



**FACHHOCHSCHUL-MASTERSTUDIENGANG
AUTOMATISIERUNGSTECHNIK-INIF**

**Erstellung eines seriellen Mehrschichtenprotokolls auf
Basis des ISO OSI Referenzmodells**

Projektarbeit

von

Bernhard Muckenhumer

08/2011

Betreuung der Projektarbeit durch

Ing. Michael Zauner, BsC

KURZFASSUNG

Um eine anpassungsfähige und sichere Datenkommunikation zu ermöglichen, empfiehlt sich ein Mehrschichtenprotokoll. Jede Schicht bietet verschiedene Dienste für die darüber- oder darunterliegende Schicht an. Diese Arbeit beschäftigt sich mit der konzeptionellen Erstellung und Implementierung eines Mehrschichtenprotokolls in C. Als Basis dient das ISO OSI Referenzmodell¹ bzw. das TCP/IP Protokoll², das sich auf fünf anstatt sieben Schichten beschränkt. Als Anwendungsgebiet für das hier vorgestellte Protokoll ist die Kommunikation zwischen Mikrocontrollern vorgesehen. Es handelt sich nicht um ein Busprotokoll sondern um Point-to-Point Verbindungen zwischen mehreren Teilnehmern mit mehreren physikalischen Interfaces im Vollduplexbetrieb. Jeder Mikrocontroller (MCU) agiert als eigenständige Einheit und kann zu jedem Zeitpunkt einen Datentransfer einleiten.

¹ Vgl. Tannenbaum (2003), S54ff

Vgl. Bauch (2009)

² Vgl. Tannenbaum (2003), S58ff

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS	III
FORMELVERZEICHNIS	IV
1 MOTIVATION	1
2 ANFORDERUNGEN UND RESTRIKTIONEN	2
2.1 Hardware.....	2
2.2 Software.....	2
2.3 Protokoll	3
3 KONZEPT UND ARCHITEKTUR.....	3
3.1 Protokollschichten des OSI Modells	4
3.1.1 Auswahl geeigneter Protokollschichten	5
3.1.2 Definierte Aufgaben der Protokollschichten	5
3.2 Kommunikation	7
3.3 Frameaufbau	8
4 SOFTWAREENTWURF.....	10
4.1 Transmitter.....	10
4.2 Code Beispiel Transmitter	11
4.3 Receiver	12
4.4 Code Beispiel Receiver.....	13
4.5 Fehlererkennung	14
4.6 Fehlerszenarien	16
5 IMPLEMENTIERUNG	17
5.1 Protokollkonfiguration.....	17
5.2 Implementierte Files	17
6 PROTOKOLLANALYSE	18
6.1 Effizienz.....	18
6.2 Zeitverhalten	20
7 ZUSAMMENFASSUNG.....	22
8 LITERATUR.....	23
8.1 Bücher.....	23
8.2 Internetquellen	23
8.3 Verwendete Publikationen.....	23

ABBILDUNGSVERZEICHNIS

Abb. 1 : ISO OSI Referenzmodell	4
Abb. 2: Festgelegte Schichten.....	5
Abb. 3: Kommunikationsschema.....	7
Abb. 4: Aufbau einer Prozessdatennachricht.....	8
Abb. 5: Zustandsautomat des Senders	10
Abb. 6: Zustandsautomat des Empfängers.....	12
Abb. 7: Polynomdivision als CRC Grundlage	14
Abb. 8: Flusssteuerung und Senderverwaltung.....	15
Abb. 9: Nachricht wird vom Empfänger nicht erkannt.....	16
Abb. 10: Bestätigungsframe wird vom ursprünglichen Sender nicht erkannt	16
Abb. 11: Protokollkonfiguration	17
Abb. 12: Implementierte Files und deren Funktionen	18
Abb. 13: Protokolleffizienzverlauf	19
Abb. 14: Empfang und Bestätigung einer Nachricht	20
Abb. 15: Senden und Empfangen einer Nachricht.....	21
Abb. 16: Empfangen einer Nachricht bis der aktivierte Task diese ausgelesen hat.....	21
Abb. 17: Zeit vom Start des Sendens bis der Empfänger reagiert	21

FORMELVERZEICHNIS

(For. 6.1.1)	18
(For. 6.1.2)	19
(For. 6.2.1)	20
(For. 6.2.2)	20

1 MOTIVATION

Das Robo-Racing-Team der Fachhochschule Wels nimmt seit dem Jahr 2007 aktiv am Bewerb Eurobot³ teil. Es hat sich in den vergangenen Jahren herausgestellt, dass das bislang verwendete modulare elektronische System⁴ nur bedingt für einen ausfallssicheren Roboter geeignet ist. Daher wird für die Eurobot 2011 eine Hauptsteuerplatine erstellt, die insgesamt drei Mikrokontroller vom Typ ATXmega256A3 enthält. Diese führen ihre Aufgaben eigenständig durch und müssen in der Lage sein, auf eine sichere Art und Weise miteinander zu kommunizieren. Dafür soll ein serielles Mehrschichtenprotokoll zum Einsatz kommen, welches auf einfache Weise eingebunden und konfiguriert werden kann. Für den Nutzer werden Schnittstellen für das Senden und Empfangen von Nachrichten zur Verfügung gestellt. Das Mehrschichtenprotokoll an sich ist für den Anwender jedoch transparent.

³ Vgl. Eurobot-Regeln (2011)

⁴ Vgl. M. Zauner (2009)

2 ANFORDERUNGEN UND RESTRIKTIONEN

Nachfolgend werden die Anforderungen und Restriktionen für das serielle Protokoll erläutert, die im Vorfeld festgelegt wurden oder durch die Hardware gegeben sind. Primär soll das Protokoll auf Mikrocontrollern der Serie ATXmega von Atmel⁵ eingesetzt werden. Auf eine einfache Konfiguration und Adaptierung des Protokolls für andere Mikrocontrollertypen von Atmel wird jedoch großer Wert gelegt.

2.1 Hardware

Die unterste Schicht einer Datenkommunikation bildet in einem Mikrocontroller der USART (universal synchronous and asynchronous receiver and transmitter). Es gibt eine Vielzahl verschiedener Typen von Mikrocontrollern, die eine unterschiedliche Anzahl von USARTs enthalten. Im Falle des ATXmega256A3 stehen sieben Hardware-USARTs zur Verfügung, welche für den seriellen Protokollstack folgendermaßen konfiguriert werden: Ein Startbit; Acht Datenbits; Ein Stoppbit; Keine Parität (8, N, 1). Es ergeben sich somit zehn Bit für die Übertragung eines Bytes. Das Start- und Stoppbit werden zur Synchronisation zwischen Sender und Empfänger benötigt.

2.2 Software

Die Tatsache, dass die Ressourcen eines Mikrocontrollers begrenzt sind, insbesondere programmierbare Timer, erfordert Alternativen. Da bereits ein kooperatives Multitasking Betriebssystem⁶ aus Vorgängerprojekten existiert, können Tasks erstellt werden, die die Timeraufgaben des Protokolls übernehmen können. Dadurch werden einerseits Ressourcen eingespart aber andererseits wird das Betriebssystem stärker belastet. Es soll aus diesem Grund darauf geachtet werden, dass das serielle Protokoll auf einfache Weise angepasst werden kann, um es auch ohne Betriebssystem betreiben zu können (z.B.: mit Timer Interrupts).

⁵ <http://www2.atmel.com/>

⁶ Vgl. M. Zauner (2009)

2.3 Protokoll

Es soll ein für den Anwender transparentes Mehrschichtenprotokoll entstehen, das jeweils einen Dienst für das Senden und das Empfangen von Daten auf der obersten Protokollschicht bereitstellt. Das Protokoll soll für den Vollduplexbetrieb ausgelegt werden. Durch ein Bestätigungsframe des Empfängers soll dem Sender der Erhalt einer Nachricht bestätigt werden. Bei Nichterhalten des Bestätigungsframes soll die Nachricht erneut aber maximal n mal gesendet werden. Um beschädigte Frames zu erkennen, soll eine geeignete Fehlererkennung implementiert werden. Es handelt sich hierbei nicht um ein Bussystem sondern um Point-to-Point Verbindungen. Da ein kooperatives Multitaskingbetriebssystem angewendet wird, soll eine geeignete Adressierung der Tasks, für die eine empfangene Nachricht bestimmt ist, erstellt werden. Die Einschränkung eines Request/Response Netzwerkes soll unbedingt vermieden werden.

3 KONZEPT UND ARCHITEKTUR

Ausgangsbasis für die Konzepterstellung des seriellen Mehrschichtenprotokolls ist das OSI Referenzmodell und insbesondere das TCP/IP Protokoll, da es anstatt sieben Schichten nur fünf benötigt. Nach *Tannenbaum* hat es sich herausgestellt, dass die aus dem OSI Referenzmodell stammenden Schichten „Presentation“ und „Session“ in den meisten Fällen keine Anwendung finden. Das TCP/IP Protokoll unterscheidet jedoch nicht so klar zwischen Dienst, Schnittstelle und Protokoll wie es das OSI Referenzmodell macht⁷. Im folgenden Kapitel werden die geeignete Auswahl der Schichten und die Struktur des zu erstellenden seriellen Protokolls dargelegt.

⁷ Vgl. Tannenbaum (2003), S61ff

3.1 Protokollschichten des OSI Modells

Das OSI⁸ (open system interconnections reference model) wurde 1983 von der ISO (Internationale Organisation für Normung) standardisiert. Es wurde als Entwurfsgrundlage für Datenkommunikation in Rechnernetzen entwickelt und besteht, wie in Abb. 1 ersichtlich, aus sieben Schichten (eng. Layer), wobei jede Schicht definierte Dienste und Schnittstellen zur Verfügung stellt. Die gleichwertigen Schichten auf Sender- und Empfängerseite kommunizieren über ein Protokoll. Mit jeder Schicht nimmt der Abstraktionsgrad bezüglich der Kommunikation zu.

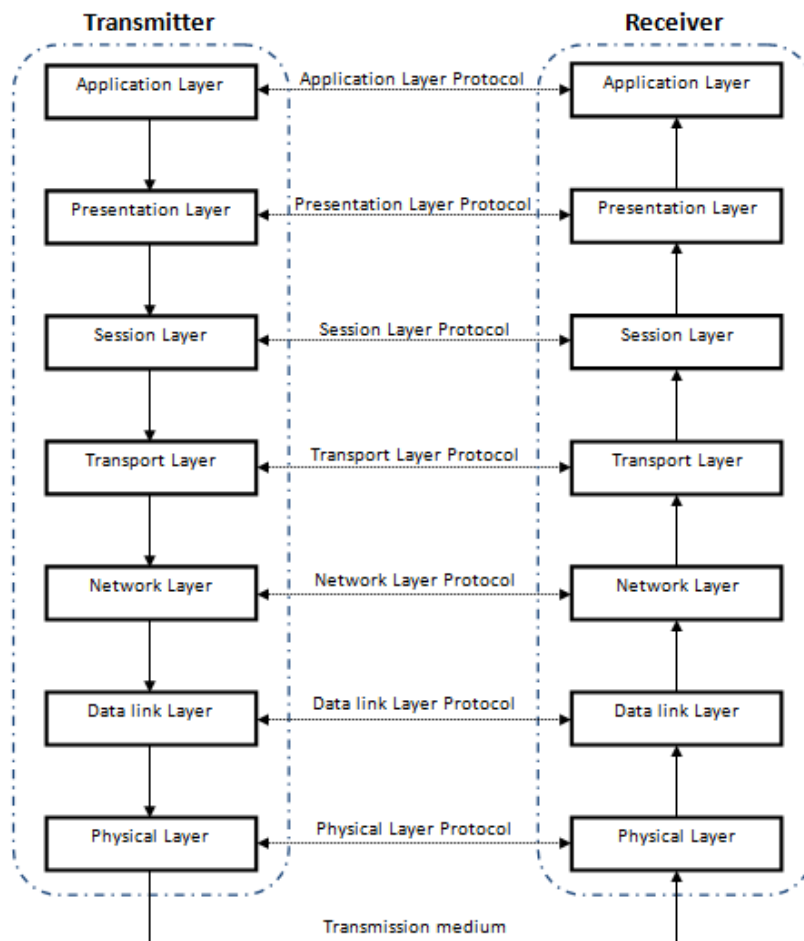


Abb. 1 : ISO OSI Referenzmodell

⁸ Vgl. [ISO/IEC standard 7498-1](#)

3.1.1 Auswahl geeigneter Protokollschichten

Für die Erstellung des seriellen Protokolls wurden vier Schichten ausgewählt. In Abb. 2 ist der Aufbau ersichtlich. Wie beim TCP/IP Protokoll wird auf den Presentation Layer und den Session Layer verzichtet. Da es sich bei dem zu entwerfenden Protokoll um ein Point-to-Point Netzwerk handelt und die Teilnehmer physikalisch adressiert werden, entfällt auch der Network Layer.

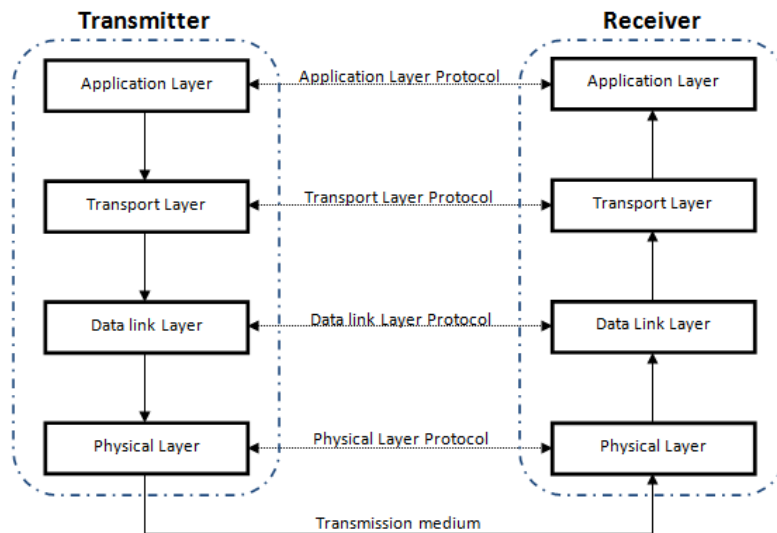


Abb. 2: Festgelegte Schichten

3.1.2 Definierte Aufgaben der Protokollschichten

- *Physical Layer:*

In dieser Schicht werden Bits über ein Medium übertragen. Die Unterscheidung zwischen den logischen Signalen Low und High erfolgt über den Spannungspegel. Die Spannungssignale der Hardware-USARTs eines Mikrokontrollers können natürlich durch eine geeignete Folgehardware in einen anderen Standard, wie zum Beispiel RS422 konvertiert werden, wodurch mittels Differenzsignalauswertung die Störsicherheit erhöht werden kann. Ein Standard wie RS485 kann nicht verwendet werden, da die Vollduplexfähigkeit nicht gegeben wäre.

- *Data link Layer:*

Im Data link Layer erfolgt der Frameaufbau beim Senden, die Prüfung der empfangenen Frames auf Gültigkeit und Fehler, sowie die Bestätigung der Empfangsdaten. Durch eine Timeoutüberwachung des Bestätigungsframes können die Daten erneut gesendet werden, wenn diese fehlerhaft oder gar nicht empfangen wurden.

- *Transport Layer:*

Im Transport Layer werden die empfangenden Daten dem Zieltask zugestellt, indem dieser vom Transport Layer aktiviert bzw. informiert wird. Der Zieltask kann die Daten dann mit einer minimalen Verzögerung, resultierend aus der Abarbeitungszeit der restlichen Systemtasks, auslesen und Verarbeiten. Somit kann die Datenverarbeitung beim Empfänger sowohl eventbasiert als auch zyklisch erfolgen.

- *Application Layer:*

Der Application Layer stellt die Serviceprimitiven für das Senden und Empfangen von Nachrichten zur Verfügung. Dabei soll der Protokollstack für den Anwender transparent sein. Der Anwender gibt beim Senden an, welche Daten er über welche Schnittstelle an welchen Zielport senden will. Bei Empfang einer Nachricht durch den Protokollstack wird jener Task, der dem Zielport zugewiesen ist, aktiviert. Dieses Prinzip ähnelt dem der Sockets⁹ bei TCP/IP Verbindungen. Der Socket repräsentiert einen Verbindungskanal mit zugehöriger Portnummer und muss beim TCP/IP Protokoll instanziiert werden. Mit dem Bind Befehl wird dann eine IP Adresse zugewiesen. Hier liegt der grundlegende Unterschied zum hier vorgestellten seriellen Protokoll. Anders als bei TCP/IP muss nicht erst eine Verbindung angefordert und eine IP Adresse zugewiesen werden. Es handelt sich ausschließlich um physikalische Point-to-Point Verbindungen. Dadurch kann es auch zu keinen Kollisionen bei der Datenübertragung kommen und der Vollduplexbetrieb kann problemlos angewendet werden.

⁹ Vgl. Tannenbaum (2003), S533ff

3.2 Kommunikation

Die Zustellung einer gültig empfangenen Nachricht erfolgt über eine Destinationportnummer (DP 1 bis n), die fest im Frame eingebettet ist. Über die Destinationportnummer kann der Task für den die Daten bestimmt sind aktiviert werden. In Abb. 3 ist das Kommunikationsschema dargestellt.

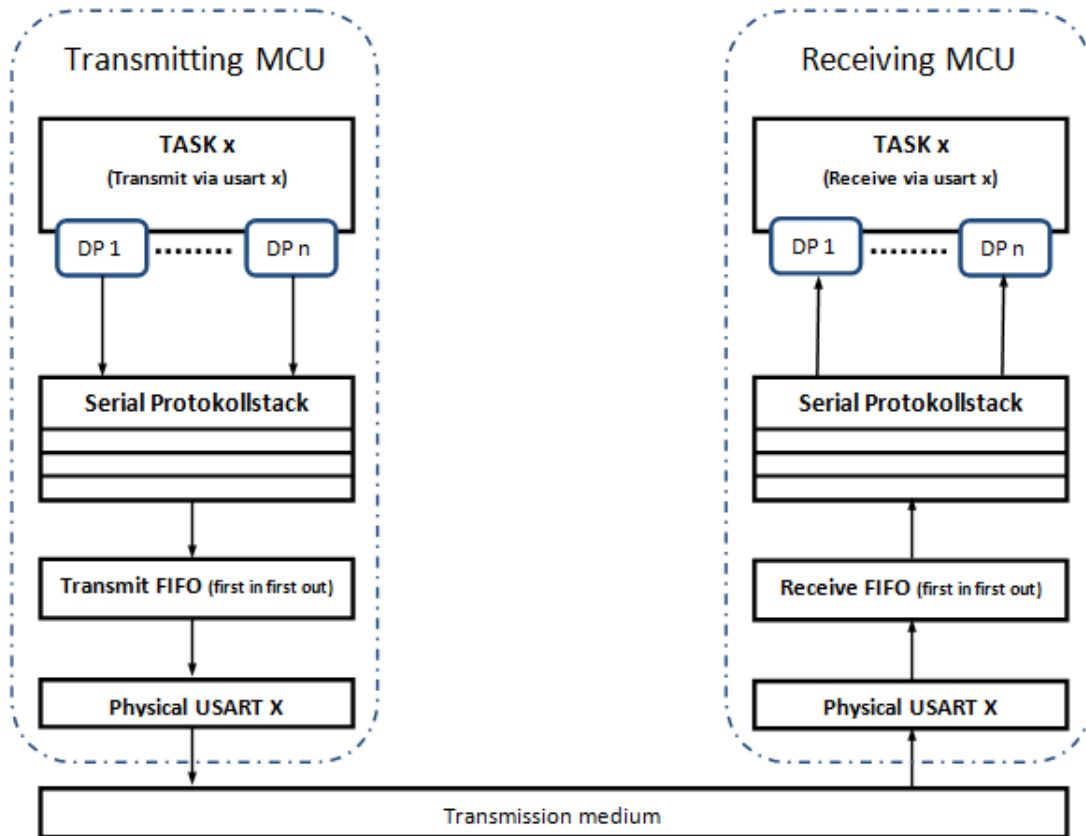


Abb. 3: Kommunikationsschema

3.3 Frameaufbau

Die Nutzdaten, die gesendet werden sollen, durchlaufen den Protokollstack und erhalten in jeder Schicht zusätzliche Headerdaten, die schließlich im Data link Layer den vollständigen Frame ergeben. In Abb. 4 ist der Aufbau je Protokollschicht ersichtlich.

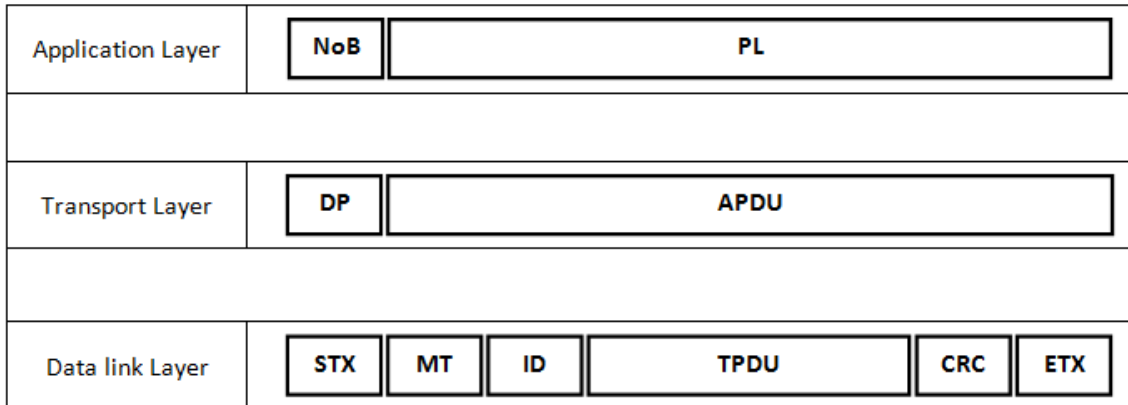


Abb. 4: Aufbau einer Prozessdatennachricht

NoB **Number of Bytes:**

Die Anzahl der Nutzdatenbytes (Payload), die in der Nachricht enthalten sind.

Wertebereich: 0x00 – 0xFF (1 Byte)

DP **Destination port:**

Durch die Zielporennummer wird die Nachricht für einen Task des Empfängers adressiert.

Wertebereich: 0x00 – 0xFF (1 Byte)

STX..... **Start of transmission:**

Startsequenz eines Frames.

Der numerische Wert ist festgelegt: 27; 28 (2 Byte) → *ESC*; *ESC* + 1

ETX **End of transmission:**

Endsequenz eines Frames.

Der numerische Wert ist festgelegt: 27; 29 (2 Byte) → *ESC*. *ESC* + 2

Um zu vermeiden, dass das Zeichen *ESC* in den Nutzdaten als *End of transmission* fehlinterpretiert wird, werden sogenannte Esquapesequenzen eingeführt. Die Folge *ESC*, *ESC*+1 entspricht dabei dem *STX* und die Folge *ESC*, *ESC*+2 entspricht *ETX*. Diese Methode wird in der Literatur als Byte-Stuffing¹⁰ oder Character-Stuffing bezeichnet.

¹⁰ Vgl. Tannenbaum (2003), S115ff

MT

Message type:

Bits 3 – 7:

Enthält die Art der Nachricht. Für das Protokoll werden zwei Typen unterschieden:

TYPE_DATA: Es handelt sich um eine Prozessdatennachricht.

TYPE_ACK: Es handelt sich um eine Bestätigungsnachricht (keine Nutzdaten).

Bits 0 – 2:

Enthält die Anzahl der Versuche des Senders, eine Prozessdatennachricht zu versenden. Damit kann die zugehörige Bestätigungsnachricht identifiziert werden. Maximal können damit sieben Sendeversuche abgesetzt werden.

ID **Message identification:**

Der Sender vergibt für jede Prozessdatennachricht eine Identifikationsnummer, die auch in die Bestätigungsnachricht integriert wird. Somit kann der Sender die eingehenden Bestätigungsnachrichten eindeutig zuweisen. Die IDs jener Nachrichten, die bereits bestätigt wurden, werden für nachfolgende Nachrichten wieder freigegeben.

Wertebereich: 0x00 – 0xFF (1 Byte)

CRC..... **Checksum:**

Der Rest, der sich bei der Polynomdivision durch ein bestimmtes Polynom (16 Bit) ergibt, wird an die Nachricht angehängt. Es handelt sich dabei um das CRC(cyclic redundancy check)¹¹ Verfahren. Der CRC Rest belegt 2 Byte der Nachricht (Beispiel: [siehe 4.3 Fehlererkennung](#)).

PL, APDU, TPDU..... **Process Data Unit**

Dies sind die Nutzdaten jeder Schicht, die jeweils in die Nachricht eingebettet und mit Headern versehen werden.

Es ergibt sich damit ein Gesamtoverhead von 10 Bytes pro Nachricht. Um die Effizienz zu erhöhen ist es deshalb vorteilhaft, die Anzahl der Nutzdaten deutlich über diesem Wert anzusetzen.

¹¹ Vgl. Tannenbaum (2003), S224ff

4 SOFTWAREENTWURF

In diesem Kapitel wird die Vorgehensweise beim Softwareentwurf des Protokollstacks vorgestellt. Der Sender sowie der Empfänger sind als Zustandsautomaten (eng. State machine) ausgeführt.

4.1 Transmitter

Der Sender muss nach dem definierten Protokoll den Frame erstellen und diesen dann in den Sendebuffer schreiben. Das Senden erfolgt anschließend mittels Interrupts. In Abb. 5 ist die sequenzielle Abfolge der Sendeframeerstellung ersichtlich.

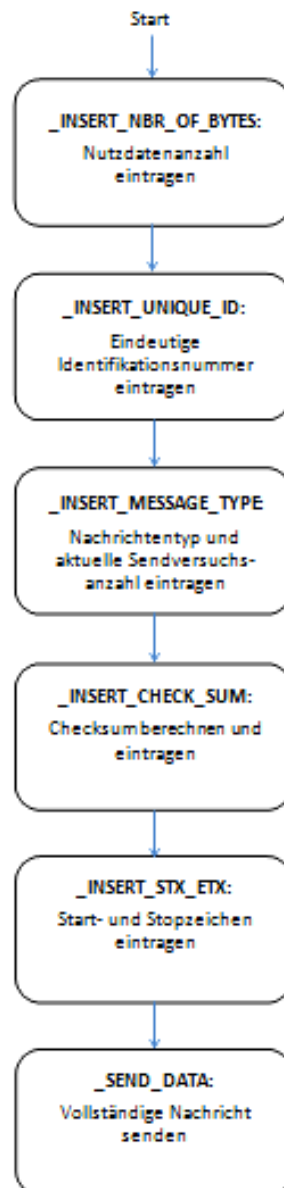


Abb. 5: Zustandsautomat des Senders

4.2 Code Beispiel Transmitter

Nachfolgend ist ein Beispiel angeführt, wie der Aufruf des Dienstes für das Senden aus dem *ApplicationLayer* aussehen kann:

```
uint8_t i;  
uint8_t TransmitArray[10];  
  
for(i = 0; i < 10; i++)  
{  
    TransmitArray[i] = 1;  
}  
  
Send_Application_Data(&USART_data_D0, TASK_1_PORT, &TransmitArray[0], 10);
```

USART_data_D0: Hardware USART des Mikrokontrollers über den die Nachricht gesendet wird.

TASK_1_PORT: Die Zielportnummer des Empfängers, für die die Nachricht bestimmt ist.

TransmitArray[]: Lokaler Sendebuffer für die Sendedaten.

4.3 Receiver

Abb. 6 zeigt den Aufbau des Empfängers und dessen Arbeitsweise. Für den Vollduplexbetrieb muss dieser Zustandsautomat zyklisch für jede Schnittstelle aufgerufen werden. Jeder Zustand erfüllt eine Teilauswertung des empfangenen Frames und veranlasst bei einem Fehler eine Rückkehr in den Startzustand (`_FIND_STX_I`). Der fehlerhafte Frame wird somit verworfen.

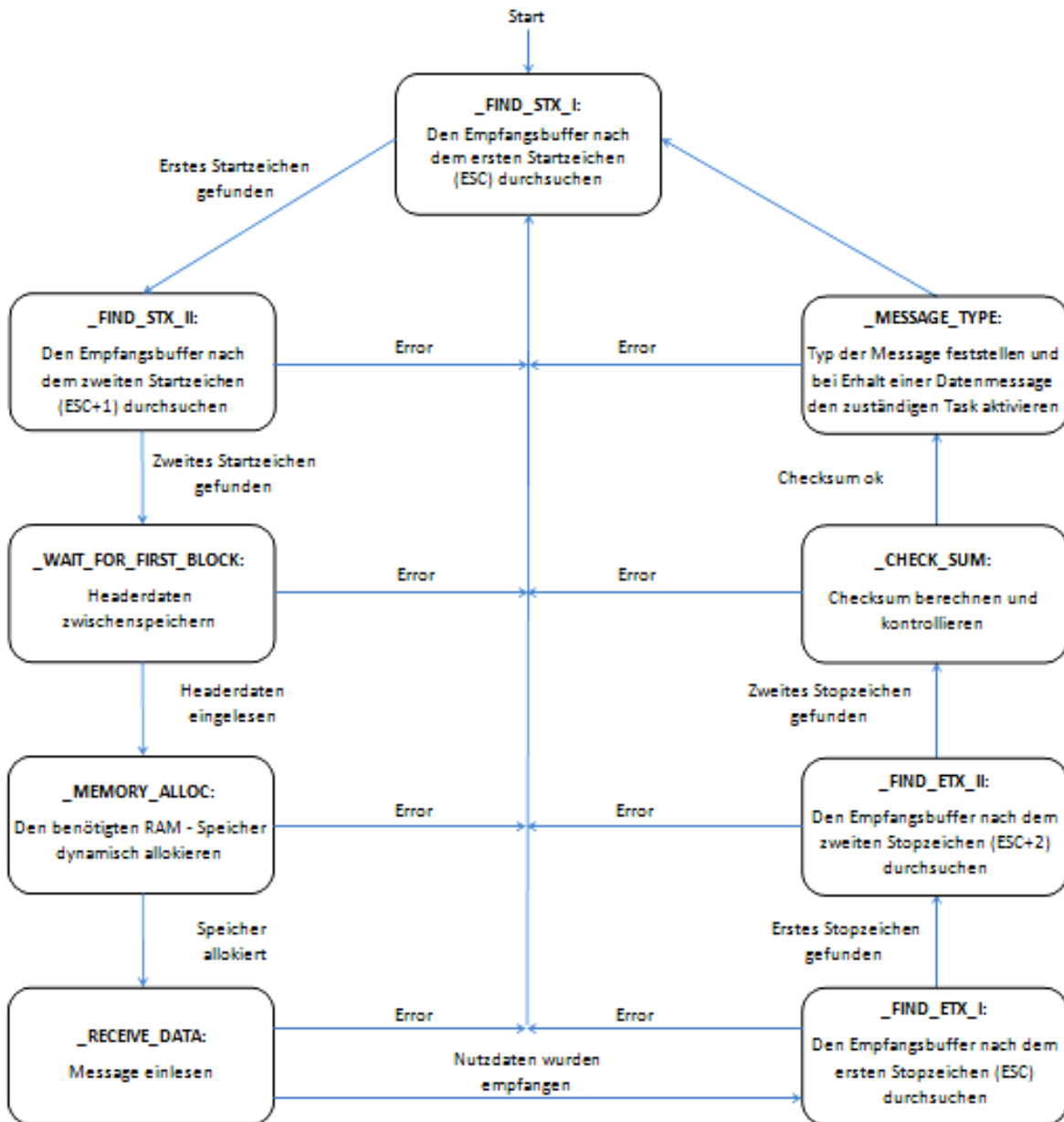


Abb. 6: Zustandsautomat des Empfängers

4.4 Code Beispiel Receiver

Nachfolgend ist ein Beispiel angeführt, wie der Aufruf des Dienstes für das Empfangen aus dem *ApplicationLayer* aussehen kann:

```
uint8_t nbr_of_bytes = 0;
uint8_t i;
uint8_t receiveArray[10];

if(Received_AppData_Available(&USART_data_D0, TASK_1_PORT, &nbr_of_bytes))
{
    Receive_Application_Data(&USART_data_D0 , TASK_1_PORT, receiveArray);

    for(i = 0; i < nbr_of_bytes; i++)
    {
        printf("%d ",receiveArray[i]);
    }
}
```

USART_data_D0: Hardware USART des Mikrokontrollers von der die Nachricht empfangen wurde.

TASK_1_PORT: Die Zielporntnummer des Empfängers, für die die Nachricht bestimmt ist.

nbr_of_bytes: Anzahl der empfangenen Nutzdatenbytes.

receiveArray[]: Lokaler Empfangsbuffer für die empfangenen Daten.

4.5 Fehlererkennung

Für die Überprüfung der empfangenen Daten wird der Fehlererkennungsalgorithmus CRC (cyclic redundancy check) mit einem 16 Bit Generatorpolynom eingesetzt. Für eine bessere Übersichtlichkeit wird in folgendem Beispiel (Abb. 7) ein 5 Bit Generatorpolynom verwendet, um den Rest (Remainder) zu berechnen.

Beispiel:

Frame (zu versendende Daten als Bitstrom): $1101011011 = x^9 + x^8 + x^6 + x^4 + x^3 + x^1 + 1$

Generator (Generatorpolynom): $10011 = x^4 + x + 1$

Die Ordnung des Rests ist immer um einen Grad niedriger als jene des Generatorpolynoms. In diesem Fall ist der Rest ein Polynom 3. Ordnung. Aus diesem Grund wird der Nutzdatenbitstrom um vier Nullen erweitert. Nach der Polynomdivision (logisch XOR) ergibt sich der Rest von $1110 = x^3 + x^2 + x$. Dieser Rest wird an Stelle der erweiterten Nullen an den Nutzdatenbitstrom angehängt und übertragen (transmitted frame). Auf Empfängerseite wird auf die gleiche Weise eine Polynomdivision mit dem empfangenen Bitstrom durchgeführt, wobei sich offensichtlich kein Rest ergeben darf. Entspricht der Rest einem Wert $\neq 0$, so muss es sich um einen Übertragungsfehler handeln.

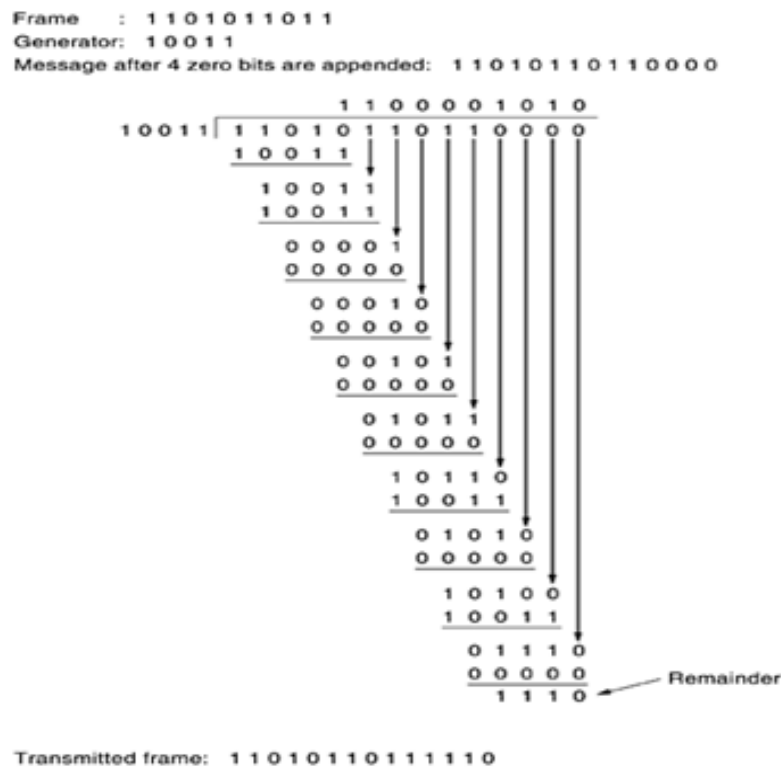


Abb. 7: Polynomdivision als CRC Grundlage

Bei Erkennung eines fehlerhaften Frames wird kein Bestätigungsframe an den Sender zurückgeschickt, was zu einem erneuten Übertragen des ursprünglichen Frames des Senders führt. Es muss ein Mechanismus entwickelt werden, der die bereits versendeten Datenframes zwischenspeichert und erst nach Erhalt des Bestätigungsframes wieder freigibt, da es offensichtlich mehrere Nachrichten gibt, die innerhalb eines Timeoutzyklus versendet werden können. TCP/IP verwendet dafür sogenannte sliding windows¹² in der zweiten Protokollschicht, die auch für die Flusststeuerung zuständig sind. Für das hier vorgestellte serielle Protokoll wird ein ähnlicher, wenngleich einfacherer Mechanismus bereitgestellt. In Abb. 8 ist das Prinzip dargestellt.

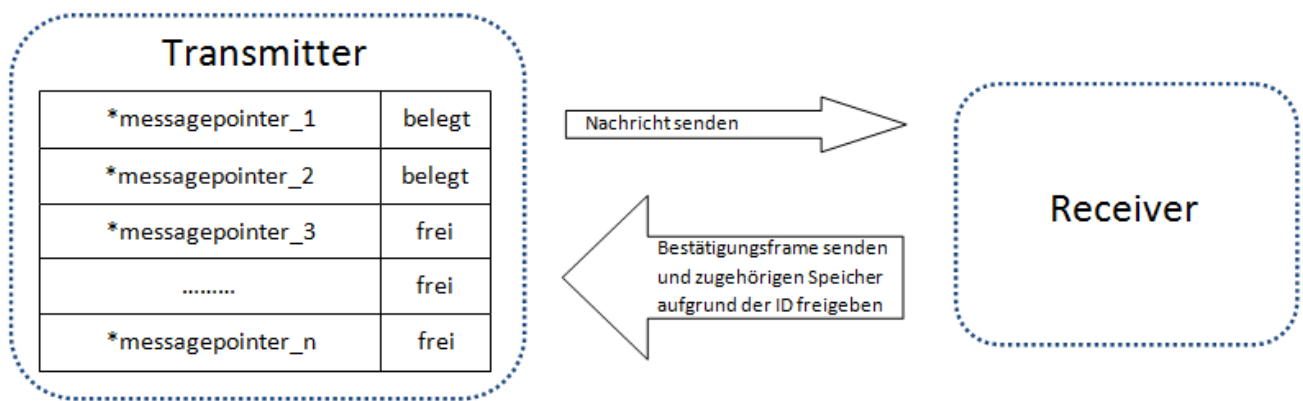


Abb. 8: Flusssteuerung und Senderverwaltung

Die **messagepointer_X* bezeichnen die Startadressen der zwischengespeicherten Messages. Wird ein korrekter Bestätigungsframe vom Empfänger gesendet, so wird der als *belegt* markierte Index wieder freigegeben. Die Zugehörigkeit der Pointer zu den Bestätigungsframes wird durch IDs ermittelt. Ein im Hintergrund laufender Task kontrolliert die Zeit, die vom Zeitpunkt des Sendens bis zum Erhalt des Bestätigungsframes verstreicht. Wird eine festgelegte Zeit überschritten, so wird der Frame erneut versendet. Dies wird n mal wiederholt, bis eine Fehlermeldung an den Benutzer des Protokolls ausgegeben wird. Bei jeder Wiederholung muss die ID angepasst werden, um zu vermeiden, dass ein Artefakt, das kurz nach Ablauf der Timeout eingelangt, als aktueller Bestätigungsframe erkannt wird.

¹² Vgl. Tannenbaum (2003), S239ff

4.6 Fehlerszenarien

1. Die Nachricht wird während des Sendens derart beschädigt, dass sie vom Empfänger nicht als solche erkannt und somit verworfen wird oder die Checksummenprüfung schlägt fehl. (Abb. 9):

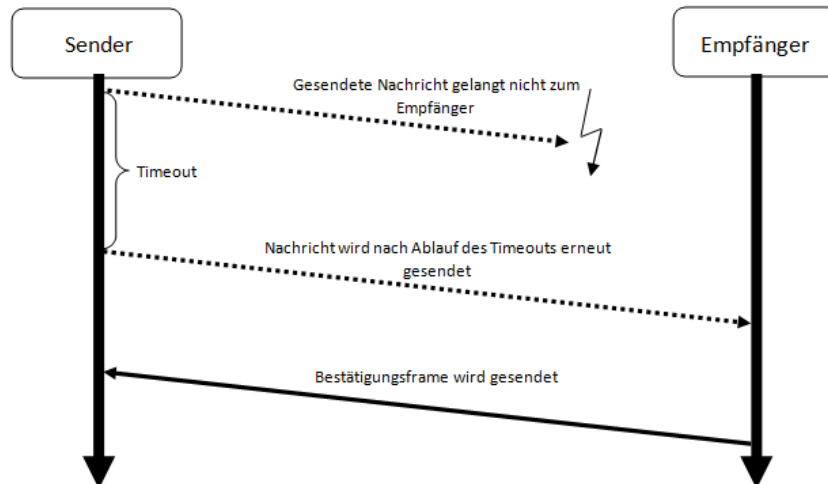


Abb. 9: Nachricht wird vom Empfänger nicht erkannt

2. Die Nachricht wird korrekt empfangen und verarbeitet, aber das Bestätigungsframe wird derart beschädigt, dass es beim ursprünglichen Sender nicht mehr als solches erkannt und somit verworfen wird oder die Checksummenprüfung schlägt fehl (Abb. 10):

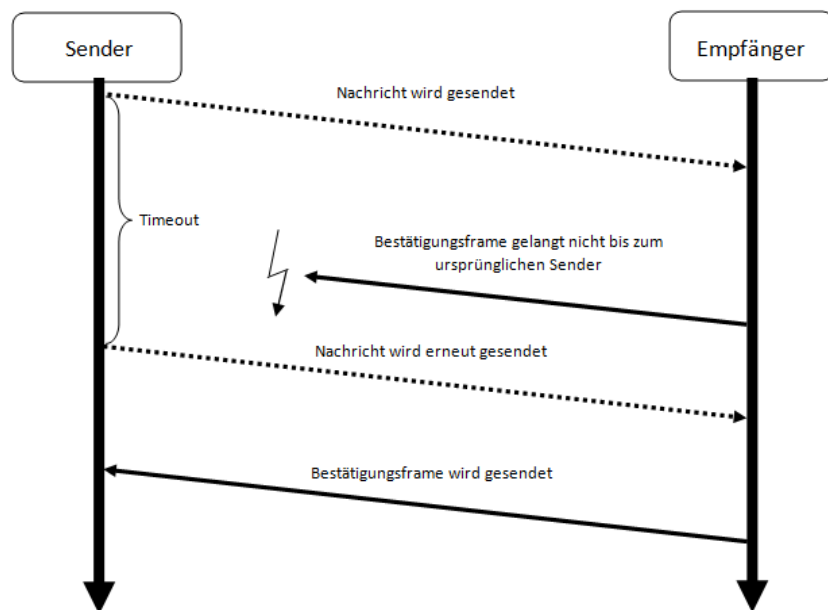


Abb. 10: Bestätigungsframe wird vom ursprünglichen Sender nicht erkannt

5 IMPLEMENTIERUNG

5.1 Protokollkonfiguration

Für die Steigerung der Wiederverwendbarkeit wird eine Konfigurationsdatei bereitgestellt, mit dessen Hilfe der Anwender des Protokolls festlegen kann, welche USARTS verwendet werden und mit welcher Baudrate diese betrieben werden sollen. In Abb. 11 ist ein Beispiel zum besseren Verständnis angegeben.

```
//#define _SERIAL_INTERFACE_C0
//#define BAUDRATEC0 9600
//#define _SERIAL_INTERFACE_C1
//#define BAUDRATEC1 9600
#define _SERIAL_INTERFACE_D0
#define BAUDRATED0 230400
#define _SERIAL_INTERFACE_D1
#define BAUDRATED1 230400
//#define _SERIAL_INTERFACE_E0
//#define BAUDRATEE0 9600
//#define _SERIAL_INTERFACE_E1
//#define BAUDRATEE1 9600
//#define _SERIAL_INTERFACE_F0
//#define BAUDRATEF0 9600
```

Abb. 11: Protokollkonfiguration

Die USARTS des ATXmega256A3¹³ werden mit (C0, C1, D0, D1, E0, E1, F0) bezeichnet und können wie in Abb. 11 ersichtlich einfach in den Code aufgenommen werden.

5.2 Implementierte Files

Bei der Implementierung liegt das Hauptaugenmerk auf der Trennung der Protokollschichten und Teilaufgaben in einzelne Softwarefiles. Es gibt für jedes Sourcefile ein Headerfile mit den Funktionsprototypen und den spezifischen Variablen. Der gesamte Code ist in C erstellt. In Abb. 12 sind die Files und deren Aufgaben ersichtlich.

¹³ Vgl. http://www.atmel.com/dyn/resources/prod_documents/doc8068.pdf (2010), S38

<i>Sourcefile</i>	<i>Headerfile</i>	<i>Aufgaben</i>
rrt_ApplicationLayer.c	rrt_ApplicationLayer.h	Dienste für das Senden und Empfangen von Daten;
rrt_TransportLayer.c	rrt_TransportLayer.h	Dienste für das Zuweisen von Portnummern zu den Applikationen(Tasks); Dienste für das Aktivieren der zuständigen Applikationen;
rrt_DataLinkLayer.c	rrt_DataLinkLayer.h	Dienste für das Erstellen des Sendeframes und das Parsen der empfangenen Messages; Dienste für die Berechnung der CRC Checksum;
rrt_ReceiveTask.c	rrt_ReceiveTask.h	Dienste für das zyklische Abfragen der Empfangs FIFOS;
rrt_TimeoutManager.c	rrt_TimeoutManager.h	Dienste für das Erkennen von Zeitüberschreitungen nach dem Senden einer Nachricht, bis zur Bestätigung durch den Empfänger;
rrt_Serialconfig.c	rrt_Serialconfig.h	Konfiguration der USARTS
rrt_Usart_driver.c	rrt_Usart_driver.h	Treiber für die USARTS des Mikrokontrollers;

Abb. 12: Implementierte Files und deren Funktionen

6 PROTOKOLLANALYSE

6.1 Effizienz

Die Protokolleffizienz $E[\%]$ (For. 6.1) lässt sich durch das Verhältnis von Headerbytes H zur gesamten, versendeten Datenmenge pro Frame (Headerbytes (H) + Nutzdatenbytes (N)) angeben. Offensichtlich steigt die Protokolleffizienz mit steigender Anzahl der Nutzdatenbytes.

$$E[\%] = \left(1 - \frac{H}{H + N} \right) * 100 \quad (\text{For. 6.1.1})$$

Bei einer Nutzdatenanzahl von 50 Bytes ergibt sich eine Effizienz E von ca. 83%, wohingegen bei einer Nutzdatenanzahl von 10 Bytes nur eine Effizienz von 50% erreicht werden kann. Die Berechnung aus For. 6.1.1 berücksichtigt keine Effizienzminderung der USARTS des Mikrokontrollers. Da es sich um eine asynchrone Verbindung handelt muss die Synchronisation der Abtastung mittels Start- und Stopbit erfolgen. Bei der Einstellung der USARTS auf 8 Datenbits, keiner Paritätsprüfung und einem Stopbit (8, n, 1) kann die Effizienz E einer USART ausgedrückt werden durch:

$$E_{\text{USART}} = 1 - \frac{2}{2+8} * 100 = 80\%$$

Diese Tatsache verändert For. 6.1.1 zu For. 6.1.2:

$$E[\%] = \left(1 - \frac{H}{H+N}\right) * E_{\text{USART}} \quad (\text{For. 6.1.2})$$

Somit ergibt eine Nutzdatenanzahl von 50 Bytes eine Effizienz E von ca. 66,7% und eine Nutzdatenanzahl von 10 Bytes nur mehr 40% (Abb. 13).

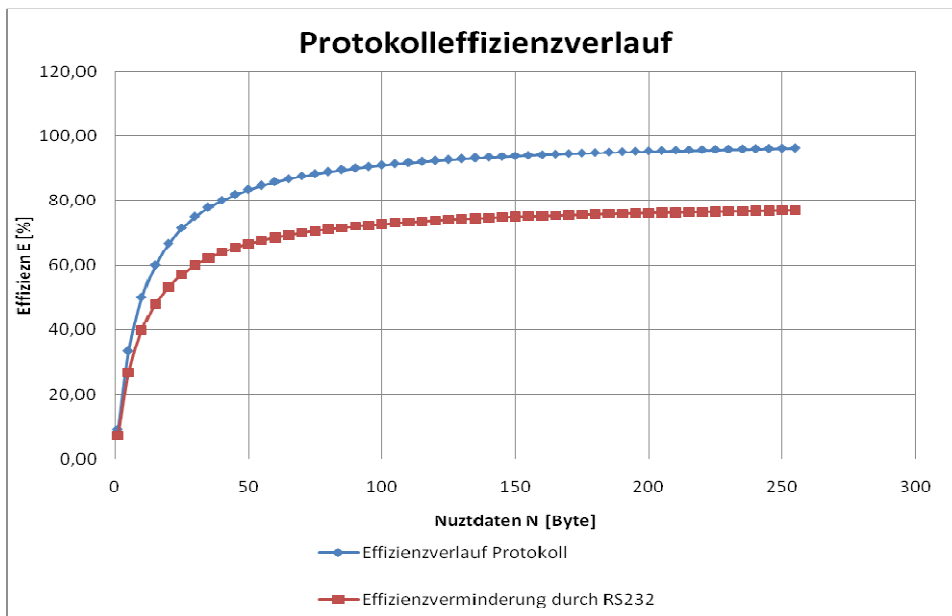


Abb. 13: Protokolleffizienzverlauf

6.2 Zeitverhalten

Für die Analyse des Zeitverhaltens wurden folgende Zeiten gemessen:

1. Die Zeitdauer des Empfangens der Nachricht.
2. Die Zeitdauer des Sendens des Bestätigungsframes.
3. Die Zeitdauer des Sendens der Nachricht.
4. Die Zeitdauer nach dem Empfang der Nachricht bis der Zieltask diese ausgelesen hat.
5. Die Zeitdauer vom Beginn des Sendens bis zum Empfangen der Nachricht.

Es wurde dafür ein Testaufbau mit zwei ATXmega256a3 Mikrocontroller erstellt. Beide Mikrocontroller versenden zyklisch (10ms) eine 30 Byte Nachricht (inkl. Protokollheader) auf zwei USARTS im Voll-Duplex Betrieb mit 230400baud bei einer Taktfrequenz von 32MHz.

Zu 1. und 2.:

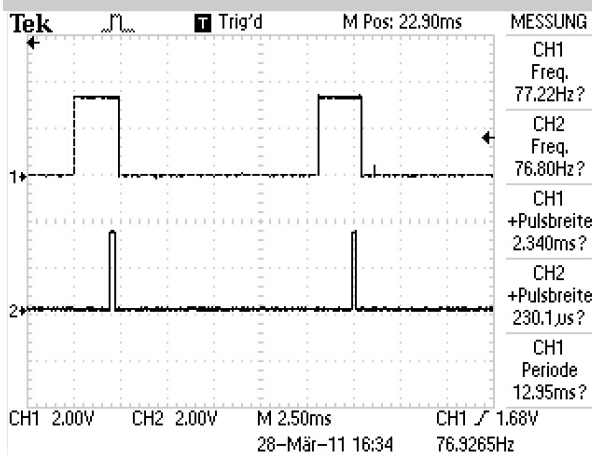


Abb. 14: Empfang und Bestätigung einer Nachricht

Aus Abb. 14 folgt, dass die Empfangsdauer der Nachricht 2,34ms beträgt (Kanal 1). Diese Zeit beinhaltet auch die Dauer nach dem Empfang der Nachricht, bis der Zieltask diese ausgelesen hat. Nach dem gültigen Empfang der Nachricht wird vom Empfänger ein Bestätigungsframe gesendet. Die Zeit für die Bestätigung beträgt 230,1µs (Kanal 2).

Die Empfangszeit (t_e) setzt sich aus der USART Sendezeit ($t_u(b,u,n)$) in Abhängigkeit der Baudrate (b), der Anzahl der Bits pro gesendetem Byte (u) und der Anzahl der Sendebytes (n), aus der Durchlaufzeit des Protokolls (t_d) und aus der Zeit bis der Scheduler dem aktivierten Task die CPU Rechenleistung zuweist (t_s).

$$t_u(b, u, n) = \frac{1}{b} * u * n = \frac{1s}{230400baud} * 10bit * 30Byte = 1,3ms \quad (\text{For. 6.2.1})$$

$$t_e = t_u(b, u, n) + t_d + t_s \quad (\text{For. 6.2.2})$$

Die Zeit t_s hängt von der Anzahl und Abarbeitungszeit der Systemtasks ab und verändert sich damit in jeder Anwendung. Die Durchlaufzeit des Protokolls beträgt hier ca. 1ms.

Zu 1. und 3.:

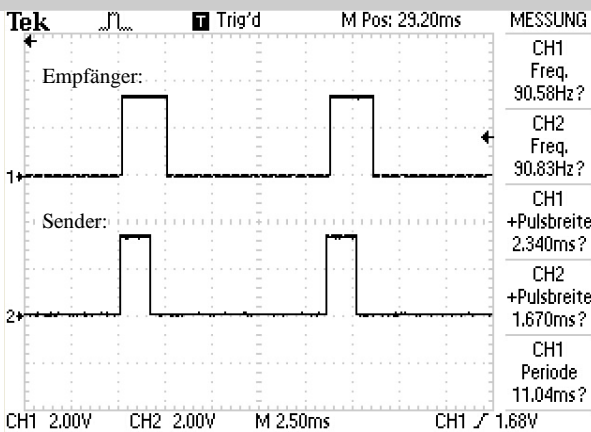


Abb. 15: Senden und Empfangen einer Nachricht

Abb. 15 zeigt, dass das Senden einer Nachricht deutlich schneller erfolgt als das Empfangen. Dies ist auf die Tatsache zurückzuführen, dass beim Empfang einer Nachricht die Auswertung der Daten und die Gültigkeitsprüfung erfolgen muss.

Zu 4.:

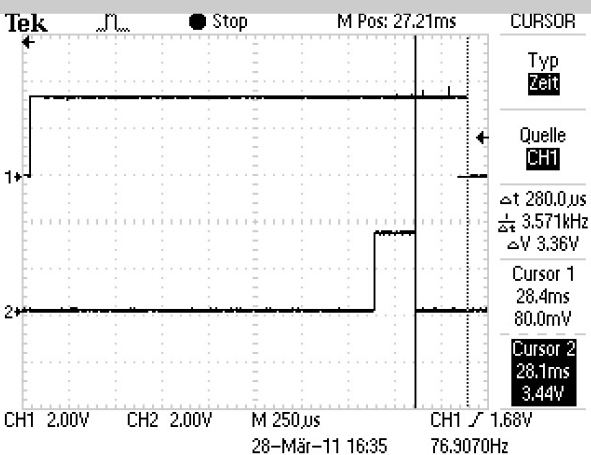


Abb. 16: Empfangen einer Nachricht bis der aktivierte Task diese ausgelesen hat

In Abb. 16 ist die benötigte Zeit (t_s) dargestellt, bis die Daten vom Zieltask ausgelesen wurden. In der Versuchskonstellation wurden fünf Systemtasks eingebunden, die zyklisch aufgerufen wurden aber nur eine sehr geringe Abarbeitungszeit aufwiesen. Die Zeit t_s für diesen Fall beläuft sich auf $280\mu s$. Es kann aber davon ausgegangen werden, dass sich diese Zeit mit steigenden Steueraufgaben deutlich erhöht.

Zu 5.:

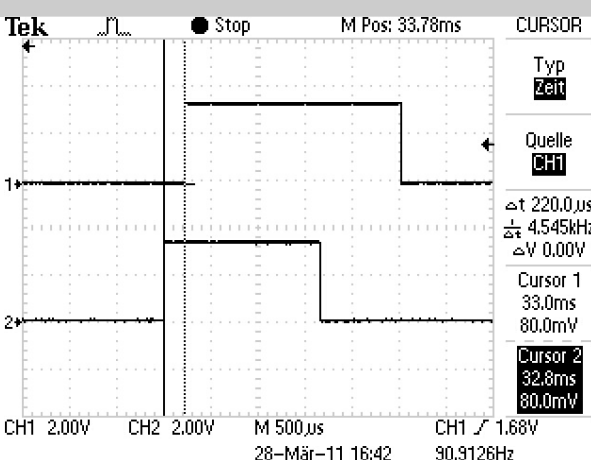


Abb. 17: Zeit vom Start des Sendens bis der Empfänger reagiert

Da die Empfangsbuffer zyklisch abgefragt werden, ist zwischen Sendebeginn und Empfangsbeginn eine Latenzzeit zu erkennen, die maximal einem Schedulingzyklus des Multitaskingsystems entspricht (Abb. 17).

7 ZUSAMMENFASSUNG

Das hier vorgestellte serielle Mehrschichtenprotokoll wurde bereits erfolgreich auf zwei Robotern der Fachhochschule Wels eingesetzt, die für die Eurobot 2011 konstruiert wurden. Es hat sich herausgestellt, dass ein Großteil der versendeten Nachrichten zwischen den Steuermikrokontrollern in einem Bereich von zwei bis fünf Nutzdatenbytes liegt und somit der Anteil der Headerdaten (10Bytes) eine Effizienzmindering ergibt. Durch die relativ hohe Baudrate hat sich diese Einschränkung aber nicht negativ ausgewirkt. Werden künftig noch mehrere Mikrokontroller über dieses Protokoll vernetzt, so ist es vorteilhaft die Headerdatenmenge zu reduzieren um die Leistungsfähigkeit zu steigern.

Eine denkbare Erweiterungsmöglichkeit ist die Datensegmentierung bei größeren Datenmengen wie zum Beispiel Bilddateien. Dies könnte als weiterer Dienst in der Transportschicht realisiert werden.

8 LITERATUR

8.1 Bücher

[A. Tannenbaum, 2003] Tannenbaum, Andrew S.: Computernetzwerke, Hrsg. Pearson Education
Deutschland GmbH

[R. Bauch, 2009] Bauch, Roland; B. Thomas: Netzwerk – Grundlage, Hrsg. Herdt-Verlag

8.2 Internetquellen

[Eurobot-Regeln, 2011] http://www.eurobot.org/commonfiles/docs/2011/E2011_Rules-EN.pdf,

Stand vom 27.09.2010

[ISO/IEC Standard 7498-1, 1994] http://www.iso.org/iso/catalogue_detail.htm?csnumber=20269,

Stand vom 21.06.2000

8.3 Verwendete Publikationen

Die angeführten Bachelorthesen sind unter folgendem Link veröffentlicht:

<http://rrt.fh-wels.at/publications.html>

[M. Zauner, 2009] Zauner, Michael: Modulares und skalierbares elektronisches System für autonome Roboter

[M. Zauner, 2009] Zauner, Michael: Konzept und Programmierung der Pfadplanung und Strategie eines Roboters für die EUROBOT open