Hochschule
Bonn-Rhein-Sieg
Fachbereich 03: Elektrotechnik, Maschinenbau, Technikjournalismus

Diplomarbeit

Entwicklung und Implementierung eines Webservers zur dynamischen Erzeugung von HTML-Code auf Basis des 8-bit Mikrocontrollers 8051F120

Zur Erlangung des akademischen Grades
Diplom-Ingenieur (FH)
an der Hochschule Bonn-Rhein-Sieg

vorgelegt von
Oliver Polch
aus
Sinzig

Sankt-Augustin 2010

1. Prüfer:
Prof. Dr. rer. nat. Bernd Klein

2. Prüfer:
Prof. Dr. -Ing. Marco Winzker

I Inhaltsverzeichnis

II Abbildungsverzeichnis

III Tabellenverzeichnis

IV Listings-Verzeichnis

V Abkürzungsverzeichnis

ASCII	American Standard Code for Information Interchange
CSS	Cascading Stylesheets
DHCP	Dynamic Host Configuration Protocol
FIFO	First-In-First-Out
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
MAC-Adresse	Media-Access-Control-Adresse / Hardware-Adresse
OSI	Open System Interconnect
PC	Personal Computer
PRAM	Programmable Random-Access-Memory
PROM	Programmable Read-Only-Memory
RAM	Random-Access-Memory
ROM	Read-Only-Memory
SD-Speicherkarte	Secure Digital Speicherkarte
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
URI	Uniform Ressource Identifier
URL	Uniform Ressource Locator

1 Einleitung

Betrachtet man aktuelle Elektronikprodukte aus den Bereichen Prozesssteuerung, Messdatenaufnahme oder Sicherheitstechnik, so fällt auf, dass viele der auf dem Markt erhältlichen Geräte über die Möglichkeit der *Fernwartung* verfügen. Hierbei ist in den meisten Fällen ein kleiner Webserver auf Basis eines Mikrocontrollers auf dem Gerät implementiert, der dem Anwender die Möglichkeit bietet, über einen Webbrowser auf sein Gerät zuzugreifen. Auf diese Weise können zum Beispiel Messdaten aus Data-Loggern ausgelesen, Regelparameter von Heizungsanlagen gesetzt oder Bilder einer IP-Kamera verfolgt werden.
Diese Art der Fernwartung ist besonders komfortabel, da außer einem PC mit installiertem Webbrowser keine Soft- oder Hardware benötigt wird um auf ein Gerät zuzugreifen.
Um einen Webserver auf Basis eines Mikrocontrollers umzusetzen, muss zum einen eine Schnittstelle für die Netzwerkkommunikation geschaffen werden, zum anderen ist eine Funktionalität zu implementieren, die es dem Webserver ermöglicht, Anfragen eines Client-Rechners auszuwerten. Der Webseitenprogrammierer soll weiterhin eine komfortable Möglichkeit haben, Webinhalte auf dem Webserver abzulegen und zu bearbeiten.
Diesen Überlegungen wird in der hier vorgestellten Arbeit nachgegangen. Ausgehend davon werden im nachfolgenden Kapitel 2 die Ziele der Arbeit aufgeführt. In Kapitel 3 ist der aktuelle Stand der Technik beleuchtet. Kapitel 4 beschreibt den eingeschlagenen Realisierungsweg.
Die Hauptteile der Arbeit sind die Beschreibung der durchgeführten Arbeiten bei der Entwicklung des Webservers in Kapitel 5 sowie der Ergebnisteil in Kapitel 6.
Eine Zusammenfassung, die den Nutzen der Arbeit beschreibt und einen Ausblick auf zukünftige Weiterentwicklungen gibt, schließt die Arbeit ab.

2 Ziele

Ziel dieser Arbeit ist die Entwicklung eines Webservers zur Erzeugung von HTML-Code auf Basis eines 8-bit Mikrocontrollers. Der Webserver soll in der Lage sein, Anfrage-Nachrichten von Client-Rechnern zu analysieren und geeignete Antwort-Nachrichten zu versenden. Hierzu muss dem Webserver eine Funktionalität implementiert werden, die einerseits die Netzwerkkommunikation über standardisierte Netzwerkprotokolle wie TCP/IP[1] sicherstellt, andererseits sind geeignete Funktionen zu programmieren, die eine Analyse von HTML-Zeichenketten ermöglichen.
Die fünf Kernaspekte der Arbeit sind die *Festlegung eines geeigneten Hardware-Designs*, die *Entwicklung effizienter Softwarestrukturen zur Analyse von Anfragenachrichten*, die *Verarbeitung großer Datenmengen* bei der Generierung von Webseiten, die Umsetzung eines geforderten Funktionsumfangs zur *Erzeugung von HTML-Elementen* auf einer Webseite sowie das Erstellen eines *Handbuchs*, welches die Funktionalität des Webservers dokumentiert.

- **Festlegung eines geeigneten Hardware-Designs**

Bei der Auswahl geeigneter Hardware ist eine Marktanalyse durchzuführen. Fertige Produktlösungen sind mit Entwicklungsansätzen gegenüberzustellen. Sollte keine fertige Lösung auf dem Markt erhältlich sein, ist ein geeignetes Hardware-Konzept für eine Neuentwicklung zu erarbeiten. Hierbei sind Aspekte wie *flexible Einsatzmöglichkeit* und *schlankes Design* zu berücksichtigen.

- **Entwicklung effizienter Softwarestrukturen**

Um Anfragenachrichten zu analysieren müssen geeignete Funktionen zur Zeichenkettenanalyse implementiert werden. Bei der Softwareentwicklung ist insbesondere auf effiziente Softwarestrukturen zu achten. Im späteren Betrieb wird der Webserver auf einem Mikrocontroller eingesetzt, der zusätzlich zu der Webserveranwendung noch weitere Aufgaben ausführt. Daher muss die Webserveranwendung über schlanke, nichtblockierende Programmstrukturen realisiert werden, die *Multitasking*[2] ermöglichen.

- **Verarbeitung großer Datenmengen**

Beim Generieren und Verarbeiten von Webseiten können mitunter große Datenmengen entstehen. Diese sind auf Mikrocontrollern mit begrenzten

[1] Das Transmission Control Protocol / Internet Protocol (TCP/IP) ist eine Familie von Netzwerkprotokollen, die den Standard für moderne Netzwerkkommunikation festlegen.

[2] Auf einem Mikrocontroller kann Multitasking nie nebenläufig umgesetzt sein, da Programme seriell abgearbeitet werden. Man spricht daher von einem quasi parallelen Ausführen von mehreren Prozessen auf einem Prozessor.

Speicherressourcen nur sehr schwer zu bewältigen. Daher ist ein Verfahren zu entwickeln, das den Umgang mit diesem Datenmengen auf dem Webserver ermöglicht.

- **Umsetzen eines geforderten Funktionsumfangs**

Es bestehen feste Vorgaben über den Funktionsumfang, den der Webserver dem späteren Anwender zur Verfügung stellen soll. Er muss eine komfortable Möglichkeit bieten, die in Tabelle 2-1 aufgeführten HTML-Elemente erstellen und auf einer Webseite platzieren zu können. Zusätzlich ist gefordert, Webseiten formatieren zu können. Hierzu sind geeignete Funktionen zu entwickeln.

Tabelle 2-1: Geforderte HTML-Elemente

Element	***HTML-Bezeichnung***[3]	***Beispiel***
Einzeiliges Eingabefeld	Inputfield	Test
Mehrzeiliges Eingabefeld	Multiline-Inputfield	Zeile_1 Zeile_2
Auswahlliste	List-Box	Auswahl_1 Auswahl_1 Auswahl_2
Tabelle	Table	Zelle_1 | Zelle_2
Schaltfläche	Button	Button_1
Text	Text	Text auf einer Homepage
Überschrift	Headline	**Webseitenüberschrift**
Optionsfeld	Radio-Button	
Auswahlfeld	Check-Box	
Liste	List	**- Listeneintrag 1** **- Listeneintrag 2**
Verknüpfung	Link	http://www.beispiel.de

- **Erstellen eines Handbuchs**

Um den Funktionsumfang des Webservers zu dokumentieren, soll ein Handbuch angefertigt und an diese Arbeit angehängt werden.

[3] Der Standard für die „Hypertext Markup Language“ ist in ISO/IEC 15445 festgehalten. Sie wird von dem „World Wide Web Consortium“ (W3C) weiterentwickelt.

3 Stand der Technik

3.1 Funktionsweise eines Webservers

Die Aufgaben eines Webservers lassen sich in zwei Bereiche unterteilen. Zum einen ist dies die Kommunikation im Netzwerk, welche die Grundlage für den Datenaustausch zwischen einem Webserver und einem Client-Rechner darstellt. Zum anderen dient die eigentliche Webserver-Software dazu, Anfragen eines Client-Rechners zu verarbeiten und zu beantworten. Im Folgenden werden diese zwei Teilapplikationen mit *Kommunikationsmodul* und *Webserveranwendung* bezeichnet.
Bild 3-1 veranschaulicht deren Funktionsweise anhand eines Beispiels[4]. Der Client-Rechner fordert hier über einen Webbrowser eine Webseite auf dem Webserver an. Er erzeugt hierzu eine Anfrage, auf welche der Webserver mit einer Antwort reagiert.

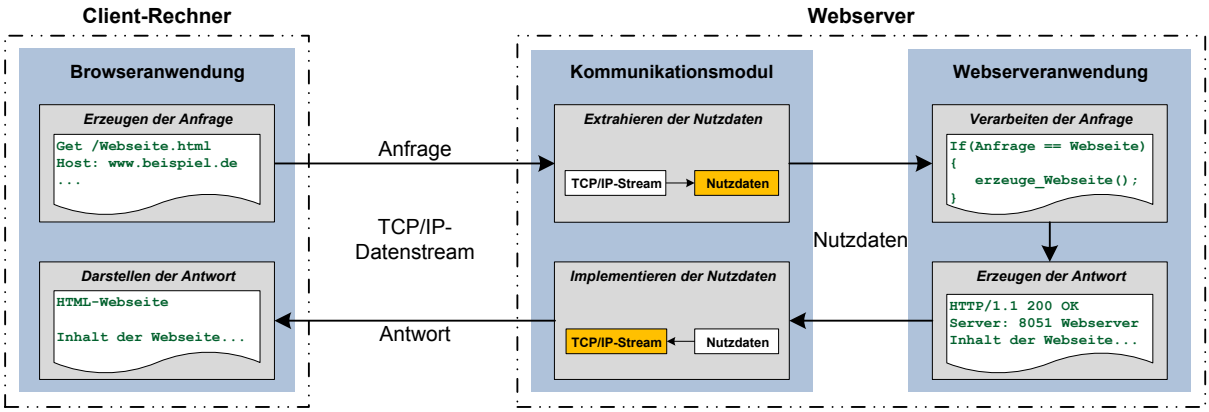

Bild 3-1: Netzwerkkommunikation zwischen Webserver und Client-Rechner

- **Das Kommunikationsmodul**

Das Kommunikationsmodul ist für die Kommunikation des Webservers mit anderen Geräten im Netzwerk zuständig. Diese Kommunikation wird über die Protokolle der *TCP/IP-Protokollfamilie* realisiert. Grundlage hierfür ist das *TCP/IP-Referenzmodell*, welches sich vom *OSI-Referenzmodell* ableitet [1]. Die TCP/IP-Protokollfamilie beinhaltet standardisierte Kommunikationsprotokolle, die es Geräten ermöglichen, Daten auch über die Grenzen lokaler Netzwerke hinaus auszutauschen. In dem Kommunikationsmodul sind Funktionen implementiert, die eine Kommunikation über diese Protokolle ermöglichen.
Hauptaufgabe des Moduls ist es Anfragen anzunehmen, die von Client-Rechnern an den Webserver gerichtet werden. In einem weiteren Schritt die Nutzdaten aus diesen Anfragen zu extrahieren und aufbereitet an die Webserveranwendung weiterzuleiten.

[4] Alle in Bild 3-1 aufgeführten Code-Beispiele sind als Pseudocode zu verstehen. Sie dienen nur zum leichteren Verständnis des Ablaufs. Ähnlich dem Webserver besteht auch der Client-Rechner aus mehreren Teilapplikationen. Aus Gründen der Übersichtlichkeit wird auf diese jedoch verzichtet.

In entgegengesetzter Richtung erhält das Kommunikationsmodul Nutzdaten von der Webserveranwendung und leitet diese an den Client-Rechner weiter.

- **Die Webserveranwendung**

Die eigentliche Webserveranwendung verarbeitet Daten, die ihr von dem Kommunikationsmodul übermittelt wurden. Dies sind die Nutzdaten aus den Anfragen des Client-Rechners. Die Anwendung beantwortet diese Anfragen durch Zurücksenden von Dokumenten, welche aus statischen oder dynamisch erzeugten Daten bestehen können. In dem in Bild 3-1 dargestellten Beispiel erzeugt die Webserveranwendung als Antwortnachricht den HTML-Code der angeforderten Webseite.

3.2 Entwicklungsansätze

3.2.1 Hardware

Webserver auf Basis eines Mikrocontrollers können nach heutigem Stand der Technik hardwareseitig über zwei Entwicklungsansätze realisiert werden[5]. Am meisten verbreitet sind Lösungen mit zwei getrennten Chips, im Folgenden als *Multi-Chip-Modul* bezeichnet. Es werden jedoch auch *Single-Chip-Module* angeboten, bei denen der gesamte Webserver auf einem Mikrochip realisiert ist. Die verschiedenen Ansätze sind in den folgenden Kapiteln näher erläutert.

3.2.1.1 Multi-Chip-Modul

Bei diesem Entwurfsansatz besteht das Webservermodul aus zwei getrennten Mikrocontrollern (gelegentlich auch ein FPGA und ein Mikrocontroller). Einem sogenannten *Ethernet-Interface-Chip*, welcher die Kommunikation des Moduls im Netzwerk sicherstellt, und einem sogenannten *Webserver-Chip*, auf dem die Webserver-Software ausgeführt wird. Die Kommunikation der beiden Mikrocontroller untereinander ist meist über einen Datenbus realisiert.

- **Ethernet-Interface-Chip**

Der Ethernet-Interface-Chip ist für die Kommunikation des Webservers im Netzwerk zuständig. Übertragen auf das in Bild 3-1 dargestellte Schema, dient er als Kommunikationsmodul des Webservers.
Ethernet-Interface-Chips sind in verschiedenen Ausführungen erhältlich. Die Unterschiede liegen hierbei in den darin implementierten Übertragungsprotokollen. Einige der auf dem Markt erhältlichen Produkte realisieren die Kommunikation bis in die Transportschicht des TCP/IP-Referenzmodells. Sie verarbeiten somit

[5] Die Verbreitung der verschiedenen Realisierungsansätze ist durch Marktanalyse ermittelt worden.

selbstständig alle Protokolle der TCP/IP-Protokollfamilie. Andere Chips hingegen sind nur mit der Funktionalität zur Behandlung der Netzzugangsschicht und/oder der Vermittlungsschicht ausgestattet. Hier muss der Anwender die nötigen Transport-Protokolle selber implementieren.

- **Webserver-Chip**

Auf dem Webserver-Chip wird die eigentliche *Webserver-Software* ausgeführt. Hier kann der Anwender Programme zur dynamischen Erzeugung von HTML-Webseiten oder statische Webseiten ablegen. Die Webserver-Software bearbeitet Anfragen von einem Client-Rechner und stellt Dokumente mit den angeforderten Informationen zur Verfügung. Übertragen auf das in Bild 3-1 dargestellte Schema, dient der Webserver-Chip dazu, die *Webserveranwendung* auszuführen.

3.2.1.2 Single-Chip-Modul

In Single-Chip-Modulen sind prinzipiell die gleichen Komponenten vorhanden, wie sie auch in Multi-Chip-Modulen eingesetzt werden. Der Unterschied liegt jedoch darin, dass die komplette Funktionalität in *einem Chip* implementiert ist. Dieser Ansatz ist deutlich kompakter, da nur ein einziger Chip auf der Schaltung zum Einsatz kommt. Er bietet im Gegensatz zu Multi-Chip-Modulen jedoch weniger Flexibilität. Vollintegrierte Single-Chip-Lösungen sind auf dem Elektronikmarkt nur wenige erhältlich. Ist man in einem Projekt auf einen Mikrocontroller mit möglichst vielen Digital- Analogwandler angewiesen, ist dies mit einem Single-Chip-Modul möglicherweise nicht realisierbar, da ein solches Modul auf dem Markt nicht existiert. Bei getrennten Chips besteht dieses Problem nicht.

3.2.2 Software

Um Webinhalte per Mikrocontroller zu erzeugen, sind softwareseitig hauptsächlich zwei Ansätze verbreitet. Da bei der Generierung von Webseiten mitunter große Datenmengen entstehen, die auf Mikrocontrollern mit begrenzten Speicherressourcen nicht vorgehalten werden können, verfolgen beide Ansätze das Ziel, effizient mit den Datenmengen umzugehen. Eine Möglichkeit besteht darin, die Webinhalte auf einen *externen Speicher* auszulagern. Eine weitere Möglichkeit bietet die Methode der *dynamischen Erzeugung* von Webinhalten. Beide Varianten sind im Folgenden näher erläutert.

3.2.2.1 Auslagerung der Webinhalte

Bei dieser Methode werden die Webinhalte (meist HTML- oder Java-Code zur Erzeugung von Webseiten) auf einem externen Speicher ausgelagert. Häufige Verwendung finden hier SD-Speicherkarten, die zum einen sehr viel Speicherplatz zur Verfügung stellen und zum anderen mit wenig Aufwand über einen Datenbus

angebunden werden können. Die Daten auf der Speicherkarte sind in einem für PCs lesbaren Dateiformat abgelegt, somit können Webinhalte komfortabel über einen PC auf der Speicherkarte angelegt und editiert werden. Der Mikrocontroller liest diese Daten blockweise von der Speicherkarte aus und sendet sie an einen Empfänger.
Diese Herangehensweise verfolgt zwei Ziele. Zum einen kann das Problem der großen Datenmengen auf einem Mikrocontroller vermieden werden. Zum anderen ist es für den Anwender sehr einfach möglich, Webinhalte zu erstellen oder zu verändern. Hierzu muss er lediglich die Dateien auf der Speicherkarte abändern. Eine Neuprogrammierung des Webserver-Chips ist nicht nötig.
Diese Methode wird sehr oft von Herstellern sogenannter *Embedded Webserver* (siehe Kapitel 3.3) gewählt. Sie macht den Webserver auch für Anwender interessant, die nur über geringe Kenntnisse in der Mikrocontrollerprogrammierung verfügen. Ein Basiswissen in der Programmierung von HTML- und Java-Webseiten reicht aus um den Webserver zu betreiben.

3.2.2.2 Dynamische Erzeugung der Webinhalte

Ein weiterer Ansatz Webinhalte mit Hilfe eines Mikrocontrollers zu generieren, ist das *dynamische Erzeugen* von Webseiten. Der Vorteil dieser Methode ist, dass kein externer Speicher benötigt wird, auf dem die Webinhalte abzulegen sind. Bei jeder Anfrage, die ein Client-Rechner an den Webserver richtet, erzeugt dieser die Antwort dynamisch. Durch effiziente Programmierung kann somit der Speicherbedarf auf dem Mikrocontroller minimiert werden. Webseiten werden bei dieser Methode mit Hilfe fest implementierter Funktionen generiert.
Die Anforderungen an den Anwender sind hier jedoch deutlich höher als bei der zuvor vorgestellten Methode (siehe Kapitel 3.2.2.1). Der Anwender benötigt zur Implementierung seiner Webanwendungen grundlegende Kenntnisse in der Mikrocontrollerprogrammierung. Daher findet diese Variante deutlich häufiger ihren Einsatz in der Entwicklung von Speziallösungen. Dem geübten Anwender bieten sich dabei große Vorteile durch die flexible Einsetzbarkeit des Moduls. Er kann seine Webanwendung ideal auf seine Bedürfnisse anpassen und ist nicht auf den oftmals geringen Funktionsumfang beschränkt, den ihm eine fertige Lösung liefert.

3.3 Fertige Produktlösungen

Fertige Produktlösungen, die den in diesem Projekt gestellten Anforderungen gerecht werden, finden sich auf dem Elektronikmarkt nur sehr wenige. Die hohe Flexibilität, die von dem Webservermodul gefordert ist, bietet keines der auf dem Markt erhältlichen Produkte vollständig.
Ein sehr interessantes Produkt wird unter der Bezeichnung *Embedded Webserver* angeboten. Dabei handelt es sich um vollintegrierte Webserverlösungen, die nach geringem Konfigurationsaufwand einsatzbereit sind und dem Anwender trotzdem

eine gewisse Flexibilität bieten. Der Funktionsumfang solcher Module wird im Folgenden beispielhaft an einem Produkt der Firma *Laser & Co. Solutions GmbH*[6] erläutert. Module anderer Hersteller bieten bis auf einige Details den gleichen Funktionsumfang.

Der Embedded Webserver myEthernet

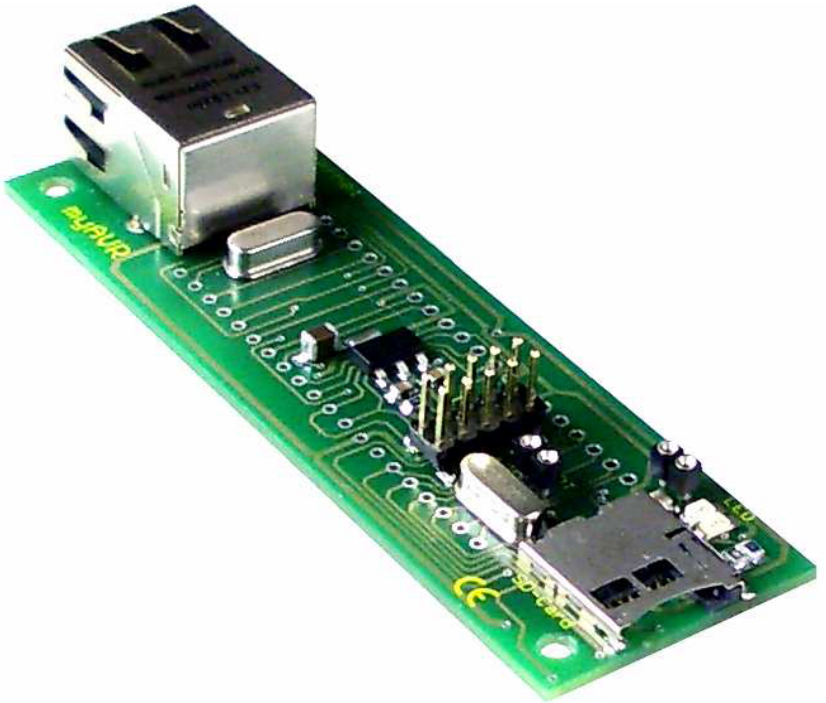

Bild 3-2: Der Embedded Webserver myEthernet

Das Modul *myEthernet* der Firma Laser & Co. Solutions GmbH (siehe Bild 3-2) ist als *Multi-Chip-Modul* (siehe Kapitel 3.2.1.1) realisiert. Es basiert auf einem ATmega644P des Herstellers Atmel. Als Ethernet-Interface-Chip kommt der ENC28J60 der Firma Microchip zum Einsatz. Zusätzlich ist auf dem Board ein MicroSD-Kartenhalter angebracht. Angaben zum Leistungsumfang des Produkts sind in Tabelle 3-1 aufgeführt. Weitere Details zu dem Modul können in [3] nachgelesen werden.

6 Die Firma Laser & Co. Solutions GmbH vertreibt über ihre Webseite www.myavr.de diverse Elektronikmodule.

Tabelle 3-1: Leistungsmerkmale des Embedded Webservers myEthernet

Leistungsmerkmal	*Ausführung*
Ethernet	10 Mbit Ethernet mit ENC28J60 von Microchip
Webservercontroller	ATmega644P 20 MHz mit vorinstalliertem Webserver
FLASH-Speicher	64 kbit
SRAM-Speicher	4 kByte
EEPROM-Speicher	2 kByte
Schnittstellen	TWI / UART / SPI / I²C / Ethernet
Digital I/O	max. 22
Analog-Eingänge	max. 8
PWM-Ausgänge	2

- **Konfiguration und Anwendung des Moduls**

Die Konfiguration des Moduls, sowie die Implementierung der Webseiten werden ausschließlich über die mitgelieferte Speicherkarte realisiert. Diese enthält eine Konfigurationsdatei für das Kommunikationsmodul und weitere Dateien in denen die Webinhalte hinterlegt sind. Weder auf dem Ethernet-Interface-Chip noch dem ATmega644P muss C-Code editiert oder entwickelt werden.
In der Konfigurationsdatei des Kommunikationsmoduls werden unter anderem die Parameter für den Aufbau einer TCP/IP-Verbindung festgelegt. Hier sind zum Beispiel die IP-Adresse, die Subnetzmaske und ein Standardgateway einzutragen. Das Modul bietet zusätzlich die Möglichkeit, die IP-Adresse von einem DHCP-Server zu beziehen.
Die eigentlichen Webdateien können in HTML- oder Java-Code programmiert werden. Eine speziell für dieses Modul entwickelte Syntax ermöglicht es, folgende Aktionen über eine Webseite ausführbar zu machen:

1. Setzen von Signalen an Ausgangs-Pins des Mikrocontrollers
2. Visualisieren von Signalen an Eingangs-Pins des Mikrocontrollers

Hierzu integriert der Anwender Befehle in den Webseitencode, welche dann abhängig von einer Aktion auf der Webseite (zum Beispiel bei Drücken einer Schaltfläche) auf dem Mikrokontroller ausgeführt werden. Die Syntax ist ausführlich dokumentiert und leicht verständlich. Basiskenntnisse in der HTML- und Java-Programmierung reichen aus, um einfache Webseiten auf dem Webservermodul

umzusetzen. Die Daten auf der Speicherkarte sind in dem Dateiformat FAT32 abgelegt. Die gesamte Konfiguration des Webservers kann somit am PC geschehen.

- **Einsatzgebiet**

Man erkennt schnell, dass dieses Modul ohne zusätzliche Hardware nicht sinnvoll einzusetzen ist. Wie die meisten anderen Hersteller bietet auch Firma Laser & Co. Solutions GmbH das myEthernet als Erweiterungsmodul zu ihren Mess- und Steuerungsplatinen an. Standardisierte Steckverbinder verbinden die einzelnen Komponenten zu einem Gesamtsystem. Zusammen mit dem Modul myTWI (siehe Bild 3-3) welches zur Temperaturmessung dient, besteht so zum Beispiel die Möglichkeit, Temperaturen von angeschlossenen Temperatursensoren auf einer Webseite zu visualisieren.

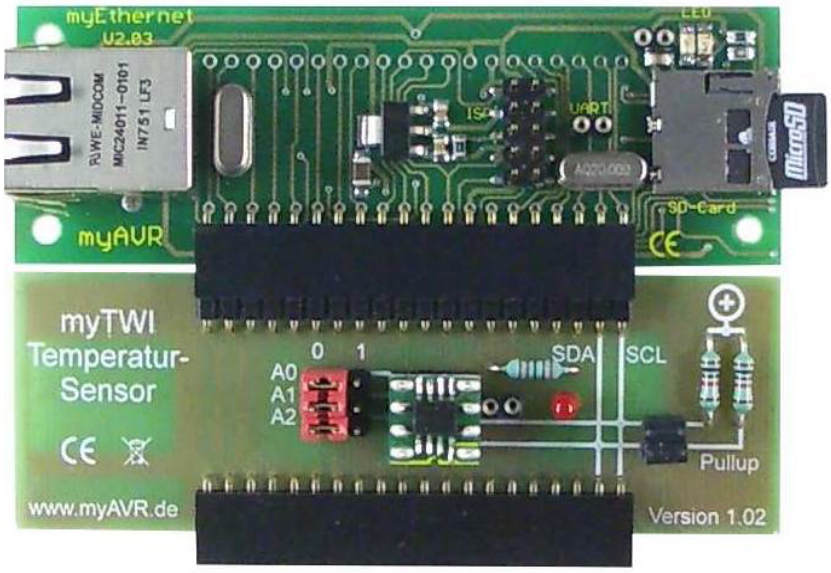

Bild 3-3: Das Modul myEthernet mit dem Temperaturmodul myTWI

4 Eingeschlagener Realisierungsweg

4.1 Hardware

4.1.1 Generelles Hardware-Design

In Kapitel 3.2.1 ist beschrieben, welche Realisierungsansätze es nach aktuellem Stand der Technik gibt, einen Webserver auf Basis eines Mikrocontrollers zu entwickeln. Das hier beschriebene Projekt wird als *Multi-Chip-Modul* (siehe Kapitel 3.2.1.1) realisiert. Hauptkriterium für diese Entscheidung ist die Flexibilität, die dieser Entwicklungsansatz für spätere Erweiterungen bietet. Durch die Separation des Ethernet-Interface-Chips von dem Webserver-Chip, kann die Webserveranwendung auf verschiedensten Mikrocontrollern implementiert werden.
Eines der Ziele dieses Projekts besteht darin, das Webserver-Modul für verschiedene Anwendungsfälle nutzbar zu machen. Durch die Wahl des Multi-Chip-Lösungsansatzes ist dies gegeben. Ist zum Beispiel gefordert, eine bestehende Schaltung um einen Webserver zur Visualisierung einiger Parameter zu erweitern, muss hierzu lediglich der Ethernet-Interface-Chip an den vorhandenen Mikrocontroller angeschlossen werden. Dies kann über einige freie Anschlusspins des Mikrocontrollers geschehen und ist mit geringem Aufwand realisierbar. Nach Implementierung der Webserveranwendung und Programmierung einer auf den Anwendungsfall zugeschnittenen Webseite, kann der Webserver seinen Betrieb aufnehmen.

4.1.2 Der Webserver-Chip

Die Wahl des Mikrocontrollers für die Webserveranwendung ist sehr stark davon beeinflusst, auf welcher Hardware die Anwendung im späteren Betrieb implementiert werden soll. Der Mikrocontroller *8051F120* der Firma *Silicon Laboratories* gilt im Labor als betriebsbewährt und wurde schon für viele Anwendungen verwendet. Es ist sehr wahrscheinlich, dass auch das Webservermodul später auf diesem Mikrocontroller implementiert wird. Da das zu entwickelnde Testboard die späteren Betriebsumstände möglichst gut wiederspiegeln muss, wird in diesem Projekt der 8051F120 verwendet. Dies macht weiterhin Sinn, weil Entwicklungskomponenten wie Evolution-Boards und eine Software-Entwicklungsumgebung im Labor vorhanden sind. Somit werden teure Neuanschaffungen vermieden.

4.1.3 Der Ethernet-Interface-Chip

Für den Ethernet-Interface-Chip bietet sich das Model *W5300* der Firma *Wiznet* an. Dieser Chip ist in der Abteilung ebenfalls schon *betriebsbewährt* und wird in einigen

vorangegangenen Projekten erfolgreich eingesetzt. Ein großer Vorteil des W5300 besteht darin, dass er die Protokolle der TCP/IP-Protokollfamilie bis in die Transportschicht des TCP/IP-Referenzmodells implementiert hat. Viele der auf dem Markt erhältlichen Ethernet-Controller bieten nur die Protokolle der Netzzugangsschicht. Der Programmierer ist somit dafür verantwortlich, die weitere Kommunikation in seiner Anwendung zu realisieren. Aufgrund der Komplexität der Übertragungsprotokolle ist die Entwicklung sehr aufwendig und fehleranfällig.

4.2 Software

4.2.1 Generelles Softwaredesign

Wie in Kapitel 3.2.2 dargestellt ist, sind softwareseitig zwei Ansätze verbreitet, einen Webserver auf einem Mikrocontroller umzusetzen. Für dieses Projekt wird der Ansatz zur *dynamischen Erzeugung von Webinhalten* gewählt (siehe Kapitel 3.2.2.2). Ausschlaggebend dafür ist die Flexibilität, die dieser Ansatz dem Anwender beim Erstellen von Webinhalten bietet. Der Programmierer kann somit den Webserver ideal auf spezielle Anwendungen anpassen.
Da das hier zu entwickelnde Modul in verschiedensten Schaltungen zum Einsatz kommen soll, ist es weiterhin wichtig darauf zu achten, dass möglichst wenig zusätzliche Hardware zu implementieren ist. Platz für externen Speicher, wie er bei der in Kapitel 3.2.2.1 vorgestellten Methode nötig ist, steht bei vielen Anwendungen nicht zur Verfügung.

4.2.2 Nichtblockierende Softwarestrukturen / Multitasking

Um die geforderten *nichtblockierenden* Softwarestrukturen zu realisieren, wird ein Ansatz über *kooperatives Multitasking* verwendet [5]. Beim kooperativen Multitasking gibt es keinen sogenannten Scheduler, der den Ablauf der einzelnen Prozesse kontrolliert. Vielmehr sind die Prozesse so ausgeführt, dass sie Ressourcen selbstständig wieder freigeben. Zustandsautomaten helfen bei der Umsetzung dieser Methode. Hierbei sind die einzelnen Prozesse mittels *Switch-Case-Anweisungen* in mehrere kleine Segmente unterteilt. Ist ein Prozesssegment abgearbeitet, wird überprüft ob die Übergangsbedingung für das nächste Segment erfüllt ist. Ist dies nicht der Fall, wird eine Status-Variable gesetzt und es erfolgt ein Sprung aus der Anweisung heraus. Der Mikroprozessor hat nun die Möglichkeit weitere Prozesse auszuführen. Beim nächsten Sprung in den Zustandsautomat startet der Prozess an der Stelle, wo er verlassen wurde.
Diese Methode findet sehr häufig Anwendung auf Mikrocontrollern, da sie mit vergleichsweise geringem Aufwand implementiert werden kann.

4.3 Nachrichtenlänge eines HTML-Streams ermitteln

Ein wichtiger Parameter bei der Kommunikation zwischen Webserver und Client-Rechner ist die Länge der Antwortnachricht, welche vom Webserver an den Client-Rechner übermittelt wird. Um die Bedeutung dieses Parameters näher zu erläutern, ist im Folgenden ein typischer Kommunikationsablauf zwischen Webserver und Client-Rechner beschrieben. Bild 4-1 zeigt diesen Ablauf in vereinfachter Form, ohne die protokollspezifischen Kontrollkommandos. Der Ablauf wird typischerweise in drei Sequenzen unterteilt [1].

- **Verbindungsaufbau:**

Der Client-Rechner sendet eine Verbindungsanfrage an den Webserver. Ist dieser bereit eine Verbindung herzustellen, bestätigt er den Verbindungsaufbau. Hat der Webserver keine freien Kapazitäten eine weitere Verbindung einzugehen, lehnt er die Anfrage ab.

- **Nutzdatenübertragung:**

Der Client-Rechner formuliert seine Anfrage, im Folgenden „*Request*“ genannt, und übermittelt diese an den Webserver. Nachdem der Webserver die angeforderten Informationen zusammengestellt hat, übermittelt er seine Antwort, im Folgenden „*Response*“ genannt, an den Client-Rechner.

- **Verbindungsabbau:**

Sind alle Anfragen des Client-Rechners abgearbeitet, wird die Verbindung kontrolliert abgebaut. Hierzu teilt der Client-Rechner dem Webserver mit, dass keine weiteren Anfragen vorliegen. Der Webserver bestätigt dies und die Verbindung wird terminiert.

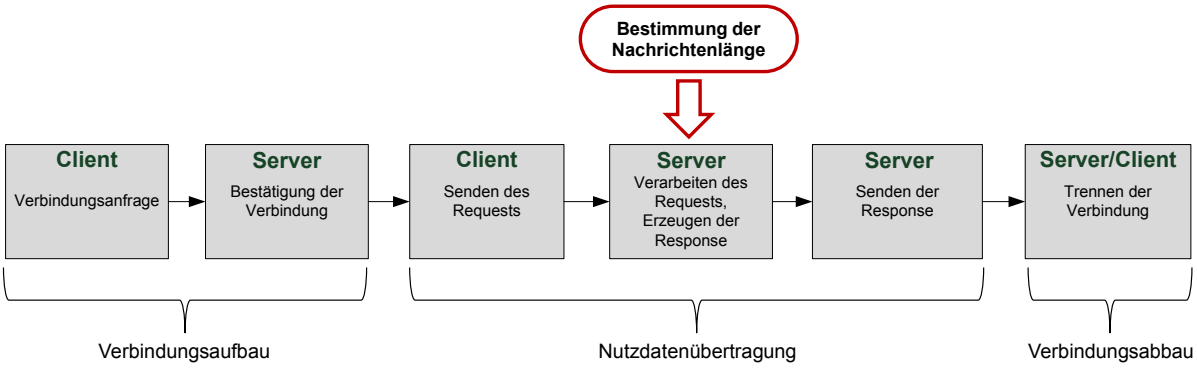

Bild 4-1: Kommunikationsablauf zwischen Webserver und Client-Rechner

4.3.1 Beispiel einer Request- und Response-Nachricht

Im Folgenden Beispiel sind eine typische Request- und Response-Nachricht aufgeführt (siehe Listing 4-1 und Listing 4-2). Der Client-Rechner fordert über einen

sogenannten *GET-Request* die Seite *index.html* auf dem Host *www.beispiel.de* an (die verschiedenen Request-Methoden werden später näher erläutert). Als Response sendet der Webserver eine Nachricht bestehend aus dem sogenannten *Header* (enthält Statusinformationen) und dem *Nachrichten-Body* (enthält die Nutzdaten, hier HTML-Code).

- **Request**

```
GET /index.html HTTP/1.1
Host: www.beispiel.de
```

Listing 4-1: Beispiel eines Requests

- **Response**

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7
Content-Length: 438
Content-Type: text/html

<html>
     <head>
          <title>Startseite</title>
     </head>
     <body>
          Text auf der Startseite
     </body>
</html>
```

Header

Body

Listing 4-2: Beispiel einer Response

Die Länge der Response ist in diesem Beispiel in dem Header-Feld *Content-Length* angegeben (für eine nähere Erläuterung siehe Kapitel 4.3.2). Der hier eingetragene Wert bezieht sich nur auf die Länge des *Nachrichten-Bodys*, nicht der gesamten Nachricht. Der Client benötigt die Nachrichtenlänge um festzustellen, ob eine Nachricht vollständig empfangen wurde. Diesem Parameter wird eine große Bedeutung zugemessen, da er es dem Client-Rechner ermöglicht, unvollständig übertragene Nachrichten zu identifizieren, um diese im Anschluss zu verwerfen oder neu anzufordern.
Die Nachrichtenlänge kann nach verschiedenen Ansätzen bestimmt und übermittelt werden. Nach RFC2616[7] gelten die in Kapitel 4.3.2 bis 4.3.6 aufgeführten Verfahren als standardisiert [4]. Die Implementierung aller Verfahren ist aufgrund begrenzter

[7] Die Requests for Comments (RFC) sind eine Reihe von technischen und organisatorischen Dokumenten zum Internet, die am 7. April 1969 begonnen wurden. Bei der ersten Veröffentlichung noch im ursprünglichen Wortsinne zur Diskussion gestellt, behalten RFC auch dann ihren Namen, wenn sie sich durch allgemeine Akzeptanz und Gebrauch zum Standard entwickelt haben. Die verschiedenen Methoden zur Bestimmung der Nachrichtenlänge sind in Kapitel 4.4 definiert.

Ressourcen auf einem Mikrocontroller nicht möglich. Somit gilt es in der folgenden Diskussion die für dieses Projekt geeigneten Verfahren auszuwählen.

4.3.2 Content-Length

Bei diesem Verfahren wird zunächst der gesamte Body der Response auf dem Webserver erzeugt. Anschließend kann die Nachrichtenlänge durch einen geeigneten Algorithmus bestimmt werden. Im Header-Bereich der Nachricht ist der ermittelte Wert in dem Feld mit der Bezeichnung *Content-Length* eingetragen. In dem unten aufgeführten Beispiel fordert der Client wieder die Seite *index.html* auf dem Host *www.beispiel.de* an. Die Antwort des Webservers besteht aus einem Header und dem Body, welcher die Nutzdaten (hier HTML-Code) enthält.

- **Request**

```
GET /index.html HTTP/1.1
Host: www.beispiel.de
```

Listing 4-3: Content-Length, Request

- **Response:**

```
HTTP/1.1 200 OK
Server: 8051_Webserver
Content-Length: <Länge des Bodys>
Content-Type: text/html

<Nachrichten-Body mit HTML-Quellcode>
```

Listing 4-4: Content-Length, Response

- **Vorteile**

Ein großer Vorteil dieser Methode ist, dass sie eines sehr geringen Implementierungsaufwands bedarf. Der Server kann die gesamte Nachricht erzeugen, die Länge des Bodys bestimmen und im Anschluss in einem Stück versenden.

- **Nachteile**

Um die Länge der gesamten Nachricht ermitteln zu können, muss diese im Speicher des Webservers vorgehalten werden. Für Webserver die auf speicherarmen Hardwarearchitekturen aufgebaut sind, stellt diese Speicherung einer gesamten Nachricht (dies könnte zum Beispiel der Quellcode einer größeren Homepage sein) ein immenses Problem dar.

4.3.3 Chunked Transfer-Encoding

Das Verfahren des Chunked Transfer-Encodings („blockweise Datenübertragung“) ermöglicht es, die Nachricht in kleinere Blöcke zu unterteilen und nacheinander zu versenden. Die Länge eines Blocks wird dabei als Hexadezimalwert vor den jeweiligen Block gesetzt.

- **Request**

```
GET /index.html HTTP/1.1
Host: www.beispiel.de
```

Listing 4-5: Chunked Transfer-Encoding, Request

- **Response[8]:**

```
HTTP/1.1 200 OK
Server: 8051_Webserver
Transfer-Encoding: chunked
Content-Type: text/html

0x0EE1
<Die ersten 0x0EE1 Zeichen des Nachrichten-Bodys>

0x0FFB
<Weitere 0x0FFB Zeichen des Nachrichten-Bodys>

0
<!-- Eine 0 gefolgt von einer Leerzeile terminiert die Nachricht -->
```

Listing 4-6: Chunked Transfer-Encoding, Response

- **Vorteile**

Besonders auf Webservern mit begrenzten Speicherressourcen hat diese Methode Vorteile. Sie ermöglicht es, die Nachricht blockweise zu generieren, um die einzelnen Blöcke umgehend an den Client-Rechner zu versenden. Im Gegensatz zu der Methode Content-Length (siehe Kapitel 4.3.2) ist es nicht nötig, die gesamte Nachricht im Speicher des Webservers vorzuhalten.

- **Nachteile**

Der Implementierungsaufwand dieser Methode ist deutlich höher als es bei anderen Methoden der Fall ist. Man muss geeignete Algorithmen entwickeln, die das Erstellen der Nachricht an geeigneten Stellen unterbrechen, die Länge des entstandenen Blocks berechnen um diesen anschließend an den Client-Rechner zu versenden. Ein weiterer Nachteil ist der hierdurch größer werdende Netzwerkverkehr: Jeder Block

[8] Kommentare in HTML-Code haben die folgende Syntax: <!-- Kommentar -->.

wird separat versendet und erhält zusätzlich zu den Nutzdaten weitere netzwerkprotokollspezifische Header-Daten.

4.3.4 Multipart / Byteranges

Eine weitere Möglichkeit, eine große Nachricht in mehreren Blöcken zu versenden, bietet das Verfahren Multipart / Byteranges. Um dem Client zu signalisieren, dass eine Multipart-Nachricht versendet wurde, ist im Header der Response im Feld *Content-Type* das Attribut *multipart/byteranges* anzugeben.
Der Nachrichten-Body setzt sich dann aus mehreren Blöcken zusammen, die jeweils einen eigenen Header bestehend aus den Feldern *Content-Type* und *Content-Range* enthalten. Die einzelnen Blöcke sind durch einen sogenannten Seperator-String getrennt, der ebenfalls im Header definiert wird. Die Länge der Blöcke ist in folgender Syntax anzugeben.

```
1000-1999/2000
<Anfangsbyte> - <Endbyte> / <Gesamtlänge>
```

Listing 4-7: Syntax der Längenangabe bei der Methode Multipart / Byteranges

- **Request**

```
GET /index.html HTTP/1.1
Host: www.beispiel.de
```

Listing 4-8: Multipart / Byteranges, Request

- **Response**

```
HTTP/1.1 206 Partial content
Server: 8051_Webserver
Content-type: multipart/byteranges; boundary=SEPERATOR

SEPERATOR <!-- SEPERATOR trennt die einzelnen Blöcke -->
Content-Type: text/html
Content-Range: bytes 0-999/2000
<Byte 0 bis 999>
SEPERATOR
Content-Type: text/html
Content-Range: bytes 1000-1999/2000
<Byte 1000 bis 1999>
SEPERATOR
```

Listing 4-9: Multipart / Byteranges, Response

- **Vorteile**

Ähnlich dem Verfahren des Chunked Transfer-Encodings (siehe Kapitel 4.3.3) kann die Nachricht in mehreren Blöcken generiert und versendet werden. Außerdem muss

bei dieser Methode nicht immer der gesamte Inhalt einer Nachricht versendet werden. Der Client-Rechner hat die Möglichkeit nur bestimmte Bereiche (Byte-Bereiche) anzufordern.

- **Nachteile**

Die Verwendung dieses Verfahrens setzt implizit voraus, dass die Gesamtlänge der Nachricht bereits vor dem Versenden des ersten Blocks bekannt ist. Dies führt zurück zu dem Ausgangsproblem, dass die gesamte Nachricht im Speicher des Webservers (hier Mikrocontroller) vorgehalten werden muss. Nur so ist die Gesamtlänge der Nachricht zu ermitteln. Außerdem ist der Implementierungsaufwand dieses Verfahrens sehr hoch.

4.3.5 Trennung der Verbindung durch den Webserver

Nachdem der Webserver seine Daten vollständig übertragen hat, trennt er bei diesem Verfahren die Verbindung zum Client-Rechner. Der Client setzt die bis zur Trennung der Verbindung empfangenen Daten zusammen und erhält somit die gesamte Nachricht. Die Nachricht enthält keine Angabe über ihre Länge.

- **Request**

```
GET /index.html HTTP/1.1
Host: www.beispiel.de
```

Listing 4-10: Trennung der Verbindung durch den Webserver, Request

- **Response:**

```
HTTP/1.1 200 OK
Server: 8051_Webserver
Content-Type: text/html

<Body mit HTML-Quellcode>
<!-- Trennung der Verbindung durch den Server -->
```

Listing 4-11: Trennung der Verbindung durch den Webserver, Response

- **Vorteile**

Der sehr geringe Implementierungsaufwand stellt einen großen Vorteil dieser Methode dar. Nachdem eine Nachricht vollständig versendet worden ist, kann die Verbindung einfach serverseitig getrennt werden. Es sind keine Algorithmen zur Längenbestimmung notwendig.

- **Nachteile**

Dieses Verfahren bietet keinerlei Sicherheit bei der Übertragung einer Nachricht. Trennt der Webserver durch einen Fehlzustand die Verbindung, obwohl noch nicht alle Daten erzeugt und zugestellt wurden, interpretiert der Client-Rechner die empfangenen Daten möglicherweise falsch.

4.3.6 Nachrichten ohne Nachrichten-Body

Dieser Typ einer Response stellt in dem hier diskutierten Kontext einen Sonderfall dar. Besteht eine Response ausschließlich aus Header-Daten, muss die Nachrichtenlänge weder berechnet noch übermittelt werden. Dies ist der Fall, wenn der Client-Rechner durch einen HEAD-Request (siehe Kapitel 4.4.3) signalisiert, dass er nur Header-Informationen benötigt. Dieser kann beispielsweise verwendet werden um nur den Servernamen abzufragen.

- **Request**

```
HEAD /Webseite/ HTTP/1.1
Host: www.beispiel.de
```

Listing 4-12: Nachrichten ohne Nachrichten-Body, Request

- **Response:**

```
HTTP/1.1 200 OK
Server: 8051_Webserver
Content-Type: text/html
<!-- Eine Leerzeile terminiert den Header -->
```

Listing 4-13: Nachrichten ohne Nachrichten-Body, Response

- **Vorteile**

Vorteil dieser Methode ist die Reduzierung des Datentransfers, wenn ein Client-Rechner nur Header-Informationen anfordert. Würde der Server dieses Verfahren nicht unterstützen, müsste bei jedem HEAD-Request der mitunter sehr große Body mit gesendet werden.

- **Nachteile**

Das Verfahren wird ausschließlich angewendet um HEAD-Requests zu verarbeiten. Diese werden in dem Einsatzbereich des hier entwickelten Webservers jedoch sehr selten verwendet.

4.3.7 Diskussion der Verfahren

Bei Gegenüberstellung der vorgestellten Verfahren wird sehr schnell deutlich, dass einige der Verfahren für eine Anwendung in diesem Projekt nicht geeignet sind. Hierunter fallen die Verfahren „*Content-Length*“ und „Multipart / *Byteranges*“. Beide Verfahren sind auf einem Webserver auf Basis eines Mikrocontrollers mit begrenzten Speicherressourcen nicht zu realisieren. Die Notwendigkeit, die Länge der gesamten Nachricht vor ihrer Versendung zu kennen, macht diese Verfahren nicht praktikabel auf der verwendeten Hardware.
Von der Anwendung des Verfahrens „*Trennung der Verbindung durch den Webserver*“ wird ebenfalls abgesehen, da es keinerlei Sicherheit bei der Übertragung der Nachricht bietet. Eine fehlerhaft dargestellte Webseite, die zum Beispiel Prozessdaten einer Anlage visualisiert, kann zu Fehleinschätzungen des Prozesses führen.
Das Verfahren „Nachrichten *ohne Nachrichten-Body*“ ist hier ebenfalls nicht interessant, weil hierbei nur HEAD-Requests beantwortet werden können. Wie in Kapitel 4.4 ersichtlich wird, ist in diesem Projekt auf die Implementierung einer Funktionalität zur Verarbeitung von HEAD-Requests verzichtet worden. Das Verfahren hat somit hier keine Relevanz.

Zur Übermittlung der Länge einer Response-Nachricht wird das Verfahren *Chunked Transfer-Encoding* gewählt. Es bietet Sicherheit bei der Datenübertragung, da jeder zu übertragende Block mit einer Längenangabe versehen ist. Des Weiteren stellt es eine flexible Lösung dar, auch speicherintensive Webseiten auf einer Hardware mit begrenzten Speicherressourcen erzeugen zu können. Dies wird erreicht, indem eine große Nachricht zerlegt und blockweise versendet wird.

4.4 HTTP-Request-Methode

Es gibt verschiedene standardisierte Methoden, wie ein Client-Rechner seine Anfrage an einen Webserver richten kann. Diese Methoden werden als „*Request-Methoden*“ bezeichnet und sind in RFC2616 definiert [4]. Unter Verwendung der verschiedenen Methoden, hat der Client-Rechner zum Beispiel die Möglichkeit, Inhalte vom Webserver anzufordern, Daten auf dem Webserver zu verändern oder Statusinformationen über die Verbindung zu erhalten. In Tabelle 4-1 sind die am weitesten verbreiteten Methoden aufgeführt.
Aufgrund der begrenzten Ressourcen, die auf einem Mikrocontroller zur Verfügung stehen, ist es nicht möglich alle aufgeführten Methoden zu implementieren. Der Bedarf hierzu besteht in diesem Projekt jedoch auch nicht.
Daher gilt es im Folgenden, die für dieses Projekt sinnvollen Request-Methoden zu ermitteln. Hierzu sind die in Tabelle 4-1 aufgeführten Varianten in den Kapiteln 4.4.1 bis 4.4.6 erläutert und abschließend diskutiert.

Tabelle 4-1: Standardisierte Request-Methoden

Methode	*Kurzbeschreibung*
GET	Inhalte vom Webserver anfordern.
POST	Inhalte vom Webserver anfordern.
HEAD	Anfordern eines Nachrichten-Headers.
PUT	Daten auf einem Webserver anlegen und verändern.
DELETE	Daten auf einem Webserver löschen.
TRACE	Überprüfen der Verbindung.

4.4.1 GET-Request

Ein GET-Request stellt die am weitesten verbreitete Methode dar, um Inhalte von einem Webserver anzufordern. Parameter, die dem Webserver mit der Anfrage übergeben werden sollen, können einfach an den *Uniform Ressource Locator* (URL) der Webseite angehangen werden. Hierbei finden Datenpaare mit folgender Syntax häufige Verwendung.

```
Parameter_Name_1=Parameter_Wert_1
Parameter_Name_2=Parameter_Wert_2
...
```

Listing 4-14: Syntax der Datenpaare

Diese Datenpaare sind deshalb so gebräuchlich, weil sich viele Aktionen auf einer Webseite auf diese Weise beschreiben lassen. Soll ein Webseitenbesucher zum Beispiel seinen Namen über ein Eingabefeld an den Webserver senden, erhält der Webserver als Resultat ein Datenpaar bestehend aus den in Listing 4-15 aufgeführten Daten.

```
<Name_des_Eingabefelds>=<Name_des_Besuchers>
```

Listing 4-15: Datenpaar bei der Übermittlung der Daten eines Eingabefelds

Der Webserver könnte nach Auswertung des Datenpaars zum Beispiel den privaten Bereich des Benutzers anzeigen (aus Sicherheitsgründen wäre hierzu zusätzlich die Abfrage eines Passwortes zu empfehlen). Ähnlich sieht das Datenpaar einer Anfrage bei den Aktionen *Aktivieren einer Auswahlbo*x, *Markieren eines Eintrages einer Listbox* oder *Drücken einer Schaltfläche* aus.
Um mehrere Datenpaare in einer Anfrage zu übermitteln, müssen diese durch ein Kaufmannsund („&“) voneinander getrennt werden. Die Paare sind durch ein

Fragezeichen („?“) nach der Angabe der Webseite eingeleitet. Das Ende des Parameterfeldes wird dem Webserver durch ein Leerzeichen signalisiert.

- **Request**

Die in Listing 4-16 dargestellte Nachricht würde ein Webserver bei einer GET-Anfrage eines Clients erhalten. Hier wird die Seite „*Webseite*“ auf dem Host „*www.beispiel.de*“ angefragt. Teil dieser Anfrage sind die Datenpaare „*Feld_1=Max*“ und „*Feld_2=Meier*“, die in diesem Beispiel Vorname und Nachname eines Benutzers darstellen sollen.

```
GET /Webseite/?Feld_1=Max&Feld_2=Meier HTTP/1.1
Host: www.beispiel.de
```

Listing 4-16: GET-Request

Ein Blick auf die Adresszeile eines Webbrowsers (siehe Bild 4-2) der einen GET-Request abgesendet hat, zeigt wie die Datenpaare übermittelt werden. Sie sind einfach an die URL der angewählten Homepage angehängt. Die entstehende Zeichenkette bestehend aus der Adresse der Webseite und den angehängten Datenpaaren wird mit *Uniform Ressource Identifier* (URI) bezeichnet.

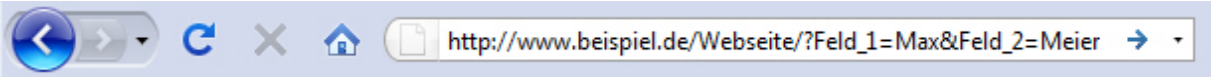

Bild 4-2: GET-Request in der Adresszeile eines Webbrowsers

- **Vorteile**

Die Implementierung dieser Methode ist auch auf Webservern mit begrenzten Ressourcen sehr gut realisierbar. Alle Informationen, die ein Client an den Webserver überträgt, befinden sich in dem URI. Um den URI in seine Einzelkomponenten zu zerlegen und die übermittelten Parameter zu erhalten, müssen nur einige Algorithmen zur Analyse von Zeichenketten programmiert werden.

- **Nachteile**

Ein großer Nachteil dieser Methode, ist die fehlende Sicherheit bei der Behandlung von sensiblen Daten. Hierzu ist folgendes in RFC2616 nachzulesen:

> *Authors of services which use the HTTP protocol SHOULD NOT use GET based forms for the submission of sensitive data, because this will cause this data to be encoded in the Request-URI. Many existing servers, proxies, and user agents will log the request URI in some place where it might be visible to third parties.*

Des Weiteren ist die maximale Länge eines URI-Strings auf 255 Bytes begrenzt. Möchte man Daten aus einem Textfeld übermitteln, ist diese Grenze sehr schnell überschritten.

4.4.2 POST-Request

Ähnlich der GET-Methode (siehe Kapitel 4.4.1) dient der POST-Request ebenfalls dazu, Inhalte von einem Webserver anzufordern. Auch der POST-Request kann die bereits angesprochenen Datenpaare mit seiner Anfrage übermitteln. Der große Unterschied liegt hierbei in der Syntax der Anfrage. Zur Veranschaulichung ist im Folgenden ein POST-Request aufgeführt, der wie im Beispiel des GET-Requests, den Vor- und Nachnamen eines Benutzers an den Webserver überträgt.

- **Request**

```
POST /Webseite/ HTTP/1.1
Host: www.beispiel.de
Content-Length: 23

Feld_1=Max&Feld_2=Meier <!-- Datenpaare im Body der Nachricht -->
```

Listing 4-17: POST-Request

Ein Blick auf die Adresszeile eines Webbrowsers (siehe Bild 4-3) zeigt, dass die übermittelten Datenpaare hier nicht erscheinen. Sie werden im Body der Nachricht an den Webserver übermittelt. Zusätzlich erhält der Header der Request-Nachricht ein Feld mit der Bezeichnung „*Content-Length*“, in dem die Länge der Datenpaarzeichenkette eingetragen ist.

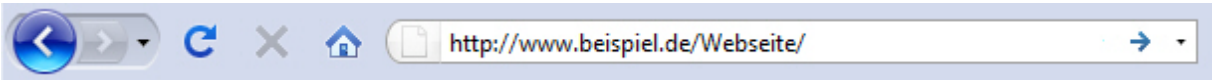

Bild 4-3: POST-Request in der Adresszeile eines Webbrowsers

- **Vorteile**

Diese Methode erlaubt im Gegensatz zu einem GET-Request den Umgang mit sensiblen Daten. Da die zu übermittelnden Datenpaare nicht in der URI sondern separat im Nachrichtenbody übermittelt werden, ist die Gefahr geringer, dass Dritte an diese Daten herankommen.

- **Nachteile**

Diese Methode weist keine signifikanten Nachteile auf. Der Implementierungsaufwand ist ähnlich hoch wie bei einem GET-Request. Die Umsetzung auf einem Mikrocontroller ist gut realisierbar.

4.4.3 HEAD-Request

Ein HEAD-Request wird verwendet, um ausschließlich Header-Informationen bei einem Webserver abzufragen. Der Webserver interpretiert diesen Request ähnlich einem GET-Request (siehe Kapitel 4.4.1), seine Antwort beinhaltet jedoch nur den Header, nicht den Nachrichten-Body. Ein Browser kann mit Hilfe dieser Methode überprüfen, ob eine Webseite oder eine Datei auf dem Webserver seit der letzten Anforderung modifiziert wurde. Ist dies nicht der Fall, zeigt der Browser die im Browser-Cache vorgehaltenen Daten an, ohne den Nachrichten-Body erneut anfordern zu müssen. Im Folgenden sind ein Head-Request und eine darauffolgende Antwort eines Webservers aufgeführt.

- **Request**

```
HEAD /Webseite/ HTTP/1.1
Host: www.beispiel.de
```

Listing 4-18: HEAD-Request

- **Response**

```
HTTP/1.1 200 OK
Server: 8051_Webserver
Content-Length: 159
Content-Type: text/html
```

Listing 4-19: Response auf einen HEAD-Request

- **Vorteile**

Vorteil dieser Methode ist die Reduzierung des Datentransfers, wenn ein Client-Rechner nur Header-Informationen anfordert.

- **Nachteile**

Die Funktionalität dieser Methode kann auch über einen GET-Request realisiert werden. Hierzu wird ein solcher an den Webserver gesendet. Aus der Antwortnachricht werden dann nur die Header-Informationen extrahiert.
Somit stellt der zusätzliche Implementierungsaufwand dieser Methode, besonders auf Webservern mit begrenzten Ressourcen, einen Nachteil dar.

4.4.4 PUT-Request

Ein PUT-Request weist den Webserver an, die im Body der Nachricht übermittelten Daten unter einer angegebenen Adresse abzuspeichern. Es ist somit möglich, neue Datenquellen auf dem Webserver zu erzeugen oder bestehende zu verändern. In

dem gezeigten Beispiel wird eine neue Webseite */Webseite/Neue_Seite/* auf dem Host *www.beispiel.de* erzeugt.

- **Request**

```
PUT /Webseite/Neue_Seite/ HTTP/1.1
Host: www.beispiel.de

<html>
     <head>
          <title>Titel der Webseite</title>
     </head>
     <body>
          Text auf der Webseite
     </body>
</html>
```

Listing 4-20: PUT-Request

- **Vorteile**

Mit Hilfe dieser Methode können direkt neue Datenquellen auf einem Webserver erzeugt werden. Um dies mit anderen Request-Methoden zu realisieren, sind aufwendige Programmstrukturen nötig.

- **Nachteile**

Um diese Methode sicher auf einem Webserver anwenden zu können, ist ein sehr hoher Programmieraufwand nötig. Da es mittels dieser Methode möglich ist Daten auf dem Webserver zu verändern, müssen alle Anfragen des Clients serverseitig überwacht und überprüft werden. Anderenfalls können korrupte Daten in einem PUT-Request zu Fehlfunktionen des Webservers führen. Die Methode bietet große Gefahren und ist daher selbst auf vielen großen Webservern deaktiviert.

4.4.5 DELETE-Request

Diese Methode bietet die entgegengesetzte Funktionalität zu einem PUT-Request. Der DELETE-Request weist den Webserver an die Daten an der angegebenen Adresse zu löschen. In dem gezeigten Beispiel wird die zuvor mit der PUT-Methode erzeugte Webseite */Webseite/Neue_Seite/* auf dem Host *www.beispiel.de* wieder gelöscht.

- **Request**

```
DELETE /Webseite/Neue_Seite/ HTTP/1.1
Host: www.beispiel.de
```

Listing 4-21: DELETE-Request

- **Vorteile**

Mit Hilfe dieser Methode können sehr schnell Datenquellen auf einem Webserver entfernt werden. Um dies mit anderen Request-Methoden zu realisieren, sind aufwendige Programmstrukturen nötig.

- **Nachteile**

Ähnlich dem PUT-Request, birgt auch diese Methode große Gefahren. Da sie eine generelle Möglichkeit bietet, Inhalte auf einem Webserver zu löschen, ist bei unbewusster Fehlanwendung die Beschädigung des Webservers nicht auszuschließen. Um dies zu vermeiden, sind auch hier umfangreiche Sicherheitsalgorithmen nötig. Aufgrund dieser Gefahren ist die Methode auf den meisten Webservern deaktiviert.

4.4.6 TRACE-Request

Ein TRACE-Request wird vom Webserver so an den Client-Rechner zurückgesendet, wie er empfangen wurde. So kann überprüft werden, ob und wie die Anfrage auf dem Weg zum Webserver verändert worden ist. Diese Methode ist sinnvoll um Fehler bei der Übertragung ausfindig zu machen. Das unten aufgeführte Beispiel zeigt einen TRACE-Request und die vom Webserver folgende Antwort im Falle einer fehlerfreien Übertragung.

- **Request**

```
TRACE /Webseite/Neue_Seite/ HTTP/1.1
Host: www.beispiel.de

<Nachrichtentext>
```

Listing 4-22: TRACE-Request

- **Response**

```
HTTP/1.1 200 OK
Server: 8051_Webserver
Content-Length: 74
Content-Type: message/http

TRACE /Webseite/Neue_Seite/ HTTP/1.1
Host: www.beispiel.de

<Nachrichtentext>
```

Listing 4-23: Response auf einen TRACE-Request

- **Vorteile**

Der TRACE-Request ermöglicht das Finden von Fehlern bei der Datenübertragung. Ist die Antwort eines Webserver auf eine TRACE-Anfrage nicht identisch mit der eigentlichen Anfrage, sind Daten bei der Übertragung verändert worden.

- **Nachteile**

Diese Methode dient ausschließlich zur Verbindungsüberprüfung und hat für den allgemeinen Betrieb des Webservers nur geringe Bedeutung. Der zusätzliche Implementierungsaufwand und Speicherbedarf ist somit als Nachteil anzusehen.

4.4.7 Diskussion der Methoden

Die Methoden *PUT* und *DELETE* finden in diesem Projekt keine Anwendung. Für den geforderten Funktionsumfang ist es nicht notwendig, dass ein Anwender direkt Daten auf dem Webserver verändern oder löschen kann. Außerdem ist die Realisierung dieser Methoden und der dazugehörigen Sicherheitsalgorithmen auf einem Mikrocontroller nur schwer möglich.
Wie in Kapitel 4.4.3 beschrieben ist, stellt ein HEAD-Request eine abgewandelte Form eines GET-Requests dar. Als Vorteil ist dabei der verminderte Datentransfer zu nennen, da der Webserver nur Headerinformationen zurücksendet. Bei der Beurteilung der Tauglichkeit dieser Methode für das Projekt überwiegt jedoch der Nachteil des Speicherbedarfs für die Programmierung einer weiteren Methode. Da die Funktion des HEAD-Requests auch durch einen GET-Request nachgebildet werden kann und bei kleineren Webseiten selten zum Einsatz kommt, wird diese Methode hier nicht berücksichtigt.
Von der Verwendung des TRACE-Requests wird aus ähnlichen Gründen abgesehen. Diese Methode dient dem Finden von Fehlern in der Datenübertragung. Sie stellt dem Anwender eine nützliche Funktion zur Fehlersuche dar, auf die in diesem Projekt aus Gründen der geringen Speicherressourcen jedoch verzichtet wird.

In diesem Projekt wird eine Funktionalität zur Verarbeitung von *GET*- und *POST-Requests* implementiert. Die beiden Methoden stellen die idealen Lösungen dar, um die an das Webserver-Modul gestellten Anforderungen umzusetzen. Der Webserver soll über die Funktionalität verfügen, HTML-Code in Abhängigkeit der vom Client-Rechner gesendeten Anfragen dynamisch zu erzeugen. Der Funktionsumfang, den dieses Modul dem Anwender bieten soll, umfasst zum Beispiel das Setzen von Prozessparametern, das Auslesen eines Eventlogs oder die Übermittlung von Messdaten. Diese Aufgaben sind hervorragend über GET- und POST-Requests abzudecken. Bei den geschilderten Fällen muss der Anwender die Möglichkeit haben, dem Webserver eine Anfrage inklusive einiger Parameter zu senden. Dies ist

mit den beiden Request-Methoden sehr gut realisierbar. Für gängige Anfragen kann ein GET-Request eingesetzt werden. Handelt es sich bei der Übertragung um sensible Daten oder muss dem Webserver eine große Anzahl an Parametern übermittelt werden, kommt der POST-Request zum Einsatz.

5 Beschreibung der durchgeführten Arbeiten

5.1 Entwicklung und Aufbau des Testboards

Wie in zuvor erläutert wurde, sind das generelle Hardware-Design und die Wahl der Hauptkomponenten, wie Ethernet-Interface-Chip und Webserver-Chip, im Rahmen dieser Arbeit festgelegt worden. Die eigentliche Schaltungsentwicklung und -implementierung ist jedoch nicht Teil der Diplomarbeit. Vielmehr stand das fertige Testboard inklusive der dazugehörigen Dokumentation bereits zur Verfügung.

5.2 Kommunikation der Chips über den Daten- / Adressbus

Zur Kommunikation zwischen dem 8051F120 und dem W5300 wird das sogenannte *External Data Memory Interface* verwendet. Dies ist eine im Mikrocontroller implementierte Hardware, die es ermöglicht, Register externer Geräte in den Speicherbereich des Mikrocontrollers einzubinden. Dieser Vorgang wird als „*Memory-Mapping*“ bezeichnet. Der 8051F120 und das externe Gerät (hier W5300) sind hierzu über Adress-, Daten-, und Steuerleitungen miteinander verbunden (siehe Bild 5-1). Der Programmierer kann die Register des externen Geräts verwenden, als seien es Register des Mikrocontrollers. Die Kommunikation zwischen den Geräten wird hierbei von der im Mikrocontroller implementierten Hardware realisiert.

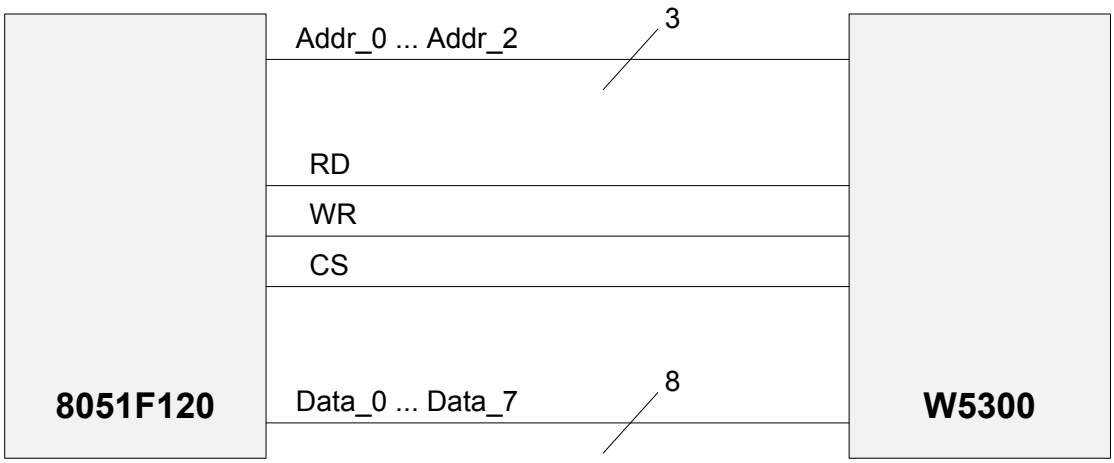

Bild 5-1: Kommunikation zwischen 8051F120 und W5300

5.3 Das Kommunikationsmodul

Der Ethernet-Interface-Chip W5300 der Firma Wiznet dient in diesem Projekt als Kommunikationsmodul des Webservers (siehe Kapitel 4.1.3). Seine Aufgabe ist die Netzwerkkommunikation zwischen Webserver und Client-Rechner.

Bild 5-2 zeigt das Blockschaltbild des Chips [6]. Die grundlegenden Leistungsmerkmale sind im Folgenden näher beschrieben. Für weitere Details kann das vom Hersteller angebotene Handbuch herangezogen werden [6].

- **Integrierter TCP/IP-Core**

Ein wesentlicher Bestandteil dieses Chips und ein Hauptgrund für seine Wahl ist der integrierte TCP/IP-Core. Dieser Core ermöglicht dem W5300 die sichere Kommunikation im Netzwerk über die Protokolle der TCP/IP-Familie. Die Protokolle sind mit allen protokollspezifischen Kontrollmechanismen vollständig im W5300 implementiert. Nach erfolgreicher Initialisierung bietet der W5300 dem Anwender eine komfortable Schnittstelle für die Kommunikation im Netzwerk.

- **Acht Verbindungs-Sockets**

Der Chip bietet weiterhin die Möglichkeit bis zu acht TCP-Verbindungen, sogenannte Socket-Verbindungen, mit Client-Rechnern aufzubauen. Dies ermöglicht die quasi parallele Verarbeitung mehrerer Anfragen verschiedener Client-Rechner. Vor allem für Webserveranwendungen ist dies von Vorteil, da mitunter mehrere Client-Rechner gleichzeitig Anfragen an den Webserver richten.

- **Integrierter Sende- und Empfangsspeicher**

Für die Zwischenspeicherung der empfangenen und der zu sendenden Daten hat der W5300 einen 128 kByte großen Sende- und Empfangsspeicher. Dieser Speicher ist als PRAM ausgeführt und bietet die Funktionalität eines FIFO-Speichers. Hierbei wird die gesamte Verwaltung des Speichers vom Chip übernommen. Der Anwender schreibt bei einem Sendevorgang Daten in den Sende-FIFO und veranlasst den Chip diese Daten zu versenden. Beim Datenempfang fragt er beim Chip an, wie viele Daten im Empfangs-FIFO vorhanden sind und ruft diese im Anschluss ab. Der genaue Sende- und Empfangsvorgang ist in Kapitel 5.3.5 detailliert erläutert.

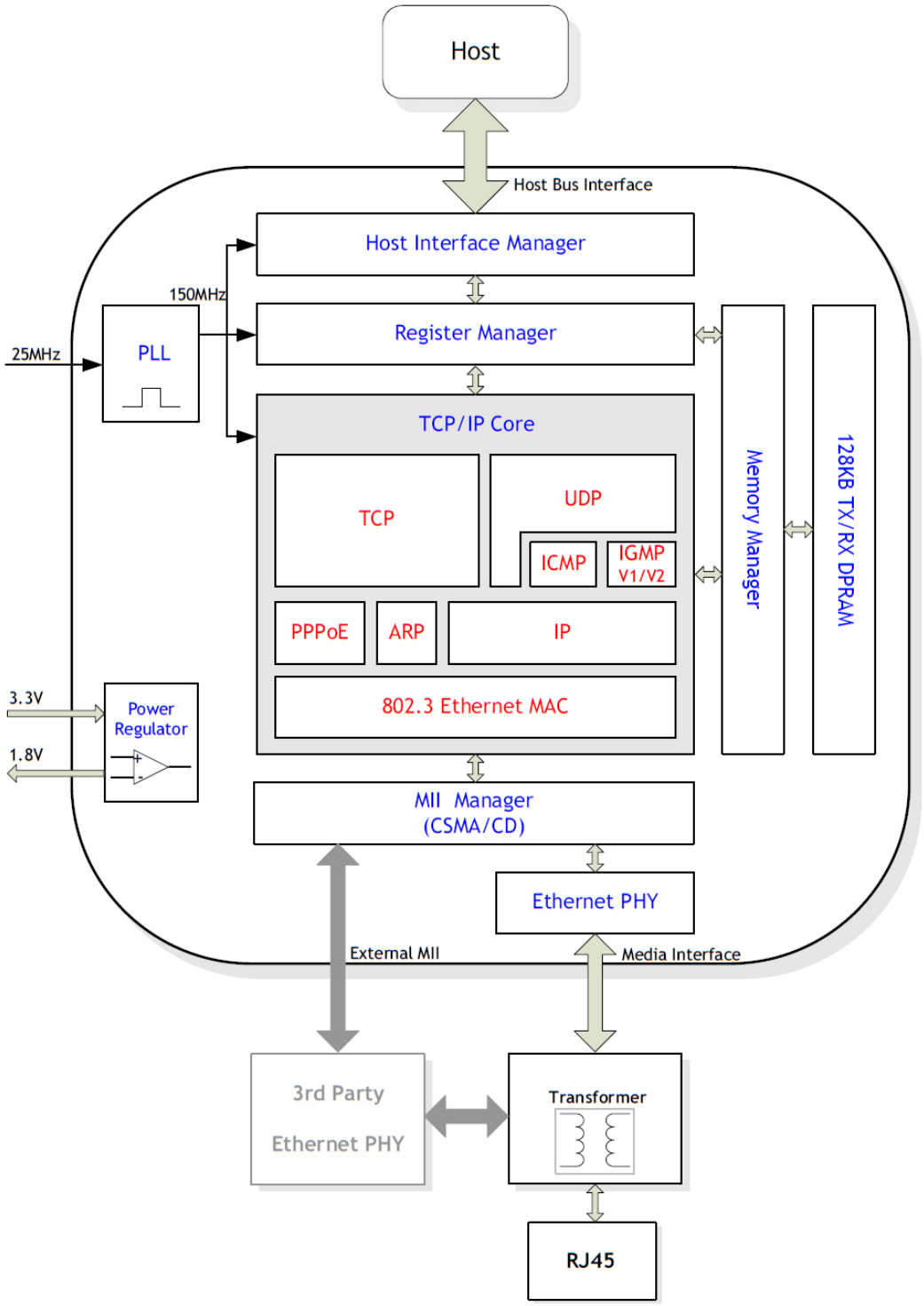

Bild 5-2: Blockschaltbild des Ethernet-Interface-Chips W5300

5.3.1 Register des W5300

Die Register des W5300 sind in vier Gruppen unterteilt. Jede dieser Registergruppen steht für eine bestimmte Funktionalität des Chips. Die einzelnen Gruppen sind im Folgenden aufgeführt und erläutert.

- **Mode-Register**

In diesem Register werden Einstellungen bezüglich der Betriebsart des W5300 vorgenommen. Hier ist zum Beispiel die Adressierungsmethode (siehe Kapitel 5.3.2) oder die Datenbusbreite einzustellen (der Chip kann an einem 8-bit oder 16-bit breiten Datenbus betrieben werden). Weiterhin kann der Chip über dieses Register in einen Reset-Zustand versetzt werden, was für den Initialisierungsvorgang von Bedeutung ist.

- **Common-Register**

In den Common-Registern werden grundlegende Einstellungen wie zum Beispiel MAC-Adresse, IP-Adresse, Standard-Gateway oder Subnetzmaske festgelegt. Diese Register sind in der Regel einmalig einzustellen (zum Beispiel beim Programmstart) und bedürfen während des Betriebs nur selten einer Änderung.

- **Socket-Register**

Socket-Register beinhalten verbindungsbezogene Daten und Parameter. Wie zuvor bereits erwähnt, kann der W5300 acht voneinander getrennte Verbindungen mit Client-Rechnern aufbauen. Für jede dieser Verbindungen stehen separate Socket-Register zur Verfügung. Hierüber wird der Sende- und Empfangsprozess gesteuert und die Verbindung mit dem Client-Rechner verwaltet.

- **Indirect-Mode-Register**

Wird der W5300 indirekt adressiert, sind oben genannte Socket- und Common-Register ausschließlich über das Indirect-Mode-Address- und das Indirect-Mode-Data-Register zu erreichen. Der Adressierungsvorgang ist im folgenden Kapitel detailliert erläutert.

5.3.2 Adressierung des W5300

Die Kommunikation zwischen Webserver-Chip und W5300 kann mittels *direkter* oder *indirekter Adressierung* erfolgen. Dies betrifft die Adressierung der Register des W5300, wenn sie vom Webserver-Chip gesetzt oder ausgelesen werden. Die beiden Methoden sind im Folgenden näher erklärt.

- **Direkte Adressierung**

Bei Verwendung der direkten Adressierung wird dem W5300 über zehn Adressleitungen direkt eine gewünschte Registeradresse angelegt. Danach kann über die Datenleitungen der Wert in dem adressierten Register gelesen oder geschrieben werden. Man benötigt somit zehn Leitungen zur Adressierung und

weitere 8-16 Leitungen für den Datenbus (abhängig davon ob ein 8-bit oder ein 16-bit Datenbus gewählt ist).

- **Indirekte Adressierung**

Die Methode der indirekten Adressierung benötigt nur drei Adressleitungen. Über diese wird zunächst das *Indirect-Mode-Address-Register* angewählt. Dort kann über die Datenleitungen die Adresse des gewünschten Registers (Common- oder Socket-Register) eingetragen werden. Dann ist das *Indirect-Mode-Data-Register* zu adressieren. Über die Datenleitungen kann nun der Wert des angeforderten Registers gelesen oder geschrieben werden. Vorteil dieser Methode ist das Einsparen von sieben Adressleitungen. Nachteilig wirkt sich die Tatsache aus, dass jede Aktion die doppelte Anzahl an Befehlen benötigt.

5.3.3 Setzen oder Auslesen eines Registers des W5300

Die gesamte Steuerung des Ethernet-Interface-Chips basiert auf Registerzugriffen. Unabhängig davon ob die Webserveranwendung Daten an den W5300 sendet oder von ihm abruft, geschieht dies immer über den Zugriff auf ein Register des W5300. Hierzu wird das Register zuerst adressiert, um anschließend einen Wert auszulesen oder zu setzen. Ein typischer Registerzugriff ist in Listing 5-1 aufgeführt. Hier wird ein Wert in das Sende-Register von Socket 0 geschrieben. Dies ist über die Methode der indirekten Adressierung realisiert, da sie auch in diesem Projekt verwendet wird.

```
////////////////////////////////////////////////////
// Setzen des Sende-Registers von Socket_0 im W5300 //
////////////////////////////////////////////////////

// Anwahl des Indirect-Mode-Address-Registers
W5300_ADDRESS = INDIRECT_MODE_ADDRESS_REGISTER;

// Schreiben der Adresse des Sende-Registers von Socket_0
// in das Indirect-Mode-Address-Register
W5300_DATA = SENDE_REGISTER_SOCKET_0;

// Anwahl des Indirect-Mode-Data-Registers
W5300_ADDRESS = INDIRECT_MODE_DATA_REGISTER;

// Schreiben eines Werts für das Sende-Register von Socket_0
// in das Indirect-Mode-Data-Register
W5300_DATA = 0xFE35;
```

Listing 5-1: Registerzugriff bei indirekter Adressierung

5.3.4 Initialisierung des W5300

Damit der Ethernet-Interface-Chip seine Arbeit als Kommunikationsmodul des Webservers aufnehmen kann, ist er beim Programmstart zu initialisieren. Dies

geschieht in zwei aufeinanderfolgenden Schritten: *Basisinitialisierung* und *Initialisierung der Verbindungs-Sockets*.

- **Basisinitialisierung**

Während der Basisinitialisierung werden einige grundlegende Parameter in den *Mode-* und *Common-Registern* gesetzt, die nicht socket-spezifisch sind und damit als globale Einstellungen bezeichnet werden können. In Bild 5-3 ist der Ablauf der Basisinitialisierung anhand eines Flussdiagramms dargestellt und beschrieben.

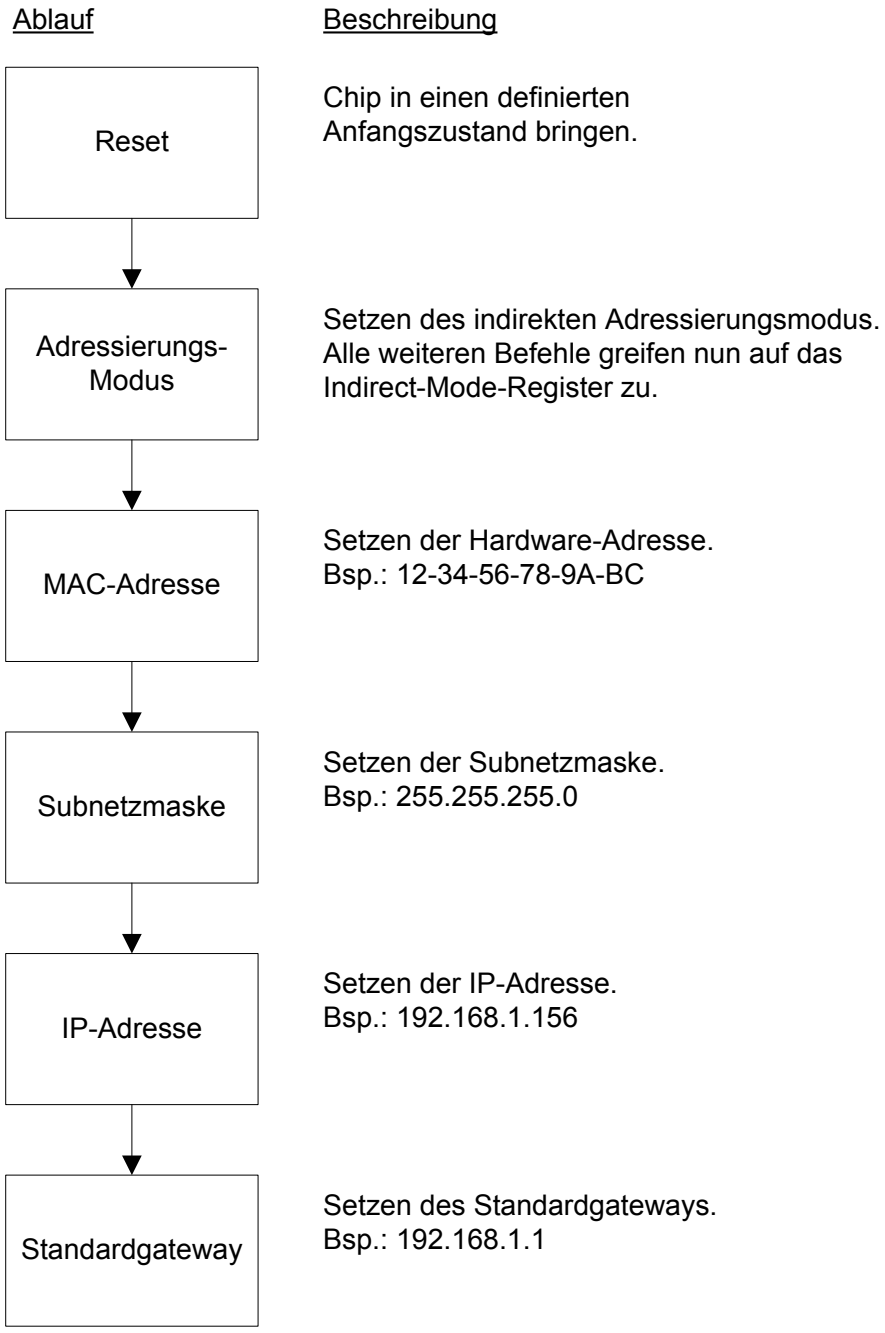

Bild 5-3: Flussdiagramm der Basisinitialisierung des W5300

- **Initialisierung der Verbindungs-Sockets**

Wie bereits zuvor beschrieben bietet der W5300 die Möglichkeit acht voneinander unabhängige Verbindungen mit Client-Rechnern aufzubauen. Jede dieser Verbindungen wird über die dazugehörigen Socket-Register verwaltet.
Im zweiten Schritt der Initialisierungsroutine sind einige dieser Register zu setzen. Bild 5-4 zeigt den Ablauf der Socket-Initialisierung in einem Flussdiagramm.

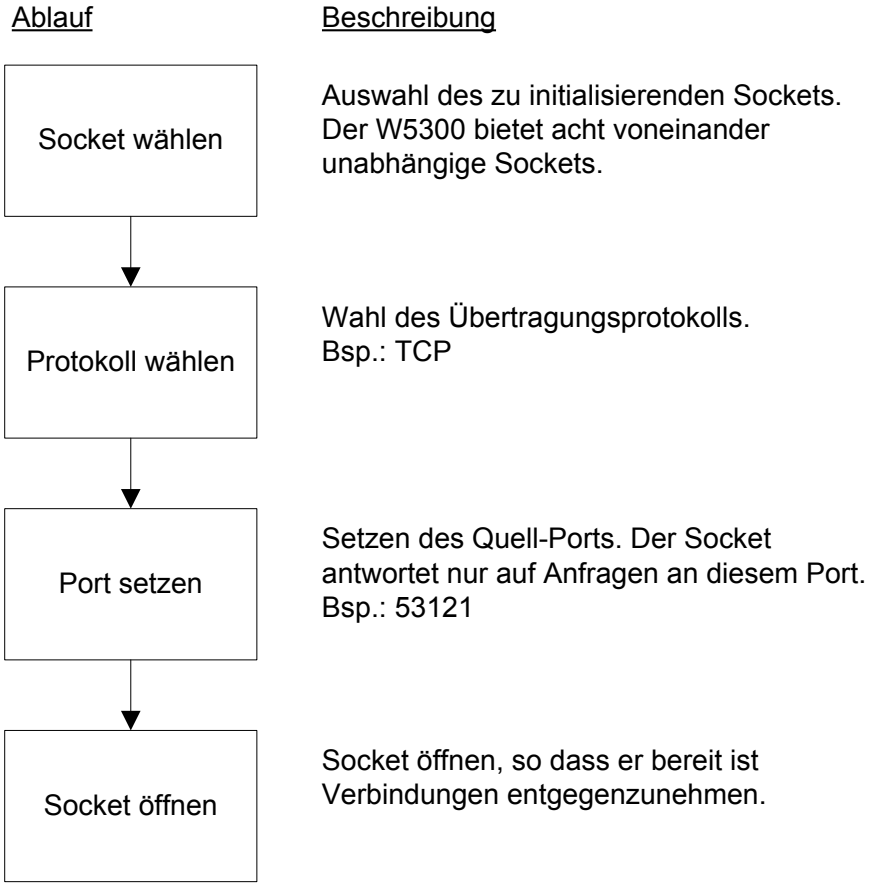

Bild 5-4: Flussdiagramm der Socket-Initialisierung des W5300

5.3.5 Datentransfer mit dem W5300

Hauptaufgabe des W5300 als Kommunikationsmodul des Webservers ist das Senden und Empfangen von Daten. Gesteuert werden diese beiden Prozesse über den Webserver-Chip. Da dieser im späteren Betrieb nicht alleine den Webserver, sondern primär Steuerungsprozesse ausführt, ist es wichtig den Sende- und Empfangsprozess als *nichtblockierende Prozesse* zu implementieren. Hierbei wird der in Kapitel 4.2.2 vorgestellte Ansatz des *kooperativen Multitaskings* verwendet.
Ein Prozess ist hierzu in Form eines Zustandsautomaten realisiert und mittels einer Switch-Case-Anweisung in mehrere kleine Segmente unterteilt. In Listing 5-2 ist ein solcher nichtblockierender Zustandsautomat beispielhaft dargestellt.

```
// Nichtblockierender Zustandsautomat
void nonblocking_statemachine()
{
     // Die Status-Variable muss als static deklariert sein
     // somit bleibt ihr Wert bei einem Sprung aus dem Prozess erhalten
     static int zustand = 0;

     switch (zustand)
     {
          // Erster Abschnitt
          case (0):
          {
               // Hier erfolgt die eigentliche Arbeit
               // des ersten Abschnitts
               Funktionsaufruf_1();

               // Überprüfen ob die Übergangsbedingung erfüllt ist
               if (uebergangsbedingung_zustand_1 == 1)
               {
                    // Zustands-Variable inkrementieren
                    zustand++;
               }
               else
               {
                    // Übergangsbedingung ist nicht erfüllt,
                    // es erfolgt ein Sprung aus dem Zustandsautomat
                    break;
               }
          }
          //Zweiter Abschnitt
          case (1):
          {
               // Hier erfolgt die eigentliche Arbeit
               // des zweiten Abschnitts
               Funktionsaufruf 2 ();

               // Überprüfen ob die Übergangsbedingung erfüllt ist
               if (uebergangsbedingung_zustand_2 == 1)
               {
                    // Zustands-Variable inkrementieren
                    zustand++;
               }
               else
               {
                    // Übergangsbedingung ist nicht erfüllt,
                    // es erfolgt ein Sprung aus dem Zustandsautomat
                    break;
               }
          }
          // Weitere Abschnitte

          default:
          {
               // Standardaktion
          }
     }
}
```

Listing 5-2: Nichtblockierender Zustandsautomat

Wie diese nichtblockierenden Prozesse anzuwenden sind, kann im Handbuch des Webservers nachgelesen werden (siehe Anhang A). Dort ist auch eine Variante vorgestellt, den Webserver in einem *blockierenden* Modus zu betreiben. Diese Variante ermöglicht eine kompaktere Webseitenprogrammierung, sie birgt jedoch die Gefahr, dass der Mikroprozessor bei Wartezeiten in einem Prozess verweilt.

5.3.5.1 Daten mit dem W5300 empfangen

Die Kommunikation zwischen einem Webserver und einem Client-Rechner läuft generell nach dem in Kapitel 4.3 bereits vorgestellten Schema ab. Der Vorgang ist in Bild 5-5 in abgeänderter Form dargestellt. Hierbei richtet der Client-Rechner seine Verbindungsanfrage an den Webserver. Für den serverseitigen Aufbau der Verbindung ist das Kommunikationsmodul (hier W5300) verantwortlich. Verfügt es über einen freien Verbindungs-Socket, bestätig es dem Client den Verbindungsaufbau. Nun können Daten zwischen dem Client-Rechner und dem W5300 ausgetauscht werden.

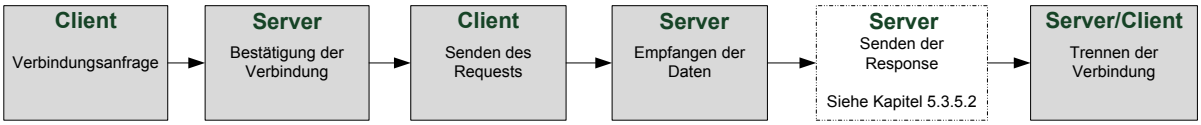

Bild 5-5: Kommunikationsablauf zwischen Webserver und Client-Rechner

Da die HTTP-Kommunikation in der Regel durch eine Anfrage des Client-Rechners (siehe Kapitel 4.4) eingeleitet wird, beginnt dieser den Datentransfer. Der W5300 empfängt die Anfrage und legt die empfangenen Nutzdaten in einem dem Socket zugeordneten Empfangsspeicher ab. Dort können sie von der Webserveranwendung abgeholt werden. Diese analysiert die Daten und sendet gegebenenfalls eine Antwort an den Client-Rechner. Nach Abschluss der Kommunikation wird die Verbindung getrennt. Um Fehler im Kommunikationsablauf abzufangen, ist eine Timeout-Routine implementiert. Sie trennt die Verbindung nach einer bestimmten Zeit der Inaktivität und setzt alle Status-Variablen zurück, so dass der Zustandsautomat beim nächsten Aufruf von vorne beginnt.
Der im Mikrocontroller implementierte Programmablauf ist in Bild 5-6 anhand eines Flussdiagramms dargestellt. Der Sendeprozess, welcher in diesem Diagramm nur angedeutet ist, wird im folgenden Kapitel detailliert erläutert.

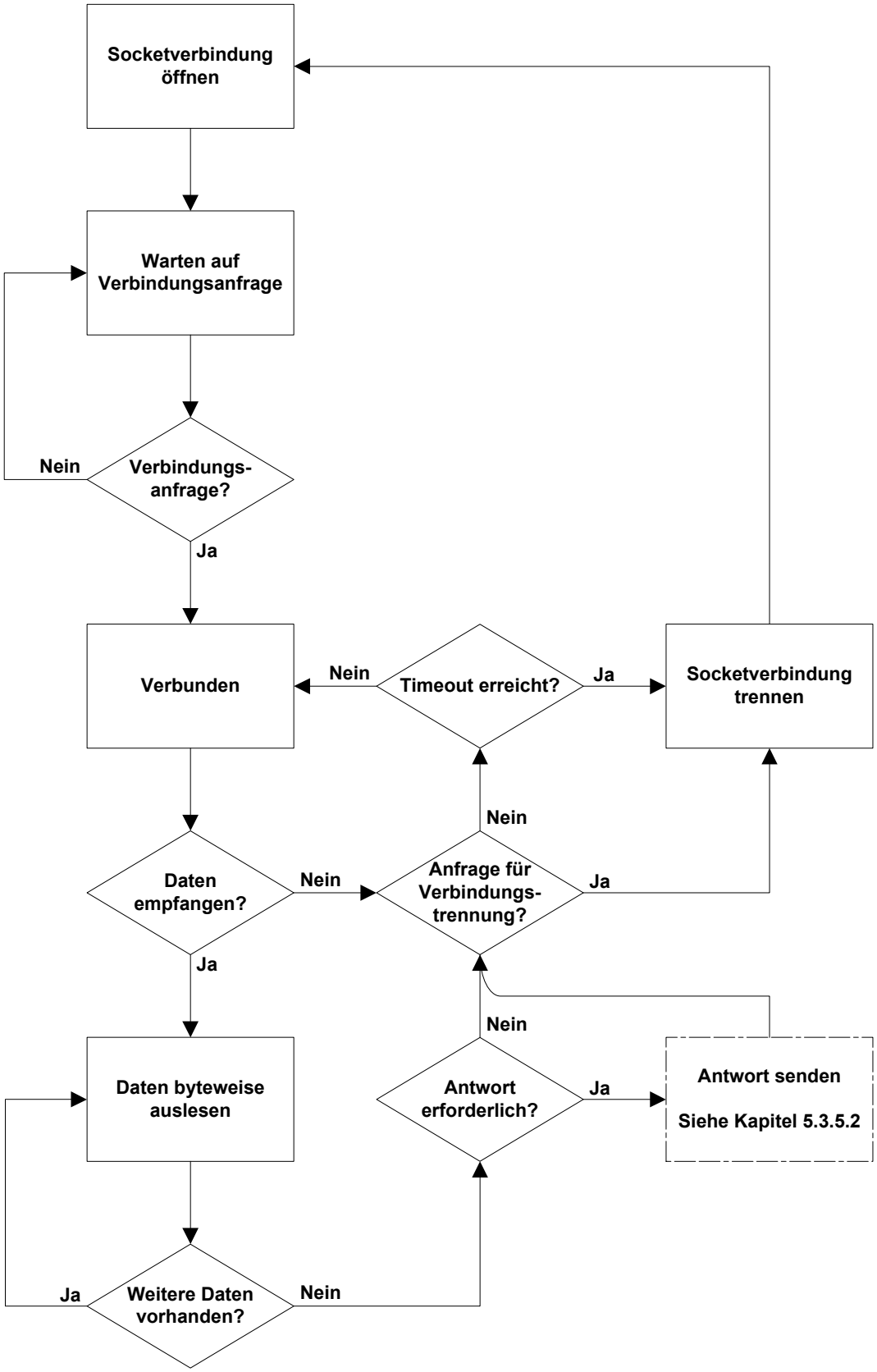

Bild 5-6: Flussdiagramm des Datenempfangsprozesses

5.3.5.2 Daten mit dem W5300 senden

Die meisten Anfragen eines Client-Rechners dienen dazu Daten von einem Webserver anzufordern. Die Anfrage hat also eine Antwortnachricht des Webservers zur Folge. Im vorangegangenen Kapitel ist das generelle Schema eines Kommunikationsablaufes bereits dargestellt worden. Nun wird dieser Ablauf um das Senden der Antwortnachricht ergänzt.
Nachdem der Webserver eine Anfrage erhalten hat, wertet er diese aus, um festzustellen ob der Client-Rechner Antwortdaten anfordert. Ist dies der Fall, geht der Webserver in den Sendeprozess über. Hierfür ist keine neue Verbindung herzustellen. Erst nach Senden der Antwortnachricht wird die bestehende Verbindung beendet. Die im vorigen Kapitel eingeführte Grafik des Kommunikationsablaufs ist in Bild 5-7 um den Sendevorgang erweitert.
Um Fehler im Sendeprozess abfangen zu können, ist auch hier eine Timeout-Routine implementiert. Sie unterbricht den Sendevorgang nach einer bestimmten Zeit der Inaktivität, sendet eine Fehlermeldung an den Client-Rechner und setzt alle Status-Variablen zurück, so dass der Zustandsautomat beim nächsten Aufruf von vorne beginnt.

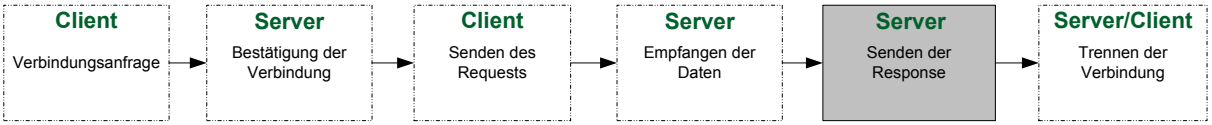

Bild 5-7: Kommunikationsablauf zwischen Webserver und Client-Rechner

Der im Mikrocontroller implementierte Programmablauf zur Realisierung des Sendeprozesses ist in Bild 5-8 als Flussdiagramm dargestellt.

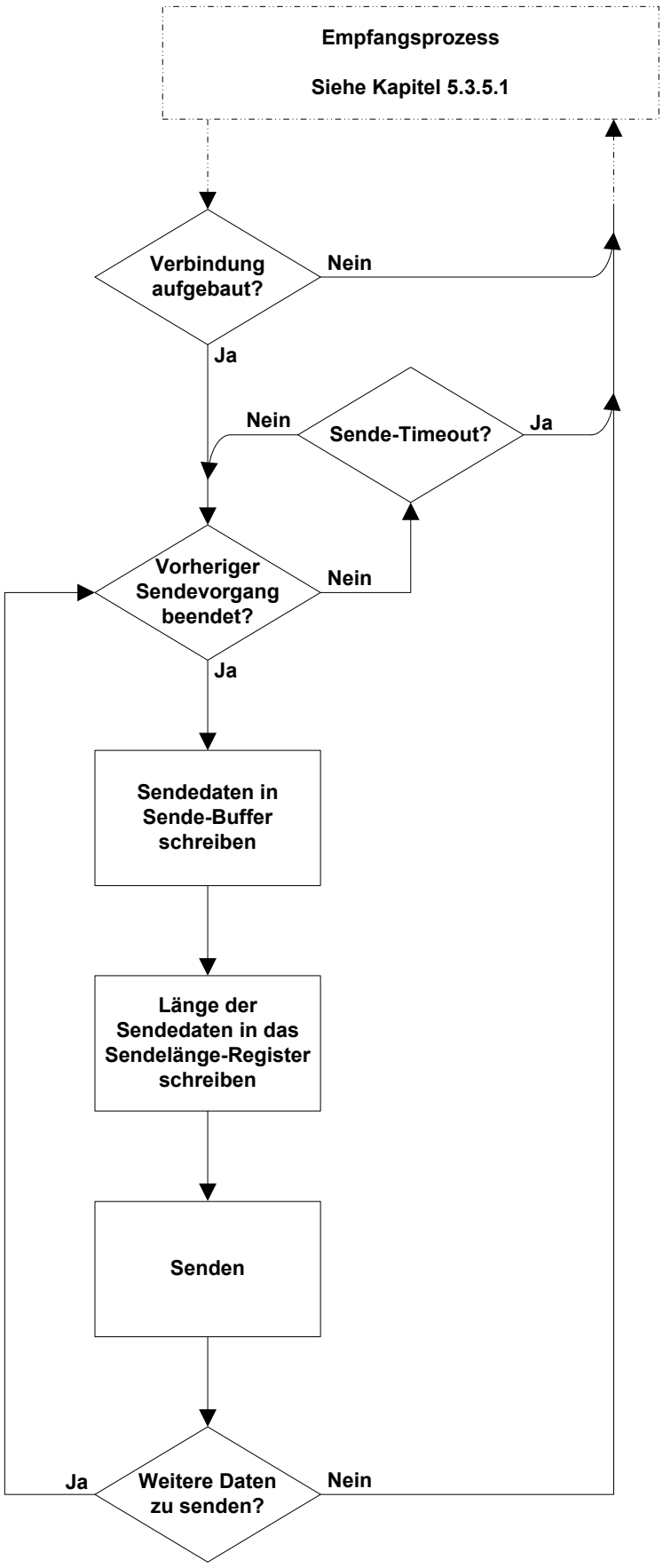

Bild 5-8: Flussdiagramm des Datensendeprozesses

5.4 Die Webserveranwendung

Die Aufgabe der eigentlichen Webserveranwendung besteht darin, Anfragen der Client-Rechner auszuwerten und gegebenenfalls zu beantworten. Hierzu steuert sie das Kommunikationsmodul beim Sende- und Empfangsprozess (siehe Kapitel 5.3.5). Damit der Anwender die Möglichkeit hat, HTML-Code dynamisch von der Webserveranwendung erzeugen zu lassen, ist ihr eine Auswahl an Funktionen implementiert. Mit Hilfe dieser Funktionen ist es möglich Textfelder, Tabellen, Schaltflächen und ähnliche Elemente auf einer Webseite zu platzieren.
In den folgenden Kapiteln ist erläutert, wie die Webserveranwendung ankommende Anfragen analysiert und Antwortnachrichten erzeugt.

5.4.1 Auswerten ankommender Anfragen

Damit die Webserveranwendung auf ankommende Anfragen reagieren kann, muss zuvor bekannt sein welcher Art die Anfragen sein können. Aus den verschiedenen standardisierten Request-Methoden sind hierzu die für dieses Projekt sinnvollen ausgewählt worden (siehe Kapitel 4.4). Es wurde festgelegt, dass der Webserver die Funktionalität bieten soll, GET- und POST-Requests auszuwerten.
Im ersten Schritt der Auswertung muss somit ermittelt werden, um welchen Request-Typ es sich bei der Anfrage handelt. Diese Unterscheidung wird anhand der Nachrichten-Header durchgeführt. Zur Veranschaulichung sind im Folgenden die Nachrichten-Header eines GET- und eines POST-Requests dargestellt.

```
GET /Webseite/?Feld_1=Max&Feld_2=Meier HTTP/1.1
Host: www.beispiel.de
```

Listing 5-3: Header eines GET-Requests

```
POST /Webseite/ HTTP/1.1
Host: www.beispiel.de
```

Listing 5-4: Header eines POST-Requests

Hierbei ist zu erkennen, dass der Header jeweils mit einem Schlüsselwort beginnt, welches die Request-Methode anzeigt. Die Unterscheidung zwischen den beiden Methoden ist somit über einen simplen String-Vergleich zu realisieren. Bild 5-9 zeigt diesen Vorgang anhand eines Flussdiagramms. Dieses Diagramm wird in den folgenden Kapiteln vervollständigt.

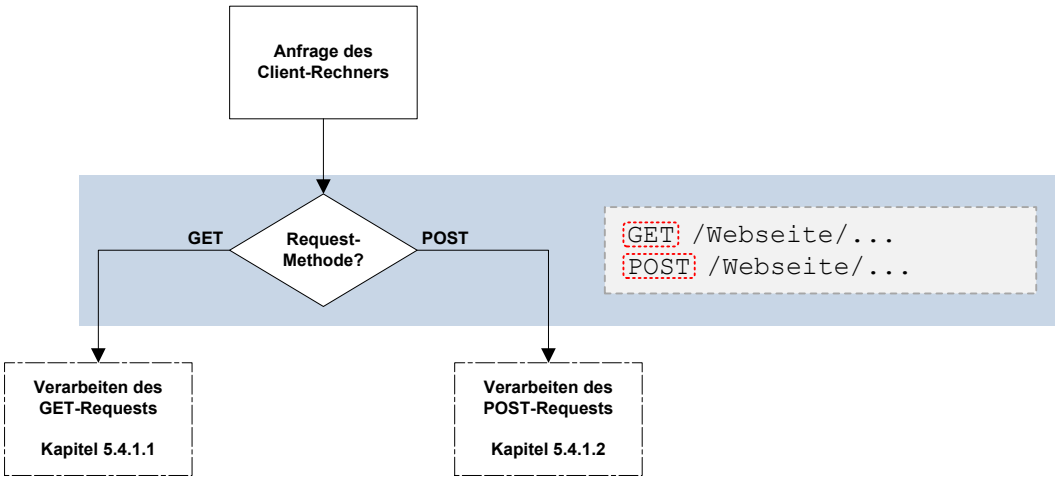

Bild 5-9: Ermitteln der Request-Methode

Bei allen in diesem Projekt entwickelten Analysefunktionen sind Sicherheitsalgorithmen implementiert. Wäre durch einen Übertragungsfehler ein Schlüsselwort nicht in der Zeichenkette vorhanden, bricht die Funktion nach einer gewissen Anzahl von Durchläufen ab, um nicht endlos in einer Schleife zu verweilen. Der Webserver sendet dann eine Fehler-Nachricht an den Client-Rechner, so dass dieser seine Anfrage erneut stellen kann.

5.4.1.1 GET-Request

Um die Auswertung eines GET-Requests zu veranschaulichen, ist in Bild 5-10 die zuvor bereits erwähnte Syntax dieser Request-Methode dargestellt. Die wesentlichen Informationen, die es zu extrahieren gilt, sind zum einen die angeforderte *Webseite* und zum anderen die übermittelten *Datenpaare*.

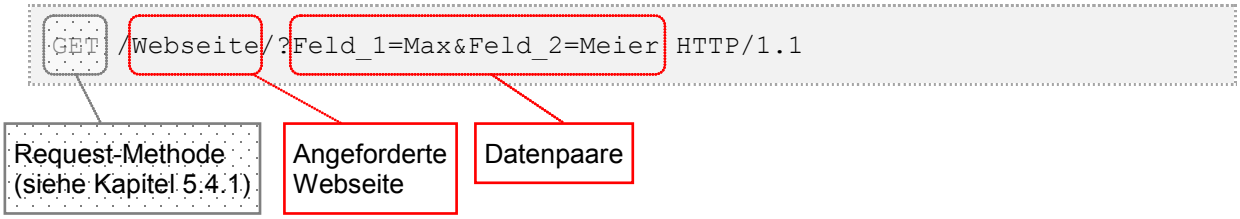

Bild 5-10: Syntax eines GET-Requests

Die in Bild 5-10 dargestellten Nachrichtensegmente werden mit Hilfe hierfür entwickelter Funktionen zur Zeichenkettenanalyse separiert. Der Vorgang ist im Folgenden anhand eines Flussdiagramms erläutert.

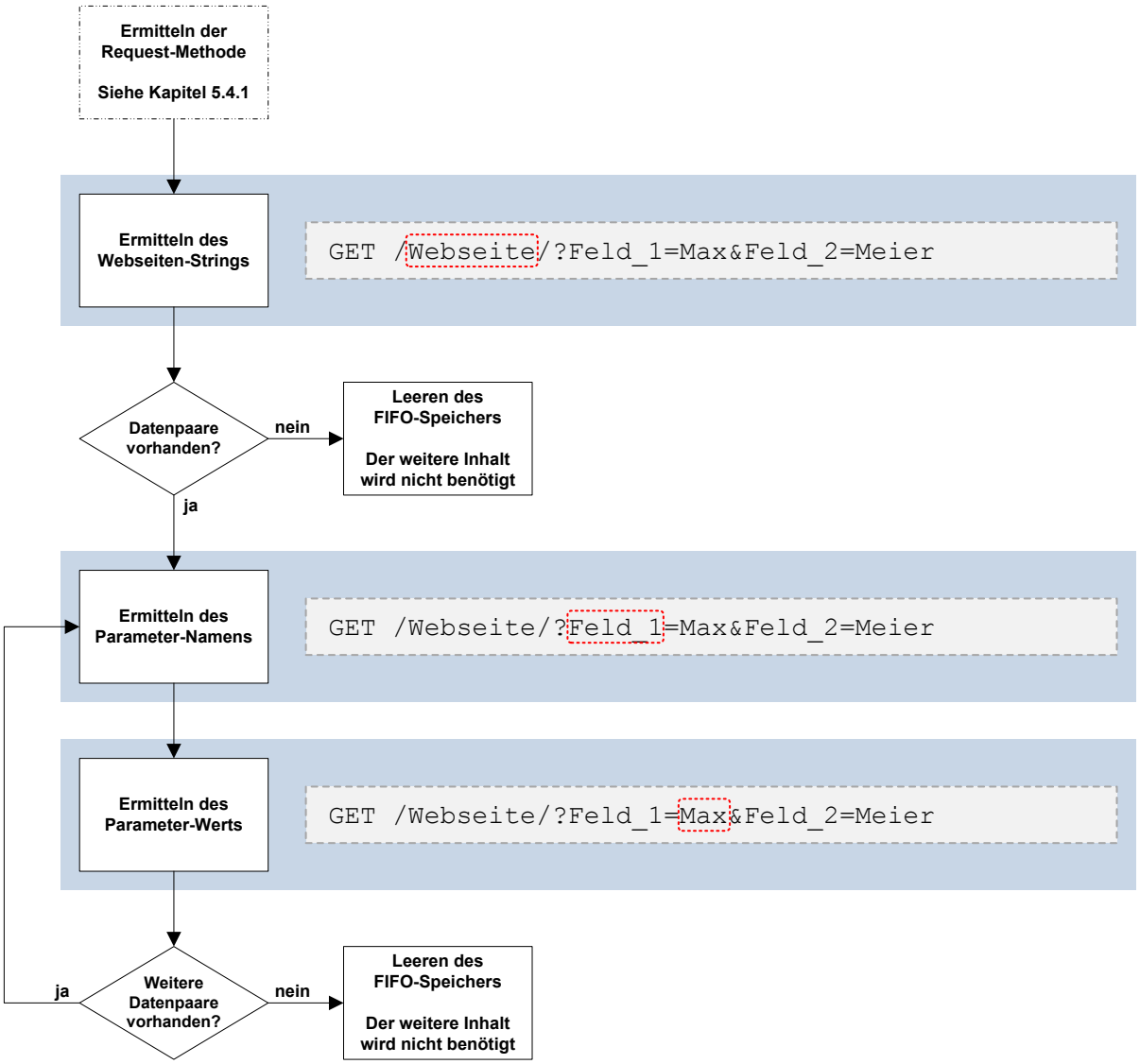

Bild 5-11: Flussdiagramm zur Analyse eines GET-Requests

Der Ablauf des in Bild 5-11 dargestellten Flussdiagramms ist in Tabelle 5-1 näher erläutert. Zur Extraktion der einzelnen Zeichenketten (*Webseiten-String*, *Parameter-Name*, *Parameter-Wert*) ist die Request-Nachricht auf bestimmte Zeichen zu untersuchen. Hierzu wird die Nachricht byteweise aus dem FIFO-Speicher des W5300 entnommen und analysiert. Die so ermittelten Parameter werden in Character-Arrays abgespeichert, um sie dem Anwender zur Verfügung zu stellen. Das Programm-Listing der Funktion „search_next_occurence“, welche einzelne Zeichen in einer Zeichenkette findet um somit die Datenpaare zu extrahieren, ist in Anhang B.1 angefügt.

Tabelle 5-1: Beschreibung des Flussdiagramms zur Analyse von GET-Requests

Schritt	*Beschreibung*
Ermitteln des Webseiten-Strings	Der Webseiten-String wird durch ein Leerzeichen („ “) eingeleitet und durch ein Fragezeichen („?“) oder ein Leerzeichen terminiert. Steht das Fragezeichen im Anschluss an den Webseiten-String, so folgen Datenpaare. Das Leerzeichen zeigt an, dass die Nachricht keine Datenpaare enthält.
Ermitteln des Parameter-Namens	Parameter-Name und Parameter-Wert sind durch ein Gleichheitszeichen („=“) voneinander getrennt. Zur Extraktion des Parameter-Namens werden so lange einzelne Bytes aus dem W5300 entnommen, bis ein Gleichheitszeichen folgt.
Ermitteln des Parameter-Werts	Der Parameter-Wert ist entweder durch ein Kaufmannsund („&“) oder ein Leerzeichen terminiert. Steht das Kaufmannsund im Anschluss an den Parameter-Wert folgen weitere Datenpaare. Das Leerzeichen impliziert, dass keine weiteren Datenpaare gesendet wurden.

5.4.1.2 POST-Request

Um die Auswertung eines POST-Requests zu erläutern ist im Folgenden die bereits erwähnte Syntax dieser Request-Methode dargestellt (siehe Bild 5-12). Die wesentlichen Informationen, die es herauszufiltern gilt, entsprechen denen eines GET-Requests. Dies sind zum einen die angeforderte Webseite und zum anderen die übermittelten Datenpaare.

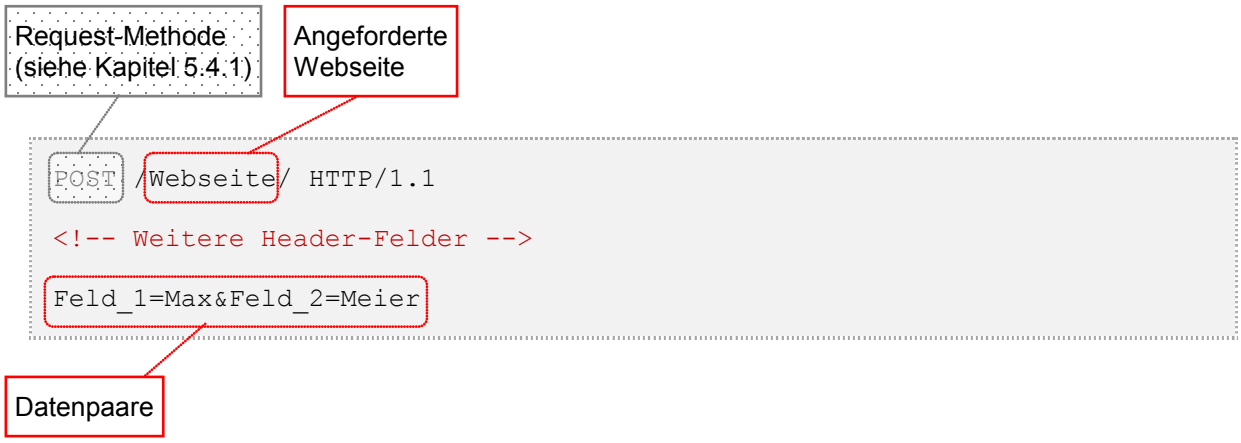

Bild 5-12: Syntax eines POST-Requests

Die in Bild 5-12 dargestellten Nachrichtensegmente sind mit Hilfe hierfür entwickelter Funktionen zur Zeichenkettenanalyse separiert. Der Ablauf eines solchen Analysevorgangs ist im Folgenden in einem Flussdiagramm dargestellt.

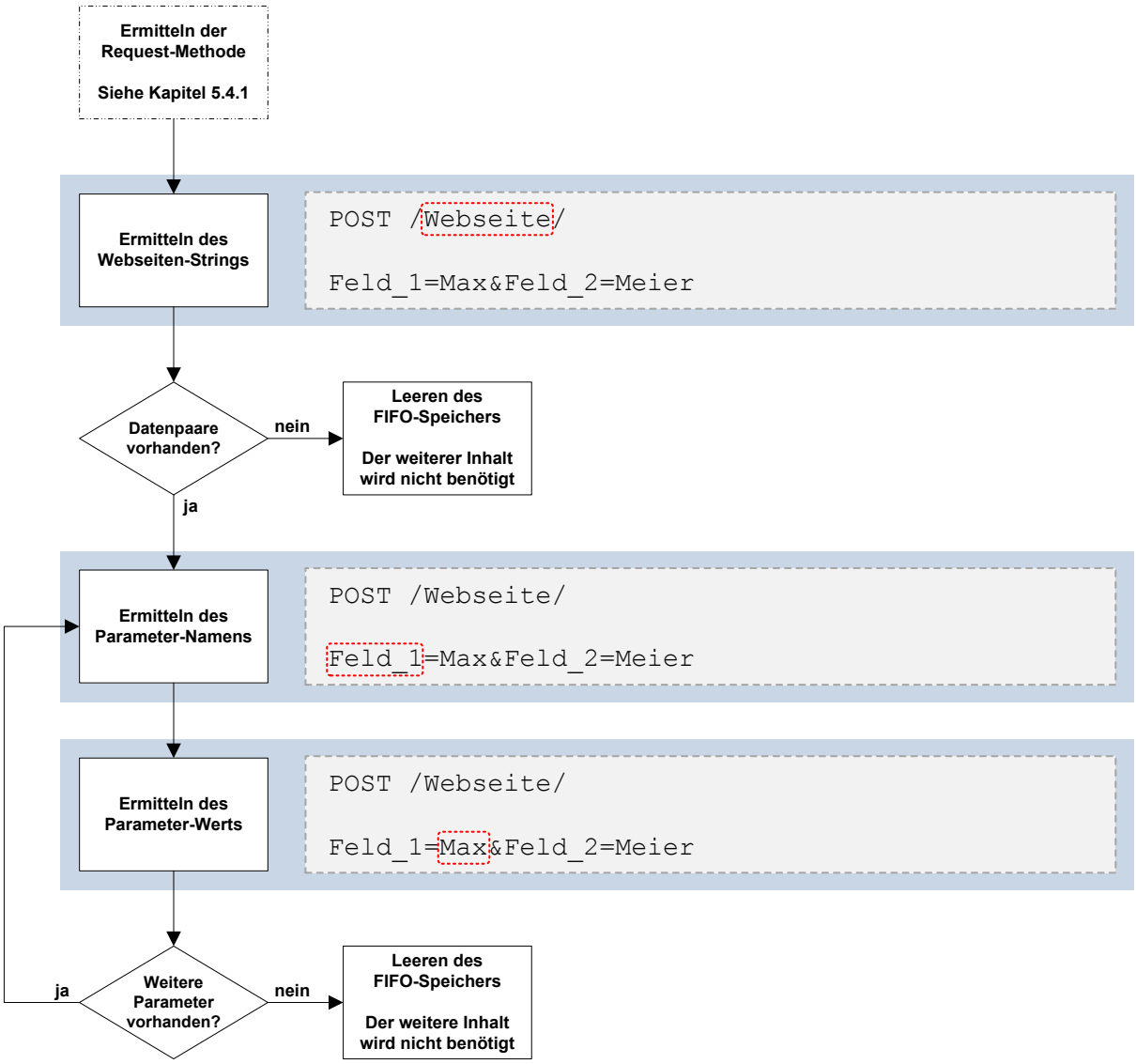

Bild 5-13: Flussdiagramm zur Analyse eines POST-Requests

Der Ablauf des Flussdiagramms ist in Tabelle 5-2 näher beschrieben. Der wesentliche Unterschied gegenüber der Verarbeitung eines GET-Requests liegt darin, dass die Datenpaare im Nachrichten-Body übermittelt werden. Der Body der wird hierbei durch eine Leerzeile, als ASCII-Zeichen ist dies „\r\n\r\n“, eingeleitet. Somit ist hier eine Funktion zu entwickeln, die den Nachrichtentext nicht auf ein einzelnes Zeichen (vgl. Kapitel 5.3.1.1 GET-Request), sondern auf eine Kette von Zeichen, ein Schlüsselwort, untersucht. In Anhang B.2 ist das Programm-Listing der Funktion „search_pattern“ angefügt. Sie dient dazu Schlüsselwörter in einer Zeichenkette zu finden.

Tabelle 5-2: Beschreibung des Flussdiagramms zur Analyse von POST-Requests

Schritt	*Beschreibung*
Ermitteln des Webseiten-Strings	Der Webseiten-String wird durch ein Leerzeichen eingeleitet und durch ein weiteres Leerzeichen terminiert.
Ermitteln des Parameter-Namens	Der erste Parameter-Name wird durch eine Leerzeile („\r\n\r\n“) eingeleitet. Deshalb gilt es diese Zeichenkette zu finden. Parameter-Name und Parameter-Wert sind durch ein Gleichheitszeichen voneinander getrennt. Zur Extraktion des Parameter-Namens werden so lange einzelne Bytes aus dem FIFO-Speicher des W5300 entnommen, bis ein Gleichheitszeichen erkannt wird.
Ermitteln des Parameter-Werts	Der Parameter-Wert ist entweder durch ein Kaufmannsund oder ein Leerzeichen terminiert. Steht das Kaufmannsund im Anschluss an den Parameter-Wert, so folgen weitere Datenpaare. Das Leerzeichen indiziert, dass keine weiteren Datenpaare gesendet wurden.

5.4.2 Erzeugen der Antwortnachricht

Nachdem eine Anfrage abgearbeitet und in ihre Einzelparameter aufgeteilt wurde, erwartet der Client-Rechner in den meisten Fällen eine Antwort (meist werden durch eine Anfrage Inhalte vom Webserver angefordert). Damit der Programmierer der Webanwendung diese Antwort möglichst komfortabel erzeugen kann, sind geeignete Funktionen zu entwickeln, die es ermöglichen HTML-Code dynamisch zu generieren. Die in der Webserveranwendung implementierten Funktionen zur Erzeugung einer Antwortnachricht sind im Folgenden am Beispiel eines Eingabefelds ausführlich beschrieben. Hierzu sind Listings aus dem entstandenen Programmcode aufgeführt und erläutert.

Darstellen eines Eingabefelds

Der HTML-Code, welcher ein Eingabefeld auf einer Webseite erzeugt, ist in Bild 5-14 aufgeführt. Der Code enthält zwei variable und einen statischen Parameter. Der interne *Variablen-Name* („*name*“) und die Länge des Felds („*size*“) sind vom Programmierer frei wählbar. Der Typ des Felds („*type*“) ist vorgegeben und legt fest, dass es sich bei diesem Element um ein Eingabefeld für Texteingaben handelt.

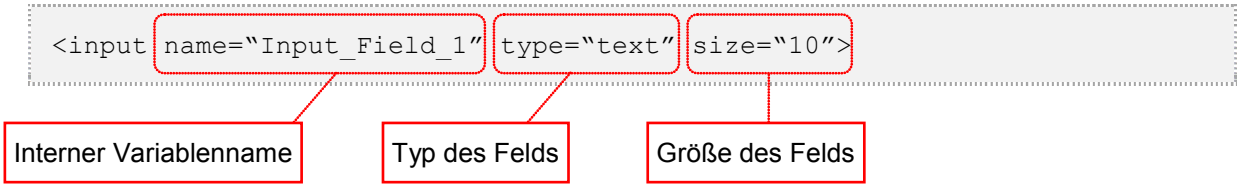

Bild 5-14: HTML-Code eines Eingabefelds

Im Folgenden ist beschrieben, wie oben dargestellter HTML-String erzeugt und versendet wird. Die Beschreibung ist in zwei Abschnitte unterteilt. Der erste Abschnitt erläutert die *Erzeugung des HTML-Strings*, der zweite beschreibt das eigentliche *Versenden des HTML-Strings*.

- **Erzeugen des HTML-Strings**

Zur Erzeugung des in Bild 5-14 dargestellten HTML-Strings, müssen die vom Programmierer frei wählbaren Parameter in den statischen Teil des Strings eingearbeitet werden. Dies ist über die hierfür entwickelte String-Funktion „*strcat*“ realisiert, welche zwei Zeichenketten zu einer zusammensetzt[9]. Listing 5-5 erläutert ihre Funktionsweise.

```
// Aufruf der Funktion „strcat“

// Deklarieren eines Character-Arrays in welches
// die beiden Strings kopiert werden
char array[64] = “”;

// Kopieren des ersten Strings
// an den Anfang des Char-Arrays
strcat(array, “String_1”);

// Kopieren des zweiten Strings
// hinter den ersten String
strcat(array, “String_2”);
```

Listing 5-5: Die Funktion strcat

Die Ausgabe des Character-Arrays liefert folgendes Ergebnis:

```
String_1String_2
```

Listing 5-6: Ausgabe der zusammengesetzten Zeichenkette

Die Funktion *strcat* ist ein wichtiges Werkzeug bei der Erzeugung des HTML-Strings. Mit ihrer Hilfe ist es möglich, die verschiedenen Segmente des HTML-Strings zusammenzufügen. Im Folgenden ist nun die Funktion *create_inputfield* dargestellt, welche die einzelnen Segmente des Strings zusammenführt. Ihr werden beim Aufruf zwei Parameter übergeben, der *interne Variablenname* und die *Größe des Eingabefelds*.
Zur Speicherung des HTML-Strings wird ein Character-Array angelegt. Die ersten sechs Stellen dieses Arrays bleiben unberührt, weil hier im nächsten Schritt die

[9] Der Compiler-Entwickler stellt die Funktion strcat auch als fertige Library-Funktion zur Verfügung. Sie ist in der Library string.h enthalten. Auf das Einbinden von Library-Funktionen wird jedoch aus Gründen der Speicherplatzoptimierung verzichtet.

hexadezimale Länge des HTML-Strings eingefügt wird[10]. Dieser Vorgang ist im Abschnitt „Versenden des HTML-Strings“ detailliert beschrieben.

```
// Funktion zur dynamischen Erzeugung von HTML-Code
// zur Darstellung eines Eingabefelds

int create_inputfield(char *Name, char *Size)
{
     // Variablendeklaration
     char Data_Buffer[128] = "";  // Char-Array zur Speicherung
                                  // des HTML-Strings
     char Size_Buffer[6];     // Char-Array zur Speicherung
                              // des Längen-Strings
     int I_Ret = 0;           // Rückgabewert
     int count_cpy = 0;       // Zählvariable

     // Erzeugen des HTML-Strings
     Data Buffer[6] = 0;      // Stelle 6 des Data-Buffers muss 0 sein,
                              // hier beginnt der HTML-String

     // Zusammensetzen der statischen und
     // der variable Stringsegmente
     strcat((Data_Buffer + 6), "<input name=\"");
     strcat((Data_Buffer + 6), Name);
     strcat((Data_Buffer + 6), "\" type=text size=\"");
     strcat((Data_Buffer + 6), Size);
     strcat((Data_Buffer + 6), "\">\r\n\r\n");

     // Aufruf des Makros SEND_HTML_STRING
     // um den zuvor erzeugten HTML-String zu versenden
     SEND_HTML_STRING(Data_Buffer, Size_Buffer);
}
```

Listing 5-7: Die Funktion create_inputfield

- **Versenden des HTML-Strings**

Zum Versenden des zuvor erzeugten HTML-Strings wird die in Kapitel 4.3.3 vorgestellte Methode des *Chunked Transfer-Encodings* verwendet. Mit ihrer Hilfe ist es möglich, die Gesamtnachricht in mehreren kleineren Blöcken zu versenden. Die Syntax eines solchen Blocksegments ist in Bild 5-15 dargestellt. Hierbei wird dem eigentlichen HTML-String ein Hexadezimalwert mit der Länge des HTML-Strings vorangestellt. Anhand dieser Angabe kann der Client-Rechner beim Empfang der Nachricht feststellen, ob diese vollständig übermittelt wurde.

[10] Beim Chunked Transfer-Encoding wird die Länge eines Datenblocks als Hexadezimalwert vor diesem eingefügt.

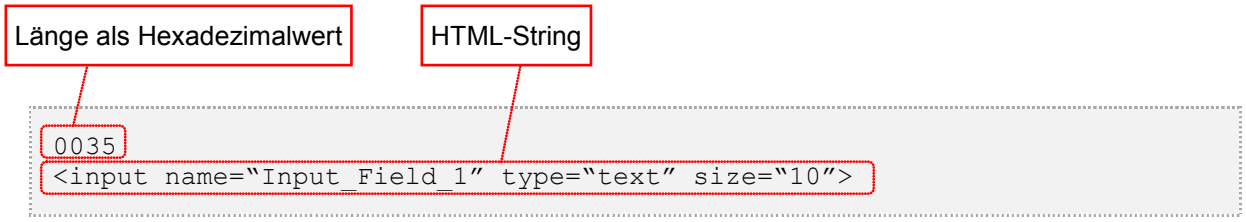

Bild 5-15: Syntax eines Nachrichtenblocks beim Chunked Transfer-Encoding

Der HTML-String wird über ein Makro mit dem Namen *SEND_HTML_STRING* versendet. Das Makro ermittelt die Länge des HTML-Strings als Integerwert und konvertiert diesen in einen Hexadezimalwert, um ihn vor dem eigentlichen HTML-String in die Zeichenkette einzufügen. Der dabei entstehende String wird im Anschluss über das Kommunikationsmodul an den Client-Rechner versendet.
Dieser Aufwand wird betrieben, weil somit nur ein großes Array angelegt werden muss. Würde man den HTML-String hinter den Hexadezimalwert setzen, wären zwei Arrays nötig. Hierdurch wird Speicherplatz gespart.

```
/////// SEND_HTML_STRING ////////
// Dieses Makro erfüllt die folgenden Aufgaben:
// - ermitteln der Länge des HTML-Strings
// - konvertieren des Integerwerts in einen Hexadezimal-String
// - kopieren des Hexadezimal-Strings vor den HTML-String
// - versenden des Gesamt-Strings
////////////////////////////////
SEND_HTML_STRING(DATA_BUFFER, SIZE_BUFFER)

// Ermitteln der Länge des HTML-Strings und
// konvertieren des Ergebnisses in einen Hexadezimal-String
integer_to_hexString((strlen(DATA_BUFFER + 6)), SIZE_BUFFER);

// Anfügen eines Zeilenumsprungs hinter den Hexadezimalwert
strcat(SIZE_BUFFER, "\r\n");

// Kopieren des Hexadezimalwerts an den Anfang des Buffer-Arrays
for (count_cpy = 0; count_cpy < 6; count_cpy++)
{
     DATA_BUFFER[count_cpy] = SIZE_BUFFER[count_cpy];
}

// Versenden des Nachrichtenblocks
I_Ret = sende(DATA_BUFFER);  // sende() ist die Funktion, welche die
                             // Daten an den W5300 weiterleitet und den
                             // eigentlichen Sendevorgang veranlasst
return(I_Ret);               // Rückgabe des Statuswerts
```

Listing 5-8: Das Makro SEND_HTML_STRING[11]

[11] Die Kommentare und Zeilenumbrüche in dem dargestellten Listing sind nur zum Verständnis eingefügt. Das später implementierte Makro enthält diese nicht.

Der Programmierer ruft die Funktion *create_inputfield* mit der in Listing 5-9 aufgeführten Befehlssyntax auf. Der Rückgabewert enthält Informationen über den Status des Sendevorgangs.

```
ret = create_inputfield("Name_des_Felds", "13");
```

Listing 5-9: Aufruf der Funktion create_inputfield

5.4.3 Formatieren der Webseite

Um dem Programmierer die Formatierung einer Webseite und der darauf angebrachten Elemente möglichst einfach und komfortabel zu gestalten, werden in diesem Projekt sogenannte *Stylesheets* eingesetzt. Stylesheets bieten die Möglichkeit Formatvorlagen für eine Webseite zu erstellen. Diese Formatvorlagen werden zu Beginn einer Webseite angelegt und sind für die gesamte Seite gültig (ähnlich wie es bei Formatvorlagen in anderen Dokumenten der Fall ist). Stylesheets sind sehr flexibel einsetzbar. Es lässt sich zum Beispiel eine Formatvorlage für den gesamten Nachrichten-Body einer Webseite anlegen. Weiterhin besteht die Möglichkeit, Formatvorlagen für alle Elemente einer Gruppe (zum Beispiel alle Eingabefelder) oder für einzelne Elemente zu definieren.
Als Stylesheet-Sprache wird in diesem Projekt „Cascading Stylesheets“ (CSS) verwendet (für nähere Informationen zu CSS siehe [2]). Listing 5-10 zeigt den HTML-Code einiger Stylesheets.

```
<style type="TEXT/CSS" MEDIA="SCREEN">
    #Style_1 { font-size:30px; text-align:left; font-family:Arial; }
    #Style_2 { font-size:15px; text-align:center; font-family:Arial; }
    table { font-size:30px; border:2px solid; margin-bottom:6px; }
    body { background-color:#663333; color:#FFCC99; font-family:Arial; }
</style>
```

Listing 5-10: Beispielhafte Stylesheets

Die mit einem Nummernzeichen („#“) eingeleiteten Zeilen stellen Formatvorlagen dar, die für einzelne HTML-Elemente verwendet werden. Man könnte die Vorlage *#Style_1* zum Beispiel einem Textfeld zuweisen, welches dadurch folgendermaßen formatiert würde (siehe Tabelle 5-3).

Tabelle 5-3: Stylesheet-Beispiel 1

Attribut	***Wert***
Schriftart	Arial
Textausrichtung	Links
Schriftgröße	30 px

Formatvorlagen ohne ein führendes Nummernzeichen sind automatisch für alle Elemente einer Gruppe definiert. So dient die Vorlage *table* dazu, alle Tabellen einer Webseite einheitlich zu formatieren. Über die Formatvorlage *body* wird der gesamte Body einer Webseite formatiert. Dies ist sinnvoll um die Hintergrundfarbe oder die Schriftfarbe für eine Webseite festzulegen. In dem oben dargestellten Beispiel hätte der Body der Webseite die in Tabelle 5-4 aufgeführten Eigenschaften.

Tabelle 5-4: Stylesheet-Beispiel 2

Attribut	***Wert***
Schriftart	Arial
Hintergrundfarbe	#663333 „Dunkelrot“
Schriftfarbe	#FFCC99 „Rot-Orange“

6 Ergebnisse

6.1 Auf dem Webserver programmierte Webseiten

Um darzustellen, welchen Funktionsumfang der Webserver bei der Erzeugung von Webseiten bietet, sind im Folgenden beispielhaft einige auf ihm programmierte Webseiten aufgeführt. Sie repräsentieren die typischen Einsatzgebiete des Webservers.
Die Webseiten gliedern sich in zwei Bereiche: Ein *Navigationsmenü*, welches die Navigation auf der Webseite realisiert, und ein sogenannter *Content-Frame*, auf dem die angeforderten Informationen bereitgestellt werden.
Das Grundgerüst mit dem Navigationsmenü ist bei allen Webseiten identisch und enthält die in Kapitel 6.1.1 erläuterten Bestandteile. Lediglich der Content-Frame variiert in Abhängigkeit der dargestellten Informationen.
In Kapitel 6.1.2 ist eine *Willkommensseite* gezeigt, auf der dem Anwender eine Beschreibung der Webseite dargestellt wird. Auf einer weiteren Webseite werden *Prozessparameter* visualisiert (siehe Kapitel 6.1.3). Die in Kapitel 6.1.4 vorgestellte Webseite bietet dem Anwender die Möglichkeit, verschiedene *Netzwerkeinstellungen* des Kommunikationsmoduls zu ändern.
Die Programmierung dieser Webseiten ist im Handbuch zum Webserver in Anhang A ausführlich erläutert.

6.1.1 Grundgerüst der Webseiten

Das Grundgerüst der in den folgenden Kapiteln aufgeführten Webseiten ist in Bild 6-1 gezeigt. Zur Erläuterung der einzelnen Bestandteile ist im Folgenden der vom Webserver erzeugte HTML-Code dargestellt und beschrieben.

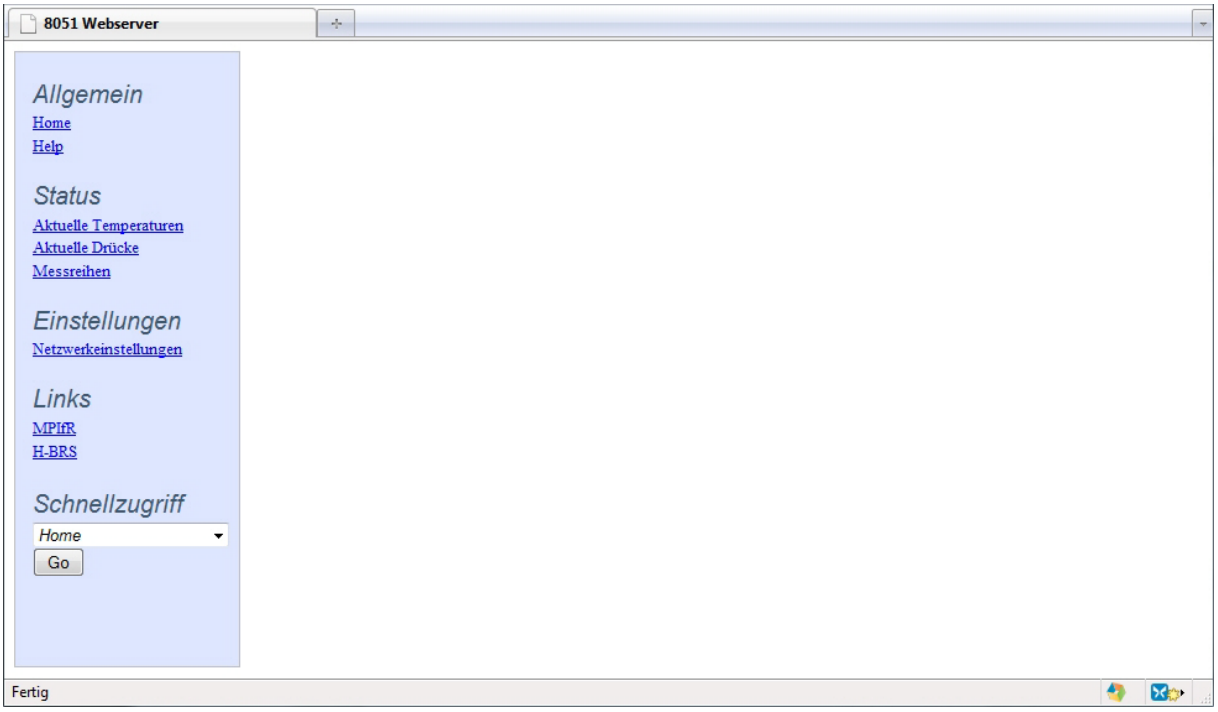

Bild 6-1: Grundgerüst der Webseiten

- **Header**

Der Header enthält keinen HTML-Code sondern ergänzende Zusatzinformationen zu den eigentlichen Nutzdaten. Er wird bei jeder Kommunikation vor die eigentlichen Nutzdaten gesetzt und enthält Parameter wie den *Servernamen*, die verwendete *HTTP-Version* und die *Übertragungsmethode*. Der vom Webserver erzeugte Header ist in Listing 6-1 dargestellt.

```
HTTP/1.1 200 OK <!-- HTTP-Version -->
Server: 8051 Webserver <!-- Server-Name -->
Content-Type: text/html <!-- Medientyp -->
Transfer-Encoding: chunked <!-- Übertragungsmethode -->
```

Listing 6-1: Header der Webseite

- **Head**

Der Head (Kopf) der Webseite enthält Informationen wie den *Webseitentitel*, das *Dokumentenformat* oder die *Stylesheets*. Er ist in Listing 6-2 dargestellt.

```
<!-- Dokumentenformat -->
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" lang="de">
<head>
```

```
<!-- Titel der Webseite -->
<title>8051 Webserver</title>

<!-- Stylesheets -->
<style type="TEXT/CSS" MEDIA="SCREEN">
ul#list_menue { float: left; width: 10.5em; background-color: #DDE5FF;
margin:0; border: 1px solid silver; color: #40586F; height: 30em;
padding-bottom: 1em; padding-left: 0.8em; }
ul#list_menue li { font-size:13px; list-style: none; margin: 0;
padding: 0.1em; }
div#menue_class { font-size:20px; text-align:left; font-family:Arial;
font-style:italic;padding-top:1em; }
div#menue_title { font-size:22px; font-family:Arial; padding: 0; text-
decoration:underline; }
div#content { margin-left: 12em; height: 30em; padding-top: 1em;
padding-left:1em;padding-right: 1em; border: 1px solid silver; max-
width:50em; }
ul#list_menue select { font-size:13px; font-family:Arial; font-
style:italic; }
div#content h1 { background-color: #DDE5FF; color: #40586F; margin:0;
padding-left:0.2em }
div#content h2 { font-size:18px; font-family:Arial; padding-left:0.2em;
}
div#description { font-size:6px; font-family:Arial; padding-left:0.2em;
}
table { font-size:20px; font-family:Arial; margin-left:2em; margin-
top:2em; width:25em; }
#td_left { text-align:left; padding:0.4em; }
</style>
</head>
```

Listing 6-2: Head der Webseite

- **Navigationsmenü**

Um die Navigation auf der Webseite zu ermöglichen wird ein Navigationsmenü erzeugt. Es ist über eine Liste realisiert, bei der die einzelnen Einträge unterschiedliche Formatvorlagen zugewiesen bekommen. Die Listeneinträge sind als Links ausgeführt, die bei Ausführen einen GET-Request an den Webserver senden. Listing 6-3 stellt den dazugehörigen HTML-Code dar.

```
<!-- Beginn des Bodys der Webseite -->
<body>

<!-- Beginn der Liste -->
<ul id="list_menue">

<!-- Kapitelüberschrift „Allgemein“ -->
<li>
<div id="menue_class">
Allgemein
</div>
</li>
```

```
<!-- Listeneintrag „Home“ -->
<li>
<a href="http://134.104.28.209:53121">Home</a>
</li>

<!-- Listeneintrag „Help“ -->
<li>
<a href="http://134.104.28.209:53121">Help</a>

<!-- Kapitelüberschrift „Status“ -->
<div id="menue_class">
Status
</div>
</li>

<!-- Listeneintrag „Aktuelle Temperaturen“ -->
<li>
<a href="http://134.104.28.209:53121/current_temp">Aktuelle
Temperaturen</a>
</li>

<!-- Listeneintrag „Aktuelle Drücke“ -->
<li>
<a href="http://134.104.28.209:53121/current_pressure">Aktuelle
Dr&#252cke</a>
</li>

<!-- Listeneintrag „Messreihen“ -->
<li>
<a href="http://134.104.28.209:53121/measurement">Messreihen</a>
<div id="menue_class">
Einstellungen
</div>
</li>

<!-- Listeneintrag „Netzwerkeinstellungen“ -->
<li>
<a href="http://134.104.28.209:53121/network">Netzwerkeinstellungen</a>

<!-- Kapitelüberschrift „Links“ -->
<div id="menue_class">
Links
</div>
</li>

<!-- Listeneintrag „MPIfR“ -->
<li>
<a href="http://www.mpifr.de">MPIfR</a>
</li>

<!-- Listeneintrag „H-BRS“ -->
<li>
<a href="http://www.h-brs.de">H-BRS</a>
</li>

<!-- Ende der Liste -->
</ul>
```

Listing 6-3: Navigationsmenü der Webseite

6.1.2 Darstellen einer Willkommensseite

Bild 6-2 zeigt eine Willkommensseite, die auf dem hier vorgestellten Webserver programmiert ist. Der HTML-Code des Content-Frames ist in Listing 6-4 aufgeführt.

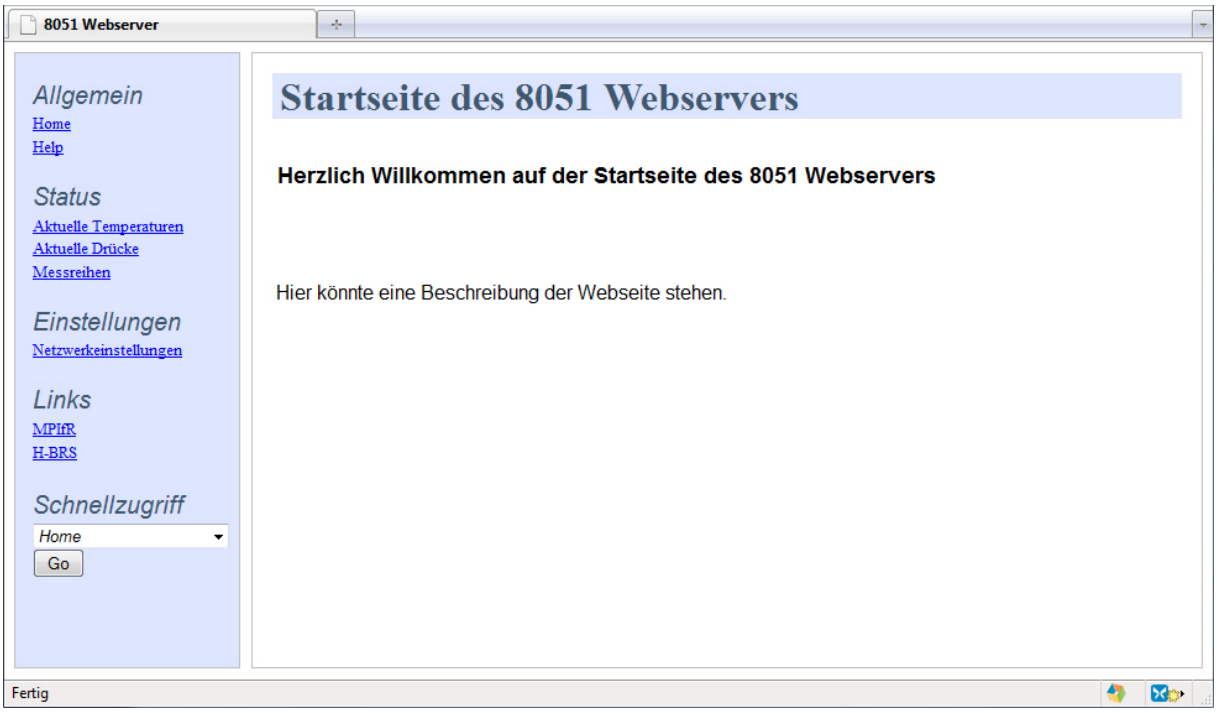

Bild 6-2: Beispiel einer Willkommensseite

- **Content-Frame**

```
<!-- Überschrift der Webseite -->
<div id="content">
<h1>Startseite des 8051 Webservers</h1>
<br/>

<!-- Willkommenstext -->
<h2>Herzlich Willkommen auf der Startseite des 8051 Webservers</h2>
<br/><br/>
<div id="description">
<br/>

<!-- Beschreibungstext -->
Hier k&#246nnte eine Beschreibung der Webseite stehen.
</div>
</div>

<!-- Ende der Webseite -->
</body>
</html>
```

Listing 6-4: Content-Frame der Willkommensseite

6.1.3 Visualisierung von Prozessparametern

In Bild 6-3 sind die aktuellen Werte einiger Temperatursensoren aufgeführt. Die Ausrichtung des Texts ist mittels einer Tabelle realisiert.

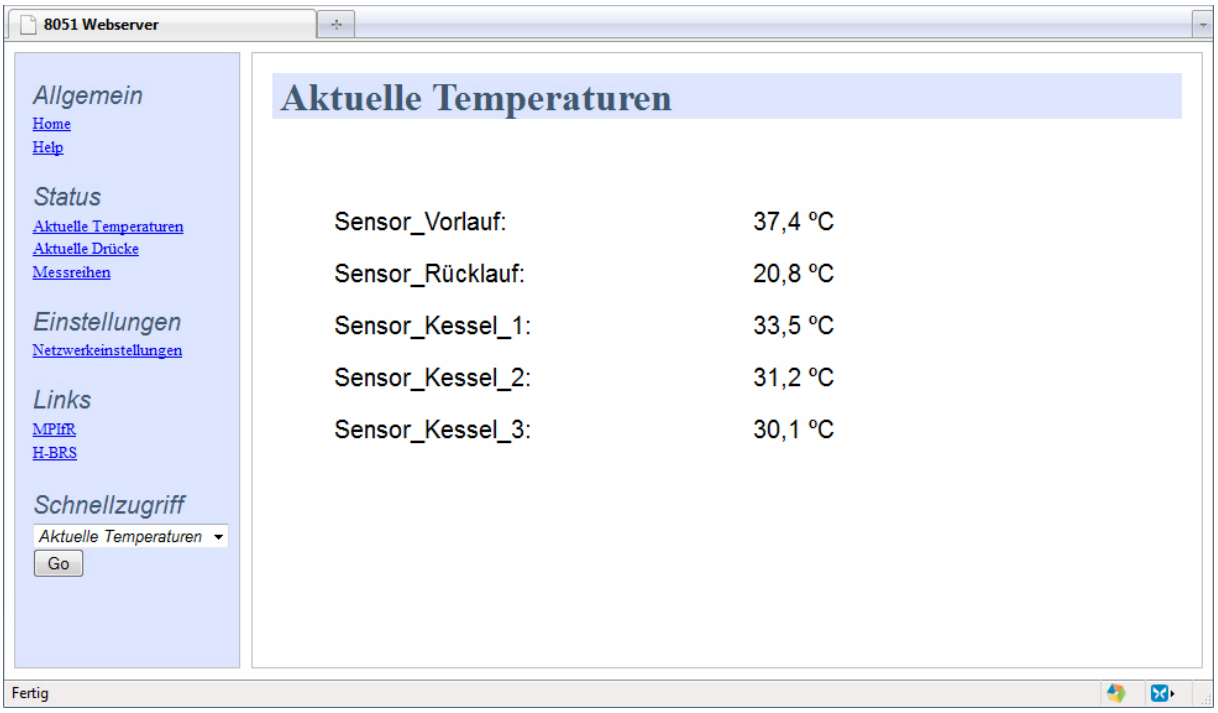

Bild 6-3: Beispiel einer Webseite zur Darstellung von Prozessparametern

- **Content-Frame**

```
<!-- Überschrift der Webseite -->
<div id="content">
<h1>Aktuelle Temperaturen</h1>
<br/>

<!-- Beginn der Tabelle -->
<table>
<tr>

<!-- Sensorname -->
<td id="td_left">
Sensor_Vorlauf:
</td>

<!-- Temperaturwert -->
<td id="td_left">
37,4 &#186C <!-- Sonderzeichen sind durch „&#" eingeleitet -->
</td>

<!-- Beginn einer neuen Tabellenzeile -->
</tr>
<tr>
<td id="td_left">
Sensor_R&#252cklauf:
```

```
</td>
<td id="td_left">
20,8 &#186C
</td>

<!-- Beginn einer neuen Tabellenzeile -->
</tr>
<tr>
<td id="td_left">
Sensor_Kessel_1:
</td>
<td id="td_left">
33,5 &#186C
</td>

<!-- Beginn einer neuen Tabellenzeile -->
</tr>
<tr>
<td id="td_left">
Sensor_Kessel_2:
</td>
<td id="td_left">
31,2 &#186C
</td>

<!-- Beginn einer neuen Tabellenzeile -->
</tr>
<tr>
<td id="td_left">
Sensor_Kessel_3:
</td>
<td id="td_left">
30,1 &#186C
</td>
</tr>

<!-- Ende der Tabelle -->
</table>
</div>

<!-- Ende der Webseite -->
</body>
</html>
```

Listing 6-5: Content-Frame der Webseite zur Prozessparametervisualisierung

6.1.4 Anzeigen und ändern von Netzwerkeinstellungen

Die in Bild 6-4 dargestellte Webseite dient zur Konfiguration des Kommunikationsmoduls. Hier können Parameter wie die IP-Adresse, die Subnetzmaske, das Standard-Gateway und der DNS-Server eingestellt werden.

Bild 6-4: Beispiel einer Webseite zum Einstellen von Netzwerkparametern

- **Content-Frame**

```
<!-- Überschrift der Webseite -->
<div id="content">
<h1>Netzwerkeinstellungen</h1>
<br/>

<!-- Beginn der Tabelle -->
<table>
<tr>

<!-- Netzwerkparameter -->
<td id="td_left">
IP-Adresse:
</td>

<!-- Wert -->
<td id="td_left">
192.168.1.123
</td>
<td id="td_left">

<!-- Schaltfläche -->
<input type="submit" name="btn_ip" value="Edit">
</td>
</tr>

<!-- Beginn einer neuen Tabellenzeile -->
<tr>
<td id="td_left">
Subnetzmaske:
</td>
```

```
<td id="td_left">
255.255.255.0
</td>
<td id="td_left">
<input type="submit" name="btn_subnet" value="Edit">
</td>
</tr>

<!-- Beginn einer neuen Tabellenzeile -->
<tr>
<td id="td_left">
Standard-Gateway:
</td>
<td id="td_left">
192.168.1.1
</td>
<td id="td_left">
<input type="submit" name="btn_gateway" value="Edit">
</td>
</tr>

<!-- Beginn einer neuen Tabellenzeile -->
<tr>
<td id="td_left">
DNS-Server:
</td>
<td id="td_left">
192.168.1.1
</td>
<td id="td_left">
<input type="submit" name="btn_dns" value="Edit">
</td>
</tr>

<!-- Ende der Tabelle -->
</table>
</div>

<!-- Ende der Webseite -->
</body>
</html>
```

Listing 6-6: Content-Frame der Webseite zum Einstellen von Netzwerkparametern

6.2 Datenübertragung per Chunked Transfer-Encoding

Eines der Kernziele dieser Arbeit ist die Verarbeitung von großen Datenmengen beim Erstellen und Versenden von Webseiten auf dem Mikrocontroller. Hierzu wird das Verfahren des *Chunked Transfer-Encodings* angewendet (siehe Kapitel 4.3.3). Um die Arbeitsweise des Webservers bei der Datenübertragung per *Chunked Transfer-Encoding* zu erläutern, ist im Folgenden das Übertragungsprotokoll einer Antwort-Nachricht in Auszügen erläutert. Daran ist zu erkennen, wie die Nachricht in Form von mehreren Einzelblöcken an den Client-Rechner versendet wird.

In diesem Beispiel fragt ein Client-Rechner die in Kapitel 6.1.2 dargestellte Startseite des Webservers an. Der Webserver sendet die angeforderten Daten in mehreren Blöcken an den Client-Rechner.
Aus Gründen der Übersichtlichkeit werden nicht alle Blöcke der Nachricht aufgeführt. Sollten Blöcke ausgelassen sein, ist dies in der Beschreibung vermerkt. Das Listing der gesamten Nachricht ist in Anhang B.3 angefügt.

- **Header der Webseite**

Der Nachrichten-Header erhält als einziger Block keine Längenangabe, sondern wird durch eine Leerzeile terminiert. Im Feld *Transfer-Encoding* wird das Übertragungsverfahren festgelegt. Alle nachfolgenden Nachrichtenblöcke werden mit einer hexadezimalen Längenangabe eingeleitet.

```
HTTP/1.1 200 OK
Server: 8051 Webserver
Content-Type: text/html
Transfer-Encoding: chunked <!-- Angabe des Übertragungsverfahrens -->
<!-- Eine Leerzeile terminiert den Header -->
```

Listing 6-7: Header der Webseite

- **Head der Webseite**

Der Head (Kopf) der Nachricht ist als erster Block mit einer hexadezimalen Längenangabe eingeleitet. Er enthält Informationen zum Webseitenformat und den Webseitentitel.

```
010D <!-- Länge des Blocks -->
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" lang="de">
<head>
<title>Startseite</title> <!-- Titel der Webseite -->
```

Listing 6-8: Head der Webseite

- **Stylesheets**

Stylesheets werden in mehreren Blöcken erzeugt. Der erste Block öffnet den Stylesheet-Bereich, die darauffolgenden Blöcke erzeugen die einzelnen Stylesheet-Einträge und der letzte Block schließt den Stylesheet-Bereich ab. Aus Gründen der Übersichtlichkeit sind nur zwei Stylesheet-Einträge aufgeführt. Die Webseite enthält mehrere Stylesheet-Einträge.

```
0028
<style type="TEXT/CSS" MEDIA="SCREEN">
```

Listing 6-9: Beginn des Stylesheet-Bereichs

```
00B6
ul#list_menue { float: left; width: 10.5em; background-color:#DDE5FF;
margin:0; border: 1px solid silver; color: #40586F; height: 30em; }
```

Listing 6-10: Stylesheet-Eintrag 1

```
0093
div#content { margin-left: 12em; height: 30em; padding-top: 1em;
padding-left:1em;padding-right: 1em; border: 1px solid silver; max-
width:50em; }
```

Listing 6-11: Stylesheet-Eintrag 2

```
001B
</style>
</head>
<body>
```

Listing 6-12: Ende des Stylesheet-Bereichs, Beginn des Webseiten-Bodys

- **Navigationsmenü**

Das Navigationsmenü ist wie in Kapitel 6.1.1 bereits erläutert als Liste realisiert. Diese Liste wird ähnlich dem Stylesheet-Bereich auch in mehreren Blöcken erzeugt. Aus Gründen der Übersichtlichkeit ist hier nur ein Listeneintrag aufgeführt.

```
001C
<ul id="list_menue">
<li>
```

Listing 6-13: Liste öffnen

```
000B
Allgemein
```

Listing 6-14: Listeneintrag erzeugen

```
000E
</li>
</ul>
```

Listing 6-15: Liste abschließen

- **Content-Frame**

Der Content-Frame dient zur Darstellung der angeforderten Informationen. Der den Content-Frame einschließende Rahmen ist über einen formatierten Container realisiert.

```
0014
<div id="content"> <!-- Beginn des Formatierungs-Containers -->
```

Listing 6-16: Formatierungs-Container öffnen

```
0029
<h1>Startseite des 8051 Webservers</h1>
```

Listing 6-17: Überschrift der Webseite erzeugen

```
0045
<h2>Herzlich Willkommen auf der Startseite des 8051 Webservers</h2>
```

Listing 6-18: Weitere Überschrift erzeugen

```
0038
Hier k&#246nnte eine Beschreibung der Webseite stehen.
```

Listing 6-19: Beschreibungstext erzeugen

```
0008
</div> <!-- Ende des Formatierungs-Containers -->
```

Listing 6-20: Container schließen

- **Übertragung abschließen**

```
0
<!-- Eine Null gefolgt von einer Leerzeile terminiert die Nachricht -->
```

Listing 6-21: Übertragung abschließen

6.3 Nichtblockierende Softwarestrukturen / Multitasking

Ein weiteres Kernziel dieser Arbeit ist die Entwicklung effizienter, nichtblockierender Softwarestrukturen. Dies ist besonders wichtig, da der Webserver auf einem Mikrocontroller eingesetzt wird, der zusätzlich zu der Webserveranwendung noch weitere Aufgaben ausführt. Um dies sicherzustellen sind Softwarestrukturen implementiert, die ein *kooperatives Multitasking* auf dem Mikrocontroller ermöglichen. Wie in Kapitel 5.3.5 und im Handbuch zum Webserver (siehe Anhang A) erläutert, kann der Webserver trotzdem auch in einem *blockierenden* Übertragungsmodus

betrieben werden. Ist kein Multitasking erforderlich, kann so die Webseitenprogrammierung noch effizienter werden.
Um den Erfolg der nichtblockierenden Strukturen aufzuzeigen, wird zusätzlich zu der Webserveranwendung ein weiterer Prozess in die Main-Funktion des Webserver-Chips integriert. Dieser soll den Arbeitsprozess darstellen, den der Webserver-Chip im späteren Betrieb primär auszuführen hat. Die While-Schleife der Main-Funktion ist in Listing 6-22 dargestellt. Bei jedem Durchlauf wird lediglich ein Ausgangs-Pin umgeschaltet.

```
while (1)
{
     // Webserver Main-Funktion
     Server_Main();

     // Ausgangs-Pin umschalten
     if (PIN_B1 == 0)
     {
          PIN_B1 = 1;
     }
     else
     {
          PIN_B1 = 0;
     }
}
```

Listing 6-22: Main-Funktion des Mikrocontrollers

Im Folgenden ist nun dargestellt, wie der Arbeitsprozess durch die Anfrage eines Client-Rechners beeinflusst wird. Hierzu wird der Signalverlauf am Ausgangs-Pin mit einem Oszilloskop analysiert. Um den Vorgang mit dem Oszilloskop aufzuzeichnen wird ein Pulsweiten-Trigger verwendet. Dieser reagiert sobald ein Schwellenwert für die Pulsdauer überschritten wird.
Die Messung wird sowohl für den blockierenden als auch den nichtblockierenden Übertragungsmodus durchgeführt um die Ergebnisse anschließend gegenüberstellen zu können.
Bild 6-5 zeigt den Signalverlauf bei blockierendem Übertragungsmodus. Es ist zu erkennen, dass der Arbeitsprozess für die Dauer von 77,1 ms unterbrochen wird (siehe roten Kasten). Der Arbeitsprozess ist somit für einen langen Zeitraum blockiert. Ein Mikroprozessor könnte in dieser Zeit viele weitere Aufgaben ausführen, woran er durch die Webserveranwendung gehindert wird.

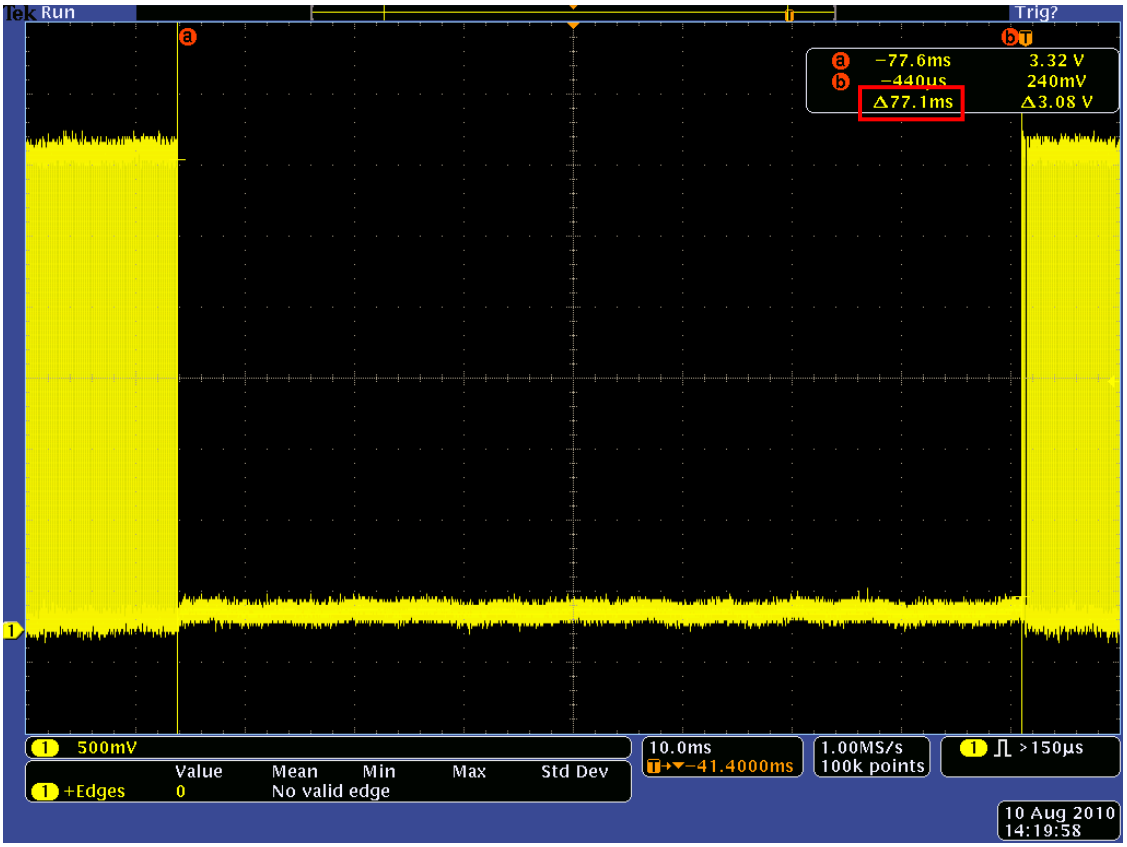

Bild 6-5: Signalverlauf bei blockierendem Übertragungsmodus

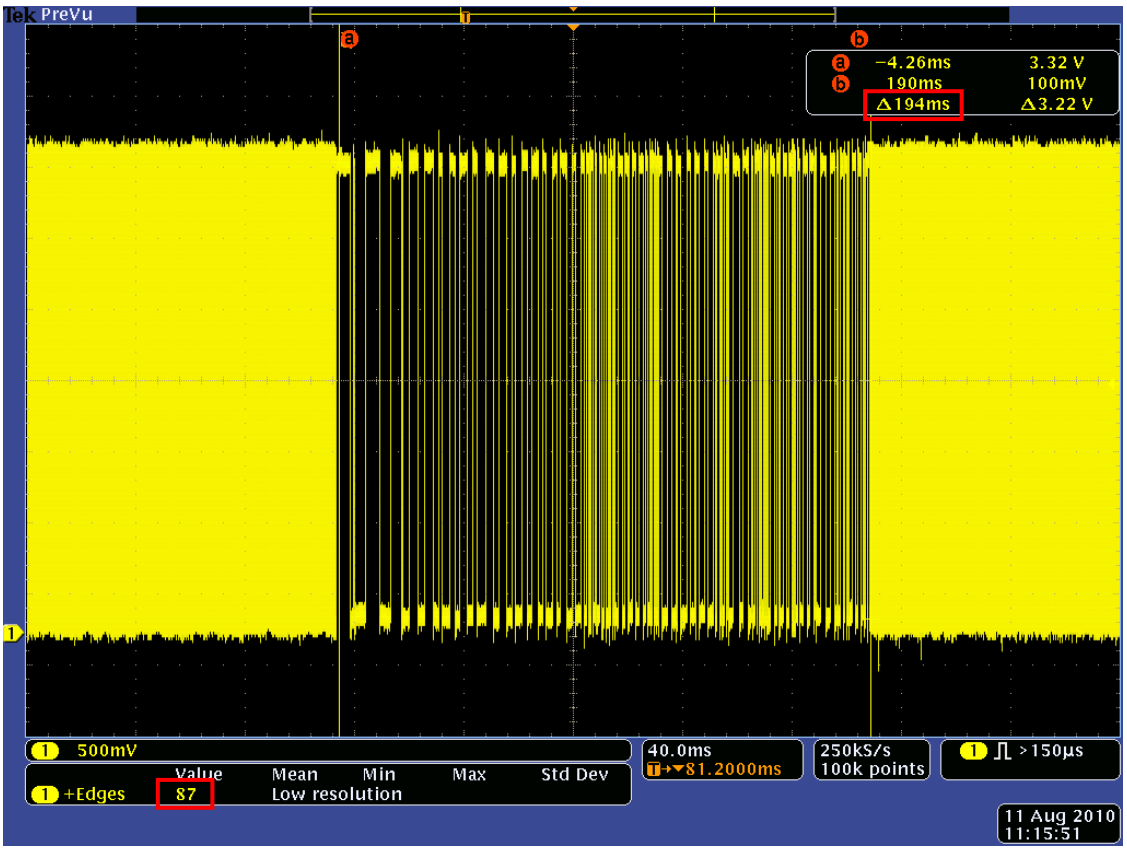

Bild 6-6: Signalverlauf bei nichtblockierendem Übertragungsmodus

In Bild 6-6 ist der Signalverlauf bei nichtblockierendem Übertragungsmodus abgebildet. Die Verarbeitung der Anfrage dauert mit 194 ms (siehe roter Kasten oben) zwar deutlich länger als beim blockierenden Übertragungsmodus, der Arbeitsprozess wird während dieser Phase jedoch 174-mal ausgeführt. Der Wert im unteren roten Kasten zeigt die Anzahl an positiven Flanken während der Anfrage. Da zu jeder positiven Flanke auch eine negative gehört, muss dieser Wert verdoppelt werden.

Um die maximale Dauer zu ermitteln, die der Arbeitsprozess unterbrochen wird, ist in Bild 6-7 das zuvor gezeigte Signal mit einem größeren Maßstab abgebildet. Die maximale Verzögerungszeit (diese entsteht in der Regel beim Eintritt in die Webserveranwendung) beträgt 5,46 ms.

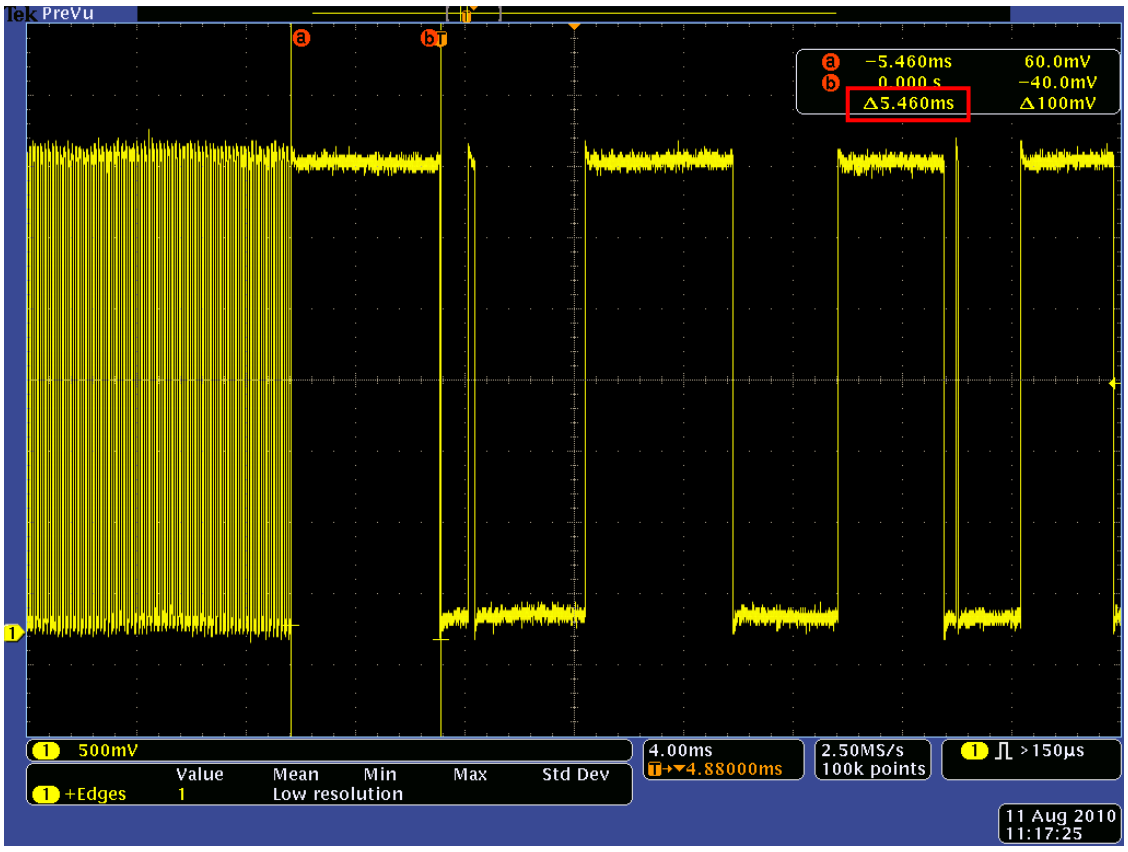

Bild 6-7: Maximale Verzögerung bei nichtblockierendem Übertragungsmodus

Die hier ermittelten Werte bestätigen die erfolgreiche Implementierung eines multitaskingfähigen Systems. Beim nichtblockierenden Übertragungsmodus ist die Dauer der Unterbrechung signifikant geringer als beim blockierenden. Sie ist ungefähr vierzehnmal kürzer.

6.4 Programmspeicherauslastung

Eine geringe Programmspeicherauslastung ist ebenfalls eines der zuvor definierten Ziele dieser Arbeit. Die Webserveranwendung soll im späteren Betrieb auf einem Mikrocontroller implementiert werden, welcher primär andere Prozesse steuern muss. Aus diesem Grund muss die Webserveranwendung zum einen durch nichtblockierende Funktionen realisiert sein, zum anderen darf die Implementierung der Anwendung nicht zu viel Programmspeicher belegen.
Um den Speicherbedarf der Webserveranwendung zu ermitteln, ist in Bild 6-8 und Bild 6-9 die Speicherauslastung des 8051F120 mit und ohne Webservermodul dargestellt. Die Bilder zeigen je einen Screenshot des Programms „8051_IDE“, welches in diesem Projekt als integrierte Entwicklungsumgebung dient. In dem Programmfenster ist der Inhalt der Datei *output.mem* dargestellt, die nach einem Build-Vorgang erzeugt wird. In dem rot gekennzeichneten Kasten lässt sich der verwendete Programmspeicher ablesen.

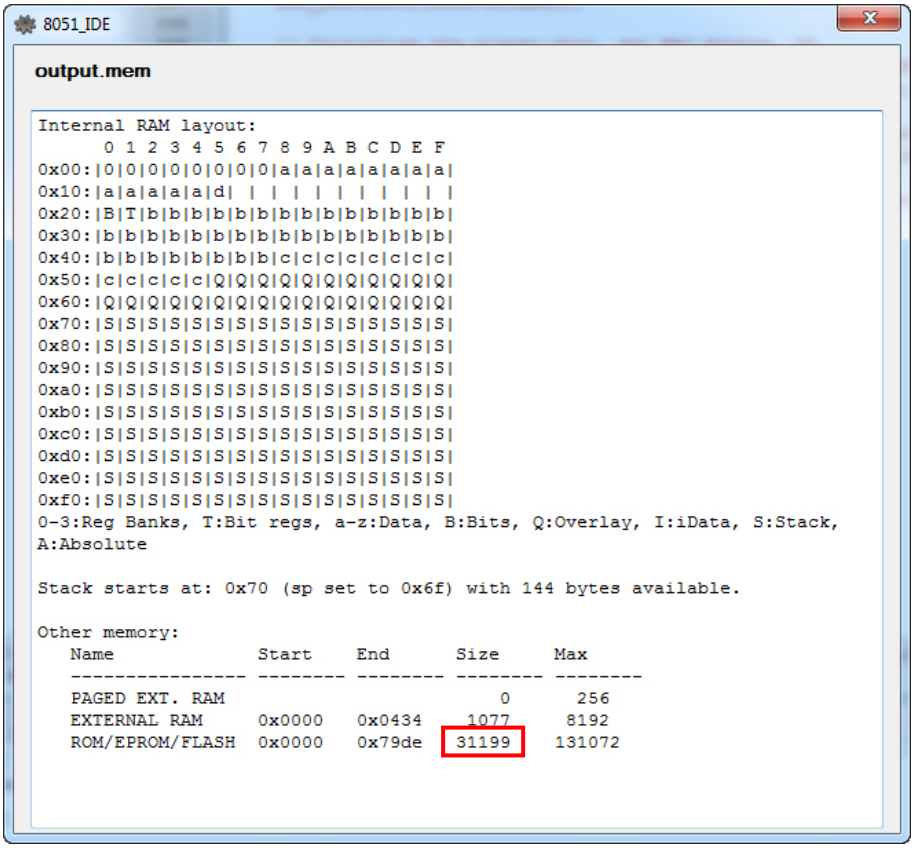

Bild 6-8: Speicherauslastung mit eingebundenem Webservermodul

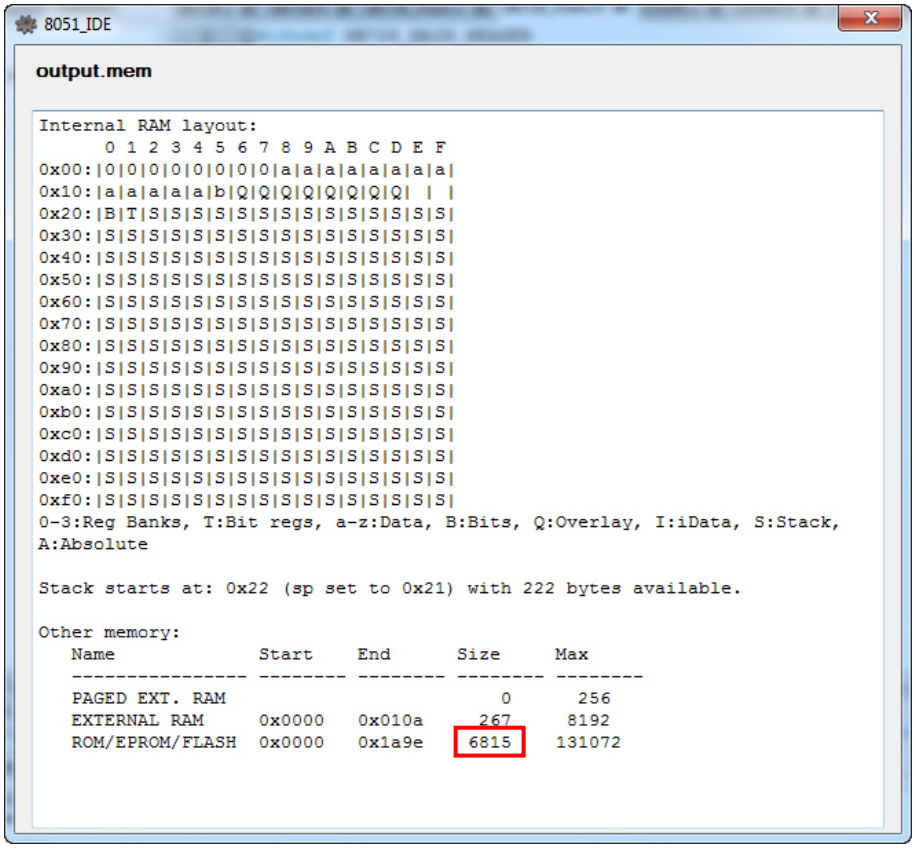

Bild 6-9: Speicherauslastung ohne eingebundenes Webservermodul

Der vom Webservermodul benötigte Programmspeicher wird durch die Subtraktion der beiden aufgeführten Werte berechnet. Formel 6-1 zeigt die Berechnung des Absolutwerts. Die Berechnung des relativen Speicherbedarfs ist in Formel 6-2 dargestellt.

$$\begin{aligned} \text{Speicherbdarf}_{\text{Webserver}} &= \text{Speicherbedarf}_{\text{Gesamt}} - \text{Speicherbedarf}_{\text{Basismodul}} \\ &= 31199\ \text{Byte} - 6815\ \text{Byte} = 24384\ \text{Byte} \end{aligned} \tag{6-1}$$

$$\begin{aligned} \text{Speicherbedarf}_{\text{Webserver\%}} &= \frac{\text{Speicherbedarf}_{\text{Webserver}}}{\text{Gesamtspeicher}} * 100\% \\ &= \frac{24384\ \text{Byte}}{128\ \text{kByte}} * 100\% = 18{,}6\% \end{aligned} \tag{6-2}$$

Der Speicherbedarf des Webservermoduls beträgt 24384 Byte. Dies entspricht einer relativen Speicherauslastung von 18,6%. Diese Werte stellen ein gutes Ergebnis dar, da der verbleibende Speicherplatz ausreichend ist, auch speicherintensive Applikationen auf dem Webserver-Chip zu programmieren.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

In dieser Diplomarbeit ist ein Webserver zur Erzeugung von HTML-Code auf Basis eines 8-bit Mikrocontrollers vom Typ 8051F120 entstanden. Er ermöglicht einerseits die Netzwerkkommunikation über das standardisierte Netzwerkprotokoll TCP/IP. Andererseits bietet er dem Anwender eine komfortable Möglichkeit, Webseiten zu generieren, zu formatieren und darzustellen. Der Webserver analysiert Anfragen von Client-Rechnern vollautomatisch und stellt dem Anwender die in den Anfragen angeforderten Informationen aufbereitet zur Verfügung.

Kernaspekte der Arbeit waren insbesondere die *Entscheidung für ein geeignetes Hardware-Konzept*, die *Entwicklung effizienter Softwarestrukturen* zur Analyse von Anfragen, die *Verarbeitung großer Datenmengen* bei der Generierung von Webseiten, die Umsetzung eines geforderten Funktionsumfangs zur *Erzeugung von HTML-Elementen* sowie das Erstellen eines *Handbuchs*, welches die Funktionalität des Webservers dokumentiert.

- Eine umfassende Marktanalyse hat ergeben, dass ein fertiges Produkt mit dem geforderten Funktionsumfang nicht auf dem Markt erhältlich ist. Die angebotenen Komplettlösungen bieten weder die nötige Flexibilität noch den gewünschten Umfang an Konfigurationsmöglichkeiten. Eine Neuentwicklung des Webservermoduls war daher der naheliegende Ansatz. Als Entwicklungsansätze standen hierbei hardwareseitig zwei verbreitete Methoden zur Auswahl. Ein Single-Chip-Modul ist zwar kompakter, da ein Multi-Chip-Modul jedoch flexibler einsetzbar ist, wurde dieser Entwicklungsansatz gewählt.

- Zur Analyse der von Client-Rechnern gesendeten Anfragen wurden Algorithmen entwickelt, die es ermöglichen die wesentlichen Informationen aus den Anfragen zu extrahieren und diese dem Anwender zugänglich zu machen. Insbesondere sind dies Algorithmen zur Zeichenkettenanalyse. Im späteren Betrieb wird der Webserver auf einem Mikrocontroller eingesetzt, der zusätzlich zu der Webserveranwendung noch weitere Aufgaben ausführt. Daher sind die Hauptprozesse des Webservers als *nichtblockierend* implementiert. Dies wurde über Zustandsautomaten erreicht, die die Möglichkeit bieten, einen Prozess an einer bestimmten Stelle zu verlassen, andere Aufgaben auszuführen und an die Stelle im Prozess zurückzukehren. Mit Hilfe dieser Strukturen wurde ein *multitaskingfähiges* System implementiert.

- Die Verarbeitung großer Datenmengen stellt für Mikrocontroller mit begrenzten Speicherressourcen ein erhebliches Problem dar. Da Webseiten-Code mitunter mehrere Kilobyte umfasst und der Mikrocontroller nicht so viel Code bevorraten kann, wurde in diesem Projekt ein Verfahren zur blockweisen Datenübertragung, das sogenannte *Chunked Transfer-Encoding*, gewählt. Durch den Einsatz dieses Verfahrens konnte das Problem der Verarbeitung von großen Datenmengen erfolgreich gelöst werden.

- Um dem Anwender die Möglichkeit zu bieten, Elemente wie Eingabefelder, Auswahllisten oder Optionsfelder komfortabel generieren und auf einer Webseite platzieren zu können, wurden geeignete Funktionen auf dem Webserver implementiert. Diese Funktionen erzeugen den nötigen HTML-Code und versenden ihn an den Client-Rechner, der die Webseite in einem Webbrowser darstellt. Der Anwender soll zusätzlich in der Lage sein, Webseiten und darauf platzierte Elemente zu formatieren. Hierzu werden Formatvorlagen, sogenannte *Stylesheets*, eingesetzt. Mit deren Hilfe kann der Anwender verschiedene Formatvorlagen für eine Webseite anlegen und diese den einzelnen Elementen auf der Webseite zuweisen. Die Funktionalität der Stylesheets wird dem Anwender ebenfalls über komfortable Funktionen zugänglich gemacht.

- Um die Funktionsweise des Webservers zu dokumentieren, ist das Handbuch mit dem Titel „Handbuch 8051F120-Webserver“ entstanden. Hierin sind die Initialisierung sowie der Betrieb des Webservers beschrieben. Das Handbuch ist im Anhang dieser Arbeit angefügt.

7.2 Ausblick

Der hier entwickelte Webserver bietet die geforderte Funktionalität und wird in naher Zukunft in einem *IF-Prozessor* eingesetzt. IF-Prozessoren dienen in der Radioastronomie dazu, Zwischenfrequenz-Signale für eine nachfolgende Digitalisierung zu modifizieren. Der Webserver wird dabei zur Fernwartung der Anlage genutzt.
Für die Weiterentwicklung des Webservermoduls liegen zwei Ansätze vor. Es ist geplant den Webserver um ein externes Speichermodul zu erweitern, um so die Möglichkeit zu haben, Bilddateien in eine Webseite einzubinden. Bei diesen Bilddateien könnte es sich um Diagramme erzeugt aus Messdaten oder um Logos handeln. Als Grafikformat würde sich hier Portable Network Graphics (PNG) eignen, welches ein Grafikformat für Rastergrafiken ist.
Weiterhin ist geplant Emails automatisiert mit dem Webserver zu versenden. Diese Emails könnten tägliche Statusmeldungen oder auch spezielle Ereignismeldungen beinhalten. Hierzu müsste dem Webserver die Funktionalität implementiert werden, mit einem SMTP-Server zu kommunizieren.

Anhang

A Handbuch zum 8051F120-Webserver

Handbuch

8051F120-Webserver

Version 1.0.0

A.1 Versionsverlauf

Tabelle 7-1: Versionsverlauf des 8051F120-Handbuchs

Version	Datum	Beschreibung
1.0.0	02. August, 2010	Handbuch erstellt

A.2 Einleitung

Dieses Handbuch beschreibt die Konfiguration und Anwendung des 8051F120-Webserver-Moduls. Das Modul dient dazu, einen Webserver auf Basis des 8-Bit Mikrocontroller 8051F120 aufzusetzen. Zur Initialisierung des Moduls sowie zur Webseitenprogrammierung bietet der Webserver eine Auswahl an vorgefertigten Funktionen und Makros, welche in diesem Dokument detailliert erläutert werden.

Anmerkung: In diesem Handbuch werden eine Reihe von Funktionen vorgestellt, um das Modul „8051F120-Webserver“ zu konfigurieren und mit seiner Hilfe Webseiten zu erstellen. Das Modul beinhaltet Funktionen, die nötig sind um die Basisfunktionalität des Webservers sicherzustellen und andere, die als optional gelten. Um zu signalisieren, in welchen Bereich eine Funktion fällt, wird folgende Bezeichnung vereinbart:

Basis — Basis-Funktionen

Optional — Optionale Funktionen

A.3 Einbinden der benötigten Dateien

Bestandteil des 8051F120-Webservers sind folgende Dateien:

- w5300.c
- w5300.h
- webserver.c
- webserver.h
- web_stdlib.c
- web_stdlib.h
- io.h

Diese fünf Dateien sind an einem Speicherort zu platzieren, sodass sie vom Compiler eingebunden werden können.

Damit die in diesem Dokument erläuterten Funktionen in Ihrem Projekt verfügbar sind, werden die folgenden zwei Headerfiles in den Programmcode eingebunden.

```
#include <w5300.h> // Einbinden des Kommunikationsmoduls
#include <webserver.h> // Einbinden der Webserveranwendung
```

A.4 Initialisierung des Webservers

Um den Webserver korrekt zu initialisieren, müssen folgende Schritte nacheinander ausgeführt werden. In Kapitel A.6.1 ist der Initialisierungsvorgang zusätzlich anhand eines Beispiels erläutert.

A.4.1 Anlegen der Struct-Variablen zur Initialisierung des Webservers

Folgende Struct-Variablen werden benötigt:

```
t_w5300 WizNet;
t_w5300_socket TCP_53121_01;
```

A.4.2 Reset des W5300

Vor der Initialisierung muss der W5300 in einen definierten Anfangszustand gebracht werden. Dies geschieht über den Pin 66 (/Reset) am W5300. Dieser wird hierzu für 20ms auf Ground gesetzt. Anschließen muss weitere 20ms gewartet werden bis der Chip einen definierten Anfangszustand angenommen hat.

```
// Pseudocode für einen Hardware-Reset des W5300
W5300_Pin66 = 0;
delay(20ms);
W5300_Pin66 = 1;
delay(20ms);
```

A.4.3 Basisinitialisierung des Kommunikationsmoduls

Funktionsname:	w5300_init_base **Basis**
Funktionsaufruf:	w5300_init_base(t_w5300 *t_w5300_Ptr, U16 BaseAddress, U16 MAC_Bottom, U16 MAC_Middle, U16 MAC_Top)
Übergabeparameter:	t_w5300_Ptr: Pointer auf eine Strukturvaliable vom Typ t_w5300
	BaseAddress: Basis-Adresse für das External Data Memory Interface, Bsp.: 0xFC00
	IP_Address: IP-Adresse, Bsp.: 0x86681CD1
	Netmask: Subnetzmaske, Bsp.: 0xFFFFFF00
	Gateway: Stanard-Gateway, Bsp: 0x86681D17
	MAC_Bottom:

	LSB und Byte 1 der MAC-Adresse, Bsp.: 0x2005
	MAC_Middle: Byte 2 und Byte 3 der MAC-Adresse, Bsp.: 0xC2AB
	MAC_Top: Byte 4 und MSB der MAC-Adresse, Bsp.: 0xDD01
Rückgabeparameter:	ret: 0 Initialisierung erfolgreich -1 Fehler bei der Initialisierung
Beispielaufruf:	`ret = w5300_init_base(&t_w5300_Ptr, 0xFC00, 0x86681CD1, 0xFFFFFF00, 0x86681D17, 0x2005, 0xC2AB, 0xDD01);`

A.4.4 Initialisierung des Verbindungs-Sockets

Funktionsname:	w5300_init_socket **Basis**
Funktionsaufruf:	`w5300_init_socket(t_w5300_socket SockPtr, t_w5300 *P_w5300, U8 Socketno, U16 Portno)`
Übergabeparameter:	SockPtr: Pointer auf eine Strukturvaliable vom Typ t_w5300_socket
	P_w5300: Pointer auf eine Strukturvaliable vom Typ t_w5300
	Socketno: Socket-Nummer (0-7), Bsp.: 1
	Portno: Port-Nummer, Bsp.: 1629
Rückgabeparameter:	ret: 0 Initialisierung erfolgreich -1 Fehler bei der Initialisierung
Beispielaufruf:	`ret = w5300_init_socket(&SockPtr, &P_w5300, 1, 1629);`

A.4.5 Initialisierung der Webserveranwendung

Funktionsname:	webs_init_wserver **Basis**
Funktionsaufruf:	`webs_init_wserver(t_w5300_socket *ConnDesc)`
Übergabeparameter:	ConnDesc: Pointer auf eine Strukturvaliable vom Typ t_w5300_socket
Rückgabeparameter:	ret: 0 Initialisierung erfolgreich -1 Fehler bei der Initialisierung

Beispielaufruf:	`ret = webs_init_wserver(&ConnDesc);`

A.5 Verwendung des Webservers

Es besteht die Möglichkeit, den Webserver im *nichtblockierenden* sowie im *blockierenden* Übertragungs-Modus zu betreiben.

Im nichtblockierenden Modus wird die Webseitenprogrammierung über Zustandsautomaten realisiert. Da Webseiten in mehreren Blöcken generiert und an den Client-Rechner versendet werden, kann es zwischen den einzelnen Sendevorgängen zu Wartezeiten kommen. Während dieser Wartezeiten können keine weiteren Daten vom Webserver verarbeitet werden. Bei der Verwendung des nichtblockierenden Modus besteht nun die Möglichkeit, den Zustandsautomat zur Erzeugung der Webseite bei Wartezeiten zu verlassen, andere Aufgaben auszuführen und die Erzeugung der Webseite später fortzusetzen.

Der blockierende Modus bietet die zuvor geschilderte Funktionalität nicht. Bei Wartezeiten verweilt der Mikrocontroller in der Schleife zur Erzeugung der Webseite. Wie in späteren Beispielen zu sehen ist, können Webseiten unter Verwendung des nichtblockierenden Modus schlanker programmiert werden, da die Zustandsautomat mitunter sehr aufwendiger switch-case-Anweisungen bedürfen. Hier ist für jeden Anwendungsfall abzuwägen, welcher Ansatz sinnvoll ist.

Nähere Informationen zu den Übertragungsmodi sind in Kapitel A.6 in Form einiger Beispiele aufgeführt.

A.5.1 Setzen des Konfigurationsregisters

Das Konfigurationsregister dient zur Konfiguration des Webservers. In dieser Version wird hier nur der Übertragungsmodus festgelegt.

Funktionsname:	`w5300_set_config_reg` Basis
Funktionsaufruf:	`w5300_set_config_reg(t_w5300_socket *SockPtr, U8 Reg_Value)`
Übergabeparameter:	`SockPtr:` Pointer auf eine Strukturvaliable vom Typ t_w5300_socket
	`Reg_Value:` 8-Bit Register-Variable Registerbelegung: Bit 7-1: In dieser Version nicht verwendet Bit 0: Blockierender(1) oder nichtblockierenden(0) Modus

Rückgabeparameter:	ret: 0 Setzen des Registers erfolgreich -1 Fehler beim Setzen des Registers
Beispielaufruf:	`ret = w5300_set_config_reg(&SockPtr, 0x01);`

A.5.2 Überprüfen ob eine Anfrage an den Webserver vorliegt

HTTP-Verbindungen werden in der Regel clientseitig initialisiert. Hierzu sendet der Client-Rechner eine Anfrage an den Webserver. Um zu überprüfen, ob eine Anfrage beim Webserver vorliegt, ist die Funktion *check_for_request* im Webservermodul implementiert.

Funktionsname:	`check_for_request` Basis
Funktionsaufruf:	`check_for_request(char *Webpage, int Len_Webpage)`
Übergabeparameter:	Webpage: In diesem Char-Array speichert der Webserver die vom Client-Rechner angeforderte Webseite
	Len_Webpage: Länge des Char-Arrays Webpage, Tip: `sizeof(Webpage)`
Rückgabeparameter:	ret: 0 Es liegt keine Anfrage vor 1 Es liegt ein GET-Request vor, es wurden keine Datenpaare mitgesendet 2 Es liegt ein GET-Request vor, es wurden Datenpaare mitgesende 3 Es liegt ein POST-Request vor, es wurden Datenpaare mitgesendet
Beispielaufruf:	`ret = check_for_request(Webpage, sizeof(Webpage));`

A.5.3 Datenpaare abrufen

Wurde ein GET- oder ein POST-Request mit angefügten Datenpaaren an den Webserver gerichtet, können diese Datenpaare über die Funktion *get_cgi_data* beim Webserver abgeholt werden:

Funktionsname:	`get_cgi_data` Optional
Funktionsaufruf:	`get_cgi_data( char *Name, char *Value, int Len_Name, int Len_Value)`
Übergabeparameter:	Name: Char-Array für den Parameternamen
	Value: Char-Array für den Parameterwert

	Len_Name: Länge des Char-Arrays „Name“
	Len_Value: Länge des Char-Arrays „Value“
Rückgabeparameter:	ret: 0 Keine Datenpaare vorhanden 1 Datenpaar erfolgreich abgeholt, weitere Datenpaare vorhanden 2 Datenpaar erfolgreich abgeholt, keine weitere Datenpaare vorhanden -1 Fehler
Beispielaufruf:	`ret = get_cgi_data(Name, Value, sizeof(Name), sizeof(Value));`

A.5.4 Grundgerüst einer Webseite erstellen

Zu Beginn der Webseitenprogrammierung muss das Grundgerüst der Webseite erstellt werden. Dieses Grundgerüst beinhaltet den Header der Webseite sowie den Head-Bereich mit dem Webseitentitel. Die eigentlichen Nutzdaten werden dann im nächsten Schritt auf der Webseite platziert.

A.5.4.1 Header erstellen

Funktionsname:	`create_header` Basis
Funktionsaufruf:	`create_header()`
Übergabeparameter:	Keine
Beispielaufruf:	`ret = create_header();`

A.5.4.2 Kopf der Webseite öffnen

Der sogenannte Kopf (Head) einer Webseite beinhaltet den Webseitentitel und die Formatvorlagen (Stylesheets) welche auf der Webseite angewendet werden sollen. Da die Stylesheets mitunter sehr lang sind, wird der Head-Bereich in mehreren Schritten erzeugt. Die Funktion *create_head_start* öffnet den Head-Bereich und erst nachdem die Stylesheets eingefügt wurden wird er über die Funktion *create_head_end* abgeschlossen.

Funktionsname:	`create_head_start` Basis
Funktionsaufruf:	`create_head_start(char *Webpage_Title)`
Übergabeparameter:	Webpage_Title: Titel der Webseite
Beispielaufruf:	`ret = create_head_start(“Startseite des Webservers”);`

A.5.4.3 Stylesheets erstellen

Stylesheets dienen zum Formatieren von Webseiten. Der Stylesheet-Bereich wird in drei Schritten angelegt: „Öffnen des Stylesheet-Bereichs“, „Stylesheets eintragen“ und „Schließen des Stylesheet-Bereichs“. Die Verwendung von Stylesheets ist optional.

- Stylesheet-Bereich öffnen

Funktionsname:	`create_stylesheet_start` Optional
Funktionsaufruf:	`create_stylesheet_start()`
Übergabeparameter:	Keine
Beispielaufruf:	`ret = create_stylesheet_start();`

- Stylesheet-Einträge erzeugen

Funktionsname:	`create_stylesheet_entry` Optional
Funktionsaufruf:	`create_stylesheet_entry(char *Style_Name, char *Entry)`
Übergabeparameter:	Style_Name: Name des Stylesheet-Eintrag
	Entry: Einzelner Stylesheet-Eintrag
Beispielaufruf:	`ret = create_stylesheet_entry("table", "font-size:20px; font-family:Arial; margin-left:2em; margin-top:2em; width:25em;");`

- Stylesheet-Bereich abschließen

Funktionsname:	`create_stylesheet_end` Optional
Funktionsaufruf:	`create_stylesheet_end()`
Übergabeparameter:	Keine
Beispielaufruf:	`ret = create_stylesheet_end();`

A.5.4.4 Kopf der Webseite abschließen und Webseiten-Body öffnen

Funktionsname:	`create_head_end` Basis
Funktionsaufruf:	`create_head_end()`
Übergabeparameter:	Keine
Beispielaufruf:	`ret = create_head_end();`

A.5.4.5 Webseite abschließen

Nachdem der Body-Inhalt programmiert wurde, muss die Webseite abgeschlossen werden. Dies geschieht mittels der folgenden Funktion.

Funktionsname:	`create_document_end` Basis
Funktionsaufruf:	`create_document_end()`
Übergabeparameter:	Keine
Beispielaufruf:	`ret = create_document_end();`

A.5.5 Formulare erzeugen

Mit Hilfe von Formulardaten können zum Beispiel Eingaben eines Benutzers in einem Eingabefeld an den Webserver gesendet werden. Hierzu öffnet man ein Formular, definiert die Anfrage-Methode (Request-Methode), erstellt einen Button und schließt das Formular ab. Durch Betätigen des Buttons werden die Daten aus dem Eingabefeld an den Webserver gesendet.

- Formular öffnen

Funktionsname:	`create_formular_start` Optional
Funktionsaufruf:	`create_formular_start(char *Action, char *Method)`
Übergabeparameter:	Action: URL an die die Formulardaten beim Absenden des Formulars übertragen werden sollen
	Method: Gibt die HTTP-Übertragungsmethode an, Bsp.: GET
Beispielaufruf:	`ret = create_formular_start("http://www.beispiel.de", "GET");`

- Formular abschließen

Funktionsname:	`create_formular_end` Optional
Funktionsaufruf:	`create_formular_end()`
Übergabeparameter:	Keine
Beispielaufruf:	`ret = create_formular_end();`

A.5.6 Elemente auf der Webseite platzieren

Tabelle 7-2 zeigt alle HTML-Elemente, die mit diesem Webserver erzeugt werden können. Als zusätzliche Gliederungselemente können *Container* verwendet werden.

In den folgenden Kapiteln wird gezeigt, wie die einzelnen Elemente zu programmieren sind.

Tabelle 7-2: Verfügbare HTML-Elemente

Element	*HTML-Bezeichnung*	*Beispiel*
Einzeiliges Eingabefeld	Inputfield	Test
Mehrzeiliges Eingabefeld	Multiline-Inputfield	Zeile_1 Zeile_2
Auswahlliste	List-Box	Auswahl_1 Auswahl_1 Auswahl_2
Tabelle	Table	Zelle_1 | Zelle_2
Eingabeknopf	Button	Button_1
Text	Text	Text auf einer Homepage
Überschrift	Headline	**Webseitenüberschrift**
Optionsfeld	Radio-Button	
Auswahlfeld	Check-Box	
Liste	List	• **Listeneintrag 1** • **Listeneintrag 2**
Verknüpfung	Link	http://www.beispiel.de

A.5.6.1 Einzeiliges Eingabefeld / Inputfield

Funktionsname:	`create_inputfield` Optional
Funktionsaufruf:	`create_inputfield(char *Name, char *Size)`
Übergabeparameter:	`Name:` `Interner Variablenname des Eingabefelds`
	`Size:` `Größe des Eingabefelds (Anzahl der Zeichen)`
Beispielaufruf:	`ret = create_inputfield("I_Field_1", "30");`

A.5.6.2 Mehrzeiliges Eingabefeld / Multiline-Inputfield

Funktionsname:	`create_multiline_inputfield` Optional
Funktionsaufruf:	`create_multiline_inputfield(char *Name, char *Cols, char *Rows)`
Übergabeparameter:	`Name:` Interner Variablenname des Eingabefelds
	`Cols:` Anzahl der Spalten
	`Rows:` Anzahl der Zeilen
Beispielaufruf:	`ret = create_multiline_inputfield("Multi_I_Field_1", "1", "15");`

A.5.6.3 Auswahlliste / List-Box

Auswahllisten werden in drei Schritten programmiert:

- Auswahlliste öffnen

Funktionsname:	`create_listbox_start` Optional
Funktionsaufruf:	`create_listbox_start(char *Name, char *Size)`
Übergabeparameter:	`Name:` Interner Variablenname der List-Box
	`Size:` Anzahl der Einträge in der List-Box
Beispielaufruf:	`ret = create_listbox_start("LB_1", "5");`

- Eintrag zur Auswahlliste hinzufügen

Funktionsname:	`create_listbox_entry` Optional
Funktionsaufruf:	`create_listbox_entry(char *Style, char *Entry)`
Übergabeparameter:	`Style:` Anzuwendende Formatvorlage
	`Entry:` List-Box Eintrag
Beispielaufruf:	`ret = create_listbox_entry("lb_style", "Eintrag_1");`

- Auswahlliste abschließen

Funktionsname:	`create_listbox_end` Optional

Funktionsaufruf:	`create_listbox_end()`
Übergabeparameter:	`Keine:`
Beispielaufruf:	`ret = create_listbox_end();`

A.5.6.4 Tabelle / Table

Tabellen werden über fünf Funktionen programmiert. Der eigentliche Zelleninhalt wird nach dem Aufruf der Funktion *create_table_cell_start* eingefügt. Hier können alle hier vorgestellten HTML-Elemente eingefügt werden.

- Tabelle und erste Tabellenzeile öffnen

Die Funktion *create_table_start* öffnet den Tabellenbereich und die erste Tabellenzeile. Um später weitere Tabellenzeilen anzulegen dient die Funktion *create_table_new_row*.

Funktionsname:	`create_table_start` Optional
Funktionsaufruf:	`create_table_start(char *Style)`
Übergabeparameter:	`Style:` `Anzuwendende Formatvorlage`
Beispielaufruf:	`ret = create_table_start("table_style");`

- Weitere Tabellenzeile öffnen

Funktionsname:	`create_table_new_row` Optional
Funktionsaufruf:	`create_table_new_row(char *Style)`
Übergabeparameter:	`Style:` `Anzuwendende Formatvorlage`
Beispielaufruf:	`ret = create_table_new_row("table_row_style");`

- Tabellenzelle öffnen

Funktionsname:	`create_table_cell_start` Optional
Funktionsaufruf:	`create_table_cell_start(char *Style)`
Übergabeparameter:	`Style:` `Anzuwendende Formatvorlage`
Beispielaufruf:	`ret = create_table_cell_start("table_cell_style");`

- Tabellenzelle abschließen

Funktionsname:	`create_table_cell_end` Optional

Funktionsaufruf:	`create_table_cell_end()`
Übergabeparameter:	Keine:
Beispielaufruf:	`ret = create_table_cell_end();`

- Tabelle abschließen

Funktionsname:	`create_table _end` Optional
Funktionsaufruf:	`create_table_end()`
Übergabeparameter:	Keine:
Beispielaufruf:	`ret = create_table_end();`

- Beispielhafte Programmierung einer Tabelle

```
// Tabelle und erste Tabellenzeile öffnen
create_table_start("style_for_table");

// Tabellenzelle öffnen
create_table_cell_start("style_align_right");

// Text in die Zelle eintragen
create_text("Vorname:");

// Zelle abschließen
create_table_cell_end();

// Weitere Zelle in der selben Zeile öffnen
create_table_cell_start("style_align_left");

// Eingabefeld erzeugen
create_inputfield("I_Filed_1", "30");

// Zelle abschließen
create_table_cell_end();

// Neue Tabellenzeile erzeugen
create_table_new_row("");

// Tabellenzelle öffnen
create_table_cell_start("style_align_right");

// Text in die Zelle eintragen
create_text("Nachname:");

// Zelle abschließen
create_table_cell_end();

// Weitere Zelle in der selben Zeile öffnen
create_table_cell_start("style_align_left");

// Eingabefeld erzeugen
create_inputfield("I_Filed_2", "30");
```

```
// Tabelle abschließen
create_table_end();
```

Listing 7-1: Programmieren einer Tabelle

A.5.6.5 Eingabeknopf / Button

Funktionsname:	create_button Optional
Funktionsaufruf:	create_button(char *Type, char *Name, char *Value)
Übergabeparameter:	Type: Eingabeknopftyp, Bsp.: „submit“ oder „reset“
	Name: Interner Variablenname
	Value: Beschriftung auf dem Button
Beispielaufruf:	ret = create_button(“submit”, “btn_1”, “Buttontext”);

A.5.6.6 Text

Funktionsname:	create_text Optional
Funktionsaufruf:	create_text(char *Text)
Übergabeparameter:	Text: Unformatierter Text
Beispielaufruf:	ret = create_text(“Herzlich Willkommen”);

A.5.6.7 Überschrift / Headline

Funktionsname:	create_headline Optional
Funktionsaufruf:	create_headline(char *Text, char *Headline_Level)
Übergabeparameter:	Text: Text der Überschrift
	Headline_Level: Ebene der Überschrift (Zahl von 1 bis 6)
Beispielaufruf:	ret = create_headline(“Webseitenueberschrift”, “2”);

A.5.6.8 Optionsfeld / Radio-Button

Funktionsname:	create_radio_button Optional
Funktionsaufruf:	create_radio_button(char *Name, char *Value)

Übergabeparameter:	Name: Interner Variablenname
	Value: Bezeichnerwert des Optionsfelds
Beispielaufruf:	`ret = create_radio_button("Artikel", "Kondensator");`

A.5.6.9 Auswahlfeld / Checkbox

Funktionsname:	`create_check_box` Optional
Funktionsaufruf:	`create_check_box(char *Name, char *Value)`
Übergabeparameter:	Name: Interner Variablenname
	Value: Bezeichnerwert des Auswahlfelds
Beispielaufruf:	`ret = create_check_box("Artikel", "Widerstand");`

A.5.6.10 Liste / List

Die folgenden drei Funktionen dienen dazu, ein Grundgerüst einer Liste zu erstellen. Die eigentlichen Listeneinträge können nach dem Öffnen der Liste bzw. nach dem Vorbereiten neuer Listeneinträge z.B. über die Funktion *create_text* erstellt werden. Die Formatierung der Liste wird über die bereits vorgestellten Stylesheets realisiert.

- Liste öffnen und erste Listeneintrag vorbereiten

Funktionsname:	`create_list_start` Optional
Funktionsaufruf:	`create_list_start(char *Style)`
Übergabeparameter:	Style: Anzuwendende Formatvorlage
Beispielaufruf:	`ret = create_list_start("list_style");`

- Neuen Listeneintrag vorbereiten

Funktionsname:	`create_list_new_entry()` Optional
Funktionsaufruf:	`create_list_new_entry()`
Übergabeparameter:	Keine
Beispielaufruf:	`ret = create_list_new_entry();`

- Liste abschließen

Funktionsname:	create_list_end() Optional
Funktionsaufruf:	create_list_end()
Übergabeparameter:	Keine
Beispielaufruf:	ret = create_list_end();

- Beispielhafte Programmierung einer List

```
// Liste öffnen
create_list_start("style_for_list");

// Listeneintrag erzeugen
create_text("Erster Eintrag");

// Einen weiteren Listeneintrag vorbereiten
create_list_new_entry();

// Listeneintrag erzeugen
create_text("Zweiter Eintrag");

// Einen weiteren Listeneintrag vorbereiten
create_list_new_entry();

// Listeneintrag erzeugen
create_text("Dritter Eintrag");

// Liste abschließen
create_list_end()
```

Listing 7-2: Programmieren einer Liste

A.5.6.11 Verknüpfung / Link

Funktionsname:	create_link Optional
Funktionsaufruf:	create_link(char *Link, char *Text)
Übergabeparameter:	Link: Ziel der Verknüpfung
	Text: Linktext
Beispielaufruf:	ret = create_link("http://www.h-brs.de", "H-BRS");

A.5.6.12 Container

- Conainer öffnen

Funktionsname:	create_container_open Optional

Funktionsaufruf:	`create_container_open(char *Container_type, char *Style)`
Übergabeparameter:	`Container_type:` `Typ des Containers, Bsp.: div`
	`Style:` `Anzuwendende Formatvorlage`
Beispielaufruf:	`ret = create_container_open("div", "Container_Style");`

- Container abschließen

Funktionsname:	`create_container_end` Optional
Funktionsaufruf:	`create_container_end()`
Übergabeparameter:	`Keine`
Beispielaufruf:	`ret = create_container_end();`

A.5.7 Die verschiedenen Übertragungsmodi

Die Funktionen zur Erzeugung von HTML-Code sind, wie in den vorigen Kapiteln bereits dargestellt, identisch aufgebaut. Die generelle Syntax ist Folgende:

```
int ret = create_<html_element>(<Parameter_0>,..., <Parameter_n>)
```

Abhängig davon welcher Übertragungsmodus verwendet wird (blockierend oder nichtblockierend) variiert jedoch der Rückgabewert dieser Funktionen.

- Nichtblockierender Übertragungsmodus

Rückgabeparameter:	`ret:`
	`0 Sendevorgang aktiv, Wartezeit`
	`1 Sendevorgang erfolgreich abgeschlossen`
	`-1 Fehler, Sendevorgang unvollständig abgebrochen`

- Blockierender Übertragungsmodus

Rückgabeparameter:	`ret:`
	`1 Sendevorgang erfolgreich abgeschlossen`
	`-1 Fehler, Sendevorgang unvollständig abgebrochen`

Wie die verschiedenen Modi anzuwenden sind ist im nächsten Kapitel anhand einiger Beispiele erläutert.

A.6 Beispiele

A.6.1 Beispiel 1: Einbinden des Webserver-Moduls und Implementieren einer Request-Abfrage

In diesem Beispiel ist dargestellt, wie das Webservermodul in ein Projekt eingebunden wird und wie eine generelle Programmstruktur zur Request-Abfrage aussehen könnte.

```
// Einbinden des Webserver-Moduls
#include <w5300.h>
#include <webserver.h>

// Main-Funktion des Webservers bekanntmachen
void Server_Main();

// Main-Funktion des Projekts
void main()
{
     // Struct-Variablen anlegen
     t_w5300 WizNet;
     t_w5300_socket TCP_53121_01;

     // Rueckgabe-Variable anlegen
     int Ret = 0;

     // HardwareReset W5300
     WNet_Reset = 0;
     mdelay(20);
     WNet_Reset = 1;
     mdelay(20);

     // W5300 initialisieren
     Ret = w5300_init_base(  &WizNet,
                    0xFC00,        // Base-Address
                    0x86681CD1,    // IP_Addr:    134.104.28.209
                    0xFFFFFF00,    // Subnet:     255.255.255.0
                    0x86681D17,    // Gateway:    134.104.29.23
                    0x2005,        // MAC Top:    20-05
                    0xC2AB,        // MAC_Mid:    C2-AB
                    0xDD01 ); // MAC_Low:    DD-01-->20-05-C2-AB-DD-01

     // TCP-socket initialisieren
     Ret = w5300_init_socket(&TCP_53121_01,
                    &WizNet,
                    0,             // Socketnumber
                    53121 );       // Portnumber

     // Webserveranwendung initialisieren
     Ret = webs_init_wserver(&TCP_53121_01);

     // Config-Register setzen
     w5300_set_config_reg(&TCP_53121_01, 0x00); // nichtblockierend

     while (1)
     {
```

```
          // In der Funktion Server_Main() findet die
          // Request-Abfrage statt
          Server_Main();

          // Weitere Aufgaben ausführen...
     }
}

// Die Funktion Server_Main() ist die Main-Funktion des Webservers
void Server_Main()
{
     // Variablendeklaration
     char Web_Page[64] = "";
     char Parameter_Name[64] = "";
     char Parameter_Value[64] = "";
     int Ret = 0;

     // Überprüfen ob eine Anfrage vorliegt
     Ret = check_for_request(WebPage, sizeof(WebPage));

     if (Ret == 1) // GET-Request ohne Datenpaare liegt vor
     {
          // Anfrage anhand der angeforderten Webseite auswerten und
          // Antwort erzeugen
     }
     else if (Ret == 2) // GET-Request mit Datenpaaren
     {
          // Datenpaare anfordern
          Ret = get_cgi_data(Parameter_Name, Parameter_Value,
                         sizeof(Parameter_Name),
sizeof(Parameter_Value));

          // Anfrage anhand der angeforderten Webseite und der Datenpaare
          // auswerten und Antwort erzeugen
          // ...

          // Überprüfen ob weitere Datenpaare vorliegen
          if (Ret == 2) // Weitere Datenpaarer vorhanden
          {
               // Datenpaare anfordern und auswerten
          }

          // ...
     }
     else if (Ret == 3) // POST-Request mit Datenpaaren
     {
          // Datenpaare anfordern und auswerten
          // ...
     }
}
```

Listing 7-3: Einbinden des Webservermoduls und Request-Abfrage

A.6.2 Beispiel 2: Webseitenprogrammierung mit nichtblockierendem Übertragungsmodus

Dieses Beispiel baut auf Beispiel 1 (siehe Kapitel A.6.1) auf und erweitert es um die Funktionalität, eine Webseite zu erzeugen. Hierzu wird der nichtblockierende Übertragungsmodus angewendet.

```
// Makro um den Sendefortschritt bei der nichtblockierenden
// Übertragungsmethode zu überprüfen
#define CHECK_SEND_STATE \
if (Ret == 1) \ // Sendevorgang erfolgreich
{ \
     // Zustandsvariable inkrementieren
     state++; \
} \
else if (Ret == 0) \ //Sendevorgang aktiv, Wartezeit
{ \
     // Zustandsautomat verlassen
     break;\
} \
else \ // Fehler bei der Übertragung
{ \
     // Fehlerbehandlung...
     state = 0; \
     break; \
} \

// Die Funktion create_webpage erzeugt eine
// Webseite mit minimalem Funktionsumfang
void create_webpage(void)
{
     // Zustandsvariable der Statemachine
     static int state = 0;
     int Ret = 0;

     // Statemachine zur nichtblockierenden Webseitenerzeugung
     switch (state)
     {
          case(0): // Header erzeugen
               Ret = create_header();

               // Sendestatus überprüfen
               CHECK_SEND_STATE

          case(1): // Head erzeugen
               Ret = create_head_start("8051 Webserver");

               // Sendestatus überprüfen
               CHECK_SEND_STATE

          case(2): // Stylesheet-Bereich öffnen
               Ret = create_stylesheet_start();

               // Sendestatus überprüfen
               CHECK_SEND_STATE
```

```
          case(3): // Stylesheet-Eintrag erzeugen
               Ret = create_stylesheet_entry("body",
                              "background-color:#663333; color:#FFCC99;
                              font-family:Arial;");

               // Sendestatus überprüfen
               CHECK_SEND_STATE

          // Hier könnten weitere Stylesheets eingefügt werden
          // ...
          // ...

          case(4): // Stylesheet-Bereich abschließen
               Ret = create_stylesheet_end();

               // Sendestatus überprüfen
               CHECK_SEND_STATE

          case(5):  // Head-Bereich abschließen und
                    // Nachrichten-Body öffnen
               Ret = create_head_end();

               // Sendestatus überprüfen
               CHECK_SEND_STATE

          // Im Nachrichten-Body können nun die aufgeführten
          // HTML-Elemente platziert werden. In diesem Beispiel wird
          // hier nur eine Überschrift und ein Eingabefeld platziert.

          case(6): // Überschrift erzeugen
               Ret = create_headline("Willkommen auf dem
                                   Webserver_8051F120", "1");

               // Sendestatus überprüfen
               CHECK SEND STATE

          case(7): // Eingabefeld erzeugen
               Ret = create_inputfield("I_Field_1", "30");

               // Sendestatus überprüfen
               CHECK_SEND_STATE

          case(8): // Webseite abschließen
               Ret = create_document_end();

               // Sendestatus überprüfen
               CHECK_SEND_STATE

          default:
               // Standardaktion ausführen ...
     }
}
```

Listing 7-4: Programmierung mit nichtblockierendem Übertragungsmodus

A.6.3 Beispiel 3: Webseitenprogrammierung mit blockierendem Übertragungsmodus

In diesem Beispiel wird die in Beispiel 2 programmierte Webseite mit blockierendem Übertragungsmodus realisiert. Hierbei ist die erste Funktion (*create_header*) so aufgerufen, dass eine Fehlerbehandlung möglich wäre. Dies ist nicht notwendig, wie bei den anderen Funktionsaufrufen zu sehen ist.

```
// Die Funktion create_webpage erzeugt eine
// Webseite mit minimalem Funktionsumfang
void create_webpage()
{
     // Header erzeugen
     if (create_header() == (-1))
     {
          // Fehlerbehandlung
     }

     // Head erzeugen
     Ret = create head start("8051 Webserver");

     // Stylesheet-Bereich oeffnen
     Ret = create_stylesheet_start();

     // Stylesheet-Eintrag erzeugen
     Ret = create_stylesheet_entry("body", "background-color:#663333;
                                   color:#FFCC99; font-family:Arial;");

     // Hier könnten weitere Stylesheets eingefügt werden
     // ...
     // ...

     // Stylesheet-Bereich schließen
     Ret = create_stylesheet_end();

     // Head-Bereich schließen und Nachrichten-Body öffnen
     Ret = create_head_end();

     // Im Nachrichten-Body koennen nun die aufgeführten
     // HTML-Elemente platziert werden. In diesem Beispiel wird
     // hier nur eine Überschrift und ein Eingabefeld platziert.

     // Überschrift erzeugen
     Ret = create_headline("Willkommen auf dem
                                   Webserver_8051F120", "1");

     // Eingabefeld erzeugen
     Ret = create_inputfield("I_Field_1", "30");

     // Webseite abschließen
     Ret = create_document_end();
}
```

Listing 7-5: Programmierung mit blockierendem Übertragungsmodus

B Weitere Listings

B.1 Die Funktion search_next_occurence

```
////////////////////////////
// search_next_occurence //
////////////////////////////
// <description>
// analyses the data in RX-Fifo and searches for single
// bytes, it can find either the character "Sign" or "Breaker"
// </description>
// <Sign> first character the function looks for </Sign>
// <Breaker> second character the function looks for </Breaker>
// <Buffer> where the singled bytes are stored </Buffer>
// <Buffer_Len> length of Buffer </Buffer_Len>
// <return> count of readed bytes </return>
int search_next_occurrence(char Sign, char Breaker, char *Buffer,
                                                    int Len_Buffer)
{
     int i = 0;
     signed int Ret = 0;
     int cmp_result = 0;
     char *Waste = 0;

     // Get the first byte out of the RX-Fifo
     Ret = tcp_read(WServerPtr->ConnDesc, &Buffer[i], 1);

     // If there are no Bytes in Input-Buffer, return 0
     if (Ret == (-1))
     {
          return(0);
     }

     // Check if first byte is the Sign
     cmp_result = (Buffer[i] - Sign);

     // If not, check if the first byte is the Breaker
     if (cmp_result != 0)
     {
          cmp_result = (Buffer[i] - Breaker);
     }

     // Run while:
     // - Sign/Breaker ist not found &&
     // - Input-Buffer has data &&
     // - Buffer-Length is not reached
     while ((cmp_result != 0) && (Ret != -1) &&(i < (Len_Buffer - 2)))
     {
          i++;

          // Read one more byte
          Ret = tcp_read(WServerPtr->ConnDesc, &Buffer[i], 1);
          if (Ret != (-2) && (Ret != 0))
          {
               // Check if the byte is the Sign
               cmp_result = (Buffer[i] - Sign);
```

```
                // If not, check if the byte is the Breaker
                if (cmp_result != 0)
                {
                     cmp_result = (Buffer[i] - Breaker);
                }
           }
           else
           {
                i--;
           }
      }
      // If we reached the Buffer-Length, return -1 (Failure)
      if (i >= (Len_Buffer-1))
      {
           return(-1);
      }

      // Set Buffer to 0 at position i+1 to terminate the string
      Buffer[i+1] = 0;
      return(i);
}
```

Listing 7-6: Die Funktion search_next_occurence

B.2 Die Funktion search_pattern

```
////////////////////
// search_pattern //
////////////////////
// <description>
// analyses the data in RX-Fifo and searches for a pattern of bytes
// therefor it works as an ring buffer
// </description>
// <Pattern> String the function looks for </Pattern>
// <Len_Pattern> length of the Pattern-string </Len_Pattern>
// <return> 0 -> Pattern found; !=0 -> Pattern not found</return>
int search_pattern(char *Pattern, int Len_Pattern)
{
     char Buffer[32] = ""; // Buffer is used as a ring buffer
     int write_cnt = 0;
     int read_cnt = 0;
     int read_cnt_old = 0;
     int cmp_result = 1;
     int i = 0;
     int runs = 0;
     signed int Ret = 0;

     // Read Bytes at the count of "Len" out of the
     // RX-Fifo to fill the Buffer_Array
     for (write_cnt = 0; write_cnt < (Len - 1); write_cnt++)
     {
          Ret = tcp_read(WServerPtr->ConnDesc, &Buffer[write_cnt], 1);
     }

     // Run while:
```

```
// - Pattern is not found &&
// - RX-Fifo has data &&
// - MAX_RUNS is not reached (used for timeout purpose)
while ((cmp_result != 0) && (Ret != -1) &&
                                    (runs < SEARCH_PATTERN_MAX_RUNS))
{
     runs++;

     // Read one more byte out of the RX-Fifo
     Ret = tcp_read(WServerPtr->ConnDesc, &Buffer[write_cnt], 1);

     // Count up the write_counter, if it reaches
     // the Buffer_Size set it to 0
     if (write_cnt < ((sizeof(Buffer) - 1)))
     {
          write_cnt++;
     }
     else
     {
          write_cnt = 0;
     }

     // Store the read_counter
     read_cnt_old = read_cnt;

     // Check if Buffer contains the Pattern
     for (i = 0; i < Len; i++)
     {
          // Check if the Buffer[read_cnt] and Pattern[i] are equal
          cmp_result = (Buffer[read_cnt] - Pattern[i]);

          if (cmp_result != 0) // if not
          {
               // Set read_counter to read_counter_old + 1,
               // if it reaches the Buffer Size set it to 0
               if (read_cnt_old < (sizeof(Buffer)-1))
               {
                    read_cnt = (read_cnt_old + 1);
               }
               else
               {
                    read_cnt = 0;
               }
               break;
          }

          // If the bytes are equal, count up the read_counter,
          // if it reaches the Buffer_Size set it to 0
          // and check the next byte
          if (read_cnt < (sizeof(Buffer)-1))
          {
               read_cnt++;
          }
          else
          {
               read_cnt = 0;
          }
     }
```

```
            i = 0;
        }
        return(cmp_result);
}
```

Listing 7-7: Die Funktion search_pattern

B.3 Übertragungsprotokoll beim Chunked Transfer-Encoding

```
HTTP/1.1 200 OK
Server: 8051 Webserver
Content-Type: text/html
Transfer-Encoding: chunked
Connection: close

0111
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" lang="de">
<head>
<title>8051 Webserver</title>

0028
<style type="TEXT/CSS" MEDIA="SCREEN">

00B6
ul#list_menue { float: left; width: 10.5em; background-color: #DDE5FF;
margin:0; border: 1px solid silver; color: #40586F; height: 30em; }

0053
ul#list_menue li { font-size:13px; list-style: none; margin: 0; padding:
0.1em; }

006C
div#menue_class { font-size:20px; text-align:left; font-family:Arial;
font-style:italic;padding-top:1em; }

005F
div#menue_title { font-size:22px; font-family:Arial; padding: 0; text-
decoration:underline; }

0093
div#content { margin-left: 12em; height: 30em; padding-top: 1em;
padding-left:1em;padding-right: 1em; border: 1px solid silver; }

0050
ul#list_menue select { font-size:13px; font-family:Arial; font-
style:italic; }

005C
div#content h1 { background-color: #DDE5FF; color: #40586F; margin:0;
padding-left:0.2em }

004C
```

```
div#content h2 { font-size:18px; font-family:Arial; padding-left:0.2em;}

004D
div#description { font-size:16px; font-family:Arial; padding-left:
0.2em; }

005B
table { font-size:20px; font-family:Arial; margin-left:2em; }

002E
#td_left { text-align:left; padding:0.4em; }

000A
</style>

0011
</head>
<body>

001C
<ul id="list_menue">
<li>

0018
<div id="menue_class">

000B
Allgemein

0008
</div>

000D
</li>
<li>

0030
<a href="http://134.104.28.209:53121">Home</a>

000D
</li>
<li>

0030
<a href="http://134.104.28.209:53121">Help</a>

0018
<div id="menue_class">

0008
Status

0008
</div>

000D
</li>
<li>
```

```
004E
<a href="http://134.104.28.209:53121/current_temp">Aktuelle
Temperaturen</a>

000D
</li>
<li>

0050
<a href="http://134.104.28.209:53121/current_pressure">Aktuelle
Dr&#252cke</a>

000D
</li>
<li>

0042
<a href="http://134.104.28.209:53121/measurement">Messreihen</a>

0018
<div id="menue_class">

000F
Einstellungen

0008
</div>

000D
</li>
<li>

0052
<a
href="http://134.104.28.209:53121/network_settings">Netzwerkeinstellunge
n</a>

0018
<div id="menue_class">

0007
Links

0008
</div>

000D
</li>
<li>

0029
<a href="http://www.mpifr.de">MPIfR</a>

000D
</li>
<li>
```

```
0029
<a href="http://www.h-brs.de">H-BRS</a>

000D
</li>
<li>

0018
<div id="menue_class">

0010
Schnellzugriff

0008
</div>

000E
</li>
</ul>

0014
<div id="content">

0029
<h1>Startseite des 8051 Webservers</h1>

0007
<br/>

0045
<h2>Herzlich Willkommen auf der Startseite des 8051 Webservers</h2>

000C
<br/><br/>

0018
<div id="description">

0007
<br/>

0038
Hier k&#246nnte eine Beschreibung der Webseite stehen.

0008
</div>

0008
</div>

0012
</body>
</html>

0
```

Listing 7-8: Chunked Transfer-Encoding

C Zusätzliche Dokumente auf der beigefügten CD

Tabelle 7-3: Auf der CD befindliche Dokumente

Dateiname	***Beschreibung***
diplomarbeit.docx	Vorgelegte Diplomarbeit im Format *.docx* (ab Microsoft Office Word 2007)
diplomarbeit.pdf	Vorgelegte Diplomarbeit im Format *.pdf*
webserver.c	Quellcodedatei der Webserveranwendung
webserver.h	Header-Datei der Webserveranwendung
w5300.c	Quellcodedatei des Kommunikationsmoduls
w5300.h	Header-Datei des Kommunikationsmoduls
web_stdlib.c	Quellcodedatei der Standard-Library
web_stdlib.h	Header-Datei der der Standard-Library
io.h	Header-Datei als Schnittstelle zwischen Webserveranwendung und Kommunikationsmodul

Literaturverzeichnis

1. Anatol Badach and Erwin Hoffmann. Technik Der IP-Netze TCP/IP Incl. IPv6. 2., akualisierte und erw. Aufl. 2007. München u.a., Hanser. Kapitel 1.6, Kapitel 2.3.

2. Eric Freeman and Elisabeth Freeman. HTML Mit CSS Und XHTML Von Kopf Bis Fuss. 2., korr. Nachdruck. 2008. Kln u.a., O'Reilly. Kapitel 8.

3. Laser & Co.Solutions GmbH. MyEthernet - Technische Beschreibung. 2009.

4. R.Fielding and J.Gettys. Request for Comments 2616 Hypertext Transfer Protocol HTTP/1.1. 1999. Network Working Group. Request for Comments.

5. Manfred Schwabl-Schmidt. Systemprogrammierung Für AVR-Mikrocontroller. 2009. Aachen, Elektor-Verlag. Kapitel 5.

6. Inc. WIZnet Co. W5300 Manual, High-Performance Internet Connectivity Solution. Version 1.2.3. 2008. http://www.wiznet.co.kr , WIZnet Co.,Inc.

Danksagung

Mein besonderer Dank gilt Herrn Prof. Dr. Bernd Klein für die Bereitstellung des Themas und die Betreuung während des Diplomsemesters.

Herrn Prof. Dr. Marco Winzker danke ich dafür, dass er durch seine Vorlesung in mir das Interesse an der Digitaltechnik geweckt hat. Außerdem möchte ich mich bei ihm für die Übernahme des Koreferats bedanken.

Ich bedanke mich bei allen Mitarbeitern der Digitaltechnikgruppe des Max-Planck-Institutes für Radioastronomie in Bonn für das angenehme Arbeitsklima und die außerordentliche Bereitschaft, mich bei der Diplomarbeit zu unterstützen.

Besonders bedanke ich mich bei Herrn Dipl. Ing. Andreas Bell, der meine Diplomarbeit seitens des Max-Planck-Instituts betreut hat.

Vor allem möchte ich meinen Eltern danken, ohne die mir dieses Studium nicht möglich gewesen wäre. Meiner Freundin Eva Külshammer danke ich für die bedingungslose Unterstützung während des gesamten Studiums.

Erklärung der Selbstständigkeit

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel verwendet. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum	Unterschrift