zwischen einem EtherCAT Slave Controller und einem 16-Bit Mikrokontroller wird in Abbildung 19 auf Seite 46 dargestellt. Da ein 16-Bit breiter paralleler

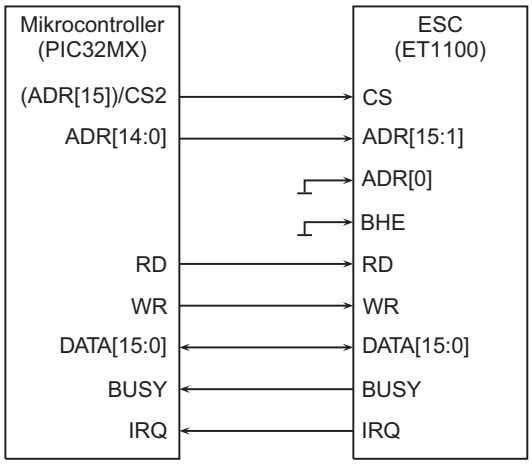



Abbildung 19: ESC Mikrokontroller Interface [3]

Port verwendet wird, wird keine Byte-Adressierung verwendet. Deswegen wird das kleinste Adress-Bit (A[0]) fest mit 0V verbunden. Auch das BHE[13]-Bit wird hier nicht verwendet und wird daher ebenfalls auf Masse gelegt. Das höchstwertigste Adress-Bit ist beim PIC32MX gleichzeitig das Chip-Select Bit. Diese Eigenschaft bedeutet, dass der Dual-Port-RAM (DPRAM) des ASIC vom PIC32 aus über die Adresse 0x8000 angesprochen wird. Um dennoch den kompletten DPRAM adressieren zu können, werden die Adress-Bits um ein Bit verschoben. Das führt dazu, dass die Adresse 0x8000 die beiden Bytes 0x0000 und 0x0001

[13]Byte High Enable Bit: Mit diesem Bit wird gewöhnlich zwischen Byte und Wort Zugriff unterschieden.

anspricht und die Adresse 0x8001 die beiden Bytes 0x0002 und 0x0003. Dadurch können die gesamten 64 kByte des DPRAM verwendet werden. In der Mikrokontroller-Software muss diese Adressverschiebung berücksichtigt werden. Der ASIC wird vom Mikrokontroller als externer RAM gesehen und muss auch so verwendet werden, weswegen die Lese- (RD - Read) und Schreib-Signale (WR - Write) mit dem ESC verbunden werden müssen. Der 16-Bit breite Datenbus kann in beide Richtungen kommunizieren. Vom EtherCAT Slave Controller kommen zusätzlich noch die beiden Signale BUSY und IRQ. Das BUSY Signal zeigt an, wenn auf den DPRAM des ESC nicht zugegriffen werden kann. IRQ ist das EtherCAT Interrupt Request Signal, mit dem eine Kommunikation initialisiert werden kann.

Auf der anderen Steckkarte soll eine RS232 Schnittstelle mit Verbindung zum Test-Rechner aufgebaut werden. Mit Hilfe der seriellen Schnittstelle sollen die Peripherie Geräte des PIC32MX getestet werden. Hierfür wird ein RS232-Baustein benötigt. Dieser Baustein verbindet das UART Modul des Mikrokontrollers mit der seriellen Schnittstelle eines Computers. Die Ausgangssignale werden vom Baustein auf das benötigte RS232-Verhalten gebracht. Das serielle Modul in einem PC erwartet für eine logische Null eine Spannung von -3V bis -15V. Eine logische Eins wird durch ein Spannungsniveau von +3V bis +15V dargestellt. Der MAX232A Baustein wandelt die Signale, welche vom Mikrokontroller kommen, nach diesen Vorgaben um. Der Schaltplan für die Steckkarte zur seriellen Kommunikation ist in Abbildung 20 auf Seite 47 zu sehen. Der Jumper P1 wird verwendet, weil nicht bekannt ist, welche Art von Kabel verwendet wird. Es gibt die Möglichkeit, dass das RX und TX Signal

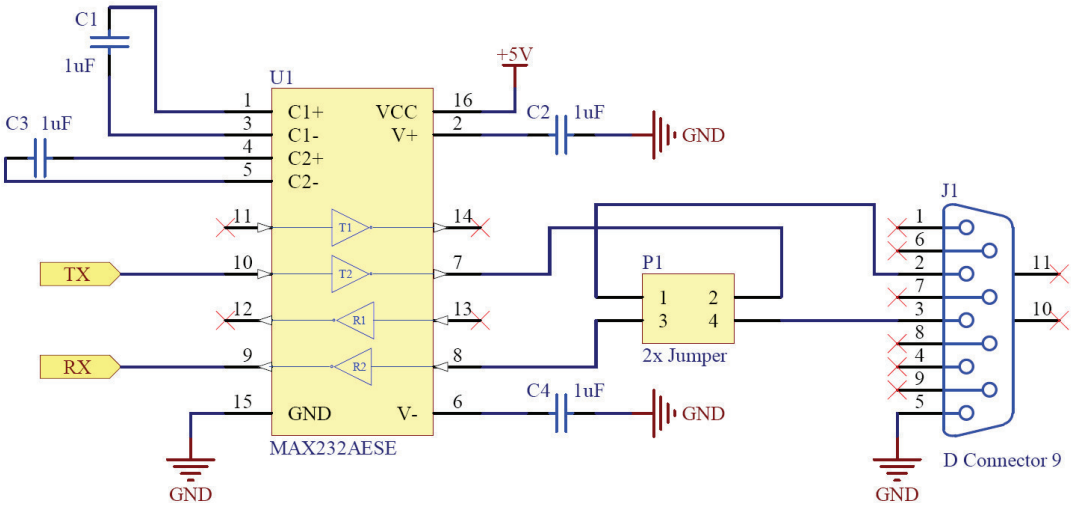



Abbildung 20: Schaltplan der Zusatzkarte für die serielle RS232 Schnittstelle

auf Pin 2 oder 3 des Sub-D Stecker liegen. Um nicht das Kabel tauschen zu müssen, kann die Position der Jumper getauscht werden. Haben die Jumper die Position 1-2 und 3-4, so ist das TX Signal des Mikrokontrollers mit Pin 2 verbunden und das RX Signal mit Pin 3. Werden die Jumper geändert, so dass die Verbindungen 1-3 und 2-4 verwendet werden, dann werden die Pins getauscht. Das TX Signal ist nun mit Pin 3 verbunden und das RX Signal mit Pin 2. Auf diese Weise kann leicht ein Pin-Tausch vorgenommen werden, ohne dass die Hardware geändert werden muss.

**Software - PIC32 spezifisch** Für den Softwareteil des Projekts muss zuerst die zu verwendende Peripherie des Mikrokontrollers getestet werden. Hierzu gehören die Timer, der A/D-Umsetzer, der parallele Port und das UART Modul. Dieses sollte als erstes in Betrieb genommen werden, damit über das HyperTerminal die anderen Module getestet werden können. Die serielle Schnittstelle soll mit 38400 Baud/s kommunizieren. Außerdem sollen acht Daten Bits, ein Stopp Bit und kein Bit für die Parität übertragen werden. Diese Einstellungen sollen mit einer Funktion initialisiert werden. Weitere Funktionen werden benötigt um einzelne Zeichen auszugeben, um eine Zeichenkette zu übertragen oder um ein Zeichen, welches vom Test-PC übertragen wird, zu empfangen. Dazu kommt eine Interrupt Routine, die die verschiedenen UART Interrupts verarbeiten wird.
Die Funktion der Timer soll festgestellt werden. Die Frequenz der Timer soll über Definitionen schnell zu ändern sein, damit bei einer späteren Frequenzänderung nicht neu gerechnet werden muss. Ein Timer soll kontinuierlich in festen Abständen einen Interrupt auslösen, der für diverse Funktionen genutzt werden kann.
Der A/D-Umsetzer des Mikrokontrollers hat zehn Bit. Der ADU soll die vier analogen Eingänge nacheinander umsetzen und anschließend einen Interrupt auslösen. Die Konsistenz der Daten muss gewährleistet sein.
Der parallele Port wird mit dem DPRAM des ASIC getestet. Es sollen alle Adressen angesprochen und Daten gelesen und geschrieben werden können. Es sind Funktionen zu schreiben, um in ein beliebiges Byte oder Wort des DPRAM zu lesen oder zu schreiben. Auch die oben genannte Adressverschiebung muss hierbei Berücksichtigung finden.

**Software - SSC spezifisch** Nachdem die Peripheriemodule des Mikrokontrollers getestet worden sind, muss der EtherCAT Slave Sample Code (SSC) an die Hardware angepasst werden. Der Slave Sample Code stellt eine Grundlage dar, mit der ein Mikrokontroller mit einem ASIC kommunizieren kann. Der von der EtherCAT Technology Group erhaltene SSC

ist im Anhang A2. Dieser Code muss an einigen Stellen an die Hardware angepasst werden. Außerdem müssen Definitionen gemacht werden, die auf den PIC32MX zugeschnitten sind. Die Kommunikationsprotokolle müssen, wenn sie benötigt werden, aktiviert werden. Benötigt werden für dieses Projekt das CoE (CAN over EtherCAT) Protokoll und für Testzwecke das AoE (ADS[14] over EtherCAT) Protokoll. Zusätzlich muss das Mikrokontroller Interface (MCI) auf den PIC32 zugeschnitten werden. Die Datei für die Hardwareinitialisierung wird dabei auf den PIC abgestimmt. Die Verwendung des parallelen Ports ist für den Datenaustausch einzubinden. Da der SSC mit einem Zeiger System arbeitet, welches nicht vom PIC32MX unterstützt wird, müssen die Zugriffe auf den DPRAM mit Hilfe von Funktionen ausgeführt werden. Deswegen werden alle Zeiger-Zugriffe auf den ESC mit einer entsprechenden Funktion ersetzt. Außerdem sind die Prozessdatenobjekte (PDO) auf die Anforderungen der BIN-Karte im Slave Sample Code anzupassen. Hierfür ist die vorhandene Struktur der Prozessdatenobjekte durch die Struktur der Daten der BIN-Karte zu ersetzen.

**ESI-Datei** Die EtherCAT Slave Information (ESI) Datei muss für die BIN-Karte erstellt werden. Alle Hauni LNI Electronics Baugruppen werden in einer XML Konfigurationsdatei gespeichert und können auf diese Weise leicht mit jeder Soft-SPS importiert werden. In der ESI-Datei müssen der Hersteller (Hauni LNI) und die LNI Baugruppen gespeichert werden. Es soll auch das Logo der Firma, sowie ein Symbol für die Baugruppen erstellt werden.

**RxPdo** (2)

| | = Fixed | = Mandatory | = Sm | ⟨⟩ Index | ⟨⟩ Name | ⟨⟩ Entry |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | #x1601 | Digital Outputs | Entry (2) |
| 2 | 1 | 1 | 2 | #x1602 | Analog Output | Entry (1) |

**TxPdo** (5)

| | = Fixed | = Mandatory | = Sm | ⟨⟩ Index | ⟨⟩ Name | ⟨⟩ Entry |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | #x1a00 | Status | Entry (7) |
| 2 | 1 | 1 | 3 | #x1a01 | Digital Inputs | Entry (13) |
| 3 | 1 | 1 | 3 | #x1a02 | Digital Inputs - Counter | Entry (11) |
| 4 | 1 | 1 | 3 | #x1a03 | Counter Value | Entry (10) |
| 5 | 1 | 1 | 3 | #x1a04 | Analog Inputs | Entry (4) |

Abbildung 21: XML-Datei in Gitterstruktur

Die Logos und Symbole haben jeweils 16x14 Pixel. Jede Baugruppe hat einen eigenen Datensatz, in denen unterschiedliche Eigenschaften definiert werden können. Es werden die Startadressen der SyncManager im Bereich Sm festgelegt. Es ist darauf zu achten, dass diese genügend Abstand haben. Die Länge eines SyncManagers im Buffered Mode ist dreimal so groß, wie die zu übertragende Datenmenge, weil die drei Puffer im Speicher direkt

[14] Automation Device Specification: Ein von der Firma Beckhoff verwendetes EtherCAT Protokoll

aufeinander folgen. Der Zugriff erfolgt jedoch immer über den kleinsten Adressbereich. Die SyncManager Kontrol-Register werden ebenfalls in diesem Bereich festgelegt. Die Prozessdatenobjekte sind der wesentliche Bestandteil dieser Daten. Bei den PDOs kann durch Verwendung von Indizes eine Ordnerstruktur erstellt werden, um die Übersichtlichkeit im System Manager zu verbessern. Den Prozessdatenobjekten muss ein SyncManager zugewiesen werden, damit der TwinCAT System Manager Ein- und Ausgänge unterscheiden kann. Zusätzlich kann jedem Index ein Name gegeben werden, welcher als Ordnername im System Manager auftaucht. Jedem Index werden danach Einträge zugewiesen. Diese enthalten eine weitere Index Nummer, Sub-Index Nummer, die Bit Länge, den Daten Typ und einen Namen für den Eintrag. Der Name taucht ebenfalls wieder im System Manager für die jeweilige Variable auf. Es wird das Programm Altova DiffDog verwendet, um die ESI-Dateien zu editieren.

```
<Device Physics="YY">
  <Type ProductCode="#x000000003" RevisionNo="#x01">BIN01</Type>
  <Name LcId="1033"><![CDATA[BIN01]]></Name>
  <Name LcId="1031"><![CDATA[BIN01]]></Name>
  <GroupType>LNI Device</GroupType>
  <Fmmu>Outputs</Fmmu>
  <Fmmu>Inputs</Fmmu>
  <Sm DefaultSize="128" StartAddress="#x1000" ControlByte="#x26" Enable="1">MBoxOut</Sm>
  <Sm DefaultSize="128" StartAddress="#x1080" ControlByte="#x22" Enable="1">MBoxIn</Sm>
  <Sm StartAddress="#x1100" ControlByte="#x24" Enable="1">Outputs</Sm>
  <Sm StartAddress="#x1180" ControlByte="#x20" Enable="1">Inputs</Sm>
```

Abbildung 22: XML-Datei in Textform

XML-Dateien können zwar auch mit einem einfachen Editor geschrieben werden, aber mit diesem XML Editor ist es möglich, die Datei in einer Gitterstruktur (Abbildung 21, Seite 49) anzuzeigen, was die Lesbarkeit der Datei deutlich verbessert. Die Textform (Abbildung 22, Seite 50) ist flexibler, weil jeder Editor diese anzeigen kann.

# 5 Implementierung

## 5.1 Erstellung der Test-Hardware

Die Test-Hardware ist aus der Zusammenführung des PIC32MX Starter-Kit von Microchip und dem Piggybackboard FB1111-140 von Beckhoff erstellt worden. Zusätzlich wurde die Hardware für eine serielle RS232 Schnittstelle, welche mit dem HyperTerminal eines Computers kommunizieren kann, nach dem Schaltplan aus Abbildung 20 (Seite 47) erstellt. Da der im Schaltplan verwendete MAX232AESE Baustein nicht zur Verfügung gestanden hat, wurde der ICL232CPE von Harris verwendet. Die Bausteine sind Pin kompatibel und deswe-

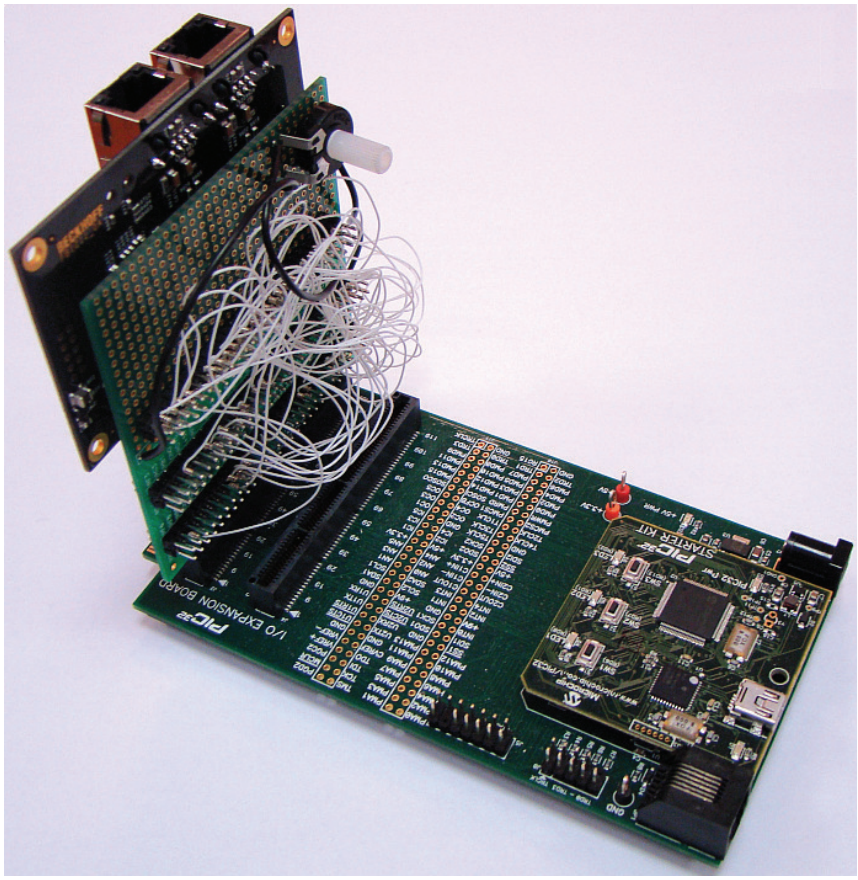


Abbildung 23: PIC32MX Starter Kit mit Piggybackboard Steckkarte

gen konnte die serielle Schnittstelle nach Plan aufgebaut werden. Um den ASIC des Piggybackboard an den Mikrokontroller anzuschließen, sind die Signale aus Abbildung 19 (Seite 46) mit den Pins des PIC32MX Starter Kit verbunden worden (Abbildung 23, Seite 51). Dies

wird mit Hilfe von Wrap-Drähten vorgenommen. Das Piggybackboard wird auf einer Steckkarte montiert, welche in das I/O Expansion Board des PIC32 Starter Kit gesteckt wird. Die untere RJ45 Buchse verbindet das Piggybackboard mit dem Test-PC. Über die andere Buchse können weitere Teilnehmer an das EtherCAT Netzwerk angebunden werden.
Der Stecker rechts unten (Abbildung 24, Seite 52) verbindet die Hardware mit dem In-Circuit Emulator MPLAB REAL ICE von Microchip. Mit Hilfe des Emulators wird der PIC32 pro-


Abbildung 24: PIC32 Starter Kit verbunden mit dem In-Circuit Emulator MPLAB REAL ICE

grammiert und debugged. Der Anschluss an den Computer erfolgt über ein USB-Kabel. Mit Hilfe eines hinzugefügten Potentiometers können die analogen Eingänge mit einer Spannung von 0V-3,3V getestet werden. Dieses Potentiometer wurde auf der Zusatzkarte mit dem Piggybackboard untergebracht und es befindet sich oberhalb der Wrap-Drähte. Es wird ein Potentiometer mit 100 k$\Omega$ verwendet.

## 5.2 Adaption des Slave Sample Code

Der Slave Sample Code (SSC) besitzt keine eigene main.c, da das Hauptprogramm über die Datei ecatappl.c aufgerufen worden ist. Um aber möglichst eine Trennung des SSC und des

Quellcodes für die PIC32 Peripherie zu erhalten, ist eine main.c erstellt worden, welche die Funktionen MainInit und MainLoop aufruft. In der neu erstellten Haupt-Datei wird die PIC32 Konfiguration durchgeführt. Besonders wichtig ist hierbei die Berechnung des Peripherietaktes aus dem Systemtakt über Teiler. Es wird ein Peripherietakt von 20MHz verwendet. Der Peripherietakt wird für die Timer, den A/D-Umsetzer und die serielle Schnittstelle benötigt. In der neuen Datei main.c werden auch die Interrupt Handler Routinen beschrieben. Diese rufen die jeweiligen Interrupt Service Routinen in anderen Quell-Dateien auf. In der Funktion main werden zuerst alle Initialisierungen durchgeführt und danach die Funktion MainLoop in einer while Schleife kontinuierlich aufgerufen.
Es sind zwei Timer konfiguriert worden, aber nur Timer1 wird bisher verwendet. Der Timer löst alle 200$\mu$s einen Interrupt aus. In der Interruptroutine werden mehrere Instruktionen abgearbeitet. Zu Beginn werden die Werte des A/D-Umsetzer in einem Feld gespeichert. Für eine Mittelwertbildung werden die letzten acht Messwerte des jeweiligen Eingangs aufaddiert. Um eine möglichst hohe Genauigkeit zu behalten, wird der aufaddierte Wert nicht durch die Anzahl geteilt, sondern als aufaddierter Wert übertragen (siehe Abbildung 25, Seite 53). Um bei einer möglichen Änderung des ADU nicht das SPS-Programm ändern

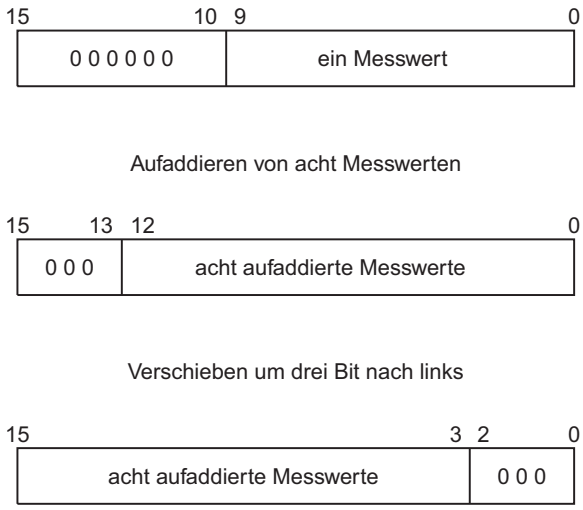



Abbildung 25: Verschiebung eines 10 Bit analogen Wertes auf 16 Bit

zu müssen, wird der Gesamtwert so verschoben, dass das höchstwertigste Bit aus der A/D-Umsetzung das Bit 15 der Variable ist. Nun könnte auch ein 12-Bit ADU verwendet werden ohne das SPS-Programm zu ändern. Auch kann die Mittelwertbildung über eine höhere Anzahl von Messwerten erstellt werde, ohne eine Änderung im SPS-Programm

vorzunehmen. Es muss lediglich die Anzahl der zu schiebenden Bits geändert werden. Es wird jeweils nur für ein analogen Eingang der Mittelwert gebildet. Dadurch wird die Dauer der Routine reduziert. In Abbildung 26 auf Seite 54 ist die Aufteilung über die Timer Interruptrou-

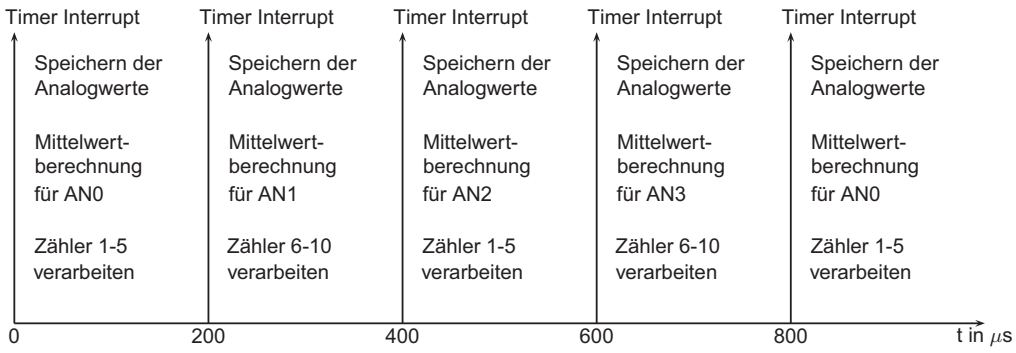



Abbildung 26: Darstellung der Timer Interrupts und die Verteilung der Prozesse

tinen zu sehen. Der ADU wird am Ende der Timer Routine gestartet und am Ende der ADU Interruptroutine wieder gestoppt. Der ADU fragt automatisch die vier analogen Eingänge nacheinander ab und löst dann den Interrupt aus. Das Verarbeiten der Zählereingänge wird auch in der Timer Routine aufgerufen. Um auch die Zeit dafür besser zu verteilen, werden immer nur fünf Zählereingänge abgearbeitet. Jeder Eingang wird dementsprechend alle 400$\mu$s geprüft, was einer Abfrage-Frequenz von 2,5kHz entspricht. Die Frequenz, mit welcher sich die Zählereingänge voraussichtlich ändern wird, wird mit höchstens 1kHz erwartet.

Die Kommunikation des Mikrokontrollers mit dem ASIC erfolgt über den parallelen Port. Hierfür wurden die benötigten Funktionen geschrieben, um auf Speicherbereich zuzugreifen, beziehungsweise um in diesen zu schreiben. Die folgende Funktion zeigt einen Lese-Zugriff auf einen zwei Byte großen Bereich.

```
unsigned short pmp_readW_ESC(unsigned short address)
{
  unsigned short value_r;
  while(PMMODEbits.BUSY);
  PMADDR = pmp_address_esc(address);
  while(PMMODEbits.BUSY);
  value_r=PMDIN;
  while(PMMODEbits.BUSY);
```

```
  value_r=PMDIN;
  while(PMMODEbits.BUSY);
  return value_r;
}
```

Als Funktionsparameter wird nur die untere Byte Adresse des ESC übergeben. Der Rückgabewert ist ein Wort mit den Daten. Es wird eine temporäre Variable erstellt, in welcher die Daten zwischengespeichert werden. In den while-Schleifen wird das Busy Bit des parallelen Ports abgefragt. Die Adresse wird über die Funktion pmp_address_esc berechnet. Dies ist notwendig, da die Adresse im ESC und die Adresse, welche der parallele Port verwendet, unterschiedlich sind. Das liegt daran, dass die Adress-Bits um ein Bit verschoben sind und weil das höchstwertigste Adress-Bit das Chip-Select Bit ist. Das bedeutet, dass eine Adresserhöhung des parallelen Ports zwei Adresserhöhungen im ESC zur Folge hat. Da der parallele Port aber immer zwei Byte anspricht, ist es möglich alle Bytes zu erreichen. Es werden stets zwei Lesezugriffe benötigt, da der erste Zugriff ein Dummy-Zugriff ist. Informationen werden vom letzten Lesebefehl gelesen und ein Neuer wird gestartet. Durch den zweiten Lesezugriff werden dann die benötigten Informationen in die temporäre Datei geschrieben und zurückgegeben. Die folgende Funktion zeigt einen Schreibzugriff auf ein einzelnes Byte.

```
char pmp_readB_ESC(unsigned short address)
{
  char valueByte;
  unsigned short value;

  value = pmp_readW_ESC(address);
  if(address & 0x0001)
  {
    value = value >> 8;
    valueByte = (value & 0xFF);
  }
  else
  {
    valueByte = (value & 0x00FF);
  }
  return valueByte;
```

```
}
```

Um ein einzelnes Byte auszulesen, muss trotzdem ein Lesezugriff auf das ganze Wort gemacht werden, da das kleinste Adress-Bit und das BHE nicht verwendet werden können. Mit Hilfe der Parität der Adresse wird daraufhin das obere oder untere Byte zurückgegeben.
Beim Schreibzugriff auf den DPRAM des ESC gibt es zwei Parameter, die übergeben werden müssen: die Adresse des ESC und die Daten, welche geschrieben werden sollen.

```
void pmp_writeW_ESC(unsigned short address, unsigned short
   value)
{
  while(PMMODEbits.BUSY);
  PMADDR = pmp_address_esc(address);
  while(PMMODEbits.BUSY);
  PMDIN = value;
}
```

Auch in diesem Fall muss die Adresse erst umgerechnet werden, damit der richtige Speicherbereich angesprochen wird. Die Funktion zur Adressumrechnung sieht wie folgt aus:

```
unsigned short pmp_address_esc(unsigned short esc_address)
{
  unsigned short pmp_address;
  esc_address = esc_address >> 1;
  pmp_address = 0x8000 + esc_address;
  return pmp_address;
}
```

Der Parameter esc_address ist die Byte Adresse im DPRAM des ESC. Um die Adresse für den parallelen Port zu bestimmen, muss die Adresse um ein Bit verschoben werden. Um das Chip-Select Bit richtig anzusprechen, muss zur Adresse noch 0x8000 addiert werden.
Innerhalb des SSC wird häufig auf einzelne Register des ESC zugegriffen. Dies geschieht über ein Zeiger System. Dieses Zeiger System wird nicht vom PIC32 unterstützt. Daher sind alle Zugriffe auf den ASIC geändert und durch die Funktionen des parallelen Ports ersetzt worden.

```
pEsc->Regs.AlEvent.Word[0]

pmp_readW_ESC(AL_EVENT_REQUEST_1)
```

Die obere Zeile zeigt den ursprünglichen Zugriff mit Hilfe einer Zeigerfunktion. Bei dieser wird auf eine vorkonfigurierte Struktur gezeigt, die weitere Strukturen enthält. In diesem Fall wird die Struktur Regs, welche alle Register des ESC enthält, aufgerufen und in dieser wird auf die Struktur AlEvent gezeigt. Das AlEvent Register beinhaltet zwei Datenworte, daher wird nur das Wort Null ausgewählt. In der unteren Zeile wird das gleiche Wort gelesen, nur wird hierbei die Funktion des parallelen Ports verwendet. Die Adresse wird über eine Definition festgelegt. Mit Hilfe der Endung _1 und _2 wird zwischen den beiden Worten unterschieden. Alle Adressdefinitionen sind in der Header Datei Define.h gespeichert.
Um die Prozessdatenobjekte zu übermitteln, ist ursprünglich auch mit Zeigern gearbeitet worden. Die Prozessdaten liegen in zwei Strukturen vor: Eine ist für die Eingangsdaten und eine für die Ausgangsdaten vorgesehen. Um die Daten leichter zu übertragen, wird über die Strukturen eine Zweite gelegt, welche eine Wort-Struktur besitzt.

```
UINT16  *pInput = (UINT16 *) &sPDOInput;
```

In dieser Zeile wird ein Zeiger mit Wort-Elementen erzeugt, welcher auf die erste Adresse der Eingangs-Prozessdatenobjekte zeigt. Mit Hilfe dieses Zeigers werden im weiteren Verlauf der Funktion die PDOs in einer Schleife wortweise über den parallelen Port übertragen. In der folgenden Funktion werden die Prozessdatenobjekte vom Mikrokontroller in den DPRAM des ASIC geschrieben.

```
void pmp_writemoreWaddr_ESC(unsigned short address, unsigned
    short * array, unsigned short length)
{
  int i;

  PMCONbits.ON   = 0;
  PMMODEbits.INCM  = 1;
  PMCONbits.ON   = 1;
  while(PMMODEbits.BUSY);
  PMADDR = pmp_address_esc(address);

  for(i=0;i<length;i++)
  {
    while(PMMODEbits.BUSY);
    PMDIN = array[i];
  }
```

```
}
```

Als Parameter werden die Startadresse der Prozessdatenobjekte, der Zeiger auf die PDOs im Mikrokontroller und die Länge in Worten übergeben. Da für die Übertragung die automatische Adresserhöhung des parallelen Ports verwendet wird, muss für die Zeit der Konfiguration das Modul kurzfristig deaktiviert werden. Während dieser Zeit wird die automatische Adresserhöhung aktiviert. Bevor die Startadresse mit der bekannten Adressfunktion eingestellt wird, wird noch geprüft, ob der parallele Port bereit ist. In der for-Schleife werden dann die Daten übertragen. Vor jedem Schreibzugriff wird der parallele Port auf Bereitschaft abgefragt. Anfangs sind die Daten ohne automatische Adresserhöhung übertragen worden. Durch weiterführende Überlegungen ist, um Zeit zu sparen, dieser Weg gewählt worden. Der in Kapitel 6.2 beschriebene Test geht auf diese Thematik im Detail ein.
Alle editierten und neu erstellten Quelldateien für den PIC32 sind im Anhang A3 zu finden.

## 5.3 Erstellung der ESI-Datei

Zur Erstellung der ESI-Datei ist die XML-Konfigurationsdatei für das FB1111-0140 Piggybackboard kopiert und die Einträge sind nach den Anforderungen der BIN-Karte geändert worden. Hierfür ist das Programm DiffDog von Altova verwendet worden. Die erstellte XML-Konfigurationsdatei für TwinCAT befindet sich im Anhang A4. Zu Beginn der Konfigurationsdatei werden die Herstellerinformationen eingetragen. Der Name und die von der EtherCAT Technology Group erhaltene Vendor ID sind in dem entsprechenden Bereich einzutragen. Abbildung 27 auf Seite 59 zeigt dies in der Gitterstruktur. Zusätzlich ist noch das Firmen

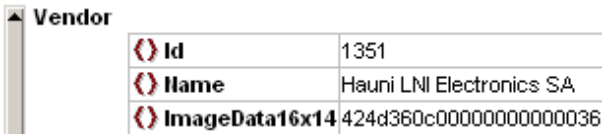



Abbildung 27: Herstellerangaben in der ESI-Datei

Logo mit eingebunden. Der Hexadezimal Wert wird nicht komplett angezeigt, da dieser sehr lang ist. Das Logo wird von TwinCAT auf 16x14 Pixel skaliert. Danach beginnt der “Description” Bereich, in dem die Gruppe von Bauelementen (Groups) und die Bauelemente (Devices) definiert werden. In Abbildung 28 auf Seite 59 sind die einzutragenden Informationen für die verwendete Gruppe zu sehen. Als Name für die Gruppe ist in dieser Datei “Hauni LNI De-

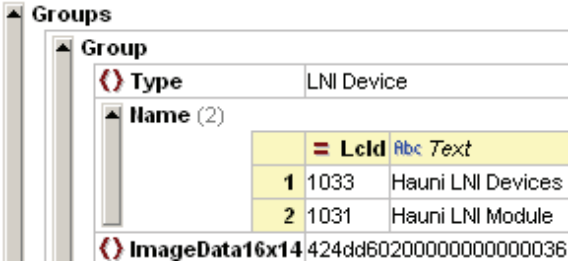



Abbildung 28: Gruppenerstellung für LNI Module in der ESI-Datei

vices” für die englische Version (LcID[15] = 1033) und “Hauni LNI Module” für die deutsche Version (LcID = 1031) genommen worden. Es ist auch möglich, einen Namen für weitere Sprachen zu integrieren. Je nachdem welche Sprache im System Manager ausgewählt ist, wird der entsprechende Name angezeigt. Zusätzlich ist für die Gruppe der Typ “LNI Device” definiert worden. Auf den Type beziehen sich später die einzelnen Baugruppen. Außerdem

[15] Locale Identifier: Eine standardisierte Sprach ID von Microsoft für XML-Dateien

ist auch ein Symbol für LNI Module erstellt und in hexadezimal Schreibweise abgelegt worden. Danach werden die Baugruppen definiert. Unter dem Eintrag “Devices” werden alle Baugruppen aufgelistet, welche durch diese Konfigurationsdatei hinzugefügt werden sollen. Abbildung 29 auf Seite 60 zeigt, wie der Anfang der Baugruppen Beschreibung der BIN-Karte aussieht. Im Eintrag “Physics” wird die physikalische Schicht der Baugruppe festgelegt. Der

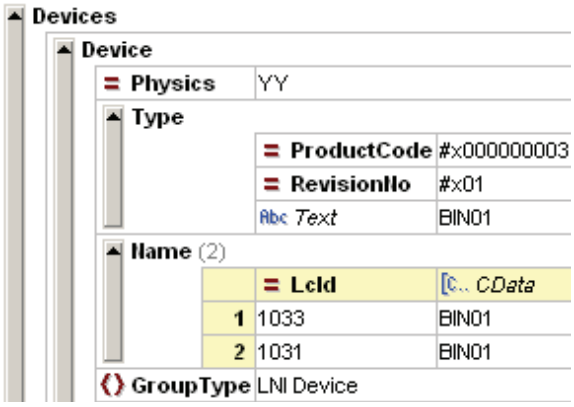



Abbildung 29: Beschreibung der Baugruppe in der ESI-Datei

Wert “Y” steht für einen EtherCAT Port und “K” für einen EBUS Port. Während der Implementierung ist die Test-Hardware direkt über das Piggybackboard an EtherCAT angebunden. Die BIN-Karte hingegen wird den Eintrag “KK” haben, da die Anbindungen der Ports über EBUS realisiert wird. Im Feld “Type” wird der Produktionscode und die Revisions Nummer eingetragen. Danach kann wieder mit Hilfe der LcID Nummer ein Name zugewiesen werden. Ebenfalls nötig ist die Zuordnung des Gruppen Types. Da nur eine Gruppe erstellt worden ist, wird diese auch verwendet.
Mit diesen Informationen ist es bereits möglich, nach dem Import der XML-Datei in den TwinCAT System Manager, die Baugruppe einzufügen (Abbildung 30, Seite 60). Die BIN-Karte

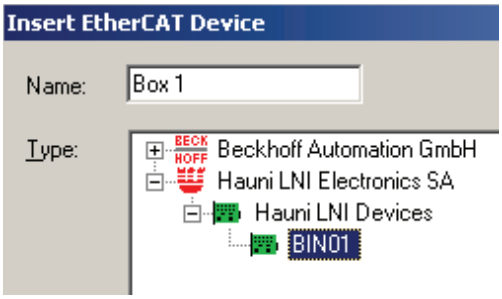



Abbildung 30: Einfügen einer EtherCAT Baugruppe

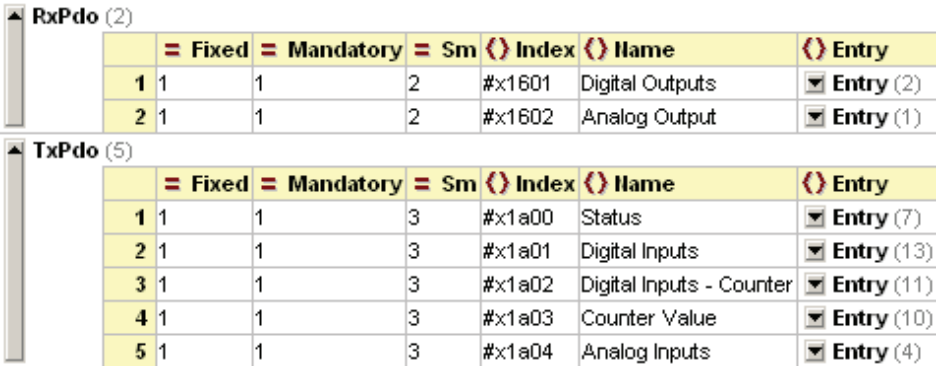
ist als Element der Hauni LNI Devices aufgelistet. Diese Gruppe ist, wie vorgesehen, unter dem Hersteller Hauni LNI Electronics SA eingetragen.
Danach sind die Informationen für die SyncManager eingepflegt worden. In Abbildung 31 auf Seite 61 ist der Ausschnitt für die SyncManager zu sehen. Es sind die Daten für vier

Sm (4)

| | DefaultSize | StartAddress | ControlByte | Enable | Text |
|---|---|---|---|---|---|
| 1 | 128 | #x1000 | #x26 | 1 | MBoxOut |
| 2 | 128 | #x1080 | #x22 | 1 | MBoxIn |
| 3 | | #x1100 | #x24 | 1 | Outputs |
| 4 | | #x1180 | #x20 | 1 | Inputs |

Abbildung 31: Beschreibung der SyncManager in der ESI-Datei

SyncManager gespeichert worden. Hinterlegt werden die Startadresse der SyncManager, das jeweilige Kontrol-Byte und die Information, ob der SM verwendet wird. Zur leichteren Zuordnung wird auch ein Name zugewiesen. Für die SyncManager der Mailbox wird zusätzlich die Standardgröße (DefaultSize) von 128 Byte eingetragen. Danach sind die Einträge für die Prozessdatenobjekte vorgenommen worden (Abbildung 32, Seite 61). Diese werden in



RxPdo (2)

| | Fixed | Mandatory | Sm | Index | Name | Entry |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | #x1601 | Digital Outputs | Entry (2) |
| 2 | 1 | 1 | 2 | #x1602 | Analog Output | Entry (1) |

TxPdo (5)

| | Fixed | Mandatory | Sm | Index | Name | Entry |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | #x1a00 | Status | Entry (7) |
| 2 | 1 | 1 | 3 | #x1a01 | Digital Inputs | Entry (13) |
| 3 | 1 | 1 | 3 | #x1a02 | Digital Inputs - Counter | Entry (11) |
| 4 | 1 | 1 | 3 | #x1a03 | Counter Value | Entry (10) |
| 5 | 1 | 1 | 3 | #x1a04 | Analog Inputs | Entry (4) |

Abbildung 32: Beschreibung der Prozessdatenobjekte in der ESI-Datei

RxPdo (Kommunikation von TwinCAT zum Mikrokontroller) und TxPdo (Kommunikation vom Mikrokontroller nach TwinCAT) unterteilt. Wichtig sind hier die Spalten Sm, Index, Name und Entry. In der Spalte Sm wird der SyncManager zugewiesen. Der SM2 ist für Ausgangsdaten und der SM3 für Eingangsdaten vorgesehen. Mit Hilfe der Spalte Index wird eine Ordnerstruktur erstellt. Die Namen der Ordner werden aus der Spalte Name entnommen. Die Zahl in Klammern hinter der Bezeichnung Entry steht für die Anzahl der Einträge, die unter dem Index gespeichert sind. Abbildung 33 auf Seite 62 zeigt die Ansicht im TwinCAT System Manager. Eingangsdaten werden mit einem gelben und Ausgangsdaten mit einem roten Symbol

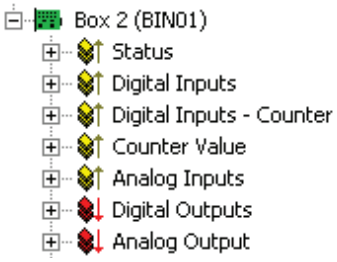



Abbildung 33: Ordnerstruktur im TwinCAT System Manager

dargestellt. Die Einträge (Entry) werden für Aus- und Eingangsdaten, sowie für jeden Index separat angezeigt. Abbildung 34 auf Seite 62 veranschaulicht die beiden Einträge “Digital Outputs” und “Analog Output”. Für jede zu übergebende Variable ist eine Zeile erstellt wor-

Entry

Entry (2)

| | Index | SubIndex | BitLen | Name | DataType |
|---|---|---|---|---|---|
| 1 | #x7010 | 1 | 1 | Reset Counters | BOOL |
| 2 | #x0 | 0 | 15 | | |

Entry (1)

| | Index | SubIndex | BitLen | Name | DataType |
|---|---|---|---|---|---|
| 1 | #x7020 | 1 | 16 | Analog Output | UINT |

Abbildung 34: Einträge der Indizes Digital Outputs und Analog Output in der ESI-Datei

den. Als digitaler Ausgang ist die Variable “Reset Counters” angelegt worden. Der Datentyp und die Bitlänge müssen übereinstimmen. Eine zusätzliche Zeile ist eingefügt worden, damit jeder Eintrag auf ganze Daten-Worte kommt. Dies ist wichtig, weil die Übertragung der Daten wortweise geschieht. Abbildung 35 auf Seite 62 zeigt die vorher in der ESI-Datei beschriebenen Prozessdatenobjekte im TwinCAT System Manager. Durch die Verwendung des Index

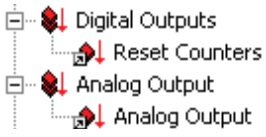



Abbildung 35: Darstellung der Prozessdatenobjekte im TwinCAT System Manager

Null für die Füll-Zeile zeigt TwinCAT diesen Eintrag nicht an. Die Auswirkung betrifft nur den Adressbereich der Prozessdatenobjekte.
Durch die Erstellung dieser ESI-Datei können Variablen des SPS-Programms über den System Manager mit Variablen des Mikrokontrollers verknüpft werden.

# 6 Leistungsbewertung

## 6.1 Durchlaufzeit des Mikrokontrollerprogramms

Es ist getestet worden, wie sich die Durchlaufzeit des Slave Sample Code verhält. Hierfür wurde ein digitaler Ausgang verwendet, welcher direkt vor der MainLoop Funktion invertiert wird. Um ein Bild davon zu bekommen, welche Unterschiede sich zwischen verschiedenen Einstellungen ergeben, wurde der digitale Ausgang mit einem Oszilloskop gemessen. Während der Messung wird der ADU in verschiedenen Modi verwendet. Auch die Mittelwertberechnung wird variiert und die Verarbeitung der Zählereingänge ist unterschiedlich durchgeführt worden. Damit sollen Unterschiede verdeutlicht und Annahmen aus dem Entwurf und der Implementierung bestätigt werden.
Die Abbildung 36 auf Seite 63 zeigt ein Oszillogramm, bei der der SSC wie in der Implementierung verwendet wird. Für die Messung ist das digitale Oszilloskop DL9040 von Yoko-

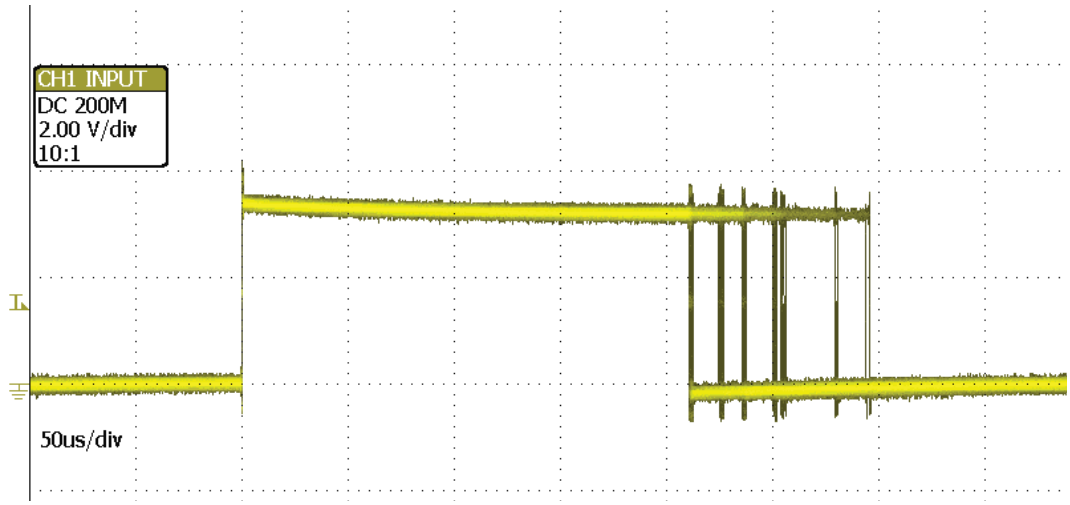



Abbildung 36: Oszillogramm der Durchlaufzeit des Slave Sample Code

gawa verwendet worden. Es ist eine Aufnahme mit 50 Messungen hintereinander gemacht worden. Für die Messung ist es notwendig, dass sich der ASIC im Zustand “Operational” befindet. Deswegen muss ein kleines Testprogramm von der Soft-SPS ausgeführt werden. Dieses Programm ist im Anhang A5. Es werden während eines Zyklus der Zustand der Hardware abgefragt und ein Zähler erhöht.
Zu sehen ist, wie die Durchlaufzeiten des SSC variieren. Das liegt daran, dass die Zahl der Interrupts, die aufgerufen werden, sich unterscheidet.
Tabelle 5 auf Seite 64 zeigt die verschiedenen Voraussetzungen, die für die Messung der

Durchlaufzeiten verwendet worden sind. Es sind vier unterschiedliche Bedingungen ausgewählt worden, bei denen erwartet wird, dass sie großen Einfluss auf die Durchlaufzeit haben. In der Tabelle 6 auf Seite 64 sind Messungen von verschiedenen Programmeinstellungen zusammengetragen worden. Als Informationen werden jeweils die minimale und maximale Durchlaufzeit angegeben, sowie die Differenz zur Testumgebung T1.

Tabelle 5: Testbedingungen bei der Messung von Durchlaufzeiten

| Testumgebung | Testbedingung |
|---|---|
| T1 | • ADU wird im ADU-Interrupt deaktiviert<br>• Mittelwertbildung der analogen Eingänge wird abwechselnd vorgenommen<br>• Verarbeitung der Zählereingänge wird aufgeteilt |
| T2 | • ADU wird nicht im ADU-Interrupt deaktiviert, sonst wie T1 |
| T3 | • Die Mittelwertbildungen werden mit jedem Timer-Interrupt durchgeführt, sonst wie T1 |
| T4 | • Die Verarbeitung der Zählereingänge wird in jedem Timer-Interrupt durchgeführt, sonst wie T1 |

Tabelle 6: Durchlaufzeiten des Hauptprogramms mit verschiedenen Eigenschaften

| Testumgebung | Minimale Zykluszeit | Maximale Zykluszeit | Differenz zu T1 |
|---|---|---|---|
| T1 | 210$\mu$s | 295$\mu$s | 0$\mu$s |
| T2 | 458$\mu$s | 590$\mu$s | 295$\mu$s |
| T3 | 279$\mu$s | 506$\mu$s | 211$\mu$s |
| T4 | 218$\mu$s | 312$\mu$s | 17$\mu$s |

Es ist deutlich der starke Einfluss des ADU zu erkennen, wenn dieser durchgängig verwendet wird. Die Durchlaufzeit des Programms ist doppelt so groß, wie in der Testumgebung T1. Auch die Mittelwertberechnung könnte einen sehr großen Teil der Zykluszeit in Anspruch nehmen, wenn diese nicht aufgeteilt werden würde. Durch diese Verteilung der Berechnungen wird der zeitliche Aufwand deutlich verringert. Die Zeiteinsparung durch die Aufteilung

der Verarbeitung der Zählereingänge ist dazu vergleichsweise gering. Hierbei muss aber beachtet werden, dass die Zählereingänge zeitkritischer sind als die anderen Vorgänge. Wenn die Abfrage-Frequenz zu klein ist, kann es passieren, dass Änderungen nicht wahrgenommen werden. Daher ist nur eine Halbierung der Zeit vorgenommen worden.

## 6.2 Übertragungsgeschwindigkeit des parallelen Ports

Bei diesem Versuch soll die Geschwindigkeit der Datenübertragung vom PIC32 Mikrokontroller zum ASIC und in die entgegengesetzte Richtung gemessen werden. Es werden vier Messungen durchgeführt: zwei Lese- und zwei Schreibzugriffe. Diese unterscheiden sich in der Einstellung des parallelen Ports. Bei einer Messung wird die Adresse innerhalb der Übertragungsfunktion hochgezählt und bei der anderen wird die Adresse durch die Peripherie automatisch erhöht. Damit soll die Zeitersparnis der Funktion aufgezeigt werden. Vor Beginn der Übertragung wird ein digitaler Ausgang auf Eins gesetzt und direkt nach Ende der Übertragung wieder auf Null. An dem entsprechenden Pin wird danach mit einem Oszilloskop die High-Zeit gemessen. Da bekannt ist, wie viele Daten übertragen werden, kann auf diese Weise eine Übertragungsrate berechnet werden. Für die Messungen werden je 100 Byte geschrieben oder gelesen, um gleiche Voraussetzungen zu haben. Es sind jeweils 200 Aufnahmen hintereinander gemacht und alle in einem Bild gespeichert worden. Während der Messungen werden alle Interrupts deaktiviert, um nur die reine Lese- oder Schreibdauer zu erhalten.
In Abbildung 37 auf Seite 65 ist das Oszillogramm zur Messung einer Schreiboperation mit automatischer Adresserhöhung zu sehen. Bei den Oszillogrammen der anderen Messungen

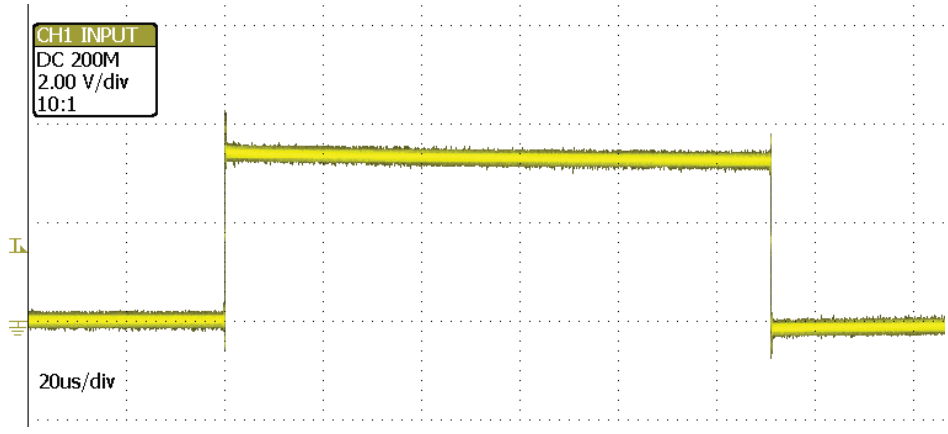



Abbildung 37: Oszillogramm der Übertragungsdauer vom PIC32 zum ASIC

ändert sich nur die Dauer der High-Zeit.
Zur Berechnung der Datenrate wird folgende Formel verwendet:

$$Datenrate = \frac{Datenmenge}{Transferdauer}$$

In der Tabelle 7 auf Seite 66 sind die Messergebnisse zusammengetragen. Die Spalte Lesen/Schreiben zeigt an, ob es sich um eine Lese- oder Schreiboperation handelt. Die Spalte Auto Inc. zeigt an, ob die Adresse automatisch durch die Peripherie erhöht worden ist oder ob dieser Vorgang durch die Software durchgeführt worden ist. Die Transferdauer ist in der vierten Spalte eingetragen. In der letzten Spalte ist die berechnete Datenrate.
Es ist der deutliche Unterschied zwischen der automatischen Adresserhöhung und der Soft-

Tabelle 7: Datenrate bei unterschiedlichen Zugriffsarten

| Lesen/Schreiben | Datenmenge | Auto Inc. | Transferdauer | Datenrate |
|---|---|---|---|---|
| Lesen | 100 Byte | Nein | 236$\mu$s | 423,7 kByte/s |
| Lesen | 100 Byte | Ja | 111$\mu$s | 900,9 kByte/s |
| Schreiben | 100 Byte | Nein | 161$\mu$s | 621.1 kByte/s |
| Schreiben | 100 Byte | Ja | 111$\mu$s | 900,9 kByte/s |

warelösung zu sehen. Bei dem Lesezugriff ist der Unterschied signifikanter, weil der obligatorische Dummy-Zugriff bei jeder Adressänderung durchgeführt werden muss und sich die Dauer dadurch deutlich verlängert. Bei automatischer Adresserhöhung ist die Datenrate vom Lese- und Schreibzugriff identisch. Die Datenrate könnte noch weiter erhöht werden, wenn das BUSY Signal des ESC direkt von der Peripherie des parallelen Ports verarbeitet werden könnte. In diesem Fall muss aber die Worst-Case Zeit berücksichtigt werden.

## 6.3 Zeitplan

In diesem Abschnitt soll der anfänglich angesetzte Zeitplan mit der tatsächlich benötigten Zeit verglichen werden. Dabei werden aufgetretene Abweichungen begründet. Tabelle 6.3 auf Seite 67 zeigt die während des Projekts benötigte Bearbeitungsdauer. In Klammern ist die entsprechend geplante Zeit für das Thema eingetragen. In der Einarbeitungsphase ist weniger Zeit verwendet worden, als zuvor geplant. Viele Eigenschaften und Besonderheiten haben sich erst während der Verwendung ergeben. Die Implementierung der $I^2C$ Kommunikation wurde nicht implementiert, da für die Test-Hardware die Bausteine nicht zur Verfügung gestanden haben. Für die Analyse und Adaption des Slave Sample Code ist deutlich mehr Zeit benötigt worden, als erwartet aufgrund des unterschätzten Umfangs an durchzuführenden Änderungen. In der Planungsphase ist erwartet worden, dass sich die Adaption

| Thema | Bearbeitungsdauer (Plan) |
|---|---|
| Einarbeitung EtherCAT Protokoll | 5 (7) Tage |
| Einarbeitung ASIC | 5 (7) Tage |
| Einarbeitung TwinCAT (System Manager) | 3 (5) Tage |
| Einarbeitung Mikrokontroller | 5 (5) Tage |
| Kommunikations Interface ASIC <> Mikrokontroller | 2 (2) Tage |
| Umsetzung des ASIC <> Mikrokontroller Interface | 2 (2) Tage |
| Implementierung der $I^2C$ Kommunikation | 0 (5) Tage |
| Analyse des Slave Sample Code | 15 (10) Tage |
| Adaption des SSC an den $\mu$C für zyklische Datenübertragung | 30 (15) Tage |
| Adaption des SSC an den $\mu$C für Mailboxkommunikation | 0 (10) Tage |
| Erstellung der ESI-Datei | 2 (2) Tage |
| Testphase | 20 (10) Tage |
| Schreiben der Diplomarbeit | 21 (20) Tage |
| Gesamtdauer | 110 (100) Tage |

auf einen den Mikrokontroller spezifischen Bereich beschränkt. Da dies nicht eingetreten ist, mussten im gesamten Bereich des SSC Änderungen vorgenommen werden. Daher hat die Zeit gefehlt, um die Mailboxkommunikation für den PIC32 zu implementieren. Auch die Testphase hat mehr Zeit in Anspruch genommen, als in der Planung vorgesehen. Dies ist damit zu erklären, dass auf Grund der ersten Testergebnisse Änderungen im Programm durchgeführt worden sind und die Tests dadurch wiederholt werden mussten. Zusammenfassend ist insgesamt mehr Zeit für die Entwicklung benötigt worden. Auf Grund des Umfangs der Adaption ist dies jedoch nachvollziehbar.

# 7 Zusammenfassung und Ausblick

Die EtherCAT Verbindung zwischen einem 32-Bit PIC Mikrokontroller und einer Soft-SPS ist nach den Vorgaben entwickelt worden. Da die E/A-Karte nicht zur Verfügung gestanden hat, ist die Software mit Hilfe einer Test-Hardware entwickelt und getestet worden. Alle essentiellen Eigenschaften der E/A-Karte konnten darüber implementiert werden. Ein Test mit der später verwendeten Hardware steht jedoch zu diesem Zeitpunkt noch aus. Die ESI-Datei ist nach den Vorgabe entwickelt worden.
Während der Durchführung des Projekts wurden Erkenntnisse über das Verhalten der Übertragung von Daten über EtherCAT gewonnen. Dadurch können bei zukünftigen Projekten einige Fehler vermieden werden, die bei dieser Entwicklung noch sehr viel Zeit in Anspruch genommen haben. Zusätzlich wurde ein verbessertes Verständnis von Abläufen eines Mikrokontrollers erreicht. Es sind viele Informationen zur Optimierung von C-Programmen erworben worden. Einige später verwendeten Verbesserungen sind nicht im Entwurf bedacht und erst während der Implementierung und im Laufe der Testphase entwickelt und verifiziert worden.
Wenige Merkmale, auf die in der Spezifikation eingegangen worden ist, sind noch nicht umgesetzt worden. Eine Eigenschaft, die noch nicht implementiert worden ist, ist die Mailboxkommunikation. Diese ist für die derzeitig geplante Verwendung zwar nicht notwendig, sie soll jedoch für zukünftige Erweiterungen zur Verfügung stehen, da über die Mailbox eine asynchrone Kommunikation realisiert werden kann. Die Entwicklung der Mailboxkommunikation wird im weiteren Verlauf des Projekts durchgeführt. Die Kommunikation über den $I^2C$-Bus konnte an der Test-Hardware nicht getestet werden, da die Bauteile nicht rechtzeitig vorgelegen haben. Diese werden nach der Inbetriebnahme der BIN-Karte implementiert werden. Auch weitere Möglichkeiten zur Optimierung der Durchlaufzeit sollen erforscht werden.

# Literatur

[1] Beckhoff Automation GmbH, *EtherCAT - EL9800 Basisplatine*, www.Beckhoff.de, 2008, Download am 17.02.2009.

[2] Beckhoff Automation GmbH, *EtherCAT Piggyback Controller Boards*, www.Beckhoff.de, 2009, Download am 17.02.2009.

[3] Beckhoff Automation GmbH, *Hardware Data Sheet ET1100 - EtherCAT Slave Controller*, www.Beckhoff.de, 2008, Download am 09.03.2009.

[4] EtherCAT Technology Group, *EtherCAT Application Layer Protocol Specification*, www.EtherCAT.org, 2008, Download am 02.03.2009.

[5] EtherCAT Technology Group, *EtherCAT Application Layer Service Definition*, www.EtherCAT.org, 2008, Download am 02.03.2009.

[6] EtherCAT Technology Group, *EtherCAT Communication - Communication Principles*, Beckhoff, www.EtherCAT.org, 2008, Download am 02.03.2009.

[7] EtherCAT Technology Group, *EtherCAT Data Link Layer Protocol Specification*, www.EtherCAT.org, 2008, Download am 02.03.2009.

[8] EtherCAT Technology Group, *EtherCAT Data Link Layer Service Definition*, www.EtherCAT.org, 2008, Download am 02.03.2009.

[9] EtherCAT Technology Group, *EtherCAT Introduction*, Beckhoff, www.EtherCAT.org, 2008, Download am 02.03.2009.

[10] EtherCAT Technology Group, *EtherCAT Physical Layer Service and Protocol Specification*, www.EtherCAT.org, 2008, Download am 02.03.2009.

[11] EtherCAT Technology Group, *EtherCAT Slave Device Description*, www.EtherCAT.org, 2009, Download am 11.03.2009.

[12] Microchip Technology Inc., *MPLAB C32 C Compiler User's Guide*, www.microchip.com, 2007, Download am 17.04.2009.

[13] Microchip Technology Inc., *PIC32 Peripheral Libraries for MPLAB C32 Compiler*, www.microchip.com, 2007, Download am 24.03.2009.

[14] Microchip Technology Inc., *PIC32MX3XX/4XX Family Data Sheet*, www.microchip.com, 2008, Download am 26.02.2009.

[15] Microchip Technology Inc., *32-Bit Language Tools Libraries*, www.microchip.com, 2009, Download am 17.04.2009.

[16] NXP Semiconductors, *$I^2C$-Bus Specification and User Manual*, www.nxp.com, 2007, Download am 26.02.2009.

# Anhang

| | |
|---|---|
| A1 | Schaltplan BIN Karte - gezeichnet von Pascal Liechti mit Unterstützung von Oliver Gräfe |
| A2 | Slave Sample Code (SSC) im Original v4.10 - Zur Verfügung gestellt durch die EtherCAT Technology Group |
| A3 | SSC nach der Adaption für den PIC32MX - editierte Dateien des SSC und Programmcode für die PIC spezifischen Quelldateien |
| A4 | Hauni LNI ESI-Datei für TwinCAT - erstellt von Oliver Gräfe |
| A5 | SPS Testprogramm - geschrieben von Beckhoff, modifiziert von Oliver Gräfe |

Die Anhänge sind in elektronischer Form auf einer CD abgelegt und beim Prüfer Prof. Dr. rer. nat. H. Hasemann einzusehen.

*Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

Hamburg, 24. August 2009 Oliver Gräfe