

LumWeb

Universelle Maschinensteuerung mit Webinterface

HTL Paul-Hahn

Informationstechnologie

Diplomarbeit von:

Martin Anzinger

Florian Hahn

Betreut von:

Prof. Mag. Herbert Jachs

Linz, 26. Mai 2010

Version 1.1



Kurzfassung

Heutzutage ist es Standard, große Maschinen und Anlagen (zum Beispiel Heizungen) zusätzlich zu den üblichen Bedienkonsole auch über einen Webbrower bedienen zu können. Realisiert wird dies oft mit einem externen Server (PC), der mit der Maschine verbunden ist. Dadurch ergeben sich aber Probleme. Zum Beispiel muss die Bedienoberfläche für die Bedienkonsole und für das Webinterface separat entwickelt werden, es werden also zwei Entwickler benötigt, wodurch sich auch höhere Entwicklungskosten ergeben. Zweitens sind die Kosten für einen herkömmlichen PC um einiges höher als ein Embedded System, bei der Anschaffung, als auch im laufenden Betrieb. Ein Embedded System ist robuster als ein herkömmlicher PC, da es in der Regel keine beweglichen Komponenten wie Lüfter oder Festplatten benötigt.

Das Ziel des Projektes LumWeb ist die Vereinfachung der Userinterface-Erstellung der Maschinensteuerungs-Interfaces für Heizungen und andere Maschinen um dadurch die oben genannten Probleme zu beseitigen.

Erreicht wird dieses Ziel durch das Zusammenführen von Embedded-Webserver und Embedded-Webclient auf einem Microcontroller-System.

Das Userinterface wird als HTML Seite mit Speziellen Tags (sogenannte SSI-Tags) definiert. Die Bedienseiten sind auf einem Serversystem gespeichert. Das Serversystem ist zuständig für die Kommunikation mit der Maschine und mit dem Benutzer. Die Webseiten werden über HTTP an die Clients verteilt. Auf PC-Systemen übernimmt der Browser die Darstellung, auf Mikrocontroller-Systemen über nimmt das der Emedded-Webclient.

Der Webserver fügt beim Auswerten der Tags zudem spezielle HTML Kommentare ein, die der Emedded-Webclient zur Darstellung wiederum auswertet. Der Client kann nur solche Tags darstellen, für die Darstellungsmethoden implementiert sind.

Abstract

LumWeb is a simple solution for building universal user interfaces.

The main point is that the interface information is broadcast over the HTTP protocol as a web page. The user interface can be accessed by any web browser.

The graphic web-client on the micro controller board is an embedded web-browser with a graphical touchscreen interface.

This solution increases the productivity because only one designer is needed. Settings can be accessed from different user interfaces.

The second advantage of LumWeb is the embedded tag-library. For designing interfaces, the tags from the tag-library can be used.

Another feature is, that the LumWeb project can be used as stand-alone appliance. This means that the server-system and the client-system are on one micro-controller board.

But there are many different operating modes, for example a separate server-system and one or more client-systems. If the server-system is connected to the intranet or the public web, the server-system is accessible by normal web-browsers.

Ehrenwörtliche Erklärung

Die Diplomarbeit „LumWeb“ wurde selbstständig von Anzinger Martin und Hahn Florian erstellt. Alle verwendeten Quellen beziehungsweise Hilfsmittel sind in der Dokumentation vermerkt, und es wurden keine unerlaubten Hilfsmittel verwendet.

Datum

Anzinger Martin

Hahn Florian

Präambel

Das Projekt wurde als Ingenieursprojekt gemäß entsprechender Verordnung des Bundesministeriums für Unterricht, Kunst und Kultur durchgeführt. Betreut wurde das Projekt durch Prof. Mag. Herbert Jachs in Zusammenarbeit mit der Firma Hainzl.

Vorwort

LumWeb ist aus der Idee entstanden, ein Projekt mit moderner 32-Bit Mikrocontroller-Architektur in Kombination mit einem Real-Time Betriebssystem zu realisieren. Ein besonderer Dank gebührt unserem Betreuungslehrer Prof. Mag. Herbert Jachs, der uns dieses Projekt mit der Firma Hainzl vorgeschlagen und den ersten Kontakt hergestellt hat, ohne ihn wäre die Diplomarbeit in der jetzigen Form nicht zu Stande gekommen. Er ist uns immer beratend zur Seite gestanden und hat uns beratend unterstützt. Weiters möchten wir uns auch bei der Firma Hainzl und im besonderen bei unserer Kontaktperson Gerhard Scheiblhofer bedanken, die es ermöglicht haben, die Diplomarbeit umzusetzen und uns dabei mit zwei LuminaryMicro DK-LM3S9B96 Boards eine realistische Hardware-Umgebung zu Entwicklungszwecken zur Verfügung gestellt haben.

Das grundlegende Konzept der Diplomarbeit wurde von Martin Anzinger und Florian Hahn gemeinsam entwickelt und entworfen. Martin Anzinger war speziell für die Entwicklung des Clients inklusive Touch-Interface, Embedded HTTP Client und der Tag-Library zuständig. Er hat auch die Entwicklungsumgebung Eclipse für unserer Zwecke zusammengestellt und konfiguriert. Florian Hahn war für die Entwicklung des Servers zuständig, inklusive Erweiterungen des lwIP-Webservers um SSI-Tags, der SSI Parameterunterstützung und der CGI-Funktion, Implementierung des Com-Tasks und der Task-Kommunikation, weiters hat er die Code-Dokumentationsumgebung mit Doxygen erstellt.

Unerwartete Anforderungen wurden gemeinsam besprochen und gelöst, zum Beispiel die Portierung von FreeRTOS und lwIP von dem schulintern verwendeten Luminay Micro-Board auf das Board der Firma Hainzl.

Inhaltsverzeichnis

1 Projektumgebung.....	10
1.1 Entwicklungsumgebung.....	10
1.1.1 Eclipse.....	10
1.1.2 GCC.....	10
1.1.3 OpenOCD.....	11
1.1.4 DoxyGen.....	11
1.2 Hardwareumgebung.....	12
1.2.1 Luminary Micro DK-LM3S9B96.....	12
1.2.2 ARM Cortex-M3.....	15
1.3 Softwareumgebung.....	16
1.3.1 FreeRTOS.....	16
1.3.2 lwIP.....	20
1.3.3 Stellaris® Graphic Library.....	24
1.3.4 Stellaris® Peripheral Driver Library.....	25
1.3.5 FatFS.....	25
1.3.6 Make.....	26
2 Konfigurationen.....	30
2.1 FreeRTOS-Konfiguration.....	30
2.1.1 Tasks.....	30
2.1.2 Queues.....	31
2.2 lwIP-Konfiguration.....	31
2.3 LumWeb-Konfiguration.....	32
2.3.1 Allgemeine Einstellungen.....	32
2.3.2 Debuginformationen.....	33
2.3.3 Systemeinstellungen.....	34
3 LumWeb.....	37
3.1 Konzept.....	37
3.1.1 Betriebsarten.....	38
3.1.2 Tasks.....	39
3.1.3 Com-Task.....	39
3.1.4 lwIP-Task.....	39

3.1.5 Graphic-Task.....	39
3.1.6 Real-Time-Clock-Task.....	40
3.1.7 Intertask-Kommunikation	40
3.2 Tag-Library.....	41
3.2.1 Aufbau.....	41
3.2.2 Funktionen in der Taglib.....	41
3.2.3 Erstellen eines Tags.....	43
3.3 Webserver.....	45
3.3.1 SSI.....	45
3.3.2 CGI.....	59
3.3.3 Ablauf.....	61
3.3.4 Erstellen einer Bedienseite.....	63
3.4 Webclient.....	65
3.4.1 Interface.....	65
3.5 Com-Task.....	67
3.6 Debugging und Logging.....	70
3.6.1 UARTPrintf.....	70
3.6.2 AppendToLog.....	70
4 Abbildungsverzeichnis.....	72
24 Anhang.....	74

Versionstabellen

Version	Datum	Änderung	Autor
0.1	23.03.2010	Erstellen der SW Doku	Anzinger Hahn
0.2	06.04.2010	Konzept erweitert	Hahn
0.3	09.04.2010	FreeRTOS, lwIP und Com-Task dokumentiert	Hahn
0.4	16.04.2010	Ablaufpläne, Übersicht	Hahn
0.5	24.04.2010	Graphic dokumentiert und Englischen Abstract verfasst	Anzinger
0.6	05.05.2010	Dokumentation erweitert	Hahn, Anzinger
0.7	06.05.2010	SSI Parameter, Konfigurationsdatei dokumentiert	Hahn
0.8	07.05.2010	Vorwort, Korrektur-Lesung	Hahn
0.8.1	09.05.2010	SSI-Tags neu dokumentiert, Abbildungsverzeichnis	Hahn, Anzinger
0.8.2	10.05.2010	Stellaris Graphic Library Dokumentation	Anzinger
0.9	11.05.2010	Korrektur-Lesung eingearbeitet	Hahn
0.9.1	12.05.2010	Touchscreenbilder eingefügt	Anzinger
0.9.2	13.05.2010	Kleine Korrekturen	Anzinger
0.9.3	13.05.2010	Automatische Übernahme der Versionsnummer	Anzinger
0.9.4	13.05.2010	Korrekturen, Debugging, Logging und FatFS dokumentiert, Verweise hinzugefügt	Hahn
0.9.5	15.05.2010	Makfile dokumentiert, FatFS Beispiele eingefügt	Hahn
1.0	16.05.2010	Quellenangabe, Korrekturen	Hahn
1.1	26.05.2010	Verbesserte Version für Bibliothek	Anzinger

1 Projektumgebung

1.1 Entwicklungsumgebung

1.1.1 Eclipse

Eclipse¹ ist ein quelloffenes Programmierwerkzeug zur Entwicklung von Software verschiedenster Art. Ursprünglich wurde Eclipse als integrierte Entwicklungsumgebung für die Programmiersprache Java genutzt, aber mittlerweile wird Eclipse aufgrund seiner Erweiterbarkeit auch für viele andere Entwicklungsaufgaben eingesetzt. Für Eclipse gibt es eine Vielzahl von Erweiterungen sowohl quelloffen als auch von kommerziellen Anbietern.

Bei der Entwicklung von LumWeb wurde Eclipse 3.5² mit der Erweiterung CDT für C/C++-Programmierung verwendet.

1.1.2 GCC

GCC ist der Name der Compiler-Suite des GNU-Projekts.

GCC wird von einer Reihe von Systemen als Standardcompiler benutzt, darunter viele Linux-Distributionen, BSD, Mac OS X, NextStep, und BeOS bzw. ZETA. GCC wurde auf mehr Systeme und Rechnerarchitekturen portiert als jeder andere Compiler. GCC bietet sich besonders für Betriebssysteme an, die auf verschiedenen Hardware-Plattformen laufen sollen. Der GCC lässt sich auch als Cross-Compiler³ verwenden.

Insgesamt unterstützt GCC mehr als 60 Plattformen.

Bei der Entwicklung von LumWeb wurde die *GNU Toolchain for ARM Processors* von CodeSourcery in Version 2009q1 verwendet. Dieses Paket enthält alle benötigten Programme und Bibliotheken, um Programme für ARM CPUs zu kompilieren, zu linken und wurde speziell für ARM CPUs optimiert.

1 Wikipedia - Eclipse (IDE) – April 2010, http://de.wikipedia.org/w/index.php?title=Eclipse_%28IDE%29&oldid=72909998

2 Eclipse 3.5 Download mit CDT - http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR2/eclipse-cpp-galileo-SR2-linux-gtk-x86_64.tar.gz

3 Compiler, der auf einem bestimmten System (auch Hostplattform genannt) läuft, aber Kompilete (Objektdateien oder ausführbare Programme) für andere Systeme erzeugt.

1.1.3 OpenOCD

Der OpenOCD⁴ JTAG Server ist eine Open Source On-Chip Debugginglösung für ARM basierte Zielsysteme. Unterstützt werden unter anderem: ARM7, ARM9 oder Cortex-M3. OpenOCD ermöglicht Debugging mit dem Standard GNU Debugger GDB für die ARM Architektur. Zusätzlich wird die Programmierung des internen und externen Flash Speichers unterstützt. Bei LumWeb wurde OpenOCD zum Flashen der Hardware verwendet.

1.1.4 DoxyGen

DoxyGen ist ein Werkzeug zum automatischen Erstellen von Softwaredokumentation. Dazu wird der Quelltext nach Definitionen von Funktionen, Strukturen, Variablen und speziellen Kommentaren durchsucht, diese können als HTML oder LaTex dargestellt werden.

⁴ Beschreibung OpenOCD – April 2010, <http://www.amontec.com/openocd.shtml>

1.2 Hardwareumgebung

1.2.1 Luminary Micro DK-LM3S9B96

Als Entwicklungssystem wurde ein LuminaryMicro DK-LM3S9B96⁵ verwendet. Im Zuge der Diplomarbeit wurde zuerst das an der Schule verwendete Luminary-Board verwendet, welches dann aber durch das DK-LM3S9B96 ersetzt wurde. Dieses wurde uns von der Firma Hainzl zur Verfügung gestellt und stellt mit einem Touch-Screen ein praxisnäheres Entwicklungssystem dar. Luminary Micro ist eine US-amerikanischer Halbleiter-Firma, die Microkontroller auf Basis der Cortex M3 Architektur von ARM herstellt.

Wichtige Eckdaten:⁶

- 3.5“ Farb-LCD
 - Auflösung: 320 x 240
 - Resistiver Touch-Screen
- 80 MHz LM3S9B9 uC
 - 256kB Flash
 - 96kB Sram
 - integrierter Ethernet Controller
 - Card Reader
 - ARM Cortex M3 CPU

5 Luminary Micro DK-lm3s9b96 Produktbeschreibung - <http://www.luminarymicro.com/products/dk-lm3s9b96.html>

6 Luminary Micro – April 2010, <http://www.luminarymicro.com/products/dk-lm3s9b96.html>

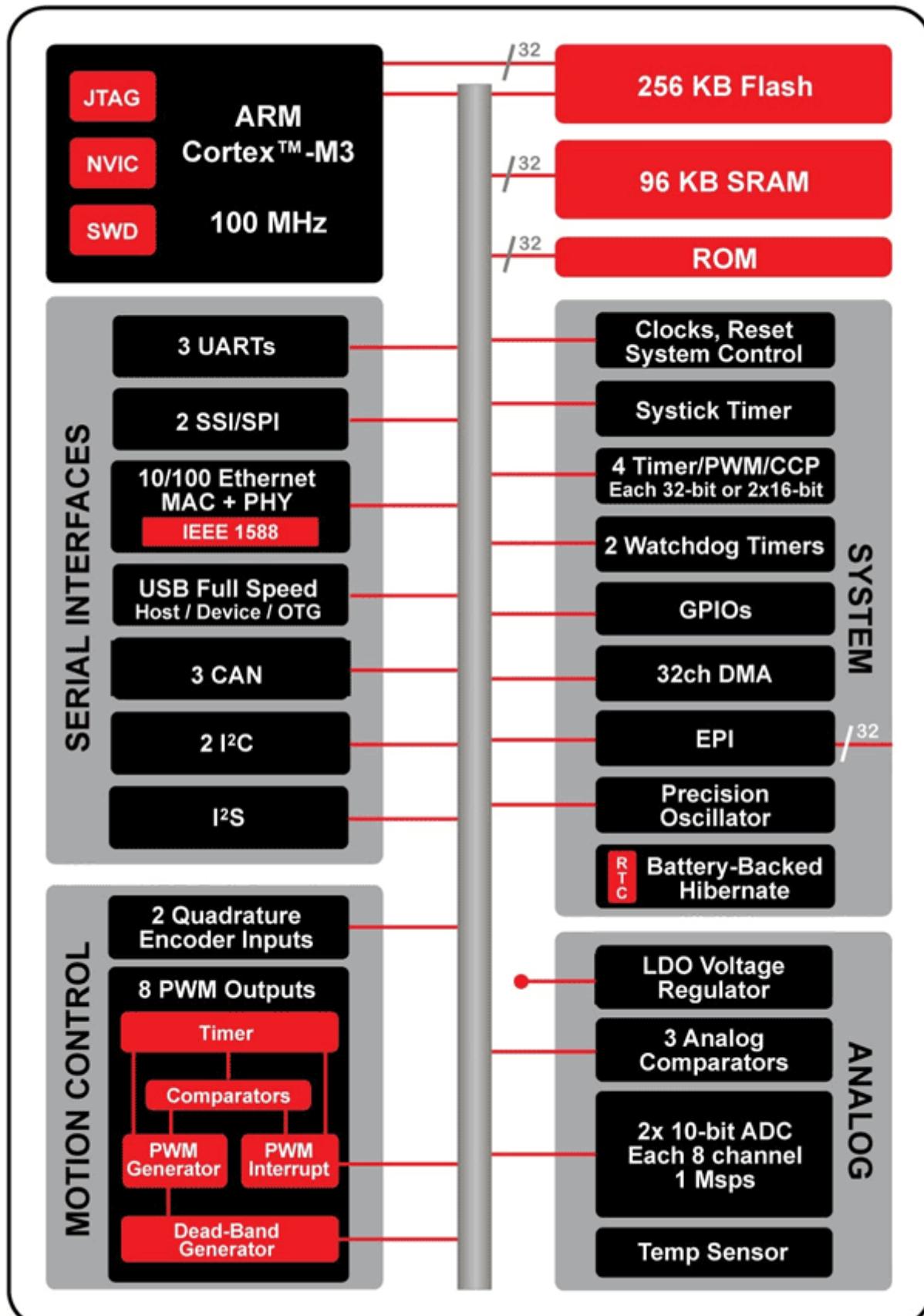


Abbildung 1.1: Blockschaltbild Stellaris Micro Controller (Quelle: Luminary Micro
<http://www.luminarmicro.com/images/stories/stellarisblockdiagram.gif>)

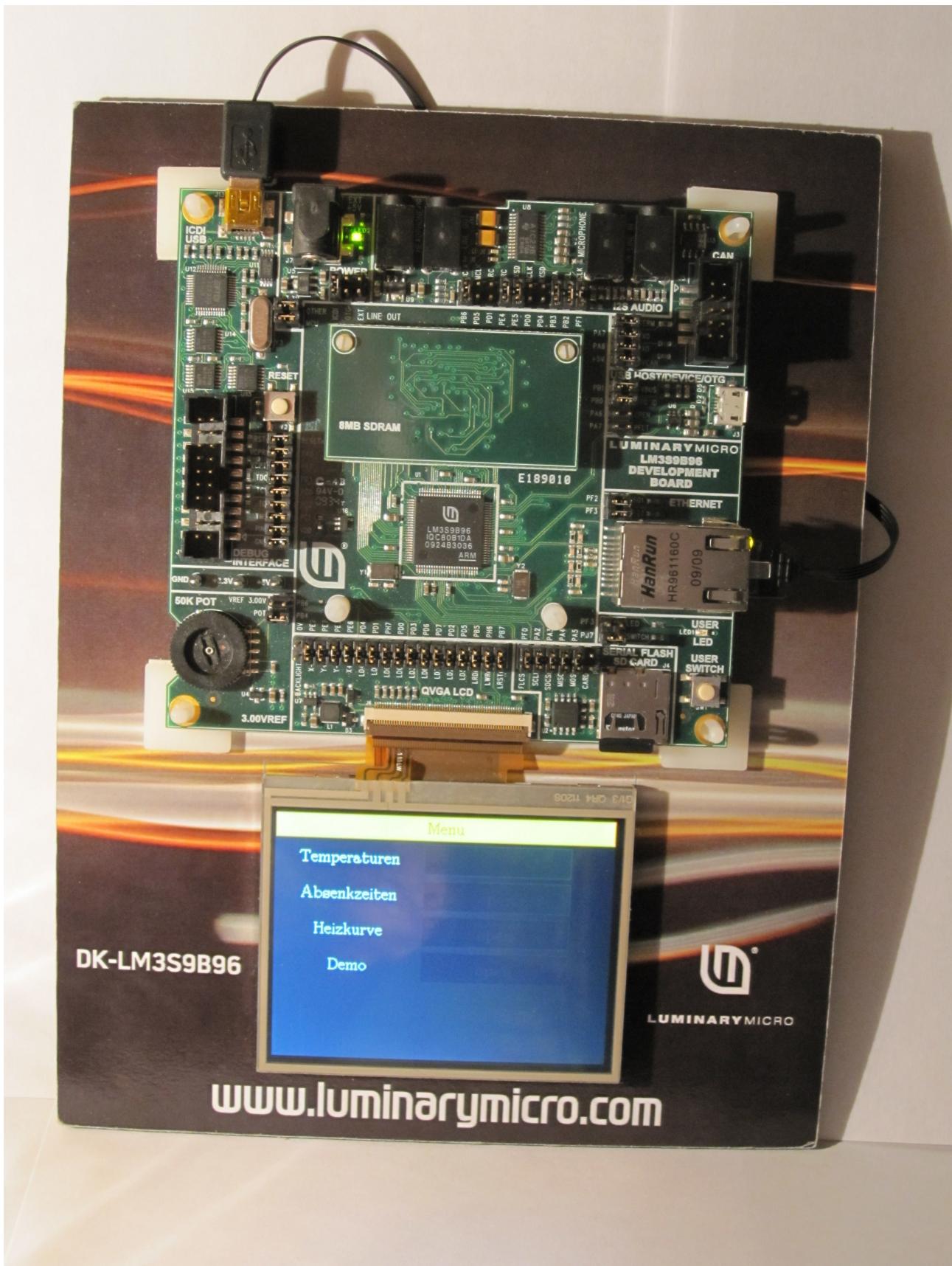


Abbildung 1.2: Luminary Developmentkit (LM3S9B96) (Quelle: Anzinger, Hahn)

1.2.2 ARM Cortex-M3

Luminary Micro setzt für ihre Microcontroller-Systeme auf die ARM Cortex M3 Architektur.

Der **Cortex-M3**⁷ ist eine Architektur (ARMv7-M) für Mikroprozessoren von ARM. Diese Architektur kann als Nachfolger für die ARM7 im Bereich der Mikrocontroller betrachtet werden.

Der Cortex-M3 ist eine völlig neu entwickelte Architektur von ARM mit dem Ziel, einen sehr leistungsfähigen, aber vom Programmiermodell her weniger komplexen Prozessor zu entwerfen, der im Bereich der jetzigen und zukünftigen Mikrocontroller des (klassischen) 8- und 16-Bit-Bereichs eingesetzt werden kann. Wie alle ARM-Architekturen hat auch der M3 (intern) eine 32-Bit-Architektur, arbeitet aber ausschließlich mit dem neuen Thumb2-Befehlssatz.

Herzstück des Cortex-M3-Prozessors ist der Cortex-M3-Kern mit dreistufiger Pipeline, basierend auf der Harvard-Architektur. ARM-Prozessoren sind für beide Architekturen, Von-Neumann (wo ein Daten- und Adressbus zum Laden von Befehlen und Daten verwendet wird), und Harvard verfügbar. Die Harvard-Architektur zeichnet sich dadurch aus, dass zwei getrennte Bussysteme (und zwei getrennte Speicher) zum Laden von Daten und Befehlen existieren, d.h. der Prozessor kann gleichzeitig sowohl Daten als auch Befehle lesen (bzw. Daten in den Speicher zurückschreiben). Nach außen hin (Programmiermodell) ist der Cortex-M3 allerdings ein Von-Neumann-Modell, das bedeutet, dass sein ganzer (geteilter) Adressraum linear programmiert werden kann. Das erspart aufwendige Zugriffe auf den Programmspeicher, wenn dort Konstanten abgelegt sind.

Bei einem 32-Bit-Prozessor ist der ansprechbare Speicherbereich mit 4 Gigabyte für einen Mikrocontroller überdimensional groß. Daher existieren genügend Adressen, um beide Speicher im gemeinsamen Adressraum anzusprechen. Für den Programmierer des Systems ist dies ein überraschender Vorteil, da Daten, welche im Flash (Programmspeicher) abgelegt werden (Konstanten, Strings, etc.) direkt linear adressiert werden können und nicht erst mit umständlichen Befehlen geladen werden müssen.

⁷ Wikipedia - ARM Cortex-M3, Mai 2010, http://de.wikipedia.org/w/index.php?title=ARM_Cortex-M3&oldid=73886321

1.3 Softwareumgebung

1.3.1 FreeRTOS

Als Betriebssystem wurde das in C geschriebene Open Source Realtime Operating System FreeRTOS⁸ in Version 6.0 verwendet, da es lizenzfrei verwendet werden kann und sehr stabil ist. Von uns wurde es in Kombination mit einer ARM Cortex M3 CPU verwendet, FreeRTOS ist aber auch für andere Hardwareplattformen verfügbar.

Im Gegensatz zu kommerziellen Systemen, wie embOS und CMX-RTX, fehlen FreeRTOS allerdings wichtige Synchronisationsmechanismen wie "Event Flags". Mutexe⁹ sind in neueren Versionen vorhanden.

1.3.1.1 Funktionsweise¹⁰

Um Netzwerkanfragen, die grafische Darstellung am Display und die Ansteuerung der Maschine quasi-parallel realisieren zu können wird eine Multitaskingumgebung benötigt. FreeRTOS bietet diese Umgebung in Kombination mit Queues zur Interprocesskommunikation.

Um eine gute Wartbarkeit zu gewährleisten, wird FreeRTOS weitestgehend in C entwickelt, lediglich wenige Funktionen sind in Assembler realisiert.

1.3.1.2 Multitasking

Der Begriff **Multitasking** bezeichnet die Fähigkeit eines Betriebssystems, mehrere Aufgaben (Tasks) nebenläufig auszuführen. Dabei werden die verschiedenen Prozesse in so kurzen Abständen immer abwechselnd aktiviert, dass der Eindruck der Gleichzeitigkeit entsteht.

Die Nutzung von Multitasking bietet folgende Vorteile:¹¹

- Durch Multitasking und Intertask-Kommunikation können komplexe Anwendungen in kleiner, besser wartbare Teile zerlegt werden
- Die Zerlegung von Anwendungen vereinfacht das Testen, die Wiederverwendbarkeit und die Arbeitsteilung in Teams

8 FreeRTOS Homepage <http://www.freertos.org/>

9 Mutexe verhindern, dass nebenläufige Prozesse gleichzeitig gemeinsam genutzte Datenstrukturen unkoordiniert verändern

10 Wikipedia – FreeRTOS, April 2010, <http://de.wikipedia.org/w/index.php?title=FreeRTOS&oldid=69842333>

11 RTOS Fundamentals – Multitasking, April 2010, <http://www.freertos.org/implementation/index.html>

- Komplexe Timing-Funktionen können von der Anwendung ins Betriebssystem ausgelagert werden

FreeRTOS bietet 2 Möglichkeiten, um Multitasking zu verwenden, *Tasks* und *Co-routines*¹².

Eine Real Time Anwendung, die FreeRTOS verwendet, kann als Sammlung von separaten Tasks realisiert werden. Jeder Task wird unabhängig von anderen Tasks in seinem eigenen Kontext¹³ ausgeführt. Es kann zu jedem Zeitpunkt nur ein Task ausgeführt werden. Die Zeiteinteilung wird vom Scheduler¹⁴ gemanaged. Der Scheduler ist konfigurierbar, so dass präemptiver und kooperativer Betrieb möglich ist.

Task-States

Jeder Task hat immer nur einen der folgenden „States“:

- *Running*
wenn ein Task ausgeführt wird, bezeichnet man ihn als *Running*
- *Ready*
Tasks die ausgeführt werden könnten, aber im Moment nicht ausgeführt werden, da ein anderer Task mit gleicher oder höherer Priorität gerade ausgeführt wird, bezeichnet man als *Ready*
- *Blocked*
Tasks im *Blocked* State warten auf ein externes oder zeitliches Event
- *Suspended*
Tasks im *Suspended* State werden vom Scheduler ignoriert, dieser State wird nur durch explizites Aufrufen von *vTaskSuspend()* erreicht

12 FreeRTOS Dokumentation – Tasks and Co-routines, April 2010, <http://www.freertos.org/taskandcr.html>

13 Als **Prozesskontext** bezeichnet man die gesamte Information, die für den Ablauf und die Verwaltung von Prozessen von Bedeutung ist.

14 Ein **Prozess-Scheduler** ist ein Steuerprogramm, das die die zeitliche Ausführung mehrerer Prozesse in Betriebssystemen regelt

Task Priorität

Jedem Task wird eine Priorität von 0 bis (configMAX_PRIORITIES - 1) zugewiesen.

Eine niedrige Prioritätsnummer bedeutet eine niedrige Priorität (Idle Task Priorität ist 0).

Der Scheduler preferiert Tasks mit höherer Priorität.

Task implementieren

Ein Task sollte folgende Struktur haben:

```
void vATaskFunction( void *pvParameters )
{
    for( ; ; )
    {
        -- Task application code here. --
    }
}
```

Inter-Task Kommunikation¹⁵

LumWeb verwendet zur Kommunikation zwischen Tasks ausschließlich Queues, deshalb wird hier nur auf diese Methode eingegangen.

Queues

Queues stellen die primäre Form der Inter-Task Kommunikation in FreeRTOS dar. Sie werden zum Senden und Empfangen von Nachrichten zwischen Tasks und Interrupts¹⁶ verwendet. In den meisten Fällen werden sie als thread-sichere FIFO¹⁷ Buffer verwendet.

Queues können Elemente mit „fixer“ Größe enthalten. Beim Erstellen einer Queue muss der Datentyp und die maximale Anzahl der Elemente angegeben werden.

Elemente werden nicht als Referenz, sondern als Kopie eingefügt, darum sollten die Elemente nicht zu groß sein, um den Kopieraufwand zu gering zu halten.

Um große Elemente in Queues einzufügen sollten Zeiger verwendet werden. Dabei muss aber sichergestellt werden, dass klar definiert ist, welcher Task die Daten „besitzt“, um Speicherzugriffsfehler zu vermeiden.

¹⁵ FreeRTOS Dokumentation - Inter-task Communication - <http://www.freertos.org/Inter-Task-Communication.html>

¹⁶ Unter **Interrupt** versteht man die kurzfristige Unterbrechung eines Programms, um eine andere, meist kurze, aber zeitkritische Verarbeitung durchzuführen.

¹⁷ **First In – First Out** (engl. etwa „Erster rein – Erster raus“)

Queues Erstellen

Mit dem Befehl `xQueueCreate18` wird eine neue Queue erstellt:

```
xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );
```

Aus Queue lesen

Mit dem Befehl `xQueueRecieve19` wird ein Element aus der Queue gelesen:

```
xQueueReceive( xQueue, &( pxRxedMessage ), ( portTickType ) 10 )
```

Queue senden

Mit dem Befehl `xQueueSend20` wird ein neues Element in die Queue eingefügt:

```
xQueueSend( xQueue1, newElement, ( portTickType ) 10 );
```

¹⁸ FreeRTOS Dokumentation – `xQueueCreate`, April 2010, <http://www.freertos.org/a00116.html>

¹⁹ FreeRTOS Dokumentation – `xQueueRecieve`, April 2010, <http://www.freertos.org/a00118.html>

²⁰ FreeRTOS Dokumentation – `xQueueSend`, April 2010, <http://www.freertos.org/a00117.html>

1.3.2 lwIP

Als Netzwerkstack wird lwIP verwendet, da dieser gegenüber uIP viele Vorteile bietet. Zuerst wurde der uIP Webserver verwendet. Im Zuge des Boardwechsels wurde auch der Wechsel zu lwIP vollzogen. Zu den Vorteilen zählen DHCP Unterstützung und die Möglichkeit mehrere TCP Ports zu verwenden. Dadurch können Web Client und Web Server auf dem selben System implementiert werden.

lwIP (*lightweight IP*) ist ein weit verbreiteter Open Source TCP/IP stack für Microcontroller.

Der Fokus der lwIP TCP/IP Implementierung ist ein reduzierter Ressourcenverbrauch, bei voll Umfang von TCP. Dadurch eignet sich lwIP für Embedded Systems mit wenigen 10 kB RAM Speicher und ca. 40 kB Code Speicher.

1.3.2.1 lwIP Funktionen²¹

lwIP bietet folgende Funktionen, die bei LumWeb Verwendung finden werden hervorgehoben:

- **IP** (Internet Protocol) inklusive Packet Forwarding über mehrere Network Interfaces
- **ICMP** (Internet Control Message Protocol) für Netzwerkwartung und Debugging
- **IGMP** (Internet Group Management Protocol) für Multicast Traffic Management
- **UDP** (User Datagram Protocol)
- **TCP** (Transmission Control Protocol)
- Raw/native API für verbesserte Performance
- Optional Berkeley-like socket API
- **DNS** (Domain names resolver)
- **SNMP** (Simple Network Management Protocol)
- **DHCP** (Dynamic Host Configuration Protocol)
- AUTOIP (für IPv4, konform mit RFC 3927)
- **PPP** (Point-to-Point Protocol)
- **ARP** (Address Resolution Protocol) für Ethernet

²¹ lwIP Homepage – April 2010, http://lwip.wikia.com/wiki/LwIP_Wiki

1.3.2.2 TCP/IP-Referenzmodell

Die **TCP/IP-Protokoll-Familie** ist eine Familie von rund 500 Netzwerkprotokollen, die die Basis für die Netzkommunikation im Internet bilden.

TCP/IP-Schicht	OSI-Schicht	Beispiele
Anwendung	5 bis 7	HTTP, FTP
Transport	4	TCP, UDP
Internet	3	IP
Netzzugang	1 bis 2	ARP

1.3.2.2.1 Anwendungsschicht

Die Anwendungsschicht (*engl.: Application Layer*) umfasst alle Protokolle, die mit Anwendungsprogrammen zusammenarbeiten und die Netzwerkinfrastruktur für den Austausch anwendungsspezifischer Daten nutzen.

LumWeb verwendet das HTTP und DNS als Anwendungsprotokolle.

HTTP

Das **Hypertext Transfer Protocol** (HTTP, dt. *Hypertext-Übertragungsprotokoll*) ist ein Protokoll zur Übertragung von Daten über ein Netzwerk. Es wird hauptsächlich eingesetzt, um Webseiten aus dem World Wide Web (WWW) in einen Webbrower zu laden.

DNS²²

Das **Domain Name System (DNS)** ist einer der wichtigsten Dienste im Internet. Seine Hauptaufgabe ist die Beantwortung von Anfragen zur Namensauflösung.

DHCP²³

Das **Dynamic Host Configuration Protocol (DHCP)** ermöglicht die Zuweisung der

²² Wikipedia – DNS, April 2010, http://de.wikipedia.org/w/index.php?title=Domain_Name_System&oldid=72335434

²³ Wikipedia – DHCP, April 2010, http://de.wikipedia.org/w/index.php?title=Dynamic_Host_Configuration_Protocol&oldid=72275481

Netzwerkkonfiguration an Clients durch einen Server.

1.3.2.2.2 Transportschicht

Die Transportschicht (*engl.: Transport Layer*) stellt eine Ende-zu-Ende-Verbindung her.

LumWeb verwendet TCP und UDP als Transportprotokolle.

TCP²⁴

Das Transmission Control Protocol (TCP) (zu dt. *Übertragungssteuerungsprotokoll*) ist eine Vereinbarung (Protokoll) darüber, auf welche Art und Weise Daten zwischen Computern ausgetauscht werden sollen. Alle Betriebssysteme moderner Computer beherrschen TCP und nutzen es für den Datenaustausch mit anderen Rechnern. Das Protokoll ist ein zuverlässiges, verbindungsorientiertes, paketvermittelndes Transportprotokoll in Computernetzwerken. Es ist Teil der Internetprotokollfamilie, der Grundlage des Internets.

UDP²⁵

Das **User Datagram Protocol**, kurz **UDP**, ist ein minimales, verbindungsloses Netzwerkprotokoll, das zur Transportschicht der Internetprotokollfamilie gehört. Aufgabe von UDP ist es, Daten, die über das Internet übertragen werden, der richtigen Anwendung zukommen zu lassen.

1.3.2.3 Internetschicht

Die Internetschicht (*engl.: Internet Layer*) ist für die Weitervermittlung von Paketen und die Wegewahl (Routing) zuständig. Auf dieser Schicht und den darunterliegenden Schichten werden Direktverbindungen betrachtet. Die Aufgabe dieser Schicht ist es, zu einem empfangenen Paket das nächste Zwischenziel zu ermitteln und das Paket dorthin weiterzuleiten.

LumWeb verwendet IP als Internetprotokoll.

24 Wikipedia – TPC, April 2010, http://de.wikipedia.org/w/index.php?title=Transmission_Control_Protocol&oldid=72692312

25 Wikipedia – UDP, April 2010, http://de.wikipedia.org/w/index.php?title=User_Datagram_Protocol&oldid=72571839

IP²⁶

Das **Internet Protocol (IP)** ist ein in Computernetzen weit verbreitetes Netzwerkprotokoll und stellt die Grundlage des Internets dar. Es ist die Implementierung der Vermittlungsschicht des TCP/IP-Modells bzw. der Vermittlungsschicht (engl. *Network Layer*) des OSI-Modells.

1.3.2.2.4 Netzzugangsschicht

Die Netzzugangsschicht (*engl.: Link Layer*) ist im TCP/IP-Referenzmodell spezifiziert, enthält jedoch keine Protokolle der TCP/IP-Familie. Sie ist vielmehr als Platzhalter für verschiedene Techniken zur Datenübertragung von Punkt zu Punkt zu verstehen. Die Internet-Protokolle wurden mit dem Ziel entwickelt, verschiedene Subnetze zusammenzuschließen. Daher kann die Host-an-Netz-Schicht durch Protokolle wie Ethernet, FDDI, PPP (Punkt-zu-Punkt-Verbindung) oder 802.11 (WLAN) ausgefüllt werden.

LumWeb verwendet ARP und Ethernet auf der Netzzugangsschicht.

ARP²⁷

Das **Address Resolution Protocol (ARP)** ist ein Netzwerkprotokoll, das zu einer Netzwerkadresse der Internetschicht die physikalische Adresse (HardwareAdresse) der Netzzugangsschicht ermittelt und diese Zuordnung gegebenenfalls in den sogenannten ARP-Tabellen der beteiligten Rechner hinterlegt. Es wird fast ausschließlich im Zusammenhang mit IPv4-Adressierung auf Ethernet-Netzen, also zur Ermittlung von MAC-Adressen zu gegebenen IP-Adressen verwendet, obwohl es nicht darauf beschränkt ist.

Ethernet

Ethernet ist eine Technik für ein kabelgebundenes Datennetz, das ursprünglich für lokale Datennetze (LANs) gedacht war und daher auch als LAN-Technik bezeichnet wird. Sie ermöglicht den Datenaustausch in Form von Datenpaketen zwischen den in einem lokalen Netz (LAN) angeschlossenen Geräten (Computer, Drucker und dergleichen).

²⁶ Wikipedia – IP, April 2010, http://de.wikipedia.org/w/index.php?title=Internet_Protocol&oldid=72025803

²⁷ Wikipedia – ARP, April 2010, http://de.wikipedia.org/w/index.php?title=Address_Resolution_Protocol&oldid=71170206

1.3.3 Stellaris® Graphic Library

Die Stellaris Graphic Library ist eine Sammlung an Widgets zum Darstellen grafischer Objekte.

Durch den modularen und treiber-basierten Aufbau ist es möglich nur mit ändern des Grafiktreibers die gesamte Oberfläche auf eine andere Displayhardware zu portieren. Der Treiber ist zuständig für den Ablauf beim Zeichnen eines Pixels.

Der modulare Aufbau mit Widgets ist der zweite große Vorteil. Dadurch kann mit einfachen Mittel sehr schnell eine anspruchsvolle graphische Oberfläche erstellt werden. Die Widgets werden dabei in eine Baumstruktur eingehängt, die den Z-Index, also welche Widgets es überdeckt und von welchen Widgets es überdeckt wird, der Objekte festlegt.

Beispiele für Widgets sind:

- Buttons
- Checkboxes
- Slider
- Images
- Geometrische Figuren (CanvasWidget, Circle, ...)
- und viele mehr

Weiters beinhaltet die Stellaris Graphic Library auch die Unterstützung für die Touch-Screen Einheit. Diese Touch-Screen Funktionen werden mit dem Zeichnen eines Widgets sofort für selbiges aktiviert. Das heißt es ist nur mehr das wiederholende Aufrufen des Touch-Screen-Handlers erforderlich, jedoch keine Abfrage über den aktuellen Stand des Touch-Screens. Dies ist Aufgabe der Stellaris Graphic Library.

Aufruf der zyklischen Touch-Screen Aktionen:

```
while (1)
{
    //
    // Process any messages in the widget message queue.
    //
    WidgetMessageQueueProcess();
}
```

Die Informationen des Touch-Screens werden über einen Interrupt des ADC's in die WidgetMessageQueue geschrieben. Im obigen Beispiel wird der Aufruf des Message-Handlers

demonstriert (WidgetMessageQueueProcess). Damit dieser korrekt funktioniert muss der TouchScreenIntHandler in das Interrupt-Array eingetragen werden.

1.3.4 Stellaris® Peripheral Driver Library

Die Stellaris® Peripheral Driver Library ist eine Sammlung von Funktionen zur Ansteuerung der Peripherie-Bausteine der Stellaris Familie der ARM Cortex-M3 basierten Mikrocontroller. Sie ermöglichen die einfache Nutzung der Peripherie-Bausteine.

Bei LumWeb wird die Peripheral Driver Library unter anderem zum Zugriff auf die UART-Schnittstelle, die Netzwerk-Schnittstelle und Ports verwendet.

1.3.5 FatFS

Die Bibliothek *FatFS* stellt Funktionen für das Arbeiten mit einem Fat-16 Dateisystem zur Verfügung. Sie beinhaltet Funktionen zum Zugriff und zur Manipulation von Dateien:

```
FRESULT f_mount (BYTE, FATFS*); /* Mount/Unmount a logical drive */
FRESULT f_open (FIL*, const XCHAR*, BYTE); /* Open or create a file */
FRESULT f_read (FIL*, void*, UINT, UINT*); /* Read data from a file */
FRESULT f_write (FIL*, const void*, UINT, UINT*); /* Write data to a file */
FRESULT f_lseek (FIL*, DWORD); /* Move file pointer of a file object */
FRESULT f_close (FIL*); /* Close an open file object */
FRESULT f_opendir (DIR*, const XCHAR*); /* Open an existing directory */
FRESULT f_readdir (DIR*, FILINFO*); /* Read a directory item */
FRESULT f_stat (const XCHAR*, FILINFO*); /* Get file status */
FRESULT f_getfree (const XCHAR*, DWORD*, FATFS**); /* Get number of free
clusters on the drive */
FRESULT f_truncate (FIL*); /* Truncate file */
FRESULT f_sync (FIL*); /* Flush cached data of a writing file */
FRESULT f_unlink (const XCHAR*); /* Delete an existing file or directory */
FRESULT f_mkdir (const XCHAR*); /* Create a new directory */
FRESULT f_chmod (const XCHAR*, BYTE, BYTE); /* Change attriburte of the file/dir */
FRESULT f_utime (const XCHAR*, const FILINFO*); /* Change timestamp of the
file/dir */
FRESULT f_rename (const XCHAR*, const XCHAR*); /* Rename/Move a file or
directory */
FRESULT f_forward (FIL*, UINT(*)(const BYTE*,UINT), UINT, UINT*); /* Forward
data to the stream */
FRESULT f_mkfs (BYTE, BYTE, WORD); /* Create a file system on the drive */
```

```
FRESULT f_chdir (const XCHAR*); /* Change current directory */
FRESULT f_chdrive (BYTE); /* Change current drive */
```

Bei LumWeb wird FatFS vom Webserver und den Logging-Funktionen zum Zugriff auf die SD-Karte verwendet. Daraus ergeben sich Einschränkungen bei den Dateinamen. Dateinamen dürfen maximal acht Zeichen lang sein, darauf folgt ein Punkt. Die Dateierweiterung darf maximal drei Zeichen lang sein.

1.3.5.1 Beispiele - Dateioperationen

- Öffnen

```
FRESULT rc = FR_NO_FILE;
log_file = (FIL*) pvPortMalloc(sizeof(FIL));
rc = f_open(log_file, LOG_FILE_PATH, FA_WRITE);
```

- Daten einlesen

```
FRESULT rc;
UINT usBytesRead;
char buffer[30];
rc = f_read(log_file, buffer, 30, &usBytesRead);
```

- Daten schreiben

```
FRESULT rc;
UINT usBytesWritten;
char buffer[30];
rc = f_write(log_file, buffer, strlen(buffer), &usBytesWritten);
f_sync(log_file);
```

- Schließen

```
f_close(log_file);
```

1.3.6 Make

Das Programm *make* dient zum Compileren und Linken von großen Programmen. *Make* erkennt automatisch, welche Dateien sich geändert haben, nur diese werden neu kompiliert. Dazu muss ein sogenanntes „Makefile“ erstellt werden. In diesem werden alle Abhängigkeiten, Quelltext-Dateien und Kompilereinstellungen des Projekts angegeben. Danach kann das Projekt mit einem Aufruf von *make* komplett kompiliert werden.

LumWeb verwendet die GNU Implementierung von *Make*, da diese am weitesten verbreitet ist.

1.3.6.1 Erstellen eines Makefiles

In einem *Makefile* können Variablen verwendet werden:

- Setzen einer Variable
`VAR1=Wertstring` – der Variable `VAR1` wird der String „Wertstring“ zugewiesen
- Verwenden von Variablen
`$(VAR1)` – auf den Wert der Variable wird zugegriffen

Die Quelltext-Dateien werden im *Makefile* registriert.

Ausschnitte:

```
SOURCE_DIR=./uInterface

SOURCE=    syscall.c \
           $(SOURCE_DIR)/main.c \
           $(SOURCE_DIR)/setup.c \
           $(SOURCE_DIR)/timer.c \
           $(SOURCE_DIR)/realtime.c \
           $(SOURCE_DIR)/lmi_fs.c \
```

Folgende Kompiler- und Linker-Einstellungen werden im *Makefile* vorgenommen, um ein ausführbares Programm auf der Zielplattform (LM3S9B96) erzeugen zu können :

```
CC=arm-none-eabi-gcc
OBJCOPY=arm-none-eabi-objcopy
LDSCRIPT=standalone.ld
```

```
LINKER_FLAGS= -Xlinker -o$(NAME).axf -Xlinker -M -Xlinker -Map=rtosdemo.map -T$(LDSCRIPT)
#-nostartfiles -Xlinker --no-gc-sections
DEBUG=-g
```

`OPTIM=-O1`

```
CFLAGS=-I $(SOURCE_DIR) \
-I external \
-I $(SOURCE_DIR)/ethernet \
-I $(LWIP_COMMON_DIR)/src/include \
-I $(LWIP_COMMON_DIR)/src/include/ipv4 \
-I $(LWIP_COMMON_DIR)/port/LM3S \
-I $(RTOS_SOURCE_DIR)/include \
-I $(RTOS_SOURCE_DIR)/portable/GCC/ARM_CM3 \
-I $(LUMINARY_DRIVER_DIR) \
-I $(LUMINARY_DRIVER_DIR)/drivers \
-I $(LUMINARY_DRIVER_DIR)/driverlib \
-I $(LUMINARY_DRIVER_DIR)/inc \
-D inline= -mthumb -mcpu=cortex-m3 \
$(OPTIM) $(DEBUG) \
-ffunction-sections \
-Wall \
-D sprintf=usprintf -D snprintf=usnprintf \
-D _sprintf=usprintf -D _snprintf=usnprintf \
-D printf=UARTprintf \
-Dgcc \
-D malloc=pvPortMalloc -D free=vPortFree \
```

Dabei werden mit „-I“ die Verzeichnisse angegeben, die nach Header-Dateien durchsucht werden, mit „-mcpu“ der CPU-Typ des Zielsystems gesetzt, mit `-D sprintf=usprintf` Funktionen der C-Standardbibliothek durch optimierte Versionen ersetzt (z.B: die jedes Vorkommen von `sprintf` wird vom Kompiler im erzeugten Code durch `usprintf` ersetzt - „-D sprintf=usprintf“) und weitere Optimierungen vorgenommen.

Der Aufruf des Compilers erfolgt über eine sogenannte „Target“-Anweisung, dabei werden die zu kompilierenden Quelltext-Dateien angegeben. Folgendes Beispiel kompiliert die Datei `startup.c` und erzeugt die Object-Datei `startup.o`:

```
startup.o : startup.c
$(CC) -c $(CFLAGS) -O1 startup.c -o startup.o
```

1.3.6.2 LumWeb kompilieren

Das gesamte Projekt kann mit dem Aufruf von „make all“ im Verzeichnis *src/* übersetzt werden.

Auf einem Unix-System wird in einer *Shell* zuerst mit dem Befehl *cd* in das Verzeichnis gewechselt und „make all“ aufgerufen:

```
~/] cd projects/lumweb/src  
~/] make all
```

Mit dem Aufruf von „make flash“ wird das ausführbare Programm auf das LM3S9B96-Board mit OpenOCD übertragen.

2 Konfigurationen

2.1 FreeRTOS-Konfiguration

Die global Konfiguration für FreeRTOS befindet sich in der Header Datei *uInterface/FreeRTOSConfig.h*. In dieser Datei wird das Verhalten von FreeRTOS konfiguriert. Da die Einstellungen über Define-Anweisungen²⁸ in C-Headerdateien getroffen werden, können sie nur zur Übersetzungszeit verändert werden, d.h. Änderungen werden erst wirksam, wenn der Quelltext neu kompiliert und auf das Board geladen wird.

2.1.1 Tasks

Alle taskspezifischen Einstellungen, wie Stack-Größe, Task-Name, Task-Priorität und Task-Handler werden in der Datei *uInterface/taskConfig.h* gemacht.

2.1.1.1 lwIP-Task

- *LWIP_STACK_SIZE*
Stack Größe des lwIP-Tasks, Standardwert: 512*2 (Speicherstellen)
- *LWIP_TASK_NAME*
Name des lwIP-Tasks, Standardwert: "lwip"
- *LWIP_TASK_PRIORITY*
Priorität des lwIP-Tasks, Standardwert: (configMAX_PRIORITIES - 2)

2.1.1.2 Graphic-Task

- *GRAPH_STACK_SIZE*
Stack Größe des Graphic-Tasks, Standardwert: 512 (Speicherstellen)
- *GRAPH_TASK_NAME*
Name des Graphic-Tasks, Standardwert: "graphics"
- *GRAPH_TASK_PRIORITY*
Priorität des Graphic-Tasks, Standardwert: (configMAX_PRIORITIES - 2)

28 Makro-Anweisungen für den **C-Präprozessor**

2.1.1.3 Com-Task

- *COM_STACK_SIZE*
Stack Größe des Com-Tasks, Standardwert: 128*2 (Speicherstellen)
- *COM_TASK_NAME*
Name des Com-Tasks, Standardwert: "com"
- *COM_TASK_PRIORITY*
Priorität des Com-Tasks, Standardwert: (configMAX_PRIORITIES - 2)

2.1.1.4 Real Time Task

- *TIME_TASK_NAME*
Name des Real Time Tasks, Standardwert: "clock"
- *TIME_TASK_PRIORITY*
Priorität des Real Time Tasks, Standardwert: (configMAX_PRIORITIES)

2.1.2 Queues

Alle queuespezifischen Einstellungen, wie Queue-Größe und Queue-Handler werden in der Datei *uInterface/queueConfig.h* vorgenommen.

2.1.2.1 Com-Queue

- *COM_QUEUE_SIZE*
Anzahl der Elemente, die eine Com-Queue aufnehmen kann, Standardwert: 6

2.1.2.2 HTTPD.Queue

- *HTTPD_QUEUE_SIZE*
Anzahl der Elemente, die eine HTTPD-Queue aufnehmen kann, Standardwert: 6

2.2 lwIP-Konfiguration

Die Einstellungen für den lwIP-Stack werden in der Datei *uInterface/ethernet/lwipopts.h*

vorgenommen..

- *SYS_LIGHTWEIGHT_PROT*
Wert: 1
- *NO_SYS*
Wert: 1, lwIP wird in Kombination mit einem Realtime Operation System verwendet, der lwIP Stack braucht sich um das Timing nicht zu kümmern.

2.3 LumWeb-Konfiguration

2.3.1 Allgemeine Einstellungen

Allgemeine Einstellungen zu LumWeb werden in der Datei *uInterface/setup.h* vorgenommen.

Da die Einstellungen über Define-Anweisungen in C-Headerdateien getroffen werden, können sie nur zur Übersetzungszeit verändert werden, d.h. Änderungen werden erst wirksam, wenn der Quelltext neu kompiliert und auf das Board geladen wird.

- *SYSTICK_INT_PRIORITY*
Priorität des Systick Interrupts, Standardwert: 0x80
- *ETHERNET_INT_PRIORITY*
Priorität des Ethernet Interrupts, Standardwert: 0xC0
- *ENABLE_LOG*
Aktiviert die Ausgabe von Statusmeldungen in die Log-Datei auf der SD Karte */log/sys.log* , Standardwert: 0
- *ENABLE_GRAPHIC*
Aktiviert den grafischen Client , Standardwert: 0
- *ENABLE_SNTP*
Aktiviert den lwIP SNTP Client, Standardwert: 0
- *ENABLE_DNS*
Aktiviert den lwIP DNS Client, Standardwert: 0
- *ENABLE_NET_BIOS*
Aktiviert den lwIP NETBios Client, Standardwert: 0

2.3.2 Debuginformationen

Die Ausgabe von Debug-Informationen über die UART Schnittstelle²⁹ kann in der Datei uInterface/setup.h für die jeweiligen Teilbereiche aktiviert werden.

Da die Einstellungen über Define-Anweisungen in C-Headerdateien getroffen werden, können sie nur zur Übersetzungszeit verändert werden, d.h. Änderungen werden erst wirksam, wenn der Quelltext neu kompiliert und auf das Board geladen wird.

- *DEBUG_MEMORY*
Aktiviert die Ausgabe von Debug-Informationen für das Speicherhandling, Standardwert: 0
- *DEBUG_GRAPHIC*
Aktiviert die Ausgabe von Debug-Informationen für die grafische Ausgabe, Standardwert: 0
- *DEBUG_HTTPC*
Aktiviert die Ausgabe von Debug-Informationen für den Webclient, Standardwert: 0
- *DEBUG_GRAPHIC_EDITOR*
Aktiviert die Ausgabe von Debug-Informationen für das Editieren von Werten mit der grafischen Oberfläche, Standardwert: 0
- *DEBUG_SSI*
Aktiviert die Ausgabe von Debug-Informationen für die SSI Funktionen (siehe [3.3.1.SSI](#)), Standardwert: 0
- *DEBUG_SSI_PARAMS*
Aktiviert die Ausgabe von Debug-Informationen für die SSI Parameterverarbeitung (siehe [3.3.1.2. SSI Parameter](#)), Standardwert: 0
- *DEBUG_LOG*
Aktiviert die Ausgabe von Debug-Informationen für das Mitprotokollieren in die Logdatei (siehe [3.5.Debugging und Logging](#)), Standardwert: 0
- *DEBUG_TAGS*
Aktiviert die Ausgabe von Debug-Informationen für die Tag-Library (siehe [3.2.Tag Library](#)), Standardwert: 0

²⁹ Wikipedia – UART, Mai 2010, http://de.wikipedia.org/w/index.php?title=Universal_Asynchronous_Receiver_Transmitter&oldid=64893899

- *DEBUG_CGI*
Aktiviert die Ausgabe von Debug-Informationen für die CGI-Funktionen (siehe [3.3.2.CGI](#)),
Standardwert: 0
- *DEBUG_COM*
Aktiviert die Ausgabe von Debug-Informationen für den COM Task (siehe [3.4.Com-Task](#)),
Standardwert: 0

2.3.3 Systemeinstellungen

Über die Konfigurationsdatei auf der SD Karte (*/conf/ipconfig.cnf*) werden die Einstellungen beim Starten des Systems gesetzt. Die Einstellungen beim Systemstart werden aus der Konfigurationsdatei gelesen. Es ist also möglich, diese Einstellungen mit einem Neustart zu ändern, ohne das Programm neu übersetzen zu müssen.

Die Einstellungen betreffen die Netzwerkkonfiguration und Einstellungen zur Betriebsart von LumWeb (siehe [3.1.1.Betriebsarten](#)).

Folgende Einstellungen sind vorhanden:

- *USE_DHCP*
Legt fest, ob DHCP zur Adressvergabe verwendet wird
Mögliche Werte:
 - true
 - false (Standardwert)
- *DEFAULT_MENU_PAGE*
Legt die Startseite des Webclients fest
Mögliche Werte:
 - Name einer vorhandenen Webseite auf dem Serversystem
 - Standardwert: index.ssi
- *DEFAULT_SET_PAGE*
Name der CGI-Funktion, die zum Verarbeiten der Änderungen verwendet wird
Mögliche Werte:
 - Name der CGI-Funktion am Server
 - Standardwert: set.cgi
- *IS_SERVER*

Legt fest, ob das System als HTTP-Server arbeitet (siehe [3.1.1.Betriebsarten](#))

Mögliche Werte:

- true
- false
- *IS_CLIENT*

Legt fest, ob das System als HTTP-Client arbeitet (siehe [3.1.1.Betriebsarten](#))

Mögliche Werte:

- true
- false
- *REMOTE_IP*

Gibt die IP Adresse des Servers an, von dem das Benutzerinterface geladen wird und wohin die Änderungen übertragen werden (nur für Client-Systeme relevant)

Mögliche Werte:

- valide IPv4 Adresse³⁰
 - *TIME_ZONE*
- Gibt die Differenz zur UTC³¹ an (nur relevant, wenn SNTP aktiviert ist, siehe [2.3.1.Allgemeine Einstellungen](#))

Mögliche Werte:

- zum Beispiel 1 für die Timezone UTC+1 (Österreich)
- *IP_ADDRESS*

Gibt die IP Adresse des Systems an

Mögliche Wert:

- valide IPv4-Adresse
- *IP_SUBNETMASK*

Gibt die Subnetzmaske des Systems an

Mögliche Werte:

- valide IPv4-Subnetzmaske³²
- *IP_GATEWAY*

Gibt den Gateway³³ des Systems an

³⁰ IPv4 benutzt 32-Bit-Adressen, die in vier Blöcke je 0 bis 255 unterteilt sind

³¹ Die **koordinierte Weltzeit**, international **UTC** (engl. *Universal Time Coordinated*)

³² Wikipedia – Subnetz, Mai 2010, <http://de.wikipedia.org/w/index.php?title=Subnetz&oldid=73749190>

³³ Ein **Gateway** erlaubt es Netzwerken, die auf völlig unterschiedlichen Protokollen basieren, miteinander zu

Mögliche Werte:

- valide IPv4-Adresse

kommunizieren.

3 LumWeb

LumWeb besteht aus 3 Komponenten:

- **Embedded Webserver**

stellt die Benutzeroberfläche in Form von HTML-Seiten über HTTP zur Verfügung und wertet die Eingaben der Clients aus

- **Embedded WebClient**

stellt die HTML-Seiten auf dem Touch-Screen dar und überträgt die Eingaben zum Server

- **Com-Task**

verwaltet die Kommunikation zwischen Webserver und der zu steuernden Maschine.

3.1 Konzept

Das Ziel von LumWeb ist die Vereinfachung der Userinterface-Erstellung.

Erreicht wird das Ziel durch das Zusammenspiel der drei oben genannten Komponenten.

Das User-Interface wird als HTML Seite mit SSI-Tags ([3.3.1 SSI](#)) definiert. Die Bedienelemente sind auf einem Serversystem gespeichert. Das Serversystem ist zuständig für die Kommunikation mit der Maschine und mit dem Benutzer. Die Seiten werden den Clients über das HTTP-Protokoll zur Verfügung gestellt. Auf PC-Systemen übernimmt der Webbrower die Darstellung, auf Embedded-Systems übernimmt der WebClient die Darstellung. Dazu wertet der WebClient spezielle HTML-Kommentare aus, die für die SSI-Tags vom Server eingefügt werden.

Um eine Bedienelemente zu erstellen werden in den HTML-Code Bedienelemente in Form von SSI-Tags eingebaut, zum Beispiel der Tag „IntegerInputField“ für die Eingabe einer Zahl. Zusätzlich muss eine eindeutige Id angegeben werden, die das Element mit einer Einstellung auf der Maschine verbindet. Der Com-Task wertet diese Tag-ID aus und liefert den entsprechenden Wert. Der Webserver fügt beim Auswerten der Tags außerdem spezielle HTML-Kommentare ein, die der WebClient zur Darstellung auswertet. Der Client kann jedoch nur solche Tags darstellen, für die er Darstellungsmethoden implementiert hat.

3.1.1 Betriebsarten

Ein LumWeb System kann auf zwei verschiedene Arten betrieben werden:

- **Server**

Zentrales System, das sich um die Bereitstellung des Benutzerinterfaces (über HTTP) und die Kommunikation mit der Maschine kümmert

- **Client**

Ein System, an dem Einstellungen über ein Benutzerinterface (welches vom Server geladen wird) vorgenommen werden können und Eingaben zurück an das Serversystem gesendet werden

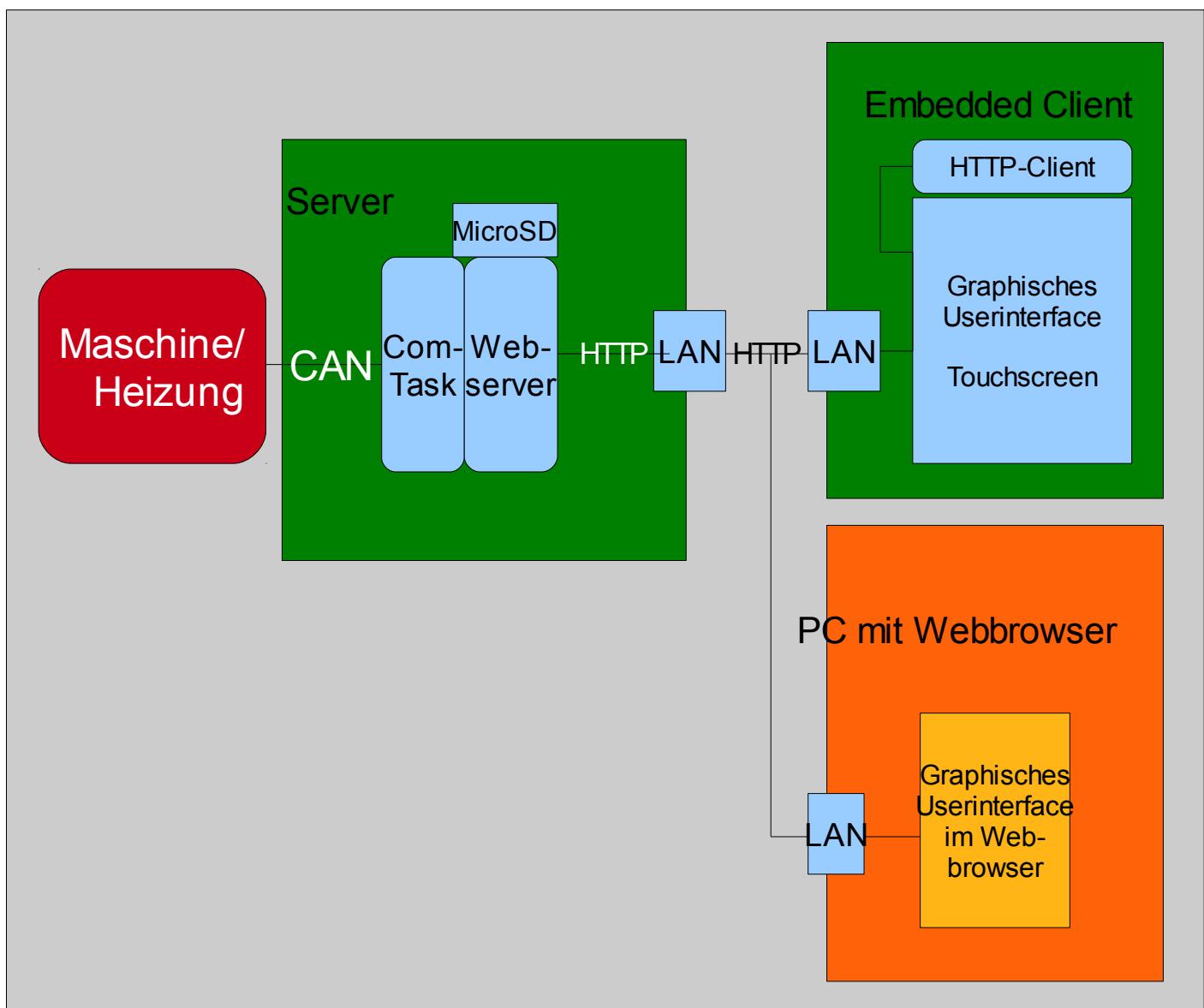


Abbildung 3.1: Schematischer Aufbau (Quelle: Anzinger, Hahn)

3.1.2 Tasks

LumWeb ist modular aufgebaut und die Logik ist in mehrere Tasks aufgeteilt. Folgende Grafik gibt einen Überblick, in welchem Zusammenhang diese stehen und wie sie miteinander kommunizieren.

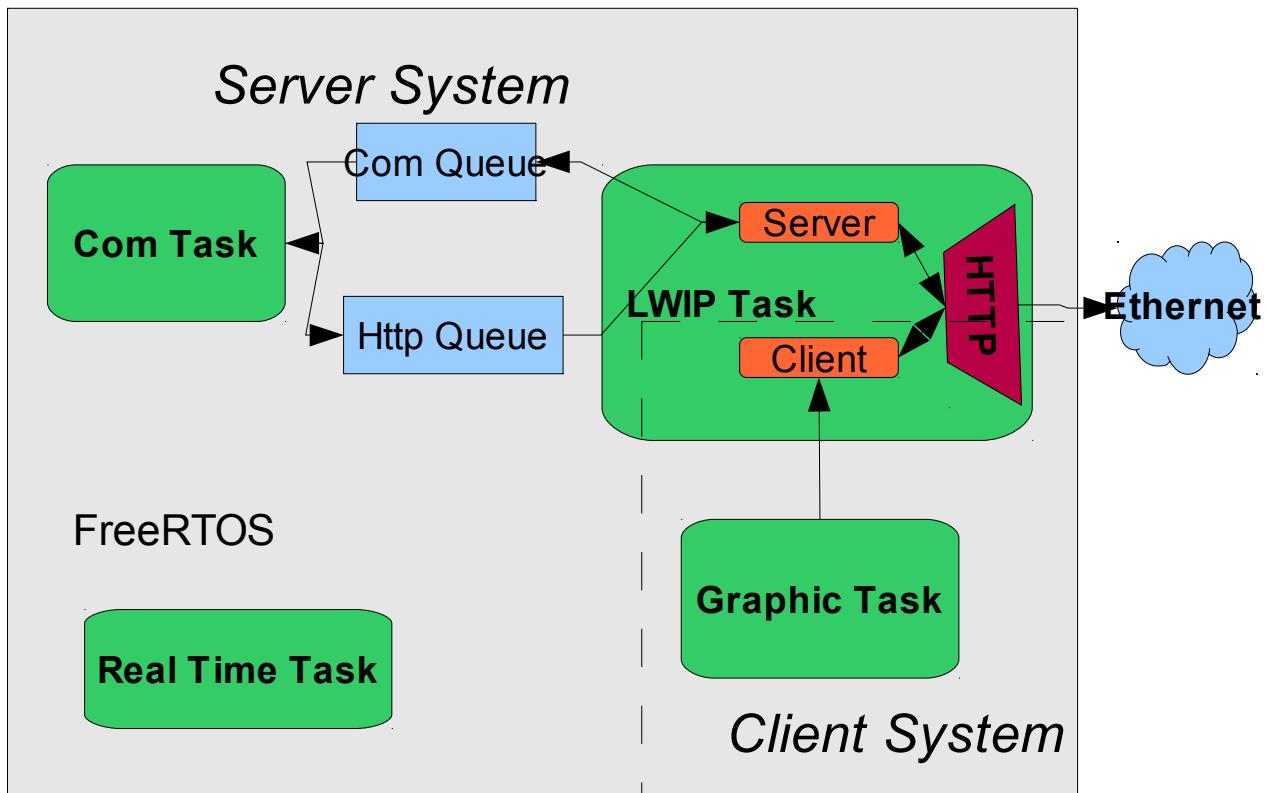


Abbildung 3.2: Intertaskkommunikation (Quelle: Anzinger, Hahn)

3.1.3 Com-Task

Dieser Task ist für die Kommunikation mit der Maschine verantwortlich. Der Webserver stellt Anfragen an den Task via Com-Queue und liest Werte via Http-Queue zurück. Der Com-Task ist nur auf dem Server-System aktiv, nicht auf Client-Systemen. (siehe [3.4.Com-Task](#))

3.1.4 lwIP-Task

Auf dem Server-System wird der Webserver im lwIP-Task betrieben, im Client-System ist der Webserver deaktiviert und der lwIP-Task nur für die Verwaltung der Netzwerkschnittstelle verantwortlich. (siehe [1.3.2.lwIP](#), [3.3.Webserver](#))

3.1.5 Graphic-Task

Der Graphic-Task ist für die Darstellung des Nutzerinterfaces auf einem Touch-Screen verantwortlich. Dazu verwendet er den WebClient (siehe [3.4.Webclient](#)).

3.1.6 Real-Time-Clock-Task

Der Real Time Clock Task verwaltet die Systemuhr und stellt Zeit und Datumsfunktionen zur Verfügung. Ist SNTP³⁴ aktiviert und verfügt das System über eine Internetverbindung wird die Systemzeit mit einem Zeitserver im Internet synchronisiert.

3.1.7 Intertask-Kommunikation

Die Kommunikation zwischen dem lwIP Task (dem Webserver) und dem Com-Task ist über Queues realisiert. (Siehe [1.3.1.2. Multitasking](#), [3.5.Com-Task](#))

³⁴ Das **Network Time Protocol (NTP)** ist ein Standard zur Synchronisierung von Uhren in Computersystemen über paketbasierte Kommunikationsnetze

3.2 Tag-Library

3.2.1 Aufbau

Die Tag-Library oder kurz Taglib ist ein Kernstück des LumWeb Web-Servers und des Embedded Clients. Sie beinhaltet alle Tags, die für die Funktionalität der Oberfläche verwendet werden können.

Ein Tag ist im programmiertechnischen Sinn eine Struktur mit indirekten Funktionsaufrufen. Ein Tag setzt sich wie folgt zusammen:

```
typedef struct __taglibStruct
{
    int tagindex;
    char* tagname;

    void (* renderSSI)(char * pcBuf, int iBufLen, pSSIParam *params);

    void (* onLoad)(char*, int, void* basicDisplayLine);
    tWidget* (* onDisplay)(void* basicDisplayLine, int);
    void (* onEditValue)(void* basicDisplayLine);
    void (* onDestroy)(void* basicDisplayLine);
    char* (* strFormatter)(void* basicDisplayLine);

    void* userSpace;
} taglib;
```

3.2.2 Funktionen in der Taglib

3.2.2.1 renderSSI

```
void myRenderSSI(char * pcBuf, int iBufLen, pSSIParam *params);
```

Diese Funktion wird vom Webserver aufgerufen, um den SSI-Tag in ein standardkonformes HTML zu übersetzen und den für die GUI notwendigen HTML Kommentar zu erzeugen.

3.2.2.2 *onLoad*

```
void myOnLoad (char*, int, void* basicDisplayLine);
```

Diese Funktion wird vom Embedded Client aufgerufen um aus dem GUI Kommentar eine *basicDisplayLine* zu erzeugen und in die Liste aller anzuzeigenden Objekte einzubinden. Es ist auch möglich, dass nur Attribute für die GUI gesetzt werden, wie zum Beispiel zum Anzeigen des „Speichern“ oder „Menu“ Buttons ein Bit zu setzen.

3.2.2.3 *onDisplay*

```
tWidget* myOnDisplay (void* basicDisplayLine, int zeilennummer);
```

Diese Funktion wird beim Anzeigen des Objekts am Display ausgeführt und ist zuständig zum generieren der Stellaris GUI-Library-Objekte. Als Parameter werden das darzustellende Objekt angegeben und die Zeilennummer in der das Objekt angezeigt wird.

3.2.2.4 *onEditValue*

```
void myOnEditValue (void* basicDisplayLine);
```

Diese Funktion kann von der „onDisplay“ Funktion aufgerufen werden. Sie ist gedacht als Funktion, die den Editor beinhaltet. Dieser kann zum Beispiel eine eigene Oberfläche zur komfortablen Manipulation von nummerischen Werten erstellen.

3.2.2.5 *onDestroy*

```
void myOnDestroy (void* basicDisplayLine);
```

Diese Funktion wird beim Löschen des Objekts aufgerufen. Dabei müssen nur Speicherbereiche freigegeben werden, die zusätzlich zur *basicDisplayLine* angefordert wurden.

3.2.2.6 *strFormatter*

```
char* myStrFormatter (void* basicDisplayLine);
```

Der String-Formatter ist für die korrekte Umwandlung des Wertes zuständig, der auf der GUI dargestellt wird.

3.2.2.7 UserSpace

Der userSpace ist ein undefinierter Pointer zur Speicherung von Tag-Spezifischen Informationen. Dies wurde nur als erweiterung für Aufwendigere Tags gedacht und ist bei unseren Beispielen nicht verwendet worden.

3.2.3 Erstellen eines Tags

Zum Erstellen eines Tags muss zuerst der statische Speicher dafür reserviert werden. Dies erfolgt durch das Erhöhen eines Wertes in der *taglib/tags.h* Datei um die gewünschte Taganzahl:

```
#define NUM_CONFIG_TAGS
```

Anschließend müssen folgende Dateien im Ordner *taglib/tags* angelegt werden:

- <gewünschter Name>.h (Headerdatei)
- <gewünschter Name>.c (Quelltextdatei)

Die Headerdatei beinhaltet die Prototypen der einzelnen Funktionen. Diese Funktionen müssen dem Aufbau der **taglib**-Struktur entsprechen. Implementiert werden die Funtkionen in der entsprechenden Quelltextdatei.

Danach werden in der Datei *taglib/tags.h* die entsprechende include-Anweisung getätigert.

```
#include "taglib/tags/<gewünschter Name>.h"
```

Schließlich werden noch die Zuweisungen in der Datei *taglib/tags.c* getätigert. Diese sind wie in dem folgenden Beispiel vorzunehmen:

```
xTagList[8].tagindex = 8;
xTagList[8].tagname = "FloatInputField";
xTagList[8].renderSSI = vFloatRenderSSI;
xTagList[8].onLoad = vFloatOnLoad;
xTagList[8].onDisplay = xFloatOnDisplay;
xTagList[8].onEditValue = vDummyOnEditValue;
xTagList[8].onDestroy = vDummyOnDestroy;
xTagList[8].strFormatter = pcFloatStrFormatter;
```

```
xTagList[8].userSpace = NULL;
```

Hier ist die 8 durch den aktuellen Index und die Funktionsnamen rechts vom Gleichheitszeichen durch die eigenen Funktionen zu ersetzen.

Danach wird das Projekt jetzt neu kompiliert und auf das Board gespielt, somit können die Tags in den Seiten verwendet werden.

3.3 Webserver

Der Webserver stellt die zentrale Komponente des Systems dar. Der LumWeb-Webserver ist eine modifizierte Version des lwIP-Webservers.

Zu den Modifikationen zählen:

- Unterstützung von SSI-Paramtern (siehe [3.3.1.2 SSI Parameter](#))
- weitere SSI-Tags (siehe [3.3.1 SSI](#))
- CGI Routine zum Speichern von Werten via Com-Task (siehe [2.1.1.3 Com-Task](#))

3.3.1 SSI

Server Side Includes³⁵ (dt.: Serverseitige Einbindungen), auch bekannt als **SSI**, sind in (meist HTML-) Dokumente eingebettete, einfach zu nutzende Scriptbefehle, die auf dem Webserver ausgeführt werden, bevor das Dokument an den Client ausgeliefert wird. Sie stellen eine einfacher zu verwendende Alternative zu Programmen/Scripten dar, die das ganze Dokument dynamisch generieren. Bei LumWeb werden SSI-Tags unter anderem dazu verwendet, aktuelle Werte von der Maschine dynamisch in einem HTML-Dokument darzustellen.

3.3.1.1 Syntax

Server Side Includes haben die folgende Syntax:

```
<!--#tagname parametername1="wert" parametername2="wert" -->
```

Das einleitende <!-- und das abschließende --> entsprechen den Zeichen für Beginn und Ende eines HTML- oder XML-Kommentares und sorgen dafür, dass der Browser die Befehle nicht anzeigt, falls SSI deaktiviert ist. Ist SSI aktiviert, wird der Kommentar durch den Rückgabe-Text einer SSI Funktion ersetzt.

Der lwIP Webserver unterstützt SSI-Tags in Form von C Funktionen.

³⁵ Wikipedia - Server Side Includes, April 2010, http://de.wikipedia.org/w/index.php?title=Server_Side_Includes&oldid=68847699

Die Signatur einer solchen C Funktion muss wie folgt aussehen:

```
void vCheckboxRenderSSI(char * pcBuf, int iBufLen, pSSIParam *params)
```

Die SSI C Funktion schreibt das Ergebnis in den Rückgabebuffer pcBuf. Dieser wird dann vom Webserver ausgegeben.

Dazu muss die C Funktion noch mit einem SSI-Tagnamen verbunden werden. Das erfolgt über das Feld *tagname* in der Struktur *xTagList*.

Beispielinitialisierung:

```
xTagList[0].tagname = "IntegerField";  
xTagList[0].renderSSI = vIntegerRenderSSI;
```

Der Webserver überprüft beim Parsen der HTML Datei ob SSI-Tags vorkommen. Wird ein Tag gefunden, wird überprüft, ob eine SSI Funktion am Server zur Verfügung steht. Dazu wird der Tagname aus dem SSI überprüft und die zugehörige Funktion vom SSIMHandler aufgerufen. Diese Funktion wird durch das durchsuchen der Taglib-Liste nach dem Tagnamen herausgefunden.

3.3.1.2 SSI Parameter

Der lwIP-Webserver unterstützt keine SSI-Parameter, daher wurde diese Funktion im Rahmen der Diplomarbeit entwickelt und in den lwIP-Webserver eingehängt. Dazu wurde die Funktion `send_data` des lwIP-Webservers (befindet sich in *uInterface/ethernet/httpd/httpd.c*) um entsprechende Parsing-Funktionen erweitert.

Die SSI Parameter werden über eine einfach verkettete Liste verwaltet:

```
typedef struct SSIPParam
{
    char *name;
    char *value;
    struct SSIPParam* next;
} SSIPParam;
```

Die Liste ist in die Struktur `http_state` des lwIP-Webservers eingehängt, um sie an die einzelnen SSI-Funktionen übergeben zu können. Dazu wird die Parameterliste im SSI-Handler (SSIHandler) an die SSI-Funktion übergeben.

Zur Manipulation der Parameterliste stehen folgende Funktionen zur Verfügung:

Fügt eine Element hinzu

```
int SSIPParamAdd(pSSIPParam* root, char* nameValue);
```

Gibt das Element aus der Liste mit \$name zurück

```
pSSIPParam SSIPParamGet(pSSIPParam root, char* name);
```

Löscht alle Elemente aus der Liste und gibt den Speicher frei

```
void SSIPParamDeleteAll(pSSIPParam* root);
```

Gibt den Wert des Elements mit \$name aus der Liste zurück

```
char* SSIPParamGetValue(pSSIPParam root, char* name);
```

Die genaue Funktionsdokumentation ist in der Code-Dokumentation zu finden.

3.3.1.3 Vorhandene SSI-Tags

3.3.1.3.1 Eingabe-Tags

Über Eingabe-Tags können Werte manipuliert werden.

3.3.1.3.1.1 IntegerInputField

Ein IntegerInputField dient zur Manipulation von ganzzahligen Werten. Der aktuelle Wert wird in einem HTML-Eingabefeld angezeigt. Zusätzlich werden zu dem Feld noch Plus und Minus Buttons angezeigt.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

SSI-Tag

```
<!--#IntegerInputField id=id_fuer_maschine label=Integer max=10 min=0 increment=1 -->
```

Parameter

- id → Verbindet das Eingabefeld mit einer Einstellung auf der Maschine, der aktuelle Wert von *id* wird per Com-Task von der Maschine geladen und angezeigt
- label → Name, der beim Eingabefeld angezeigt wird
- max → Maximalwert; ganzzahlig
- min → Minimalwert; ganzzahlig
- increment → Schrittweite, um die der Wert beim drücken der Plus oder Minus Buttons erhöht, bzw erniedrigt wird

HTML-Darstellung



Abbildung 3.3:

Browserdarstellung

Integer Eingabefeld

(Quelle: Anzinger,

Hahn)

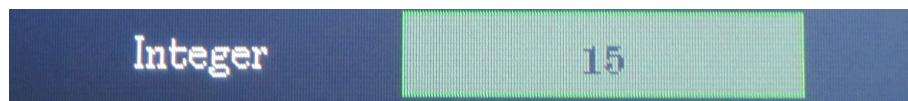
Touchscreen-Darstellung

Abbildung 3.4: Menüdarstellung Integerwert (Quelle: Anzinger, Hahn)

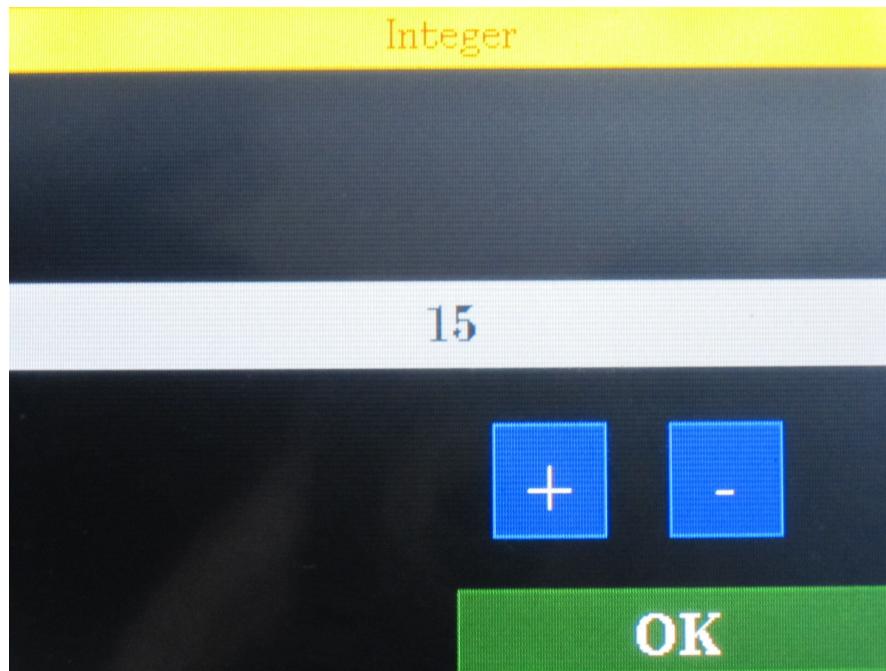


Abbildung 3.5: Editor für Integerwerte (Quelle: Anzinger, Hahn)

3.3.1.3.1.2 FloatInputField

Ein FloatInputField dient zur Manipulation von Gleitkommawerten mit einer Nachkommastelle. Es wird der aktuelle Wert in einem HTML-Eingabefeld angezeigt. Zusätzlich werden zu dem Feld noch Plus und Minus Buttons angezeigt.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

SSI-Tag

```
<!--#FloatInputField id=kurve label=Heizkurve max=2 min=1 increment=0.1 -->
```

Parameter

- *id* → Verbindet das Eingabefeld mit einer Einstellung auf der Maschine, die Einstellung mit *id* wird per Com-Task von der Maschine geladen und angezeigt
- *label* → Name, der beim Eingabefeld angezeigt wird
- *max* → Maximalwert; Gleitkomma
- *min* → Minimalwert; Gleitkomma
- *increment* → Schrittweite, um die der Wert beim drücken der Plus oder Minus Buttons erhöht, bzw erniedrigt wird

HTML-Darstellung



A screenshot of a web browser displaying a form element. The label 'Heizkurve' is followed by a text input field containing '1.7'. To the right of the input field are two small buttons: a '+' sign and a '-' sign, which are typically used for incrementing and decrementing the value in a slider control.

Abbildung 3.6:

Browserdarstellung

Gleitkommawert

Eingabefeld (Quelle:

Anzinger, Hahn)

Touchscreen-Darstellung

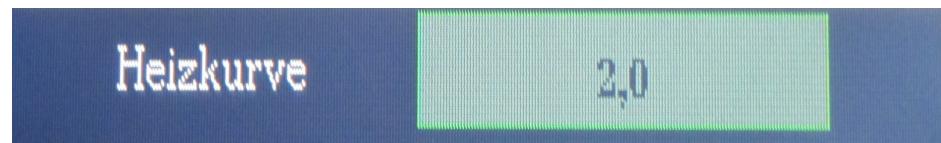


Abbildung 3.7: Eingabefeld für Gleitkommawerte (Quelle: Anzinger, Hahn)

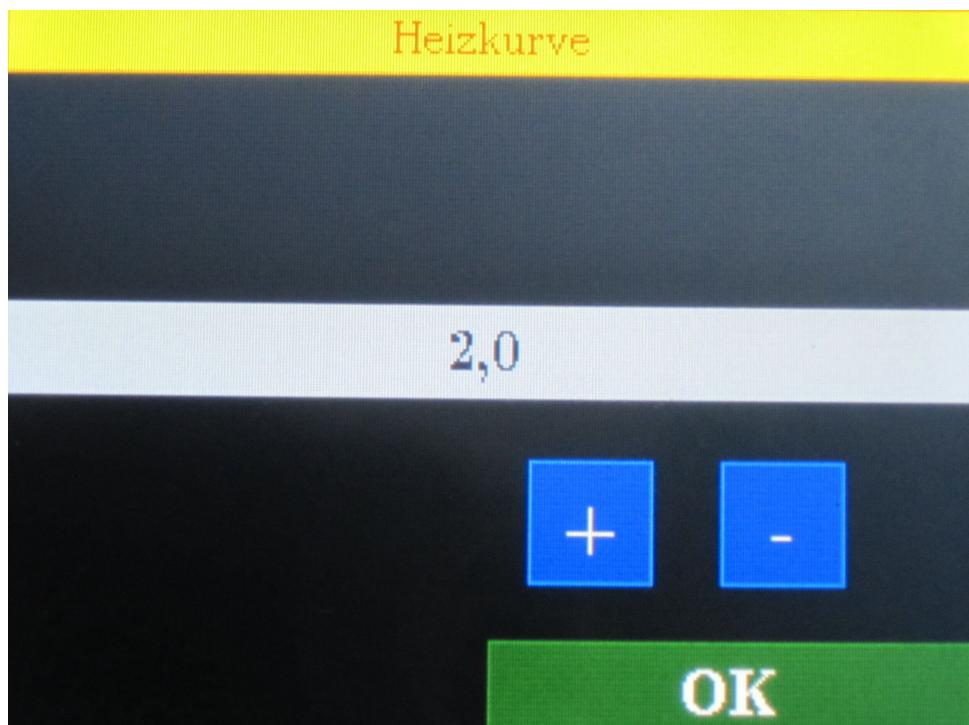


Abbildung 3.8: Editor für Gleitkommawerte (Quelle: Anzinger, Hahn)

3.3.1.3.1.3 SubmitInputField

Ein SubmitInputField dient zum Abspeichern der Werte. Es wird ein HTML-Button angezeigt.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

SSI-Tag

```
<!--#SubmitInputField label=Speichern form_id=demo -->
```

Parameter

- label → Name, der beim Eingabefeld angezeigt wird
- form_id → id des Formulars, das gespeichert werden soll

HTML-Darstellung



*Abbildung 3.9:
Speichern Button
(Quelle: Anzinger,
Hahn)*

Touchscreen-Darstellung



*Abbildung 3.10 Speichernbutton auf dem Touchscreen
(Quelle: Anzinger, Hahn)*

3.3.1.3.1.4 CheckboxInputField

Ein CheckboxInputField dient zum Setzen von boolschen Werten (gesetzt oder nicht gesetzt). Es wird eine HTML-Checkbox angezeigt.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

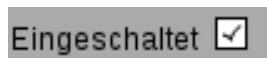
SSI-Tag

```
<!--#CheckboxInputField id=running label=Eingeschaltet -->
```

Parameter

- id → Verbindet das Eingabefeld mit einer Einstellung auf der Maschine, die Einstellung mit *id* wird per Com-Task von der Maschine geladen und angezeigt
- label → Name, der bei der Checkbox angezeigt wird

HTML-Darstellung



*Abbildung 3.11:
Browserdarstellung
Checkbox (Quelle:
Anzinger, Hahn)*

Touchscreen-Darstellung



Abbildung 3.12: Checkbox aktiviert (Quelle: Anzinger, Hahn)



Abbildung 3.13: Checkbox unchecked (Quelle: Anzinger, Hahn)

3.3.1.3.1.5 TimeInputField

Ein TimeInputField dient zur Manipulation eines Zeitwerts. Es werden zwei Eingabefelder (für Stunden und Minuten) angezeigt, dazu Plus- und Minus-Buttons.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

SSI-Tag

```
<!--#TimeInputField id=night label=Ende-->
```

Parameter

- `id` → Verbindet das Eingabefeld mit einer Einstellung auf der Maschine, die Einstellung mit `id` wird per Com-Task von der Maschine geladen und angezeigt
- `label` → Name, der bei den Eingabefeldern angezeigt wird

HTML-Darstellung



Abbildung 3.14: Browserdarstellung
Zeiteingabefeld (Quelle: Anzinger, Hahn)

Touchscreen-Darstellung



Abbildung 3.15: Menüansicht für Uhrzeiten
(Quelle: Anzinger, Hahn)

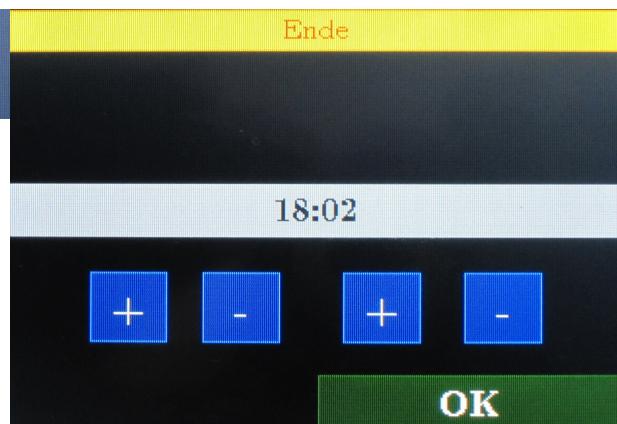


Abbildung 3.16: Editor für Uhrzeiten (Quelle:
Anzinger, Hahn)

3.3.1.3.2 Ausgabe-Tags

Über Ausgabe-Tags können Informationen und Hyperlinks ausgegeben werden.

3.3.1.3.2.1 SavedParams

Gibt die Anzahl der erfolgreich zur Maschine gesendeten Werte aus.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

SSI-Tag

```
<!--#SavedParams -->
```

Parameter

Keine

HTML-Darstellung

2 Parameter gesetzt

Abbildung 3.17:

Browserdarstellung

Statusmeldung (Quelle:

Anzinger, Hahn)

3.3.1.3.2.2 Hyperlink

Ein Hyperlink dient zum Erstellen eines Verweises auf eine andere Bedienseite. Es wird ein HTML-Hyperlink erzeugt.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

SSI-Tag

```
<!--#Hyperlink label=Menue value=/index.ssi-->
```

Parameter

- label → Name des Verweises
- value → URI der Zielseite

HTML-Darstellung



Abbildung 3.18: Link

(Quelle: Anzinger,
Hahn)

Touchscreen-Darstellung

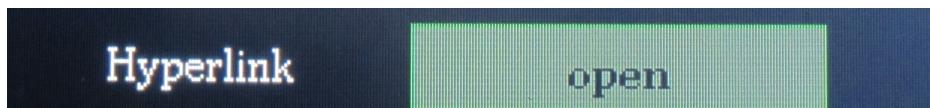


Abbildung 3.19: Hyperlink im Menü (Quelle: Anzinger, Hahn)



Abbildung 3.20: Hyperlink zur Menü-
Seite (Quelle: Anzinger, Hahn)

3.3.1.3.2.3 Titel

Ein Titel dient zur Darstellung der Überschrift einer Bedienseite. Es wird ein HTML-Heading Element erzeugt.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

SSI-Tag

```
<!--#Titel label=Temperaturen -->
```

Parameter

- label → Überschrift

HTML-Darstellung



*Abbildung 3.21: Browserdarstellung
Titel (Quelle: Anzinger, Hahn)*

Touchscreen-Darstellung



Abbildung 3.22: Seitentitel (Quelle: Anzinger, Hahn)

3.3.1.3.2.4 Group

Ein Group-Tag dient zum Gruppieren von anderen SSI-Tags mit Hilfe von Überschriften. Es wird ein HTML-Heading Element erzeugt.

Weiters wird ein entsprechender Kommentar für den Embedded Client erzeugt.

SSI-Tag

```
<!--#Group label=Gruppe -->
```

Parameter

- label → Überschrift

HTML-Darstellung



*Abbildung 3.23:
Browserdarstellung Gruppe
(Quelle: Anzinger, Hahn)*

Touchscreen-Darstellung



Abbildung 3.24: Gruppierung auf dem Touchscreen (Quelle: Anzinger, Hahn)

3.3.2 CGI

Die Kommunikation vom Benutzer zurück zur Maschine, also das Durchreichen von Einstellungen, wird über das **Common Gateway Interface (CGI)**³⁶ realisiert. CGI ist eine schon länger bestehende Variante, Webseiten dynamisch bzw. interaktiv zu machen, deren erste Überlegungen auf das Jahr 1993 zurückgehen.

Der lwIP-Webserver unterstützt CGI in Form von C-Funktionen. Dazu muss eine Funktion mit folgender Signatur implementiert werden, die die übergebenen Parameter auswertet und eine Ergebnisseite liefert.

```
static char * SetCGIHandler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[])
```

Diese Funktion muss mit einer URI verknüpft werden, damit der Webserver CGI Aufrufe erkennt und die entsprechende Funktion aufruft. Das „Mappen“ erfolgt über das Feld `g_psConfigCGIURIs`:

```
static const tCGI g_psConfigCGIURIs[] =
{
    { "/set.cgi", SetCGIHandler }, // CGI_INDEX_CONTROL
};
```

3.3.2.1 Set.cgi

`Set.cgi` ist die CGI-Funktion, die bei LumWeb das Auswerten der Änderungen übernimmt. Über `set.cgi` können Werte auf der Maschine geändert werden, dazu werden die Werte als GET-Parameter an die CGI-Funktion übergeben.

Beispieldaufruf:

http://lumWebServer/set.cgi?id1=1&f_id2=

³⁶ CGI ist ein Standard für den Datenaustausch zwischen einem Webserver und dritter Software, die Anfragen bearbeitet

Unterstützte Datentypen

- **Integer**

Ganze Zahlen werden als *id=Zahl* übergeben und werden in Integer umgewandelt, bevor sie an den Com-Task geschickt werden.

Beispiel: Wert von *id1* ist 1 → *id1=1*

- **Float (eine Nachkommastelle)**

Gleitkommazahlen werden als *f_id=Vorkommastelle.Nachkommastelle* übergeben. Mit der CGI-Funktion wird der String in eine ganze Zahl umgewandelt, bevor der Wert an den Com-Task gesendet wird. Dazu wird der String in Vorkomma- und Nachkommastelle zerlegt, die Vorkommastelle mit 10 multipliziert und die Nachkommastelle addiert.

Beispiel: Wert von *id1* ist 1.1 → *f_id1=1.1*, an den Com-Task wird 11 gesendet.

Grund dafür ist die zu komplexe Übertragung von Gleitkommazahlen über die Bus-Systeme. Gleitkommazahlen verbrauchen außerdem mehr Speicher als ganzzahlige Werte

- **Zeit (Stunden:Minuten)**

Zeiten werden als *t_id1=stunden&t_id1=minuten* übertragen. Mit der CGI-Funktion werden Stunden und Minuten als Minuten zusammengerechnet (Stunden * 60 + Minuten), bevor der Wert an den Com-Task gesendet wird

Beispiel: Wert von *id1* ist 10:02 → *t_id1=10&t_id1=2*, an den Com-Task wird 602 gesendet.

- **Boolean (Checkboxen)**

Gesetzte Checkboxen werden als *id=on* übertragen. Mit der CGI-Funktion wird eine gesetzte Checkbox in den Integerwert 1 umgewandelt bevor der Wert an den Com-Task gesendet wird.

Beispiel: gesetzte Checkbox *id1* → *id1=on*, an den Com-Task wird 1 gesendet

3.3.3 Ablauf

3.3.3.1 Seitenaufruf

Client beantragt SSI Seite vom Server:

1. Benutzer stellt Anfrage an den HTTP Server
2. Datei von SD Karte einlesen
3. Datei nach SSI-Tags und Parameter durchsuchen
4. SSI-Tag Funktion am Server aufrufen
5. aktuelle Einstellungswerte vom Com-Task anfordern
6. SSI-Tag durch die aktuellen Werte ersetzen
7. fertige HTML Seite an Client senden

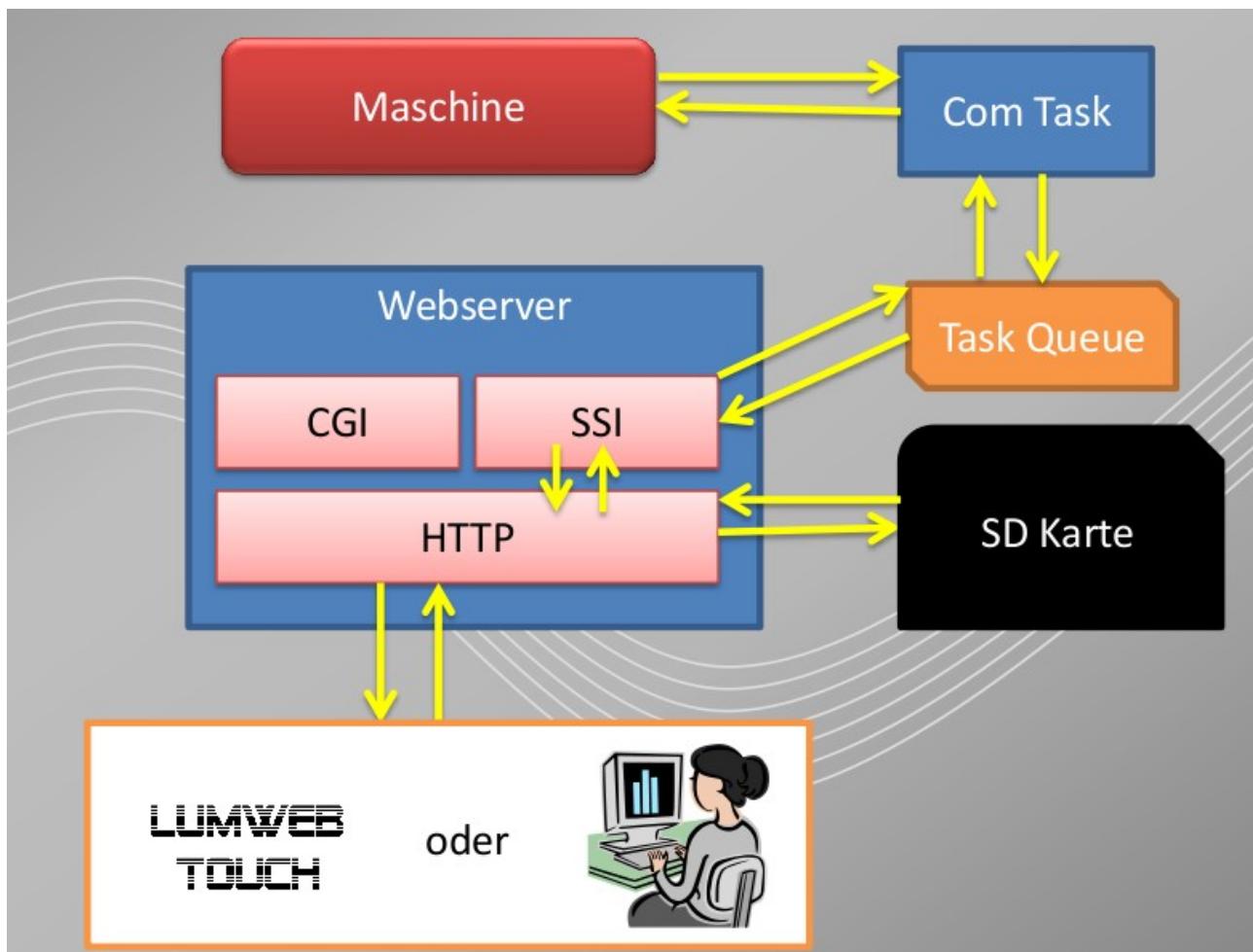


Abbildung 3.25: Ablauf Seitenaufbau mit Werte laden (Quelle: Anzinger, Hahn)

3.3.3.2 Speichern von Werten

Client speichert Werte per Set.cgi -Client sendet GET Anfrage an set.cgi mit den zu speichernden Werten als GET Parameter

1. Benutzer stellt Anfrage an HTTP Server
2. Parameterstring wird von der CGI-Funktion verarbeitet und überprüft ob gültige Werte übergeben wurden (nur Nummern gelten als gültige Parameter)
3. Wert per Com-Task setzen
4. Wert per Com-Task zurücklesen (Kontrolle ob die Werte erfolgreich auf der Maschine gesetzt wurden)
5. Ergebnisseite zurücksenden wenn alle Parameter abgearbeitet sind

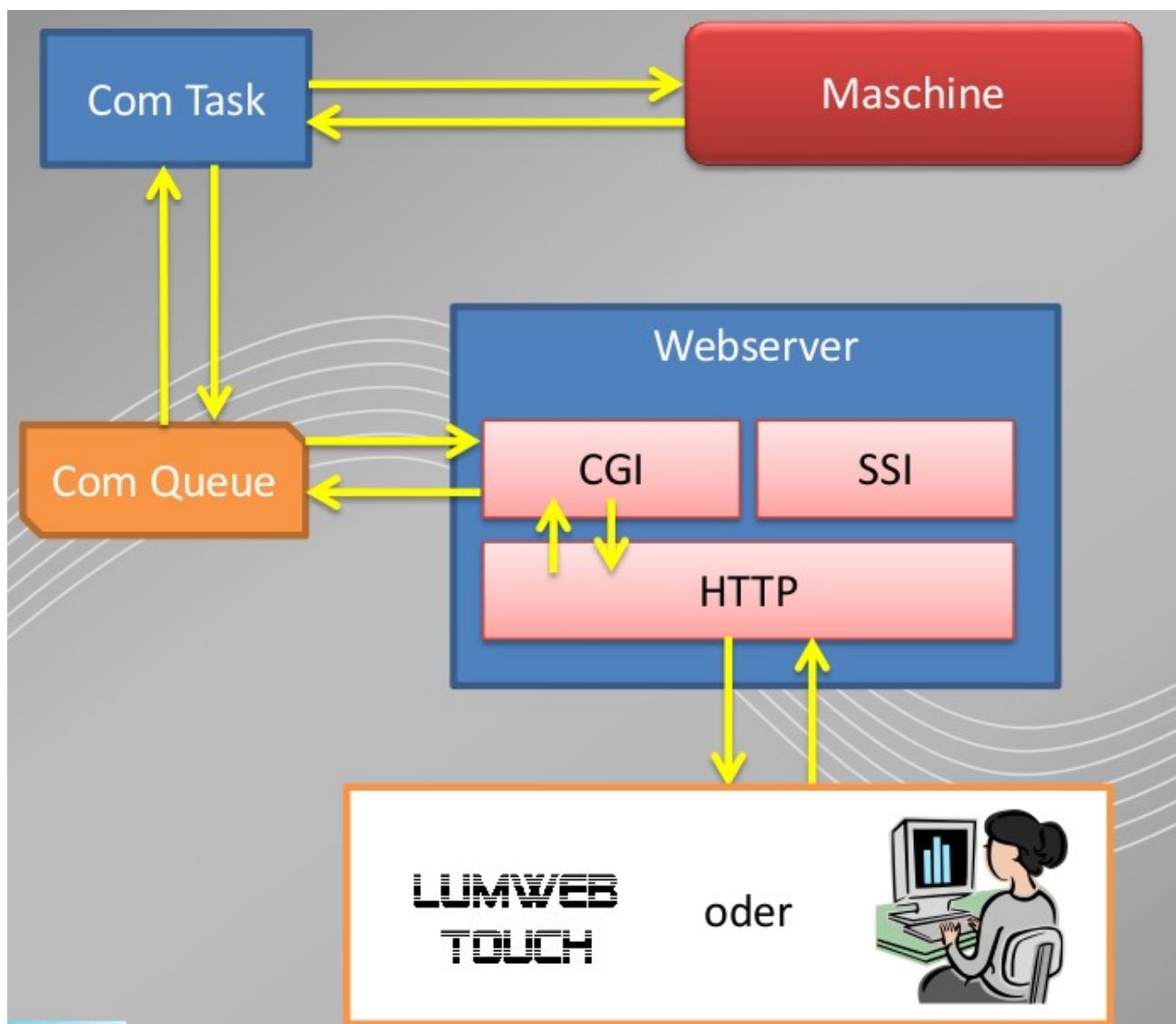


Abbildung 3.26: Ablauf Seiteninhalt zurücksenden an die Maschine (Quelle: Anzinger, Hahn)

3.3.4 Erstellen einer Bedienseite

Das Layout der Seite im Browser wird in Form von Standard-HTML erzeugt. In dieses HTML-Dokument werden dann SSI-Tags zum Bearbeiten von Werten eingebettet. Die Bedienseite wird einmal definiert und kann vom Browser und vom Embedded-Client ausgewertet und dargestellt werden.

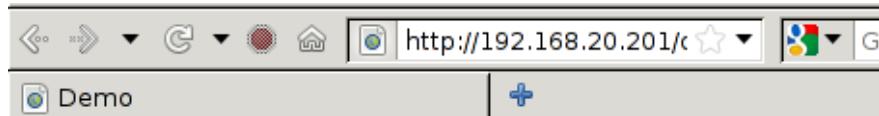
Diese Seiten müssen mit der Endung `.ssi` abgespeichert werden und im `httpd-fs` Ordner auf der SD Karte abgelegt werden.

Folgende Bedienseite ermöglicht das Verändern eines Float-Wertes (in diesem Beispiel die Steigung einer Heizkurve) mit der Id `kurve` in dem Bereich von 1 bis 2. Zu der Eingabebox wird der Name `Heizkurve` ausgegeben. Erhöht wird der Wert um 0.1.

```
<html>
  <head>
    <title>Demo</title>
    <script type="text/javascript" language="javascript"
src="js/funcs_c.js"></script>
  </head>
  <body>
    <div id="container">
      <center>
        <form method="get" action="set.cgi" name="demo" id="form-demo"
onsubmit="return submit_form(this)">
          <!--#Titel label=Demo --><br/>

          <!--#FloatInputField id=kurve name=Heizkurve min=1 max=2
increment=0.1 --><br/>

          <!--#SubmitInputField label=Speichern form_id=demo -->
        </form>
      </div>
    </body>
</html>
```

Darstellung im Web-Browser

Demo

Heizkurve + -

Abbildung 3.27: Browserdarstellung einer Demoseite (Quelle: Anzinger, Hahn)

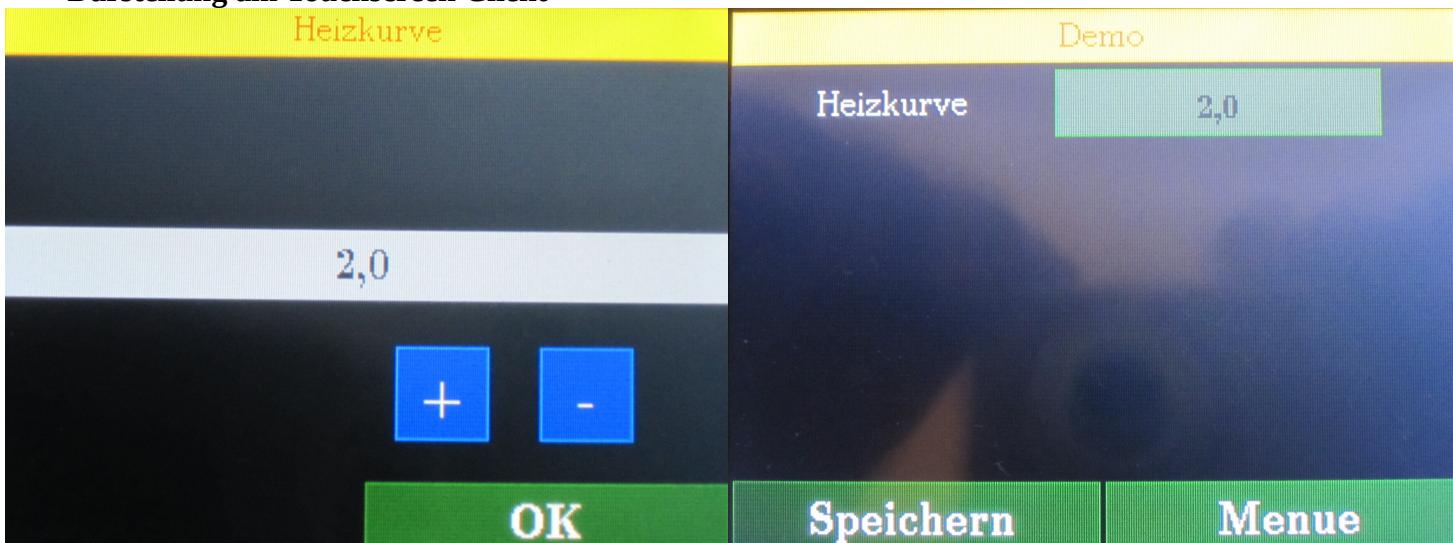
Darstellung am Touchscreen Client

Abbildung 3.29: Demoseite am Touchscreen
(Editoransicht) (Quelle: Anzinger, Hahn)

Abbildung 3.28: Demoseite am Touchscreen
(Listenansicht) (Quelle: Anzinger, Hahn)

3.4 Webclient

3.4.1 Interface

Das Interface des Webclients ist eine auf der Basis von Stellaris Graphic Library³⁷ erstellte Oberfläche zur Anzeige und Manipulation der Daten. Als Quelle für die Daten werden modifizierte HTML-Kommentare, hier im Dokument „Graphic Tags“ genannt, verwendet. Diese Kommentare werden durch das Parsen der SSI-Tags am Webserver erzeugt und in das HTML-Dokument eingebettet.

Der Aufbau bei den Graphic-Tags ist wie bei den SSI-Tags. Der Unterschied besteht nur darin, dass die Graphic Tags den Wert beinhalten, welcher zur Darstellung benötigt werden.

Der Aufbau eines Graphic Tags ist wie folgt:

```
<! - - ? befehl parametername1="wert" value="wert" - ->
```

Für das Touch-Interface stehen also alle Tags zur Verfügung die für den Webserver vorhanden sind. Diese Tags müssen für eine ordnungsgemäße Funktion in der uInterface/taglib/tags.c Datei registriert werden. (siehe [3.3.3.Erstellen eines Tags](#))

3.4.1.1 Livecycle der Grafik Objekte

Da die grafischen Objekte der Webseite dynamisch aufgebaut und in den Speicher geladen werden, gibt es auch einen speziellen Lifecycle. Dieser Lifecycle beginnt mit dem Laden der Menü-Seite vom Webserver und dem Speichern der darin enthaltenen Objekte in einer Liste (taglib-Funktion: **onLoad**). Ist die Seite fertig geladen werden die ersten DISPLAY_LINES_PER_VIEW (5) Objekte angezeigt. Zur Darstellung wird die taglib-Funktion **onDisplay** aufgerufen.

Gibt es mehr als DISPLAY_LINES_PER_VIEW Objekte, so erscheinen auf der Seite des Touchinterfaces die Buttons mit der Aufschrift „Up“ und „Down“. Mit diesen Buttons kann der Offset der Objekte verändert werden, und somit auch Objekte, welche weiter hinten in der Liste gespeichert sind angezeigt werden. Bei jedem Scroll-Vorgang wird die jeweilige **onRender** Funktion des entsprechenden Objekts aufgerufen.

DISPLAY_LINES_PER_VIEW ist als Definition in der Datei uInterface/graphic/gui/displayStyle.h standartmäßig mit 5 festgelegt, da dies für den verwendeten Display ein idealer Wert ist.

Bei diesem Scrollen wird auch der Speicher nicht dargestellter Objekte freigegeben. Dabei ist zu

³⁷ Luminary Graphic Library: http://www.luminarmicro.com/products/stellaris_graphics_library.html

beachten, dass nur der Speicher der GUI freigegeben wird, nicht der Speicher des Objektes mit den Attributen.

Beim Auftreten einer Aktion wird die im Tag hinterlegte Funktion **onValueEdit** aufgerufen. Diese muss jedoch selbst in der **onDisplay** Funktion festgelegt werden.

Wird ein Hyperlink-Tag auf dem Touchinterface gewählt, so wird das Laden einer neuen Seite angestoßen. Dabei werden dann die Objekte vollständig gelöscht, alle Attribute in der Liste der Objekte werden automatisch freigegeben. Wenn zusätzliche Variablen allokiert worden sind, so müssen diese mit der taglib-Funktion **onDelete** manuell verwaltet werden. **OnDelete** wird automatisch beim Löschen des Objektes aufgerufen.

Wird jedoch ein Objekt wie „IntegerField“ gewählt, so wird der entsprechende Editor zum Verarbeiten der Werte aufgerufen. Dabei werden aber die Objekte nicht freigegeben.

Nach dem Aufruf von **onDelete** ist der Lifecycle des Objektes beendet.

Achtung: Die Variable **userSpace** ist eine Globale Variable für alle Objekte eines Taglib-Typen. Daher wird bei einer Änderung der Variable diese Änderung für alle Objekte des selben TAG-Typs übernommen. Sie ist für die Speicherung zusätzlicher Attribute gedacht.

3.5 Com-Task

Der Com-Task ist für die Kommunikation zwischen Maschine und Webserver zuständig. Dazu stellt er ein Softwareinterface zur Verfügung. Je nach Maschine und Verbindungsart (CAN, RS232) muss der Com-Task modifiziert werden. Da zum Zeitpunkt der Diplomarbeit noch kein Kommunikationsprotokoll zur Verfügung stand, speichert der Com-Task die Werte in Dateien auf der SD-Karte anstatt mit einer Maschine zu kommunizieren. Diese Konfiguration dient aber ausschließlich Testzwecken.

3.5.1.1 Schnittstelle für den Com-Task

Tasks kommunizieren mit dem Com-Task über die Queue „*xComQueue*“. Dieser gibt die Antwort über *xHttpdQueue* an den Task zurück. Die Queues verwenden die Struktur *xComMessage* zum Nachrichtenaustausch. Nachdem ein Task eine Anfrage an Com-Task geschickt hat, muss dieser Task pausiert werden (suspend). *xComMessage* muss einen TaskHandler (*taskToResume*) enthalten, der auf den aktuellen Task (den Anfragenden) zeigt. Der Com-Task verwendet diesen TaskHandler um den anfragenden Task wieder aufzuwecken, nachdem die Antwort in die *xHttpdQueue* enqueue wurde.

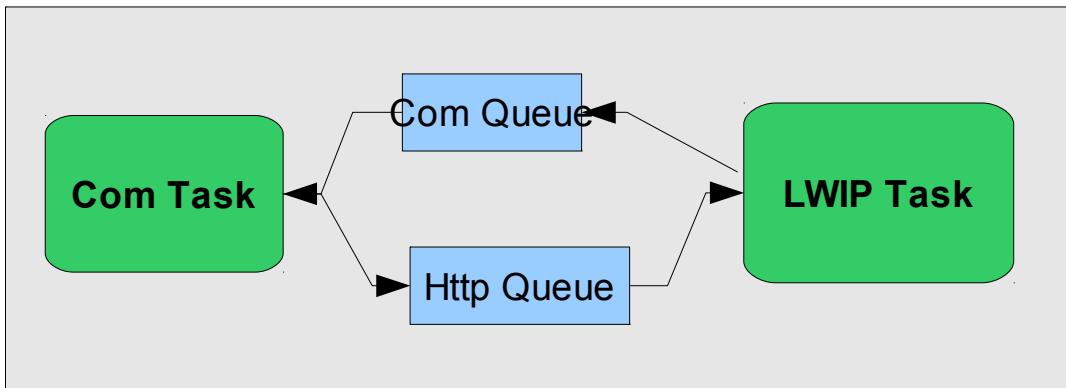


Abbildung 3.30: Com-Task Zugriff (Quelle: Anzinger, Hahn)

3.5.1.1.1 xComMessage

xComMessage ist die Nachrichtenstruktur für die Kommunikation zwischen Com-Task und lwIP Task.

```
typedef struct
{
    enum com_commands cmd; //e.g. 'get', 'set'
    enum com_dataSource dataSource; // e.g. 'conf', 'data'
    char *item; // name of the selected item
    int value; // value if a Item is set
    char *errorDesc; // if not null, an error has occurred
    tBoolean freeItem; // if true, free item in Com-Task
    xQueueHandle from; // address to return answer (name of the Queue)
    xTaskHandle taskToResume; // If not null the specific task will be resumed
} xComMessage
```

Felder:

- *cmd* – Typ der Operation, entweder Wert setzen oder Wert auslesen
- *dataSource* – Datenquelle, zum Beispiel Konfigurationsdatei
- *item* – ID des Elements, das ausgelesen bzw verändert werden soll
- *value* – Wert des Elements
- *errorDesc* – Fehlercode, im Fehlerfall ist dieses Feld ungleich NULL
- *freeItem* – gibt an, ob der Speicher des Elements im Com-Task freigegeben werden muss
(True → Speicher des Elements wird im Com-Task freigegeben)
- *from* – gibt an die Queue an, in die die Antwort zurückgegeben werden muss (Name der Queue)
- *taskToResume* – gibt an, welcher Task vom Com-Task aufgeweckt werden muss

3.5.1.2 Beispiele

Anfrage vom Com-Task an den lwIP-Task:

```
xComMessage xCom_msg; // xComMessage Variable deklarieren
// Einstellungen treffen
```

```

xCom_msg.cmd = GET; // Element auslesen
xCom_msg.dataSouce = DATA;
xCom_msg.from = xHttpdQueue;
xCom_msg.taskToResume = xLwipTaskHandle;
xCom_msg.freeItem = pdFALSE;
xCom_msg.item = id; // id des Elements angeben

// Nachricht in xComQueue stellen
xQueueSend(xComQueue, &xCom_msg, (portTickType) 0);

```

3.5.1.3 Kommunikation mit der Maschine

Der Com-Task stellt zwei Funktionen zur Verfügung, in denen der Zugriff auf die spezielle Maschine implementiert werden kann, zum Beispiel über CAN-Bus³⁸s oder RS232.

Folgende Funktionen müssen implementiert werden:

- `int sendToMachine(char* id, int value)`
- `int getFromMachine(char* id)`

3.5.1.3.1 sendToMachine

Mit dieser Funktion wird ein Wert an die Maschine gesendet. Dazu wird die ID der Einstellung und der Wert der Einstellung übergeben. Diese Ids werden in den SSI Dateien (Interface) definiert. Die ID muss gegebenenfalls noch in eine für die Maschine verarbeitbare Form umgewandelt werden.

Der Rückgabewert ist:

- 0 → Wert erfolgreich gesendet
- ungleich 0 → Fehler beim Senden

3.5.1.3.2 getFromMachine

Diese Funktion lest eine Option von der Maschine und gibt den Wert zurück. Dazu wird der Funktion die ID der Option übergeben. Diese Ids werden in den SSI Dateien (Interface) definiert.

³⁸ Der CAN-Bus ist ein asynchrones, serielles Bussystem und gehört zu den Feldbussen

3.6 Debugging und Logging

LumWeb bietet die Möglichkeit Statusmeldungen über die UART Schnittstelle oder in eine Log-Datei auf der SD-Karte auszugeben. Dazu stellt LumWeb zwei Funktionen zur Verfügung:

3.6.1 UARTPrintf

Diese Funktion dient zur Ausgabe von Statusmeldungen über die UART Schnittstelle.

Sie kann wie die *Printf*³⁹ Funktion der C-Standardbibliothek (*stdio.h*) verwendet werden, d.h. *UARTPrintf* akzeptiert die selben Parameter. *UARTPrintf* implementiert also die *Printf* Funktion, die Aufrufe von *printf* werden vom Compiler über einen Befehl im Makefile in Aufrufe der Funktion *UARTPrintf* umgewandelt. Der Prototyp der Funktion befindet sich in der Datei *uInterface/uart/uartstdio.h*.

Prototyp

```
int UARTprintf(const char *pcString, ...)
```

Beispieldaufruf

```
printf("SetCGIHandler: Found Checkbox %s\n", name);
```

Gibt den String „SetCGIHandler: Found Checkbox Test \n“ auf die UART Schnittstelle aus (falls die Variable *name* den String „Test“ beinhaltet.

3.6.2 AppendToLog

Diese Funktion dient zur Ausgabe von Statusmeldungen in eine Log-Datei (*/log/sys.log*) auf der SD-Karte. Die Meldung wird am Ende der Datei hinzugefügt, d.h. die letzten Meldungen befinden sich am Ende der Datei. Vor dem ersten Aufruf von *AppendToLog* muss das Logging-System zuerst initialisiert werden. Dies geschieht mit dem Aufruf von *initLog()*. Der Prototyp der Funktion befindet sich in der Datei *uInterface/log/logging.h*.

AppendToLog akzeptiert als Parameter einen einzelnen String. Dieser wird dann an die Datei angefügt.

39 <http://www.cplusplus.com/reference/clibrary/cstdio/printf/>

Die Funktion liefert einen *FRESULT* Wert zurück:

```
typedef enum {
    FR_OK = 0,                  /* 0 */
    FR_DISK_ERR,                /* 1 */
    FR_INT_ERR,                 /* 2 */
    FR_NOT_READY,               /* 3 */
    FR_NO_FILE,                 /* 4 */
    FR_NO_PATH,                 /* 5 */
    FR_INVALID_NAME,             /* 6 */
    FR_DENIED,                  /* 7 */
    FR_EXIST,                   /* 8 */
    FR_INVALID_OBJECT,           /* 9 */
    FR_WRITE_PROTECTED,          /* 10 */
    FR_INVALID_DRIVE,             /* 11 */
    FR_NOT_ENABLED,               /* 12 */
    FR_NO_FILESYSTEM,              /* 13 */
    FR_MKFS_ABORTED,              /* 14 */
    FR_TIMEOUT                  /* 15 */
} FRESULT;
```

Prototyp

```
FRESULT appendToLog(char *msg);
```

Beispielaufruf

```
appendToLog("Starting Firmware");
```

Schreibt den String „Starting Firmware“ in die Datei */log/sys.log*.

4 Abbildungsverzeichnis

Abbildung 1.1: Blockschaltbild Stellaris Micro Controller (Quelle: Luminary Micro http://www.luminarmicro.com/images/stories/stellarisblockdiagram.gif).....	12
Abbildung 1.2: Luminary Developmentkit (LM3S9B96) (Quelle: Anzinger, Hahn).....	13
Abbildung 3.1: Schematischer Aufbau (Quelle: Anzinger, Hahn).....	37
Abbildung 3.2: Intertaskkommunikation (Quelle: Anzinger, Hahn).....	38
Abbildung 3.3: Browserdarstellung Integer Eingabefeld (Quelle: Anzinger, Hahn).....	47
Abbildung 3.4: Menüdarstellung Integerwert (Quelle: Anzinger, Hahn).....	48
Abbildung 3.5: Editor für Integerwerte (Quelle: Anzinger, Hahn).....	48
Abbildung 3.6: Browserdarstellung Gleitkommawert Eingabefeld (Quelle: Anzinger, Hahn).....	49
Abbildung 3.7: Eingabefeld für Gleitkommawerte (Quelle: Anzinger, Hahn).....	50
Abbildung 3.8: Editor für Gleitkommawerte (Quelle: Anzinger, Hahn).....	50
Abbildung 3.9: Speichern Button (Quelle: Anzinger, Hahn).....	51
Abbildung 3.10 Speichernbutton auf dem Touchscreen (Quelle: Anzinger, Hahn).....	51
Abbildung 3.11: Browserdarstellung Checkbox (Quelle: Anzinger, Hahn).....	52
Abbildung 3.12: Checkbox aktiviert (Quelle: Anzinger, Hahn).....	52
Abbildung 3.13: Checkbox unchecked (Quelle: Anzinger, Hahn).....	52
Abbildung 3.14: Browserdarstellung Zeiteingabefeld (Quelle: Anzinger, Hahn).....	53
Abbildung 3.15: Menüansicht für Uhrzeiten (Quelle: Anzinger, Hahn).....	53
Abbildung 3.16: Editor für Uhrzeiten (Quelle: Anzinger, Hahn).....	53
Abbildung 3.17: Browserdarstellung Statusmeldung (Quelle: Anzinger, Hahn).....	54
Abbildung 3.18: Link (Quelle: Anzinger, Hahn).....	55
Abbildung 3.19: Hyperlink im Menü (Quelle: Anzinger, Hahn).....	55
Abbildung 3.20: Hyperlink zur Menü-Seite (Quelle: Anzinger, Hahn).....	55
Abbildung 3.21: Browserdarstellung Titel (Quelle: Anzinger, Hahn).....	56
Abbildung 3.22: Seitentitel (Quelle: Anzinger, Hahn).....	56
Abbildung 3.23: Browserdarstellung Gruppe (Quelle: Anzinger, Hahn).....	57
Abbildung 3.24: Gruppierung auf dem Touchscreen (Quelle: Anzinger, Hahn).....	57
Abbildung 3.25: Ablauf Seitenaufbau mit Werte laden (Quelle: Anzinger, Hahn).....	60
Abbildung 3.26: Ablauf Seiteninhalt zurücksenden an die Maschine (Quelle: Anzinger, Hahn).....	61
Abbildung 3.27: Browserdarstellung einer Demoseite (Quelle: Anzinger, Hahn).....	63
Abbildung 3.28: Demoseite am Touchscreen (Listenansicht) (Quelle: Anzinger, Hahn).....	63

Abbildung 3.29: Demoseite am Touchscreen (Editoransicht) (Quelle: Anzinger, Hahn).....	63
Abbildung 3.30: Com-Task Zugriff (Quelle: Anzinger, Hahn).....	66

5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

24 Anhang

- Anhang A: Stellaris® Peripheral Driver Library – User Guide
- Code-Dokumentation (Doxygen)