Eingangsspannung	5 V bis 24 V Gleichspannung, gegen Verpolung und Überspannung geschützt. Zuführung über 2,5-mm-Hohlstecker, Pluspol innenliegend
Stromaufnahme	ca. 30 mA
Netzwerkinterface	10Base-T (10 MBit/s, Twisted-Pair-Kabel, RJ45-Steckverbinder), galvanisch mittels Impulstransformator abgetrennt
Weitere Schnittstellen	JTAG (für Debugging/In-Circuit-Programming), RS232

Tabelle 9: Technische Daten der Baugruppe

wird diese mit der höchsten zulässigen Taktfrequenz betrieben. Hierfür kommt ein Schwingquarz (*Q2*) zum Einsatz, der eine Frequenz von 8 MHz erzeugt. Dieser erfordert den Einsatz zweier Blockkondensatoren von jeweils etwa 15 pF Kapazität (*C4, C5*) [MSP430DS].

Den letzten wichtigen Bestandteil der Webserver-Baugruppe stellt die Stromversorgung dar. Am Steckverbinder *X1* kann dazu ein handelsübliches Steckernetzgerät mit Hohlstecker (Pluspol innenliegend) angeschlossen werden. Eine Suppressor-Diode (*D3*) im Eingangsbereich dient dem Schutz der Baugruppe vor Verpolung und Überspannung. Zur Gewinnung der Betriebsspannung (VCC) von 3,3 V wird ein Linear-Spannungsregler (*IC4*) des Typs »LD1086DT33« der Firma ST-Microelectronics eingesetzt. Dessen Ausgangspotential (VCCOUT) ist über den Jumper *SJ1* vom eigentlichen Versorgungspotential (VCC) getrennt. Durch Öffnen dieses Jumpers kann eine Fremdspeisung der Platine über den Pfostensteckverbinder *X3* erfolgen, ohne den bestückten Spannungsregler zu beschädigen. Die Leuchtdiode *D4* (grün) zeigt das korrekte Anliegen der Versorgungsspannung an [P6KE_DS] [LD1086].

Alle unbenutzten sowie wichtigen Portpins des Mikrocontrollers und die Versorgungspotentiale stehen der Außenwelt über die Steckverbinder *X3* und *X4* zur Verfügung. Dadurch lässt sich die Baugruppe »Huckepack« auf eine Art Mutterplatine aufstecken, welche eine kundenspezifische Schaltungsapplikation enthält. Hierfür sind zweckmäßigerweise die beiden Pfostensteckverbinder auf der Unterseite zu bestücken. Die fertig aufgebaute und bestückte Baugruppe ist auf Seite 24 dargestellt. Sie besitzt die Abmessungen 75 mm x 60 mm. An dieser Stelle sei nochmals darauf hingewiesen, dass der Schaltplan, der Bestückungsplan sowie die Leiterplattenlayouts dem Anhang ab Seite 45 zu entnehmen sind. Aus Gründen wie günstigere Möglichkeiten der Signalführung und verbesserte EMV-Eigenschaften wurde die Leiterplatte 4-lagig realisiert (Die Versorgungslagen VCC und GND liegen innen, siehe Seite 48).

Tabelle 9 gibt einen Überblick über die technischen Daten der Baugruppe. Der größte Teil des dort angegeben Stromverbrauchs geht dabei auf das Konto des Ethernet-Controllers. Dieses IC stellt eine beträchtliche Sammlung an integrierter Logik dar, noch dazu wird es von einem 20-MHz-Quarz gespeist. Außerdem arbeitet der Analogteil dieser Schaltung, der für die physikalische Anbindung an das Netzwerk sorgt, natürlich auch mit gewissen Strömen.

Applikation	● möchte mittels Ethernet und TCP/IP-Protokoll Daten übertragen ● benutzt dazu die Bibliotheksfunktionen (API) des TCP/IP-Moduls, welche den kompletten TCP/IP-Stack kapseln und vor der Anwendung verbergen
TCP/IP-Modul tcpip.c tcpip.h	● stellt Bibliothek für Anwendungsentwicklung dar ● implementiert die Protokolle ARP, ICMP, IP, TCP ● reagiert auf Ereignisse (Frame-Eingang, API-Aufrufe durch Anwendungsprogramm)
Ethernet-Modul cs8900.c cs8900.h	● Treiber für Nutzung des Ethernet-Controllers CS8900A ● stellt Funktionen zur Konfiguration, zum Beschreiben/Auslesen von Register und zum Senden/Empfangen von Ethernet-Frames bereit
Ethernet	● physikalisches Medium, über welches der Datentransport stattfindet. Zugang erfolgt mittels LAN-Controller

Tabelle 10: Zusammenspiel der Softwaremodule

Software - Die eigentliche Lösung

Die folgenden Abschnitte beschreiben den in C implementierten Protokollstapel. Basis für die Software stellt die gerade vorgestellte Webserver-Baugruppe mit dem Mikrocontroller MSP430F149 und dem Ethernet-Controller CS8900A dar. Als Entwicklungsumgebung kam die »Embedded Workbench« von IAR Systems zum Einsatz. Um eine gute Übersicht und Trennung der einzelnen Programmteile zu erreichen, erfolgte eine Aufteilung des Quellcodes in verschiedene, zweckmäßige Module. Tabelle 10 verdeutlicht die Zusammenhänge zwischen diesen Modulen.

Ethernet-Modul

Aufgabe des Ethernet-Moduls `cs8900.c` ist die Kapselung verschiedener, für die Datenübertragung wichtiger Funktionen des Ethernet-Controllers durch einfach zu handhabende C-Routinen. Weiterhin wird das für die Kommunikation mit dem CS8900A notwendige ISA-Bus-Taktregime erzeugt. Bei Portierung des Quellcodes auf ein Mikrocontrollersystem, welches im Gegensatz zum MSP430F149 über einen externen Daten- und Adressbus verfügt, kann der Entwickler in diesem Modul die dafür notwendigen Anpassungen vornehmen.

Die wohl wichtigste Einstellungsmöglichkeit findet sich gleich zu Beginn des Header-Files `cs8900.h`: Durch Definition der symbolischen Konstanten `MYMAC_1` bis `MYMAC_6` wird die 48 Bit lange MAC-Adresse des Ethernet-Controllers festgelegt, beginnend mit dem »Most Significant Byte«. Diese ist wei-

testgehend beliebig konfigurierbar, es ist jedoch auf eine Eindeutigkeit der Adresse im lokalen Netzwerk zu achten. Weiterhin darf sie keinesfalls »FF-FF-FF-FF-FF-FF« betragen (reserviert als »Broadcast«-Adresse).
Tabelle 11 zeigt die einzelnen Funktionen des Ethernet-Moduls in einer Übersicht. Als Grundgerüst zum Übertragen von Daten über das Ethernet unter Nutzung der vorgestellten Funktionen soll Bild 18 auf Seite 30 dienen. Zunächst ist durch Aufruf der Funktion `Init8900()` der Ethernet-Controller zu initialisieren. Dazu wird ein softwaremäßiger Chip-Reset durchgeführt, und die in der Datenstruktur `InitSeq[] (cs8900.h)` hinterlegte Konfigurationssequenz übertragen. Dabei besteht jedes Element aus einem Adress- und einem Datenwort. Die Bedeutung der einzelnen angesprochenen Register und der zugehörigen Flags ist der Quelle [Crystal] zu entnehmen. Nach erfolgter Initialisierung kann die eigentliche Datenübertragung beginnen.

TCP/IP Modul

Im Modul `tcpip.c` werden die für die Datenübertragung via TCP/IP notwendigen Protokolle implementiert, damit stellt dieses Modul wohl den wichtigsten und interessantesten Teil dieses Projekts dar. Es baut einerseits auf dem gerade beschrieben Ethernet-Modul auf, stellt aber andererseits Funktionen und Statusflags bereit, mit denen eine übergeordnete Anwendung auf einfache Art und Weise eine Verbindung zu einem entfernten TCP/IP-Netzknoten aufbauen, mit diesem Daten austauschen und schließlich die Verbindung wieder beenden kann. Durch Einsatz des »Transmission Control Protokolls« kann ein noch zu erstellendes Anwendungsprogramm von allen Vorzügen einer zuverlässigen, streamorientieren Kommunikation zweier Endpunkte profitieren.

Funktion	Aufgabe
`void Init8900(void)`	Konfiguriert die entsprechenden Portpins der MCU, an welche der CS8900A angeschlossen ist, führt einen Software-Chip-Reset durch und konfiguriert das MAC-Interface.
`void Write8900(unsigned char Address, unsigned int Data)`	Schreibt den 16-Bit-Wert **`Data`** in eines der acht möglichen Register (**`Address`**) des CS8900A im I/O-Betrieb. Das Wort wird in »Little-Endian Byte-Order« (s.u.) im Register abgelegt.
`void WriteFrame8900 (unsigned int Data)`	Schreibt den 16-Bit-Wert **`Data`** im Little-Endian-Format in das I/O-Register **`TX_FRAME_PORT`**. Dient dem Übertragen eines Wortes zum Sendepuffer.
`unsigned int Read8900 (unsigned char Address)`	Liest einen 16-Bit-Wert im Little-Endian-Format aus dem angegeben I/O-Register **`Address`** des CS8900A.
`unsigned int ReadFrame8900 (void)`	Liest einen 16-Bit-Wert im Little-Endian-Format vom I/O-Register **`RX_FRAME_PORT`**. Dient dem Auslesen des Empfangspuffers.
`unsigned int ReadHBIST8900 (unsigned char Address)`	Liest ebenfalls einen 16-Bit-Wert im Little-Endian-Format aus dem angegeben I/O-Register aus. Dabei wird jedoch das höherwertige Byte zuerst ausgelesen. Diese Funktion ist zum Ansprechen bestimmter Register des CS8900A im 8-Bit-Modus notwendig.
`unsigned int ReadFrameBE8900 (void)`	Liest einen 16-Bit-Wert im Big-Endian-Format vom I/O-Register **`RX_FRAME_PORT`**. Dient der Vermeidung permanenten »Byte-Swappings« beim Auslesen der Daten höherer Protokolle.
`void CopyToFrame8900(void *Source, unsigned int Size)`	Überträgt **`Size`** Bytes aus dem MCU-Speicher, beginnend bei **`Source`** in das I/O-Register **`TX_FRAME_PORT`** des CS8900A. Dient hauptsächlich dem Senden eines im Speicher vorbereiteten Frames.
`void CopyFromFrame8900(void *Dest, unsigned int Size)`	Liest **`Size`** Bytes aus dem Frame-Empfangspuffer des LAN-Controllers und legt diese im MCU-Speicher ab Adresse **`Dest`** ab.
`Void DummyReadFrame8900 (unsigned int Size)`	Liest und ignoriert **`Size`** Bytes aus dem I/O-Register **`RX_FRAME_PORT`**, wobei nur die Angabe einer gerade Anzahl von Bytes erlaubt ist. Dient dem Überspringen nicht benötigter Abschnitte eines empfangenen Ethernet-Frames.
`void RequestSend (unsigned int FrameSize)`	Fordert **`FrameSize`** Bytes Speicher im Sendepuffer des CS8900A an. Diese Funktion muss **vor** dem Übertragen eines zu sendenden Frames zum LAN-Controller aufgerufen werden.
`Unsigned int Rdy4Tx(void)`	Prüft, ob der zum Senden angeforderte Speicherplatz zur Verfügung gestellt wurde und somit Daten zum CS8900A übertragen werden dürfen (Rückgabewert ungleich null).

Tabelle 11: Funktionen des Ethernet-Moduls, »Little Endian« bedeutet, dass das niederwertige Byte an der niederwertigen Adresse steht (auch Intel-Format genannt)

Grundsätzlich stellt das Modul `tcpip.c` eine Sammlung verschiedener Ereignisbehandlungs-Routinen dar, ergänzt um einige Hilfsfunktionen (z.B. zur Zahlenkonvertierung). In einem TCP/IP-Stack können Ereignisse unterschiedlichster Art und Herkunft auftreten:

- Ein Frame wurde aus dem Netzwerk empfangen,
- Benutzeraktion (Verbindung aufbauen, Daten senden etc.),
- Ablauf eines Zeitlimit (Timeout),
- Auftreten eines Fehlers (Netzwerk-Fehler, Zurücksetzen der Verbindung etc.).

Um den Quelltext und die Funktionsweise des Moduls verstehen zu können, sind zunächst einige Erläuterungen zu den getroffenen Vereinbarungen notwendig. Aus Gründen der Geschwindigkeit und des Speicherbedarfs erfolgt softwareseitig keine strenge Unterteilung der Schichten TCP und IP. Vielmehr werden deren einzelne Header meistens sogar von der gleichen Funktion ausgewertet oder generiert.

- Pufferspeicher:

Zur Arbeit mit ein- und ausgehenden Frames werden im MCU-Speicher drei Speicherbereiche reserviert (Bild 19 auf Seite 31). Deren Größe lässt sich im Header-File `tcpip.h` über die Konstantendefinitionen in gewissen Grenzen an den verfügbaren Speicherplatz anpassen. Dabei führt die Vergrößerung der Puffer zu einer dramatischen Erhöhung der Übertragungsgeschwindigkeit, da diese Implementierung jeweils nur ein TCP-Segment zwischenspeichern kann. Dieses darf erst verworfen werden, wenn beim Senden eine entsprechende Bestätigung eingegangen ist, (`TxFrame1`) oder beim Empfangen der Empfangspuffer wieder freigegeben wurde (`RxTCPBuffer`). Wegen der langen Signallaufzeiten in einem WAN (Wide Area Network) und der Verzögerung von »ACK«-Segmenten durch manche Implementierungen (z.B. Microsoft Windows 2000) wird bei der Übertragung sehr kleiner Segmente die mögliche Nettodatenrate sonst stark reduziert.

- Globale Variablen:

Alle Informationen über den derzeitigen Status des TCPs werden in globalen Variablen abgelegt, damit sie für alle Prozeduren des Stacks gleichermaßen gut zugänglich sind. Dazu zählen beispielsweise Lokale IP-Adresse und Lokaler TCP-Port, Remote-IP-Adresse und Remote-TCP-Port, Sequenznummern, Timer sowie diverse Flag-Register. Die wohl wichtigste Variable ist `TCPStateMachine`. In ihr wird der aktuelle Zustand des TCP-Automaten entsprechend [RFC-793] abgelegt.

Die Umsetzung der Protokolle stützt sich weitestgehend auf deren entsprechende Quellen (RFCs, siehe Seite 71). Die im Vorfeld gelieferten Kurzbeschreibungen dieser Protokolle sollten jedoch für ein Verständnis des Quellcodes ausreichen. Ausgangspunkt aller Reaktionen auf Ereignisse stellt die Routine `DoNetworkStuff()` dar, welche von einem übergeordneten Anwendungsprogramm periodisch aufgerufen werden muss. Dies resultiert aus der Tatsache, dass innerhalb dieser Routine verschiedene Status-Flags (im Ethernet-Controller oder globale Variablen) ausgewertet werden (Polling) und zur entsprechenden Ereignisbehandlungs-Routine verzweigt wird. Je öfter das erfolgt, desto schneller ist die Arbeits- und Reaktionsgeschwindigkeit des TCP/IP-Stacks. Der Ablaufplan dieser Routine ist in Bild 20 auf Seite 32 dargestellt.

Einen Sonderfall der Ereignisbehandlungsroutinen stellen die möglichen Benutzer-Aktionen dar. Diese Sammlung von Unterprogrammen wird direkt durch die Anwendungsschicht aufgerufen. Der nachfolgende Abschnitt »Jetzt heißt es zugreifen« stellt diese API-Funktionen ab Seite 34 ausführlich und im Detail vor. Aufgrund der engen Verzahnung und der Abhängigkeiten innerhalb des Quelltextes kommen bestimmte Funktionen bei der folgenden Beschreibung des TCP/IP-Stacks bereits zur Sprache.

- Demultiplexing empfangener Frames:

Wurde innerhalb der Routine `DoNetworkStuff()` der Empfang eines Frames durch den LAN-Controller erkannt, erfolgt zunächst ein Prüfen der Zieladresse. Durch Abfrage des Registers `PP_RxEvent` des CS8900A ist eine Unterscheidung zwischen individueller (IA, Zieladresse entspricht der MAC-

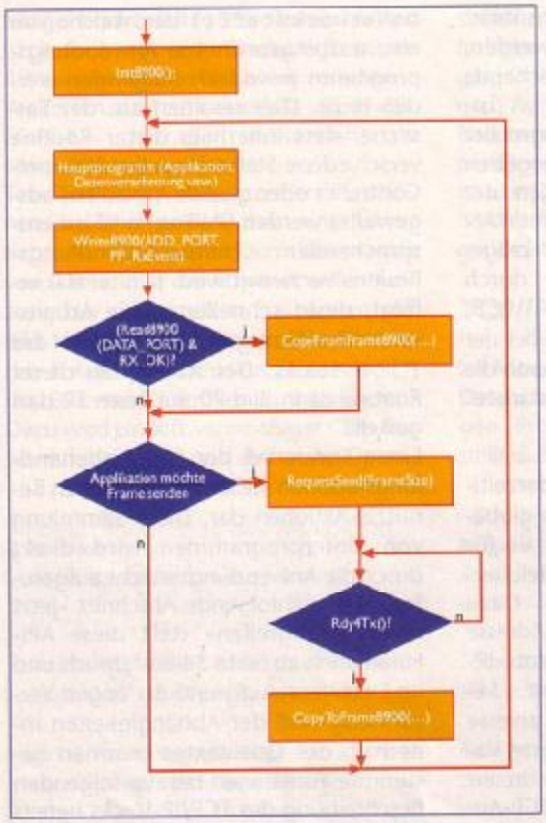


Bild 18: Nutzung des Ethernet-Moduls

Adresse der vorliegenden Baugruppe) und Broadcast-Adressierung möglich. Danach verzweigt das Programm zur entsprechenden Ereignisbehandlungs-Funktion `ProcessEthIAFrame()` oder `ProcessEthBroadcastFrame()`. Bild 21 auf Seite 33 verdeutlicht diesen Prozess des Demultiplexing.

Beim Empfang eines Broadcast-Frames erfolgt das Prüfen auf einen »ARP-Request«. Wenn in dieser Anforderung die IP der vorliegenden Baugruppe angegeben ist, folgt das Generieren eines Antwort-Frames durch Aufruf der Funktion `PrepareARP_ANSWER()`. Dieser Frame wird dazu im Puffer `TxFrame2` abgelegt und durch Setzen des Flags `SEND_FRAME2` im Register `TransmitControl` zum Senden markiert. Damit wird dem fragenden Netzknoten die eigene MAC-Adresse mitgeteilt.

Wenn ein individuell an die eigene Baugruppe adressierter Frame empfangen wird, prüft die Routine `ProcessEthIAFrame()` zunächst, ob es sich um eine Antwort auf einen vorangegangenen »ARP-Request« des eigenen TCP/IP-Stacks handelt. Trifft das zu, und wird ein solcher Frame erwartet, filtert die Routine die MAC-Adresse des Absenders heraus, ansonsten verwirft sie den Frame. Dieser Mechanismus ist für den möglichen aktiven Verbindungsaufbau notwendig, wobei der lokale TCP/IP-Stack die MAC-Adresse des anderen TCPs herausfinden muss, um diesen direkt ansprechen zu können. Handelt es sich dagegen um einen IP-Daten tragenden Frame, erfolgt nach dem Testen auf Übereinstimmung der IP-Zieladresse mit der lokalen IP-Adresse eine Verzweigung entsprechend der vom Stack unterstützten Protokolle (im Moment ICMP und TCP).

Die Funktion `ProcessICMPFrame()` wertet zunächst den Typ der ICMP-Nachricht aus. Falls es sich um ein »ICMP-Echo« handelt, wird mittels der Funktion `PrepareICMP_ECHO_REPLY()` ein Antwort-Frame erzeugt und zum Senden markiert (Setzen des Flag `SEND_FRAME2`). Dadurch erfolgt eine Reaktion auf ein »Ping« eines Remote-Host. Andere ICMP-Nachrichten werden verworfen. Die Software gewährleistet die ständige Reaktionsbereitschaft auf eingehende »ARP«- und »ICMP-Echo«-Requests, da bei Eintritt in die Routine `DoNetworkStuff()` der dafür benötigte Sendepuffer `TxFrame2` immer freigegeben ist.

Die Auswertung von TCP-Frames erfolgt durch Aufruf der Funktion `ProcessTCPFrame()`. Dieser Programmabschnitt stellt einen sehr wichtigen Teil des Stacks dar. Hier erfolgt die Umsetzung der in [RFC-793] vorgeschlagenen Reaktion auf den Eingang eines TCP-Frames. Zunächst wird aber geprüft, ob gerade eine »TCP-Session« aktiv ist und das eingegangene Segment zur aktiven Session gehört, oder ob eine Anforderung eines Remote-TCPs zum Verbindungsaufbau vorliegt (Segment trägt ein *SYN*-Flag). Danach werden Zustandsübergänge des TCP-Automaten (`TCPStateMachine`) durchgeführt, Daten empfangen und eingegangene Segmente durch Senden von *ACK*-Flags bestätigt.

Dabei werden nur TCP-Segmente akzeptiert, welche sich von ihrer Sequenznummer her **nahtlos** an das vorher Empfangene anfügen, da aufgrund mangelnder Hardware-Ressourcen keine Zwischenspeicherung von in falscher Reihenfolge eingegangenen Segmenten erfolgen kann. Zum Senden von **nicht** datentragenden TCP-Segmenten (also solchen Segmenten, die im wesentlichen nur TCP-Flags enthalten) dient die Funktion `PrepareTCPFrame()`. Als Parameter kann dabei eine beliebige (erlaubte!) Kombination der TCP-Flags übergeben werden, die Ablage des erzeugten Frames erfolgt ebenfalls im Puffer `TxFrame2`. Der Aufruf von `PrepareTCPFrame(TCP_CODE_ACK)` führt beispielsweise zum Erzeugen eines Bestätigungssegmentes.

Um beim Empfangen von Daten einen Schutz vor Überlaufen des Empfangspuffers zu gewährleisten, wurde ein Handshake-Mechanismus realisiert. Hierzu dient das Flag `SOCK_DATA_AVAILABLE` im Register `SocketStatus`. Daten werden vom Stack nur in den Empfangspuffer `RxTCPBuffer` übertragen, wenn dieses Flag gelöscht ist. Auch das Bestätigen von empfangenen Daten durch Senden eines *ACK*-Segments erfolgt nur bei gelöschtem `SOCK_DATA_AVAILABLE`-Flag. Konnten Daten erfolgreich in den Empfangspuffer übertragen werden, zeigt der Stack dies der Applikation durch Setzten des Flags `SOCK_DATA_AVAILABLE` an. Ist die Routine aufgrund eines vollen Empfangspuffers gezwungen, ein empfangenes Datensegment zu verwerfen, gehen dennoch keine Daten verloren. Denn der Remote-TCP startet eine zeitverzögerte erneute Übertragung, wenn er das entsprechende Bestätigungssegment nicht empfängt. Im Regelfall sollte das Anwendungsprogramm aber für das rechtzeitige Abholen der empfangenen Daten sorgen.

◆ Verbindungsaufbau:

Der Aufbau einer Verbindung erfolgt entweder aktiv oder passiv und wird durch den Aufruf der entsprechenden API-Funktion `TCPActiveOpen()` bzw. `TCPPassiveOpen()` ausgelöst. Beim aktiven Verbindungsaufbau versucht die Routine zunächst, über das Senden eines »ARP-Requests« die zur vorher festgelegten »Remote IP« gehörende MAC-Adresse herauszufinden. Dazu prüft die Funktion `PrepareARP_REQUEST()` zunächst, ob sich die Ziel-IP im konfigurierten Subnetz befindet. Trifft das zu, wird der andere TCP-Knoten direkt mit seiner IP angesprochen, ansonsten leitet die Software den »ARP-Request« auf die IP

Bild 19: Pufferkonzept

des Gateways um. Die Subnetzmaske wird durch die Definition der symbolischen Konstanten `SUBMASK_1..4`, die IP-Adresse des Gateways durch `GWIP1..4` festgelegt. Kommt bei passiv geöffnetem Kanal ein Verbindungsaufbau nicht aus dem Subnetz, tritt automatisch der Gateway als Kommunikationspartner der Baugruppe in Erscheinung. Für den lokalen Stack scheint es, der TCP-Stack außerhalb des Subnetzes hätte die MAC-Adresse des Gateways. Es spielt somit keine Rolle, ob die Verbindung vom aktuellen Subnetz oder von außerhalb aus aufgebaut werden soll. Weitere Informationen über Subnetzmasken und Gateways sind der Fachliteratur zu entnehmen [CNetw] [TCP/IP].

Das erste von unserem Stack zu sendende TCP-Segment weist nun einige Besonderheiten auf. Zum einen trägt es das *SYN*-Flag und damit die Initiale Sequenznummer (siehe Abschnitt »Interrupt Service Routine«), zum anderen die TCP-Option »Maximum Segment Size« (MSS). Zum Senden dieses Frames dient wiederum die Funktion `PrepareTCPFrame()`. Wenn bei deren Aufruf als Parameter das Flag *SYN* enthalten ist, wird dem Frame automatisch die »MSS«-Option hinzugefügt. Mit dieser Option wird dem anderen TCP-Knoten mitgeteilt, wie viele Nutzdatenbytes er höchstens mit einem Segment senden darf. Hier wird die Größe des lokalen Empfangspuffers (`MAX_TCP_RX_DATA_SIZE`) eingetragen. Somit ist gewährleistet, das kein empfangenes Segment größer ist als der zur Verfügung stehende Pufferspeicher. TCP-Segmente können theoretisch bis maximal 64 KByte groß werden – einfach zuviel für unseren Mikrocontroller.

◆ Datenübertragung:

Wenn eine Verbindung erfolgreich hergestellt wurde (das Register `TCPStateMachine` geht in den Zustand ***ESTABLISHED*** über), kann die eigentliche Datenübertragung beginnen. Der aktuelle Status einer Verbindung wird regelmäßig im Register `SocketStatus` hinterlegt (siehe Beschreibung im Kapitel »Jetzt heißt es zugreifen« ab Seite 34).

Der Empfang von Daten und das Kopieren in den Empfangspuffer erfolgt durch die bereits vorgestellte Routine `ProcessTCPFrame()`. Die Anzahl im Empfangspuffer ab-

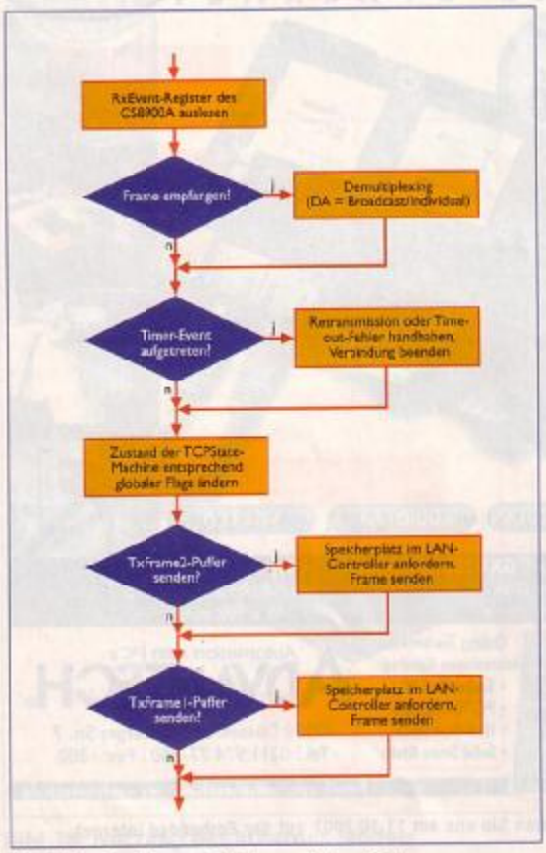


Bild 20: Routine »DoNetworkStuff()«

gelegter Bytes wird im Register `TCPRxDataCount` hinterlegt. Erst nach Freigabe des Empfangspuffers durch den Aufruf der API-Funktion `TCPReleaseRxBuffer()` ist der Empfang neuer Daten wieder möglich.

Um Daten an den anderen TCP zu senden, muss das Anwendungsprogramm diese Daten in den TCP-Datenbereich des `TxFrame1`-Puffers ablegen. Zum einfachen Direktzugriff auf diesen Bereich dient der Zeiger `TCP_TX_BUF`. Durch Aufruf der API-Funktion `TCPTransmitTxBuffer()` wird der Sendevorgang ausgelöst. Dazu prüft die Funktion zunächst die Gültigkeit des aktuellen TCP-Zustands und kennzeichnet den Sendepuffer als belegt. Das Setzen des Flags `SEND_FRAME1` im Register `TransmitControl` führt dann innerhalb der Routine `DoNetworkStuff()` zum eigentlichen Senden. Nach dem Empfangen eines Bestätigungssegments des anderen TCP wird der Sendepuffer wieder freigegeben, und die Applikation kann in diesen erneut Daten eintragen und senden.

◆ Verbindungsabbau:

Für das Trennen der TCP-Verbindung kann es verschiedene Gründe geben. Im Regelfall wird das Beenden entweder lokal durch Aufruf der API-Funktion `TCPClose()` eingeleitet, welche zum Senden eines *FIN*-Segments führt, oder durch den Remote-Stack (Empfangen eines Segmentes mit gesetztem *FIN*-Flag). Das Auftreten eines Übertragungs-Timeouts oder das Empfangen eines *RST*-Segments führen ebenfalls zum Lösen der Verbindung. Diese und andere Fehlerfälle werden im Register `SocketStatus` angegeben.

◆ Timer:

Bei der Umsetzung des TCP-Protokolls sind bestimmte Abläufe zeitgesteuert zu gestalten. Zu diesem Zweck konfiguriert die Software in der Initialisierungsphase dieses Stacks bei Aufruf der API-Funktion `TCPLowLevelInit()` den Timer A des MSP430F149. Eine Forderung aus [RFC-793] ist die Generierung der für einen Verbindungsaufbau benötigten, 32 Bit langen »Initialen Sequenznummer« (ISN) aus einem 250-kHz-Taktgenerator. Dazu wird zunächst der 8-MHz-Quarztakt unserer Baugruppe durch ein Gesamt-Teilerverhältnis von 1:32 auf 250 kHz umgesetzt. Dieser Takt speist einen freilaufenden 16-Bit-Zähler, welcher bei jedem Überlauf (d.h. alle 262 ms) einen Interrupt auslöst. Dieser Zähler ist über das MCU-Register *TAR* zugänglich und lässt sich direkt zur Gewinnung der niederwertigen 16 Bit der ISN nutzen. In der Interrupt-Service-Routine (ISR) `TCPClockHandler()` wird bei jedem Zählerüberlauf die globale 16-Bit-Variable `ISNGenHigh` inkrementiert. Sie stellt damit die höherwertigen 16 Bit der ISN bereit.

Zusätzlich wird bei jedem Timer-Interrupt das Register `TCPTimer` erhöht. Dieser Zähler dient der Generierung von Timer-Ereignissen des TCP. Diese werden zum Beispiel gebraucht, wenn das Ausbleiben einer Bestätigung (ACK-Segment) für gesendete Daten zu einer Übertragungswiederholung oder zum Lösen der Verbindung führt. Ein anderes Beispiel ist das Abwarten einer bestimmten Zeitspanne nach dem normalen Beenden der Verbindung im TCP-Zustand *TIME WAIT*.

◆ Retransmission:

Damit ein Verlust von gesendeten Daten nicht zum Zusammenbrechen der TCP-Verbindung führt, steht ein zeitgesteuerter Mechanismus für die Übertragungswiederholung (Retransmission) zur Verfügung. Dazu legt der Stack nach dem Senden jedes Frames dessen Typ im Register `LastFrameSent` ab. Bleibt nach Ablauf der Zeitspanne `RETRY_TIMEOUT` eine Bestätigung aus, wird automatisch die Routine `TCPHandleRetransmission()` aufgerufen und das letzte Frame erneut gesendet. Wenn nach einer durch die Variable `MAX_RETRYS` festgelegten Anzahl von Versuchen immer noch keine Bestätigung eingegangen ist, trennt die Routine die aktuelle TCP-Verbindung. Dieser Mechanismus wird nicht für TCP-Segmente ohne Daten verwendet, welche lediglich ein *ACK*-Flag enthalten. Solche Segmente werden generell nicht bestätigt, die beiden TCPs würden sich sonst gegenseitig mit Bestätigungssegmenten »zumüllen«.

Bei der Portierung des Stacks auf andere Controllersysteme ist besonderer Wert auf das dort verwendete Zahlenformat zu legen. Bei MCUs des Typs MSP430 erfolgt die Zahlendarstellung im »Little-Endian«-Format. Dies bedeutet, dass das niederwertige Byte an der niederwertigen Adresse steht (auch Intel-Format genannt). Da jedoch die

Rechnersystem / CPU-Typ	Betriebssystem bzw. TCP/IP-Stack
PC / Athlon 1 GHz	Windows 2000
PC / Athlon 1 GHz	Linux mit Kernel 2.2.16
PC / Pentium 233 MHz	Windows 98
PC / 486DX2 66 MHz	Windows 95
Apple Macintosh / 68030	System 7.5 mit Open Transport 1.1.2
Commodore Amiga / 68030	Kickstart 3.0 mit Miami 2.1
Cassiopeia / MIPS 150 MHz	Windows CE 3.0

Tabelle 12: Getestete Konfigurationen

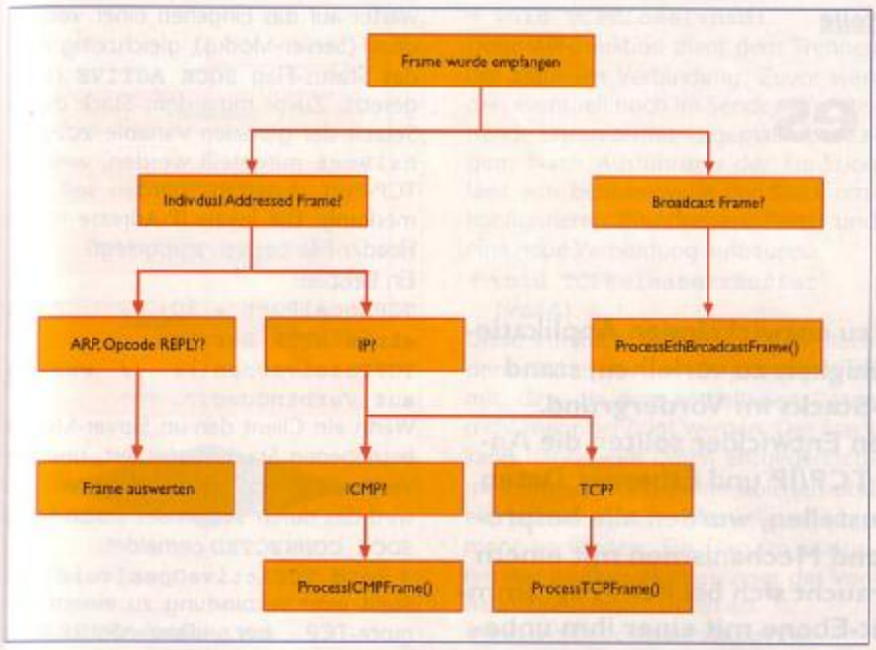


Bild 21: Demultiplexing empfangener Frames

meisten Protokolle des Internet das »Big-Endian«-Zahlenformat nutzen, waren im hier vorliegenden Quelltext sehr oft Konvertierungen durch Nutzung des Makros `SWAPB()` bzw. der Funktion `SwapBytes()` notwendig. Alle Quelltextzeilen, welche mittels Zeiger direkt auf Speicherinhalte zugreifen, sollten somit genau überprüft und gegebenenfalls überarbeitet werden.

Das Ergebnis

Obwohl bei der Entwicklung des vorgestellten TCP/IP-Stacks auf Grund von Ressourcenknappheit viele Kompromisse nötig waren, zeichnete sich im Praxistest doch eine erstaunliche Kompatibilität von Baugruppe und Software ab. Tabelle 12 gibt einen Auszug aus der Liste der TCP/IP-Stacks an, mit denen erfolgreich und problemlos Daten ausgetauscht werden konnten.

Die wichtigsten getroffenen Einschränkungen der Protokoll-Spezifikationen sind:

- Nur eine mögliche aktive TCP-Session
- Kein Zusammensetzen fragmentierter IP-Datagramme (Reassembling)
- kein Zwischenspeichern von in falscher Reihenfolge gelieferten TCP-Segmenten
- keine Checksummenüberprüfung von eingehenden Daten
- keine Unterstützung der Type Of Service (TOS) und Security-Optionen
- Ignorierung eingehender TCP-Optionen

Die beschriebene Kompatibilität wird einerseits durch die Implementierung der wesentlichsten Funktionen und Mechanismen eines TCP/IP-Stacks, andererseits aber auch durch die Toleranz der getesteten Kommunikationspartner erreicht.

Der vorgestellte TCP/IP-Stack mit eingebundenem Ethernet-Modul benötigt bei Compilierung für einen MSP430F149 folgende Speicher-Ressourcen:

- Etwa 4,2 KByte Flash-EEPROM für den Programmspeicher,
- ca. 100 Byte Flash-EEPROM als Konstantenspeicher sowie
- ungefähr 700 Byte RAM, inklusive zweier jeweils 256 Nutzdaten-Byte fassender Sende- und Empfangspuffer für TCP-Datentransfer.

Ein interessanter Punkt ist die Frage nach der erreichbaren Übertragungsgeschwindigkeit der vorgestellten Kombination aus Hard- und Software. Darauf ist leider keine absolute Antwort möglich, die gemessenen Datenraten reichen von zwei bis zu mehreren Dutzend KByte pro Sekunde und variieren damit sehr stark. Diese Schwankungen hängen mit verschiedenen Randbedingungen zusammen.

Die Datenübertragung in Richtung des vorgestellten TCP/IP-Stacks läuft generell mit recht hoher Geschwindigkeit ab. Bei der Datenübertragung zu einem anderen TCP gilt es dagegen zu differenzieren. Eine besonders schnelle Kommunikation konnte mit dem TCP/IP-Stack von Linux hergestellt werden, die Microsoft-Implementierungen erwiesen sich als wesentlich langsamer. Eine Analyse mit dem Netzwerkmonitor ergab den Grund der Differenzen. Beim Senden eines TCP-Segments durch den vorgestellten Stack wartet dieser auf die Bestätigung durch die Gegenstelle, bevor ein neues Segment gesendet werden kann.

Dies ist nötig, um den Pufferinhalt für eine eventuelle erneute Sendung des Segments zu konservieren. Manche TCP/IP-Implementierungen senden aber nicht sofort nach dem Empfang von Daten ein Bestätigungssegment, sondern etwas zeitverzögert (»Delayed ACK«).

Hintergrund: Der empfangende TCP-Stapel kann somit erst einige Segmente puffern und diese durch das Senden eines einzelnen *ACK*-Segments bestätigen, was sich vorteilhaft auf die Netzlast auswirkt. Dieses Feature stellt bei der Kommunikation zweier TCP-Stacks, welche die gesamten Spezifikationen umsetzen, keine Beschränkung der Übertragungsgeschwindigkeit dar. Um mit dem hier vorgestellten Konzept dennoch hinreichend schnelle Datenraten erreichen zu können, ist der Sendepuffer möglichst groß zu gestalten [RFC-1122]. (cg)

Nutzung der Applikations-Schnittstelle

Jetzt heißt es zugreifen!

Der Wunsch, bestehenden oder zu entwickelnden Applikationen möglichst einfach Internetfähigkeit zu verleihen, stand bei der Entwicklung des TCP/IP-Stacks im Vordergrund. Ohne immensen Aufwand für den Entwickler sollten die Anwendungen in der Lage sein, via TCP/IP und Ethernet Daten auszutauschen. Um dies sicherzustellen, wurden alle besprochenen Protokolle, Funktionen und Mechanismen mit einem API gekapselt. Der Anwender braucht sich bei der Programmerstellung nun nicht mehr auf Bit-Ebene mit einer ihm unbekannten und komplexen Thematik herumzuschlagen, sondern kann sich praktisch einer fertigen Lösung bedienen.

Unbedingt notwendig zur Gewährleistung der ordnungsgemäßen Funktion des hier vorgestellten TCP/IP-Stacks ist das periodische Aufrufen der Funktion `DoNetworkStuff()` durch das Anwendungsprogramm. Der in Bild 22 dargestellte Programmierstil gewährleistet dies optimal. Die mit »sonstiges Anwenderprogramm« bezeichneten Abschnitte dürfen hierbei keinesfalls die Möglichkeit haben, diesen Ablauf zu blockieren (z.B. endloses Warten auf das Eintreten irgendeines Ereignisses). Vielmehr ist durch Nutzung von Hilfs-Flags und Timern die ständige Abarbeitung des dargestellten Zyklus zu gewährleisten.

Ein Beispiel für die Anwendung der API-Funktionen und eine **nicht-blockierende** Programmierweise ist der im nächsten Kapitel vorgestellte HTTP-Server. Im Folgenden werden die für die Nutzung des Stacks notwendigen Funktionen und Status-Flags vorgestellt.

Funktionsaufrufe

◆ `void TCPLowLevelInit(void)`
Diese Funktion führt grundlegende Vorbereitungen durch und muss daher vor Benutzung des Softwaremoduls aufgerufen werden. Zu ihren Funktionen zählen beispielsweise das Initialisieren des Ethernet-Controllers, das Konfigurieren und Bereitstellen des für eine Verbindung nötigen Timers und das Rücksetzen aller internen Statusflags.

◆ `void TCPPassiveOpen(void)`
Durch Aufruf dieser Funktion wechselt der Stack in den Zustand ***LISTEN*** und wartet auf das Eingehen einer Verbindung (Server-Modus), gleichzeitig wird das Status-Flag `SOCK_ACTIVE` (s.u.) gesetzt. Zuvor muss dem Stack durch Setzen der globalen Variable `TCPLocalPort` mitgeteilt werden, welcher TCP-Port abgefragt werden soll. Bemerkung: Die lokale IP-Adresse ist im Header-File `tcpip.h` abgelegt.
Ein Beispiel:

```
TCPLocalPort = 80; //    Port eines HTTP-Servers
TCPPassiveOpen(); // warten auf Verbindung...
```

Wenn ein Client den im Server-Modus betriebenen Stack kontaktiert, und die Verbindung erfolgreich hergestellt ist, wird das durch Setzen des Status-Flags `SOCK_CONNECTED` gemeldet.

◆ `void TCPActiveOpen(void)`
Stellt eine Verbindung zu einem Remote-TCP her. Das Status-Flag `SOCK_ACTIVE` wird gesetzt, und ein *ARP*-Request zum Ermitteln der Ethernet-MAC-Adresse des anderen TCPs gesendet. Wenn die Ziel-IP nicht im Subnetz liegt, wird das Gateway adressiert (beides im Headerfile `tcpip.h` konfigurierbar). Nach dem Herstellen der Verbindung, der gegenseitigen Synchronisation beider TCPs und dem Übergang des lokalen TCP in den Zustand ***ESTABLISHED*** wird wiederum das Status-Flag `SOCK_CONNECTED` gesetzt. Danach kann die Datenübertragung durch Nutzung der entsprechenden API-Funktionen erfolgen. Schlägt die Funktion fehl, weil z.B. der Ziel-Host nicht erreichbar ist, wird eine entsprechende Fehlermeldung im Register `SocketStatus` (s.u.) hinterlegt. Wie beim passiven Öffnen ist hier der lokale TCP-Port, zusätzlich die Ziel-IP-Adresse und der Ziel-Port festzulegen.
Ein Beispiel für die Nutzung der Funktion »`TCPActiveOpen`« zeigt Beispiel 9.
Wenn die Baugruppe via Router mit dem Internet verbunden ist, und sowohl Gateway (Router-IP!) wie Sub-

```
*(unsigned char *)RemoteIP = 24;              // Ziel-IP: 24.8.69.7
*((unsigned char *)RemoteIP + 1) = 8;
*((unsigned char *)RemoteIP + 2) = 69;
*((unsigned char *)RemoteIP + 3) = 7;

TCPLocalPort = 2025;                          // Lokaler Port ist beliebig (>1024)
TCPRemotePort = 17;                           // Port: »Zitat des Tages«

TCPActiveOpen();                              // Verbindung herstellen
```

Beispiel 9: Herstellen einer Verbindung zu einem Remote-IP mit der Funktion »`TCPActiveOpen`«

Bild 22 Verwendung von »DoNetworkStuff()«

netzmaske richtig konfiguriert sind, kann der Anwender nach dem Ausführen dieser Zeilen das »Zitat des Tages« eines Internet-Servers im Empfangspuffer bewundern. In Bild 23 ist ein Screenshot des Debuggers »C-SPY« von IAR Systems nach dem Beenden der Verbindung dargestellt. In der Speicherdarstellung ist ab Adresse »0x04A6« (Beginn des Empfangspuffers `RxTCPBuffer`) solch ein Zitat unmittelbar nach dem Übertragen dargestellt.

- `void TCPClose(void)`

Diese API-Funktion dient dem Trennen der aktuellen Verbindung. Zuvor werden eventuell noch im Sendepuffer stehende Daten ordnungsgemäß übertragen. Nach Ausführung der Funktion lässt sich beispielsweise der Stack umkonfigurieren (IP-Adressen, Ports) und eine neue Verbindung aufbauen.

- `void TCPReleaseRxBuffer (void)`

Diese Funktion teilt dem Stack nach dem Auslesen des Empfangspuffers mit, dass die dort enthaltenen Daten nicht mehr benötigt werden. Der Stack kann dann neue Daten entgegennehmen, im Empfangspuffer ablegen und ein entsprechendes Bestätigungssegment versenden. Ein Flag im Statusregister `SocketStatus` zeigt das Vorhandensein neuer Daten an.

- `void TCPTransmitTxBuffer (void)`

Mit dieser API-Funktion kann die Applikation ein Datensegment über die geöffnete Verbindung versenden. Zunächst ist zu prüfen, ob das Anwendungsprogramm den Sendepuffer mit Daten füllen darf. Dazu ist das Flag `SOCK_TX_BUF_RELEASED` im Statusregister `SocketStatus` auszuwerten. Wenn dieses Flag gesetzt ist, darf der Vorgang fortgesetzt werden. Zur Adressierung des Beginns des Datenbereiches innerhalb des `TxFrame1`-Puffers dient der Zeiger `TCP_TX_BUF`. Von dieser Speicherzelle beginnend,

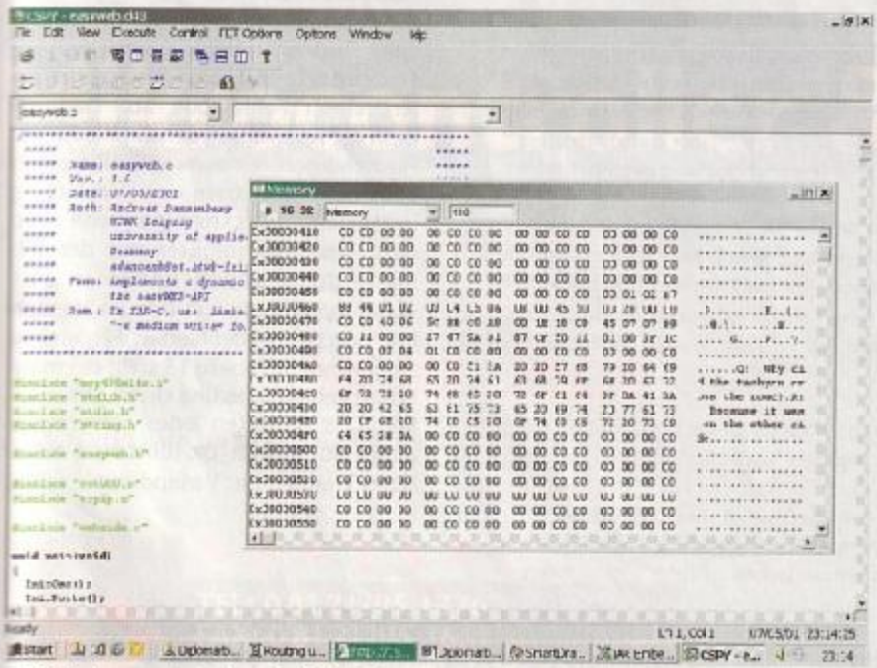


Bild 23: Screenshot des »C-SPY«-Debuggers mit dem »Zitat des Tages«

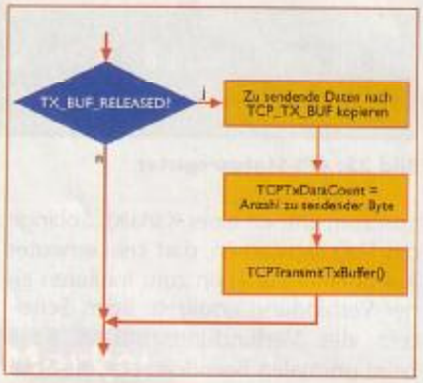


Bild 24: Senden von Daten

kann die Applikation eine maximale Anzahl von `MAX_TCP_TX_DATA_SIZE` Bytes ablegen, um die reservierte Puffergröße nicht zu überschreiten. Im Anschluss daran ist die Anzahl zu sendender Bytes dem Stack durch Eintragen des entsprechenden Werts in die Variable `TCPTxDataCount` mitzuteilen, und das Senden letztendlich durch Aufruf der API-Funktion auszulösen (Bild 24).

Statusflags

Die API-Statusflags werden in der globalen 8-Bit-Variablen `SocketStatus` angegeben. Auf dieses Register ist ausschließlich Lesezugriff zulässig. Die Position der im Folgenden beschriebenen Flags ist Bild 25 auf Seite 36 zu entnehmen.

- `SOCK_ACTIVE` (Bit 0)

Dieses Flag ist gesetzt, wenn der TCP/IP-Stack gerade mit dem Aufbau einer Verbindung beschäftigt ist (vorangegangener Aufruf einer API-Funk-

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Error_3	Error_2	Error_1	Error_0	TX BUF RELEASED	DATA AVAILABLE	CONNECTED	ACTIVE

Bild 25: API-Statusregister

tion zum Öffnen eines Kanals). Solange das Flag aktiviert ist, darf kein erneuter Aufruf einer Funktion zum Initiieren einer Verbindung erfolgen. Beim Scheitern des Verbindungsaufbaus bzw. beim normalen Beenden setzt die Software auch dieses Flag zurück. Ein eventuell aufgetretener Fehler wird im oberen Nibble (Halbbyte) des Status-Registers (`ERROR`) ausgegeben.

◆ `SOCK_CONNECTED` (Bit 1)
Gibt an, dass sich der TCP-Stack im Zustand ***ESTABLISHED*** befindet. Bei gesetztem Flag dürfen unter Benutzung der entsprechenden API-Funktionen Daten übertragen werden. Beim Beenden oder Abbrechen einer Verbindung wird dieses Flag zurückgesetzt

◆ `SOCK_DATA_AVAILABLE` (Bit 2)
Mit diesem Flag wird die Anwendung über den Empfang eines neuen TCP-Datensegment unterrichtet. Das Programm kann daraufhin den Empfangspuffer auslesen. Der Zeiger `TCP_RX_BUF` zeigt dabei auf den Beginn des Datensegments und das Register `TCPRxDataCount` enthält die Anzahl dort abgelegter Datenbytes. Nach dem Auslesen des Datenbereiches sollte der Empfangspuffer wieder freigegeben werden, um dem Stack das Ablegen neuer Daten zu erlauben (Bild 26).
Werden über einen längeren Zeitraum keine Daten abgeholt bzw. der Empfangspuffer nicht freigegeben, kann dies – abhängig auch vom TCP-Stack des Kommunikationspartners – zum Abbruch der Verbindung führen.

◆ `SOCK_TX_BUF_RELEASED` (Bit 3)
Dieses Flag gibt an, ob das Anwenderprogramm den Sendepuffer bzw. das

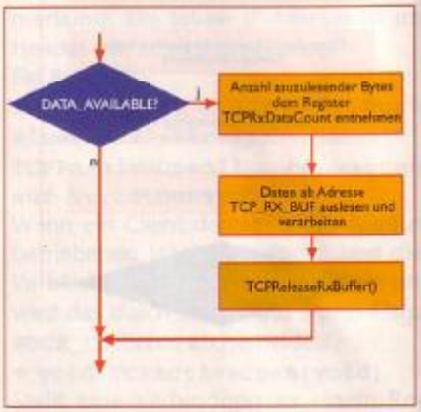


Bild 26: Empfangen von Daten

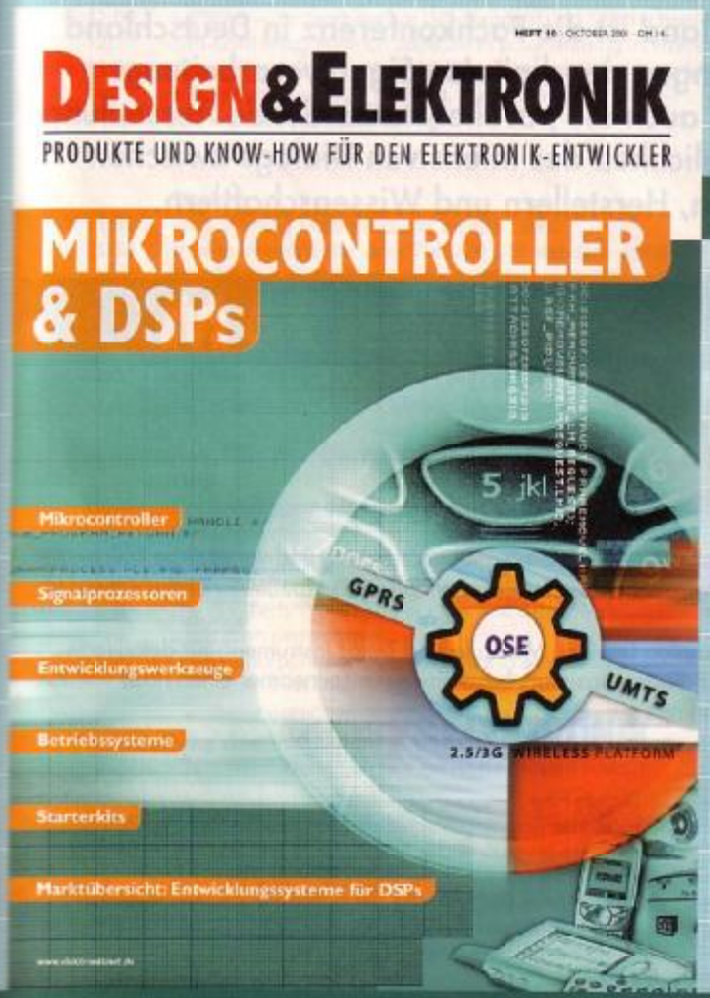
Register `TCPTxDataCount` ändern darf. Erst wenn der Empfang des gesendeten TCP-Daten-Frames durch den TCP-Stack des Kommunikationspartners bestätigt wurde, wird dieses Flag gesetzt, und somit der Sendepuffer der Applikation zurückgegeben. Dadurch wird ein Handshakemechanismus realisiert, und der Sendepuffer somit für den Fall, vor Zerstörung geschützt, dass dessen Daten noch benötigt werden könnten. Die allgemeine Verfahrensweise beim Senden von Daten ist in der Beschreibung der API-Funktion `TCPTransmitTxBuffer()` aufgeführt.

◆ `ERROR_CODE` (Bits 4 bis 7)
Beim Auftreten eines Fehlers legt der Stack im oberen Nibble des Statusregisters einen Fehlercode ab. Dadurch kann die Applikation den Grund für das Fehlschlagen eines Verbindungsaufbaus oder das Lösen der Verbindung feststellen. Zweckmäßigerweise wird dazu das Statusregister mit der Bitmaske `SOCK_ERROR_MASK (0xF0)` »UND«-verknüpft, um den im Header-File `tcpip.h` definierten Fehlercode zu gewinnen. Tabelle 13 stellt die möglichen Fehlercodes und deren Ursachen vor. Das Auftreten jedes Fehlercodes außer `SOCK_ERR_OK` führt zum sofortigen Trennen der Verbindung. (cg)

Fehlercode	Bedeutung/Ursache
`SOCK_ERR_OK`	kein Fehler aufgetreten
`SOCK_ERR_ARP_TIMEOUT`	Fehler beim ARP-Request aufgetreten. Die MAC-Adresse des anderen Hosts konnte nicht ermittelt werden. Der Host ist entweder nicht mit dem Netzwerk verbunden oder nicht in der Lage zu antworten.
`SOCK_ERR_TCP_TIMEOUT`	Fehler bei der TCP-Verbindung aufgetreten. Trotz mehrmaligen Sendens wurde das Segment durch das andere TCP nicht bestätigt. Kann durch häufigen Verlust von Segmenten bei einer sehr schlechten Verbindung auftreten, aber auch durch eine permanente Unterbrechung des Netzpfades zum Remote-Host.
`SOCK_ERR_CONN_RESET`	Aufzubauende oder bestehende Verbindung wurde durch den Remote-Host zurückgesetzt. Entweder »wünscht« das Remote-TCP keinen Kontakt mit dem gewählten Port, oder die Anwendungsschicht des Kommunikationspartners hat die Verbindung zurückgesetzt (z.B. Betätigen des »STOP«-Buttons eines Internet-Browsers).
`SOCK_ERR_REMOTE`	Ein schwerwiegender Softwarefehler des Remote-TCP führte zum Senden eines ungültigen Segments.
`SOCK_ERR_ETHERNET`	Der Stack konnte über das Ethernet keine Daten senden (es wurde kein Pufferspeicher bereitgestellt). Dieser Fehler tritt beim Trennen des Netzwerkkabels auf (LINK geht verloren).

Tabelle 13: Fehlercodes des TCP/IP-Stacks

Leicht kalkulierbar!



Merke:

90% Entwickler
in der Bezieherschaft
+ 90% Kaufabsichten
im Bereich aktive Bauelemente

=100% Werbeerfolg

Quelle: INTERSEARCH 2001

Mikrocontroller & DSPs

ANZEIGENSCHLUSS: 07|09|2001

ERSCHEINUNGSTERMIN: 08.10.2001

KONTAKT

TEL: 0 81 21/95-1376
FAX: 0 81 21/95-1651
cstadler@design-elektronik.de

www.elektroniknet.de/media

☐ Ja, ich will nähere Informationen zur Anzeigenschaltung im Themenheft MIKROCONTROLLER & DSPs

☐ Bitte senden Sie mir ein Themenheft MIKROCONTROLLER & DSPs für DM 14.– plus Versandkosten.

Firma

Ansprechpartner

Straße

PLZ/Ort

Telefon

Fax

E-Mail

WEKA Fachzeitschriften-Verlag GmbH · Gruber Straße 46 a · 85586 Poing

DSP Deutschland 22.–23.10. 2001 M,O,C, MÜNCHEN

MESSE&KONGRESS

Im M,O,C, München, Lilienthalallee 40, 80939 München

Die DSP Deutschland ist die Fachkonferenz in Deutschland für Anwendungen der digitalen Signalverarbeitung. Der Kongress wie auch die parallel stattfindende Ausstellung bieten die Möglichkeit des intensiven Dialogs zwischen Anwendern, Herstellern und Wissenschaftlern.

KONGRESSPROGRAMM

Montag, 22. Oktober, 09:00–16:30

Workshop: Praktische Einführung in die digitale Signalverarbeitung, Prof. Dr. Matthias Sturm HTWK Leipzig

Dieser Workshop leistet eine praktische Einführung in das Gebiet der digitalen Signalverarbeitung und versetzt die Teilnehmer in die Lage, Aufbau und Funktion von DSP-Systemen zu verstehen und eigene Lösungen auf Basis von Starterkits zu entwickeln, zu programmieren, zu implementieren und zu testen.
Im letzten Drittel des Kurses werden Vertreter der Hersteller Analog Devices, Motorola und Texas Instruments für detaillierte Fragen zu ihren DSP-Architekturen zur Verfügung stehen. Mit diesem Basiskurs bekommen die Teilnehmer einen (fast) vollständigen Einstieg in die Signalverarbeitung zu einem moderaten Preis.

Die Themen im Workshop:

- **Einführung in die digitale Signalverarbeitung**
  Grundlagen der digitalen Signalverarbeitung mit Applikationsbeispielen.
- **Analog-Digital-Wandlung**
  Erklärung der theoretischen Grundlagen der A/D-Wandlung und der daraus resultierenden praktischen Schlußfolgerungen. Praktische Demonstrationen vertiefen die erläuterte Thematik.
- **Signalprozessor-Architekturen**
  Darstellung der Architektur von Signalprozessoren, ihres inneren Aufbaus und der Funktion der einzelnen Baugruppen. Umsetzung der Architekturmerkmale in ausgewählten DSPs von Analog Devices, Motorola und Texas Instruments.
- **DSP-Praxis**
  Präsentation und Erläuterung einfacher Signalverarbeitungsvorgänge aus dem Audiobereich anhand praktischer Experimente. Darstellung des Wegs von der Idee bis zur praktischen Umsetzung.
- **Digitale Filter**
  Theoretische Grundlagen sowie die praktische Umsetzung digitaler Filteralgorithmen in verständlicher Form.
- **Aktuelle Signalprozessoren, Einsatzschwerpunkte**
  Sachkompetente Gastreferenten der Firmen Analog Devices, Motorola und Texas Instruments sprechen über Eigenschaften und Applikationsschwerpunkte ihrer Signalprozessorfamilien.
- **Diskussion**
  Prof. Dr. Sturm sowie die Gastreferenten der genannten Firmen stellen sich der Diskussion der Kursteilnehmer.
- **Verlosung von Starterkits**
  Die Kursteilnehmer haben die Gelegenheit, im Rahmen einer Verlosung einige der vorgestellten DSP-Starterkits zu gewinnen.

Kurs 1: Designmethoden — Dienstag, 23. Oktober

Zeit	Titel	Autor
09:00–09:30	From Knowledge to performance: practical DSP system optimisation using bayesian inference	Desmond K. Phillips, Cambridge Consultants
09:30–10:00	Implementierung eines matched Filter auf einem FPGA	Prof. Dr. Alfred Marganitz, TFH Berlin
10:00–10:30	Bridging the productivity gap in high performance DSP system design	Bernd Niedermeier, Altera
10:30–11:00	**Kaffeepause, Ausstellung**	
11:00–11:30	A fast method for computing decision feedback equalizer coefficients using a fixed point DSP	Ahsan Aziz, Motorola
11:30–12:00	Programmierbare Analogbauteile machen DSP-Design einfacher und flexibler	Johannes Fottner, Lattice Semiconductor
12:00–12:30	Debug analog interfaces using real-time DSP tools	Frank Walzer, Texas Instruments
12:30–13:00	Portable DSP-Bibliothek für die Audio- und Videokompression	Dr. Clemens Westerkamp, Algo Vision Systems
13:00–14:30	**Mittagspause, Ausstellung**	
14:30–15:00	Integrated SoC development framework for a Carmel DSP based application	Holger Neumann, Infineon
15:00–15:30	Extending an Embedded RISC Processor Soft Macro with DSP Functionality	Alexander Hahn, ARC International
15:30–16:00	**Kaffeepause, Ausstellung**	
16:00–16:30	A modular approach to DSP farms or multiple DSP applications	Charles Spurr, InterWorks
16:30–17:00	Some hints on DSP software development & testing	Alex Motrenko, Spirit

KONGRESSPROGRAMM

Kurs 2: Applikationen — Dienstag, 23. Oktober

Zeit	Titel	Autor
09:00–09:30	Netzwerkintegration von digitalen Signalprozessoren	Klaus-D. Walter, SSV
09:30–10:00	DSP & Communications – Using an IP stack on a DSP	James Campbell, Precise Software Technologies Prashanth M., Accord Software & Systems
10:00–10:30	File transfer using Zmodem protocol on DSP platform	
10:30–11:00	**Kaffeepause, Ausstellung**	
11:00–11:30	MP3-Musik aus der Steckdose	Prof. Dr. Francesco Volpe, FH Aschaffenburg
11:30–12:00	MP3PRO – die Synthese aus ISO-Standard und SBR-Technologie	Thomas Ziegler, Coding Technologies
12:00–12:30	GPS-GSM integration on application programming interface	S. Kiran, Accord Software & Systems
12:30–13:00	Connecting a DSP to 10/100 MBit/s Ethernet Using spare MIPS to run a TCP/IP stack with HTTP, TFTP, telnet services	Andre Schnarrenberger, Texas Instruments
13:00–14:30	**Mittagspause, Ausstellung**	

Kurs 3: Software-Entwicklung — Dienstag, 23. Oktober

Zeit	Titel	Autor
14:30–15:00	Programming Multi-processor architectures	Peter Leyssens, Windriver
15:00–15:30	How to integrate the same TMS320 DSP standard compliant algorithm in a dynamic DSP system and a static DSP system?	Elizabete De Freitas, Texas Instruments
15:30–16:00	**Kaffeepause, Ausstellung**	
16:00–16:30	Automatische Erstellung von Software für Signalverarbeitungssysteme mit MATLAB/Simulink	Cord Elias, The MathWorks
16:30–17:00	Optimierung von DSP-Code in der Praxis	Andreas Bayer, Ingenieurbüro für Digitale SV
17:00–17:30	Echtzeitsoftware neuster Mobiler Multiprozessorsysteme mit DSP	Hagen Heggenberger, OSE Systems

Kurs 4: Hardware-Architekturen — Dienstag, 23. Oktober

Zeit	Titel	Autor
14:30–15:00	The digital signal controller – DSP and microcontroller functionality on a single chip	Mike Catherwood/Martin Burghardt, Microchip Günter Plechinger, Hitachi Europe
15:00–15:30	SH-DSP – Inspiring the next personal access product	
15:30–16:00	**Kaffeepause, Ausstellung**	
16:00–16:30	The XPP-Technology: Turning task and dataflow parallelism into application performance	A. Nückel, PACT
16:30–17:00	FPGA Concept of PowerPlug Module as an extension of the Carmel Execution Unit	Holger Neumann, Infineon

Eine Veranstaltung der DESIGN&ELEKTRONIK — PRODUKTE UND KNOW-HOW FÜR DEN ELEKTRONIK-ENTWICKLER

Arthur D Little · Cambridge Consultants · The MathWorks · SPOERLE AN ARROW COMPANY · SMA · OSE systems An Esterel Company · Texas Instruments · BAYER Digital Signal Processing · HITACHI Inspire the Next · orsys · Altium

Fax-Antwort 0 81 21 / 95 16 54

Ich melde mich verbindlich an:

Montag: ❑ DSP-Basiskurs, Teilnahmegebühr DM 430,–

Dienstag: Teilnahmegebühr DM 490,–

Ganztägig:
❑ Kurs 1, Designmethoden

Vormittag:
❑ Kurs 2, Applikationen

Nachmittag:
❑ Kurs 3, Software-Entwicklung

oder
❑ Kurs 4, Hardware-Architekturen

Teilnahmegebühr 2-Tages-Besuch DM 820,–

❑ Bitte senden Sie mir eine Anfahrtsskizze

Firma

Vorname Name

Straße

PLZ Ort

Telefon Fax

Email

Datum Unterschrift DSP/DE/EX

Die Preise verstehen sich zuzügl. der gesetzl. MwSt. In diesem Betrag enthalten sind Tagungsunterlagen (Kongress) sowie Mittagsessen und Pausengetränke. Studenten (50% Rabatt) bitte Immatrikulationsbescheinigung beilegen. Bei Anmeldung bis 5 Tage vor Kongressbeginn erhalten die Teilnehmer eine Teilnahmebestätigung zusammen mit der Rechnung. Anmeldungen sind auch vor Ort möglich, es werden nachträglich Rechnungen ausgestellt. Bei Stornierung der Anmeldung bis 10 Tage vor Veranstaltungsbeginn erheben wir eine Bearbeitungsgebühr von DM 100,– (zzgl. MwSt.), bei späterer Absage oder Nichterscheinen wird die gesamte Tagungsgebühr fällig.
Info: Hilde Buchner, Tel.: 08121/95-1345, Fax: 08121/95-1654, Hbuchner@design-elektronik.de · **WEKA-Fachzeitschriften-Verlag, 85586 Poing, Gruber Straße 46a**

Ein HTTP-Server als Applikationsbeispiel

Dem Ziel ganz nah

Als Applikationsbeispiel für die vorgestellte Hard- und Softwareplattform dient ein HTTP-Server. Dieser Server soll eine im Flash-EEPROM des Mikrocontrollers abgelegte HTML-Seite zur Verfügung stellen. Im Einzelnen muss die Baugruppe auf das Herstellen einer Verbindung durch einen Client (Internet-Browser) warten, dann die HTML-Seite zum Client übertragen, die Verbindung wieder beenden und auf eine neue Verbindungsanfrage warten. Dabei soll die Webseite dynamisch aufgebaut sein, d.h. ihren Inhalt selbstständig während der Übertragung an bestimmte Größen anpassen. Konkret sind auf der Webseite zwei Werte des im MSP430F149 integrierten A/D-Wandlers mittels einer Balkenanzeige dargestellt. Weiterhin besteht der Wunsch nach einer automatischen Aktualisierung der angezeigten Webseite.

Ganz allgemein betrachtet, funktioniert ein HTTP-Server nach folgendem Schema: Zunächst wird der eingesetzte TCP/IP-Stack in den Zustand *LISTEN* versetzt, was dem passiven Öffnen des Kanals und damit dem Warten auf eine eingehende Verbindung entspricht. Dabei wird der lokale TCP-Port des Stacks der Serversoftware auf »80« (dezimal) gesetzt. Dieser Wert ist der Standardport eines jeden HTTP-Servers. Nach Eingabe eines URIs (s. Seite 19) in einen Internet-Browser versucht dieser nämlich automatisch, eine Verbindung zu Port 80 des angesprochenen Servers herzustellen.
Der Quellcode des fertigen Webservers ist im Softwaremodul `easyweb.c` (ab Seite 49) enthalten. Dieses Modul stellt praktisch ein Anwendungsprogramm dar, welches die Funktionen der darunter liegenden Softwareschichten (TCP/IP-API) nutzt. Das Hauptprogramm wurde mit dem im Kapitel API-Funktionen geforderten, nicht-blockierenden Programmierstil erstellt, um den zyklischen Aufruf der Funktion `DoNetworkStuff()` sicher zu stellen (Bild 27).

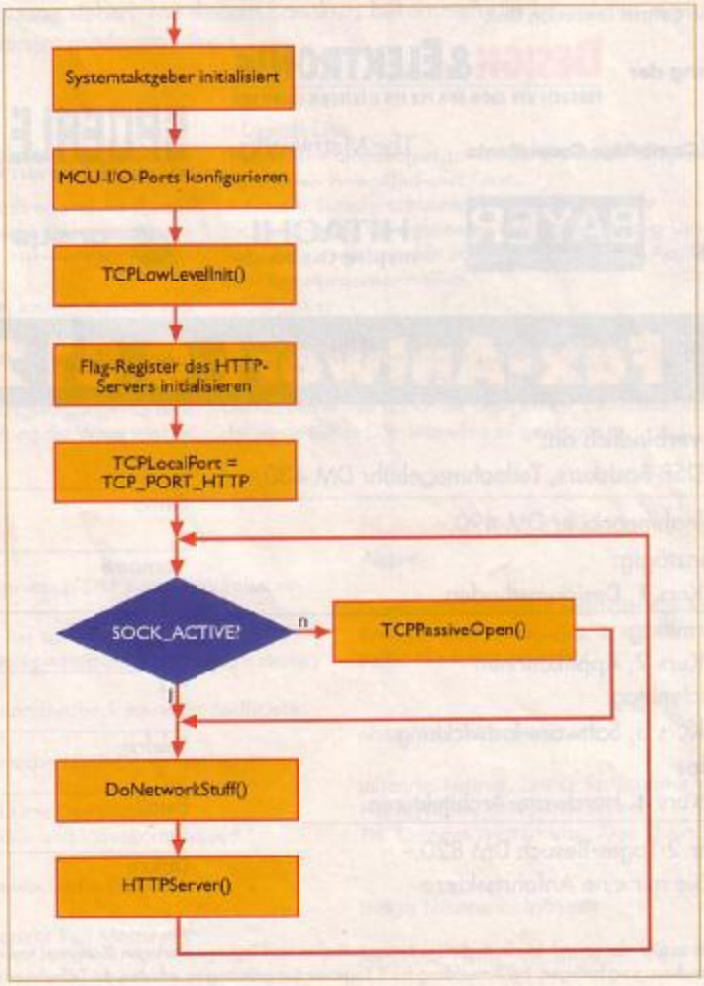


Bild 27 Webserver-Hauptprogramm

HTTP-Server im Detail

Der eigentliche HTTP-Server verbirgt sich hinter der Funktion `HTTPServer()`, der Programmablaufplan dieser Funktion ist in Bild 28 auf Seite 42 dargestellt. Zunächst wartet die Webserver-Routine auf das Herstellen einer Verbindung durch einen Client. Beim ersten Durchlauf nach dem Verbindungsaufbau ist das Flag `HTTP_SEND_PAGE` im Register `HTTPStatus` noch gelöscht. Es dient zur Abarbeitung besonderer Quelltextabschnitte direkt nach Herstellen einer Verbindung. Danach überprüft die Software den Empfangspuffer auf eingegangene Daten, eventuell empfangene Daten werden aber durch sofortiges Freigeben des Empfangspuffers praktisch »gelöscht«. Im Regelfall enthalten die hier empfangenen Daten den *GET*-Request des entsprechenden Internet-Browsers. Da der vorliegende Webserver lediglich eine Webseite unterstützt, können wir uns die Auswertung dieses *GET*-Requests sparen. Wir beginnen somit direkt nach dem Verbindungsaufbau mit dem Senden der im Flash-

EEPROM abgelegten Webseite. Der HTML-Code dieser Seite ist als C-Konstante `WebSide[]` im Modul `webside.c` (s. Seite 53) abgelegt.
Nach dem Überprüfen des Status des Sendepuffers wird nun zunächst ein Zeiger auf die Webseite eingerichtet und die gesamte Anzahl der zu sendenden Bytes (Länge der Webseite) in der Variablen `HTTPBytesToSend` eingetragen. Als weitere Besonderheit beim ersten Durchlauf dieser Routine wird der zu sendenden Webseite der »HTTP-Response-Header« vorangestellt. Dieser ist im Header-File `easyweb.h` in der Konstante `GetResponse[]` abgelegt, bestätigt dem Client den erfolgreichen Empfang seines Requests und gibt den Typ der zu übertragenden Internet-Ressource (URI) bekannt (hier HTML). Danach erfolgt das segmentweise Senden der Webseite in Abschnitten, die der Größe des Sendepuffers entsprechen. Nach dem Übertragen des letzten Datensegmentes wird die Verbindung zum Client durch Aufruf der API-Funktion `TCPClose()` wieder beendet. Die Webseite wurde nun komplett und fehlerfrei übertragen. Das Hauptprogramm führt anschließend einen erneuten Aufruf der Funktion `TCPPassiveOpen()` aus, damit kann der nächste Client die Webseite abrufen.
Wie kommt nun die Dynamik der Webseite zustande? Vor dem Senden eines jeden Segments der Webseite erfolgt der Aufruf der Funktion `InsertDynamicValues()`. Diese Funktion durchsucht den Sendepuffer nach speziellen Strings und ersetzt diese durch A/D-Wandlerwerte. Ein solcher String besteht aus vier Bytes: »AD« + Kanalnummer + »%«. Die vorliegende Programmversion ersetzt den String »AD7%« durch den A/D-Wandlerwert des Kanals 7, und den String »ADA%« durch den des Kanals 10. Fügt der Entwickler nun beim Erstellen der HTML-Seitenbeschreibung eine dieser Zeichenfolgen ein, überschreibt die Funktion diese Zeichenfolge vor dem Senden der Seite mit einem Wert von 0% bis 100%. HTML-Kenntnisse vorausgesetzt, lassen sich dadurch auf der Webseite verschiedene Effekte in Abhängigkeit der A/D-Wandlerwerte erzeugen (siehe Abschnitt »Beispiel-Webseite).
Den Prozentwert für den Wandlerkanal 7 erzeugt die Funktion

Beliebig ausbaufähig

Die Autoren sind sich sicher, dass die Publikation des vorliegenden Projekts zum Abbauen von Berührungsängsten mit dem »Mythos« TCP/IP-Protokoll beiträgt. Auch auf diesem Gebiet wird letzten Endes nur mit Wasser gekocht.
Trotz verschiedener durch die Rahmenbedingungen gegebener Einschränkungen im Umfang des Protokollstacks (z.B. kein Zusammenfügen von fragmentierten IP-Datagrammen, kein Multi-User Support und Beschränkung auf eine konkrete Segmentlänge bei der Übertragung durch das TCP) trat während der Testphase der Entwicklung nicht ein Fall auf, in dem sich keine Kommunikation mit einem beliebigen anderen TCP herstellen ließ.
Durch komplette Programmierung der Software in C ist deren Funktionsweise für einen geübten Programmierer nach kurzer Einarbeitung in die Materie nachvollziehbar. Mit dem Wissen aus der hier vorgestellten Entwicklung, und ergänzt durch erste eigene Erfahrungen, können Entwickler sowohl Hard- als auch Software in verschiedenster Art erweitern.
So ließe sich beispielsweise durch Nutzung eines 16-Bit-Datenbusses zwischen MCU und Ethernet-Controller die Möglichkeit der Interruptauslösung im Mikrocontroller beim Auftreten eines Netzwerk-Ereignisses nutzen. Dadurch wäre eine effizientere Gestaltung der Protokoll-Implementierungen möglich. Weiterhin ließe sich die Software durch den gezielten Einsatz ausgewählter Assemblerroutinen (z.B. zur Checksummenberechnung) im Hinblick auf Geschwindigkeit optimieren. Es sind jedoch auch Erweiterungen der Webserver-Applikation denkbar, wie z.B. die Ablage mehrerer Webseiten im Flash-Speicher der MCU, die Möglichkeit des Schaltens digitaler Ausgänge am Mikrocontroller durch Betätigen von Buttons auf der Webseite, oder die Fremdprogrammierung der Webseite mittels FTP-Protokoll.
Es besteht auch die Möglichkeit, die auf der Platine vorgesehene RS232-Schnittstelle als Alternative zur Ethernet-Anbindung für die serielle Übertragung von TCP/IP-Daten zu nutzen (Protokoll PPP über Modem, SLIP-Direktverbindung zu einem Server). Bei Bereitstellung von größeren RAM-Bereichen wäre auch eine umfassendere Implementierung der schon eingesetzten Protokolle bzw. die Einbindung weiterer Protokolle möglich (z.B. UDP). Diese Liste mit Ideen zur Vervollständigung bzw. Ergänzung dieser Arbeit ließe sich fast beliebig fortsetzen.
Lassen Sie Ihre Phantasie spielen!

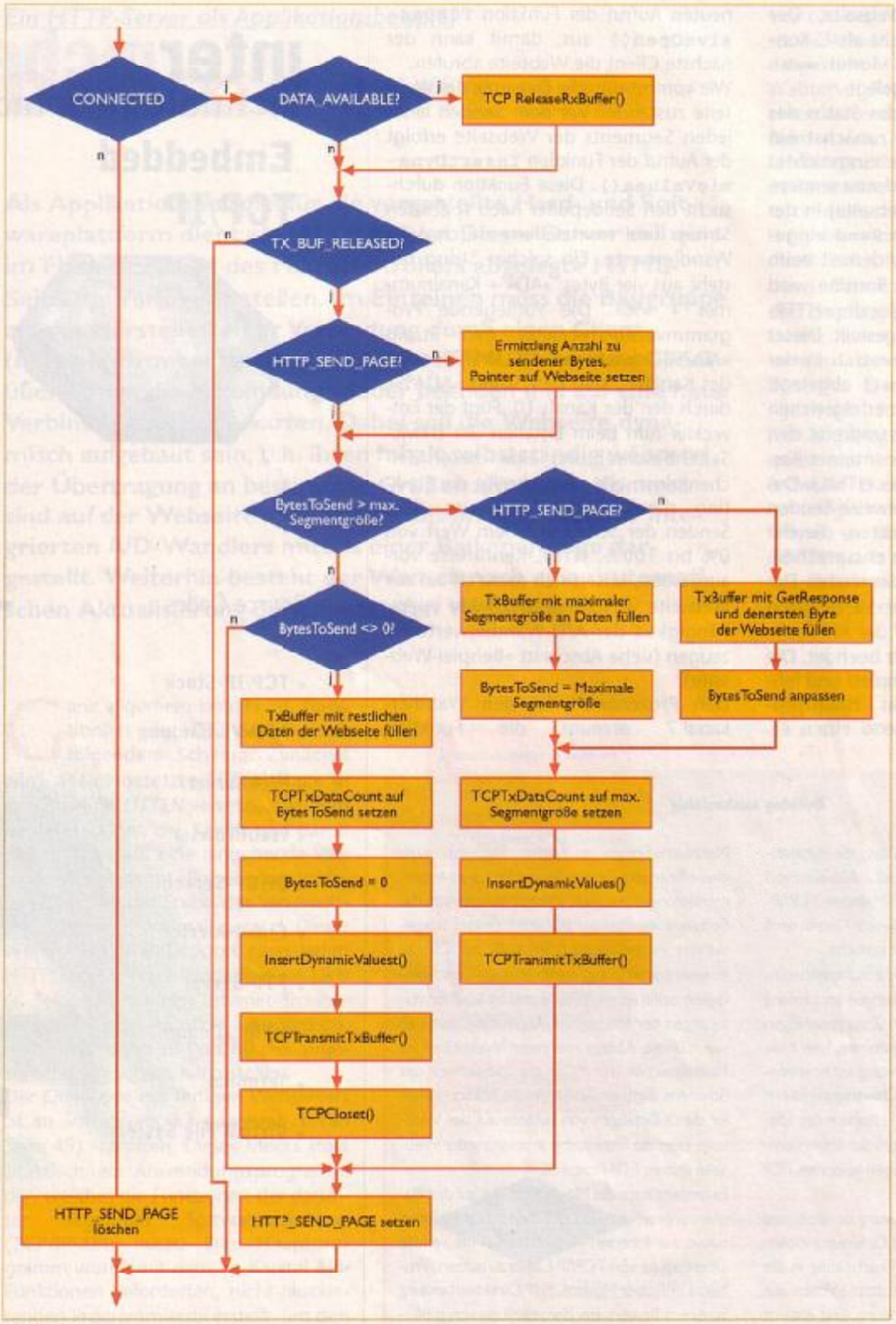


Bild 28: Routine »`HTTPServer()`«

`GetAD7Val()`. Diese konfiguriert den A/D-Wandler so, dass er eine interne Referenzspannung von 2,5 V verwendet. Da er direkt mit dem Portpin P6.7 verbunden ist, kann nun nach dem Starten des Wandlungsprozesses ein 12-Bit-Wert aus dem Registerblock des A/D-Moduls des MSP430F149 ausgelesen werden. Dieser Wert stellt die gemessene Eingangsspannung im Bereich von 0 V bis 2,5 V als Integer-Zahl von 0 bis 4095 dar. Die Umrechnung auf einen Prozentwert erfolgt durch einfaches Dividieren.

Der A/D-Kanal 10 hingegen stellt beim MSP430F149 eine Besonderheit dar. Er ist mit einer Chip-internen Referenzdiode verbunden und lässt sich zur Temperaturmessung nutzen. Die Auswertung dieser Temperaturmessung übernimmt die Funktion `GetTempVal()`. Zu diesem Zweck wird eine interne Wandler-Referenzspannung von 1,5 V eingestellt und zur Erhöhung der Genauigkeit eine Serie von acht Messwerten erfasst. Anschließend rechnet die Funktion den berechneten Mittelwert hieraus mit einer Formel wiederum auf einen Prozentwert um. Insgesamt ergibt sich damit die Abbildung einer Temperatur von 20 °C bis 45 °C in den Bereich von 0% bis 100%. Auf Grund von Exemplarstreuungen und Fertigungstoleranzen des Mikrocontroller-ICs lässt sich leider keine allgemeingültige Formel für die Umrechnung eines Wandlerwertes zu einer konkreten Temperatur angeben. Die ausgegeben Werte sind daher mit Vorsicht zu genießen, sie reichen für diese als Demonstration gedachte Applikation jedoch völlig aus.

Beispiel-Webseite

Nun aber in die Praxis: Eine einfache HTML-Seite soll sich die gerade beschriebene Methode des Ersetzens von speziellen Strings durch A/D-Wandlerwerte zu nutze machen. Hierfür soll sie neben der Darstellung eines allgemeinen Textes zwei Balkenanzeigen auf-

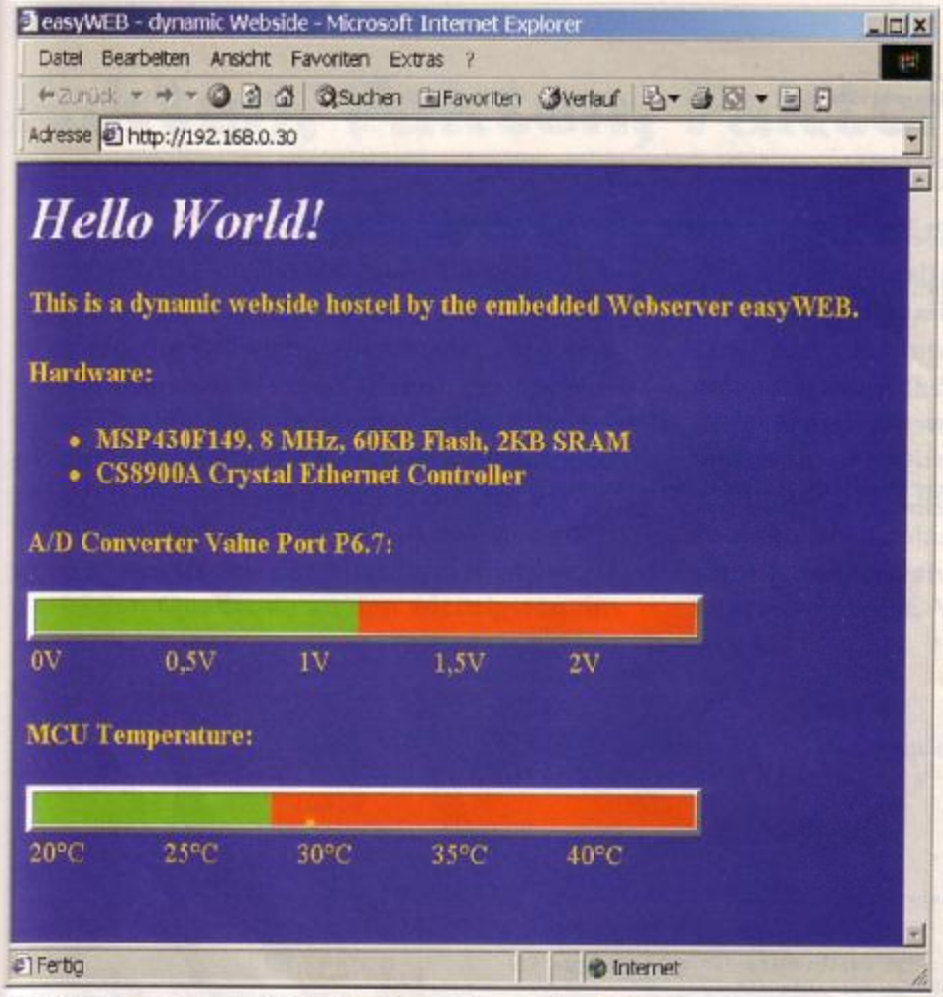


Bild 30: Screenshot der Webseite, dargestellt mit dem Internet-Explorer

weisen. Dabei soll ein Balken den A/D-Wandlerwert des Kanals 7 (Spannung am Portpin P6.7) und der andere die aktuelle Temperatur der MCU anzeigen (Kanal 10 des A/D-Wandlers). Zur effizienten Realisierung von Balkengrafiken mittels HTML-Code eignen sich Tabellen sehr gut. Hierfür werden zwei Tabellen mit einer bestimmten Breite und der Hintergrundfarbe Rot erzeugt. Innerhalb dieser Tabellen wird jeweils eine von links beginnende Spalte angelegt, deren Hintergrundfarbe grün ist. Die Breite dieser Spalten lässt sich mit einem Prozentwert auf die Gesamtbreite der Tabelle beziehen. An der Stelle, an der diese Prozentwerte definiert sind, werden unsere zwei speziellen, auszutauschenden Strings eingefügt. Damit ist eine Visualisierung von sich ändernden, physikalischen Größen realisiert. Der in diesem Projekt integrierte HTML-Code ist in Bild 29 (Seite 44) dargestellt.

Im Allgemeinen besteht der Wunsch, dass sich diese Darstellung nach Ablauf eines bestimmten Zeitintervalls selbstständig aktualisiert, um sich ändernde Größen darstellen zu können. Dazu kann ein automatisches Neuladen der Seite durch den Internet-Browser veranlasst werden. Das geschieht durch die `refresh`-Zeile im `head`-Abschnitt des HTML-Codes. Die dort eingetragene Zahl »5« gibt das Aktualisierungsintervall in Sekunden an. Der hier vorgestellte HTML-Code ist zu allen gängigen Browsern kompatibel (z.B. Microsoft Internet Explorer, Netscape Navigator). Soll der Browser die Seite aufrufen, muss der Benutzer in das Adressfeld das Übertragungsprotokoll (http://) zusammen mit der IP-Adresse des TCP/IP-Stacks (in diesem Fall 192.168.0.30) eingeben. Wie es aussieht, wenn die vom Webserver bereitgestellte Seite durch einen Internet-Browser aufgerufen und dargestellt wird, zeigt Bild 30.

Ein kleines Extra

Da ein direktes Programmieren von HTML-Code in eine C-Konstante sehr mühsam und unübersichtlich ist (siehe Modul `webside.c`), entstanc während der Softwareentwicklung für den Webserver noch

```
<html>
<head>
  <meta http-equiv="refresh" content="5">
  <title>easyWEB - dynamic Webside</title>
</head>

<body bgcolor="#3030A0" text="#FFFF00">
  <p><b><font color="#FFFFFF" size="6"><i>Hello World!</i></font></b></p>

  <p><b>This is a dynamic webside hosted by the embedded Webserver</b> <b>easyWEB.</b></p>
  <p><b>Hardware:</b></p>
  <ul>
    <li><b>MSP430F149, 8 MHz, 60KB Flash, 2KB SRAM</b></li>
    <li><b>CS8900A Crystal Ethernet Controller</b></li>
  </ul>

  <p><b>A/D Converter Value Port P6.7:</b></p>

  <table bgcolor="#ff0000" border="5" cellpadding="0" cellspacing="0" width="500">
  <tr>
    <td>
      <table width="AD7%" border="0" cellpadding="0" cellspacing="0">
          <tr><td bgcolor="#00ff00"> </td></tr>
      </table>
    </td>
  </tr>
  </table>

  <table border="0" width="500">
  <tr>
    <td width="20%">0V</td>
    <td width="20%">0,5V</td>
    <td width="20%">1V</td>
    <td width="20%">1,5V</td>
    <td width="20%">2V</td>
   </tr>
   </table>

  <p><b>MCU Temperature:</b></p>

  <table bgcolor="#ff0000" border="5" cellpadding="0" cellspacing="0" width="500">
  <tr>
    <td>
      <table width="ADA%" border="0" cellpadding="0" cellspacing="0">
        <tr><td bgcolor="#00ff00"> </td></tr>
      </table>
    </td>
  </tr>
  </table>

  <table border="0" width="500">
  <tr>
    <td width="20%">20°C</td>
    <td width="20%">25°C</td>
    <td width="20%">30°C</td>
    <td width="20%">35°C</td>
    <td width="20%">40°C</td>
  </tr>
  </table>
</body>
</html>
```

Bild 29: HTML-Quellcode der Beispiel-Webseite von Seite 43

ein kleines Hilfsprogramm. Dieses dient dem Konvertieren von HTML-Seiten (Dateinamens-Erweiterung *.htm oder *.html) in C-Quelltext. Es entstand als Kommandozeilenapplikation unter »Borland C++ Builder« für Windows. Der Quellcode sollte jedoch ohne Schwierigkeiten auch auf andere Computersysteme übertragbar sein. Das Programm wird auf einem PC unter Benutzung der Eingabeaufforderung mit zwei Parametern gestartet. Der erste Parameter gibt den Dateinamen der zu konvertierenden HTML-Seite an, der zweite den der zu erzeugenden C-Datei.

Beispiel:

```
webconverter.exe index.htm
webside.c
```

Dass sich die gewünschte Anwendung mit diesem Tool direkt in »echtem« HTML-Code schreiben und im Anschluss komfortabel konvertieren lässt, bringt für den Entwickler einen weiteren großen Vorteil mit sich: In der Entwicklungsphase, während des Feintunings der Anwendung, lässt sich die Datei in einen Internet-Browser auf dem lokalen Rechner laden und sofort prüfen. Das vermeidet das aufwändige und zeitraubende ständige Neuprogrammieren der Baugruppe.

Der Quelltext des Tools ist dem Anhang (Seite 70) zu entnehmen. (cg)

Schaltpläne, Listings & Co.

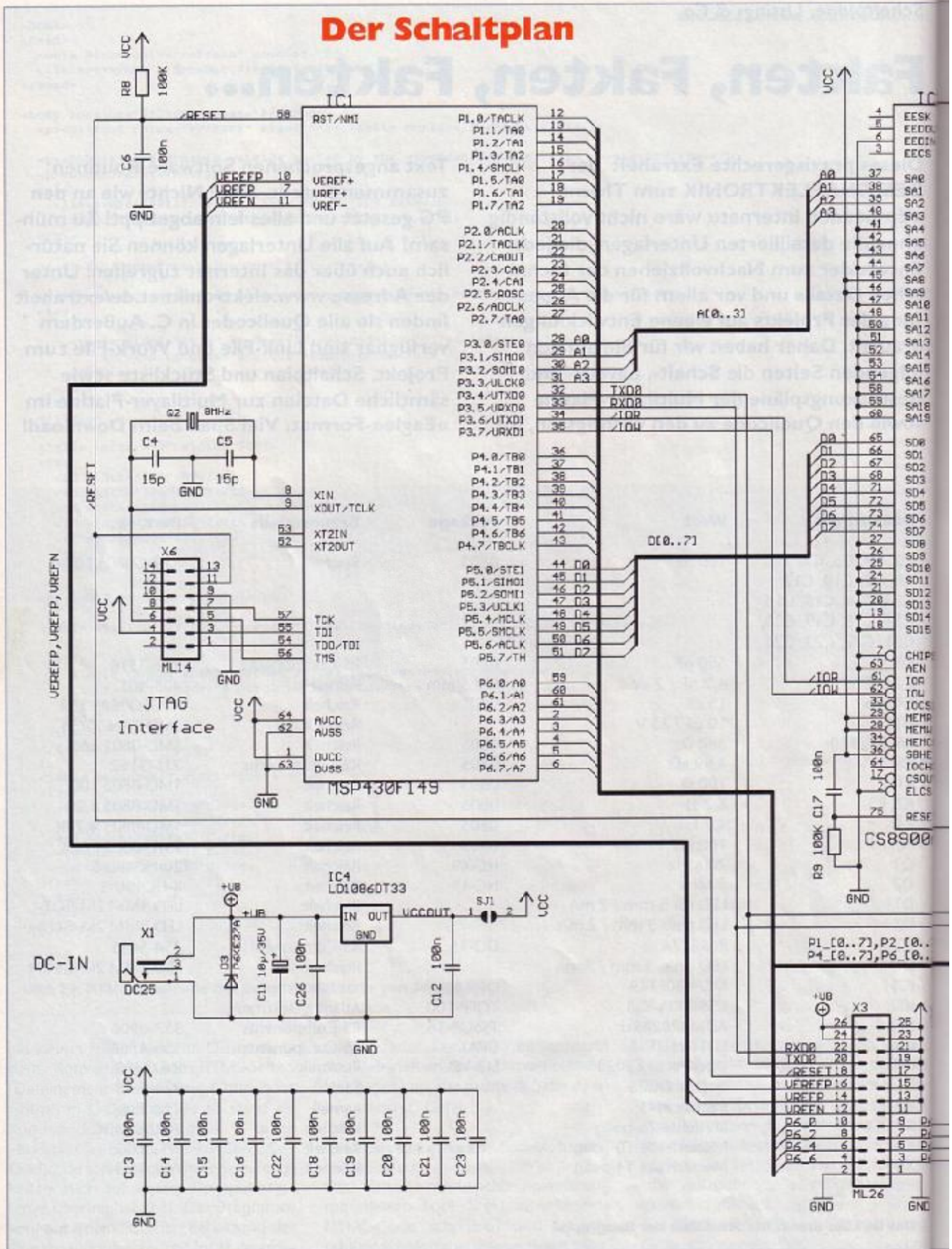
Fakten, Fakten, Fakten...

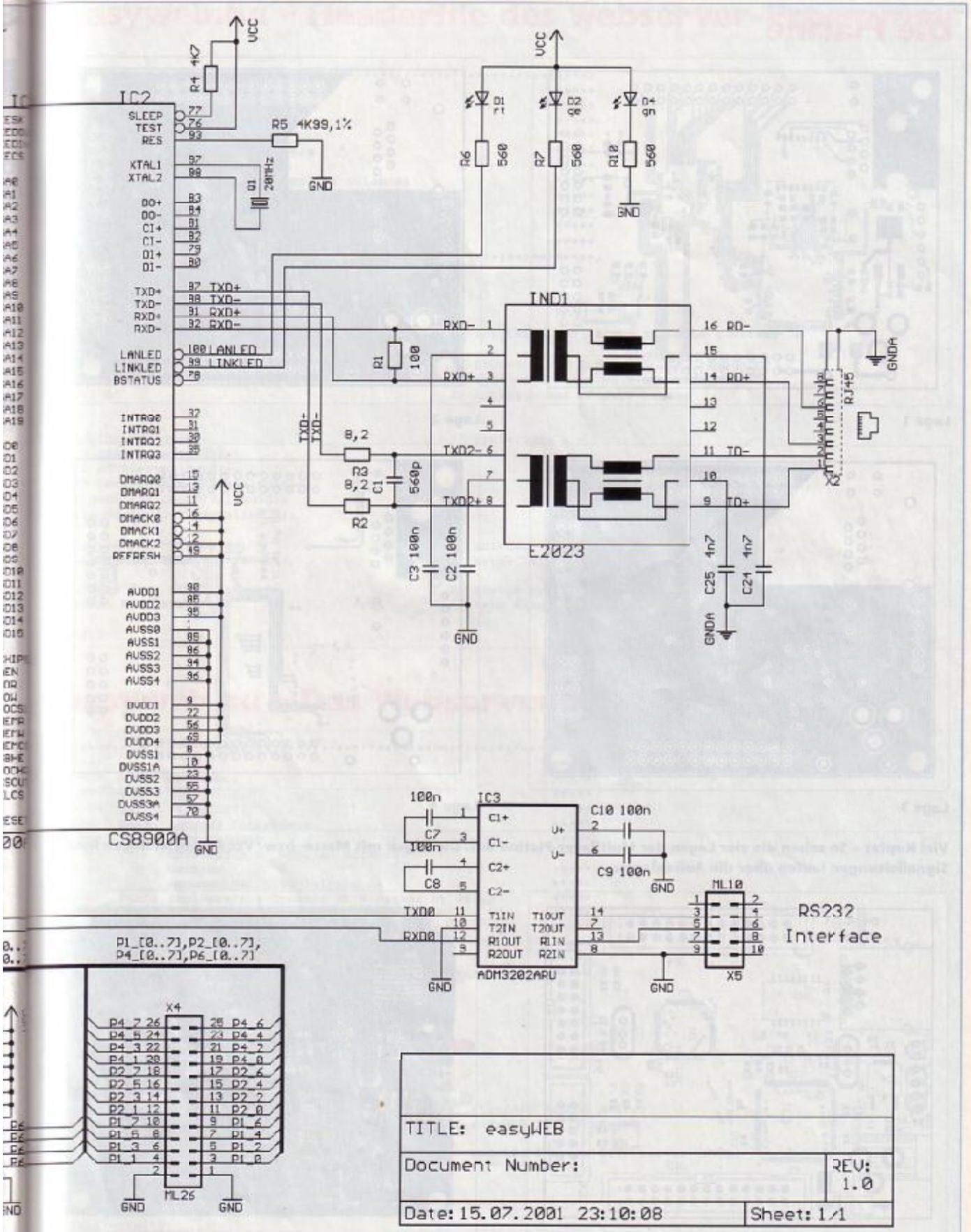
Dieses praxisgerechte Extraheft der DESIGN&ELEKTRONIK zum Thema »Embedded Internet« wäre nicht vollständig ohne die detaillierten Unterlagen, die jeder Entwickler zum Nachvollziehen der technischen Details und vor allem für die Anpassung des Projekts auf eigene Entwicklungen braucht. Daher haben wir für Sie auf den folgenden Seiten die Schalt-, Layout- und Bestückungspläne der Multilayer-Platine sowie den Quellcode zu den wichtigsten, im Text angesprochenen Software-Routinen zusammengestellt. Also: Nichts wie an den PC gesetzt und alles fein abgetippt! Zu mühsam? Auf alle Unterlagen können Sie natürlich auch über das Internet zugreifen: Unter der Adresse www.elektroniknet.de/extraheft finden sie alle Quellcodes in C. Außerdem verfügbar sind Link-File und Work-File zum Projekt, Schaltplan und Stückliste sowie sämtliche Dateien zur Multilayer-Platine im »Eagle«-Format. Viel Spaß beim Download!

Bauelement	Wert	Package	Bezugsquelle	Best.-Nr.
C2, C3, C6, C7, C8, C9, C10, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23, C26	100 nF	0805	Reichelt	X7R-G0805 100N
C1	560 pF	0805	RS-Components	211-3316
C24, C25	4,7 nF / 2 kV	RM 5 mm	Farnell	498-403
C4, C5	15 pF	0805	Reichelt	NPO-C0805 15P
C11	10 µF / 35 V		Reichelt	SMD TAN.10/35
R6, R7, R10	560 Ω	0805	Reichelt	SMD-0805 560
R5	4,99 kΩ	0805	RS-Components	215-3162
R1	100 Ω	0805	Reichelt	SMD-0805 100
R2, R3	8,2 Ω	0805	Reichelt	SMD-0805 8,20
R4	4,7 kΩ	0805	Reichelt	SMD-0805 4,70K
R8, R9	100 kΩ	0805	Reichelt	SMD-0805 100K
Q1	20 MHz	HC-49	Reichelt	20-HC49U-S
Q2	8 MHz	HC-49	Reichelt	8-HC49U-S
D1	LED rot 3 mm / 2 mA		Reichelt	LED 3MM 2M-ROT
D2	LED gelb 3 mm / 2 mA		Reichelt	LED 3MM 2M-GELB
D3	P6KE27A	DO-15	RS-Components	354-5455
D4	LED grün 3 mm / 2 mA		Reichelt	LED 3MM 2M-GRÜN
IC1	MSP430F149	QFP-64		
IC2	CS8900A-IQ3	TQFP-100	Atlantik Elektronik	
IC3	ADM3202ARU	TSSOP-16	RS-Components	357-0906
IC4	LD1086DT33	DPAK	RS-Components	355-4768
IND1	Übertrager E2023	SO-16L	Rutronik	IND3880
X1	Buchse DC25		Farnell	224-960
X2	Buchse RJ45		Farnell	473-236
X3, X4	Stiftleiste 26-polig		Reichelt	Stiftl. 2x13G
X5	Messerleiste 10-polig		Reichelt	WSL 10G
X6	Messerleiste 14-polig		Reichelt	WSL 14G

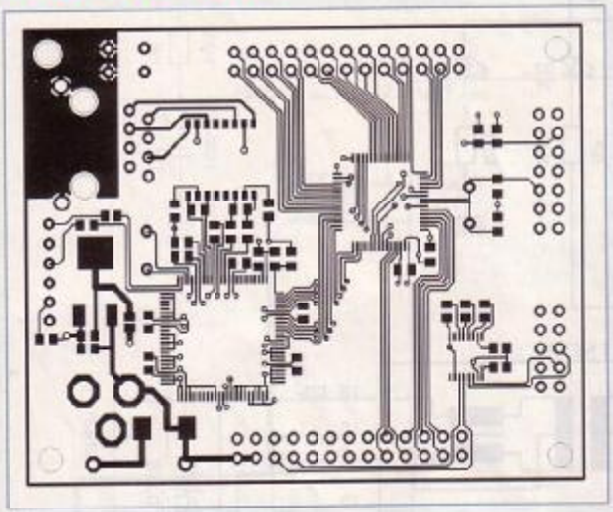
Das ist alles drauf: Die Stückliste der Baugruppe

Der Schaltplan

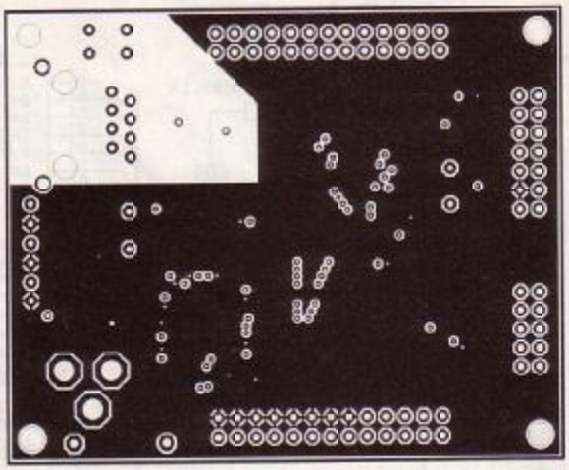



IC2
CS8900A
VCC
R4 1K7
R5 4K99,1%
GND
SLEEP
TEST
RES
XTAL1
XTAL2
Q1 20MHz
TXD+
TXD-
RXD+
RXD-
LANLED
LINKLED
BSTATUS
INTRQ0
INTRQ1
INTRQ2
INTRQ3
DMARQ0
DMARQ1
DMARQ2
DMACK0
DMACK1
DMACK2
RFRESH
AUDD1
AUDD2
AUDD3
AUSS0
AUSS1
AUSS2
AUSS3
AUSS4
DVDD1
DVDD2
DVDD3
DVDD4
DVSS1
DVSS1A
DVSS2
DVSS3
DVSS3A
DVSS4
D1
D2
D3
R6 560
R7 560
R10 560
IND1
E2023
R1 100
R2 8,2
R3 8,2
C1 560p
C2 100n
C3 100n
C25 4n7
C21 4n7
RJ45
GNDA
IC3
ADM3202ARU
C7 100n
C8 100n
C10 100n
C9 100n
C1+
C1-
C2+
C2-
V+
V-
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
TXD0
RXD0
ML10
X5
RS232
Interface
P1_[0..7],P2_[0..7],
P4_[0..7],P6_[0..7]
X4
ML25
TITLE: easyWEB
Document Number:
REV: 1.0
Date: 15.07.2001 23:10:08
Sheet: 1/1

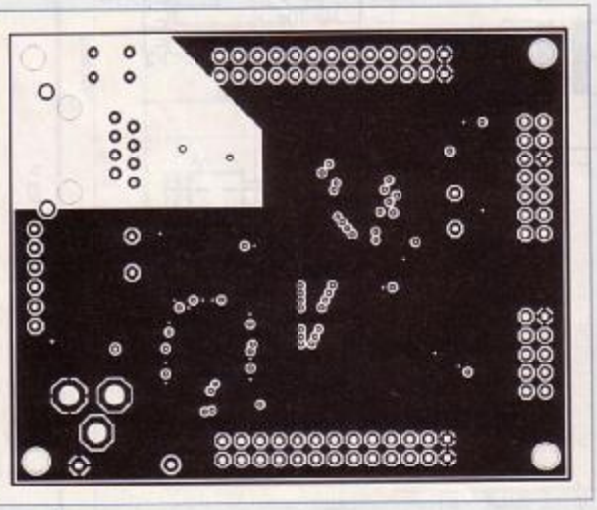
Die Platine



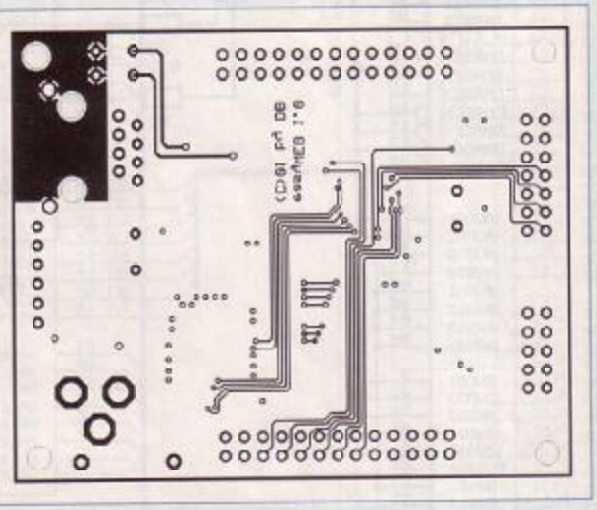
Lage 1



Lage 2

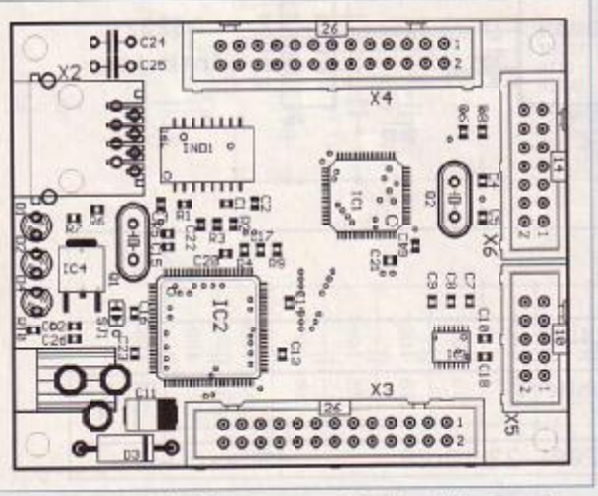


Lage 3

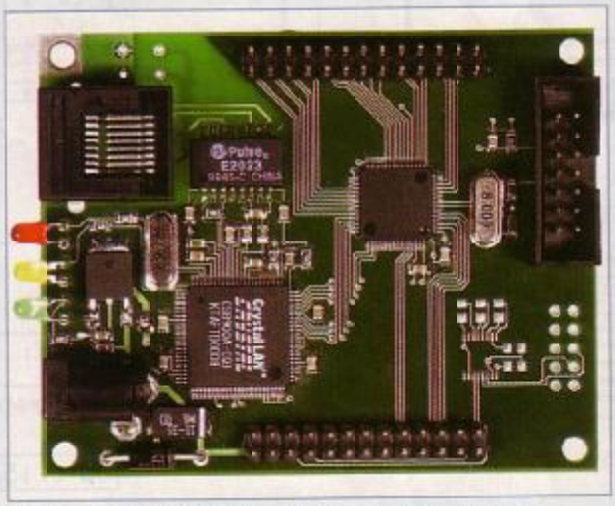


Lage 4

Viel Kupfer – So sehen die vier Lagen der Multilayer-Platine aus. Die Lagen mit Masse- bzw. Vcc-Potential liegen innen, Signalleitungen laufen über die Außenlagen.



Zeigt, wo die Teile hin müssen – Der Bestückungsplan



Und so sieht die fertig bestückte Baugruppe aus

»easyweb.h« – Headerfile des Webserver-Programms

```
/*********************************************************************
 *****                                                          *****
 *****  Name: easyweb.h                                         *****
 *****  Ver.: 1.0                                               *****
 *****  Date: 07/05/2001                                        *****
 *****  Auth: Andreas Dannenberg                                *****
 *****        HTWK Leipzig                                      *****
 *****        university of applied sciences                    *****
 *****        Germany                                           *****
 *****        adannenb@et.htwk-leipzig.de                       *****
 *****  Func: header-file for easyweb.c                         *****
 *****                                                          *****
 *********************************************************************/

#ifndef __EASYWEB_H
#define __EASYWEB_H

const unsigned char GetResponse[] =              // 1st thing our server sends to a client
{
  "HTTP/1.0 200 OK\r\n"                          // protocol ver 1.0, code 200, reason OK
  "Content-Type: text/html\r\n"                  // type of data we want to send
  "\r\n"                                         // indicate end of www-header
};

void InitOsc(void);
void InitPorts(void);
void HTTPServer(void);
void InsertDynamicValues(voi
unsigned int GetAD7Val(void)
unsigned int GetTempVal(void

unsigned char *PWebSide;
unsigned int HTTPBytesToSend

unsigned char HTTPStatus;
#define HTTP_SEND_PAGE

#endif
```

Die Software gibt es also Dabei !

»easyweb.c« -

```
/*****************************
 *****
 *****  Name: easyweb.c
 *****  Ver.: 1.0
 *****  Date: 07/05/2001
 *****  Auth: Andreas Dannen
 *****        HTWK Leipzig
 *****        university of applied sciences                    *****
 *****        Germany                                           *****
 *****        adannenb@et.htwk-leipzig.de                       *****
 *****  Func: implements a dynamic HTTP-server by using         *****
 *****        the easyWEB-API                                   *****
 *****  Rem.: In IAR-C, use  linker option                      *****
 *****        "-e_medium_write=_formatted_write"                *****
 *****                                                          *****
 *********************************************************************/

#include "msp430x14x.h"
#include "stdlib.h"
#include "stdio.h"
#include "string.h"

#include "easyweb.h"

#include "cs8900.c"                              // ethernet packet driver
#include "tcpip.c"                               // easyWEB TCP/IP stack

#include "webside.c"                             // webside for our HTTP server (HTML)

void main(void)
{
  InitOsc();
  InitPorts();
  TCPLowLevelInit();
```

»webconverter.cpp« – Konvertierung von HTML nach C

```
/*************************************************************************
 *****                                                               *****
 *****  Name: webconverter.cpp                                       *****
 *****  Ver.: 1.0                                                    *****
 *****  Date: 07/03/2001                                             *****
 *****  Auth: Andreas Dannenberg                                     *****
 *****        HTWK Leipzig                                           *****
 *****        university of applied sciences                         *****
 *****        Germany                                                *****
 *****        adannenb@et.htwk-leipzig.de                            *****
 *****  Func: converts HTML-code to a C-constant                     *****
 *****                                                               *****
 *************************************************************************/

#include <stdio.h>

#pragma hdrstop
#pragma argsused
int main(int argc, char* argv[])
{
  FILE *in, *out;
  char InChar;

  if (argc < 2)
  {
    fprintf(stdout, "Usage: WebConverter <infile> <outfile>\r\n");
    return 1;
  }

  if ((in = fopen(argv[1], "rb")) == NULL)
  {
    fprintf(stderr, "Cannot open input file.\n");
    return 1;
  }

  if ((out = fopen(argv[2], "wt")) == NULL)
  {
    fprintf(stderr, "Cannot open output file.\n");
    return 1;
  }

  fprintf(out, "const unsigned char WebSide[] = {\r\n\"");

  while (!feof(in))
  {
    InChar = fgetc(in);
    switch (InChar)
    {
      case 0x22 : fputc('\\', out);
                  fputc('"', out);
                  break;
      case 0x0D : fputc('\\', out);
                  fputc('r', out);
                  fputc('\\', out);
                  fputc('n', out);
                  fputc('"', out);
                  fprintf(out, "\r\n");
                  fputc('"', out);
                  break;
      case 0x0A : break;
      case 0xFF : break;
      default   : fputc(InChar, out);
    }
  }

  fputc('\\', out);
  fputc('r', out);
  fputc('\\', out);
  fputc('n', out);

  fputc('"', out);
  fprintf(out, "};\r\n");

  fclose(in);
  fclose(out);

  return 0;
}
```

Literatur- und Quellenverzeichnis

[NERDS] Segaller, S.: *NERDS 2.0.1 - A Brief History of the Internet.* TV Books, 1999

[TCP/IP] Washburn, K.; Evans, J.: *TCP/IP - Aufbau und Betrieb eines TCP/IP-Netzes.* Addison-Wesley, 1993

[CNetw] Tanenbaum, A.: *Computer Networks.* Prentice Hall, 1996

[Crystal] Cirrus Logic: *CS8900A Product Data Sheet.* Cirrus Logic Inc., 1999

[IEEE802.3] IEEE: *IEEE 802.3 CSMA/CD Access Method.* IEEE, 2000

[RFC-791] Postel, J.: *Internet Protocol (IP).* RFC*-791, September 1981

[RFC-792] Postel, J.: *Internet Control Message Protocol (ICMP).* RFC*-792, September 1981

[RFC-793] Postel, J.: *Transmission Control Protocol.* RFC*-793, September 1981

[RFC-826] Plummer, D.: *An Ethernet Address Resolution Protocol.* RFC*-826, November 1982

[RFC-1071] Braden, R.; Borman, D.; Partidge, C.: *Computing the Internet Checksum.* RFC*-1071, September 1988

[RFC-1122] Braden, R.: *Requirements for Internet Hosts.* RFC*-1122, Oktober 1989

[RFC-1700] Reynolds, J.; Postel, J.: *Assigned Numbers.* RFC*-1700, Oktober 1994

[MSP430DS] Texas Instruments: *MSP430x13x, MSP430x14x Mixed Signal Microcontroller Datasheet.* Texas Instruments Inc., Juli 2000

[CS-AN83] Cirrus Logic: *Application Note 83.* Cirrus Logic Inc., April 1999

[CS-AN181] Cirrus Logic: *Application Note 181.* Cirrus Logic Inc., Januar 2000

[Pulse] Pulse: *LAN Isolation Transformer Catalog.* Pulse Engineering Inc., Dezember 2000

[ADM3202] Analog Devices: *Low Power Line Drivers/ Receivers ADM3202/ADM3222/ADM1385 Data Sheet.* Analog Devices Inc., 2000

[P6KE_DS] Fairchild Semiconductor: *600 Watt Transient Voltage Suppressors P6KExx.* Fairchild Semiconductor Corporation, 1998

[LD1086] ST Microelectronics: *Positive Voltage Regulator LD1086 Series Data Sheet.* ST Microelectronics, 2000

*Request for Comments, Sammlung von Dokumenten über Internetrelevante Themen. Zu beziehen beispielsweise über http://www.faqs.org/rfcs/index.html