DIPLOMARBEIT

Realisierung eines intelligenten Kühlschranks mit Ethernet-Anbindung

EtherFridge

ausgeführt zur Erlangung des akademischen Grades
eines Diplom-Ingenieurs unter der Leitung von

O. Univ. Prof. Dipl.-Ing. Dr.techn. Dietmar Dietrich
Ass. Prof. Dipl.-Ing. Dr.techn. Thilo Sauter
Mag. Dipl.-Ing. Dr. Georg Gaderer

am

Institut für Computertechnik (E384)
der Technischen Universität Wien

durch

Thomas Bigler
Matr.Nr. 0025893
Pöckgasse 2, 2700 Wr. Neustadt

Wiener Neustadt, März 2011 _____

Kurzfassung

Die Vernetzung von Computern über das Ethernet ist heutzutage eine alltägliche und kostengünstige Technologie, die inzwischen auch in der Heimvernetzung dedizierte Bussystem ersetzen kann. In Hinblick auf neue Konzepte, wie etwa dem intelligenten Stromnetz, wird es in Zukunft notwendig sein, immer mehr Geräten einzubinden. Diese Diplomarbeit soll nun zeigen, welche Komponenten und Technologien notwendig sind, um einen handelsüblichen Kühlschrank in einen „smarten“, an das Netzwerk angebundenen Külschrank zu verwandeln. Durch diese Netzwerkanbindung können die aktuellen Daten über das Internet von jedem Ort abgefragt, aber auch Konfigurationen vorgenommen werden, um beispielsweise ein verbessertes Regelverhalten zu erreichen. Die vorhandene Hardware wird dabei durch eine Eigenentwicklung ersetzt, wobei die Kühlschrankregelung und die Netzwerkanbindung über zwei getrennte Platinen realisiert werden, um die elementaren Funktionen von den höheren Funktionen abzukoppeln. Als Frontend zum Zugriff auf die Kühl-Gefrierkombination dient eine Webseite, die über das Netzwermodul zur Verfügung gestellt wird. Durch den Einsatz der oft mit dem Schlagwort „*Web 2.0*“ bezeichneten Technologien kann eine komfortable Schnittstelle zwischen der Browser-Software und der Hardware der Kühl-Gefrierkombination erreicht werden.

Abstract

Building computer networks over Ethernet nowadays is a common and cheap technology that also has the capabilities to replace dedicated fieldbuses in the field of home-networks. In respect of new concepts, for instance „smart grids“, it will be neccessary in the future to integrate even more household devices into the network. This diploma thesis has the objective to show which components and technologies are essential to convert an off-the-shelf refrigerator into an intelligent, networked „smart fridge“ that is integrated into the Ethernet home-network. Because of the network connection, it is possible to retrieve the current status of the refrigerator from anywhere over the internet. Furthermore, the parameters can also be changed, for instance to adjust the temperature control algorithm. The existing hardware of the off-the-shelf refrigerator is replaced by an proprietary development, where the refrigerator control and the network interface are split into to separate hardware boards. As a result, the basic functionality and the higher level functionality are separated, so that, for example, a faulty webserver has no impact on the behaviour of the refrigerator. To visualize the information and to allow the access to the refrigerator a website is provided. Through the usage of the so-called „Web 2.0“ technologies it is possible to offer a comfortable interface between the browser and the SmartFridge implementation.

Inhaltsverzeichnis

Abkürzungen

ADC	Analog-Digital Converter
API	Application Programming Interface
DAC	Digital-Analog Converter
DSP	Digital Signal Processor
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIFO	First-In First-Out
FRAM	Ferroelectric Random Access Memory
GUI	Graphical User Interface
HTML	Hyper-Text Markup Language
ISR	Interrupt Service Routine
LED	Light Emitting Diode
LSB	Least Significant Bit
NTC	Negative Temperature Coefficient
PoE	Power over Ethernet
PLC	Power Line Communication
PLL	Phase Locked Loop
RAM	Random Access Memory
RMS	Root Mean Square
RTOS	Real-Time Operating System
SQL	Structured Query Language

1 Einleitung

Die Vernetzung von Geräten erreicht in zunehmendem Maße auch die Haushalte, wo durch die Ethernet-Technologie zur Zeit vor allem die Verteilung von Multimediadaten günstig realisiert werden kann. Es existieren zwar auch Konzepte und Realisierungen für die Einbindung von Haushaltsgeräten wie dem Kühlschrank, diese bieten dem Nutzer aber oft keinen nennenswerten Mehrwert gegenüber einer Variante ohne Vernetzung. Die aktuelle Entwicklung in diesem Gebiet geht unter anderem in Richtung der intelligenten Stromnetze (*Smart Grid*) und im speziellen dem *Smart Metering*, wodurch der Stromanbieter die Möglichkeit bekommt, die Leistungsdaten einzelner Haushalte zu erfassen und gegebenenfalls auf Verbraucher einzuwirken. Ziel ist dabei eine Reduzierung der Spitzenlasten und somit die bessere Ausnutzung des Stromnetzes.

Das grundsätzliche Ziel bei dieser Diplomarbeit ist es nun, für eine handelsübliche Kühl-Gefrierkombination die notwendige Hard- und Software zur Einbindung in das Ethernet zu realisieren, wobei sich diese wie ein handelsübliches Gerät verhalten und dem Benutzer die zusätzlichen Features eines „*Smart Fridge*“ zur Verfügung stellen soll. Die erfassten Verbrauchsdaten sollen im Netzwerk abrufbar sein, wobei es im Hinblick auf die jederzeit mögliche Datenauslesung auch sinnvoll ist, bestimmte Ereignisse und Verläufe (etwa der Temperatur in den verschiedenen Kühlzonen) über einen längeren Zeitraum zu protokollieren und so für spätere Auswertungen verfügbar zu machen. Besonders interessant ist in diesem Zusammenhang auch die Alarmierung per E-Mail oder SMS wenn ein bestimmter Zustand eingetreten ist, oder ein Fehler erkannt wurde. Über ein Webinterface kann jederzeit der aktuelle Status abgefragt, Regelvorgaben getroffen oder auch die Firmware aktualisiert werden.

Ausgangspunkt für diese Arbeit ist dabei die von Georg Gaderer im Jahr 2002 durchgeführte Diplomarbeit [Gad02], bei der eine Vernetzung über den Europäischen Installationsbus (*EIB*) durchgeführt wurde.

1.1 Struktur der Arbeit

In Kapitel 1 wird zunächst auf die grundsätzliche Motivation für diese Diplomarbeit eingegangen und weiters ein Überblick über den aktuellen Stand der Technik gegeben.

Danach erfolgt in Kapitel 2 die Darstellung des Ausgangszustands und die Erarbeitung einer detaillierten Aufgabenstellung nach dem Top-Down Prinzip, was zu einer Auftrennung in drei Teilaufgaben (SmartFridge, SmartPanel und Webinterface) führt.

In Kapitel 3 wird zunächst ein Überblick über die Funktionsweise eines Kühlschrankes gegeben und anschließend die in Hardware und Software realisierten Funktionsmodule für die Kühlschranksteuerung (SmartFridge) behandelt.

Kapitel 4 erläutert zunächst die Aufgaben des SmartPanels und beschreibt dann die realisierte Hardware, die Linux Treiber und die Applikationssoftware.

In Kapitel 5 werden zunächst kurz einige Grundlagen zur Erstellung von Webseiten erläutert. Anschließend wird die konkrete Implementierung beschrieben und gezeigt, wie SmartFridge, SmartPanel und das Webinterface zusammenspielen.

Im abschließenden Kapitel 6 erfolgt zuerst eine Zusammenfassung der durchgeführten Arbeit und schlussendlich auch ein Ausblick auf mögliche Verbesserungen und Erweiterungen.

1.2 Motivation

Die Motivation für ein Redesign und somit auch für diese Diplomarbeit resultiert aus den aktuellen Entwicklungen im Bereich der Heimautomatisierung. Die Ziele bei der Einbindung von Haushaltsgeräten in ein Netzwerk sind unter anderem im Kontext des Energiemanagements und den verbesserten Servicemöglichkeiten zu suchen, wobei in den meisten Fällen aber wohl die Komfortsteigerung im Vordergrund steht [Sch03]. Die Richtung geht dabei nicht nur zur Einbindung einzelner Geräte, sondern zur kompletten Vernetzung eines ganzen Haushaltes.

Viele der aktuell verfügbaren Fallstudien sind eher als Spielereien und ausloten des technisch Machbaren anzusehen [EGS03]. So können sich zurzeit wohl die wenigsten Menschen einen wirklichen Nutzen darin vorstellen, dass der Kühlschrank automatisch Waren über das Internet ordert. Sinnvoller sind hier schon die Möglichkeiten der Steuerung des Kühlschrankes über das Internet und dem Senden eines Berichts an eine zuständige Servicestelle bei Auftreten eines Fehlers [EGS03]. Dies bedeutet nicht nur Vorteile für den Nutzer, sondern auch der Hersteller kann die erfassten Daten zur Analyse und im weiteren Sinne auch zur Verbesserung seiner Produkte verwenden. Bezieht man nicht nur die Verbrauchsdaten einzelner Geräte, sondern ganzer Versorgungsgebiete in die Analyse mit ein, kann der Stromversorger der Steuerung Vorschläge zu günstigen Zeitpunkten zum Abkühlen unterbreiten. Auf diese Weise können Spitzenlasten reduziert und Kosten gesenkt werden – dies wird als *Demand Side Management* bezeichnet.

Bei der zugrunde liegenden Diplomarbeit [Gad02] wurde zur Einbindung der Kühl-Gefrierkombination auf den *Europäischen Installationsbus* – kurz *EIB* – gesetzt. Dieser ist inzwischen mit den Technologien von Batibus und EHS zum KNX-Standard verschmolzen [DPKD06], wobei dieser Standard aber immer noch kompatibel zu EIB ist. Obwohl KNX über Twisted-Pair Kabel im Bereich der Gebäudeautomatisierung eine große Rolle spielt, steht über Ethernet eine weitaus günstigere Möglichkeit für die Realisierung sog. *Smart Homes* zur Verfügung.

Da im Haushalt immer mehr Geräte eine Netzwerkanbindung benötigen (hauptsächlich um den Zugriff auf das Internet zu verteilen), hat die Vernetzung auch im privaten Bereich stark zugenommen. Durch diese hohe Verbreitung kann die notwendige Hardware zur Anbindung sehr günstig hergestellt und vertrieben werden. Zudem existieren durch die sog. *Web 2.0 Technologien* komfortable Möglichkeiten zur Präsentation der Daten und der Steuerung der Kühl-Gefrierkombination, weshalb es nur ein logischer Schritt ist, eine Einbindung in das lokale Netzwerk durchzuführen. Im Gegensatz zur Vernetzung über EIB wird hier auch kein Gateway mehr benötigt um die Kühl-Gefrierkombination über das Internet erreichbar zu machen.

1.3 Stand der Technik

Bevor nun in den nächsten Kapiteln konkret auf die Realisierung eingegangen wird, soll hier zunächst ein Überblick über den aktuellen Stand der Technik gegeben werden. Dabei kann natürlich im Rahmen dieser Übersicht nur ein kleiner Ausschnitt der aktuellen Entwicklung gegeben werden.

Grundsätzlich impliziert der Begriff „Smart“ bei Produkten eine gewisse Art von Intelligenz und wird heutzutage oft als Schlagwort verwendet – sei es *Smart Metering* bei der Messdatenerfassung von Stromzählern [91] oder auch das *Smart Grid* als Überbegriff für ein vernetztes Stromnetz. Beide Begriffe werden in diesem Abschnitt noch näher erläutert werden. Eines haben aber die meisten dieser *smarten* Produkte gemein: sie sind vernetzt [VD10].

Auch beim KNX-Standard hat man die Zeichen der Zeit erkannt und so steht neben der Möglichkeit der Vernetzung über das Stromnetz (Powerline Communication - PLC) inzwischen mit KNXnet/IP eine Variante zur Verfügung, die das IP-Protokoll nutzt [Gmb08]. Damit können KNX-Knoten innerhalb eines Ethernet-Netzwerkes miteinander kommunizieren.
Der große Vorteil besteht hierbei darin, dass Multimediafunktionen und die Steuerungstechnik in einem einzigen Netzwerktyp verschmolzen werden können. Da inzwischen in den meisten Haushalten schon mehr als ein internetfähiges Gerät zum Einsatz kommt und daher in vielen Fällen auch ein Ethernet-Netzwerk zur Verfügung steht, stellt dies nur einen logischen Schritt dar. Abbildung 1.1 zeigt in einem Schema, wie über einen KNXnet/IP-Router ein KNX-Twisted-Pair-Netzwerk in das Ethernet eingebunden werden kann.

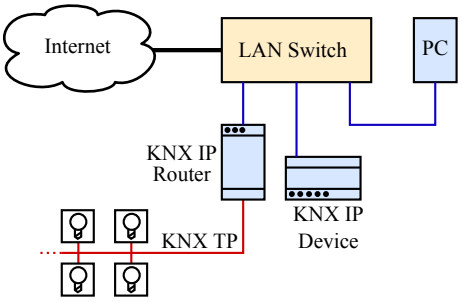

Abbildung 1.1: Anbindung der KNX-TP Bustechnologie an das Ethernet

Das Ethernet kann aufgrund der Busstruktur aber die bisherige Verkabelungstechnik (etwa über Twisted-Pair Kabel) bei der Gebäudeautomatisierung mittels KNX nicht ablösen. Während KNX auch in einer Baumstruktur verlegt werden kann, ist bei Ethernet ein Bus notwendig. Somit ist beispielsweise für die Anbindung der einzelnen Knoten in einem Raum ein Switch zur Verteilung vorzusehen.
Ein weiteres Problem bei der Vernetzung über Ethernet stellt die Stromversorgung dar. So weisen die verfügbaren Standardkomponenten eine höhere Leistungsaufnahme auf als entsprechenden Feldbusmodule. Eine Lösung kann hier *Power Over Ethernet* (PoE) sein, bei dem die Versorgung direkt über das Netzwerkkabel erfolgt.

Ein weiterer Ansatz zur Heimvernetzung soll mit *digitalSTROM* ab 2011 am Markt zur Verfügung stehen [63]. Die digitalSTROM Allianz wurde im Juli 2007 von der ETH Zürich gegründet. Mit Ausnahme des Chipdesigns handelt es sich dabei um einen offenen Standard, der zur Verteilung der Daten bestehende Standards wie XML oder auch Ajax nutzt [Sta10].

Das physikalische Medium zur Übertragung stellt das Stromnetz dar, wobei hier ein von der PLC-Technik abweichendes Verfahren eingesetzt wird, über das aber noch keine genauen Informationen existieren [Sta10].
Der nur 4 mm x 6 mm große digitalSTROM Chip kann direkt an 230 V angeschlossen werden, wodurch also für die Kommunikation und die Stromversorgung nur ein einzelner, sehr günstig herzustellender Chip benötigt wird. Dieser Chip ermöglicht dabei eine breite Palette an Funktionen, beginnend von einfachen digitalen Ein- und Ausgängen bis hin zu einem integrierten Dimmer. Die Besonderheit bei digitalSTROM besteht aber in der einfachen Installation, die auch von Nicht-Fachleuten durchgeführt werden kann. Die Einbindung einer Deckenleuchte in das Netzwerk kann so beispielsweise durch den simplen Austausch der vorhandenen Lüsterklemme durch eine farbig kodierte Variante des digitalSTROM Chips durchgeführt werden.

Verwendet man die Knoten nicht nur zum Steuern, sondern auch zum Messen der aktuell verbrauchten Leistung, so kommt man zum *Smart Grid* – dem intelligenten Stromnetz. Grundgedanke hinter einem *Smart Grid* ist die bessere Verteilung und Ausnutzung der verfügbaren Energie. Mit dem Rundsteuersignal für Tag- und Nachtstrom existiert so eine Technik zwar schon einige Jahrzehnte, allerdings ist dessen Einsatz im Haushalt eher auf Spezialfälle wie etwa den Warmwasserboiler beschränkt.

Besonders interessant wird die Technik des *Smart Grids* dann, wenn man an die unzähligen Geräte denkt, die heutzutage im Standby-Zustand gehalten werden. Diese könnten überwacht und gegebenenfalls auch gesteuert werden (beispielsweise durch das Abschalten kompletter Funktionsgruppen – etwa der Heimkinoanlage mit TV-Gerät, Musikanlage, Receiver, usw.). Über die Vernetzung können so auch Szenarien wie die Abstimmung von verschiedenen Haushaltsgeräten untereinander erreicht werden. So könnte in einem Kühlschrank der Kompressor erst dann wieder aktiviert werden, wenn die Waschmaschine fertig ist. Mit der verfügbaren Intelligenz durch die Vernetzung würde dies ohne Beeinträchtigung des Komforts oder der Funktion geschehen.

Die Abstimmung des Verbrauches kann prinzipiell aber auch in einem globaleren Maßstab geschehen, wenn etwa der Energieerzeuger die Möglichkeit bekommt, dem Kühlschrank eine Empfehlung zu geben, ab wann der Kompressor eingeschaltet werden darf. Auf diese Weise könnten durch das sog. *Demand Side Management* einerseits die Spitzenlasten im Stromnetz besser abgefangen werden, gleichzeitig könnte bei einem entsprechenden Tarifmodell aber auch der Kunde Geld sparen [Pal01].

Damit der Stromanbieter auf die Geräte im Haushalt einwirken kann, muss natürlich auch eine Schnittstelle für die Kommunikation zur Verfügung stehen. Hierzu gibt es mehrere Lösungsansätze ([EL10]) – da die meisten Haushalte aber inzwischen über einen Breitbandanschluss verfügen, wird sich die Übertragung der Daten über den privaten Internetzugang wohl durchsetzen. Die Gegenstelle für das Energieunternehmen wird im Haushalt zurzeit durch den Zähler dargestellt – man spricht in diesem Zusammenhang auch vom intelligenten Zähler oder auch dem *Smart Meter*.

Bei Yello Strom in Deutschland kommt beispielsweise der sog. *Sparzähler* zum Einsatz, auf dem das *Windows CE* Betriebssystem läuft [EL10]. Die Strecke vom Zähler zum Ethernet-Netzwerk wird dabei über das heimische Stromnetz mit handelsüblichen PLC-Modulen überbrückt. Dadurch erhält der Kunde über ein Webinterface eine genaue und jederzeit verfügbare Übersicht seines Stromverbrauches. Durch den Einsatz eines solchen Zählers kann für den Kunden der Energieverbrauch und im weiteren Sinne auch die Energieverschwendung transparent gemacht werden. Da zurzeit noch kein Fernwirken möglich ist, beschränkt sich der Gewinn für Yello Strom

hauptsächlich auf die Möglichkeit zur direkten Weiterverarbeitung der gewonnenen Daten für die Rechnungsstellung.

Der Einsatz von *Smart Metern* ist dabei aber keine technische Spielerei, sondern wird in der Energieeffizienz-Richtlinie der EU gefordert [EU06]. So müssen seit 1. Jänner 2010 alle neuen Zähleranlagen intelligente Zähler sein, um die „Erfassung und informative Abrechnung des Energieverbrauchs“ zu ermöglichen.

Auch dem intelligenten Kühlschrank wurden schon viele Forschungsprojekte gewidmet, deren besondere Eigenschaften von großen Displays über Multimediafähigkeiten [LXG+08] bis hin zur Überwachung der Lebensmittel mittels RFID-Chips [GW09] reichen. Die nachfolgende Auflistung in Tabelle 1.1 soll einen kurzen Überblick über existierende „SmartFridges“ geben (nach [LXG+08]).

Tabelle 1.1: Überblick über existierende „SmartFridges“

Bezeichnung	Features
Siemens	15’ LCD Monitor
CoolMedia FridgeFreezer	Fernbedienung und DVD-Player
LG	15”LCD Monitor und 4”Display
TV Refridgerator	Fernbedienung, Radio
	Anzeige von Wetterdaten, Kalender, Rezeptdatenbank
Elektrolux	15”LCD Touchscreen
Screen Fridge	Netzwerk/Internetanschluss
	Email, Videotelefonie, Organizer-Funktionen
Samsung	10.4”LCD Monitor
Smart Zipel Refrigerator	Netzwerk/Internetanschluss
	Multimediafunktionen

Wie man sieht, sind es durchaus namhafte Hersteller die sich bereits mit dieser Thematik auseinandergesetzt haben. Allerdings fällt auf, dass man sich hier überwiegend auf die Implementierung von Multimediafunktionen beschränkt hat, die vermutlich die wenigsten direkt am Kühlschrank nutzen möchten – vor allem wo der Trend sowieso zu einem überall verfügbaren Internet (Smart-Phones, iPad usw.) geht. Einen ähnlichen Weg geht auch Miele mit seinem *Miele@Home*-System [78], wo mittels KNX über das Stromnetz eine Verbindung zwischen Miele-Haushaltsgeräten hergestellt wird. Von jedem geeigneten Gerät können so die Daten der eingebundenen Geräte abgefragt werden. Über ein Gateway-Modul kann zudem auch ein PC oder ein spezielles Bedienpanel integriert werden. Ein ähnliches System, das ebenfalls über KNX realisiert wird, bietet auch die Firma Liebherr mit seiner net@home Produktreihe an [72].

Betrachtet man all diese Technologien, so erkennt man den Trend zur totalen Vernetzung. Ein Punkt der dabei in dieser Übersicht über den Stand der Technik nicht zur Sprache gekommen ist, ist die Datensicherheit. Hier wird es noch einigen Diskussionsstoff geben, inwieweit die Kunden überhaupt bereit sind ihre Verbrauchsdaten in die Hände der Energieversorger zu legen. Und vor allem wird zu diskutieren sein, wie die Vernetzung im Haushalt gegenüber einem Angriff von außen geschützt werden kann.

2 Aufgabenstellung

Wie schon in der Einleitung angesprochen, ist es das Ziel dieser Arbeit die Einbindung einer handelsüblichen Kühl-Gefrierkombination in das Ethernet und in weiterer Folge auch in das Internet durchzuführen. Über eine Webseite sollen zudem verschiedene Daten (wie etwa der Temperaturverlauf) dargestellt, aber auch Parameter der Regelung konfiguriert werden können. Die Kernaufgaben bei dieser Diplomarbeit bestehen also in der Erfassung der benötigten Messwerte, der Regelung der Temperaturen im Kühl- und Gefrierbereich, der Realisierung des Netzwerkinterfaces und der Präsentation der Daten über eine Webseite.

Im ersten Schritt ist zu klären, welche Anforderungen an die Hardware und Software gestellt werden. Ausgangspunkt sind dabei die Diplomarbeiten von Georg Gaderer [Gad02] und Rainer Voit [Voi99]. Bis auf die Elektronik für das Bedienteil (siehe Kapitel 3.1.3) kann die Hardware dieser Arbeiten aber nicht weiter verwendet werden, sondern muss durch eine eigene Entwicklung ersetzt werden.

2.1 Ausgangszustand

Den Ausgangszustand für diese Diplomarbeit stellt eine Kühl-Gefrierkombination des Typs *KGT 3546* der Firma Liebherr dar. Bei den beiden vorangegangenen Diplomarbeiten wurden die Steckverbinder für Sensoren und Stromversorgung des originalen Kühlschranks beibehalten. Da passende Gegenstücke für die Stecker noch zur Verfügung stehen, kann die vorhandene Steuerungsplatine problemlos durch die hier bei dieser Diplomarbeit zu entwickelnde Hardware ersetzt werden. Wie schon in der Einleitung zu diesem Kapitel angesprochen, kann zudem auch die Hardware für den Bedienteil beibehalten werden.

Als Referenz für das Betriebsverhalten der Kühl-Gefrierkombination werden die beiden Diplomarbeiten [Gad02, Voi99] herangezogen. Dort wo es erforderlich ist, oder eine Verbesserung sinnvoll ist, fließen aber auch eigene Ideen in die Realisierung ein. Die grundsätzliche Funktion und Bedienung soll aber erhalten bleiben.

Der Ausgangszustand der Hardware und das geforderte Betriebs- und Bedienverhalten werden nun nachfolgend zusammengefasst.

2.1.1 Kompressoren

Die Kühlung im Kühl- und Gefrierteil findet über zwei getrennte Kühlkreisläufe und somit auch zwei Kompressoren statt. Für beide Kompressoren gilt eine *Mindeststehzeit* von acht Minuten. Dies ist der Zeitraum in welchem die Kompressoren nach dem Abschalten nicht wieder aktiviert werden dürfen.
Eine weitere Charakteristik ist die *maximale Einschaltzeit*, bei der zwischen dem regulären Betrieb und dem *Superfrost-* bzw. *Supercool*-Betrieb unterschieden werden muss. Im ersten Fall muss ein Kompressor nach vier Stunden Dauerlauf abgeschaltet werden. Der Superfrost-Betrieb erlaubt einen durchgehenden Betrieb des Gefrierteil-Kompressors von maximal 48 Stunden und für den Supercool-Betrieb sind maximal sechs Stunden zulässig. Alle diese Vorgaben dienen zum Schutz und zur Schonung der Kompressoren.

2.1.2 Betriebsarten

Beide Kühlkreisläufe können einerseits im normalen Regelbetrieb, bei dem eine bestimmte Solltemperatur vorgegeben wird, oder in einem Dauerlaufmodus arbeiten. Der Dauerlaufmodus für den Kühlteil wird *Supercool* und für den Gefrierteil *Superfrost* genannt.

2.1.3 Das Bedienpanel

Zur Konfiguration und zur Anzeige aktueller Werte steht ein Bedienpanel mit einigen Tasten, drei LEDs und zwei Paaren von 7-Segment-Anzeigen zur Verfügung. Den Tasten kommen dabei die in Tabelle 2.1 angeführten Aufgaben zu.

Tabelle 2.1: am Bedienpanel verfügbare Tasten

Taste	Funktion
Kühlteil Ein/Aus	Schaltet den Kühlkreislauf für den Kühlteil ein oder aus.
Gefrierteil Ein/Aus	Schaltet den Kühlkreislauf für den Gefrierteil ein oder aus.
Superfrost	Schaltet in den Superfrost-Betrieb (wenn zulässig).
Supercool	Schaltet in den Supercool-Betrieb (wenn zulässig).
Alarm	Bestätigt einen aktiven Alarm und deaktiviert akustische Meldung.

Zusätzlich stehen zum Einstellen der Solltemperaturen auch jeweils zwei Tasten für den Kühl- und Gefrierbereich bereit. Beim ersten Druck auf diese Tasten wird an der zugehörigen Anzeige der eingestellte Sollwert angezeigt. Danach kann über die Tasten der Wert in Schritten von 1 K erhöht oder verringert werden.
Der einstellbare Temperaturbereich ist auf folgende, in Tabelle 2.2 ersichtlichen, Werte beschränkt.

Tabelle 2.2: am Bedienpanel einstellbarer Temperaturbereich

Kühlkreislauf	obere Grenze	untere Grenze
Gefrierteil	-14 °C	-28 °C
Kühlteil	11 °C	2 °C

Über die Tastenkombination *Alarm* und *Superfrost* wird der Status der Kindersicherung geändert. Ist diese aktiviert, so wird dies durch eine LED am Bedienpanel angezeigt und es wird nicht auf das Drücken von Tasten reagiert. Weitere Tastenkombinationen entstehen durch das Betätigen der Alarm-Taste und den Tasten zum Einstellen der Solltemperatur des Gefrierteils. Hiermit kann die Displayhelligkeit in sieben Stufen eingestellt werden.

Zur Anzeige der aktuellen Temperatur und dem Sollwert sind zwei Anzeigen mit jeweils zwei 7-Segment-Anzeigen vorgesehen. Im normalen Betrieb wird auf den Anzeigen die aktuelle Innentemperatur dargestellt. Durch Drücken auf eine der Tasten zur Sollwerteinstellung kann die aktuell eingestellte Solltemperatur für den zugehörigen Kühlkreislauf angezeigt werden.
Der maximal darstellbare Temperaturbereich ist auf die in Tabelle 2.3 angegebenen Werte festgelegt.

Tabelle 2.3: maximal darstellbarer Temperaturbereich

Kühlkreislauf	obere Grenze	untere Grenze
Gefrierteil	0 °C	-40 °C
Kühlteil	19 °C	0 °C

Außerhalb der in Tabelle 2.3 angeführten Bereiche werden auf den betroffenen Anzeigen die Zeichen „–“ dargestellt um auf ein Über- oder Unterschreiten der erlaubten Grenzen hinzuweisen.

Zur Anzeige des Status von Kühlteil, Gefrierteil und Kindersicherung sind weiters drei LEDs vorgesehen.

2.1.4 Türkontakt und Innenbeleuchtung

Das Schließen und Öffnen der Tür wird über einen sog. Reed-Kontakt erkannt, wobei die Tür dann als offen gilt, wenn der Spalt zwischen Dichtung und Gehäuse mindestens 7 mm beträgt. Beim Öffnen der Tür wird sofort die Innenbeleuchtung des Kühlteils eingeschaltet. Die Beleuchtung weist eine Leistungsaufnahme von 30 W auf und wird beim Schließen der Tür, aber spätestens nach 15 Minuten wieder deaktiviert.

2.1.5 Temperaturregelung und Alarmierung

Die Regelung wird bei beiden Kühlkreisläufen durch Zweipunktregler mit Hysterese realisiert. Beim Kühlteil wird zusätzlich die Verdampfertemperatur in die Regelung eingebunden. Befindet sich ein Kühlkreislauf im Superfrost- bzw. Supercool-Betrieb, so findet keine Regelung auf einen bestimmten Sollwert statt, sondern es gelten nur die maximalen Einschaltzeiten.

Überschreitet die aktuelle Temperatur im Kühl- oder Gefrierbereich die Solltemperatur um 4 K, so wird ein Alarm ausgegeben, der durch das Blinken der zugehörigen Anzeige und einen Signalton durch den integrierten Beeper angezeigt wird. Das gleiche Verhalten gilt beim Kühlteil auch für das Unterschreiten von -20 °C.

2.1.6 Verhalten bei Stromausfall

Nach einem Ausfall der Stromversorgung der Kühl-Gefrierkombination müssen folgende Einstellungen erhalten bleiben:

- Zustand der Kindersicherung
- Status der Kühl- und Gefrierteile
- Helligkeit der Anzeige
- Sollwertvorgaben für die Temperaturen

2.2 Die Realisierung

Bevor mit dem Design der Hardware und der Programmierung begonnen werden konnte, musste zunächst geklärt werden, welche Teile der Aufgabe in Hardware und welche in Software realisiert werden. Zudem musste an diesem Punkt auch die Entscheidung darüber getroffen werden, welche Funktionen direkt auf dem Modul für die Steuerung der Kühl-Gefrierkombination integriert und welche ausgelagert werden sollten.

Als Grundlage für die zu entwickelnde Hardware dienten, wie schon erwähnt, die Ergebnisse der zu diesem Thema vorangegangenen Diplomarbeiten. Da es elementar ist, dass die Steuerung der Kühl-Gefrierkombination zuverlässig funktioniert, war schnell klar, dass eine Abgrenzung zwischen der eigentlichen Steuerungsfunktionalität und den *höheren* Funktionalitäten – wie etwa dem Netzwerkinterface – stattfinden muss. Durch diese Trennung kann beispielsweise vermieden werden, dass ein Fehler in der Netzwerkkommunikation den Betrieb der Kühl-Gefrierkombination beeinflusst. Natürlich schließt diese Auftrennung aber einen Fehler in der Firmware der Kühlschranksteuerung nicht aus.

Das Gesamtkonzept für die Hardware wurde also in zwei Systeme aufgeteilt, die miteinander kommunizieren – diese werden im Folgenden als *SmartFridge* (das System zur Steuerung der Kühl-Gefrierkombination) und *SmartPanel* (das Netzwerkinterface) bezeichnet. Abbildung 2.1 zeigt, welche Aufgaben den beiden Systemen zukommt, wobei in dieser Unterteilung noch keine Details zur konkreten Realisierung aufgezeigt werden.

Damit an der Kühl-Gefrierkombination keine größeren mechanischen Veränderungen vorgenommen werden müssen (etwa für den Steckverbinder der Netzwerkschnittstelle) fiel die Entscheidung, die Hardware für das SmartPanel in ein eigenes Gehäuse auszulagern, das außen am Kühlschrank befestigt wird. Somit stellt die Auftrennung in SmartFridge und SmartPanel nicht nur eine rein logische, sondern auch eine physikalische dar. Die Verbindung zwischen den beiden Systemen wird durch ein Kabel hergestellt. Da die zu überbrückende Distanz eher gering ist und keine hohen Datenübertragungsraten gefordert werden, kann hierbei das serielle RS232 Protokoll verwendet werden.

Wie in Abbildung 2.1 ersichtlich ist, hat das SmartPanel neben der Anbindung an das Ethernet die Aufgabe, die erfassten Daten der SmartFridge-Hardware langfristig zu speichern und gegebenenfalls auf einem Display zu visualisieren. Gleichzeitig soll auch die Konfiguration bestimmter Eigenschaften der SmartFridge ermöglicht werden, wozu ein Bedienfeld mit Tasten, LEDs und einer akustischen Signalisierung für Rückmeldungen bereit stehen soll.
Damit die Anzeige von Daten und die Konfiguration nicht nur direkt vor Ort, sondern von nahezu jedem internetfähigen Gerät aus erfolgen kann, soll das SmartPanel auch ein Webinterface

Abbildung 2.1: Zerlegung der Aufgabenstellung in Teilaufgaben für SmartFridge und SmartPanel

zur Verfügung stellen. Dies ermöglicht über einen Browser den Zugriff auf alle relevanten Daten und Funktionen der Kühl-Gefrierkombination.

Die Kernaufgabe der SmartFridge stellt die Steuerung und Temperaturregelung der Kühl-Gefrierkombination dar. Einige der vorgesehenen Sensoren sind bereits im Ausgangszustand vorhanden und werden durch zusätzliche Sensoren ergänzt. Neben der Messung der Luftfeuchtigkeit soll über einen Stromsensor auch die aktuelle Leistungsaufnahme und beim Einschalten eines Kompressors auch dessen Stromverlauf erfasst werden. Durch die Erfassung des Einschaltstromverlaufes ist es möglich, Rückschlüsse auf einen Fehlerzustand (wie etwa einen blockierenden Kompressor) zu ziehen.
Für die über die Sensoren ermittelten Temperaturwerte soll weiters die Möglichkeit einer mittelfristigen Datenspeicherung von mindestens einer Woche vorgesehen werden. Zusätzlich sollen auch auftretende Fehler und Ereignisse für spätere Analysen in einem Datenspeicher hinterlegt werden. Zur Visualisierung und weiteren Verarbeitung werden die gespeicherten Daten regelmäßig über die Schnittstelle an das SmartPanel übertragen.

Die Stromversorgung der SmartFridge ist so auszulegen, dass die erforderlichen Gleichspannungen für die analogen und digitalen Teile der Elektronik direkt aus der Netzspannung generiert werden. Beim SmartPanel soll die Versorgung einerseits direkt über die Verbindung zur SmartFridge, aber auch über ein externes Steckernetzteil erfolgen können. Alternativ wird auch die Möglichkeit der Stromversorgung über das sog. *Power over Ethernet* vorgesehen.

Abbildung 2.2 zeigt einen Überblick über das Gesamtkonzept und die Aufteilung in drei große Module, die prinzipiell unabhängig voneinander bearbeitet werden können. Wichtig ist hierbei eine saubere Festlegung der Schnittstellen zwischen den einzelnen Modulen. Das dargestellte

Modul für die Webseite wird zwar als eigenes, abgesetztes Modul geführt, läuft aber in der realen Implementierung natürlich als Software in der SmartFridge-Hardware.

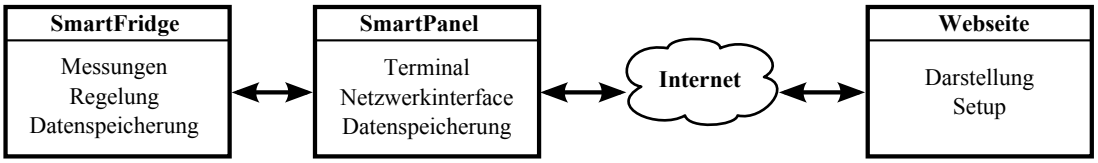

Abbildung 2.2: Aufteilung des Gesamtkonzepts in Teilsysteme

Bei der Auswahl eines Mikroprozessors für die SmartFridge-Hardware wurde das Hauptaugenmerk vor allem auf Leistungsfähigkeit, Speichergröße und die verfügbare Entwicklungsumgebung gelegt. Dies ermöglicht es einerseits zielorientiert zu arbeiten, ohne ständig den genauen Speicherverbrauch im Auge behalten zu müssen und andererseits eine gewisse Reserve für spätere Erweiterungen zur Verfügung zu haben. Gewählt wurde ein 16-Bit Prozessor mit 24 MHz Taktfrequenz und 512 kB an integriertem Flash-Speicher, der unter anderem auch bei Motorsteuerungen und im Automobilbereich zum Einsatz kommt [76]. Um die verschiedenen Teilaufgaben der Firmware miteinander synchronisieren zu können, wird auf ein Echtzeitbetriebssystem gesetzt, das natürlich möglichst ressourcenschonend sein soll.
Zu Beginn der Planung war der selbe Prozessor auch für die Hardware des SmartPanels angedacht. Dies wurde aber verworfen, da der verfügbare RAM-Speicher für Steuerungs- und Regelungsaufgaben zwar mehr als ausreichend, für die Realisierung eines Webinterfaces bei mehreren gleichzeitig agierenden Nutzern aber zu gering ist. Deshalb kommt beim SmartPanel ein Prozessor-Modul zum Einsatz, auf dem ein angepasstes Linux-Betriebssystem arbeitet. Somit kann hier der Entwicklungsaufwand auf eine Basisplatine mit den notwendigen Steckverbindern und die Programmierung der notwendigen Treiber und Applikationsprogramme beschränkt werden.

Abbildung 2.3 zeigt, wie die in Abbildung 2.1 aufgelisteten Funktionsgruppen zusammenarbeiten müssen, um das SmartFridge-System bilden zu können.

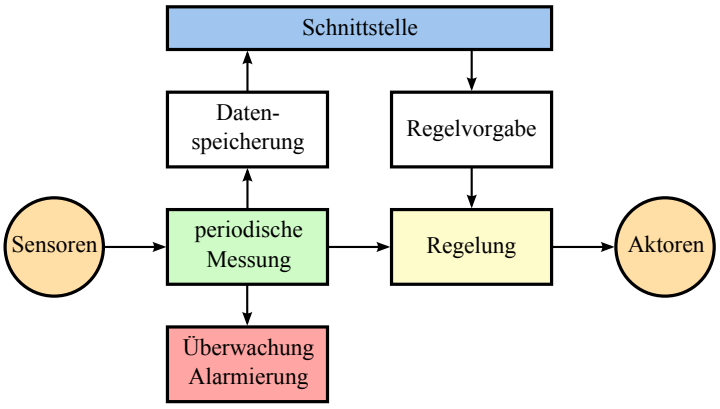

Abbildung 2.3: Zusammenwirken der Funktionsmodule zur Bildung des SmartFridge-Systems

Ausgangspunkt ist die periodische Messdatenerfassung über die verschiedenen verfügbaren Sensoren. Die ermittelten Daten werden einerseits gespeichert, aber auch für die Regelung und Alarmierung im Fehlerfall herangezogen. Über eine Schnittstelle kann einerseits auf die Daten, aber auch auf die Einstellungen der Regelung zugegriffen werden.

In Abbildung 2.4 wird der zeitliche Ablauf der Bearbeitung der drei Teilaufgaben (SmartFridge, SmartPanel und das Webinterface) gezeigt.

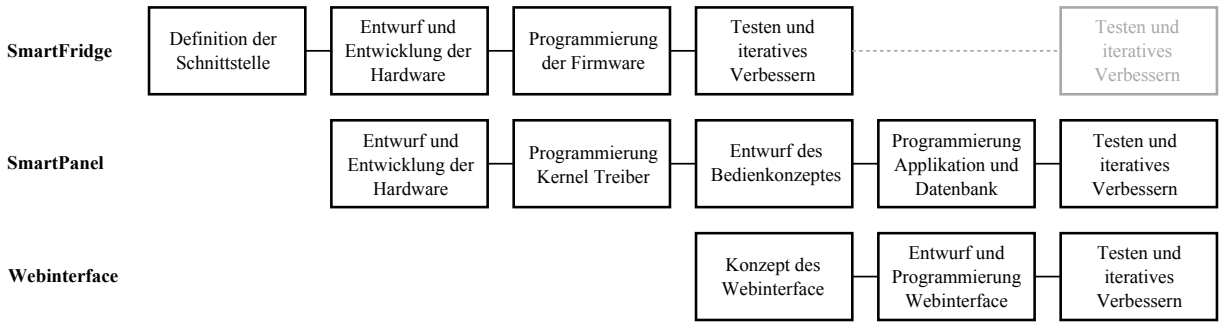

Abbildung 2.4: Zeitlicher Ablauf und Zusammenhang bei der Bearbeitung von SmartFridge, SmartPanel und dem Webinterface

Bevor die Hardware von SmartFridge und SmartPanel entwickelt werden kann, muss zuerst die Schnittstelle zwischen den beiden Modulen definiert werden. Durch diese Festlegung kann anschließend der Entwurf der Hardware von SmartFridge und SmartPanel möglichst unabhängig voneinander erfolgen. Trotzdem ist es zielführend wenn beide Module parallel entwickelt werden, um bei eventuell auftretenden Problemen mehrere Lösungsmöglichkeiten zur Verfügung zu haben. Nach der Bestückung der gefertigten Platinen, erfolgt die Programmierung der Firmware für die SmartFridge und der notwendigen Kernel-Treiber für das beim SmartPanel genutzte Linux-Betriebssystem, um erste Funktionstests durchführen zu können.

Nach der Treiberentwicklung muss beim SmartPanel ein Bedienkonzept für das Zusammenspiel der Eingabeelemente und dem Display erstellt werden. Gleichzeitig kann an diesem Punkt auch das Konzept für das Webinterface entworfen werden, da bei beiden Aufgaben ähnliche Überlegungen anzustellen sind. Anschließend erfolgt beim SmartPanel die Programmierung der eigentlichen Applikation, die einerseits über die Schnittstelle mit der SmartFridge kommuniziert um Daten und Einstellungen auszutauschen und andererseits die Daten für das Webinterface aufbereitet.

Der letzte aufgeführte Punkt ist bei allen Modulen das Testen und iterative Verbessern. Dies bedeutet, dass während dem Testen Probleme, aber auch nicht bedachte Anforderungen auftreten können, die behoben und wieder getestet werden müssen. Da eine Änderung an einem System auch durchaus Auswirkungen auf eines der beiden anderen Systeme haben kann, ist bis zur Fertigstellung des Gesamtsystems ein ständiges Testen des Zusammenspiels aller Systeme notwendig.

3 SmartFridge

In diesem Kapitel wird die komplette Hardware der Kühl-, Gefrierkombination und die auf der SmartFridge laufenden Firmware vorgestellt. Begonnen wird dabei mit der Erläuterung der grundsätzlichen Funktionsweise eines Kühlschrankes. Anschließend werden der Reihe nach die verschiedenen Funktionsgruppen erklärt. Um jeweils die wichtigsten Punkte herausarbeiten zu können erfolgt hierbei eine Trennung in einen Hardware- und einen Softwareteil. So kann etwa bei den Temperatursensoren im Hardwareabschnitt die physikalische Funktionsweise erklärt werden und im Softwareteil wird nur mehr auf die Auswertung der gemessenen Rohdaten eingegangen. Abbildung 3.1 zeigt hierzu schematisch eine Übersicht der verschiedenen Funktionsgruppen der SmartFridge

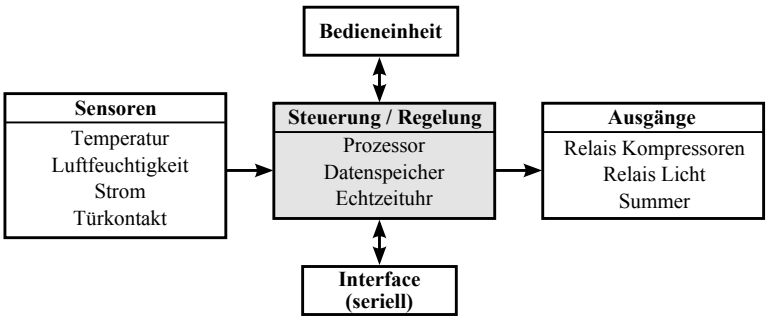

Abbildung 3.1: Übersicht der Funktionsmodule der SmartFridge

3.1 Hardware

Dieser Abschnitt beschreibt die Funktionsgruppen der realisierten Hardware. Begonnen wird mit einer Übersicht über die grundsätzliche Funktionsweise eines Kühlschrankes. Danach erfolgt die Beschreibung zum Schaltungsteil für die Stromversorgung der Elektronik, mit der die Netzspannung in die benötigten Gleichspannungen gewandelt werden. Anschließend erfolgt die Beschreibung der Bedieneinheit der Kühl-Gefrierkombination. Die nachfolgenden Punkte beschäftigen sich anschließend mit der weiteren notwendigen Elektronik und den verwendeten Bauteilen, wie etwa dem Mikroprozessor, den Sensoren für die Temperaturmessung usw.

Abbildung 3.2 zeigt hierzu eine Übersicht über die Aufteilung der Hardware in verschiedene Funktionsmodule.

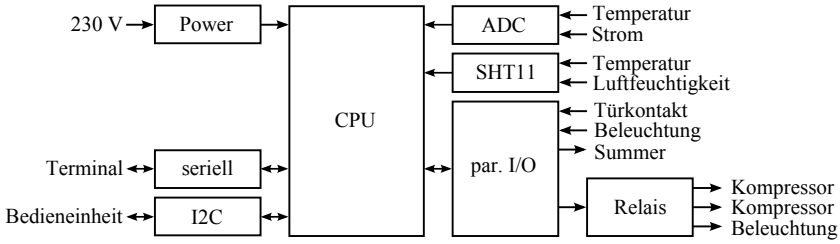

Abbildung 3.2: Übersicht über die Funktionsmodule der Hardware

3.1.1 Funktionsweise eines Kühlschranks

Bevor nun auf die entwickelte Hardware eingegangen wird, soll in diesem Abschnitt die grundsätzliche Funktionsweise eines handelsüblichen Kühlschrankes so weit erläutert werden, wie es für die Implementierung der Hardware und der durchgeführten Messungen relevant ist.

Funktionsbedingt kann man allgemein drei Typen von Kühlschränken unterscheiden: thermoelektrische Kühlschränke, Absorberkühlschränke und Kompressorkühlschränke. In den Haushalten wird normalerweise der letztgenannte Typ verwendet, weswegen hier auch nur dieser näher erläutert wird.

Das Grundprinzip eines Kühlschrankes ist, dass die Wärme aus dem Inneren des Kühlschrankes entzogen wird. Um das in einem Kompressorkühlschrank zu erreichen, sind folgende Komponenten notwendig: ein Kondensator (Verflüssiger), der sich außen am Kühlschrank befindet, ein Verdampfer im Inneren des Kühlschranks, eine Drossel zur Druckreduzierung und ein Kompressor.
Im ersten Schritt wird das Kühlmedium, das bei Normaldruck einen ungefähren Siedepunkt von typisch -30 °C aufweist, mit dem Kompressor verdichtet (auf etwa 8 Bar) und durch den Kondensator geleitet. Durch die Verdichtung im Kompressor wird das gasförmige Medium erhitzt und gibt seine Wärme über die außen am Kühlschrank liegenden Wendeln des Kondensators an die Umgebung ab. Bei der Abkühlung ändert sich der Aggregatzustand des Mediums von gasförmig zu flüssig. Im nächsten Schritt wird diese Flüssigkeit über eine Drossel geleitet, in der sie sich entspannt (also der Druck wieder herabgesetzt wird) und somit auch der Siedepunkt auf die beispielhaft erwähnten -30 °C sinkt. Da es nun im Kühlschrank wärmer als diese Siedetemperatur ist, verdampft die Flüssigkeit im Verdampfer und entnimmt die dazu notwendige Energie aus dem Inneren des Kühlschranks. Damit dieser Prozess aufrecht bleibt, muss das Gas über den Kompressor wieder verdichtet werden - der Kühlkreislauf ist geschlossen. Abbildung 3.3 veranschaulicht dieses Prinzip.

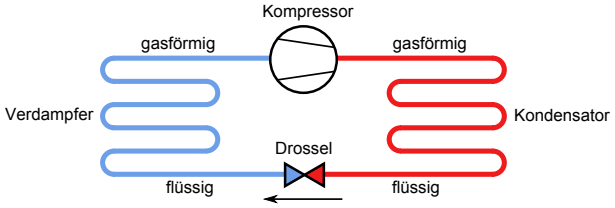

Abbildung 3.3: Prinzip des Kühlkreislaufs

Beim verwendeten Kühlschrank handelt es sich um eine Kühl-Gefrierkombination KGT 3546 von Liebherr. Dieser integriert zwei Kompressoren für den Kühl- und den Gefrierbereich.

3.1.2 Energieversorgung

Der Kühlschrank wird mit Wechselspannung (230 V, 50 Hz) versorgt, wobei die Toleranz seit dem Jahr 2009 bei ±10% liegt [89] - somit muss bei der Dimensionierung der Elektronik mit einem Effektivwert der Spannung zwischen 207 V und 253 V gerechnet werden.
Für den Betrieb der entwickelten Elektronik werden nun verschiedene Spannungen benötigt, die aus der nominalen 230 V Wechselspannung erzeugt werden müssen. Tabelle 3.1 listet diese mit den zugehörigen Funktionseinheiten auf.

Tabelle 3.1: benötigte Spannungen

Spannung	Benötigt für
±15 V	Stromsensor
	Operationsverstärker
+5 V	Digitaler Teil der Elektronik
	Versorgung der NTC-Sensoren
ca. +20 V	Relais

Hauptelement bei der Erzeugung dieser Spannungen ist ein Transformator, der entsprechend den Anforderungen ausgewählt werden muss. Tabelle 3.2 zeigt hierzu die maximale Leistungsaufnahme der für diese Betrachtung relevanten Bauteile.

Tabelle 3.2: Leistungsaufnahme der wichtigen Baugruppen

Bauteil(e)	Strom	Leistung
Relais	2 x 21.8 mA @ 24 V	643 mW
	1 x 5 mA @ 24 V	
Stromsensor	10 mA @ 15 V	150 mW
LEDs (3Stk)	60 mA @ 5 V	30 mW
Mikroprozessor	14 mA @ 5 V	70 mW
MAX202	8 mA @ 5 V	40 mW
FM31256	1.5 mA @ 5 V	7.5 mW
Display Platine	143mA @ 5 V	718 mW
Extern	500 mA @ 5 V	2500 mW
		4158.5 mW

Vernachlässigt wurden dabei einige kleinere Verbraucher und Verluste, die etwa durch die Spannungsregler entstehen. Der Transformator sollte also mindestens eine Nennleistung von 5 VA aufweisen, wobei auf der Sekundärseite nominal 15 V∼ für die Versorgung der ±15 V Linearregler bereit stehen müssen. Zudem waren auch gewisse Vorgaben zur maximalen Platinengröße und Bauteilhöhe in der Kühl- Gefrierkombination einzuhalten.

Aus den erhältlichen Transformatoren wurde ein *FL 10/15* der Firma *Block* mit einer Nennleistung von 10 VA ausgewählt. Um eine symmetrische Belastung des Transformators zu erreichen, wird eine Parallelschaltung der beiden sekundären Anzapfungen durchgeführt, wodurch am Ausgang insgesamt ein Strom von 666 mA (Effektivwert) zur Verfügung steht.

Für den positiven Spannungszweig wird diese Wechselspannung über einen Brückengleichrichter geführt – Abbildung 3.4 zeigt den entsprechenden Schaltungsabschnitt. Die +15 V für den Ana-

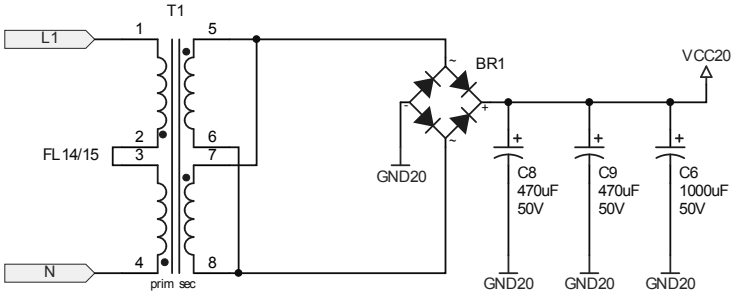

Abbildung 3.4: Schaltung Transformator und Brückengleichrichter

logteil werden über einen Linearregler direkt aus der Spannung nach dem Gleichrichter erzeugt und mit einigen Kapazitäten geglättet. Die Größe der Kapazität ist dabei ausschlaggebend für die so genannte Ripplespannung (früher auch Brummspannung genannt) – das ist eine der Gleichspannung überlagerte Wechselspannung. Bei Schaltungen mit Transformatoren entspricht die Frequenz dieser überlagerten Wechselspannung üblicherweise der Netzfrequenz – also 50 Hz. Die Amplitude der Ripplespannung bei Transformatoren und einem Brückengleichrichter kann über die Näherungsformel 3.1 aus [TSG02] berechnet werden.

$$U_{\mathrm{BrSS}} = \frac{I_a}{2 \cdot C_L \cdot f_n} \left(1 - \sqrt[4]{\frac{R_i}{2 \cdot R_v}} \right) \tag{3.1}$$

R_i stellt dabei den Innenwiderstand des Transformators, R_v den Verbraucherwiderstand, C_L die Kapazität nach dem Gleichrichter, I_a den Ausgangsstrom und f_n die Netzfrequenz dar. Besonders hervorzuheben ist die Abhängig der Ripplespannung vom Ausgangsstrom I_a. Das Ergebnis in Formel 3.1 repräsentiert den Spitzen-Spitzen-Wert der überlagerten Wechselspannung, die Ausgangsspannung schwankt also um $U_{\mathrm{BrSS}}/2$ um den Gleichspannungsanteil.
Um hier auch einen konkreten Wert zu nennen: Bei einer nominellen Eingangsspannung von 230 V und einem Innenwiderstand des Transformators von 3 Ohm, kommt man bei einem Ausgangsstrom von 100 mA und einer Kapazität von 3 mF auf eine Ripplespannung von ungefähr 235 mV.

Die Gleichspannung nach dem Brückengleichrichter beträgt bei einer durchschnittlichen Flussspannung der integrierten Dioden von 0.7 V nominal ungefähr 19.8 V. Berücksichtigt man die schon angesprochene mögliche Untergrenze der Eingangsspannung von 207 V, so ergibt sich eine minimale Spannung von 17.7 V, was für einen 15 V Linearregler gerade noch ausreichend ist (die minimale Eingangsspannung bei dem der Regler noch korrekt arbeitet beträgt laut Datenblatt 17.5 V).

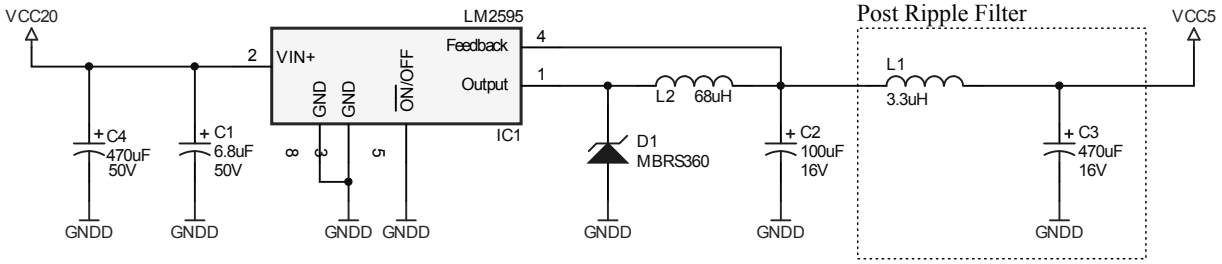

Abbildung 3.5: Schaltung des DC/DC-Wandlers

Die +5 V Betriebsspannung für den überwiegend digitalen Zweig der Elektronik wird ebenfalls aus der Spannung nach dem Brückengleichrichter erzeugt, allerdings wird hierbei ein DC/DC-Schaltregler verwendet (Abbildung 3.5). Durch diesen ist es möglich, die Spannung von etwa 19.8 V mit einem hohen Wirkungsgrad von bis 90% in die gewünschte Gleichspannung von +5 V zu wandeln. Als Wandler kommt dabei ein LM2595 von *National Semiconductor* zum Einsatz, der einen maximalen Ausgangsstrom von bis zu 1 A liefern kann. Die Schaltfrequenz des Reglers beträgt 150 kHz - das entspricht dann auch der zu erwarteten Frequenz der Spannungsschwankung nach dem Wandler. Die Amplitude der Ripplespannung weist bei diesem Schaltregler typischerweise eine Höhe von etwa 50 mV auf, kann aber durch das Hinzufügen eines sogenannten "Post Ripple Filters"[Nat99] auf ungefähr 15 mV reduziert werden – was hier über die Spule *L1* und den Kondensator *C3* auch geschehen ist.
Der erlaubte Eingangsspannungsbereich des Schaltreglers liegt dabei zwischen 7 V und 40 V, wodurch man von den Spannungsschwankungen am Eingang weitgehend unabhängig ist (sofern man die Tatsache außer Acht lässt, dass sich dadurch der Wirkungsgrad ändert).

Um Platz auf der Platine zu sparen kann für die Bereitstellung der -15 V Betriebspannung für die Operationsverstärker aufgrund deren geringen Stromverbrauchs von 3.6 mA ein Einweggleichrichter mit nachfolgendem Linearregler verwendet werden (Abbildung 3.6). Natürlich ist auch hier eine entsprechende Glättung mittels Kondensatoren notwendig.

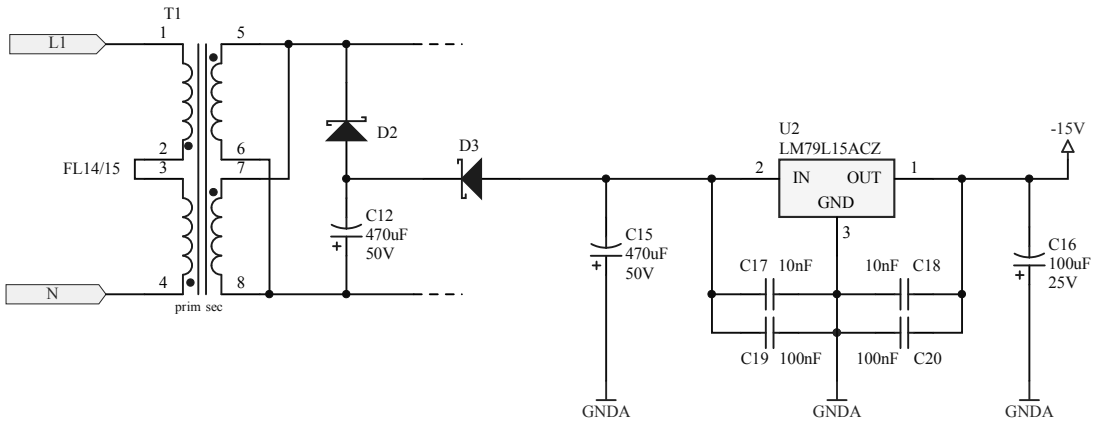

Abbildung 3.6: Schaltung zur Erzeugung der negativen Spannung

Wie in den Abbildungen der Schaltungsausschnitte für die Spannungsversorgung (3.4, 3.5 und 3.6) zu erkennen ist, werden verschiedene Bezeichnungen für das Massepotential verwendet. Dies geschieht um eine Trennung zwischen dem analogen und dem digitalen Teil der Schaltung zu ermöglichen, wobei alle verwendeten Massepotentiale an einem gemeinsamen Punkt mit dem Potential *GND20* zusammengeführt werden. Notwendig ist dies deshalb, da das Schalten von digitalen Ausgängen Störungen verursachen kann, die auf der Analogseite zu Fehlinterpretationen führen können. Um hier ein Gefühl für die Größenverhältnisse zu bekommen, sei als Beispiel ein Analog-Digital-Wandler mit einer Auflösung von 10 Bit betrachtet, der eine Messgröße des analogen Schaltungsteil umwandelt und dem Mikroprozessor zur Verfügung stellt. Bei einer Referenzspannung von 5 V ergibt sich für das LSB des ADCs ein Spannungswert von ungefähr 4.88 mV. Störungen auf der digitalen Masseleitung können sich jedoch im Bereich von einigen zehn Millivolt bis zu einigen hundert Millivolt bewegen [Wil05], wodurch es zu falschen Werten bei der Abbildung des analogen auf einen digitalen Wert kommen kann.

Die komplette Schaltung der Energieversorgung ist im Anhang 7.1 ersichtlich.

3.1.3 Bedieneinheit

Am Kühlschrank steht zur Darstellung von Werten und zur Einstellung des Betriebszustandes ein Bedienpanel zur Verfügung. Abbildung 3.7 zeigt dieses Panel und kennzeichnet die verschiedenen Tasten und Anzeigeelemente. Diese Bezeichnungen werden im Kapitel über die Software auch für die Beschreibung der verschiedenen Konfigurationsmöglichkeiten verwendet, weshalb hier nicht weiter auf die Bedeutung und Funktion der Tasten und Anzeigen eingegangen wird.

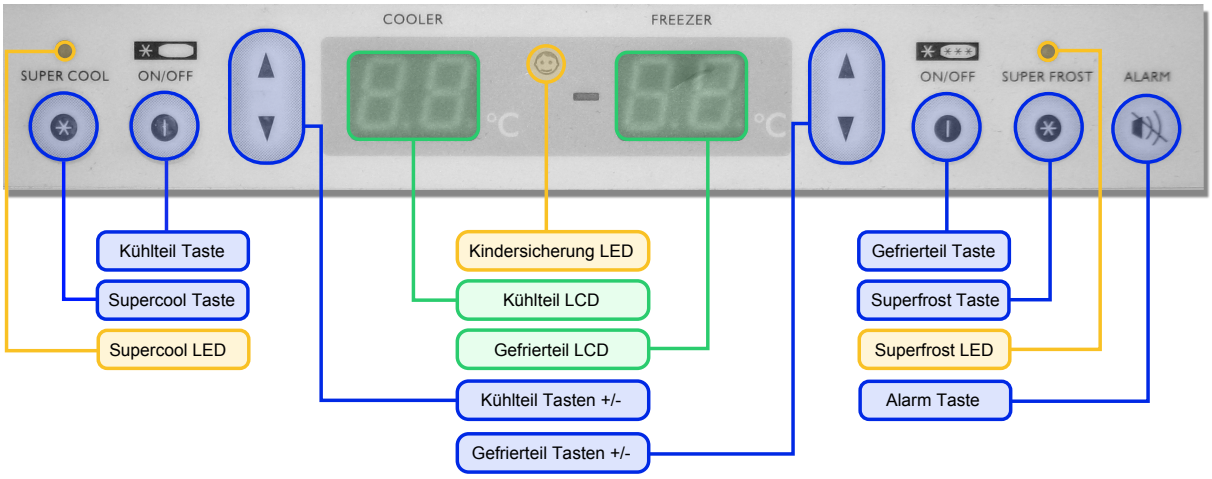

Abbildung 3.7: Bedienpanel der Kühl-Gefrierkombination

Es stehen also folgende Elemente zur Verfügung:

- 5 normale Tasten
- 2 Tasten mit +/- Funktionalität
- 2 LEDs zur Anzeige des Betriebszustandes
- 1 LED zur Anzeige des Zustandes der Kindersicherung
- 4 7-Segment-Anzeigen in zwei Gruppen

Die Hardware des originalen Bedienpanels der Kühl-, Gefrierkombination wurde dabei in der vorhergehenden Diplomarbeit durch eine Eigenentwicklung ersetzt, die hier weiter verwendet wird.

Zur Ansteuerung der vier 7-Segment-Anzeigen wird der IC SAA1064 von Philips verwendet, wobei als besonderes Feature die Leuchtstärke der Anzeige verändert werden kann. Da die Dezimalpunkte der 7-Segment-Anzeigen nicht benötigt werden, können die vier dazu vorgesehenen Ausgänge am SAA1064 zur Ansteuerung der LEDs verwendet werden. Abbildung 3.8 zeigt die zugehörige Schaltung.

Für die Erfassung der Tastenzustände kommt der Baustein PCF8574 zum Einsatz. Genauso wie beim SAA1064 erfolgt auch bei diesem IC die Kommunikation über das I2C-Protokoll [NXP07]. Damit am Prozessor die Tasten nicht ständig abgefragt werden müssen, stellt der Portexpander PCF8574 einen Interrupt-Ausgang zur Verfügung, der bei Änderungen des Zustands einer Taste aktiv geschalten wird (Low-Aktiv). Somit muss der Prozessor erst dann die Auswertung durchführen, wenn auch wirklich eine neue Information vorliegt. Näheres zur Tastenbehandlung folgt im Abschnitt 3.2.10.

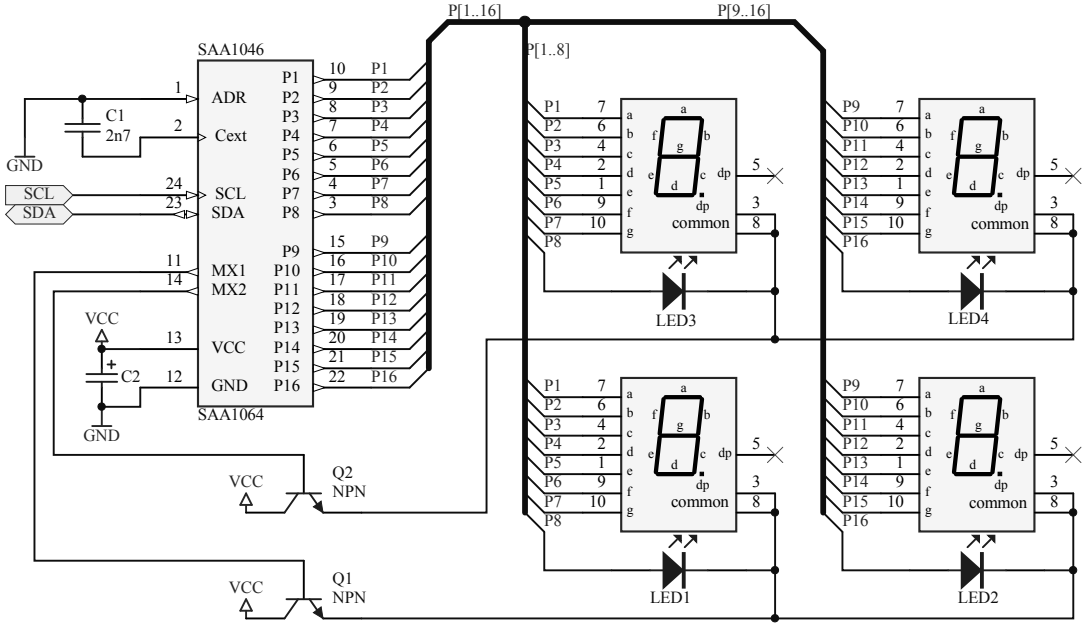

Abbildung 3.8: Schaltung der Bedienpanelanzeige

3.1.4 Mikroprozessor

Das Herzstück der Steuerung und Regelung der Kühl-, Gefrierkombination stellt ein Mikroprozessor dar. Folgende Punkte wurden bei der Auswahl dieses Mikroprozessors als Entscheidungsmerkmale festgelegt um die geforderten Aufgaben erfüllen zu können:

- ausreichend hoher Datendurchsatz
- integrierter Speicher (Flash und RAM)
- hohe Anzahl an verfügbaren Ports für die Ein- und Ausgabe
- serielle Schnittstelle für Kommunikation
- möglichst hohe Anzahl an Timern und externen Interrupts
- gute Entwicklungsumgebung mit Debuggingfunktionalität

Gewählt wurde der Mikroprozessor M16C/62P [Ren06] von Renesas (die genaue Typenbezeichnung lautet M30626FJPGP), der über einen 512 kB großen Flash-Programmspeicher und 31 kB an Datenspeicher (RAM) verfügt. Dabei handelt es sich um einen 16-Bit Prozessor, der für die Programmierung in einer Hochsprache wie C optimiert wurde [Ren04]. Tabelle 3.3 zeigt eine Übersicht über einige zusätzliche Kenndaten.

Die Entscheidung für diesen Mikroprozessor fiel aus mehreren Gründen. So steht ein vergleichsweise großer Programmspeicher bereit, der zusätzlich über einen externen Bus erweitert werden kann. Wie in der Tabelle 3.3 ersichtlich ist, integriert der Prozessor zudem bereits ein großes Spektrum an unterschiedlicher Peripherie, sodass für die komplette Schaltung der SmartFridge-Hardware nur wenige zusätzliche Bauteile notwendig sind. Laut Herstellerangaben wurden beim Design dieses Prozessors auch Maßnahmen (wie interne Isolation der unterschiedlichen Spannungsbereiche oder integrierte Entstörungsfilter) getroffen, um ein gutes EMV-Verhalten zu erreichen [75]. Dies kann sich vor allem in einem Umfeld, in dem größere Lasten (wie etwa die Kompressoren im Kühlschrank) geschalten werden müssen, als Vorteil erweisen.

Tabelle 3.3: Übersicht über Kenndaten des Prozessors

Eigenschaft/Funktion	Wert
Betriebsspannung	2.7 V bis 5.5 V
maximale Taktfrequenz	24 MHz
Stromverbrauch	14 mA (bei 24 MHz)
Adressraum	1 MiB bis zu 4 MiB in speziellem Modus
Portpins Input/Output (ohne Nutzung der Peripherie)	87
Timer	11
Interruptquellen	Intern: 2 Extern: 8
serielle Schnittstellen	5 (synchron, UART, I2C, IEBUS - nicht jeder Modus überall verfügbar)
ADC	26 Kanäle (8 oder 10 Bit)
DAC	2 Kanäle (8 Bit)
weitere Peripherie	DMA Controller CCITT-CRC Berechnungseinheit Watchdog

Schlussendlich existiert für diese Controllerfamilie von Renesas eine gute Entwicklungsumgebung inklusive Debugger, in der jederzeit die Programmausführung gestoppt und der Register- und Speicherinhalt geprüft werden kann. Dies ist bei der Programmierung von Low-Level Treibern, wie etwa für die serielle Schnittstelle oder der Displayansteuerung, ein wichtiges Hilfsmittel um die korrekte Ausführung prüfen und Fehler finden zu können. Fast noch wichtiger ist die Unterstützung durch den Debugger bei komplexeren Modulen wie beispielsweise dem Dateisystem für den FRAM-Datenspeicher (siehe Abschnitt 3.2.13) – selbst bei sorgfältiger Planung und Programmierung können Fehler auftreten, die ohne Debugger nur schwer zu lokalisieren sind. Neben dem Debugger steht auch ein Simulator zur Verfügung, mit dem ebenfalls Softwareteile simuliert und geprüft werden können. Prinzipiell könnte zwar das gesamte System inklusive der Peripherie nachgebildet werden, dies würde aber die Programmierung zusätzlicher Module für den Simulator erfordern, was hier in keinem Verhältnis zum Nutzen stehen würde. Der Simulator wurde deshalb ausschließlich zur Prüfung von Codeteilen und Abschätzung von Laufzeiten bzw. benötigten Prozessorzyklen verwendet.

3.1.5 Analog-Digital Konverter

Der im Mikroprozessor integrierte Analog-Digital Konverter (ADC) funktioniert nach dem Prinzip der sukzessiven Approximation [Mag04, Ler07] und wird in dieser Anwendung im 10-Bit Modus betrieben. Die Auflösung bei voller Ausnutzung der 5 V Referenzspannung beträgt also ungefähr 4.8 mV. Tabelle 3.4 zeigt einige hier relevante Eigenschaften des ADCs.

Die Konversionszeit, also die Zeitdauer von der Abtastung bis zum Ergebnis, wird in AD-Zyklen angegeben. Diese Zykluszeit hängt von der Wahl der Arbeitsfrequenz für den ADC ab, die aus dem Prozessortakt über einen Vorteiler erzeugt wird. Bei 24 MHz beträgt die höchste mögliche Arbeitsfrequenz des Konverters 12 MHz (also Vorteiler 2) und die niedrigste 2 MHz (Vorteiler 12).

Tabelle 3.4: Eigenschaften des ADC

Eigenschaft	Wert
Betriebsmodi	One-Shot Repeat Single Sweep Repeat Sweep
integraler Nichtlinearitätsfehler	± 3 LSB
Konversionszeit	33 AD Zyklen

Der ADC-Takt kann dabei nicht willkürlich gewählt werden, sondern es muss die Impedanz des zu messenden Elements berücksichtigt werden, siehe [Ren06]. Bei ungeeigneter Wahl wird der Kondensator der internen Sample-and-Hold Schaltung nicht vollständig geladen und es kommt zu einer Verfälschung des Messwertes.

Nach Abschluss der Messung kann der Messwert aus dem Register des verwendeten ADC-Kanals ausgelesen werden. Damit im Programmcode nicht auf das Ergebnis der Messung gewartet werden muss, besteht auch die Möglichkeit der Benachrichtigung über einen ADC-Interrupt.

3.1.6 Temperatur- und Luftfeuchtigkeitsmessung

Bei der Temperaturmessung kommen zwei verschiedene Typen von Sensoren zum Einsatz. Zum einen sind dies *NTC*-Sensoren (Negative Temperature Coefficient), von denen drei Stück schon in der Originalausführung des Kühlschranks vormontiert sind und durch zwei zusätzliche NTC-Sensoren ergänzt werden. Zum anderen wird auch ein digitaler Sensor verwendet, der auch die Möglichkeit zur Messung der Luftfeuchtigkeit bietet.

NTC-Sensoren (auch Heißleiter genannt) sind temperaturabhängige Widerstände, die bei höheren Temperaturen einen niedrigeren Widerstand aufweisen als bei tiefen Temperaturen. Der Temperaturkoeffizient, also die Änderung pro Kelvin, bewegt sich dabei in einem Bereich von -2 %/K bis -6 %/K. Als Werkstoff werden meist verschiedene Metalloxide wie Mangan, Eisen, Kobalt, Nickel, Kupfer und Zink eingesetzt, die für eine bessere thermische Stabilität mit Bindemitteln versetzt, und anschließend gesintert werden [Epc06].

Die Kennlinie von NTC-Sensoren ist grundsätzlich nichtlinear, weshalb es notwendig ist, diese Kennlinie in eine für die Weiterverarbeitung im Prozessor passende Form zu konvertieren. Prinzipiell könnte dies durch eine Näherung über eine Polygonfunktion realisiert werden. Hier wird aber eine andere Methode eingesetzt, und zwar eine Tabelle von vorberechneten Referenzpunkten zwischen denen interpoliert wird. Näheres zu dieser Auswertung erfolgt im Abschnitt über die Softwarerealisierung der SmartFridge. Die Messpositionen der drei originalen Sensoren sind jeweils im Kühl-, und Gefrierraum des Kühlschranks und am Kompressor. Die beiden zusätzlichen NTC-Sensoren werden zur Erfassung der Außentemperatur und der Verdampfertemperatur verwendet.

Da die drei integrierten Sensoren schon im Auslieferungszustand vorhanden, und keine genauen Kennlinien verfügbar waren, musste eine Rückrechnung über die angegebenen Werte der vorhergehenden Diplomarbeit [Gad02] durchgeführt werden. Tabelle 3.5 zeigt die Messwerte, die dafür

zur Verfügung standen. Damit die Anschlüsse der originalen Sensoren voneinander unterschieden werden können, weisen sowohl Stecker als auch Buchsen eine farbliche Codierung auf, was ebenfalls in Tabelle 3.5 aufgelistet ist.

Tabelle 3.5: Messwerte der NTC-Sensoren zur Rückrechnung und Farbcodierung der Steckverbinder

Sensorposition	Farbe	R bei 25 °C	R bei 5 °C
Verdampfer	grün	10 kΩ	25.011 kΩ
Kühlbereich	rot	10 kΩ	25.011 kΩ
Gefrierbereich	weiß	2 kΩ	4.584 kΩ

Zusätzlich konnte auch auf Näherungsformeln aus dem Quellcode der vorhergehenden Diplomarbeit zurückgegriffen werden. Mit der Annahme, dass es sich bei den Sensoren um keine exotischen, nicht im Handel erhältlichen Typen handelt, konnten über die Daten vom Hersteller Epcos annähernd passende Sensortypen ermittelt werden.
Der für ein NTC-Material charakteristische sog. „*B-Wert*“ kann über Formel 3.2 nach [Epc06] berechnet werden.

$$B = \frac{T \cdot T_R}{T - T_R} \cdot \ln \frac{R_R}{R_T} = \frac{T_5 \cdot T_{25}}{T_5 - T_{25}} \cdot \ln \frac{R_{25}}{R_5} \tag{3.2}$$

Damit erhält man für die Sensoren am Verdampfer und im Kühlbereich einen B-Wert von ungefähr 3800 und für den Gefrierbereich einen Wert von ungefähr 3440. Diese Werte können nun zur Auswahl eines ähnlichen Types verwendet werden, um die Charakteristik für die weitere Auswertung zu erhalten. Für die beiden zusätzlichen NTC-Sensoren wird ein Typ mit dem B-Wert von 3950 verwendet. Tabelle 3.6 zeigt die gewählten Charakteristiken für die originalen und die zusätzlichen NTC-Sensoren (die Typenbezeichnung bezieht sich dabei auf Produkte des Herstellers Epcos).

Tabelle 3.6: Charakteristiken der gewählten NTC-Sensoren

R_{25}	B-Wert	Verwendung
10 kΩ	3800	Verdampfer, Kühlbereich
2 kΩ	3450	Gefrierbereich
4.7 kΩ	3950	zusätzliche Sensoren

Über die Formel 3.3 kann nun der Widerstand eines NTC-Sensors in Abhängigkeit von der Temperatur berechnet werden. Dabei ist darauf zu achten, dass die Temperatur hier in Kelvin angegeben wird. (0 °C $\widehat{=}$ 273.16 K)

$$R_T = R_{25} \cdot e^{B \cdot \left(\frac{1}{T} - \frac{1}{T_{25}}\right)} \tag{3.3}$$

Um den Kennlinienverlauf nun über den Analog-Digital-Wandler des Prozessors einlesen zu können, ist eine Umwandlung in einen Spannungswert notwendig, der das relevante Widerstandsintervall in ein entsprechendes Spannungsintervall umwandelt und hierbei so weit wie möglich den Eingangsspannungsbereich des AD-Konverters ausnutzt. Die entsprechende Schaltung für den vorher angeführten Sensortyp zeigt Abbildung 3.9.

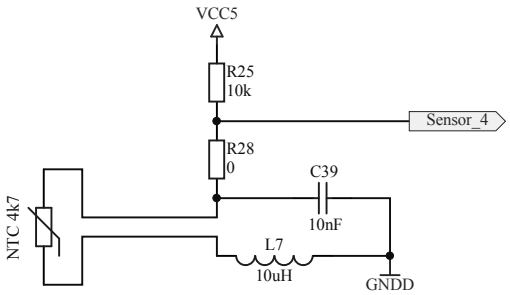

Abbildung 3.9: Schaltung NTC-Sensor

Die Dimensionierung erfolgte dabei mit dem Ziel eines möglichst linearen Zusammenhangs im relevanten Temperaturbereich von etwa -20 °C bis 30 °C zu erreichen. Die Induktivität und die Kapazität dienen hierbei zur Unterdrückung von eingekoppelten Störsignalen, die durch die doch etwas längeren Leitungen zu den Montagepunkten der Sensoren entstehen können.

Die zu dieser Schaltung zugehörige Temperatur-Spannungskennlinie zeigt Abbildung 3.10.

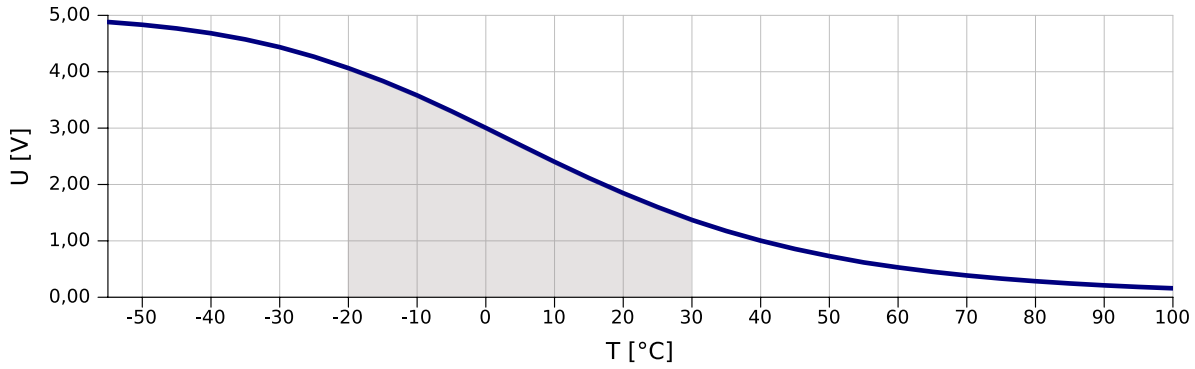

Abbildung 3.10: Verhältnis Spannung zu Temperatur bei NTC-Messung

Die hier berechneten Werte stellen aber genau genommen nur eine gute Näherung dar, da die Formel 3.3 nicht exakt ist. Eine genauere Berechnungsmöglichkeit besteht über die *Steinhart-Hart Gleichung*, die hier aber nicht näher erläutert werden soll. Um nun dennoch eine möglichst genaue Berechnung zu erhalten, wird auf die in tabellarischer Form vorberechneten Werte der Sensoren zurückgegriffen, die in den Datenblättern zu finden sind. Abbildung 3.11 zeigt für die zusätzlichen NTC-Sensoren den relativen Fehler, der durch die Näherungsformel entsteht. Man erkennt, dass die größte Abweichung im unteren Temperaturbereich vorliegt.

Interessant ist in diesem Zusammenhang noch die Betrachtung der erreichbaren Auflösung. Da, wie in Abbildung 3.10 ersichtlich ist, das Verhältnis zwischen Temperatur und Widerstands-, bzw. Spannungswert nichtlinear ist, variiert auch die erreichbare Auflösung. Das Diagramm in Abbildung 3.12 zeigt für den hier relevanten Temperaturbereich links den Zusammenhang zwischen Temperatur und dem gemessenen ADC-Wert und rechts die zugehörige Auflösung. Die maximale Auflösung wird also im Bereich von etwa 5 °C erreicht.

Wie etwas weiter oben schon angesprochen, wird noch ein zweiter Sensortyp verwendet, der über eine digitale serielle Schnittstelle ähnlich dem I2C-Protokoll [NXP07] angesprochen wird und bei dem die Messung und Aufbereitung somit schon im Sensor selber erfolgt. Es handelt sich hierbei um einen Sensor vom Typ SHT11 des Herstellers Sensirion [Sen07] - mit diesem Sensor

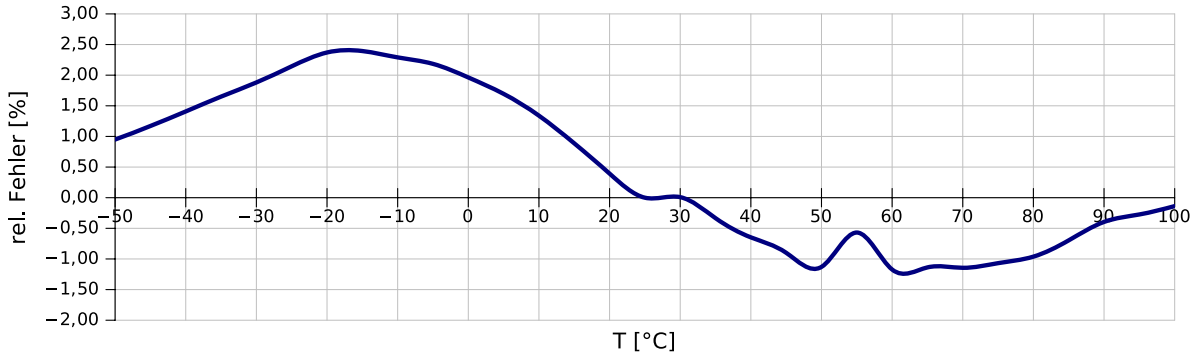

Abbildung 3.11: Relativer Fehler Formel und Tabelle

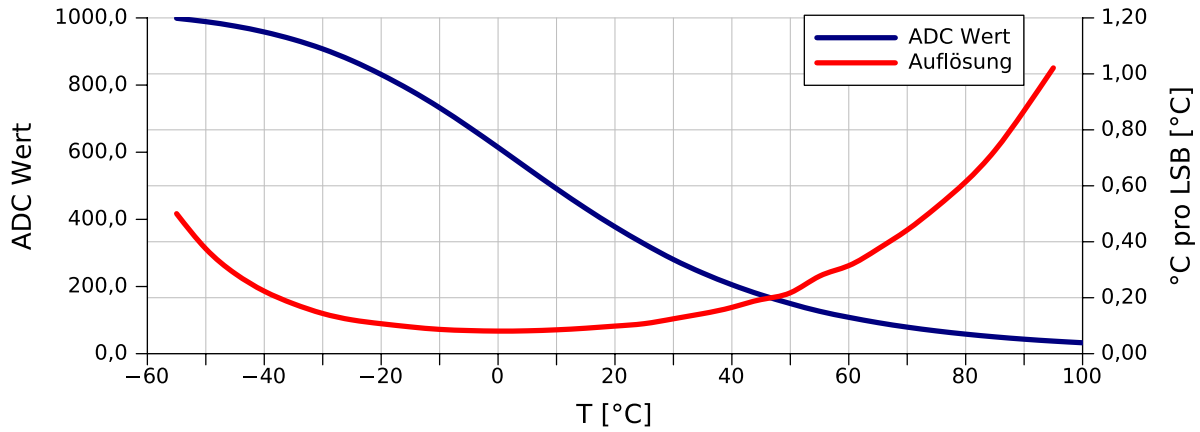

Abbildung 3.12: Maximale Auflösung bei der NTC-Sensor Auswertung

kann neben der Temperatur auch die relative Luftfeuchtigkeit gemessen werden. Zur Messung der relativen Luftfeuchtigkeit kommt bei diesem Sensor ein kapazitives Polymer zum Einsatz, während die Temperaturmessung über einen temperaturabhängigen Halbleiter erfolgt. Die besonderen Merkmale dieser Sensortypen von Sensirion sind dabei die werkseitige Kalibrierung und die exzellente Langzeitstabilität. Die folgende Auflistung zeigt einige Eckdaten des verwendeten Sensors.

- Spannungsversorgung mit 2.4 V bis 5.5 V
- Genauigkeit der Temperaturmessung (bei 25 °C): ±0.4 K
- Genauigkeit der Luftfeuchtigkeitsmessung: ±3 %
- Auflösung bei Temperaturmessung: 12 oder 14 Bit
- Auflösung bei Luftfeuchtigkeitsmessung: 8 oder 12 Bit
- hohe Langzeitstabilität und werksseitige Kalibrierung

Abbildung 3.13 zeigt hierzu die erreichbare Genauigkeit bei der Temperatur- und Luftfeuchtigkeitsmessung (aus [Sen07]).

Auch wenn die Daten über eine digitale Schnittstelle eingelesen werden, müssen die erhaltenen Werte im Prozessor noch weiter aufbereitet werden, da es sich um Rohdaten handelt, die über eine Formel aus dem Datenblatt ([Sen07]) in Temperatur und relative Luftfeuchtigkeit umgerechnet werden müssen. Näheres dazu folgt wiederum im Softwareabschnitt zu diesem Sensor (Abschnitt 3.2.5).

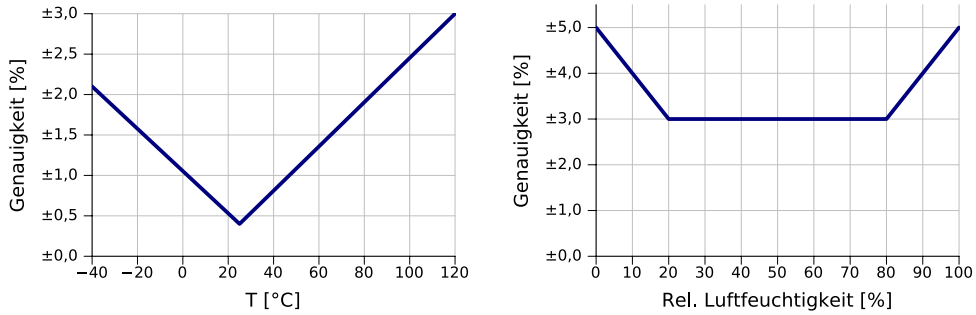

Abbildung 3.13: Genauigkeit des Sensirion SHT11 Sensors

3.1.7 Strommessung

Zur Strommessung an der Kühl-Gefrierkombination kommt der auf dem Halleffekt basierende Sensor HY05P von LEM zum Einsatz [71]. Dieser übersetzt einen Strom im Bereich von ±5 A (RMS) in einen Spannungsbereich von ±4 V mit einem Fehler von ±1%. Ein erweiterter Bereich bis zu ±15 A (Spitze) kann mit einer geringeren Genauigkeit erfasst werden. Die maximale Ausgangsspannung des Sensors liegt somit bei 12 V.

Um die in Wechselspannung umgewandelte Wechselstromgröße im Mikroprozessor erfassen zu können, ist zuvor eine Gleichrichtung und eine Anpassung an den Messbereich des ADCs von 5 V notwendig. Hierzu wird wie in der Diplomarbeit [Gad02] ein Präzisions-Vollweggleichrichter verwendet [CB01] (Abbildung 3.14).

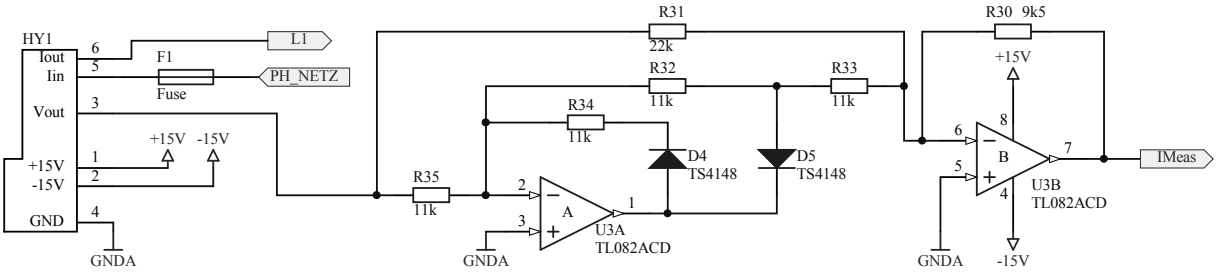

Abbildung 3.14: Schaltung des Vollweggleichrichters zur Strommessung

Wie schon angesprochen wird der gemessene Wechselstrom durch den Stromsensor in eine Wechselspannung von maximal 12 V umgesetzt. Der Ausgangsspannungsbereich des verwendeten Operationsverstärkers liegt dabei etwa 3 V unterhalb der Versorgungsspannung, weshalb der OPV mit ±15 V versorgt wird um den vollen Eingangsspannungsbereich abbilden zu können. OPV *U3A* stellt mit den Dioden *D4* und *D5* einen Einweggleichrichter dar, der bei positiver Spannung leitet und zudem auch invertierend und verstärkend wirkt. Über OPV *U3B* werden anschließend der volle Sinusverlauf über *R31* und der gleichgerichtete Verlauf addiert, wodurch sich insgesamt eine Vollweggleichrichtung des Eingangssignals ergibt. Der Unterschied zur Strommessung in [Gad02] besteht nun darin, dass kein passiver Filter zur Glättung verwendet wird. Warum das so ist, soll Abbildung 3.15 verdeutlichen.

Hier wird ein Einschalten mit maximalem Strom (15 A) simuliert. Die gestrichelte Linie stellt dabei den berechneten Effektivwert dar, während die zweite Linie die Ausgangsspannung mit passivem Filter wie in [Gad02] zeigt. Obwohl ein stoßartiger Sprung von 0 A auf 15 A in der

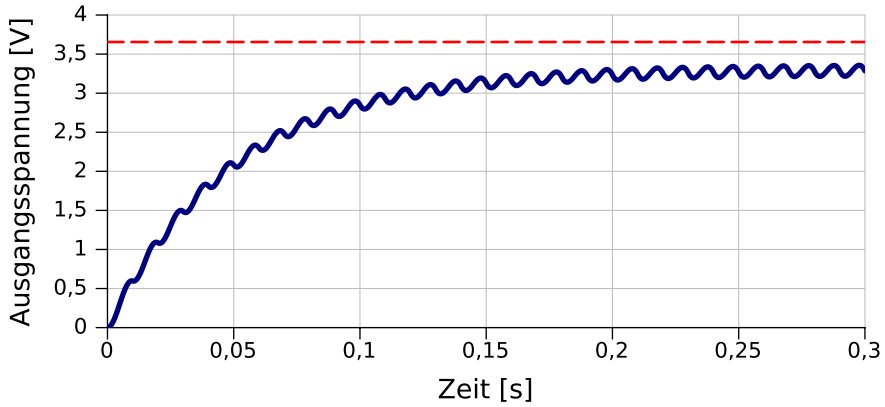

Abbildung 3.15: Diagramm Simulation Einschalten bei Strommessung

Realität bei einem Kühlschrankkompressor nicht zu erwarten ist, zeigt es doch recht deutlich, wie der stationäre Wert mit dem Filter erst nach einer gewissen Zeit erreicht wird (was aufgrund der Kapazität im Filter auch nicht anders zu erwarten ist). Um somit ein Ergebnis für die Bewertung zu erhalten, ist es hier in der Firmware erforderlich die Restwelligkeit wegzurechnen und über einen Faktor den Effektivwert zu bestimmen.
Dies ist der Grund wieso in dieser Diplomarbeit kein passives Filter eingesetzt wird, sondern die Filterung und Berechnung des Effektivwerts der gleichgerichteten Spannung direkt im Mikroprozessor erfolgt. Durch diese rein digitale Bearbeitung kann die Berechnung flexibler und schneller erfolgen, was natürlich auch einen entsprechend leistungsfähigen Mikroprozessor vorraussetzt.

3.1.8 Lampenprüfung

Auch die Prüfung der Funktionsfähigkeit der Beleuchtung in der Kühl-Gefrierkombination erfolgt mit einem etwas anderen Ansatz. Bei der Arbeit in [Gad02] wurde der Zustand der Beleuchtung über den Stromverbrauch gemessen. Da dieser bei der hier verbauten Lampe mit 30 W relativ gering ist, wurde eine statistische Methode verwendet um den aktuellen Zustand ermitteln zu können. Im Gegensatz dazu wird hier nicht der Strom bei eingeschalteter Beleuchtung gemessen, sondern direkt die Lichtstärke. Dafür wird eine simple Schaltung zur hell/dunkel-Unterscheidung nach [Bau06] verwendet, siehe Abbildung 3.16.

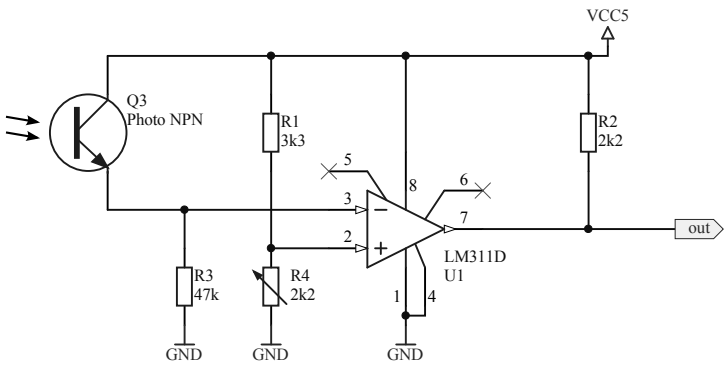

Abbildung 3.16: Schaltung zur Hell/Dunkel-Erkennung

Diese Variante hat den Nachteil, dass die Messung nur bei geschlossener Kühlschranktür durchgeführt werden kann, um eine sichere Aussage über die Funktionsfähigkeit der Leuchte treffen zu können. Da es sich hierbei aber um keine funktionskritische Komponente handelt, und es praktisch gesehen keinen Unterschied macht ob die Erkennung während oder nach dem Öffnen der Tür erfolgt (zumal der Benutzer eine defekte Lampe sowieso auch optisch wahrnimmt), kann dieser Nachteil vernachlässigt werden. Der eigentliche Sinn dieser Erkennung besteht in der Möglichkeit, entsprechende Log-Einträge zu setzen, oder Meldungen an die zuständige Stelle weiterzuleiten.

3.1.9 Datenspeicher und Echtzeituhr

Neben den integrierten Speichertypen im Prozessor (Flash und RAM) steht auch noch ein externes *FRAM* zur Verfügung, wobei diese Abkürzung für „Ferroeletric Random Access Memory“ steht [Ramer, Phi96]. Die Vorteile der FRAM-Technologie gegenüber der Flash- oder EEPROM-Technologie liegen in der höheren Schreibgeschwindigkeit, dem geringeren Energiebedarf für das Schreiben und auch der höheren Lebensdauer der FRAM-Zelle.

Wie bei einem (D)RAM-Speicher kann auch bei einem FRAM byteweise gelesen und geschrieben werden - bei Flash-Speichern in der NOR-Technologie ist dies zwar auch möglich, aber für das Schreiben wird eine wesentlich längere Zeitdauer benötigt. Zudem können Flash-Speicher nur blockweise (üblicherweise zwischen 256 Byte und 4 kByte) gelöscht werden. Vor allem die Charakteristiken der Schreibzyklen und der Blockgrößen beim Löschen machen es bei der Implementierung von Dateisystemen für Flash-Speicher notwendig, gewisse Mechanismen wie Wear-Leveling [WC07] (gleichmäßige Verteilung der Daten auf dem ganzen Flash-Speicher um die Anzahl der Löschvorgänge zu verringern) oder auch ein Log-basierendes Dateisystem [KNM95] [90] zu verwenden.

Beim verwendeten Baustein handelt es sich nun um einen FM31256 der Firma Ramtron mit 32kB an Speicher [Ram10]. In diesen sind neben dem eigentlichen Datenspeicher auch noch eine Echtzeituhr, eine Resetschaltung, ein Watchdog-Timer, eine Stromausfallerkennung, zwei 16-Bit Counter und eine einstellbar Seriennummer die zur Identifizierung von Geräte eingesetzt werden kann, integriert. Bei der SmartFridge-Hardware wird davon jedoch nur der Datenspeicher und die Echtzeituhr genutzt. Für die Funktion der Echtzeituhr ist zusätzlich noch ein externer 32.768 kHz Uhrenquarz zu beschalten. Damit die Echtzeituhr des Bausteins auch während eines Stromausfalls weiterläuft, wurde zudem ein Goldcap-Kondensator an dafür vorgesehene Eingänge des FM31256 angeschlossen.

Um eine möglichst geringe Abweichung der Echtzeituhr zu erreichen, ist eine Kalibrierung möglich. Dazu wird der FM31256 in einen speziellen Modus gesetzt, in dem an einem Ausgangspin ein Rechtecksignal mit nominal 512 Hz erzeugt wird. Durch Messung der Abweichung mittels eines genauen Frequenzzählers und anschließender Korrektur über ein Kommando an den Baustein, kann der Fehler auf ± 0.09 Minuten pro Monat (das entspricht ungefähr 2 ppm) reduziert werden.

3.2 Software

In diesem Abschnitt werden nun die verschiedenen Module der SmartFridge-Firmware vorgestellt. Begonnen wird dabei mit dem Betriebsystem, da durch dieses die zentrale Verwaltung

aller Modulfunktionen stattfindet. Aufgrund des Umfangs der Firmware können dabei nicht alle Treiber und Funktionen explizit behandelt werden, vielmehr soll die Funktionalität des Gesamtsystems herausgearbeitet werden.

Die nachfolgende Auflistung zeigt eine Zusammenfassung der Kernaufgaben, die mit der Firmware realisiert werden.

- Datenerfassung
- Datenspeicherung
- Terminal (über serielle Schnittstelle)
- Steuerung des Bedienteils

Abbildung 3.17 zeigt hierzu grafisch die Zusammenhänge zwischen den Funktionsmodulen für die zuvor aufgelisteten Kernaufgaben. Die Pfeilspitzen geben dabei die Richtung des Datenflusses an.

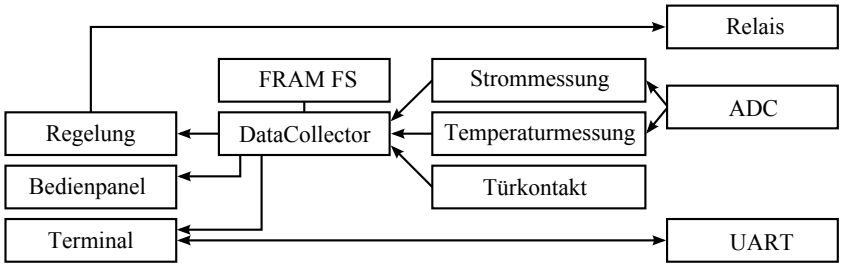

Abbildung 3.17: Zusammenhänge zwischen den Funktionsmodulen

Einen zentralen Punkt nimmt hierbei das Modul *DataCollector* ein, in dem alle Daten erfasst, gespeichert (FRAM Dateisystem) und für die Weiterverarbeitung zur Verfügung gestellt werden. Wie zuvor schon angsprochen, wird auch in dieser Grafik nur der wichtigste Teil der Treiber und Module dargestellt.

Zum Schluss dieses Abschnittes wird auch noch kurz auf den Bootloader eingegangen, mit dem die Firmware über das Terminal aktualisiert werden kann.

3.2.1 Echtzeitbetriebssystem

In der Einführung zu diesem Abschnitt wird verdeutlicht, dass sich die zu implementierende Software aus verschiedenen Teilaufgaben zusammensetzt, die zudem auch unterschiedliche Anforderungen an die Priorität ihrer Ausführung haben.

Einfache Softwaresysteme (speziell bei Mikroprozessoren) können mit einem sogenannten *Foreground/Background*-System realisiert werden, bei der in einer endlos laufenden Hauptschleife (Abbildung 3.18 links) alle notwendigen Funktionen regelmäßig aufgerufen werden (diese Endlosschleife entspricht der *Background*-Schicht). Treten Interrupts auf (z.B. durch einen internen Timer oder den Empfang eines Zeichens auf der seriellen Schnittstelle), so werden diese sofort behandelt, unabhängig davon welche Funktion in der Endlosschleife gerade abgearbeitet wird. Die *Background*-Schicht wird sinngemäß oft auch *Task-Level* und die *Foreground*-Schicht *Interrupt-Level* genannt.
Hier sind sofort einige Nachteile erkennbar. So ist es notwendig, dass gewisse Funktionalitäten, die eine hohe Priorität aufweisen, sofort in der Interrupt-Service-Routine (*ISR*) bearbeitet werden. Dies verhindert für die Dauer der Abarbeitung nicht nur die Ausführung der Funktionen

in der *Background*-Schicht, sondern auch die Bearbeitung weiterer Interrupts auf dem selben Interruptlevel (im Allgemeinen versucht man den Aufenthalt in einer ISR so kurz wie möglich zu halten). Gleichzeitig tritt auch das Problem auf, dass eine Funktion die aus einer ISR heraus angestoßen wird (z.B. durch das Setzen eines Flags) im Worst-Case erst nach einem kompletten Durchlauf der Hauptschleife zur Ausführung kommt.
Praktisch kann man also zusammenfassen, dass das Timing und die Abstimmung verschiedener Aufgaben bei dieser Art der Realisierung nicht ganz trivial ist – zumindest im Fall von etwas komplexeren Systemen.

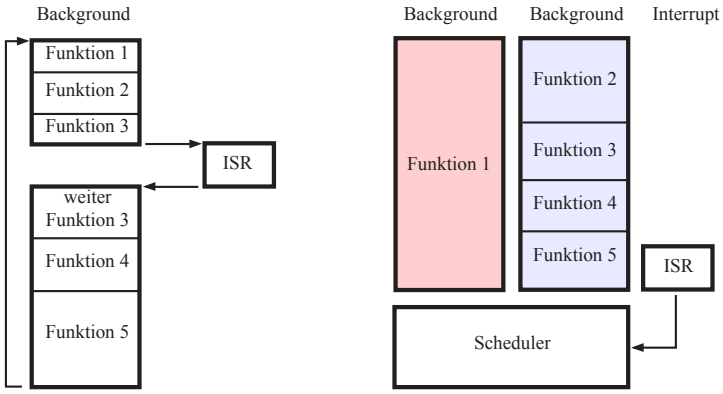

Abbildung 3.18: Unterschied Foreground/Background und Multitasking

Im Gegensatz dazu kann man sich ein Multitasking-System als mehrere parallel laufende *Background*-Level vorstellen (Abbildung 3.18 rechts). Das Software-System wird dabei in Teilaufgaben - die sogenannten Tasks oder auch Threads - geteilt, und von einem Scheduler verwaltet. Durch Multitasking ist es nun möglich die am Prozessor zur Verfügung stehenden Systemressourcen optimal auszunutzen. Der besondere Vorteil besteht auch darin, dass das gesamte System in prinzipiell voneinander unabhängige Teilaufgaben aufgespalten werden kann, was somit die Komplexität des Gesamtsystems verringern kann.

Für Mikroprozessoren sind einige freie RTOS (Real-Time-Operating-System) verfügbar wie z.B. FreeRTOS [68] - besonders empfehlenswert ist aber das hier verwendete μC/OS-II , da die komplette Funktionsweise und Anwendung in einem zugehörigen Buch beschrieben wird [Lab02].
μC/OS-II ist ein Real-Time-Kernel der von Jean J. Labrosse entwickelt wurde und über dessen Firma Micrium [77] vertrieben wird. Der Source-Code von μC/OS-II ist in dem Buch inbegriffen und kann für akademische Zwecke frei verwendet werden. Der Code ist dabei zu einem Großteil in ANSI-C geschrieben und somit leicht portierbar – nur wenige Teile sind in Assembler geschrieben und an den jeweiligen Prozessor anzupassen. Im Internet finden sich viele fertige Portierungen des Kernels – beginnend von einfachen 8-Bit Prozessoren (8051) bis zu 64-Bit Prozessoren oder auch DSPs.

Bei μC/OS-II handelt es sich um einen sogenannten präemptiven Kernel, bei welchem immer der Task mit der höchsten Priorität die Kontrolle über den Prozessor bekommt. Welcher Task als nächstes ausgeführt wird, bestimmt dabei ein Teil des Kernels, der Scheduler genannt wird. Dort erfolgt bei einem sogenannten *Context-* oder auch *Task-Switch* – bei dem die Inhalte der Prozessregister gesichert und geladen werden – die Überprüfung der Taskprioritäten und gegebenenfalls der Wechsel zu einem Task mit höherer Priorität.

Obwohl die Verwendung eines Echtzeitbetriebssystems das Design einer komplexeren Software gegenüber einem reinen Foreground/Background-Design erleichtert, ist es in der Praxis nicht

ganz so trivial. Die Anforderung an die einzelnen Tasks kann dabei grob in solche mit einem *Soft-Realtime*-Verhalten und solche mit einem *Hard-Realtime*-Verhalten unterteilt werden. Solche Tasks die ein Hard-Realtime Verhalten fordern, müssen so schnell und sobald wie möglich ausgeführt werden (z.B. direkt nach dem Auftreten eines Interrupts, der eine Semaphore setzt), während bei der Soft-Realtime Anforderung keine so enge Vorgaben der Abarbeitung gestellt werden.

Als praktisches Beispiel für eine Soft-Realtime-Anforderung kann hier die Realisierung der Terminal-Funktionalität herangezogen werden. Wird ein Zeichen über die serielle Schnittstelle erkannt, so wird dieses Zeichen in der ISR in einen Buffer geschrieben und eine Semaphore gesetzt. Im Terminal-Task wird auf diese Semaphore gewartet und danach die Abarbeitung der im Empfangsbuffer befindlichen Zeichen begonnen. Hierbei spielt es aber für den Benutzer im Allgemeinen keine Rolle, ob sich die Antwort auf die Eingabe 10 ms verzögert - diesem Task kann also eine eher niedrige Priorität zugewiesen werden.

Die Zuweisung der Taskprioritäten ist in der Praxis aber nicht immer so leicht wie im gerade beschriebenen Fall der Terminal-Funktion. Vor allem dann, wenn Tasks in Abhängigkeit zueinander stehen oder Daten miteinander ausgetauscht werden sollen. Hier kommt zum wichtigen Themengebiet der Synchronisierung bzw. dem Datenaustausch zwischen Tasks, die auf mehrere Arten erfolgen kann. Grundsätzlich geht es hier um Probleme der Art, dass zwei Tasks auf die selbe Datenstruktur im Speicher zugreifen müssen oder, dass ein Task auf das Ergebnis eines anderen Tasks warten muss. Das Betriebssystem stellt hierzu verschiedene Möglichkeiten zur Interprozesskommunikation und Synchronisierung bereit, wie etwa Semaphoren oder Message Queues.

Eine weitere Möglichkeit ist das Deaktivieren des Schedulers oder der Interrupts, was vor allem bei sehr kurzen zu schützenden Codeabschnitten eine wichtige Möglichkeit zur Synchronisierung ist. Als Beispiel kann hier der Zugriff auf eine einzelne Variable genannt werden, wo der Overhead für die Verwaltung einer Semaphore wesentlich größer wäre als die Zeitdauer, die der Interrupt oder Scheduler deaktiviert ist. Der Unterschied zwischen dem Deaktivieren der Interrupts und des Schedulers ist der, dass im ersten Fall keine weiteren Interrupts mehr erfasst werden, im zweiten Fall jedoch schon. In beiden Fällen bleibt der aktuelle Task aber solange aktiv, bis er die Kontrolle von sich aus wieder zurück an den Scheduler gibt. Genau dieser Punkt macht diese Methode der Synchronisation allerdings sehr anfällig für Fehler (bei der Programmierung).

Listing 3.1 zeigt ein Beispiel zum typischen Aufbau eines Tasks.

```
1 void DummyTask (void *pdata) {
2   while (1) {
3     Warten auf Ereignis (Semaphore, Event-Flag, ...)
4     ...
5   }
6 }
```

Listing 3.1: Beispiel zum allgmeinen Aufbau eines Tasks

Der Funktionsparameter *void *pdata* dient beim Einhängen in den Scheduler zur Übergabe von beliebigen Variablen oder auch Datenstrukturen an den Task. Die eigentliche Taskfunktionalität läuft im allgemeinen Fall innerhalb einer Endlosschleife (*while (1)*) ab. Es ist zwar auch möglich Tasks während der Laufzeit zu beenden oder überhaupt nur einmal auszuführen, dies ist aber gerade bei eingebetteten Systemen dieser Art eher selten der Fall (und wird hier auch nicht benötigt).

Danach wird im Task auf ein Ereignis gewartet, wofür verschiedene Funktionen durch das Betriebssystem zur Verfügung gestellt werden. Würde man keine dieser Funktionen in der Endlosschleife zur Synchronisierung benutzen, so könnte der Task nur von einem höher prioren Task unterbrochen werden.

Tabelle 3.7 listet die laufenden Tasks mit Taskname, Priorität und Stackgröße auf.

Tabelle 3.7: Auflistung der laufenden Tasks

Task	Prio.	Stack	Beschreibung
DColl	8	256 B	Datenerfassungsmodul Hier findet die Verwaltung von Temperatur-, Luftfeuchtigkeits- und Strommessung statt
Keys	10	128 B	Tastenbehandlung für die Bedieneinheit
STimer	13	128 B	Software-Timer Modul frei verfügbare Timer mit einer Auflösung von 1 s
Beep	14	128 B	Erzeugung der Tonfolgen
Display	15	128 B	Anzeige der aktuellen Informationen an der Bedieneinheit
SFCtrlPanel	19	128 B	Funktionsmodul um die Verbindung zwischen Eingabe und Ausgabe am Bedienfeld herzustellen
Door	22	64 B	Regelmäßige Überprüfung des Türkontaktes
SerTerm	24	256 B	Serielles Terminal zur Konfiguration und Abfrage von Daten
System	30	384 B	System-Task - stellt die „Hauptschleife“ dar Von hier werden alle anderen Tasks eingerichtet

3.2.2 Ereignisspeicher

Da es sich beim SmartFridge um ein System handelt, das die meiste Zeit unbeaufsichtigt seine Arbeit durchführt, ist es sinnvoll eine Möglichkeit zu haben verschiedenste Ereignisse zu protokollieren. Unterschieden wird dabei zwischen externen Ereignissen, wie dem Öffnen der Kühlschranktür und Warnungen/Fehler, die beim Ablauf der Firmware auftreten. Gerade das Protokollieren von Fehlern ist ein wichtiges Hilfsmittel um sie identifizieren und beheben zu können.

Das Software-Modul für den Ereignisspeicher implementiert dazu zwei getrennte Listen: ein Logfile für die Ereignisse im Programmcode und eine Eventliste für die externen Ereignisse. Diese Trennung in zwei Listen erfolgt in Hinblick darauf, für wen die Ereignisse relevant sind. Für den normalen Nutzer ist es zwar wichtig zu wissen ob ein Kompressor blockiert hat oder die Lampe defekt ist, aber nicht, ob es ein Problem beim Zugriff auf Daten im Dateisystem gab.

Tabelle 3.8 zeigt die Felder mit den Datentypen, die für die beiden Listen zur Verfügung stehen. Man sieht, dass sie sich nur in der Bezeichnung des letzten Feldes unterscheiden. Im Feld *timestamp* wird das aktuelle Datum als UNIX-Timestamp (Sekunden seit dem 1.1.1970) abgespeichert, im zweiten Feld wird der Typ des Ereignisses eingetragen. Eine Auflistung aller Typen findet sich im Anhang 7.7. Das letzte Feld benutzt den selben Datentyp, unterscheidet sich aber in der Bedeutung. Im Fall des Logfiles wird dort die Zeile des Auftretens eingetragen, wodurch in Kombination mit dem Fehlertyp der gesuchte Bereich im Programmcode sehr schnell gefunden

Tabelle 3.8: Datentypen und Feldbezeichungen der Ereignislisten

Liste	Datentypen
Eventliste	INT32U timestamp
	INT16U type
	INT16U value
Logfile	INT32U timestamp
	INT16U type
	INT16U line

werden kann. Bei der Eventliste ist der letzte Eintrag optional und kann für eine zusätzliche Information verwendet werden.

Gespeichert werden die Daten im FRAM und im RAM jeweils in zwei verschieden Dateien bzw. Strukturen. Im FRAM sind dazu jeweils 64 Einträge (das entspricht 512 Byte) vorgesehen, im RAM stehen für das Logfile 16 Einträge und für die Eventliste 64 Einträge zur Verfügung.

3.2.3 ADC-Treiber

Der ADC bietet die Wahl zwischen mehreren Betriebsmodi (siehe auch Abschnitt 3.1.5). Neben einer Einzelmessung über einen einzelnen Eingang des ADCs (*One-Shot*) können auch Mehrfachmessungen durchgeführt werden, bei dem vom ADC automatisch der Reihe nach ADC-Kanäle erfasst werden. Der Modus kann über Funktionen des ADC-Treibermoduls konfiguriert werden.

Der hier verwendete Modus ist der sog. *Repeat Sweep 0*-Modus [Ren06] bei dem durchgehend eine Messung von allen acht Eingängen des gewählten ADC-Ports erfolgt. Konfiguriert wird der ADC so, dass bei einer Auflösung von 10 Bit für jede Messung 8.25 μs benötigt werden. Bei acht verwendeten Eingängen steht also für jeden Kanal jeweils nach 66 μs ein neuer Wert bereit.

Jeder der ADC-Kanäle besitzt im Mikroprozessor ein eigenes Register, über das der letzte gemessene Wert ausgelesen werden kann. Der besondere Vorteil bei diesem ADC-Modus besteht darin, dass nicht explizit eine Messung angestoßen und auf das Ergebnis gewartet werden muss. Allerdings muss darauf geachtet werden, dass die Zeitdauer bis ein neuer Wert zur Verfügung steht ausreichend kurz für die Messaufgabe ist.

3.2.4 Auswertung der NTC-Sensoren

Wie schon bei der Erläuterung der Sensortypen für die Temperaturmessung (Abschnitt 3.1.6) angesprochen, wird hierbei eine tabellarische Auswertung der Kennlinie durchgeführt. Dazu werden bestimmte Punkte auf der Kennlinie gespeichert und dem Messwert aus dem Analog-Digitalwandler gegenübergestellt. Für jede Messung wird dann ermittelt zwischen welchen Punkten sich der gemessene ADC-Wert befindet und die zugehörige Temperatur über eine Interpolation ermittelt.

Um die Berechnung im Mikroprozessor einfach und schnell durchführen zu können, sind die einzelnen Referenzpunkte auf der Kennlinie äquidistant zueinander gespeichert - konkret wurde ein Temperaturschritt von 5 °C verwendet.
Abbildung 3.19 zeigt zu einer der Sensorcharakteristiken (B=3950, siehe Abschnitt 3.1.6) die

gespeicherten Referenzpunkte, die über Linien miteinander verbunden sind. Durch die größere Steigung der Kennlinie im oberen und unteren Temperaturbereich fallen dort Referenzpunkte sehr dicht zusammen.

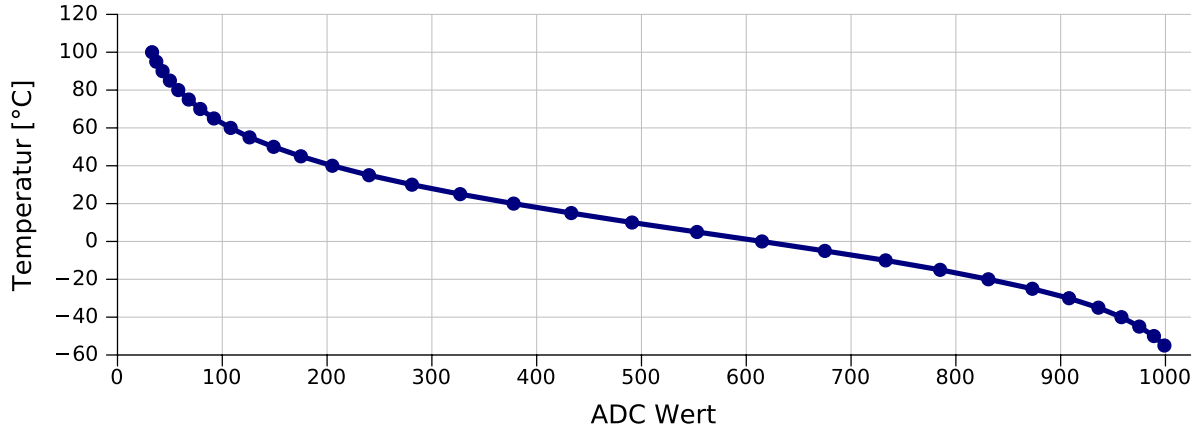

Abbildung 3.19: Referenzpunkte für die Interpolation bei der NTC Auswertung

Wie im Hardwareteil zu den NTC-Sensoren besprochen, wurde die Widerstands-Spannungs-Wandlung so ausgelegt, dass für den geforderten Temperaturbereich der Eingangsbereich des Analog-Digital Wandlers so gut (und linear) wie möglich ausgeschöpft wird (siehe Abbildung 3.10 auf Seite 25).

Im Softwaremodul für die Auswertung der NTC-Sensoren steht für jeden der drei Sensortypen eine eigene Tabelle im Speicher bereit, in der im Abstand von 5 °C die zugehörigen ADC-Werte hinterlegt sind. Der Zugriff auf diese Tabellen und die Berechnung der Zwischenwerte erfolgt über einfache Funktionen, die als Parameter den ADC-Wert enthalten.
Um den Berechnungsaufwand für den Mikrocontroller zu reduzieren, wird eine Festkommadarstellung verwendet. Alle Temperaturwerte in den Tabellen und in den Interpolationsformeln werden oder sind mit dem Faktor 10 multipliziert, wodurch die Temperatur auf ein Zehntel Grad genau gespeichert werden kann.

Die Strukturen für die Speicherung der tabellarischen Form der Temperaturwerte zeigt Listing 3.2.

```
1  typedef struct {
2    INT16S temp;
3    INT16U adc;
4  } st_ntc_data;
5
6  typedef struct  {
7    char *name;
8    INT8U entrys;
9    st_ntc_data *ntc_data;
10   INT8S deltat;
11 } st_ntc_data
```

Listing 3.2: Datenstrukturen zum Speichern der NTC-Tabelle

Listing 3.3 zeigt dazu beispielhaft einen Ausschnitt aus den Daten für den in3.19 dargestellten Verlauf.

```
st_ntc_data ntc_data_b39504k7[32] =
                    {{-550, 999},
                     {-500, 989},
                     {-450, 975},
                     ...
                     { 950, 37},
                     {1000, 33}};

st_ntc_data_table ntc_table_b39504k7 =
                    {"B3950 4k7",       // Bezeichnung
                    32,                 // Anzahl Datenpunkte
                    ntc_data_b39504k7,  // Tabelle
                    50};                // Temperaturschritt (*10)
```

Listing 3.3: Beispiel einer NTC-Tabelle

Um nun für einen über den ADC ermittelten Messwert die zugehörige Temperatur zu bestimmen, muss ausgehend von den Referenzwerten eine Interpolation durchgeführt werden. Dies soll an einem Beispiel erläutert werden.
Ausgangspunkt ist hier ein gemessener Wert von 2.19 V, was dem ADC-Wert von 450 entspricht. Abbildung 3.20 zeigt hierzu den relevanten Ausschnitt aus der Charakteristik nach Abbildung 3.19. Der ermittelte Wert liegt also zwischen den beiden Referenzpunkten mit den ADC-Werten von 433 (Punkt a) und 491 (Punkt b) und somit im Temperaturbereich zwischen 10 °C und 15 °C.

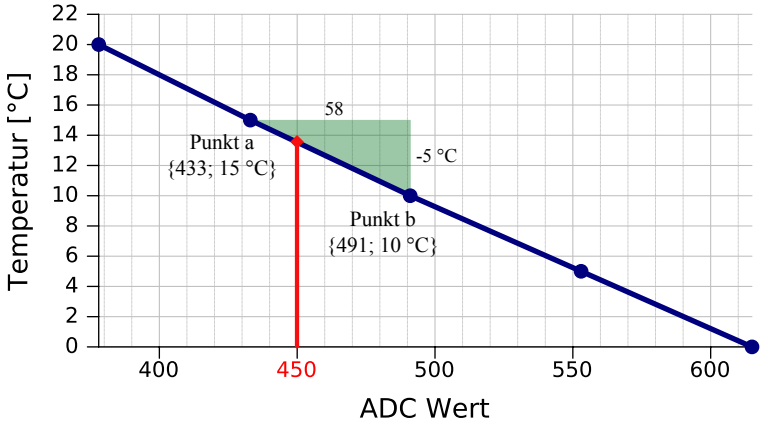

Abbildung 3.20: Beispiel zur Interpolation in der NTC-Tabelle

Die hier markierten Referenzwerte sind jedoch nach der Ermittlung des Messwertes noch nicht bekannt und müssen erst berechnet werden. Dazu kommt ein binärer Suchalgorithmus [HR06] zum Einsatz. Grundidee dahinter ist das Teilen und Vergleichen der Suchmenge (dieses Schema wird auch als *Teile und Herrsche* bezeichnet [Log10]).
Abbildung 3.21 zeigt ein vereinfachtes Beispiel zu diesem Prinzip, bei dem festgestellt werden soll, zwischen welchen Referenzwerten sich der Wert 75 befindet. Die Werte in den Rechtecken stellen dabei den Inhalt dar nach denen verglichen werden soll (entsprechen also den ADC-Werten in der NTC-Tabelle) und die Zahlen darüber den Index des Eintrages. Die farbig markierten Felder zeigen die Grenzen der betrachteten Menge - grün für den Beginn der Menge und rot für das Ende.

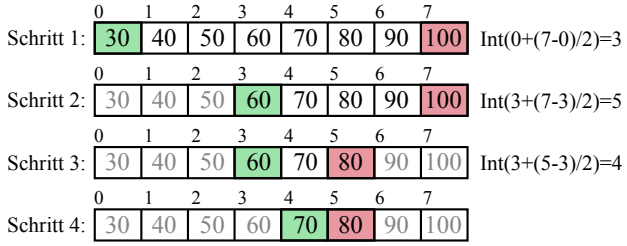

Abbildung 3.21: Binäre Suche zur Bestimmung der Stützpunkte der NTC-Tabelle

Das Halbieren der Menge ist im ersten Schritt sehr simpel, da noch der gesamte Bereich betrachtet wird. Da nur mit ganzen Zahlen gerechnet wird (entsprechend der Bearbeitung im Mikroprozessor) ergibt sich dabei der Indexwert 3 ($\lfloor 7/2 \rfloor = 3$). Nun wird der Wert im Feld von Index 3 (60) mit dem gesuchten Wert 75 verglichen. Ist der gesuchte Wert kleiner, so muss das Ende der Menge an die Position von Index 3 gesetzt werden, ist er größer dann wird der Beginn der Menge verschoben.
Im nächsten Schritt erfolgt die Halbierung der Menge von Index 3 bis Index 7, es wird also der Wert am Index 5 zum Vergleich verwendet. Da der Wert in diesem Feld größer als 75 ist, muss entsprechend der vorherigen Beschreibung das Ende der Menge zum Index 5 hin verschoben werden. Dieses Teilen und Vergleichen wird so lange fortgesetzt, bis sich Anfang und Ende der Menge nur um ein Feld unterscheiden. Somit sind die beiden Referenzwerte gefunden.
Der Spezialfall, dass der gesuchte Wert mit einem Referenzwert übereinstimmt, wurde hier in dieser Beschreibung nicht explizit behandelt, erfordert aber nur einen zusätzlichen Vergleich von gesuchtem Wert und dem Inhalt des aktuell betrachteten Feldes.

Wie in der Abbildung 3.20 ersichtlich ist, liegen die gesuchten Referenzwert für den ADC-Wert von 450 an den Positionen 433 und 491 mit den Temperaturwerten 15°C und 10°C. Wenn bekannt ist, zwischen welchen Punkten sich der gesuchte Wert befindet, kann über eine einfache Interpolation die zugehörige Temperatur errechnet werden. Dazu muss zuerst die Geradensteigung bestimmt werden, die dann mit dem Abstand zum ersten Referenzpunkt multipliziert wird. Durch Addition der Temperatur des ersten Referenzwertes erhält man den gesuchten Wert. Als Beispiel wird dies für einen ADC-Wert von 450 gezeigt (Formel 3.4).

$$\begin{aligned} \text{Steigung m} &= \Delta y/\Delta x = \frac{b_y - a_y}{b_x - a_x} \\ \text{Temperatur}_{\text{x=450}} &= a_y + m \cdot (v_x - a_x) \\ &= 15°C + \frac{10°C - 15°C}{491 - 433} \cdot (450 - 433) \\ &= 13.53°C \end{aligned} \tag{3.4}$$

3.2.5 Temperatur- und Feuchtigkeitsmessung mit dem SHT11-Sensor

Wie schon bei der kurzen Hardwarebeschreibung dieses Sensors in Abschnitt 3.1.6 erwähnt, erfolgt die Kommunikation über eine digitale, serielle Schnittstelle, dessen Protokoll stark an I2C angelehnt ist und mit maximal 10 MHz Taktfrequenz betrieben werden kann. Eine Messung von Temperatur oder Luftfeuchtigkeit läuft so ab, dass man dem Sensor den entsprechenden Befehl

schickt und nach einer bestimmten Wartezeit (die der Mess- und Auswertedauer der integrierten Elektronik entspricht) die Resultate von bestimmten Registern aus dem Sensor ausliest.

Ein Register dem eine besondere Bedeutung zukommt, ist das Status-Register, in dem verschiedene Einstellungen vorgenommen werden können. So kann die Auflösung der Messwerte, die standardmäßig auf 14 Bit für die Temperatur und 12 Bit für die relative Luftfeuchtigkeit eingestellt ist auf 12 Bit und 8 Bit herabgesetzt werden. Dies kann dann sinnvoll sein, wenn die Mess- und Auswertedauer reduziert werden muss. So stehen die Messdaten bei einer Auflösung von 14 Bit nach maximal 320 ms bereit, während das Ergebnis bei 12 Bit bereits nach 80 ms ausgelesen werden kann. Bei 8 Bit sind es gar nur mehr 20 ms.
Der Prozessor muss hierbei natürlich diese im Datenblatt angegebene maximale Zeit nicht einfach abwarten, sondern bekommt die Fertigstellung der Erfassung vom Sensor mitgeteilt.
Damit Störungen auf der Leitung zum Sensor das Messergebnis nicht verfälschen bzw. unbrauchbar machen, kann der übermittelte Sensorwert mittels einer CRC-Prüfsumme verifiziert werden.

Beim Auslesen liefert der Sensor Rohwerte, die erst weiterverarbeitet werden müssen. Dazu stehen vereinfachte Formeln zur Verfügung. Für die Berechnung der Temperatur kann die Näherungsformel 3.5 aus [Epc06] verwendet werden.

$$T = d_1 + d_2 \cdot SO_T \tag{3.5}$$

Der Faktor SO_T repräsentiert den ausgelesenen Sensorwert, d_1 und d_2 sind Werte die von Betriebsspannung, Auflösung und Einheit der Temperatur (°C oder °F) abhängen. Bei einer Versorgungsspannung von 5 V und einer Auflösung von 12 Bit beträgt der Wert für d_1 gleich -40 und der Wert für d_2 gleich 0.04.

Für die Berechnung der relativen Luftfeuchtigkeit können die vereinfachten, linearisierten Formeln in 3.6 verwendet werden [Sen06].

$$\begin{aligned} RH_{real,8Bit} &= (a \cdot SO_{RH} + b)/256 \\ RH_{real,16Bit} &= (a \cdot SO_{RH} + b)/4096 \end{aligned} \tag{3.6}$$

Die Konstanten a und b hängen in diesem Fall von der Auflösung (8 oder 12 Bit) und dem gemessenen Sensorwert ab. Je nachdem, ob sich der Sensorwert über oder unter einer bestimmten Grenze befindet, muss für b ein anderer Wert genommen werden. Im Sensirion Datenblatt [Sen06] wird dies als „two segment humidity linearization“ bezeichnet. Tabelle 3.9 zeigt die Werte für 8 und 12 Bit (bzw. 16 Bit, da der Ausgangswert und das Ergebnis in einer 16 Bit breiten Variable gehalten werden).

Tabelle 3.9: Konstanten zur Berechnung der Luftfeuchtigkeit beim SHT11-Sensor

Bereich (8 Bit)	a	b	Bereich (12 Bit)	a	b
$0 \leq SO_{RH} \leq 107$	143	-512	$0 \leq SO_{RH} \leq 1712$	143	-8192
$108 \leq SO_{RH} \leq 255$	111	2893	$1713 \leq SO_{RH} \leq 4095$	111	46288

Beide Formeln, für Temperatur und Luftfeuchtigkeit, stellen aber nur Näherungen dar, die aufgrund der einfacheren Bearbeitung durch den Mikroprozessor gewählt wurden. Abbildung 3.22

zeigt den absoluten Fehler, der bei der Auswertung mit den vereinfachten Formeln gegenüber den polynomialen Formeln zweiter Ordnung aus [Sen06] auftritt. Die zu erkennenden Abweichungen sind für die hier gestellte Aufgabe vernachlässigbar, weshalb die Näherungsformeln 3.5 und 3.6 zur Bestimmung von Temperatur und Luftfeuchtigkeit verwendet werden können.

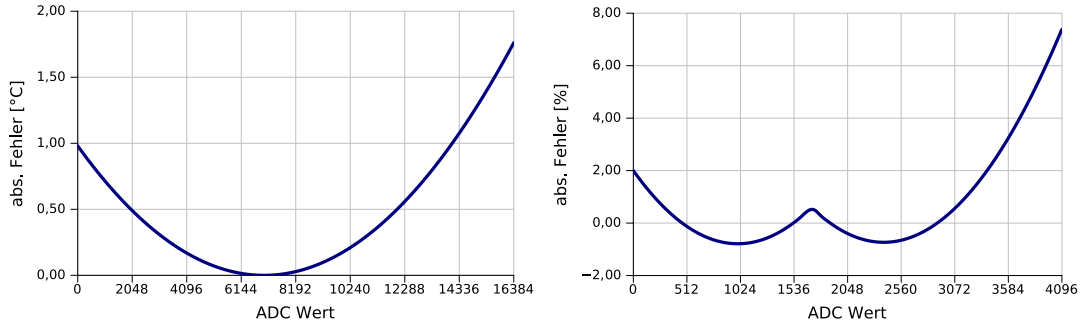

Abbildung 3.22: Absoluter Fehler bei Temperatur und Luftfeuchtigkeit bei linearisierter Auswertung des SHT11-Sensors

3.2.6 Strommessung

Um die Messung des Stroms durchführen zu können, ist es zunächst wichtig festzustellen, was dabei überhaupt gemessen werden soll. Wie schon in Abschnitt 3.1.7 über die Hardware zur Strommessung gezeigt, wird die dem Wechselstrom proportionale Gleichspannung des HY5P Sensors zuerst über einen Vollweggleichrichter gleichgerichtet und auf den Eingangsspannungsbereich des ADCs normiert.
Im Idealfall handelt es sich beim Ausgangssignal um einen reinen (gleichgerichteten) Sinusverlauf, dessen Effektivwert über den sog. Crest-Faktor (Verhältnis von Scheitelwert zu Effektivwert, $\sqrt{2}$) berechnet werden kann. In der Praxis weist das gemessene Signal allerdings aufgrund von Verzerrungen keine saubere Sinusschwingung auf - der Effektivwert muss über die allgemeine Formel 3.7 berechnet werden. Da hierbei kein kontinuierliches Signal vorliegt, sondern eine Abtastung durchgeführt wird, kann Formel 3.8 (gültig für äquidistante Abtastzeitpunkte) zur Berechnung des Effektivwertes verwendet werden [Ler07].

$$I_{eff} = \sqrt{\frac{1}{T} \int_{t_0}^{t_0+T} i^2(t)\, dt} \tag{3.7}$$

$$I_{eff} \approx \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \tag{3.8}$$

Ein kritischer Punkt ist hierbei die *Abtastfrequenz*. Diese muss dem Nyquist-Shannon-Abtasttheorem zufolge mindestens zweimal so groß sein wie die höchste Frequenz des zu erfassenden Signals, um dieses aus den abgetasteten Punkten rekonstruieren zu können. Für die Netzfrequenz von 50 Hz ergibt sich also eine Abstastfrequenz von mindestens 100 Hz. Praktisch sollte diese Frequenz allerdings weitaus höher liegen, um den Signalverlauf korrekt abzubilden.
Für die Messung des Stromverlaufs wird hier eine Abtastfrequenz von 1 kHz verwendet, d.h. es

wird jede Millisekunde ein Wert über den ADC erfasst. Abbildung 3.23 zeigt hierzu den gleichgerichteten Verlauf (rote Kurve) und zugehörige Abtastpunkte. Da keine Nullpunkterkennung durchgeführt wird, liegen die Abtastpunkte bei einer realen Messung allerdings nicht so schön in den Nullpunkten der Sinuskurve, was aber nicht problematisch ist. Es ist nur wichtig, dass die Messung über ein Vielfaches der Periode erfolgt.

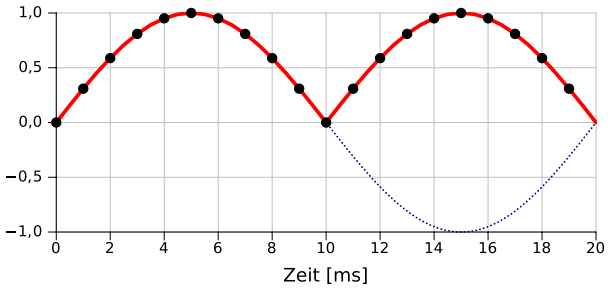

Abbildung 3.23: Abtastung der gleichgerichteten Wechselspannung

Der Programmcode für die Strommessung wurde möglichst modular aufgebaut und stellt zur Steuerung einige Funktionen nach außen zur Verfügung. Über Präprozessorkonstanten kann zudem die Charakteristik der Messung beeinflusst werden, wie etwa das Abtastintervall, die Anzahl der Messperioden, usw.

Nun ist im nächsten Schritt zu klären wie diese Abtastung in der Software implementiert wird. Klar ist, dass dies nicht über einen Task des Echtzeitbetriebsystems realisiert werden kann, da hier die maximale Aufruffrequenz eines Tasks auf 100 Hz eingestellt wird. Obwohl man diese Frequenz theoretisch auf 1 kHz erhöhen könnte (was wiederum eine Erhöhung des Overheads durch das Scheduling bedeuten würde), wäre ein Task kein geeigneter Ort für die Abtastung, da durch das Scheduling und die konkurrierenden Tasks die Abtastzeitpunkte nicht exakt eingehalten werden könnten (man spricht in diesem Zusammenhang auch von einem *Jitter*).
Die Lösung dieses Problems sind die Timer des Mikroprozessors, von denen – wie in Abschnitt 3.1.4 ersichtlich ist – insgesamt 11 zur Verfügung stehen.

Zur Messung wird also ein Timer eingerichtet, der im äquidistanten Abstand von einer Millisekunde den aktuellen Wert für die Strommessung aus dem zugehörigen ADC-Register ausliest. Die ausgelesenen Werte werden in einem Ringbuffer gespeichert und in einem Task regelmäßig ausgewertet. Eine entsprechend große Dimensionierung des Ringbuffers macht diese Auswertung im Task dabei unabhängig vom genauen Zeitpunkt (natürlich nur innerhalb eines von der Größe des Ringbuffers abhängigen Zeitfensters).
Damit der Task seine Arbeit erst dann aufnimmt wenn auch wirklich neue Daten verfügbar sind, stellt das Softwaremodul zur Strommessung nach außen einen Funktionspointer für eine sog. *Callback-Funktion* bereit. Diese ermöglicht es die Schnittstelle zwischen Strommessung und dem Datenerfassungsmodul (siehe Abschnitt 3.2.7) möglichst flexibel zu gestalten und bei Abschluss einer Messung automatisch eine definierbare Funktion aufzurufen. In dieser Funktion wiederum wird der entsprechende Eintrag in den Event-Flags (siehe Abschnitt 3.2.7) gesetzt und somit die Bearbeitung der verfügbaren Daten begonnen.
Abbildung 3.24 zeigt hierzu das prinzipielle Zusammenspiel zwischen dem Timer und dem auswertenden Task (die roten Felder im Ringbuffer stellen ungelesene Daten dar). Der Timer befüllt – wie schon beschrieben – einen Ringbuffer und erzeugt einen Event wenn eine Messreihe abgeschlossen wurde. Im Auswertetask werden die Werte aus dem Ringbuffer ausgelesen, danach wird der Effektivwert nach Formel 3.8 gebildet und in einen Messwertspeicher übertragen.

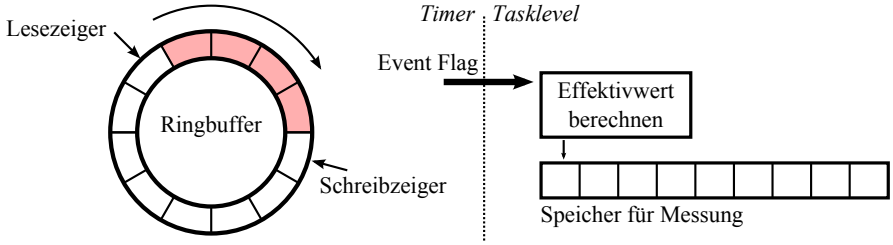

Abbildung 3.24: Zusammenspiel von Timer und Task bei Strommessung

Die Größe dieses Messwertspeichers kann prinzipiell beliebig groß gewählt werden und entspricht praktisch der *Messdauer*. Wird jeweils eine Periode der 50 Hz Schwingung erfasst und abgespeichert, so sind für eine Messdauer von 5 Sekunden 250 Werte zu erfassen ($250 \cdot 20\,\text{ms} = 5\,\text{s}$).

Wie man in Abbildung 3.24 sieht, werden beim Ringbuffer zwei Zeiger verwendet, einer für die Schreibposition und einer für die Leseposition. Wenn keine neuen Daten vorhanden sind, stehen beide Zeiger an der selben Position. Allerdings ist nur das Setzen des Lesezeigers auf die Position des Schreibzeigers ein gültiger Zustand, umgekehrt nicht. Tritt der zweite Fall auf, wurden die verfügbaren Daten noch nicht ausgelesen – der Speicher ist voll.
Um diesen Zustand zu erkennen, wird bei jedem Schreibvorgang eine entsprechende Überprüfung durchgeführt und gegebenfalls ein Flag in der Verwaltungsstruktur der Strommessung gesetzt. Für die konkrete Realisierung wurde eine Ringbuffergröße von 100 Byte gewählt, die in den Tests nie vollständig ausgenutzt wurde.

3.2.7 Datenerfassungsmodul

In diesem Softwaremodul werden alle asynchronen Messungen und auch Messauswertungen durchgeführt. Asynchron bedeutet hierbei, dass eine Messung von einem anderen Task angestoßen und im Haupttask des Datenerfassungsmoduls durchgeführt wird. Dadurch muss der Task, der die Messung beauftragt, nicht auf die Fertigstellung warten und kann seine Bearbeitung unmittelbar fortsetzen. Das Modul bietet aber auch Funktionen an, um die Temperatur direkt, also ohne Einbindung in den Task des Datenerfassungsmoduls, zu messen.
Es laufen hier also die Daten der Strommessung und der Temperatur- und Luftfeuchtigkeitsmessung zusammen. Auch die Datenspeicherung wird über dieses Modul durchgeführt.

Bei der Initialisierung dieses Moduls werden der Reihe nach folgende Punkte durchgeführt:

- Einrichten des ADC-Treibers
- Vorbereiten der Datenstrukturen für die Messungen
- Einhängen der Tabellen für die NTC-Sensoren
- Einrichten von Semaphore und Event-Flags zur Synchronisierung
- Starten des Tasks des Datenerfassungsmoduls

Nach dem Einrichten des Moduls wartet der Task des Datenerfassungsmoduls auf das Auftreten eines Events über den Event-Flag Mechanismus (siehe Abschnitt 3.2.3). Ein Event-Flag stellt praktisch gesehen ein Bitfeld dar, in dem jedes Bit einem Event entspricht. Tabelle 3.10 zeigt die hier verfügbaren Ereignisse.

Einige der Ereignisse können nur in Verbindung mit anderen Ereignissen auftreten. So kann etwa eine Analyse der Serienstrommessung natürlich nur dann durchgeführt werden, wenn auch

Tabelle 3.10: Ereignisflags für die Event-Flags im Datenerfassungsmodul

Event-Flag Aktion	Maske
Temperaturmessung NTC	0x01
Messung Temperatur und Luftfeuchtigkeit SHT11	0x02
Einzelne Strommessung	0x04
Nichtblockierende/asynchrone Serienstrommessung	0x08
Blockierende Serienstrommessung	0x10
Daten speichern	0x20
Serienstrommessung analysieren	0x80

entsprechende Daten vorhanden sind. Der Event-Flag Mechanismus bietet einige Möglichkeiten zur Konfiguration und wurde so eingestellt, dass nach der Annahme eines Ereignisses das zugehörige Bit automatisch zurückgesetzt wird. Dadurch ist es möglich, dass schon während der Abarbeitung der angestoßenen Aufgabe dasselbe Flag von einem anderen Task aus wieder gesetzt wird. Dies kann vor allem dann sinnvoll sein, wenn schon während der Bearbeitung im Datenerfassungsmodul neue Daten bereit stehen. Die Verknüpfung der Ereignisse findet disjunktiv („oder“) statt, d.h. mindestens ein Ereignis muss auftreten, damit der Task die Kontrolle erhält.

Das Kommando zum Speichern der Daten wird dabei als eigenständiges Ereignis realisiert, das aber nur in Kombination mit abgeschlossenen Messungen sinnvoll ist. Diese Trennung von Messung und Speicherung erfolgt aus dem Grund, da für die Temperaturregelung häufiger aktuelle Daten benötigt werden, als für die Datenspeicherung (siehe auch Abschnitt 3.2.8).

Die eigentliche Datenspeicherung erfolgt dann im FRAM und im RAM. Der FRAM stellt insgesamt 32 kByte zur Verfügung, von denen 7380 Bytes für die Datenspeicherung verwendet werden. Dies entspricht bei einem Speicherintervall von 5 Minuten ungefähr einer Woche. Im RAM werden nur die letzten 16 Messungen gehalten, was 80 Minuten entspricht. Der Sinn hinter der doppelten Speicherung ist der schnellere Zugriff auf den aktuellen Verlauf ohne die kompletten Daten vom FRAM auslesen zu müssen. Anwendung findet dies bei der Anzeige der Daten auf der SmartPanel-Hardware (Kapitel 4) und der Darstellung auf der Webseite (Kapitel 5).

Eine weitere Aufgabe des Datenerfassungsmoduls ist, wie schon zuvor angesprochen, die Auswertung der Strommessung (Abschnitt 3.2.6) nach dem Einschalten eines Kompressors. Bei dieser Auswertung werden verschiedene Kennwerte (wie Mittelwert, Maximum, Start- und Endwert) der Messung ermittelt und zur Bewertung eines Fehlerzustandes bei den Kompressoren herangezogen. Wurde ein blockierender Kompressor erkannt, so muss dieser sofort wieder abgeschaltet und ein Fehler akustisch angezeigt. Gleichzeitig erfolgt auch ein Eintrag in das Logfile um eine spätere Auswertung durchführen zu können.

3.2.8 Temperaturregelung

Die Regelung der Temperaturen im Kühl- und Gefrierfach stellt den Kern der SmartFridge-Firmware dar. Fast alle Module dienen praktisch gesehen der Anzeige, Konfiguration oder Speicherung von Daten der Temperaturregelung.

Bevor nun eine Regelung für die Temperatur implementiert werden kann, muss zunächst geklärt werden was überhaupt geregelt wird und welche Charakteristiken hier auftreten. Bekannt

sind dabei die Regelstrecke (Temperatur), das Stellglied (der Kompressor) und das Messglied (Umwandlung der Temperatur in eine Spannung über die NTC-Sensoren). Abbildung 3.25 zeigt hierzu ein allgemeines Blockschaltbild des Regelkreises.

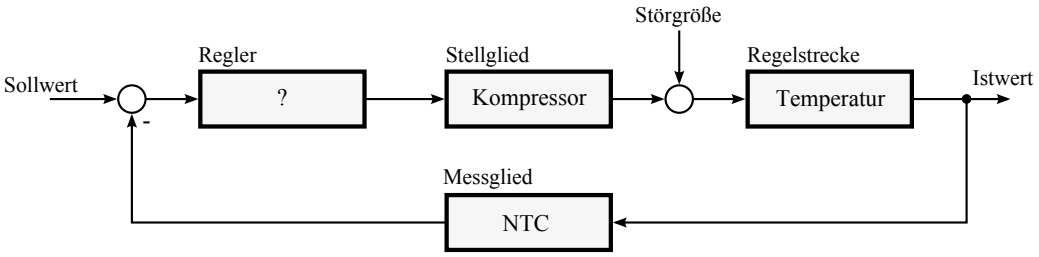

Abbildung 3.25: Allgemeines Blockschaltbild des Regelkreises

Da der Kompressor nicht kontinuierlich geregelt, sondern nur ein- oder ausgeschaltet werden kann, handelt es sich hierbei um eine sog. unstetige Regelung [Haa97, Zac08]. Die Regelstrecke, also die Temperatur innerhalb des Kühl- oder Gefrierbereichs, weist typischerweise ein PT1-Verhalten auf (siehe die Beschreibung des Erwärmungsvorgangs in [Haa97]). Als Regler für diese Aufgabe eignet sich somit ein Zweipunktregler mit Hysterese. Abbildung 3.26 zeigt hierzu das Prinzip für die Zweipunktregelung um einen festgelegten Sollwert.

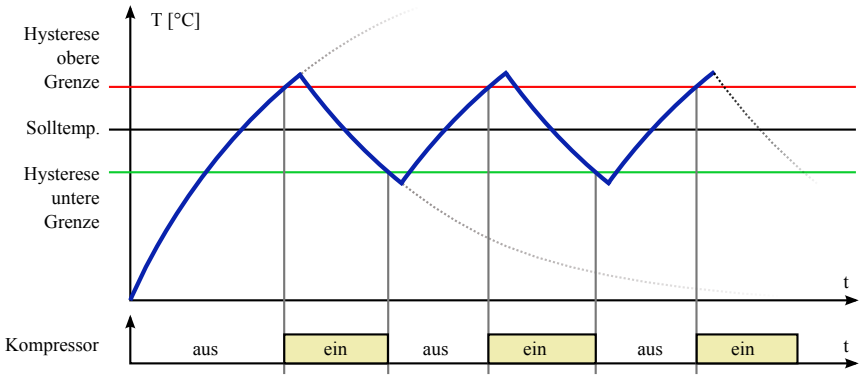

Abbildung 3.26: Prinzip des Zweipunktreglers mit Hysterese

Zu Beginn ist der Kompressor ausgeschaltet, da die Temperatur kleiner als die untere Grenze der Hysterese ist. Nun steigt die Temperatur und überschreitet die obere Grenze der Hysterese, wodurch der Kompressor eingeschaltet wird und eine Abkühlung stattfindet, durch welche die Regeldifferenz abgebaut wird. Unterschreitet die Temperatur die untere Grenze der Hysterese, so wird der Kompressor wieder abgeschaltet. Dieser Vorgang wiederholt sich ständig, es stellt sich also ein periodisches Schwingen um den festgelegten Sollwert ein.

Der Kompressor erfordert die Berücksichtigung einiger zusätzlicher Parameter, die nicht im zuvor angeführten Regelmodell integriert werden können. So darf der Kompressor nach dem Ausschalten erst nach einer bestimmten Zeit wieder eingeschaltet werden. Zudem ist auch die maximale durchgehende Betriebsdauer festgelegt, um eine Überhitzung zu vermeiden. Die Ergebnisse der Strommessung müssen ebenfalls berücksichtigt werden, da damit ein Blockieren des Kompressors erkannt werden kann (siehe Abschnitt 3.2.6). Tabelle 3.11 zeigt die voreingestellten Parameter, die über das Terminal (Abschnitt 3.2.9) auch während des Betriebs angepasst werden können.

Tabelle 3.11: zusätzliche Parameter beim Betrieb des Kompressors

Parameter	Zeit
minimale Ausschaltzeit	8 Minuten
maximale Einschaltzeit *Kühlteil normal*	4 Stunden
maximale Einschaltzeit *Kühlteil Superfreeze*	6 Stunden
maximale Einschaltzeit *Gefrierteil normal*	4 Stunden
maximale Einschaltzeit *Gefrierteil Superfrost*	24 Stunden

Die Realisierung der Regelung im Programmcode sieht nun so aus, dass alle 10 Sekunden folgende Funktionsmodule der Reihe nach abgearbeitet werden.

- Messwerte ermitteln
- Regelung
- Schalten der Kompressoren
- Aktualisieren der Timer

Es erfolgt also zuerst die Messung der aktuellen Temperatur im Kühl- und Gefrierteil. Anschließend startet die eigentliche Regelung, bei der aber noch nicht die Kompressoren direkt geschaltet, sondern nur die entsprechenden Zustandsvariablen in zugehörige Datenstrukturen gesetzt werden. Erst im nächsten Schritt liest das Modul für die Kompressorsteuerung die Zustandsvariablen aus und schaltet die Relais für die Kompressoren. Zum Schluss werden noch die internen Timer für die Überwachung der maximalen Einschalt- und minimalen Ausschaltzeiten aktualisiert.
Diese Trennung in mehrere Funktionsmodule hat zwei Gründe. Zum einen kann somit eine saubere Abtrennung der eigentlichen Regelung erfolgen, wodurch etwa bei Änderungen an der Messung oder dem Schaltverhalten der Kompressoren keine Modifikation am Code für die Regelung erfolgen muss. Zum anderen kann über das Bedienfeld (siehe Abschnitt 3.2.10) und das Terminal (siehe Abschnitt 3.2.9) in die Regelung eingegriffen werden, etwa zum Abschalten des Kühl- oder Gefrierteils. Da für das Ausschalten des Kühlteils keine Regelung notwendig ist, ist es günstig über einen Funktionsaufruf das sofortige Schalten der Kompressoren veranlassen zu können. Nachfolgend werden nun die Punkte der Messung und der Regelung noch näher ausgeführt.

Wie in der obigen Auflistung angeführt, erfolgt alle 10 Sekunden die Ermittlung der aktuellen Messwerte über die Temperatursensoren. Dazu werden im 10-Sekunden-Zeitfenster mehrere Messungen durchgeführt die dann gemittelt werden. Damit ein Öffnen der Kühlschranktür nicht zu einer sofortigen Erhöhung der für die Regelung verwendeten Temperaturen führt, wird zusätzlich noch ein sog. *exponentieller gleitender* Mittelwertfilter angewendet [94], das sehr effizient implementiert werden kann. Formel 3.9 zeigt hierzu die allgemeine Formel und die Vereinfachung mittels Schiebeoperationen. Die Variablen Aus_{neu} und Aus_{alt} repräsentieren den aktuellen und den vorhergenden Wert am Ausgang des Filters, während die Variable *Ein* den Wert am Eingang darstellt. Die Konstante α ist ein Filterparameter mit einem Wert kleiner als eins. Ist α eine Zahl der zweiten Potenz, so kann die Bildung des gleitenden Mittelwertes durch zwei Schieboperationen und einer Subtraktion erreicht werden.

$$Aus_{neu} = Ein \cdot \alpha + Aus_{alt} \cdot (1 - \alpha) \tag{3.9}$$
$$Aus_{neu} = Ein \cdot (1/16) + Aus_{alt} \cdot (1 - (1/16))$$
$$Aus_{neu} = (Ein >> 4) + Aus_{alt} - (Aus_{alt} >> 4)$$

Abbildung 3.27 verdeutlicht die Auswirkung des Parameters N für ein schnelles Ansteigen des Eingangswertes wie es durch das Öffnen der Tür erwartet werden könnte.

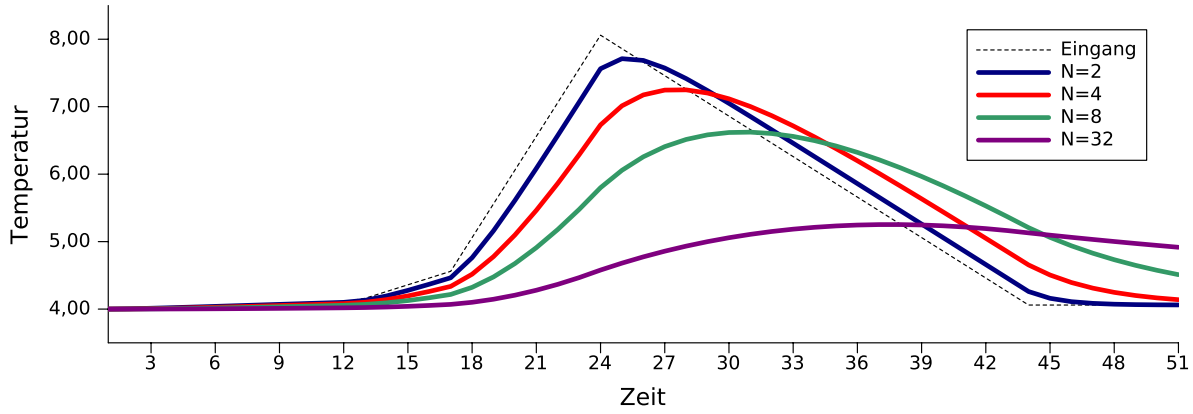

Abbildung 3.27: Beispiel zum Verlauf des exponentiellen gleitenden Mittelfilters

Für die Konstante α wurde im Programmcode der Wert 4 gewählt, da er einen guten Kompromiss zwischen Glättung des Verlaufs und Reaktionsfähigkeit darstellt.

Nachdem über den exponentiellen Filter der Wert für die Weiterverarbeitung bestimmt wurde, kann im nächsten Schritt die Regelung gestartet werden. Algorithmus 3.1 zeigt hierzu einen leicht vereinfachtes Ablauf der Regelung des Kühlteils. Die reale Regelung implementiert noch einige zusätzliche Überprüfungen, die nachfolgend auch kurz erklärt werden.

Der Ablauf für den Gefrierteil entspricht im Großen und Ganzen dem des hier gezeigten Kühlteils. Der Unterschied besteht in der Berücksichtigung der Verdampfertemperatur. Ist dessen Temperatur niedriger als der Sollwert, so kann der Kühlkreislauf noch Wärme aufnehmen und der Kompressor darf nicht eingeschaltet werden.

Wie zuvor angemerkt wurde, sind einige Details nicht in die Abbildung des Ablaufdiagrammes eingeflossen. Dazu gehört unter anderem die Strommessung bei jedem Einschalten eines Kompressors. Da die Kompressoren beim Anlaufen einen hohen Stromverbrauch haben, darf zudem immer nur ein Kompressor zur selben Zeit gestartet werden. Müssen aufgrund der Regelung beide Kompressoren aktiviert werden, so wird der Kompressor des Gefrierteils erst beim nächsten Regeldurchgang gestartet. Nach dieser Verzögerung von 10 Sekunden hat der Kompressor des Kühlteils seinen Betriebszustand erreicht.

3.2.9 Terminal

Die Kommunikation der SmartFridge-Hardware mit externen Geräten erfolgt über eine serielle Schnittstelle auf der ein Terminaldienst aufgesetzt wurde. Dieser Dienst wurde so realisiert und

Algorithmus 3.1 : Regelung des Kühlteils

```
begin Temperaturregelung Kühlteil
    if Kompressor aktiv then
        if maximale Einschaltzeit then
            Kompressor deaktivieren
            Ende der Regelung
    if Superfrostmodus aktiv then
        if maximale Einschaltzeit überschritten then
            Kompressor deaktivieren
            Ende der Regelung
    if Temperatur > (Sollwert + Hysterese) then
        if Verdampfertemperatur > Sollwert then
            if Kompressor aus $$ minimale Ausschaltzeit then
                Kompressor aktivieren
                Ende der Regelung
    if Temperatur < (Sollwert + Hysterese) then
        Kompressor deaktivieren
end
```

ausgelegt, dass eine Konfiguration der SmartFridge-Hardware entweder menschenlesbar durch verschiedene Befehle über eine Terminalsoftware oder über eine weiter abstrahierende Komponente (z.B. eine Applikation für den PC oder auch eine Webseite) erfolgen kann. D.h. der Zugriff kann nicht nur vom SmartPanel aus erfolgen, sondern von jedem PC – gegebenfalls ist dazu ein USB-Seriell-Wandler zu verwenden.

Realisiert wurde dies durch einen simplen Parser, der die über die serielle Schnittstelle des Mikroprozessors erhaltenen Zeichen auswertet, danach die entsprechenden Funktionen aufruft und anschließend gegebenenfalls eine Antwort zurückliefert.
Um dabei einen gewissen Komfort zu erreichen, ist es notwendig bei der Eingabe nicht nur die alphanumerischen Zeichen zu berücksichtigen, sondern auch einige der möglichen Steuerzeichen - wie Backspace um ein Zeichen zu löschen, Enter für die Übernahme des Befehles oder die Pfeiltasten um eine History der vorangegangenen Befehle anzuzeigen.

Die Benachrichtigung über ein neues Zeichen erfolgt durch die Interrupt-Service-Routine der seriellen Schnittstelle. Das empfangene Datenbyte wird in einen FIFO-Buffer abgelegt, wodurch das Zeichen nicht sofort bearbeitet werden muss. Danach wird der Wert einer Semaphore erhöht auf welche im Terminal-Task gewartet wird. Abbildung 3.28 zeigt dies schematisch.
Auf diese Weise kann die Zeit in der Interrupt-Service-Routine möglichst kurz gehalten werden und die eigentliche Auswertung der empfangenen Zeichen kann asynchron und mit niedriger Task-Priorität erfolgen - natürlich ist hier ein entsprechend dimensionierter FIFO-Buffer notwendig. Konkret werden 128 Bytes für den Buffer verwendet, was bei einer Baudrate von 57600 (mit einem Start und einem Stopp Bit) eine Speicherung über eine Dauer von ungefähr 22 ms garantiert. Praktisch gesehen ist der Buffer für den Normalbetrieb mit 128 Bytes sogar überdimensioniert, da keiner der bisher implementierten Terminal-Kommandos mehr als 50 Zeichen (inklusive Optionen und Werten) benötigt.

Für die Übertragung größerer Datenmengen – wie etwa den gespeicherten Datensätzen aus

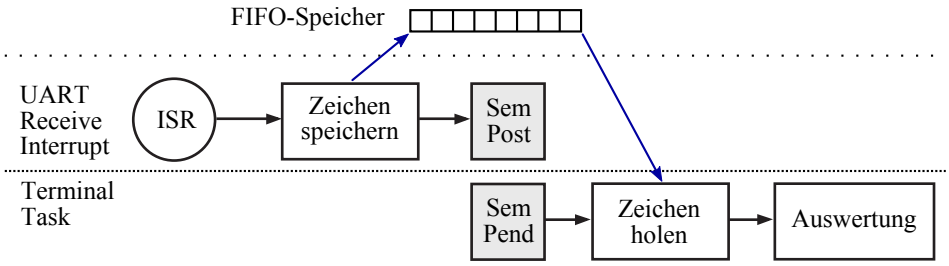

Abbildung 3.28: Verwendung eines FIFO Buffers zwischen UART Interrupt und Terminal Task

dem FRAM – wird zur Übertragung das XModem-Protokoll verwendet [For88]. Durch eine Prüfsummenberechnung steht hierbei auch gleich eine Möglichkeit zur grundlegenden Feststellung von Übertragungsfehlern zur Verfügung. Das Protokoll an sich ist dabei relativ simpel aufgebaut - grundsätzlich ist es ein *Send & Wait*-Protokoll bei dem die Daten und Steuerinformationen in 132 Byte Blöcken vom Sender zum Empfänger übertragen werden. Die Flusskontrolle dabei wird über definierte Steuerzeichen realisiert.

Da die Kommunikation im Normalfall immer zwischen SmartFridge und SmartPanel erfolgen wird, wo es nicht notwendig ist die Daten für Menschen lesbar zu übertragen, wurde zusätzlich die Möglichkeit implementiert beliebige Bereiche aus dem RAM über das XModem-Protokoll zu übertragen. Dies wird zur Übertragung kompletter Datenstrukturen im RAM genutzt - einerseits für Debugging-Zwecke aber auch um aktuelle Daten z.B. von den Sensoren zu übertragen ohne auf den FRAM-Datenspeicher zugreifen zu müssen.

Insgesamt kennt das Terminal 17 Kommandos, von denen die meisten in weitere Unterkommandos unterteilt werden können. Die Grundstruktur der meisten Befehle ist dabei „*Kommando* {*get* | *set*} ...“, d.h. direkt nach dem Kommandonamen erfolgt die Festlegung ob Daten gelesen oder geschrieben werden sollen. Danach folgt die Auswahl der Unterbefehle und gegebenenfalls die Angabe von Parametern und Werten. Dort wo es notwendig ist, weichen Befehle aber von diesem Schema ab. Anhang 7.8 zeigt eine vollständige Auflistung aller verfügbaren Kommandos.

Damit bestimmte Befehle nicht von jedem Benutzer ausgeführt werden können, ist auch eine einfache Benutzerverwaltung implementiert. Es existieren drei hierarchische Benutzertypen: *user*, *admin* und *root*. Voreingestellt ist der Benutzer *user*, über den die meisten Befehle zur Abfrage von Daten ausgeführt werden können. Über ein Kommando kann der Benutzer gewechselt werden, wobei hier auch die Eingabe eine Passwortes notwendig ist.
Die Benutzertypen *admin* und *root* bieten erweiterte Möglichkeiten, wie etwa das Löschen des FRAMs (mindestens *admin* notwendig) oder das Einspielen einer neuen Firmware (*root* Anmeldung notwendig).

3.2.10 Bedieneinheit

Dieser Teil der Firmware ist für die Steuerung der Anzeigen und Auswertung der Tasten an der Bedieneinheit des Kühlschrankes zuständig. Wie schon in Abschnitt 3.1.3 beschrieben, stehen an der Bedieneinheit zwei Anzeigen und mehrere Tasten zur Verfügung.

Die Tastenerkennung wird über eine Zustandsmaschine durchgeführt, die auch eine Unterscheidung zwischen einem langen und kurzen Tastendruck ermöglicht. Dadurch sind vielfältige Tastenkombinationen möglich. Eine Tastenwiederholungsfunktion ermöglicht dabei das periodische

Inkrementieren oder Dekrementieren einer Einstellung. Algorithmus 3.2 stellt hierzu den vereinfachten Ablauf dar.

Algorithmus 3.2 : Tastendruckerkennung

```
begin Tastenerkennung
    repeat
        Aktuellen Tastenzustand speichern
        20 ms warten
        Aktuellen Tastenzustand speichern
    until Gespeicherte Tastenzustände gleich
    repeat
        Timer starten
        while Timerwert < Zeitkonstante langer Tastendruck do
            if Taste losgelassen then kurzer Tastendruck
                Tastedruck eintragen (kurz)
                Tastenbehandlung beenden
            Timer inkrementieren
        Tastedruck eintragen (lang)
    until forever
end
```

Gestartet wird die Tastenerkennung durch einen vom Bedienpanel ausgelösten Interrupt. Im ersten Schritt wird eine einfache Entprellung durchgeführt, indem der aktuelle Zustand der Tasten eingelesen, 20 ms gewartet und anschließend nochmal der Zustand eingelesen wird. Stimmen beide eingelesenen Zustände nicht überein, so wird von einem ungültigen Zustand ausgegangen und die Entprellung wird erneut durchgefüht. War der Vergleich erfolgreich, so wird ein Timer zur Messung der Zeitdauer wie lange die Tasten gedrückt wurden gestartet. Dies dient dazu, um zwischen kurz gedrückten und lang gedrückten Tasten unterscheiden zu können. Hier sei auch festgehalten, dass die Implementierung die beliebige Kombination von Tasten erlaubt, wobei natürlich nur bestimmte Kombinationen einem gültigen Ereignis entsprechen.
Wird die Taste (bzw. die Tastenkombination) nun vor einem bestimmten Timerwert wieder losgelassen, so gilt dies als kurzer Tastendruck. Der Tastenzustand wird daraufhin in einen FIFO-Buffer eingetragen und die Bearbeitung beendet. Wird der festgelegte Timerwert überschritten, so wird ein langer Tastendruck angenommen, der ebenfalls in den FIFO-Buffer eingetragen wird. Allerdings wird danach automatisch in einen *Autorepeat*-Modus gewechselt und der Timer erneut zurückgesetzt.

Ein Detail, das in Algorithmus 3.2 nicht ersichtlich ist, betrifft die Timerwerte für den langen Tastendruck. Beim ersten Durchlauf wird ein anderer Grenzwert als bei den weiteren Durchläufen (Autorepeat) verwendet. Konkret sind für den ersten langen Tastendruck eine Zeitdauer von 1.5 s und für den Autorepeat-Modus 100 ms definiert. Auf diese Weise kann beim Einstellen von Parametern der Wert automatisch um 10 Einheiten pro Sekunde verändert werden.

Wie zuvor schon angesprochen, wird der Tastenzustand in einem FIFO-Buffer eingetragen. Dabei werden aber nicht nur die gedrückten Tasten selbst hinterlegt sondern auch einige zusätzliche Flags, die eine Unterscheidung der Art des Tastendrucks ermöglichen. So existiert auch ein Flag für den Autorepeat-Modus, wodurch es möglich ist, dass manche Tastenkombinationen nicht auf ein Autorepeat, sondern nur auf einen einzelnen langen Tastendruck reagieren. Dies ist etwa dann sinnvoll wenn es um Parameter geht, die zwar mit einem langen Tastendruck geändert

werden, aber nur zwei Zustände annehmen können (etwa „ein“ und „aus“). Ein Autorepeat würde hier zu einem ständigen Wechsel zwischen den beiden Zuständen führen.

Tabelle 7.10 im Anhang (Abschnitt 7.9) zeigt die Tastenkombinationen und zugehörigen Aktionen die hierbei aktuell implementiert sind.

Über einen langen Druck auf die Alarm-Taste kann zusätzlich in einen Setup-Modus gewechselt werden, über den verschiedene Einstellungen auch ohne Verwendung des Terminals (Abschnitt 3.2.9) geändert werden können. Das linke Display zeigt in diesem Modus eine ID des aktuell gewählten Parameters an und das rechte Display den zugewiesenen Wert. Tabelle 7.11 im Anhang listet hierzu wieder die möglichen Tastenkombinationen auf.

Die Steuerung der Anzeigen am Bedienteil wird in einem separaten Task realisiert. Dabei stehen verschiedene Funktionen zur Verfügung um die Zustände oder dargestellten Werte an den 7-Segment-Anzeigen und den LEDs zu konfigurieren. Der zuständige Task aktualisiert nun regelmäßig die Anzeige – dies kann über eine nach außen verfügbare Semaphore aber auch manuell ausgelöst werden. Einen zusätzlichen Betriebsmodus stellt das Blinken der LEDs oder der 7-Segment-Anzeigen dar - hierdurch hat man die Möglichkeit auf Veränderungen hinzuweisen oder auch Fehlerzustände anzuzeigen.

Im normalen Betriebszustand (Kühl- und Gefrierbereich sind aktiviert, es liegt kein Alarm vor und die Anzeige befindet sich nicht im Setup-Modus) wird auf den Anzeigen die aktuelle Solltemperatur von Kühl- und Gefrierbereich dargestellt. Durch einen Druck auf eine bestimmte Taste (siehe Tabelle 7.10) kann aber auch jederzeit die aktuelle Temperatur angezeigt werden.

3.2.11 Erkennung Türöffnung und Lampenprüfung

Obwohl die Auswertung des Türkontakt-Sensors im Prinzip relativ einfach ist, müssen bei der Realisierung einige Punkte beachtet werden. Die Beleuchtung in der Kühl- Gefrierkombination soll grundsätzlich beim Öffnen der Tür eingeschaltet und beim Schließen wieder ausgeschaltet werden. Wird die Tür länger als eine konfigurierbare Zeitspanne (als Standardwert sind 5 Minuten vorgegeben) offen gehalten, so wird die Lampe ebenfalls deaktiviert und eine Fehlermeldung gespeichert.
Die Beurteilung der Lampenfunktionsfähigkeit erfolgt direkt über eine Messung der auf einen Lichtsensor einfallenden Lichtstärke. Um Fehlmessungen infolge von außen einfallenden Lichts zu verhindern, wird dabei diese Messung nur bei geschlossener Tür durchgeführt. Die Algorithmen 3.3 und 3.4 zeigen vereinfachte wie Auswertung des Türkontaktes und die Lichtprüfung realisert werden.

Der erste Block zeigt den Ablaufplan für die Auswertung des Türkontakt-Sensors, die bei einer Änderung des Türkontaktzustandes gestartet wird. Im ersten Schritt wird eine Entprellung durchgeführt, damit eine eventuelles Prellen des Sensors beim Öffnen oder Schließen der Tür nicht falsch interpretiert wird. Dazu wird im Abstand von 20 ms so lange der aktuelle Sensorzustand ausgelesen, bis zwei aufeinanderfolgende Messungen das selbe Ergebnis liefern. Dies stellt eine relativ simple Entprellung dar, die aber für die gestellte Aufgabe völlig ausreichend ist.
Im nächsten Schritt wird geprüft ob sich der Zustand des Türkontaktes auch wirklich geändert hat. Ist dies nicht der Fall, so wurde die Zustandsänderung durch Kontaktprellen des Sensors verursacht und die Auswertung kann beendet werden. Andernfalls werden abhängig vom Zustand der Tür Aktionen gestartet. Wurde die Tür geöffnet (d.h. Zustandsänderung von geschloßen zu

Algorithmus 3.3 : Auswertung Türkontakt

```
begin Tastenerkennung
    Entprellung des Kontaktes
    if keine Zustandsänderung des Türkontakts then
        Beenden
    if Tür geöffnet then
        Licht ein
        Timer ein
    else
        if Licht eingeschaltet then
            Flag zur Lampenprüfung setzen
end
```

Algorithmus 3.4 : Lampenprüfung

```
begin Lampenprüfung
    if Tür geöffnet then
        Türtimer++
        if Licht eingeschaltet then
            if Licht Timeout then
                Licht aus
        Beenden
    if Flag Lichtprüfung gesetzt then
        Prüfung des Lampenzustands
        if Messung nicht erfolgreich then
            Fehler in Log eintragen
    Licht aus
end
```

geöffnet), so wird die Beleuchung eingeschaltet und die Timer für die weitere Auswertung gestartet. Wurde die Tür hingegen geschlossen, so wird bei eingeschalteter Beleuchung ein Flag für die Lichtprüfung gesetzt. Die eigentlich Messung zur Beurteilung der Lampenfunktionsfähigkeit erfolgt im nachfolgend erläuterten Funktionsblock.

In Algorithmus 3.4 wird der Ablaufplan für die regelmäßige Statusprüfung dargestellt. Der Aufruf dieses Funktionsblockes erfolgt dabei in einem Abstand von einer Sekunde. Zuerst wird geprüft ob die Tür der Kühl-Gefrierkombination geöffnet ist. Wenn ja, wird zuerst der Wert eines Zählers erhöht über den die Zeitdauer der Türöffnung erfasst wird. Anschließend wird bei eingeschalteter Beleuchtung überprüft ob die Lampe schon länger als die schon erwähnte maximale Zeitspanne aktiv ist. Ist dies der Fall wird die Lampe ausgeschaltet und eine Fehlermeldung gesetzt.
Die Funktionsprüfung der Lampe kann nur bei geschlossener Tür erfolgen und wird durch das Setzen eines Flags für die Lichtprüfung angestoßen (siehe Beschreibung zum ersten Funktionsblock). Da die eigentliche Ermittlung des Zustandes über eine externe elektronische Schaltung erfolgt, welche die Lichtstärke in eine binäre Entscheidung umsetzt (siehe Abschnitt 3.1.8), braucht hierbei nur der Zustand eines Prozessorpins, an dem diese Schaltung angeschlossen ist, abgefragt werden. Dadurch, dass der Timer beim Schließen der Tür zurückgesetzt und neu gestartet wird, findet die Messung genau eine Sekunde verzögert statt.

3.2.12 Akustische Meldungen

Über den auf der SmartFridge-Hardware vorhandenen *Beeper* können Statusmeldungen auch akustisch mitgeteilt werden. Um eine Unterscheidung von verschiedenen Stati zu erlauben, wurden dazu sieben verschiedene Tonfolgen definiert, die hier vorgestellt werden.
Abbildung 3.29 zeigt den Verlauf der verschiedenen Alarmtypen und den zugehörigen Bezeichnungen im Programm. Bei *CLICK* und *ACK* handelt es sich um einmal abgespielte Töne, während die restlichen für eine länger andauernde Tonfolge geeignet sind. Dies wird durch die strichlierten Linienzüge gekennzeichnet.

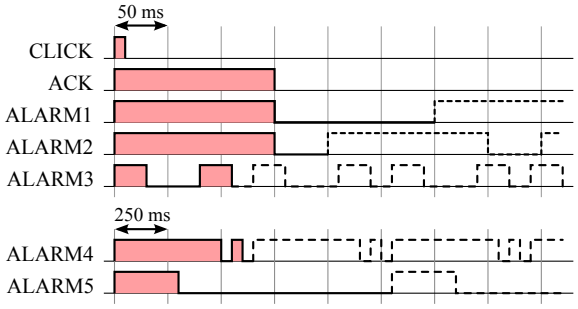

Abbildung 3.29: Tonfolgen für akustische Meldungen

Tabelle 3.12 listet die Verwendung der oben dargestellten Tonfolgen auf.

Tabelle 3.12: Verwendung der verschiedenen Tonfolgen

Tonfolge	Durchläufe	Beschreibung
CLICK	1	Rückmeldung bei gültigem Tastendruck
ACK	1	Bestätigung einer Eingabe
ALARM1	3	Befehl oder Aufgabe kann nicht ausgeführt werden
ALARM2	3	Wechsel in Setup Modus
ALARM3	1	Kindersicherung ein/aus
ALARM4	3	Beginn/Bestätigung bei Änderung Sollwerttemperatur
ALARM5	∞	Alarmzustand

3.2.13 FRAM Dateisystem

Um Konfigurationen und die erfassten Daten bequem speichern und wieder auslesen zu können, wurde für den FRAM-Datenspeicher ein eigenes simples Dateisystem entwickelt.

Folgende Auflistung fasst die Eigenschaften und Merkmale grob zusammen:

- Öffnen, Schreiben und Lesen über einfache Befehle
- Keine Verzeichnisse
- Keine Fragmentierung zulässig
- Fixe Dateigrößen
- Geringer Speicherverbrauch für Verwaltung in RAM und FRAM

Die kleinste Einheit, die in dem Dateisystem für Daten vergeben werden kann, ist auf 64 Byte festgelegt und wird nachfolgend immer als Blockgröße bezeichnet. Da das Dateisystem für eine

maximale Anzahl von 99 Dateien ausgelegt ist, stellt dies einen guten Kompromiss zwischen der Granularität (d.h. wie viel Speicherplatz geht bei einer Datei verloren die nur ein Byte an Daten benötigt) und dem notwendigen Verwaltungsaufwand dar. Bei 32 kB an verfügbarem FRAM-Speicher stehen somit 512 Blöcke mit einer Größe von jeweils 64 Byte zur Verfügung.

Der erste Block des Speichers wird für die Verwaltung des FRAM-Speichers verwendet. Es werden hier Informationen über die Bezeichnung des Speichers (sinnvoll wenn mehrere Bausteine gleichzeitig verwendet werden sollen), die maximale Anzahl der verfügbaren Blöcke, die Anzahl benutzter Blöcke und die Anzahl der Dateien abgelegt. Zu Beginn dieses Blockes ist die Zahl 0x55 als sog. *Magic Number* eingetragen, um erkennen zu können, ob der Speicher bereits formatiert wurde. In das Feld *ptr* wird schlussendlich noch die Position des ersten Datenblockes eingetragen - direkt nach der Formatierung ist dieser Wert auf null und zeigt somit an, dass noch keine Daten vorhanden sind. Tabelle 3.13 zeigt den Inhalt.

Tabelle 3.13: Verwaltungsblock im FRAM Dateisystem

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0x55	devicename										maxblock		used		files
1	files	ptr		reserviert												
2:63	reserviert															

Im zweiten Block wird eine Liste der freien Speicherblöcke verwaltet. Über diese kann das Dateisystem erkennen welche Blöcke bereits belegt sind und welche noch für neue Daten zur Verfügung stehen. Damit für diese Verwaltung nur ein Block benötigt wird, entspricht jedes Bit einem Block, d.h. ein Byte verwaltet 8 Blöcke.

Zur eigentlichen Datenspeicherung kommen zwei unterschiedliche Datenstrukturen zum Einsatz. Die erste Struktur stellt einen Dateiheader dar, der pro Datei einmal vorhanden ist (Tabelle 3.14). In diesem Header wird die Anzahl der verfügbaren Blöcke (*blocks*), die maximal verfügbare Speichergröße (*size*), der Dateiname und auch die Dateigröße (*filesize*) gespeichert. Der Unterschied zwischen *size* und *filesize* besteht darin, dass *size* den aus der Anzahl der Blöcke umgerechneten verfügbaren Speicherplatz darstellt und *filesize* den für diese Datei angeforderten Speicher.
Ein vereinfachtes Beispiel soll dies näher erläutern: Über die entsprechende Funktion des Dateisystems wird eine Datei mit der Größe von 130 Byte angelegt (dieser Wert wird im Feld *filesize* eingetragen), es werden somit 3 Blöcke benötigt, real werden dafür 192 Byte des FRAM-Speichers benötigt - dies entspricht dem Eintrag in *size*.

Tabelle 3.14: Headerblock im FRAM Dateisystem

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	type	blocks		size		next		prev		filename						
1:		filesize		Daten												

Über die Felder *next* und *prev* wird eine doppelt verkettete Liste realisiert über die man von der Wurzel beginnend (dabei handelt es sich um den ersten Dateieintrag) alle Dateien durchgehen kann.

Zwei Besonderheiten, die in der Auflistung zu Beginn dieses Abschnittes schon angeführt wurden, sind die fixe Dateigröße und die nicht zulässige Fragmentierung. Für die praktische Realisierung bedeutet dies, dass eine Datei nur dann angelegt wird, wenn der notwendige Speicher aufeinanderfolgend verfügbar ist. Dies stellt aber für die Verwendung keinen Nachteil dar, da die Größe

der Konfigurations- und Datenfiles fixe Größen aufweisen.
Mit diesen beiden Einschränkungen wäre die verkettete Liste an sich nicht notwendig. Das Dateisystem wurde aber von Anfang an so ausgelegt, dass die Funktion zur Verwaltung von Dateien mit variabler Größe relativ einfach ergänzt werden kann. Hierzu gibt es eine weitere Datenstruktur, die nur zum Verwalten von reinen Datenblöcken (also ohne Dateiinformationen) dient. Zur Unterscheidung der beiden Strukturen dient das Feld *type*. Eine Realisierung dieser Funktion würde so aussehen, dass beim Vergrößern einer Datei eine Datenblockstruktur im Speicher angelegt wird und die Zeiger für *next* und *prev* auf diesen Eintrag hin verändert werden.

Für den Zugriff auf die Daten werden über das Dateisystem verschiedene Funktionen zur Verfügung gestellt, von denen einige nachfolgend aufgelistet werden.

- st_FM31FS_FILE *FM31FS_fopen (char *filename, INT8U mode, INT8U *err)
- INT16U FM31FS_write (st_FM31FS_FILE *fp, void *data, INT16U len, INT8U *err)
- FM31FS_read (st_FM31FS_FILE *fp, void *data, INT16U len, INT8U *err)
- FM31FS_seek (st_FM31FS_FILE *fp, INT16S offset, INT8U origin)
- INT16U FM31FS_tell (st_FM31FS_FILE *fp)

Tabelle 3.15 zeigt abschließend noch die vom Dateisystem verwalteten Dateien.

Tabelle 3.15: Im FRAM verwaltete Files

Dateiname	Beschreibung
sfsetup	Einstellungen für die Regelung
events	Eventliste
err.lst	Fehlerliste / Logfile
passwd	Passwortdatei für Benutzerverwaltung
temp.dat	Datenfile für Temperatur und Luftfeuchtigkeit

3.2.14 Bootloader

Um auch während des Betriebs ein Update der Firmware zu ermöglichen, wurde ein Bootloader geschrieben, der über die serielle Schnittstelle ein Einspielen neuer Firmware ermöglicht. Gestartet werden kann der Bootloader entweder durch Drücken von zwei Tasten am Bedienpanel direkt nach dem Anstecken der Kühl-Gefrierkombination, oder während der laufenden Software über einen Befehl im Terminal.

Zuerst wird kurz der Ablauf beim Starten des Prozessors erläutert. Ausgangspunkt hierbei ist die sog. *fixed vector table*, in der wichtige Sprungadressen für den Prozessor festgelegt sind - für die Erklärung hier ist aber nur der Reset-Vektor relevant. Über diesen wird festgelegt an welche Adresse direkt nach dem Reset gesprungen wird. In Grafik 3.30 ist der Ablauf beim Starten des Mikroprozessors, der nachfolgend auch beschrieben wird, veranschaulicht.

Wie schon zuvor festgehalten, stellt der Reset-Vektor den Einsprungpunkt des Prozessors nach einem Reset dar. Im konkreten Fall verweist dieser Vektor auf die Assemblerfunktion *start*, in der die erste Initialisierung des Prozessors stattfindet. Weiters wird hier auch der (variable) Vektor zur Interrupttabelle gesetzt. Nach der Initialisierung wird die Funktion *startup* im Bootloader-Bereich aufgerufen. In dieser Funktion erfolgen eine weitere Initialisierung des Prozessors und eine Überprüfung, ob entweder die Tastenkombination gedrückt oder der Bootloader über das Terminal aktiviert wurde. Ist dies der Fall, so wird der Bootloader-Code gestartet. Wurden

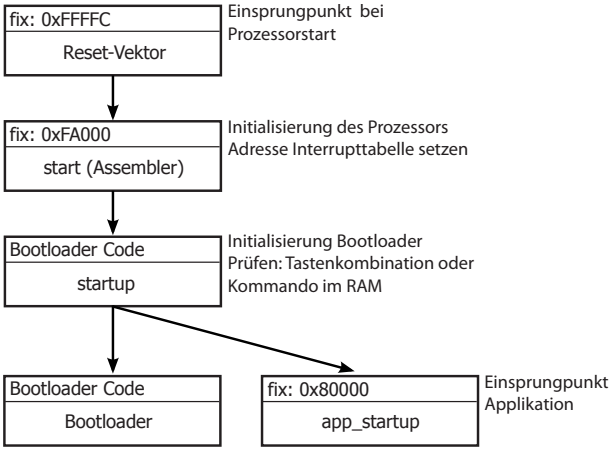

Abbildung 3.30: Ablauf beim Starten des Mikroprozessors mit Bootloader

keine der beiden Möglichkeiten erkannt, so springt der Code in den Applikationsbereich und startet dort die Funktion *app_startup*. Wenn noch keine Firmware eingespielt wurde (d.h. nur der Bootloader im Flash-Speicher des Prozessors vorhanden ist) bewirkt der Aufruf dieser Funktion automatisch eine Verzweigung in den Bootloader-Code.

4 SmartPanel

Das *SmartPanel* soll – wie schon in der Einführung kurz angesprochen – einerseits die Schnittstelle zum Netzwerk darstellen, gleichzeitig aber die Möglichkeit bieten, die gemessenen und aufbereiteten Daten der SmartFridge darzustellen, auszuwerten und gegebenenfalls auch Änderungen an der Konfiguration vorzunehmen.

Folgende Kernaufgaben sollen dabei durch die Hard- und Software realisiert werden:

- Anzeige der Daten
- Einfache Bedienung des Gerätes
- Kommunikation mit der SmartFridge-Hardware
- Langfristige Datenspeicherung
- Netzwerkanbindung

Die Anzeige der Daten und die Konfiguration erfolgt über ein grafisches Menüsystem, wobei die Bedienung natürlich so intuitiv wie möglich sein soll. Aufgrund des kleinen Displays muss hier besonders auf eine einfache aber trotzdem ansprechende Darstellung geachtet werden.
Um die von der SmartFridge übertragenen und hier zu speichernden Daten möglichst flexibel weiterverarbeiten zu können, wird eine Datenbank eingesetzt. Dadurch kann über die Abfragesprache *SQL* beispielsweise direkt eine Analyse der Daten durchgeführt werden können – etwa zum Ermitteln des höchsten Wertes innerhalb eines bestimmten Zeitraumes.
Hauptaufgabe ist natürlich die Anbindung an das Ethernet, wodurch prinzipiell von jedem Gerät im Netzwerk auf das SmartPanel zugegriffen werden kann. Damit der Zugriff unabhängig vom Betriebssystem und ohne Installation einer speziellen Client-Software erfolgen kann, wird das HTTP-Protokoll verwendet, wodurch nur ein Internet-Browser vorrausgesetzt wird. Die Realisierung des softwareseitigen Teils dieses *Webinterfaces* ist allerdings Bestandteil von Kapitel 5.

4.1 Hardware

Die Hardware für das SmartPanel besteht aus einem Basisboard, auf dem ein Modul mit einem ARM9-Prozessor aufgesteckt wird. Dadurch reduzieren sich die Aufgaben des Basisboards auf folgende Punkte:

- Stromversorgung
- Anbindung eines Netzwerkchips über den parallelen Bus

- Steckverbinder für die Schnittstellen (seriell, USB, Netzwerk)
- Steckverbinder für Display/Bedienfeld
- Anbindung einer microSD-Karte

Abbildung 4.1 zeigt hierzu schematisch die Aufteilung der Hardware in Funktionsgruppen.

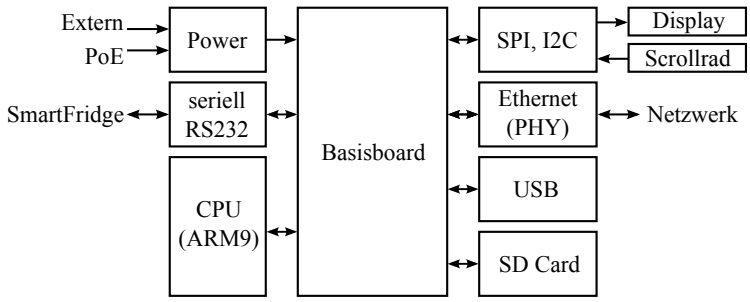

Abbildung 4.1: Aufteilung der SmartPanel-Hardware in Funktionsgruppen

Zur Darstellung der Daten und Konfiguration kommt ein Grafikdisplay zum Einsatz. Geplant war hier anfangs ein monochromes Display mit einer Auflösung von 128x64 Pixel. Durch Recherchen im Internet wurde aber ein Display gefunden, das wesentlich bessere Eigenschaften zu einem günstigeren Preis aufweist – näheres dazu folgt im Kapitel 4.1.4.

Das Prozessormodul stellt eine große Anzahl an Schnittstellen zur Verfügung, von denen aber nur einige benötigt werden. Die folgende Auflistung zeigt eine Übersicht der über das Basisboard verfügbar gemachten Schnittstellen.

- Serielle RS232-Schnittstelle zur Kommunikation
- USB Schnittstellen (jeweils ein Host und ein Device)
- Debug Schnittstelle (über einen Seriell-USB-Wandler)
- Ethernet-Schnittstelle
- SPI-Schnittstellen (Anschluss Display-Platine und Erweiterungen)
- I2C-Schnittstelle (Steuerung der Displayplatine)

Nachfolgend folgt nun eine kurze Beschreibung der Funktionalität von Elementen, die in diesem Kapitel nicht näher erläutert werden.

Bei den USB-Schnittstellen stehen jeweils eine Host- und eine Device-Variante zur Verfügung. Abgesehen davon, dass hierbei zwei verschiedene Typen von Steckverbindern verwendet werden (was an sich keine verlässlichen Rückschlüsse auf die USB-Variante zulässt), unterscheiden sich diese beiden in den möglichen Anwendungsgebieten. Den USB-Host benötigt man um z.B. USB-Sticks zur Speicherung von Daten anschließen zu können, oder um das SmartPanel mittels eines WLAN-USB-Sticks ohne Verkabelung in ein Netzwerk einbinden zu können. Ein USB Device wiederum wird dann verwendet, wenn das SmartPanel an einen PC angeschlossen werden soll.

Die Debug-Schnittstelle wurde über einen Seriell-USB-Wandler nach außen hin ebenfalls als USB-Schnittstelle realisiert, da die meisten PCs keine serielle Schnittstelle mehr zur Verfügung stellen. Die Debug-Schnittstelle ist hauptsächlich für die Entwicklung interessant, gewinnt am fertigen Gerät aber an Bedeutung, weil damit ein Terminal für Konfigurationen am Betriebssystem zur Verfügung steht.
Die SPI- und I2C-Schnittstellen dienen in der aktuellen Ausbaustufe zur Ansteuerung und Kontrolle der Displayplatine. Für beide stehen aber Steckverbinder auf der Platine zur Verfügung, die für spätere Erweitungen benutzt werden können.

Abschließend zeigt Abbildung 4.2 einen Entwurf für das geplante Aussehen des SmartPanels. Dieses Konzept wurde erstellt um schon vor der Realisierung der Hardware einen Eindruck von Größe und Aussehen des Gerätes zu bekommen.

Abbildung 4.2: Konzept für das Aussehen des SmartPanels

4.1.1 Prozessormodul

Herzstück ist ein Modul mit einem *ARM9*-Prozessor, auf dem ein Linux Betriebssystem läuft. Durch den Einsatz eines fertigen Prozessor-Moduls mit integriertem RAM- und Flash-Speicher konnte hier die Entwicklung auf ein Basisboard reduziert werden, das hauptsächlich für die Anbindung des Moduls nach außen sorgt. Durch die Verwendung eines Linux-Betriebssystems erleichtert und verkürzt sich zudem die Anwendungsprogrammierung, da die grundlegenden Funktionalitäten wie z.B. der TCP/IP-Stack von Haus aus verfügbar sind, und nicht erst programmiert, angepasst oder getestet werden müssen.

Bei dem Prozessormodul handelt es sich um das *Stamp9261*-Modul der Firma Taskit [86]. Wie der Name des Moduls schon andeutet sitzt darauf ein AT91SAM9261 von Atmel der mit einem mit 200 MHz getakteten ARM926EJ-S ARM9 arbeitet. Auf einer Platinenfläche von ungefähr 53 mm x 38 mm sind zudem auch noch der RAM-Speicher mit einer Größe von 32 MByte, ein Flash-Speicher mit 16 MByte, ein DataFlash-Speicher von Atmel, weitere Peripherie und alle für die Funktion des Prozessor notwendigen Bauteile untergebracht.
Die Anbindung des Moduls nach außen erfolgt über zwei 100-polige Steckerverbinder, über die neben den Leitungen für die diversen Schnittstellen und Portpins auch der komplette 32 Bit Datenbus verfügbar ist. Abbildung 4.3 zeigt das Modul und kennzeichnet einige wichtige Elemente.

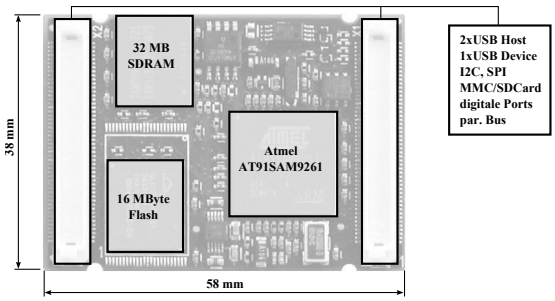

Abbildung 4.3: ARM9 Modul mit AT91SAM9216

4.1.2 Stromversorgung

Die Stromversorgung kann bei der SmartPanel-Hardware über zwei Quellen erfolgen. Zum einen über ein externes Steckernetzteil und zum anderen direkt über das Ethernet über sog. *Power over Ethernet* oder kurz *PoE*. Dies ist ein Verfahren, bei dem Datenleitungen des Netzwerkkabels für die Energieübertragung genutzt werden. Die entsprechende Normung ist dabei in IEEE 802.3af beschrieben [IEE03].

Bei *PoE* wird zwischen zwei Gerätetypen unterschieden: Zum einen gibt es die so genannten *Powered Devices* die kurz als *PD* bezeichnet werden. Diese stellen die eigentlichen Verbraucher dar, welche mit den stromliefernden Einheiten, den *Power Sourcing Equipment* oder kurz *PSE*, verbunden sind. Die über das Netzwerk verfügbare Spannung an den *PDs* beträgt dabei typisch 37 V bei maximal 350 mA, wobei die maximal verfügbare Leistung auf 12.95 W beschränkt ist. Inzwischen wurde mit IEEE 802.3at auch ein Standard festgelegt, bei dem bis zu 25.5 W geliefert werden können.

Zur Gewährleistung der vollen Kompatibilität zum IEEE 802.3af-Standard, wird auf dem SmartPanel ein TPS2375-Baustein als eigenständiger Controller für das Power Device (diese Funktion wird auch als *PD Controller* oder *PDC* bezeichnet) eingesetzt.

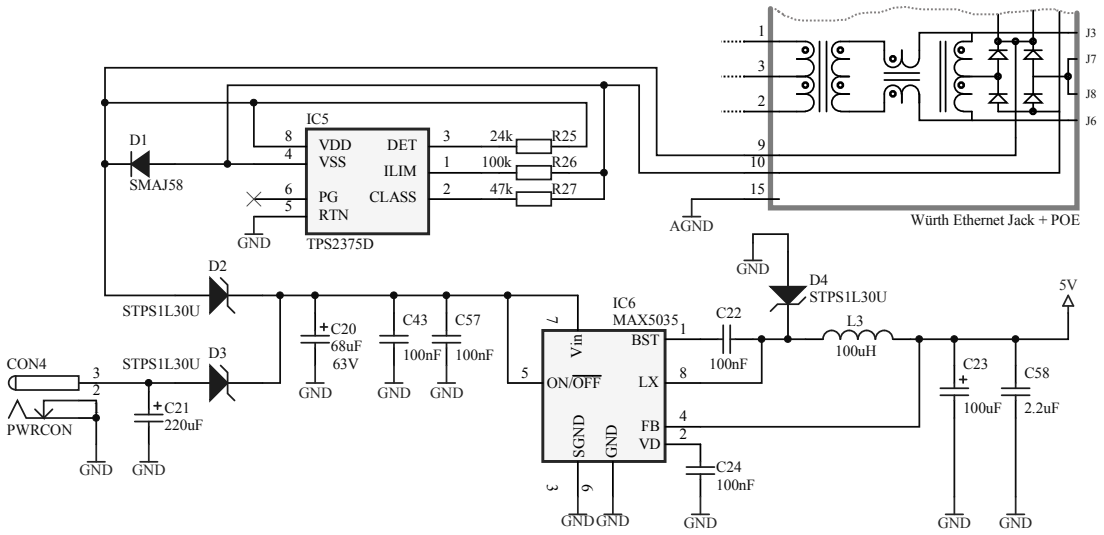

Abbildung 4.4: Schaltung zur Stromversorgung der SmartFridge-Hardware über PoE

Um den alternativen Betrieb von *PoE* und dem externen Netzteil zu ermöglichen, wird vor dem DC/DC-Wandler eine einfache Diodenschaltung vorgesehen, die praktisch gesehen die höhere Spannung „durchlässt“.
Die gewählte Spannung wird anschließend über einen DC/DC-Wandler vom Typ MAX5035 auf die von der Hardware benötigten 5 V konvertiert. Das Prozessormodul wird mit einer Spannung von 3.3 V versorgt, die über einen Linearregler vom Typ LM1117 generiert wird.

4.1.3 Netzwerkchip

Da durch das Prozessormodul keine entsprechenden Anschlüsse für die Netzwerkanbindung (Ethernet) zur Verfügung gestellt werden, musste diese Funktion über einen externen Ethernet-Controller realisiert werden. Angebunden wird dieser Chip (DM9000B vom Hersteller Davicom)

über den 32 Bit breiten parallelen Bus.
Um die Netzwerkschnittstelle auch im Linux-Betriebssytem zur Verfügung zu haben, ist es lediglich notwendig für den Kernel den entsprechenden Treiber zu konfigurieren.

4.1.4 Display

Bei dem verwendeten Display handelt es sich um einen Typ, der ursprünglich für die Mobiltelefone der Serie S65 von Siemens verwendet wurde. Von diesen Displays waren zum Zeitpunkt der Recherchen für diese Diplomarbeit auf dem Markt große Mengen zu günstigen Preisen erhältlich und viele Bastler engagierten sich beim Reverse Engineering um das Display auch in eigener Hardware verwenden zu können.
Ausgangspunkt für die Integration des Displays in die Hardware waren die Arbeiten von Christian Kranz [88], der über *Reverse Engineering* das Kommunikationsprotokoll entschlüsselte.

Das Display besitzt eine Auflösung von 176x132 Pixel, wobei für jedes Pixel maximal 16 Bit für die Farbinformation verwendet werden können – somit können bis zu 65536 verschiedene Farben dargestellt werden. Zusätzlich verfügt das Display auch über eine weiße Hintergrundbeleuchtung – was auch notwendig ist, da aufgrund der verwendeten TFT-Technologie ansonsten selbst bei Tageslicht der Bildinhalt nur schwer erkennbar wäre.
Produziert wurden insgesamt drei verschiedene Typen dieses Displays, wobei sie sich durch den verwendeten Displaytreiber und teilweise auch der Darstellungsqualität unterscheiden. Für die Diplomarbeit kam der Typ LS020 zum Einsatz, der über einen Displaycontroller von Sharp angesteuert wird.

Neben dem günstigen Preis und der guten Verfügbarkeit war ein weiterer sehr wichtiger Punkt die Ansteuerung des Displays über den SPI-Bus. Dadurch ist es sehr einfach möglich das Display an beliebige Controller anzubinden. Die nominale Taktfrequenz in einem S65-Mobiltelefon beträgt dabei 13 MHz – d.h. zur Übertragung der insgesamt 46464 Bytes ($176 \cdot 132 \cdot 2$) über die SPI-Schnittstelle sind ungefähr 40 ms notwendig, was einer maximalen Bildwiederholrate von 25 Bildern pro Sekunde entspricht. Hier ist aber ein gewisser, nicht vernachlässigbarer Overhead bei der Kommunikation noch nicht berücksichtigt. Grundsätzlich kann aber festgehalten werden, dass das Display schnell genug für eine ruckelfreie Darstellung von bewegten Inhalten ist.

Für den Betrieb des Displays wird eine Spannung von 2.9 V benötigt, die über einen Linearregler vom Typ TPS79901 erzeugt wird. Zusätzlich ist für die Hintergrundbeleuchtung eine Spannung von 3.3 V notwendig. Als besonderes Feature ist dabei die Hintergrundbeleuchtung dimmbar ausgeführt, d.h. der Grad der Helligkeit kann softwareseitig modifiziert werden. Dazu werden ein Digital-Analogwandler MCP4724 der Firma Microchip und ein LED-Treiber ZXLD1101 der Firma Zetex verwendet. Der LED-Treiber regelt aber nicht die Spannung der Hintergrundbeleuchtungs-LEDs, sondern den Strom – erst so ist eine lineare Steuerung der Lichtleistung möglich. Abbildung 4.5 zeigt den zugehörigen Schaltungsteil.

Wie schon erwähnt, erfolgt die Kommunikation mit dem Display über eine serielle SPI-Schnittstelle. Da das Display mit 2.9 V versorgt wird und die restliche Elektronik aber mit 3.3 V, ist hier ein sogenannter Pegelwandler notwendig um auf den zwei verschiedenen Spannungsebenen kommunizieren zu können. Prinzipiell wäre auch der Einsatz eines einfachen Spannungsteilers möglich (was auch in einigen Projekten zu dem Display vorgeschlagen wird), aber die sauberere Lösung ist auf jeden Fall ein Pegelwandler - hier konkret ein 74HC4050.
Der Digital-Analog-Wandler wird hingegen – wie auch die zusätzlich verfügbaren LEDs und der Baustein zum Auslesen der Tasten am Scrollrad – über das I2C-Protokoll angesteuert.

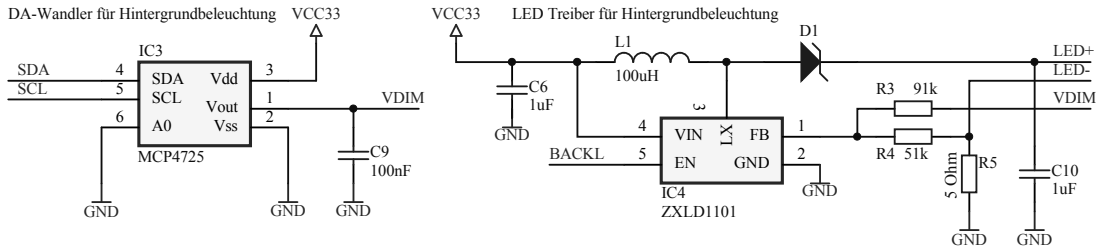

Abbildung 4.5: Schaltung der dimmbaren Hintergrundbeleuchtung des Displays

4.1.5 Scrollrad zur Eingabe

Zur Steuerung am Bedienpanel wird ein Drehencoder des Typs AMRX-1510 von Avago Technologies [57] verwendet. Dieser stellt neben dem Scrollrad auch weitere 5 Tasten zur Verfügung. Zusätzlich kann die Eingabeeinheit über eine LED auch beleuchtet werden. Abbildung 4.6 zeigt schematisch die beschriebene Funktionalität.

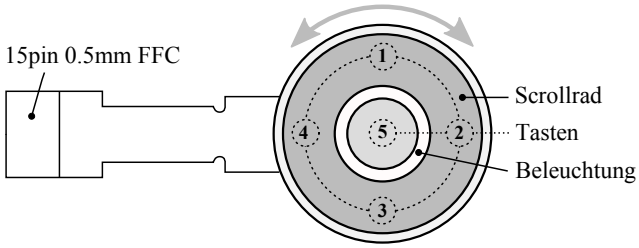

Abbildung 4.6: Funktionsschema des Scrollrades

Bei einer Umdrehung des Scrollrades werden vom Drehencoder 45 Impulse generiert. Um dabei auch die Drehrichtung feststellen zu können, wird ein Zwei-Kanal Quadraturverfahren eingesetzt bei dem zwei Signalformen mit einer bestimmten Phasenverschiebung zur Verfügung stehen. Die Auswertung dieser Signale wird dann bei der Besprechung des zugehörigen Linux-Treibers in Abschnitt 4.3.4 erläutert.

4.2 Linux Betriebssystem für Embedded Systems

Zur Einführung ein Zitat aus [DGS09]:

> Der Begriff des eingebetteten Systems bezeichnet einen Rechner (Mikrocontroller, Prozessor, DSP, SPS), welcher dezentral in einem technischen Gerät, in einer technischen Anlage, allgemein in einem technischen Kontext eingebunden (eingebettet) ist. Dieser Rechner hat die Aufgabe, das eingebettete System zu steuern, zu regeln, zu überwachen oder auch Benutzerinteraktion zu ermöglichen und ist speziell für die vorliegende Aufgabe angepasst.

Zusammengefasst lässt sich diese Definition so verstehen, dass ein eingebettetes System meist speziell auf eine Aufgabe zugeschnitten ist. Weitere Merkmale nach [DGS09] sind die, durch die feste Einbindung in einen Prozess (z.B. in einem Auto) entstehende, schlechte Wartbarkeit,

wodurch höhere Anforderungen an Zuverlässigkeit und Lebensdauer gestellt werden als etwa bei einem PC.

Als Speichermedien kommen bei Embedded-Systems selten Festplattenspeicher zum Einsatz, da diese aufgrund der mechanischen Komponenten fehleranfälliger sind. Vielmehr werden Halbleiterspeicher wie etwa Flash-Bausteine für die permanente Datenspeicherung eingesetzt - die Schwierigkeiten bei dieser Technologie wurden schon kurz im Abschnitt 3.1.9 über den FRAM-Datenspeicher erläutert.

Wie schon in der Einführung über die SmartPanel-Hardware besprochen, wird ein ARM9 Prozessor zur Realisierung der gestellten Aufgabe verwendet. Obwohl es natürlich möglich ist, ist dieser im Gegensatz zum M16C/62P Mikroprozessor, der für die Kühlschranksteuerung verwendet wurde, nicht mehr sehr so einfach auf der Hardware-Ebene (d.h. direkte Manipulation von Registern zur Ansteuerung der Peripherie) zu programmieren. Ein guter Ausgangspunkt bei der Recherche speziell für den verwendeten Prozessor AT91SAM ist dabei [56].

Als Betriebssystem für das Prozessormodul kommt ein für ARM-Prozessoren angepasstes Debian-Linux mit der Kernel Version 2.6.22 zum Einsatz.

4.2.1 Architektur des Linux Kernels und der Bootprozess

Bevor nun in den nächsten Abschnitten die Entwicklung von Treibern und Applikationen für die Linux-Plattform erläutert wird, soll hier ein kurzer Überblick über die Architektur von Linux gegeben werden. Die Aufgaben des Linux-Kernels können in 5 Subsysteme gegliedert werden [CRKH05, DGS09]:

- Prozessmanagement
- Speicherverwaltung
- Dateisystemverwaltung
- Gerätemanagement
- Netzwerkmanagement

Die Aufgabe des Prozessmanagements ist dabei das Erzeugen und Beenden der Prozesse, die Abwicklung der Kommunikation zwischen den Prozessen und auch das Scheduling - also die Zuteilung der Prozessorressourcen an die Prozesse.
Durch die Speicherverwaltung wird ein virtueller Adressraum erzeugt der weitaus größer ist als physikalischer Speicher vorhanden ist. Der Zugriff auf den Speicher erfolgt über Funktionen des Speichermanagements, wodurch sichergestellt werden kann, dass die Prozesse nicht auf Speicherbereiche anderer Prozesse oder gar des Kernels zugreifen können. Sollen Daten zwischen zwei Prozessen ausgetauscht werden, so ist dies über Betriebssystemfunktionen zur Interprozesskommunikation zu realisieren.

Das Dateisystem nimmt bei Linux eine besonders wichtige Stellung ein, da auch die Peripherie (also auch z.B. eine serielle Schnittstelle) als Datei in das System eingebunden und auch darüber angesprochen werden kann. Im Gerätemanagement werden die realen Implementierungen der für die Ansteuerung der Geräte notwendigen Treiber verwaltet – über das Dateisystem werden diese also abstrahiert dem Betriebssystem und somit auch dem User zur Verfügung gestellt. Im Abschnitt über die Treiberentwicklung folgt hierzu noch eine Übersicht über die verschiedenen Klassen von Gerätetreibern. Das Netzwerkmanagement ist schlussendlich für die

Netzwerkfunktionalität des Kernels verantwortlich - also dem Senden und Empfangen von Paketen, dem Zuweisen der Pakete zu den richtigen Prozessen oder auch dem Routing und der Adressauflösung.

Die hier erfolgte Aufteilung stellt nur eine grobe Einteilung dar, wie sie in vielen Büchern zu finden ist. Teilweise erfolgen weitere Unterteilungen des Kernels wie z.B. in das Systemcall-Interface oder ein I/O-Subsystem [QK04] - in anderen Unterlagen wird auch die Interprozesskommunikation als eigenes Subsystem angegeben [RLN06].

Abbildung 4.7 zeigt ein Schema der gerade angeführten Unterteilung des Linux-Kernels.

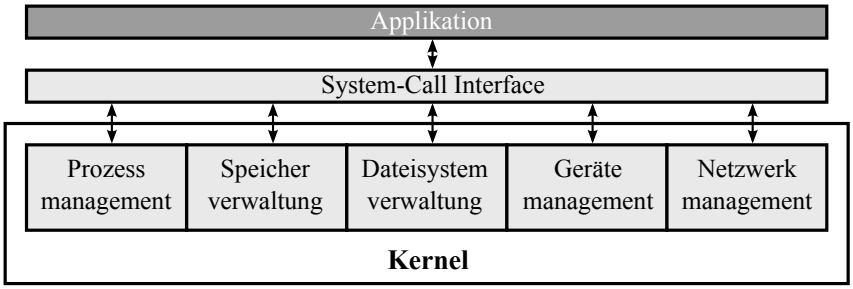

Abbildung 4.7: Aufteilung des Linux-Kernels in 5 Subsysteme

Als Bootloader zum Einspielen eines neuen Kernels wird *U-Boot* (Universal Bootloader [61]) verwendet, der sich inzwischen als Quasi-Standard für Embedded Linux Systems etabliert hat. Die Steuerung des Bootloader erfolgt über ein serielles Terminal, wobei das Einspielen des Kernels auch über das Netzwerk durchgeführt werden kann (was aufgrund der niedrigen Datenübertragungsrate der seriellen Schnittstelle auch empfohlen sei).

Um den Startvorgang auch am Display verfolgen zu können und Rückmeldungen über den aktuellen Status zu erhalten, wird über das Programm *PSplash* ein graphischer Bootscreen realisiert. [81].

4.3 Linux Treiber

In diesem Abschnitt werden zunächst die notwendigen Grundlagen für die Treiberentwicklung erläutert. Anschließend werden die für die SmartPanel-Hardware zu realisierenden Treiber vorgestellt. Hierbei kann natürlich - wie auch in den vorangegangenen Abschnitten - nur ein kleiner Ausschnitt dieser Thematik besprochen werden. Für weiterführende Informationen sei auf die angeführte Literatur zur Treiberprogrammierung verwiesen.

Zuerst sei die Einteilung der Gerätetreiber in drei grundlegende Klassen erklärt [CRKH05, Lov05]:

- Zeichenorientierte Geräte
- Blockorientierte Geräte
- Netzwerk-Schnittstellen

Bei einem zeichenorientierten Gerät steht ein Strom von Daten zur Verfügung, der wie eine Datei byteweise gelesen und geschrieben werden kann, es findet also ein sequentieller Zugriff statt.

Blockorientierte Geräte werden prinzipiell genauso angesprochen wie zeichenorientierte, allerdings besteht hier ein wahlfreier Zugriff auf die Daten und somit ist natürlich auch hier wieder eine sequenzielle Bearbeitung möglich. Allerdings ist das Lesen von Blockgeräten oft nur in Blöcken einer bestimmten Größe möglich - z.B. 512 Byte oder 4 kB bei einer Festplatte. Der wahlfreie Zugriff begründet auch die Tatsache, dass Dateisysteme nur auf blockorientierte Geräte gemountet werden können.
Der Zugriff auf Netzwerkschnittstellen erfolgt im Gegensatz zu den beiden anderen Gerätetypen nicht wie auf eine Datei, sondern über ein eigenes Interface – der *Socket*-Schnittstelle.

Eine weitere wichtige Entscheidung bei der Treiberentwicklung wird mit der Unterscheidung zwischen Kernel-Modulen und Kernel-Treibern getroffen. Module werden nach dem Start des Betriebssystems in den bereits laufenden Kernel eingebunden, während die Kernel-Treiber fixer Bestandteil des Kernels sind.

4.3.1 Überblick und Einführung

Folgende Funktionen wurden durch eigene Treiber in den Linux-Kernel integriert:

- Blinken der Status-LED
- Auswertung der Bedieneinheit
- Ansteuerung des DACs für die Steuerung der Hintergrundbeleuchtung
- Ansteuerung des LCD und Einbindung als Framebuffer

Da bei einem Embedded-System die Hardware festgelegt ist, werden bei der SmartPanel-Hardware alle Treiber direkt in den Kernel integriert.

Begonnen wird nun mit der Erläuterung des „Blink-Treibers“, der noch sehr simpel aufgebaut ist und sich somit gut für den Einstieg in die Treiberprogrammierung eignet. Die nachfolgenden Erklärungen der weiteren Treiber bauen auf diesen Grundlagen auf, wodurch die wesentlichen Informationen in den Vordergrund gestellt werden können. Alle in diesem Kapitel gezeigten Quellcodeabschnitte sind dabei auf die wesentlichen Funktionen reduziert und dienen zur besseren Veranschaulichung. Die Grundstruktur jedes Treibers enthält eine Funktion zur Initialisierung und eine Funktion die beim Beenden des Treibers aufgerufen wird.

Der Treiber für das Blinken der Status-LED wurde nun so realisiert, dass in der Initialisierungsfunktion ein sog. *Kernel-Thread* gestartet wird, der im Kontext des Kernels in einer Endlosschleife läuft und nach bestimmten Zeitintervallen den Zustand der LED ändert.

Hierzu der Source-Code in Listing 4.1 – um nur auf die wesentlichen Details einzugehen wird teilweise Pseudo-Code verwendet.

```
1  int LedBlink_Thread(void *data) {
2    daemonize("LedBlink");
3    allow_signal(SIGTERM);
4
5    while(1) {
6      // Pseudocode
7      LED = 1;
8      WARTE_500MS;
9      LED = 0;
10     WARTE_500MS;
11   }
```

```
12   do_exit(SIGSTOP);
13 }
14
15 static int __init LedBlink_init(void) {
16   kthread_run(LedBlink_Thread, NULL, "LedBlink", NULL);
17   return 0;
18 }
19
20 static void __exit LedBlink_exit(void) {}
21
22 module_init(LedBlink_init);
23 module_exit(LedBlink_exit);
24
25 MODULE_LICENSE("GPL");
```

Listing 4.1: Einführendes Beispiel zu einem Thread - Blinken einer LED

Die angesprochene Initialisierungsfunktion, die beim Einhängen des Treibers in den Kernel aufgerufen wird, stellt LedBlink_init dar. Das optionale Attribut __init sorgt dafür, dass der Kernel den Speicher der Initialisierungsfunktion wieder freigibt [CRKH05]. Für die *__exit*-Funktion, die beim Beenden des Treibers aufgerufen wird, gibt es in diesem Treiber nichts zu tun, da keine Ressourcen reserviert werden. Erst die beiden Makrofunktionen *module_init* und *module_exit* registrieren die erwähnten beiden Funktionen beim Kernel, ohne sie würde der Treiber nicht geladen bzw. entladen werden

Die Einbindung des Threads, der für das Blinken der LED verantwortlich ist, erfolgt über den Funktionsaufruf von *kthread_run*. Hierzu werden die Adresse der Funktion und eine Bezeichnung des Threads übergeben. Der erste Aufruf im Thread selber ist die Funktion *deamonize*, in welcher der Eigentümer des Threads (der sog. *Parent*) auf den Kernel-Thread *kthreadd* geändert wird. Dies geschieht, um beim Beenden des Parents nicht einen verwaisten Thread zu erhalten [Ven09] – so ein Thread wird als „Zombie-Prozess“ bezeichnet. Über *allow_signal* wird es dem Thread anschließend ermöglicht, auf Signale des Betriebssystems zu reagieren - im konkreten Fall auf das SIGKILL-Signal, das zum Beenden des Threads geschickt wird. Beim Auftreten dieses Signals erfolgt dabei in dieser Implementierung keine weitere Überprüfung, sondern der Thread wird einfach beendet [Ven09].
In der Hauptschleife wird über Kernelfunktionen jeweils eine halbe Sekunde gewartet und danach der Status der LED entweder auf 1 oder auf 0 gesetzt. Somit ergibt sich ein Blinkintervall von einer Sekunde.

4.3.2 Treiber für die Bedieneinheit

Die Aufgabe des Treibers für die Bedieneinheit ist die Erfassung der Impulse des Drehimpulsgebers und das Auslesen der Zustände der Tasten. Realisiert wird dies über zwei unterschiedliche Wege: Die Impulse des Drehimpulsgebers werden direkt an den Mikroprozessor angebunden, wohingegen die Tastenabfrage über einen I2C-Portexpander durchgeführt wird, der bei Betätigung einer Taste einen Interrupt auslöst und so das Auslesen des Tastenzustands veranlasst.

Für die geforderte Funktionalität ist also zuerst ein Treiber für die Ansteuerung des I2C-Portexpanders und darauf aufbauend der eigentliche Treiber für die Bedieneinheit zu programmieren. Die Ansteuerung der LEDs erfolgt ebenfalls über den Portexpander, wobei diese Funktionalität

nicht im Treiber für die Bedieneinheit, sondern direkt über den MCP23017-Portexpander durchgeführt wird.

4.3.3 MCP23017 Treiber

Der Linux-Kernel stellt eine Programmierschnittstelle zur Verfügung um den Zugriff auf I2C-Geräte mit möglichst geringem Aufwand realisieren zu können. Als Literatur für die Implementierung von I2C-Geräten kann hier vor allem [Ven09] empfohlen werden – in den meisten anderen Büchern über die Treiberprogrammierung wird der I2C-Bus nur grob umrissen.

Ausgangspunkt ist eine Funktion *i2c_add_driver*, die bei der Initialisierung des Treibers aufgerufen wird. Dieser Funktion wird eine Struktur übergeben in der einerseits die Bezeichnung des Treibers und andererseits Funktionen für das Ein- und Aushängen des I2C-Treibers hinterlegt ist. Dies sieht konkret so aus:

```
1 static struct i2c_driver mcp23017_driver = {
2   .driver = {
3     .name   = "mcp23017",
4   },
5   .attach_adapter = mcp23017_attach_adapter,
6   .detach_client  = mcp23017_detach_client,
7 };
8
9 static int __init mcp23017_init(void) {
10   return i2c_add_driver(&mcp23017_driver);
11 }
```

Listing 4.2: Initialisierung des MCP23017-Treibers

Hier darf aber nicht unerwähnt bleiben, dass die Struktur in dieser Form nur die minimal notwendigen Einträge enthält und mehr Möglichkeiten bietet als hier verwendet werden.

Die zuvor aufgerufene Funktion *mcp23017_attach_driver* ist aber immer noch relativ unspektakulär obwohl hier schon das eigentliche Einhängen des I2C-Treibers in das Betriebssystem beginnt – und zwar über die Kernel-Funktion *i2c_probe*.

```
1 static int mcp23017_attach_adapter (struct i2c_adapter *adapter) {
2   return i2c_probe(adapter, &addr_data, mcp23017_detect);
3 }
```

Listing 4.3: I2C Probe beim MCP23017-Treiber

Dieser Funktion werden drei Parameter übergeben, wobei nur die Adresse zur Funktion *mcp23017_detect* vom eigenen Treiber stammt. Die anderen beiden Parameter werden vom Linux-Kernel gestellt (die Struktur *addr_data* ist in der Datei *i2c.h* definiert). Wichtig an diesem Punkt ist die Definition eines Arrays mit dem Namen *normal_i2c*. In diesem werden nämlich die I2C-Adressen angegeben, an denen sich die realen Geräte für den Treiber befinden. Im konkreten Fall wird nur eine Einheit des Port-Expanders vom Typ MCP23017 verwendet wodurch die Adresse fix vorgegeben wurde.
Die Funktion *mcp23017_detect* die an *i2c_probe* übergeben und vom Kernel aufgerufen wird, ist etwas umfangreicher und wird deshalb hier nur beschrieben. Im ersten Schritt wird geprüft ob die Funktionalität zum Senden und Empfangen von Datenbytes durch den vom Betriebsystem

gestellten I2C-Adapter möglich ist. Danach wird vom Kernel Speicher für eine treiberinterne Datenstruktur angefordert und im nächsten Schritt mit den notwendigen Daten gefüllt. Zum Schluss wird der Treiber eingerichtet und initialisiert.

In der besprochenen Funktion werden auch einige sog. Attribute in das virtelle sysfs-Dateisystem von Linux eingebunden [QK04]. Als Beispiel sei hier im nachfolgenden Listing 4.4 das Lesen des *IOCON*-Registers aus dem MCP23017-Baustein erläutert. Durch den Befehl *i2c_smbus_read_byte_data* wird ein Datenbyte vom I2C-Gerät gelesen - zuvor muss lediglich über die Funktion *to_i2c_client* die zugehörige I2C-Clientstruktur angefordert werden.
Über das Makro *DEVICE_ATTR* werden der Name des Attributs im sysfs und die Zugriffsrechte festgelegt. Die Funktionen für das Lesen und Schreiben dieses Attributs müssen dabei getrennt voneinander realisiert werden - hier wird nur die Funktion für den lesenden Zugriff gezeigt.

```
1 static ssize_t mcp23017_iocon_read (struct device *dev,
2   struct device_attribute *attr, char *buf) {
3   int val;
4   struct i2c_client *client = to_i2c_client(dev);
5   val = i2c_smbus_read_byte_data(client, MCP23017_IOCON);
6   return (sprintf(buf, "%u\n", val));
7 }
8 static DEVICE_ATTR(iocon, S_IWUSR | S_IRUGO, mcp23017_iocon_read,
9   mcp23017_iocon_write);
```

Listing 4.4: Einrichten des sysfs für den MCP23017 Treiber

4.3.4 Treiber für das Scrollrad

Nun folgt die Beschreibung des eigentlichen Treibers für die Bedieneinheit, der über exportierte Funktionen mit dem MCP23017-Treiber kommuniziert. Neu hinzu kommt hier die Behandlung von Interrupts und das Registrieren eines „Input"-Gerätes.

Damit die Eingaben auf dem Bedienfeld im ganzen System verfügbar sind ohne einen speziellen Treiber ansprechen zu müssen, wird ein „Input"-Gerätetreiber erstellt über den Tastatureingaben simuliert werden können. Jede der Tasten auf dem Bedienfeld entspricht somit einem Tastencode, wodurch die Eingabe in allen Applikationen, die eine Tastatur unterstützen, verwendet werden kann. Hierzu nun auszugsweise der Sourcecode der Initialisierung der nachfolgend erläutert wird:

```
1 static int __init SmartInput_init(void) {
2   smart_input_dev = input_allocate_device();
3
4   smart_input_dev->name="SmartInput Device";
5   smart_input_dev->evbit[0]=BIT(EV_KEY);
6
7   set_bit(KEY_KPMINUS, smart_input_dev->keybit);
8   ... // set_bit auch für alle anderen Tasten
9   input_register_device(smart_input_dev);
10  request_irq(ROTARY_PIN_LEFT, ChannelA_interrupt, 0, "RotLeft", NULL)));
11  ... // request_irq auch für alle anderen Interrupts
12
13  return 0;
14 }
```

Listing 4.5: Initialisierung des Scrollrad-Treibers

Direkt zu Beginn wird mit *input_allocate_device* eine Struktur für ein Input-Device angelegt auf die der Pointer *smart_input_dev* zeigt. In diese Struktur wird die Bezeichnung des Treibers und mit *EV_KEY* die Art der Eingabe eingetragen. Welche Tasten genau mit dem Treiber simuliert werden können, wird dann mittels der Funktion *set_bit* festgelegt. Im konkreten Fall werden also die Richtungstasten, die Enter-Taste und das Plus- und Minus-Zeichen verwendet. Die eigentliche Registrierung des Input-Treibers erfolgt dann mit der Funktion *input_register_device*. Der nächste Programmteil aktiviert für die verwendeten Interrupteingänge des Drehimpulsgebers und der Interruptleitung des MCP23017 die Pull-Ups und das „deglitching“, also die Entstörung des Eingangsimpulses. Die an diese Funktionen übergebenen Parameter wie *ROTARY_PIN_LEFT* sind Präprozessorkonstanten, die im Headerfile des Treibers definiert werden und die entsprechenden Prozessorpins repräsentieren. Am Ende der Initialisierung werden noch die Interrupts für diese Pins aktiviert und die zugehörigen Interrupt-Service-Funktionen definiert. Der nächste Schritt ist die Behandlung der auftretenden Interrupts. Abbildung 4.8 zeigt hierzu die Impulsfolge an den entsprechenden Ausgängen des Drehencoders für Rechts- und Linkslauf.

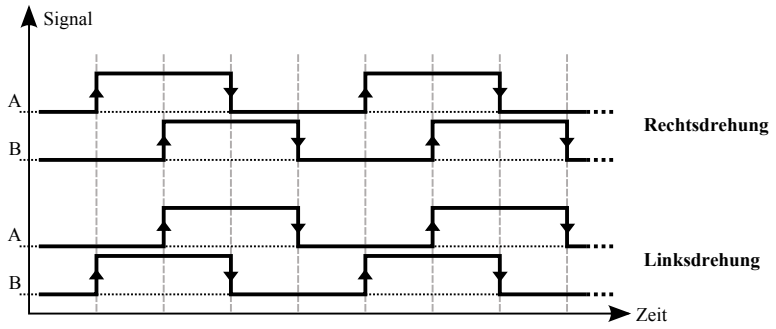

Abbildung 4.8: Signale des Scrollrades beim Rechts- und Linkslauf

Die Grafik lässt erkennen, dass sich Rechts- und Linkslauf durch die Reihenfolge des Auftretens der Signale eindeutig unterscheiden. Bei Rechtsdrehung ändert sich zuerst Kanal A auf einen bestimmten Zustand und um eine Phasenverschiebung später Kanal B - bei Linksdrehung umgekehrt. Genau diese Reihenfolge wird nun zur Auswertung der Drehrichtung herangezogen. Listing 4.6 zeigt hierzu einen Ausschnitt des Treibercodes.

```
1  static int last=0, chA=0, chB=0;
2  static irqreturn_t ChannelA_interrupt(int irq, void *dev_id) {
3    ...
4    chA = at91_get_gpio_value(ROTARY_PIN_RIGHT);
5    if(last != ROTARY_PIN_LEFT || chB != chA) {
6      last = ROTARY_PIN_RIGHT;
7      return IRQ_HANDLED;
8    }
9    last = 0;
10   ... // Tastendruck dem OS bekanntgeben
11   return IRQ_HANDLED;
12 }
13 static irqreturn_t ChannelB_interrupt(int irq, void *dev_id) {
14   ...
15   chB = at91_get_gpio_value(ROTARY_PIN_LEFT);
16   if(last != ROTARY_PIN_RIGHT || chB != chA) {
17     last = ROTARY_PIN_LEFT;
18     return IRQ_HANDLED;
19   }
```

```
20    last = 0;
21    ... // Tastendruck dem OS bekanntgeben
22    return IRQ_HANDLED;
23  }
```

Listing 4.6: Erkennung von Links- und Rechtslauf beim Scrollrad-Treiber

Tabelle 4.1 zeigt welche vier verschiedenen Zustände bei der Rechtsdrehung prinzipiell angenommen werden können.

Tabelle 4.1: Mögliche Zustände bei der Rechtsdrehung des Scrollrades

Kanal A	Kanal B	Zustand
$0 \rightarrow 1$	0	Zwischenzustand
1	$0 \rightarrow 1$	Schritt nach rechts erkannt
$1 \rightarrow 0$	1	Zwischenzustand
0	$1 \rightarrow 0$	Schritt nach rechts erkannt

Man sieht, dass nach dem Auftreten eines Interrupts immer auch auf das Auftreten des anderen Interrupts gewartet werden muss, bevor ein Schritt in eine Richtung als erkannt bewertet werden kann. Die Zwischenzustände werden dabei über die globalen Variablen *chA*, *chB* und *last* behandelt. Die Tabelle ist für eine Drehung nach links bis auf eine Vertauschung der Kanäle äquivalent.
Nachdem eine Drehung erkannt wurde, muss zum Schluss dem System noch das simulierte Drücken und Loslassen einer Taste mitgeteilt werden. Dies geschieht mit den beiden Funktionen *input_report_key* und *input_sync*. Letztere sorgt dafür, dass die übermittelte Taste sofort behandelt wird – ansonsten könnte es passieren, dass der Eintrag vom Betriebssystem nur gebuffert und erst verspätet bearbeitet wird [Ven09] [74].

Da die weiteren Tasten der Bedieneinheit nicht direkt sondern über den I2C-Portexpander MCP23017 ausgelesen werden, ist hier eine etwas andere Herangehensweise als zuvor notwendig. Wird eine beliebige dieser Tasten am Bedienfeld betätigt, so wird darüber der Prozessor und somit auch der Treiber durch das Auslösen des Interrupts in Kenntnis gesetzt. Im nächsten Schritt muss der Zustand der Tasten über den I2C-Bus vom MCP23017-Bauteil ausgelesen werden. Dies sollte natürlich nicht in der Interrupt-Service-Routine selber passieren, weshalb man sich der sog. *Tasklets* bedient [QK04].

Durch Tasklets ist es möglich längere Aufgaben, die von einem Interrupt aus angestoßen werden, bearbeiten zu lassen ohne die Interrupt-Latenzzeiten zu beeinträchtigen. Die Initialisierung der notwendigen Strukturen erfolgt dabei über ein Makro *DECLARE_TASKLET* (für die dynamische Initialisierung existiert die Funktion *tasklet_init*).
Über den Funktionsaufruf *tasklet_schedule* wird dem Kernel in der Interrupt-Service-Routine mitgeteilt, dass der Tasklet nun gestartet werden kann - wobei das im Fall der hier verwendeten Funktion mit einer niedrigen Priorität erfolgt. Wünscht man eine Bearbeitung direkt nach dem Verlassen der ISR, so muss die Funktion *tasklet_hi_schedule* verwendet werden [QK04], was im Fall einer Tastenabfrage aber nicht notwendig ist.
Der Code im Tasklet selber ist dann relativ simpel: zuerst wird der Zustand aller Tasten ausgelesen und anschließend die einzelnen Bits, welche die Zustände repräsentieren, geprüft. Wurde nun eine Taste erkannt, so wird genauso verfahren wie zuvor beim Drehencoder, indem die zu simulierende Taste dem Kernel gemeldet wird. Nach diesem Schema werden auch alle weiteren Tasten behandelt.

4.3.5 LCD Treiber und Framebuffer

Im Abschnitt über die Hardware wurde bereits das verwendete Display vorgestellt. Ziel des LCD Treibers ist es nun einerseits die Ansteuerung des Displays über den SPI-Bus zu realisieren und andererseits das Display über das Framebuffer-Interface des Linux-Kernels verfügbar zu machen. Zuerst werden hierzu aber einige grundlegende Details erläutert [RLN06, Ven09].

Eine grafische Anzeige kann grundsätzlich in die Einheiten Zeile, Spalte und Pixel unterteilt werden. Dies ist an sich trivial, aber für die kommenden Betrachtungen und Begriffserklärungen ist diese Unterscheidung essentiell.
Der Aufbau eines Bildes auf einem Display erfolgt definitionsgemäß zeilenweise von oben nach unten, d.h. nachdem alle Spalten der aktuellen Zeile mit Information gefüllt wurden, wird in die nächste Zeile gewechselt. Das komplette Neuzeichnen eines Bildschirmes wird dabei meist als *Refresh* bezeichnet – wie oft dies pro Sekunde passiert als *Refresh Rate* oder auch *Aktualisierungsrate*.
Jeder Bildpunkt wird durch genau ein Pixel repräsentiert, über das die Farbinformation am Display an genau einer Stelle dargestellt wird. Wie viele unterschiedliche Farben möglich sind, hängt dabei von der Anzahl der verfügbaren Bits pro Pixel ab. Im hier verwendeten Display können maximal 16 Bit pro Pixel verwendetet werden, was $2^{16} = 65523$ unterschiedliche Farben erlaubt. Die Farben an sich sind damit aber immer noch nicht genau definiert, da noch nicht festgelegt wurde welche und wie viele Bits jeweils den drei Grundfarben rot, grün und blau zugewiesen werden. Hierzu muss noch erwähnt werden, dass es außer dem hier beschriebenen RGB-Modell auch noch andere Farbmodelle gibt.
Tabelle 4.2 listet einige der Formate auf (aus [RLN06]), Abbildung 4.9 zeigt die Zuweisung der Bits zu den Farben speziell für das hier verwendete 16 Bit-Format RGB565.

Tabelle 4.2: RGB Farbformate

Format	Bits	Rot	Grün	Blau	Farben
Monochrome	1	-	-	-	2
Indiziert 8 Bit	8	-	-	-	256
RGB444	12	4	4	4	4096
RGB565	16	5	6	5	65536
RGB888	24	8	8	8	16 Mio.

Mit diesem Wissen kann nun leicht die benötigte Größe für einen passenden Speicherbereich berechnet werden, in dem der gesamte Bildinhalt des Displays gespeichert werden kann. Konkret sind es also $176 \cdot 132 \cdot 2 = 46464$ Bytes die benötigt werden. Aus Programmierersicht hat man also ein Array von ungefähr 46 kB in dem sich die gesamte Bildinformation befindet. Will man jetzt bestimmte Punkte auf dem Display verändern, kann man Zeile und Spalte über Formel 4.1 in eine Position innerhalb dieses Speicherbereiches umrechnen.

$$Index\_im\_Array = (Pixel\_pro\_Zeile \cdot y) + x \tag{4.1}$$

Hier wird auch klar wieso man eindeutig zwischen Zeile und Spalte unterscheiden muss, würde man die beiden vertauschen, so würde man das Pixel im Speicherbereich an eine falschen Stelle setzen.

Wie schon angesprochen geht es hier auch darum, einen *Framebuffer*-Treiber zu entwickeln. Dieser ermöglicht eine einfache und transparente Schnittstelle zwischen der Applikation und dem eigentlichen Grafiktreiber.
Ein *Framebuffer* wird dabei als zeichenorientiertes Gerät implementiert. Wie schon im einführenden Abschnitt über Linux-Treiber angeschnitten, stehen somit zur Kommunikation zwischen Applikation und Treiber die Standardfunktionen wie *open, read* und *write* zur Verfügung. Um an bestimmte Positionen innerhalb des Speichers zu gelangen kann die Funktion *fseek* verwendet werden. Hält man sich nun vor Augen, dass bei den meisten Anwendungen nie der gesamte Bildinhalt geschrieben werden muss, sondern nur bestimmte Teile wie Linien oder auch einzelne Punkte, so kann man den Overhead erahnen, der durch die Verarbeitungsweise mit read, write und fseek entstehen würde.
Das *Framebuffer-Framework*, auf dem der *Framebuffer-Treiber* aufsetzt, stellt deswegen über die Kernel-Funktion *mmap* eine Möglichkeit zum direkten Zugriff auf den Bildspeicher zur Verfügung.

Für die Implementierung des Framebuffer-Treibers müssen im ersten Schritt einige Strukturen mit den entsprechenden Daten gefüllt werden. Diese werden nun kurz in Tabelle 4.3 benannt, anschließend wird auf wichtige Einträge näher eingegangen.

Tabelle 4.3: wichtige Datenstrukturen für einen Framebuffer-Treiber

Datenstruktur	Aufgabe
fb_info.fix	Enthält nicht veränderliche Informationen über das Display
fb_info.var	Enthält variable Informationen über das Display
fb_info.fops	Hier werden die Funktionen für die Operation auf dem Display festgelegt

Einige der Einträge in *fb_info.fops* sind optional, da hier das Framebuffer-Framework Standard-Funktionen zur Verfügung stellt. Sie können dennoch implementiert werden wenn eine spezielle Bearbeitung, etwa aufgrund der Hardware, notwendig ist. Nun zu den Strukturen *fb_info.fix* und *fb_info.var*, wobei eher unwichtige oder mit Standardwerten besetzte Einträge nicht angeführt werden:

```
1  static struct fb_fix_screeninfo ls020fb_fix= {
2    .id              = "ls020",  // Identifikation des Framebuffers
3      ...
4    .line_length     = X_RES * B_PP / 8,
5  };
6
7  static struct fb_var_screeninfo ls020fb_var = {
8    .xres            = X_RES,
9    .yres            = Y_RES,
10     ...
11   .bits_per_pixel = B_PP,
12   .red             = { 11, 5, 0 },
13   .green           = { 5, 6, 0 },
14   .blue            = { 0, 5, 0 },
15 };
```

Listing 4.7: Strukturen fb_fix und fb_var für den LCD-Treiber

In der Struktur *ls020fb_fix* wird unter anderem mit dem Feld id eine Identifikation des Framebuffers angegeben und mit dem Feld *line_length* die Anzahl der Pixel pro Zeile, hier also der

Wert 352 (die Konstante B_PP steht für Bits pro Pixel, hat also den Wert 16).
Die Struktur *ls020_var* beinhaltet zum einen mit den Feldern *xres* und *yres* die Auflösung des LCDs und mit dem Feld *bits_per_pixel* die schon vorhin verwendete Anzahl an Bits pro Pixel. Interessant sind hier vor allem die Einträge in den Feldern red, green und blue, in denen die genaue Position und Länge der den Farben zugeordneten Bits eingetragen wird - siehe Abbildung 4.9.

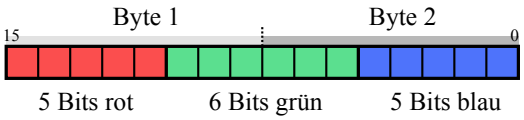

Abbildung 4.9: Zuteilung der Bits im RGB565-Modus

Kern bei der Initialisierung ist die Struktur *fb_info*, in der alle notwendigen Daten eingetragen werden müssen. Im Element *screen_base* wird der Speicherbereich definiert, der mittels *mmap* dem User zur Verfügung gestellt wird, während in *smem_start* die Adresse des internen sog. *Shadow*-Speichers darstellt, der zum LCD übertragen wird.

Etwas, was bisher nicht behandelt wurde ist die Frage, wie im konkreten Fall die Daten der Applikation zum LCD gelangen. Zum einen muss natürlich, wie schon erwähnt, dem Treiber eine mmap-Implementierung zur Verfügung gestellt werden über die die Applikation den Framebuffer-Speicher einbinden kann. Die Funktion wird hier jetzt aber nicht näher ausgeführt.

Da das Display über die serielle SPI-Schnittstelle angesprochen wird, kann natürlich nicht direkt auf den Bildspeicher im LCD zugegriffen werden. Genau aus diesem Grund existiert auch der vorhin erwähnte „Shadow"-Speicher, der im Grunde genommen zwei Aufgaben hat. Zum einen repräsentiert er den Speicher für das, was am LCD angezeigt wird – zum anderen kann so geprüft werden ob sich der Inhalt im Userspeicher (dessen Adresse in *screen_base* eingetragen ist) geändert hat und die Daten neu an das LCD übertragen werden müssen. Die Grafik 4.10 verdeutlich hierzu die Zusammenhänge.

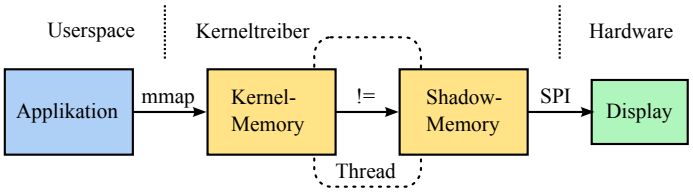

Abbildung 4.10: Zusammenspiel von Applikation,Treiber und Display

Die Applikation überträgt dabei zuerst die Daten in den Kernel-Speicher. Der Framebuffer-Treiber prüft innerhalb eines Threads 25 mal pro Sekunde ob sich der Inhalt des Kernel-Speichers vom Shadow-Speicher unterscheidet und kopiert gegebenfalls den kompletten Datenbereich. Über die SPI-Schnittstelle werden diese Daten anschließend an das Display übertragen. Listing 4.8 zeigt die Implementierung des zugehörigen Threads.

```
1 static int ls020fb_thread(void *param) {
2   struct fb_info *info = _ls020par.info;
3
4   while(!kthread_should_stop()) {
5     if(memcmp(_ls020par.screen_shdw, info->screen_base, FBMEM_SIZE)) {
6       memcpy(_ls020par.screen_shdw, info->screen_base, FBMEM_SIZE);
7       ls020_TransferFullScreen();
```

```
8       }
9       schedule_timeout_interruptible(REFRESH_TIME);
10    }
11    return 0;
12  }
```

Listing 4.8: Thread zur Aktualisierung des LCDs

4.4 Applikationssoftware

In diesem Abschnitt wird nun die Hauptapplikation vorgestellt die grundsätzlich folgende Funktionen zu erfüllen hat:

- Menü- und Anzeigesystem für das Display
- Steuerung der LEDs, der akustischen Ausgabe und der Hintergrundbeleuchtung
- Kommunikation mit der SmartFridge-Hardware
- Integrieren von Daten in die Datenbank
- Schnittstelle für die Datenabfrage von der Webseite

Der letzte Punkt wird dabei in diesem Kapitel nur kurz angeschnitten und erst im Kapitel über die Webseite etwas ausführlicher erläutert.

4.4.1 Entwicklungsumgebung

Als Entwicklungsumgebung kommt dabei „Eclipse CDT“ [64] zum Einsatz, das eine für C/C++ Programmierung angepasste Eclipse-IDE darstellt. Der Vorteil einer IDE gegenüber der Programmierung mit Texteditor und Makefiles besteht darin, dass die Entwicklung durch zusätzliche Tools vereinfacht wird. Als besonders wertvoll haben sich die Möglichkeiten zum Refactoring, also der unterstützten Umgestaltung des Sourcecodes, erwiesen.
Subjektiv musste aber festgestellt werden, dass sich die Eclipse-IDE für die C/C++ Entwicklung auf den zwei verwendeten Rechnern etwas träger verhält als die IDE für die Java-Programmierung. So dauerte es manchmal einige Sekunden bis ein zuvor kopierter oder ausgeschnittener Text an der einzufügenden Stelle wieder erschien.

4.4.2 Benutzerinterface

Vor der eigentlichen Programmierung der Applikation musste ein Konzept für die Bedienung erstellt werden. Folgende Elemente sollten dabei realisiert werden, um eine komfortable Bedienung zu erreichen:

- Menüsystem
- Anzeige von Werten und Grafiken
- Anzeige von Listen (z.B. für ein Logfile)
- modale Nachrichtenfenster
- Eingabefelder für Werte

Da das Display mit seiner Auflösung von 176 mal 132 Bildpunkten nicht sehr viel Platz bietet, wurden die Designs zuerst in einem Vektorgrafikprogramm entworfen. Der Entwurf der grafischen Elemente für das Benutzerinterface in einem Vektorgrafikprogramm brachte erhebliche Vorteile beim Testen des Zusammenspiels von Farben, Schriftarten und Größenverhältnissen mit sich. Abbildung 4.11 zeigt dies am Beispiel des Hauptmenüs.

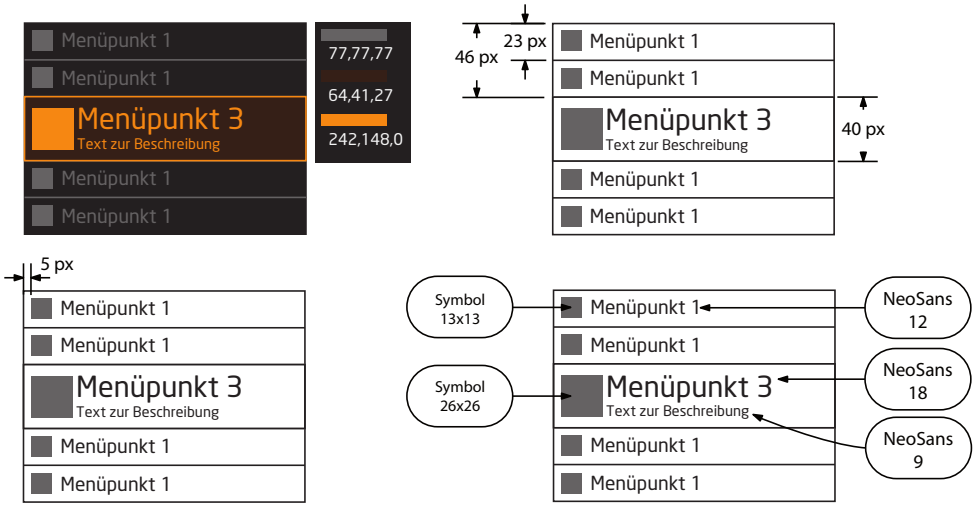

Abbildung 4.11: Farbgestaltung und Layout des Hauptmenüs

4.4.3 Framework zur grafischen Darstellung

Um das zuvor erarbeitete Konzept zu realisieren, wurde im nächsten Schritt ein sog. *Framework* gesucht, auf dem die Implementierung aufbauen konnte. Dieses Framework sollte so viel Programmierarbeit wie möglich abnehmen, sodass im Idealfall nur mehr die grafische Darstellung und die Herstellung der Verknüpfungen zwischen den einzelnen Elementen übrig bleiben sollten.

Diese Suche erwies sich jedoch nur in Teilen als erfolgreich, da die meisten der getesteten Systeme aus verschiedenen Gründen nicht für den Einsatz im SmartPanel geeignet waren. Das erste getestete Framework war QT4 für Embedded-Systems [82]. Dieses bietet zum einen eine umfangreich C++ Klassenbibliothek die über einen eigenen Präprozessor um diverse Funktionen – die in C++ an sich nicht enthalten sind – erweitert wurde.
QT4 ist ein sehr umfangreiches, mit vielen Tools ausgestattetes Framework – genau das stellte sich aber als Nachteil für die Verwendung auf der SmartFridge-Hardware heraus. Das Laden selbst einfachster Beispiele dauerte auf dem mit 200 MHz getakteten ARM9 Prozessor der in der SmartPanel-Hardware verwendet wird, manchmal bis zu 10 Sekunden. Dies ist gerade auf einem Embedded-Systems nicht tragbar. Bewegter Inhalt wurde bei voll ausgelastetem System teilweise nur ruckelnd wiedergegeben. Und zuletzt war auch der Speicherverbrauch mit 9 MByte für ein einfaches Beispiel relativ hoch. Leider konnte hier keine Möglichkeit gefunden werde die Ladezeiten zu verringern.

Ein weiteres getestetes Framework war MiniGUI [79]. Dieses schien der Beschreibung nach von der Größe und Geschwindigkeit für leistungsschwächere Embedded-Systems passender als QT4. Das Compilieren des SourceCodes funktionierte nach anfänglichen Problemen, jedoch stürzte die Testapplikation auf dem SmartPanel schon beim Start ab. Modifikationen am Sourcecode des Frameworks bewirkten zwar kleine Erfolge, die Applikation beendete sich aber immer noch bei

gewissen Aufrufen. Da nicht abzusehen war, wie viel Zeit für die komplette Anpassung des Codes benötigt werden würde (sofern dies überhaupt zum Erfolg geführt hätte, da die genaue Ursache der Abstürze nicht geklärt werden konnte), wurde auch die Verwendung dieses Frameworks verworfen.

Schlussendlich fiel die Entscheidung auf "nano-X", das unter anderem folgende Module zur Verfügung stellt:

- Farbverwaltung
- Zeichnen (Punkt, Linie, Kreis, ..)
- Textdarstellung und Fonts
- Laden von Grafiken
- Timer und Event-System

Die Eingabe über das Bedienfeld und die Ausgabe in den Framebuffer des Displays (siehe 4.3.5) konnten dabei sehr einfach über eine Konfigurationsdatei festgelegt und integriert werden. Bei nano-X handelt es sich also eher um eine Bibliothek die bei der Programmierung von Benutzerschnittstellen unterstützt, als um ein komplettes Framework wie es etwa QT4 darstellt. Damit nano-X nicht direkt in die Applikation integriert werden muss, wird es als Server eingerichtet mit dem über eine Programmierschnittstelle kommuniziert werden kann.

Für die Implementierung der Benutzerschnittstelle musste nun also ein komplettes Menüsystem entworfen und programmiert werden. Da für diese Implementierung einige Funktionalitäten benötigt wurden, die mit der Standard-C++ Bibliothek eher umständlich zu realisieren sind, wurde die Boost-C++ Library [58] eingebunden. Die Library setzt sich dabei aus vielen verschiedenen Unterbibliotheken zusammen, von denen nur einige hier benutzt wurden – etwa für die Netzwerkverbindung oder die serielle Schnittstelle. Ein guter Einstieg in die Boost-C++ Library bieten die Webseite [62] und das Buch [Kar09].

Abbildung 4.12 zeigt das Zusammenspiel der Applikation und den eingebunden Libraries.

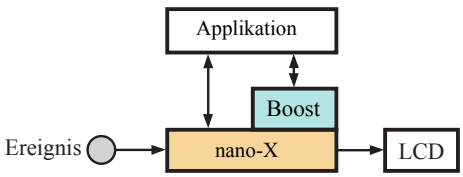

Abbildung 4.12: Zusammenspiel von Applikation und Nano-X

4.4.4 Architektur

Wie bereits im Abschnitt über das Benutzerinterface aufgelistet, müssen verschiedene Möglichkeiten zur Repräsentation und Modifikation der Daten implementiert werden. Um die notwendige Flexibilität zu erreichen wird hierbei unter C++ stark mit Vererbung gearbeitet.
Abbildung 4.13 zeigt die Klassenhierarchie für die grafischen Elemente des Benutzerinterfaces.

Die Basisklasse dieser Hierarchie ist die Klasse *swWidgetBase* von der alle anderen Klassen (die im grafischen Kontext verwendet werden) abgeleitet werden. Eine Unterteilung erfolgt dann weiter durch die Klasse *swWidgetContainer* und *swWidgetControl*. Die erste wird für das Zusammenfassen von Elementen der zweiten Kategorie verwendet. Ein Button oder ein Text kann

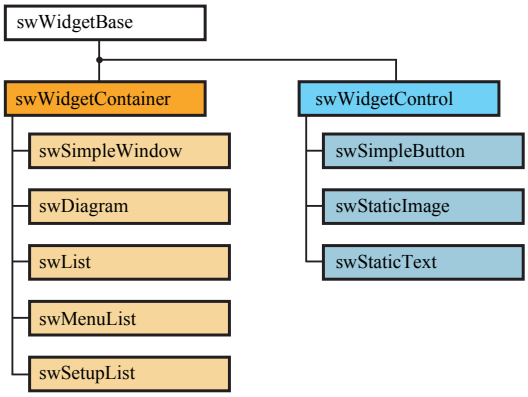

Abbildung 4.13: Klassenhierarchie des Menüsystems

also nie für sich alleine angezeigt werden, sondern muss immer in einen Container eingebunden sein.

Einige der Containerklassen bringen weitere Klassen für die Darstellungen der einzelnen Elemente mit, siehe Abbildung 4.14.

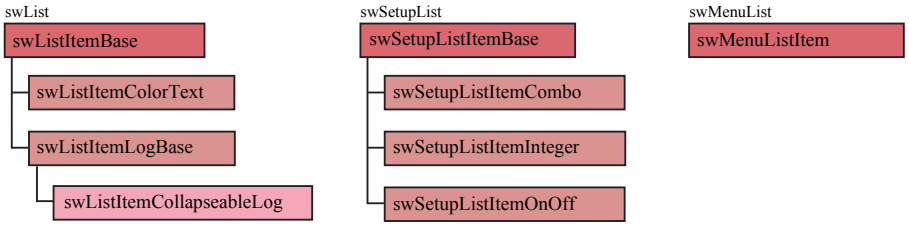

Abbildung 4.14: Unterklassen des Menüsystems

Die Hauptfunktion des Menüsystems ist in der statischen Klasse swApp implementiert. Darin wird bei der Initialisierung ein Thread gestartet, in dem auf Ereignisse gewartet wird. Dabei wird das sog. Event Capturing verwendet bei dem die Ereignisse immer vom Wurzel-Element bis zum Zielelement durchgereicht werden (im Gegensatz dazu steht das „Event Bubbling“ bei dem dies auf genau umgekehrte Weise realisiert wird). Abbildung 4.15 soll dies veranschaulichen.

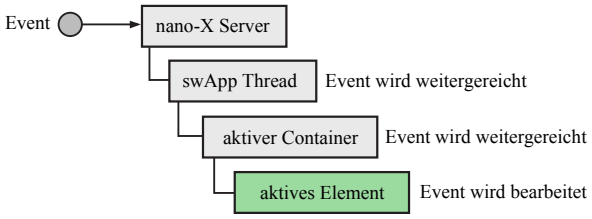

Abbildung 4.15: Beispiel zum Durchreichen eines Events

Ein Beispiel hierzu wäre ein Button in einem Container. Bei Drücken der Taste wird dieses Ereignis zuerst im nano-X Server registriert und an den swApp-Thread weitergeleitet. Dort wird das Ereignis an den aktuellen Container übermittelt der es wiederum an den Button übergibt. Zur Übergabe der Events wird dabei eine Datenstruktur verwendet die nicht nur eine Unterscheidung über die Art des Events, sondern auch Rückgabewerte an die Applikation ermöglicht.

Eine Besonderheit bei der Implementierung der Tastenauswertung soll dabei noch erwähnt wer-

den. Die Auswertung und Behandlung werden nämlich in zwei getrennten Funktionen realisiert, und zwar in *OnKeyDown* und *HandleKeyEvent*. Dies ermöglicht es, dass ein Tastendruckevent von einem Objekt auf einer niedrigeren Ableitungsebene bearbeitet wird als dem ursprünglichen Zielobjekt. Abbildung 4.16 verdeutlicht dazu ein Beispiel bei dem ein Button-Control ein Tastendruck-Ereignis erhält, dieses aber nicht bearbeitet.

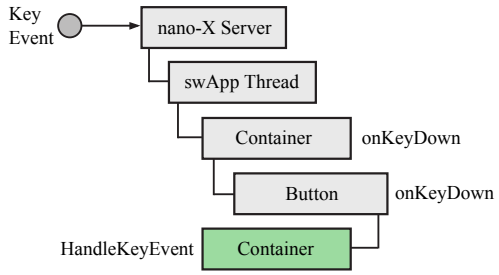

Abbildung 4.16: Beispiel zum Durchreichen eines Events bis zur HandleKeyEvent-Funktion

Das Button-Control Element übergibt also den Event an die HandleKeyEvent-Funktion des Elternobjektes zurück. Dadurch hat das Button-Control die Möglichkeit nur die Tasten zu bearbeiten die es auch bearbeiten will - alle anderen werden zurück an den Eltern-Container geschickt wo sie bei Bedarf noch behandelt werden. Die Rückgabe an sich geschieht über einen Pointer m_parent den jede von swWidgetBase abgeleitete Klasse beinhaltet.
Mit dieser Aufspaltung kann also ohne großen Programmieraufwand eine Zuweisung der Verantwortlichkeiten von Tastenereignissen erreicht werden. Wichtig ist aber, dass der Aufruf der HandleKeyEvent-Funktion nicht automatisch erfolgt sondern immer in der OnKeyDown-Funktion. Benötigt man keine Rückgabe des Ereignisses so muss auch keine HandleKeyEvent-Funktion implementiert werden.

Bei der Ausführung über die Klassenhierarchie wurde erwähnt, dass Elemente zur Anzeige immer in einem Container enthalten sein müssen. Hierzu gibt es eine Ausnahme und zwar ein sog. „Overlay“. Dieses stellt eine Art modales Element dar, das über den gerade aktiven Container gezeichnet wird. Verwendet wird es unter anderem zur Anzeige von wichtigen Informationen die unabhängig vom aktuellen Container angezeigt werden müssen – ein Beispiel hierzu wäre die Anzeige eines Fehlers bei der Kommunikation zwischen SmartPanel und SmartFridge. So lange dieses Overlay-Fenster aktiv ist, werden auch alle Ereignisse dorthin geleitet. Abbildung 4.17 zeigt hierzu die verfügbaren Klassen.

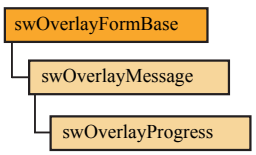

Abbildung 4.17: Klassen zum Overlay-Mechanismus

Stellt man sich nun einen Container mit mehreren Unterelementen vor (dies können weitere Container oder auch Controls sein), so muss es natürlich eine Möglichkeit geben die modifizierbaren Elemente irgendwie auszuwählen. Dies geschieht über eine Fokus-Zuordnung mittels der Rechts/Links-Tasten am Bedienfeld. Jedes aktivierte Element in der intern verwalteten Liste erhält dabei der Reihe nach den Fokus – d.h. das erste zum Container hinzugefügte Elemente bekommt den ersten Fokus. Dies gilt natürlich auch für die Elemente von untergeordneten Containern, es ist also eine beliebig tiefe Verschachtelung möglich. Ist der Fokus beim letzten

Element der Liste angelangt, so wird als nächstes wieder das erste Element gewählt. Natürlich benötigen aber nicht alle Elemente einen Fokus – so wird im Normalfall ein statisches Textelement nicht ausgewählt werden. Da die Entscheidung ob das nächste Element einen Fokus erhält durch eine Variable in der Basisklasse getroffen wird, kann das Fokus-Verhalten von jedem Element bestimmt werden. Ein statischer Text bekommt also standardmäßig keinen Fokus, kann aber bei Bedarf aktiviert werden.

Nun wurde bereits geklärt wie ein Element innerhalb eines Containers ausgewählt werden kann, nicht aber wie eine Aktion auf dieses Element ausgeführt wird - dies soll nun erläutert werden. Das grundsätzliche Prinzip hinter den Aktionen ist das sog. Signal-Slot-Konzept [84], das ursprünglich von der QT4-Bibliothek [82] kommt. Im Kern handelt es sich dabei um übliche Callback-Funktionen die in einer Klasse gekapselt sind - die verwendete Boost-Bibliothek bietet hierzu zwei Implementierungen an [58].
Wie bereits die Bezeichnung vermuten lässt, wird dieses Prinzip durch sog. Signals und Slots (auf Deutsch etwa „Steckplatz“) realisiert. Dabei können ein oder mehrere Slots an ein Signal gebunden werden, bei dessen Aktivieren die Funktionen der zugehörigen Slots aufgerufen werden. Man spricht hier auch von einer ereignisgesteuerten Programmierung, da die Funktionen erst dann aufgerufen werden wenn der Benutzer dies veranlasst.

Alle grafischen Elemente im Menüsystem die Aktionen auslösen können, stellen Funktionen zum Einbinden und Entfernen eines *Slots* in das Element zur Verfügung. Dabei reduziert sich die Anwendung aber nicht nur auf die Behandlung von Tasten-Ereignissen – auch die OnPaint-Funktion, in der die Elemente gezeichnet werden, ruft ein Signal auf, an das Slots gebunden werden können. Auf diese Art kann die Darstellung eines Elements außerhalb des Menüsystems modifiziert werden. Im nächsten Kapitel wird zur Veranschaulichung auch ein Beispiel über die Verwendung dieses Mechanismus gezeigt.

4.4.5 Schnittstellen zum Menüsystem

Ausgangspunkt für das Einrichten des Menüsystems ist die Klasse *swApp* die auch den Thread für die Eventbehandlung beinhaltet. Tabelle 4.4 zeigt die Funktionen die für die Verwaltung der Container und Overlays zur Verfügung stehen.

Tabelle 4.4: Funktionen der Menüsystemapplikation zur Container- und Overlayverwaltung

Funktionsname	Aufgabe
addContainer	Hängt eine Containerobjekt in das Menüsystem ein
removeContainer	Entfernt ein Containerobjekt aus dem Menüsystem
getActiveContainer	Liefert den aktuellen Container zurück
refreshActiveContainer	Zeichnet den aktuellen Container neu
showOverlay	Zeigt ein Overlay-Fenster an
closeOverlay	Verbirgt ein Overlay-Fenster
removeOverlay	Entfernt ein Overlay-Fenster aus dem Speicher

Ein weiterer Satz an Funktionen ist für die Steuerung der akustischen Ausgaben und der Hintergrundbeleuchtung verfügbar, die aber hier nicht dargestellt werden, da auf diese Punkte erst später eingegangen wird (Abschnitt 4.4.6).

Abbildung 4.18 zeigt links nun in UML Notation die wichtigen Funktionen der Klasse *swWidgetBase*, von der bis auf die Overlay-Klasse alle anderen Klassen zur Darstellung von Containern und Elementen abgeleitet werden. Rechts ist die Basisklasse für Container *swWidgetContainer* dargestellt, durch die das Einbinden von Controls und auch weiteren Containern ermöglicht wird - hier werden nun zusätzlich zu den wichtigen Funktionen auch einige Attribute dargestellt.

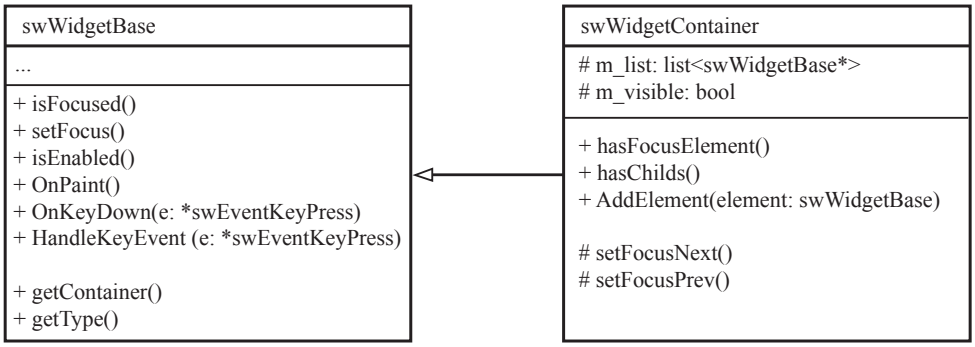

Abbildung 4.18: Die Klassen swWidgetBase und swWidgetContainer in UML Notation

Der erste Block der Funktionen in der Klasse *swWidgetBase* stellt dabei die virtuell deklarierten Funktionen dar. Die Funktion *OnPaint* ist für das Zeichnen des Containers zuständig und wird von der Eventbehandlung des Menüsystems aufgerufen. Da es sich um eine virtuelle Funktion handelt, wird somit automatisch die Implementierung des jeweiligen Containers aufgerufen.

Kern eines Containers ist die Liste *m_childs* in der alle anderen anderen Elemente (also Controls und Container) gespeichert werden. Konkret werden natürlich nicht die Elemente an sich, sondern nur die Adressen darauf gespeichert. Das Einbinden von Elementen in einen Container geschieht mittels der überladenen Funktion *AddElement* die als Option den Parameter inFront anbietet. Über diesen kann festgelegt werden ob das eingefügte Element über allen anderen derzeitigen Elementen zu sehen sein soll – praktisch wird das neue Element also entweder an den Anfang oder das Ende der Liste *m_childs* gesetzt.

Nachfolgendes Listing 4.9 zeigt wie das Menüsystem initialisiert und anschließend ein Fenster mit einem Button eingerichtet wird.

```
1 swApp App;
2 swSimpleWindow sw_demo;
3 swSimpleButton button;
4
5 void InitApp (void) {
6   App.Init();
7   button.setCaption("Button");
8   sw_demo.AddElement(button);
9   App.AddContainer(sw_demo);
10 }
```

Listing 4.9: Beispiel zum Einrichten des Menüsystems

Die meisten der Elemente implementieren zudem Möglichkeiten zur grafischen Anpassung - so kann etwa bei Containern und Buttons die Hintergrundfarbe gesetzt oder auch ein Hintergrundbild geladen werden.

Im vorangegangenen Abschnitt wurde besprochen wie ein Element über das Signal-Slot-Prinzip mit einer Funktion verbunden werden kann. Aufbauend auf der vorherigen Initialisierung wird dies nun in Listing 4.10 auch an einem konkreten Beispiel gezeigt.

```
1 void demo_key_handler (swEventKeyPress *e) {
2   button.setCaption("Button pressed");
3 }
4
5 void InitApp (void) {
6   ...
7   button.registerOnButtonDownHandler(demo_key_handler);
8 }
```

Listing 4.10: Registrieren eines Button-Handlers

Über die Funktion *registerOnButtonDownHandler* wird die Funktion (bzw. im Kontext ist dies der Slot) demo_key_handler an den Button gebunden. Wird eine beliebige Taste gedrückt, so wird automatisch diese Funktion aufgerufen. Um nur auf bestimmte Tasten reagieren zu können, muss die swEventKeyPress-Struktur, in der die gesamte Information des Events gespeichert ist, ausgewertet werden.
Man sieht, dass auf diese Art eine sehr einfache und übersichtliche Möglichkeit zur Implementierung der geforderten Funktionalität zur Verfügung steht.

4.4.6 Hintergrundbeleuchtung, LEDs und akustische Ausgaben

Neben der grafischen Anzeige sind auch noch weitere Funktionalitäten in das Menüsystem integriert. Dazu gehören die Steuerung der LEDs und Hintergrundbeleuchtung sowie die Ausgabe von akustischen Meldungen über den Piezo-Beeper. Alle diese Aufgaben werden in jeweils einem eigenen Thread bearbeitet und im Hauptthread des Menüsystems (swApp) verwaltet.

Bei der LED-Ansteuerung stehen folgende Modi zur Verfügung

- LED an oder aus
- Blinken in 3 verschiedenen Geschwindigkeiten
- Alarmanzeige durch bestimmtes Blinkmuster

Die Konfiguration erfolgt über die Funktion *setLEDMode*, an die als Parameter die Nummer der LED (0 bis 3) und der Modus übergeben wird.

Für die Hintergrundbeleuchtung stehen ebenfalls einige Optionen zur Verfügung. Das Dimmen wird über die Funktion *BacklightDimmer* durchgeführt, wobei als Parameter der Zielwert angegeben wird. So kann etwa das Display nach einer bestimmten Zeit auf die Hälfte der möglichen Lichtstärke gedimmt werden. Als optionaler Parameter kann bei dieser Funktion auch eine Zeitdauer für das Dimmen angegeben werden. Weitere zwei Funktionen stehen für das sofortige Ein- und Ausschalten der Hintergrundbeleuchtung bereit.

Die beiden Threads für die LEDs und die Hintergrundbeleuchtung werden auf zwei unterschiedliche Arten realisiert. Beide laufen natürlich in einer Endlosschleife, beim LED-Thread wird jedoch immer 100 ms gewartet und dann eine Prüfung durchgeführt, ob sich der Status einer LED geändert hat. Dies bedeutet gleichzeitig auch, dass die minimale Blink-Frequenz auf 10 Hz festgelegt ist. Der Thread für die Hintergrundbeleuchtung wiederum wird über eine Message-Queue gesteuert – d.h. erst wenn eine entsprechende Nachricht an den Thread geschickt wird,

beginnt dieser mit der Abarbeitung.
Der Grund für diese unterschiedliche Bearbeitungsweise liegt einfach darin, dass die Hintergrundbeleuchtung im Normalbetrieb die meiste Zeit deaktiviert sein wird und somit auch keine ständige Bearbeitung im Thread erfordert.

Der Thread für die akustische Ausgabe ist auf dieselbe Weise realisiert wie der Thread der Hintergrundbeleuchtung. Bisher sind zwei verschiedene akustische Meldungen implementiert - ein kurzer *Klick*, der als Tastenbestätigung dient und ein kurzer *Beep*, für die Bestätigung von Eingaben. Gestartet wird diese Ausgabe über die Funktionen *Click* und *ShortBeep*.

Die im Abschnitt 4.3.3 und 4.3.4 besprochenen Treiber für diese drei Module werden dabei über die Standardschnittstelle des Betriebssystems (fopen, fread, ...) als Dateien angesprochen.

4.4.7 Serielle Kommunikation mit SmartFridge

In diesem und den nächsten beiden Abschnitten werden nun einige Funktionalitäten besprochen, die eine für den Benutzer nicht direkt sichtbare Aufgabe haben. Den Anfang macht die Kommunikation der SmartPanel-Hardware mit der SmartFridge-Hardware.

Der Sinn der Kommunikation zwischen den beiden Geräten ist die Übertragung und Abfrage der aktuellen Daten. Dabei werden nicht die Standard-C Funktionen für die Kommunikation mit der seriellen Schnittstelle verwendet, sondern die *asio*-Implementierung [59] der Boost-C++-Bibliothek. Um eine möglichst einfache und skalierbare Möglichkeit zur Kommunikation zu haben, wurden zwei getrennte Klassen programmiert.

Die erste ist für die grundlegende Kommunikation über die serielle Schnittstelle zuständig. Neben den elementaren Funktionen zum Öffnen und Schließen der Schnittstelle stellt diese Klasse auch Funktionen zum Einlesen und Schreiben von C++-Strings bereit. Auch stehen Funktionen zum Einlesen von ganzen Zeilen oder zum Warten auf bestimmte Zeichen zur Verfügung. Diese Funktionen erleichtern in der darauf aufbauenden Klasse die Kommunikation mit dem SmartFridge-Terminal (siehe Abschnitt 3.2.9). Für die Übertragung von größeren Datenmengen wird das XModem-Protokoll verwendet, das ebenfalls in diese Klasse implementiert wurde. Die Übertragung von Daten über dieses Protokoll von der SmartFridge zum SmartPanel wird dann über den einfachen Befehl *XModemReceiveFile* angestoßen.

Die zweite Klasse baut auf der zuvor besprochenen Klasse auf. Hier werden nun die Befehle implementiert wie sie auch direkt über das Terminal eingegeben werden. So kann etwa über die Funktion „*getVersion*“ die Versionsnummer der SmartFridge-Firmware ausgelesen werden, oder über „*getRAMEventList*“ die Liste aller aktuell im RAM gespeicherten Events.
Über eine Header-Datei sind auch alle notwendigen Datenstrukturen der SmartFridge-Firmware verfügbar. Will man also etwa die aktuelle Konfiguration der SmartFridge auslesen, so kann dies wie in Listing 4.11 geschehen.

```
1 st_sf_setup sf_setup; // Struct für SmartFridge Setup
2
3 SmartFridgeSerial sfserial("/dev/ttyUSB0", 57600);
4 sfserial.getSmartFridgeSetup(sf_setup);
```

Listing 4.11: Abrufen der Setup-Daten von der SmartFridge

Die Funktionen zur Kommunikation mit der SmartFridge werden einerseits vom Menüsystem zur Anzeige der aktuellen Daten am Display verwendet, gleichzeitig aber auch vom FCGI-Server, auf den im Abschnitt 5.2.1 eingegangen wird.

4.4.8 Datenbank

Über die SmartFridge-Hardware werden in einem Abstand von 5 Minuten die gemessenen Daten im RAM und FRAM gespeichert. Im RAM liegen dabei immer Daten der letzten Stunden während im FRAM einige Tage gespeichert werden können. Ebenfalls im RAM und FRAM gespeichert werden ein Logfile und eine Eventliste.

Für die längerfristige Datenspeicherung ist das SmartPanel zuständig, das praktisch gesehen mit der verwendeten SD-Karte genügend Kapazität zur Verfügung hat um alle Datensätze über eine beliebige Zeitdauer zu speichern. Zur Speicherung der Temperaturwerte werden z.B. pro Eintrag 18 Byte benötigt, bei einer Speicherung alle 5 Minuten also 5184 Byte pro Tag. Selbst wenn man also nur 128 MByte für die Temperaturdaten zur Verfügung stellen würde, wäre dies ausreichend für über 70 Jahre.

Es gibt nun prinzipiell mehrere Möglichkeiten die Daten zu speichern. Eine Möglichkeit ist es, die Daten in Dateien abzulegen - eventuell getrennt nach Tagen oder Monaten. Das Speichern ist hierbei zwar relativ einfach, will man die Daten dann aber irgendwann auslesen oder weiterverarbeiten, wird es schon etwas schwieriger, da die einzelnen Dateien wieder zu einem Datensatz zusammengefügt werden müssen.
Eine etwas verbesserte Variante wäre das strukturierte Speichern in XML-Dateien. Hierbei könnten dann existierende XML-Parser eingesetzt werden um das Auslesen der Daten zu vereinfachen.

Die flexibelste Möglichkeit stellt aber die Verwendung einer Datenbank dar. Es existieren dabei grundsätzlich verschiedene Modelle zur Strukturierung der Daten, wie etwa objektorientiert oder, was hier zum Einsatz kommt, relational [Bux09, Chu07]. Ein relationales Datenbanksystem verwaltet und verknüpft dabei die Daten über Tabellen.

Ein Datenbanksystem besteht aus zwei Teilen: der eigentlichen Datenbank, in der die physische Speicherung der Daten erfolgt und dem Datenbank-Verwaltungssystem (*Database Management System* oder kurz *DBMS*), das für die Verwaltung und den Zugriff auf die Daten zuständig ist. Eine Datenbank bietet gegenüber dem Speichern in Datei unter anderem folgende Vorteile [Bux09]:

- Abfragesprache SQL
- sog. Transaktionen um die Daten sicher in die Datenbank einfügen zu können
- Skalierbarkeit

Es existieren verschiedene Datenbanksysteme von denen viele auch frei verwendet werden können. Eine gute Übersicht bietet die Webseite [95]. Für diese Diplomarbeit befanden sich aber nur zwei Datenbanken in der engeren Auswahl, zum einen mySQL [80] und zum anderen SQLite [85]. Vor der Entscheidung für eines der beiden Systeme musste evaluiert werden, welche Anforderungen überhaupt gestellt werden und welche Vor- und Nachteile beide jeweils haben [Kre10, Owe06]. Hierzu in Tabelle 4.5 ein kurzer, nicht vollständiger Vergleich mit einigen wichtigen Punkten.

Die Wahl fiel auf SQLite. Dieses Datenbanksystem wird nicht nur in verschiedenen Embedded-Systems eingesetzt, sondern kommt auch im Apple-Betriebssystem Mac OS X, in Webbrowsern wie Firefox oder auch als Teil von PHP5 zur Anwendung.

Im nächsten Schritt musste die Integration der SQLite-Funktionalität in die Applikation durchgeführt werden. SQLite stellt zwar eine API (Applikation Programming Interface) zur Verfügung,

Tabelle 4.5: Vergleich einiger Punkte zwischen mySQL und SQLite

Eigenschaft	mySQL	SQLite
Server-Prozess	ja	nein
Speicherverbrauch	mittel	gering
Transaktionen	ja	ja
Installation notwendig	ja	nein
max. Datenbankgröße	2-16TB [39]	2 TB
Zugriffsrecht über	Datenbank	Dateisystem

diese kann aber nicht objektorientiert verwendet werden. Um die Programmierung zu vereinfachen, wurde deswegen auf eine sog. Wrapperklasse zurückgegriffen, welche die SQLite-API in einer Klasse kapselt. Auch hier existieren wieder einige verschiedene Realisierungen. Tabelle 4.6 zeigt einen Vergleich der getesteten SQLite-Wrapper.

Tabelle 4.6: Vergleich von Wrappern für die SQLite Datenbank

Name	Codezeilen	Größe Bibliothek	Besonderheit	Doku
SOCI C++ Database Access	10775	≈2.21 MB	SQLite nur als Backend	umfrangreich Beispielcode
Kompex SQLite Wrapper	1152	≈1.86 MB		Doxygen Beispiele
sqlitewrapped	992	≈1.84 MB		keine
sdsqlite	250	≈1.3 MB	sehr simpel gehalten	nur Beispiele minimal

Die Anzahl der Zeilen (reine Codezeilen, ohne Leerzeilen und Kommentare) wurde dabei wieder mit dem Tool CLOC [60] bestimmt. Natürlich ist die reine Anzahl der Zeilen oder die Größe der erstellten Bibliothek kein Maß für die Qualität einer Software. Sie gibt aber in diesem Fall einen guten Richtwert wie viel Aufwand in den Wrapper investiert wurde und wie komfortabel damit der Zugriff auf die Datenbank erfolgen kann.
Als bester Kompromiss erwies sich der Kompex SQLite Wrapper. Dieser ist gegenüber dem SOCI C++ Wrapper wesentlich kleiner, bietet aber trotzdem eine komfortable Schnittstelle für SQLite. Abbildung 4.19 zeigt ein Schema wie der Zugriff auf die Datenbank aus der Applikation erfolgt.

Nachdem nun die Schnittstelle zwischen Applikation und Datenbank verfügbar ist, muss noch geklärt werden, wie die Daten in der Datenbank abgelegt werden sollen.
Benötigt werden vier Tabellen, eine für die Temperaturdaten, zwei Tabellen für die Log-Einträge von SmartFridge und SmartPanel und eine für die aufgetretenen Events. Tabelle 4.7 zeigt die notwendigen Tabellen mit den Datentypen.

Die Spalte *Timestamp* stellt einen sog. UNIX-Timestamp dar, also die Zeit seit dem 1.1.1970 in Sekunden. Das Feld *Date* beinhaltet die gleiche Information nur in einer direkt lesbaren Textform, die Felder *eval* bis *ext2* geben die Temperaturwerte der NTC Sensoren an und *SHTTemp* den Temperaturwert des Sensirion-Sensors (siehe Abschnit 3.2.5). *SHTHum* speichert demnach die gemessene Luftfeuchtigkeit dieses Sensors. Die markierten Felder repräsentieren den sog.

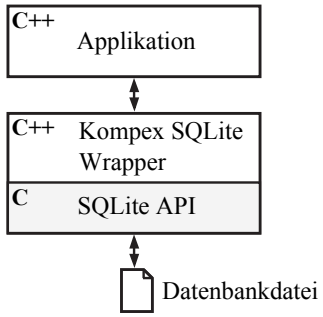

Abbildung 4.19: Schema des Zugriffs auf die Datenbank aus der Applikation

Tabelle 4.7: Struktur der Datenbanktabellen

tSensordata								
NUMERIC	*DATETIME*	*REAL*	*REAL*	*REAL*	*REAL*	*REAL*	*REAL*	*REAL*
Timestamp	Date	evap	cooler	freezer	ext1	ext2	SHTTemp	SHTHum

tSFEvents, tSFLog, tSPLog			
NUMERIC	*DATETIME*	*REAL*	*REAL*
Timestamp	Date	type	value, line

Primary-Key, über den jeder Eintrag in der Tabelle eindeutig identifiziert werden kann. Da die Speicherung der Sensordaten nur alle 5 Minuten erfolgt, kann hier durch den Timestamp alleine eine eindeutige Unterscheidung getroffen werden. Im Fall der Log- oder Eventeinträge können jedoch zu einer bestimmten Sekunde auch mehrere Ereignisse stattfinden, weshalb hier zusätzliche Felder verwendet werden müssen.

Damit in der Menüsystem-Applikation nicht direkt mit Funktionen für den Datenbankzugriff gearbeitet werden muss, wurde eine zusätzliche Datenbankklasse erstellt, die auf dem Kompex-Wrapper aufbaut und weiter vereinfachte, für die Anwendung aufbereitete Funktionen zum Zugriff auf die Daten zur Verfügung stellt.
So kann etwa über den Funktionsaufruf *FRAMDataFileToDatabase* ein von der SmartFridge-Hardware über die serielle Schnittstelle ausgelesenes Datenfile direkt in die Datenbank eingetragen werden. Ähnlich wird über die Funktion *EventsToDatabase* eine beliebige Anzahl an Eventeinträgen in der Datenbank hinzugefügt. Eingetragen werden die Daten dabei beispielhaft mittels folgender SQL-Anweisung

- *INSERT OR IGNORE INTO* tSensordata *VALUES* (12345678, '20.10.2010', 1, 2, 3, 4, 5, 6, 7)

Besonders hervorzuheben ist der Befehl „*INSERT OR IGNORE*“, der dafür sorgt, dass bereits vorhandene Einträge ignoriert werden. Da beim Auslesen aus der SmartFridge nicht geprüft wird welche Daten bereits in der Datenbank vorhanden sind und sich die ausgelesenen Daten im Normalfall immer mit den Daten in der Datenbank überschneiden werden, ist dies notwendig um keine Fehlermeldungen zu erhalten.
Das Eintragen von Datensätzen geschieht immer in sog. *Transaktionen*. Tritt während einer Transaktion ein Fehler auf, so werden alle Einträge innerhalb der Transaktion nicht übernommen (es wird ein sog. *Rollback* durchgeführt). Das nachfolgende Listing 4.12 zeigt, wie dies prinzipiell über den Kompex-Wrapper realisiert wird.

```
1 Kompex::SQLiteDatabase *pDatabase =
2   new Kompex::SQLiteDatabase(DATABASE_FILE, SQLITE_OPEN_READWRITE, 0);
```

```
3 Kompex::SQLiteStatement *pStmt = new Kompex::SQLiteStatement(pDatabase);
4
5 string query = "'INSERT OR IGNORE ..."'
6
7 pStmt->BeginTransaction();
8 pStmt->Sql(query);
9 // Hier erfolgt Eintragen der Daten
10 ...
11 pStmt->CommitTransaction();
```

Listing 4.12: Datenbank-Transaktion mit dem Kompex Wrapper

Es gibt noch einen weiteren Mechanismus zur Geschwindigkeitssteigerung, und zwar die sog. *Prepared Statements*. Hierbei wird dem Datenbanksystem die SQL-Anweisung mit Platzhaltern übergeben. Die Werte selbst können dann über *Bind*-Kommandos in die entsprechenden Spalten eingefügt und mit einem *Step*-Kommando übernommen werden. Als Beispiel zeigt hierzu Listing 4.13 wie mittels der Funktion *LogToDatabase* Logeinträge in die Datenbank geschrieben werden.

```
1 string query = "INSERT OR IGNORE INTO tSFLog VALUES (?,?,?,?)";
2
3 pStmt->BeginTransaction();
4 pStmt->Sql(query);
5
6 for (vector<st_log>::iterator it=log.begin(); it!=log.end(); ++it) {
7   entry = *it;
8   pStmt->BindDouble(1, entry.timestamp);
9   ...
10  pStmt->BindInt(4, entry.line);
11  pStmt->Step();
12 }
13
14 pStmt->ClearBindings();
15 pStmt->CommitTransaction();
```

Listing 4.13: Beispiel zu Prepared Statements mit dem Kompex Wrapper

4.4.9 Schnittstelle zur Webseite

Der letzte Punkt bei der Auflistung über die Aufgaben der Applikation war die Realisierung einer Schnittstelle zur Webseite.
Hierbei geht es darum der Webseite bzw. dem Webserver diejenigen Daten zu liefern, die nicht aus der Datenbank ausgelesen werden können. Prinzipiell wäre es eine etwas saubere Implementierung, wenn dies nicht über die Applikation für das Menüsystem geschehen würde. Da das Menüsystem aber einen ständigen Kontakt zur SmartFridge benötigt und somit die serielle Schnittstelle beansprucht, muss diese Datenbankschnittstelle hier implementiert werden.
Die Beschreibung, wie die Schnittstelle implementiert ist, folgt aus Gründen der Konsistenz im Kapitel 5 zum Webinterface.

5 Webinterface

In diesem Kapitel wird nun gezeigt, wie die durch die SmartFridge bereitgestellten Daten über die Netzwerkschnittstelle des SmartPanels schlussendlich auf einer Webseite präsentiert werden können. Dabei kommen verschiedenste Technologien zum Einsatz, angefangen bei der Schnittstelle zwischen Applikation und Webserver bis hin zur JavaScript-Bibliothek für die Diagrammdarstellung. Zuerst wird hierzu auf die Grundlagen eingegangen, während dann der Reihe nach alle für die Realisierung notwendigen Programmiersprachen und Bibliotheken erklärt werden.

Zusammengefasst werden in diesem Kapitel folgende Themen behandelt:

- Grundlagen zu HTML, CSS, JavaScript
- jQuery, jQuery UI und die Flot Bibliothek
- Ajax für "Web 2.0"
- Auswahl und Funktion des Webservers
- Die Schnittstelle zwischen Webserver und der Applikation (FCGI)
- Layout und Implementierung der EtherFridge Homepage

5.1 Grundlagen und Übersicht

In diesem Abschnitt werden nun die verschiedenen, hier angewendeten Technologien und Sprachen zwischen Webserver und Browser kurz zusammengefasst. Diese Informationen sind dann später bei der Beschreibung einiger Mechanismen elementar. Dabei kann natürlich nicht auf alle Details eingegangen werden, vielmehr sollen hier die für die nächsten Abschnitte relevanten Grundlagen erläutert werden.

Als grundsätzliches Protokoll für die Kommunikation zwischen Webserver und Browser wird das *Hypertext Transfer Protocol* oder kurz *HTTP* verwendet. HTTP selber baut wiederum auf dem *TCP/IP*-Protokoll auf. Für die Grundlagen der unter dem HTTP-Protokoll liegenden Protokolle (wie eben auch TCP/IP) sei aber auf entsprechende Literatur verwiesen.
Die Aufgabe des HTTP-Protokolls ist es, die Daten vom Webserver zum Benutzer in den Browser zu bringen. Dazu wird vom Browser ein sog. *HTTP-Request* geschickt und vom Webserver mit einem *HTTP-Response* beantwortet [GTS+02].
Die Kommunikation wird dabei über einzelne HTTP-Nachrichten (*HTTP-Messages*) durchgeführt, die eine einfache, lesbare aber genau definierte Struktur aufweisen. Es gibt dabei genau zwei Typen, und zwar die *HTTP-Request-Message* und die *HTTP-Response-Message*. Abbildung

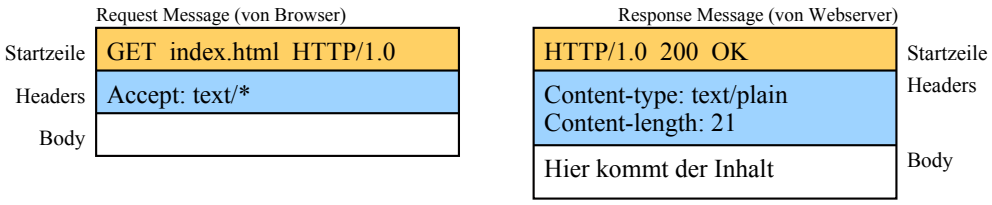

Abbildung 5.1: Beispiel zur Kommunikation mit dem HTTP-Protokoll

5.1 zeigt hierzu ein Beispiel einer Kommunikation, an der einige Details erläutert werden können [GTS+02].

Eine HTTP-Nachricht besteht aus drei Teilen: der Startzeile, dem Header und dem Body. In der Startzeile wird bei der Anfrage vom Browser die gewünschte Ressource angegeben, bei der Antwort vom Server eine Statusmeldung. Im Header können verschiedene Informationen, wie der verwendete Browser oder das Datum, integriert werden. Bei der Antwortnachricht wird zudem der Typ der Daten mitgeschickt, was wichtig für die Interpretation der Daten durch den Browser ist. Im optionalen Body werden schließlich die eigentlichen Nutzdaten übertragen. Der Inhalt dieser drei Nachrichtenteile wird nun etwas ausführlicher behandelt.
Die Startzeile einer HTTP-Request-Message besteht aus drei Feldern: der *Methode*, der *URL* (Uniform Resource Locator) und der HTTP-Version. Die URL gibt den Pfad der vom Browser angeforderten, zu übertragenden Ressource an. Über das Feld *Methode* kann die Aktion, die mit dieser Ressource durchgeführt werden soll, angegeben werden. Die möglichen Methoden sind *GET, POST, HEAD, PUT, TRACE, OPTIONS* und *DELETE*. Davon werden aber im Normallfall nur GET und POST genutzt – beide werden zur Anfrage einer bestimmten Ressource verwendet, der Unterschied wird später noch ausführlicher erläutert. Bei der Antwort vom Server enthält die Statuszeile wieder die HTTP-Version, einen Statuscode und einen Statustext.

Die Headerbereiche der HTTP-Nachrichten bestehen immer aus Paaren von Bezeichnung und zugehörigem Wert, die jeweils in einer eigenen Zeile stehen. Wichtig ist dabei bei der Antwort vor allem der Eintrag *Content-Type*, der die Art der im Body übertragenen Daten spezifiziert. Beispiele hierzu sind *text/css* für eine CSS-Datei oder *image/png* für eine Bilddatei.Im Body werden also nicht nur Daten in Klartext übertragen, sondern im Fall von Bilddateien auch in binärer Form - dies gilt natürlich auch für viele andere Formate, auf die hier nicht eingegangen wird. Eine vollständige Auflistung dieser MIME-Typen findet sich in [92].

Zum Schluss dieses Abschnitts wird nun noch kurz auf den Aufbau einer URL eingegangen, über welche die gewünschte Ressource eindeutig identifiziert werden kann. Eine *URL* weist folgendes Format auf [GTS+02]:

```
<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>
```

Tabelle 5.1 erklärt die einzelnen Elemente.

Die minimale Version einer URL enthält also das Protokoll, die Adresse des Servers und den Pfad zur Ressource. Bei den meisten heute üblichen Browsern ist die Angabe des Protokolls nicht zwingend notwendig, da bei Fehlen dieses Elementes automatisch das HTTP-Protokoll angenommen wird. Auch der Pfad zur Ressource kann praktisch oft weggelassen werden, da der Webserver üblicherweise einen leeren Pfad automatisch als Anforderung der Startseite interpretiert. Eine URL *http://www.iiss.oeaw.ac.at/index.html* kann also üblicherweise in der einfacheren

Tabelle 5.1: Elemente einer URL

Element	Beschreibung	Optional
scheme	Protokoll, z.B. http oder ftp	
user	Benutzername wenn Identifikation erforderlich	•
password	Passwort wenn Identifikation erforderlich	•
host	Name oder die IP des Zielservers	
port	Port des Zielcomputers wenn abweichend von Standardwert 80	•
path	Relativer Pfad zu Ressource	
params	Übergabe von Parametern, Bezeichnung/Wert Paar	•
query	Zu params gehörender Wert	•
frag	Ein bestimmter Teil der Ressource (z.B. eine Position in einem HTML-Dokument)	•

Form *www.iiss.oeaw.ac.at* im Browser eingegeben werden.
Auf die Elemente *params* und *query* wird in einem späteren Abschnitt noch genauer eingegangen, da diese für die Übergabe von Parametern an den Server eine fundamentale Rolle spielen.

Bei der Beschreibung der HTTP-Request Nachricht wurden die Methoden *GET* und *POST* zur Abfrage von Ressourcen angesprochen. Der Unterschied zwischen den beiden Methoden besteht kurz gesagt darin, dass mit *GET* die Argumente als Teil der URL übertragen werden und bei *POST* im HTTP-Header. Argumente bei GET sind also im Browser sichtbar (siehe Tabelle 5.1: params und query) und bei POST nicht.
Anwendung findet die POST-Methode vor allem dort, wo größere Datenmengen übertragen werden müssen bzw. wenn die Daten nicht im URL-Eingabefeld des Browser sichtbar sein sollen. Ein Beispiel hierzu sind Formulardaten. Auch in dieser Arbeit wird die POST-Methode zur Abfrage verwendet.

5.1.1 HTML

Die HyperText Markup Language ist eine sog. Auszeichnungssprache, die für die Beschreibung des Inhalts einer Webseite verwendet wird, wobei die Strukturierung durch sog. *Tags* durchgeführt wird [83].

5.1.2 Cascading Style Sheets - CSS

CSS (dies steht für *Cascading Style Sheet*) ist ein Standard, der 1995 vom W3C vorgestellt wurde um den Designern mehr Kontrolle über das Layout der Webseite zu geben [Qui10]. Hierbei erfolgt eine Trennung zwischen den Daten und dem Layout der Webseite. Dies wurde früher alles innerhalb des HTML-Dokuments über Attribute realisiert – mit CSS kann der Stil einer Webseite in einer externen Datei definiert werden, die dann für alle HTML-Seiten eingebunden werden kann. Somit kann eine einheitliche Darstellung des gesamten Internetauftritts erreicht werden [Fri07].

Das wohl wichtigste HTML-Element beim Entwurf von Webseiten mit der Unterstützung von CSS ist der *div-Container*. Dieser Container stellt praktisch gesehen einen frei positionierbaren

Rahmen dar, dem beliebige HTML-Elemente untergordnet werden können. Durch diese freie Positionierung des Containers kann zum einen das Layout pixelgenau vorgegeben werden. Zum anderen sind dadurch in Kombination mit CSS und JavaScript verschiedenste Effekte und Benutzerschnittstellen wie etwa PopUp-Fenster oder verschiebbare Elemente möglich.

Die Identifikation eines div-Containers erfolgt über das Attribut *id*. Ein weiteres Attribute ist *style*, in dem unter anderem die Größe des Containers und die Position angegeben werden.

5.1.3 JavaScript

In den beiden vorangegangen Abschnitten wurde mit CSS eine Möglichkeit zur Gestaltung des Layouts und mit HTML die grundlegende Sprache für den Inhalt vorgestellt. Damit erstellte Webseiten sind aber von statischer Natur und können nur in sehr begrenzter Weise mit dem Benutzer interagieren.
JavaScript ist nun eine Scriptsprache um genau diese Interaktion zwischen dem Benutzer und der Webseite zu ermöglichen. Die Besonderheit hierbei ist, dass JavaScript nicht auf dem Server läuft, sondern im Browser des Benutzers – somit wird für die Ausführung auch dessen verfügbare Rechenleistung aufgewendet und nicht die vom Server. Man spricht hierbei auch von einer *Client-Side Scripting Language*.

Abbildung 5.2 zeigt in einem Schichtenmodell das grundsätzliche Zusammenspiel zwischen HTML, CSS und JavaScript. Mit den beiden ersten Schichten kann das statische Layout einer Webseite beschrieben werden, während über JavaScript ein dynamisches Verhalten integriert werden kann. Der Begriff „dynamisch“ darf hierbei aber nicht missverstanden werden – oft zeigt sich die Scriptsprache nur im Hintergrund, etwa zur Validierung von Eingabefeldern.

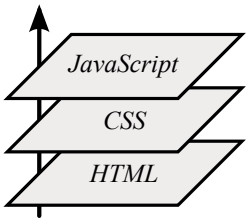

Abbildung 5.2: Schichtenmodell zum Zusammenspiel von HTML, CSS und JavaScript

5.1.4 jQuery

Einen Schritt weiter, als mit reinem JavaScript, geht man mit *jQuery* [70], einer Bibliothek von JavaScript Funktionen. Diese Bibliothek bietet Möglichkeiten zur Manipulation von Elementen des DOM (siehe Abschnitt 5.1.1), Ajax-Funktionalität (folgt im Abschnitt 5.1.6) und verschiedenste Effekte zur Animation. Sie stellt zwar nicht die einzige Bibliothek ihrer Art dar, aber die am meisten genutzte [87].

Die grundsätzliche Struktur bei jQuery folgt dem *Fluent Interface* [93], d.h. es findet eine Verkettung von Befehlen statt. Eine besondere Stellung nimmt dabei die Aktion *ready* ein. Mit dieser ist es möglich den JavaScript Code erst dann auszuführen, wenn die Seite vom Browser vollständig geladen wurde. Dies ist insbesonders dann wichtig, wenn auf Elemente in der Webseite zugegriffen wird. Listing 5.1 zeigt wie diese Aktion verwendet wird.

```
<script type="text/javascript">
  \$(document).ready(function(){
    ... Hier der jQuery/JavaScript Code
  });
</script>
```

Listing 5.1: Die *ready*-Funktion von jQuery

5.1.5 Die Flot-Bibliothek und JSON

Um die bei den Messungen erhaltenen Daten grafisch ansprechend darstellen zu können, wird das jQuery Plugin *flot* verwendet [66].
Die flot-Library bietet verschiedene Diagrammarten, die auf einfache, aber trotzdem vielfältige Weise innerhalb des JavaScript-Codes erstellt werden können. Die Generierung der Diagramme erfolgt also komplett auf der Client-Seite. Die folgende Codezeile zeigt die grundsätzliche Struktur des Aufrufs zum Zeichnen eines Diagrammes.

```
var plot = $.plot(placeholder, data, options)
```

Als *placeholder* wird im allgemeinen ein Div-Container (siehe 5.1.1) verwendet, dessen Position und Größe dann auch dem Diagramm zugeordnet werden. Danach folgen im Parameter *data* die im Diagramm darzustellenden Daten und im letzten Feld noch verschiedene Optionen – hier sei auf die Dokumentation von *flot* verwiesen [67].

Als Datenstruktur für die Diagrammdaten kommt die *JavaScript Objet Notation*, oder kurz *JSON* zum Einsatz [69]. Diese kann von JavaScript und auch der flot-Bibliothek direkt verarbeitet werden, wodurch keine Konvertierung notwendig ist und eine hohe Geschwindigkeit bei der Verarbeitung erreicht werden kann. [Eventuell etwas über erste Versuche mit eigenem Format]

JSON ist aber nicht auf JavaScript beschränkt, sondern stellt ein universelles, menschenlesbares und trotzdem für Computer einfach zu parsendes Format dar. Es kann dabei zwischen zwei elementaren Strukturen unterschieden werden: dem *object* und dem *array*. Das *array* stellt, wie in Programmiersprachen, eine geordnete Liste von Werten dar. Die *object*-Struktur ist eine ungeordnete Menge von Paaren mit Name und einem „Wert“, die über einen Doppelpunkt voneiander getrennt werden. Der Wert in einem *object* ist dabei aber nicht auf Zahlen beschränkt, sondern kann u.a. auch ein weiteres Objekt, ein String oder ein Array sein. Um dies zu veranschaulichen, folgt ein Beispiel, bei dem ein Objekt (geschwungene Klammer) mit zwei Namen/Wert-Paaren angelegt wird. Der erste Objekteintrag *label* verwaltet einen String, während der zweite Eintrag ein array von Zahlen speichert.

```
{ label: "Daten", data: [ [1,5],[2,3],[3,-1] ] }
```

Wie JSON hier genau verwendet wird, folgt in Abschnitt 5.4.

5.1.6 Ajax

Ajax, oder auch *Asynchronous JavaScript And XML*, ist eine Möglichkeit um nur Teile einer Webseite übertragen zu müssen, wenn einzelne Elemente geändert werden sollen. Dabei stellt Ajax eigentlich keine eigene Technologie dar, sondern vereint lediglich einige andere Technologien unter einem Dach.
Neben HTML und JavaScript ist wohl das XMLHttpRequest-Objekt die wichtigste davon – erst durch dieses wird ein asynchrones Übertragen von Daten zwischen Browser und Webserver ermöglicht. Asynchrone Übertragung bedeutet hierbei, dass die Abfrage von Daten zu jeder Zeit, unabhängig davon ob der Benutzer eine Aktion angestoßen hat, stattfinden kann.

Über die in Abschnitt 5.1.4 beschriebene JavaScript Bibliothek *jQuery* reduziert sich der Aufwand für die Durchführung von Ajax-Request auf einfache Befehle. So kann etwa mit der jQuery Aktion *load* ein HTML-Code nachgeladen werden. Hier bietet sich ein dynamisches Generieren über Scriptsprachen wie PHP an. Listing 5.2 zeigt ein Beispiel, in dem nach dem Click auf einen Link HTML-Code übertragen und an die Stelle *content* eingefügt wird.

```
$("ajaxlink").click(function() {
  $("content").load("ajax.php");
});
```

Listing 5.2: Beispiel zu Ajax mittels der *load*-Aktion von jQuery

Abbildung 5.3 verdeutlicht den Ablauf der Kommunikation schematisch. Zuerst wird über das HTTP-Protokoll die gewünschte Webseite geladen. Wird auf den im Listing 5.2 verwendeten Weblink „*ajaxlink*“ gedrückt, so lädt der Browser über einen *XMLHttpRequest* den gewünschten Teil nach. Hierzu kommt im Beispiel ein PHP-Script zum Einsatz, woduch der HTML-Code auch dynamisch generiert werden kann.

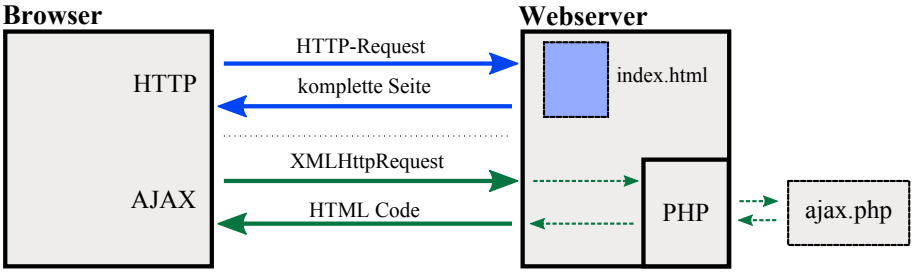

Abbildung 5.3: Kommunikationsbeispiel zum Laden eines Teiles der Webseite mittels Ajax

5.2 Webserver

Bisher wurde beschrieben welche Technik hinter einer Webseite steht. Genauso wichtig ist aber die Frage von wo die Webseite überhaupt geladen wird - nämlich von einem Webserver.

Auf dem Webserver läuft ein Dienst der auf Anfragen von außen reagiert und die Ressourcen dem Netzwerk zur Verfügung stellt. Im einfachsten Fall wird dabei eine HTML-Datei vom Server zum Client übertragen. Für komplexere Webseiten sind meist noch unterstützende Funkionen wie ein sog. *Common Gateway Interface* (CGI) oder die Anbindung einer Datenbank notwendig.

Die CGI-Schnittstelle ermöglicht es, dass der Webserver Daten einbinden kann, die von externen Quellen geliefert werden. Auf diese Weise können dynamische Inhalte realisiert werden.

Für den Einsatz auf der SmartPanel Hardware wurde der Lighttpd-Webserver [73] gewählt. Dieser weist einen geringeren Speicherverbrauch als etwa der Apache-Webserver auf, was für ein Embedded-System einen relevanten Entscheidungsfaktor darstellt. Es gibt zwar auch noch kleinere Webserver, die von der Leistungsfähigkeit durchaus geeignet wären – von diesen getesteten Webservern stellte aber keiner eine FastCGI-Schnittstelle zur Verfügung.

5.2.1 FastCGI

Um die Daten nun von der Applikation in die Webseite integrieren zu können wird die sog. *FCGI*-Schnittstelle des Lighttpd-Webservers genutzt. FastCGI steht dabei für *Fast Common Gateway Interface* - FastCGI stellt also eine Weiterentwicklung zum regulären CGI dar.

Der Unterschied zwischen CGI und FastCGI liegt nun darin, dass bei CGI der Webserver bei jeder Kommunikation das CGI Programm neu laden muss. Dies führt zwangsläufig zu einem gewissen Overhead durch das Betriebssystem. FastCGI hingegen wird nur einmal gestartet und nimmt ab diesem Zeitpunkt die Anfragen des Webservers entgegen.

Die eigentliche Kommunikation läuft bei CGI über die Umgebungsvariablen, während dies bei FastCGI über eine Netzwerkverbindung realisiert wird. Dies hat gleichzeitig auch den Vorteil, dass sich der FastCGI-Server theoretisch nicht auf derselben Hardware wie der Webserver befinden muss.

Wie bei SQLite (siehe Abschnitt 4.4.8) wurde auch hier ein Wrapper verwendet, um einen objektorientierten und möglichst komfortablen Zugriff auf die FastCGI-API zu erhalten. Hierzu wurden zwei verschiedene Implementierungen getestet, von denen sich der *fastcgipp*-Wrapper [65] als am besten geeignet erwies.

Listing 5.3 zeigt ein einfaches Beispiel zur Anwendung des fastcgipp-Wrappers, das bei Aufruf über den Webserver ein einfaches „Hello World“ im Browser anzeigt. Hierbei wird zuerst die eigene Klasse *HelloWorld* von *Fastcgipp::Request* abgeleitet. In der Klasse *HelloWorld* wird anschließend die Funktion *response* definiert, in der die Bearbeitung ankommender Nachrichten und die Generierung der Antwort über das Objekt *out* durchgeführt wird. In diesem Beispiel wurden keine Parameter übergeben, weshalb nur eine „HelloWorld“-Webseite geniert und an den Webserver geschickt wird. Zuvor muss aber in der main-Funktion ein Objekt dieser Klasse erstellt und gestartet werden.

```
1  #include <fastcgi++/request.hpp>
2  #include <fastcgi++/manager.hpp>
3
4  class HelloWorld: public Fastcgipp::Request<char> {
5    bool response() {
6      out << "<html><head>"
7      out << "<meta http-equiv='Content-Type' content='text/html' />";
8      out << "<title>Hello World</title></head><body>";
9      out << "Hello World";
10     out << "</body></html>";
11   }
12 }
13
```

```
14 int main() {
15   Fastcgipp::Manager<HelloWorld> fcgi;
16   fcgi.handler();
17 }
```

Listing 5.3: Beispiel zur Anwendung des fastcgipp-Wrappers

5.3 Layout der Webseite

Bevor ein Konzept für das Layout der Webseite erstellt werden konnte, musste zuerst einmal festgelegt werden, was überhaupt dargestellt werden sollte. Folgende Funktionen sollte die Webseite bieten:

- Anzeige der aktuellen Temperaturwerte
- Anzeige der Temperaturverläufe der letzten 24 Stunden
- Statistik mit Spitzenwerten
- Liste der Logfile-Einträge
- Liste der aufgetretenen Ereignisse
- Konfiguration der Solltemperaturen
- Aktivieren/Deaktivieren der Kindersicherung
- Anzeige des aktuellen Alarmstatus
- Bei aktivem Alarm Möglichkeit zur Bestätigung

Natürlich könnten über die Webseite wesentlich mehr Daten der Kühl-Gefrierkombination angezeigt und konfiguriert werden (etwa die Zeitpunkte des Öffnens der Tür oder die Regelparameter). Da das Prinzip dahinter aber immer gleich ist, wird hier zur Demonstration nur eine Untermenge realisiert.
Die Darstellung der oben angeführten Daten wurde auf die folgenden vier Seiten verteilt:

- Statusseite
- Temperaturprofile
- Events und Statistiken
- Administration

Die Anzeige der aktuellen Temperaturen und der Luftfeuchtigkeit erfolgt auf der *Statusseite*. Zusätzlich werden hier auch die Sollwerte für den Kühl- und Gefrierteil dargestellt. Auf der Seite *Temperaturprofile* können die Verläufe der letzten Stunden eingesehen werden. Auf der Seite *Events und Statistiken* erfolgt die Darstellung des Event-Logs und der aktuellen Spitzenwerte (Tag, Monat). Über die Seite *Administration* können schlussendlich die Sollwerte für Kühl- und Gefrierbereich eingestellt werden. Zusätzlich erfolgt hier auch die Anzeige der Log-Einträge und Alarme. Letztere können über einen Button bestätigt werden.

Das Ziel bei der Gestaltung der Webseite war es, sie so einfach und funktionell wie möglich, aber trotzdem auch optisch ansprechend zu halten. Zudem sollte die Seite keinen unnötigen Ballast in Form von umfangreichen Grafiken mitbringen, da ein schnelles Laden der Seiten an oberster Stelle steht. Bei der Farbwahl wurde auf kühle, ruhige Farben gesetzt um einen Bezug zur Thematik „Kühlen“ herzustellen. Hierbei ist es sinnvoll nicht zu viele verschiedene Farben einzusetzen und zu Beginn ein durchgängiges Farbschema für die komplette Webseite zu erarbeiten [Hof09].

Wie schon in den vorangegangenen Abschnitten erläutert, wird bei der Darstellung der Seite HTML für den Inhalt und CSS für das Layout und die Strukturierung verwendet. Zuerst wird nun hier umrissen wie der Inhalt der einzelnen Seiten realisiert wurde. Die Statusseite ist relativ einfach gestaltet, da hier nur die aktuellen Werte angezeigt werden müssen. Um hier die Informationen besser hervorzuheben, werden zusätzlich zur Standardschriftgröße zwei weitere Schriftgrößen verwendet. Die Unterscheidung von Unterpunkten erfolgt auf allen Seiten mittels grafischen Punkten am linken Rand. Für die Darstellung der Log- und Eventeinträge werden Tabellen und für die Generierung des Temperaturprofils die *flot*-Bibliothek (Abschnitt 5.1.5) benutzt.
Am umfangreichsten ist die Seite für die Administration. Die Darstellung und Modifikation der Solltemperaturen geschieht über Schieberegler, die mittels der *jQuery UI*-Bibliothek (Abschnitt 5.1.4) dargestellt werden. Die Einstellung der Kindersicherung kann über eine Standard-Checkbox beeinflusst werden. Ein Button ermöglicht schlussendlich die Übernahme von Änderungen dieser Einstellungen. Die Anzeige der Alarmzustände erfolgt in einem Unterpunkt der Administrations-Seite. Hierzu werden die *Highlight*- und *Error*-Darstellung von *jQuery UI* angwendet, die einen div-Container entsprechend der Wichtigkeit des Alarmzustands farblich kennzeichnen. Die Farbe gelb wird zur Anzeige eines bestätigten und rot für einen unbestätigten Alarmzustand verwendet. Die Bestätigung kann somit nicht nur an der Bedieneinheit des Smart-Fridge (siehe Abschnitt 3.2.10), sondern über einen Button auch über das Webinterface erfolgen.

Das Layout wird über eine externe css-Datei festgelegt, die von jeder der vier Seiten eingebunden wird. Dabei wird Gebrauch von den in Abschnitt 5.1.2 angesprochenen div-Containern gemacht, um die einzelnen Teile der Seite anzuordnen. Abbildung 5.4 zeigt hierzu schematisch das Grundgerüst.

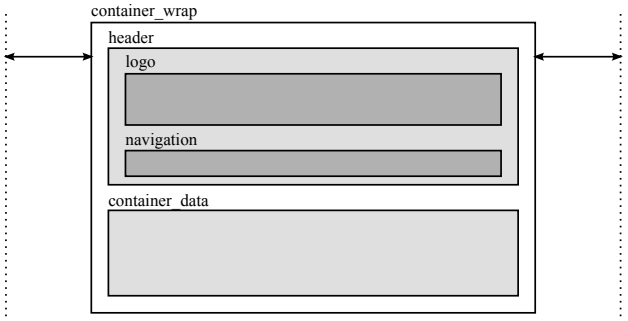

Abbildung 5.4: div-Container Grundgerüst der Webseite

Der Hauptcontainer wird durch *container_wrap* repräsentiert, der auf der Seite zentriert dargestellt wird. In diesem befindet sich dann der *header* in dem das Logo der Seite (div *logo*) und die Navigationsleiste (div *navigation*) dargestellt werden.
Im div *container_data* erfolgt die eigentliche Darstellung der Information, wobei es hier je nach Seite noch zu weiteren Unterteilungen mittels div-Containern kommt, die hier aber nicht dargestellt werden.

Abbildung 5.5 zeigt als Beispiel die Seite für die Administration. Um die Aufteilung in Information und Layout durch HTML und CSS zu veranschaulichen, wird dies in der linken Seite mit CSS und auf der rechten Seite ohne Einbindung der CSS-Datei dargestellt.

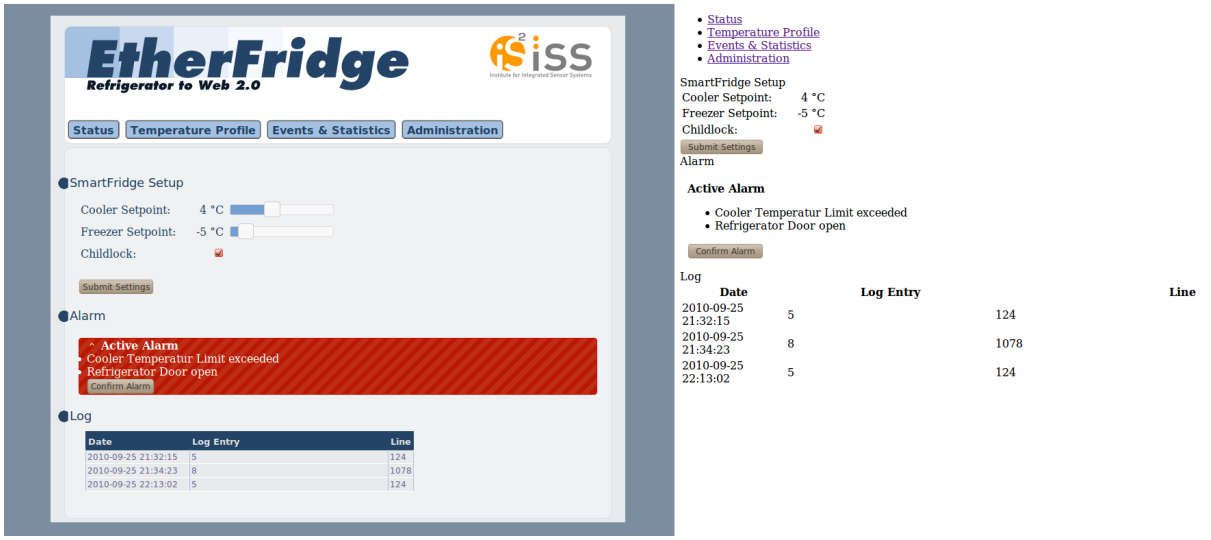

Abbildung 5.5: Administrationsseite und Vergleich mit und ohne CSS

5.4 Implementierung des Webinterfaces

In diesem Abschnitt wird nun erklärt, wie die beschriebenen Technologien zusammenwirken müssen, um eine dynamisch agierende Webseite zu erhalten.

Abbildung 5.6 zeigt hierzu ein Schema der verschiedenen Komponenten, beginnend bei der Datenerfassung bis zur Anzeige in einem Browser.
Die größte Bedeutung hat bei dieser Darstellung natürlich das SmartPanel. Die Verbindung zur SmartFridge wird über die Menüsystem-Applikation hergestellt, die einerseits die Daten auslesen, aber die SmartFridge auch steuern kann. Zusätzlich hat das Menüsystem die Aufgabe die ausgelesenen Daten in die Datenbank zu übertragen (siehe Kapitel 4.4.8).
Der Webserver, der auf der SmartPanel-Hardware läuft, nimmt die Anfragen vom Netzwerk entgegen und stellt die Verbindung zur FCGI-Applikation her. Die FCGI-Applikation wiederum kommuniziert einerseits mit der Datenbank um aktuelle Temperaturverläufe zu erhalten und anderseits direkt mit dem Menüsystem um die Steuerung und Abfrage aktueller Zustände der SmartFridge-Hardware zu ermöglichen.

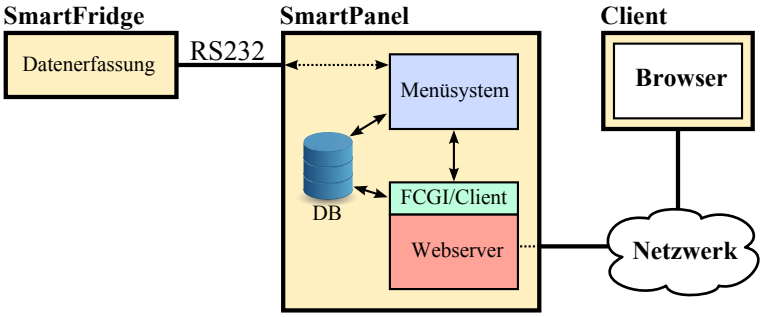

Abbildung 5.6: Übersicht über das Gesamtsystem

Das zu realisierende Gesamtsystem kann also in folgende Funktionsblöcke zerlegt werden:

- Schnittstelle und Protokoll zur seriellen Kommunikation (siehe Abschnitt 4.4.7)

- Menüsystem (siehe Abschnitt 4.4)
- Datenbankverwaltung (siehe Abschnitt 4.4.8)
- FCGI Client Applikation
- Webseite (siehe Abschnitt 5.3)

Der Webserver wird in der Auflistung bewusst nicht angeführt, da dieser lediglich installiert und entsprechend eingerichtet werden musste.

Am Beispiel der Anzeige des Temperaturverlaufs soll gezeigt werde, wie die Daten von der Smart-Fridge in die Darstellung auf der Webseite integriert werden. Die Generierung der dynamischen Daten für die anderen Seiten erfolgt hierzu äquivalent.

Ausgangspunkt der Kommunikation zwischen Webseite und dem FCGI-Client ist eine JavaScript Funktion *getTempData*, in der über die *ajax*-Methode von jQuery eine asynchrone Datenübertragung angestoßen wird, siehe Listing 5.4. Der Aufruf dieser Funktion erfolgt direkt nach dem Laden der Seite, könnte zusätzlich aber auch ohne weiteres über einen „Aktualisieren"-Button angestoßen werden.

```
function getTempData() {
  $.ajax({
    url: "/cgi-bin/dispatcher.fcgi?id=temp_hours",
    type: "POST",
    data: "hours=24&cooler=1&freezer=1&shttemp=1",
    data_type: "json",
    success: function(json_data){
      var json_array = $.parseJSON(json_data);
      $.plot($("#temp1"), json_array, {xaxis: {mode: "time"}});
    },
  });
}
```

Listing 5.4: Anstossen einer Ajax-Aktion über die getTempData-Funktion

Im Parameter *url* wird die Ressource angegeben, wobei als Parameter eine Identifikation (id) übergeben wird, mit welcher der Client eine Unterscheidung über die angeforderten Daten treffen kann. Bei diesem Beispiel ist der Temperaturverlauf gefragt, deswegen auch *temp_hours*.
Der Wert im Parameter *type* gibt an, dass der Text im Parameter *data* nicht an die URL angehängt, sondern über die POST-Methode im HTTP-Header übertragen werden soll. Konkret wird hier mit der Anfrage in *data* der Temperaturverlauf der letzten 24 Stunden für die Sensoren im Kühl- und Gefrierraum und des SHT11-Sensors angefragt.
Der nächste Parameter *data_type* legt das Format der Antwortdaten fest – hier konkret im JSON-Format (siehe Abschnitt 5.1.5).
Besonders wichtig ist nun der Paramter *success*, in den eine Funktion eingetragen wird, die bei erfolgreicher Anfrage ausgeführt wird. Die Funktion erhält dabei die Daten im Parameter *json_data*, die dann im ersten Schritt über die jQuery-Funktion *parseJSON* in ein Array konvertiert werden. Über die *plot*-Funktion der flot-Bibliothek erfolgt dann die eigentliche Anzeige des Diagramms im div-Container mit der id *temp1*.

Die Aufgabe des FCGI-Clients ist es, die von der Webseite (bzw. eigentlich dem Webserver) übertragene Anforderung auszuwerten, eine entsprechende Antwort zu generieren und diese dann zurückzusenden. Der Client stellt also das Bindeglied zwischen Webserver und der Datenbank bzw. Menüsystem dar. Listing 5.5 zeigt hierzu wie die Auswertung der Anfrage und die Generierung der Daten im JSON-Format speziell im Fall des Temperaturverlaufes realisiert wurden.

Zugunsten der Übersichtlichkeit und um das Listing möglichst kurz zu halten, werden dabei nur Auszüge aus dem Quellcode dargestellt.

```
...
else if (str_getvalue.compare("temp_hours") == 0) {
  unsigned short hours, ncurves;
  unsigned char cooler, freezer, shttemp;
  ...

  // POST Daten prüfen
  if(environment.posts.size()) {
    if (!environment.requestVarGet("hours", hours))
      hours = 6; // Standard auf 6 Stunden
    if (environment.requestVarGet("cooler", cooler))
      ncurves++;
    ...
  }

  // Abfrage der Temperaturdaten von der Datenbank
  sqlite.getTempData(sensor_data, hours);

  std::vector<sensor_entry>::iterator it;
  sensor_entry entry;

  out << "Content-Type: text/html; charset=utf-8\r\n\r\n";
  out << "[";
  if (cooler) {
    out << "{\"label\": \"cooler\", \"data\": [";
    for (it = sensor_data.begin(); it != sensor_data.end(); ++it) {
      entry = *it;
      out << "[" << entry.timestamp << "," << entry.cooler << "],";
    }
    out << "]}";
  }
  out << "]";
  ...
}
```

Listing 5.5: Auswertung der ajax-Anfrage

6 Schlussbetrachtung

In diesem Kapitel wird nun abschließend eine Zusammenfassung und Analyse der realisierten Diplomarbeit durchgeführt und ein Ausblick auf zukünftige Entwicklungen gegeben.

6.1 Zusammenfassung

Die Kernaufgabe bei dieser Diplomarbeit war es, die Einbindung einer Kühl-Gefrierkombination in das Ethernet zu realisieren. Die Aufgabenstellung wurde hierzu in drei Teile zerlegt, von der jede für sich getrennt bearbeitet werden konnte – dazu mussten zuvor natürlich die Schnittstellen zwischen den Teilsystemen spezifiziert werden.
Das erste Teilsystem (die SmartFridge-Hardware) übernimmt die Aufgaben der Datenerfassung, Regelung und einer ersten Aufbereitung der Daten, die dann an das zweite Teilsystem (das SmartPanel) übergeben werden. Das SmartPanel ist hauptsächlich für die langfristige Datenspeicherung in einer Datenbank und die Netzwerkanbindung zuständig. Als Protokoll der Netzwerkschnittstelle wird HTTP verwendet, da somit auf jedem PC ein passenden Gegenstück – der Internet Browser – zur Verfügung steht. Die dazu notwendige Webseite, die über einen Webserver im SmartPanel bereitgestellt wird, stellt den dritten Teil dieser Diplomarbeit dar.

Der Ausgangspunkt für die SmartFridge-Hardware war die in [Gad02] dargestellte Hardware. Ziel bei der Entwicklung der neuen Hardware war es, einige der Probleme, die in der alten Version auftraten, zu beseitigen. So bestand beispielsweise bei der alten Hardware ein Problem mit der Datenspeicherung im Flash-Baustein – als Lösung dafür wurde in dieser Arbeit ein externer FRAM-Baustein (siehe Abschnitt 3.1.9) zur Speicherung der Daten und Konfigurationen angebunden. Die in [Gad02] ebenfalls angeführte Problematik von mehreren parallel ablaufenden und miteinander kommunizierenden Prozessen wurde hier durch den Einsatz eines Echtzeitbetriebssystems gelöst.
Damit die aktualisierte Hardware ohne größere Umbaumaßnahmen in die Kühl-Gefrierkombination integriert werden konnte, wurde die Belegung der verschiedenen Steckverbinder direkt übernommen. Einzige Ausnahme stellt die neu hinzugekommene serielle Schnittstelle für die Anbindung des SmartPanels dar.

Über das SmartPanel wird die eigentliche Aufgabenstellung der Netzwerkanbindung gelöst. Um hier den Entwicklungsaufwand für die Hardware so gering wie möglich zu halten, wurde ein

Prozessormodul mit Linux-Unterstützung verwendet. Dadurch reduzierte sich die Hardwareentwicklung auf ein Basisboard, über das die benötigten Schnittstellen nach außen realisiert werden. Dennoch erforderte es zwei Prototypen bis zur endgültigen Version.
Zusätzlich zu den erwähnten Hauptaufgaben des SmartPanels wurde zur Anzeige aktueller Werte auch ein Display in das Design integriert. Dieses wird über eine serielle Schnittstelle angebunden.
Der große Vorteil durch den Einsatz eines Linux-Betriebssystems war die Verfügbarkeit der notwendigen Applikationen (z.B. der Webserver) und Programmierbibliotheken.

Die über das SmartPanel verfügbare Webseite hat die Aufgabe, aktuelle Daten anzuzeigen und die Konfiguration der Kühl-Gefrierkombination zu ermöglichen. Dazu kamen moderne Technologien (für die gerne das Schlagwort „Web 2.0“ verwendet wird) zum Einsatz, die eine dynamische Darstellung der Informationen ermöglichen. Aufgrund der Komplexität der Thematik „Webdesign“ und der für die Realisierung veranschlagten Zeit, konnte hierbei keine perfekte, optisch herausragende Webseite realisiert werden. Ziel war es hier vielmehr das Zusammenspiel der verschiedenen Technologien zu zeigen, was auch gelungen ist.

Zusammenfassend kann in dieser Arbeit gezeigt werden, dass sich der Hardwareaufwand für die Realisierung eines intelligenten Kühlschranks in Grenzen hält und mit den heute vorhandenen Technologien günstig realisierbar ist. Der größte Kostenfaktor bei der Entwicklung eines solchen Systems liegt demnach im Bereich der Softwareentwicklung.

Abbildung 6.1 zeigt abschließend Bilder der realisierten Hardware (links die SmartFridge-Platine und rechts die SmartPanel-Platine).

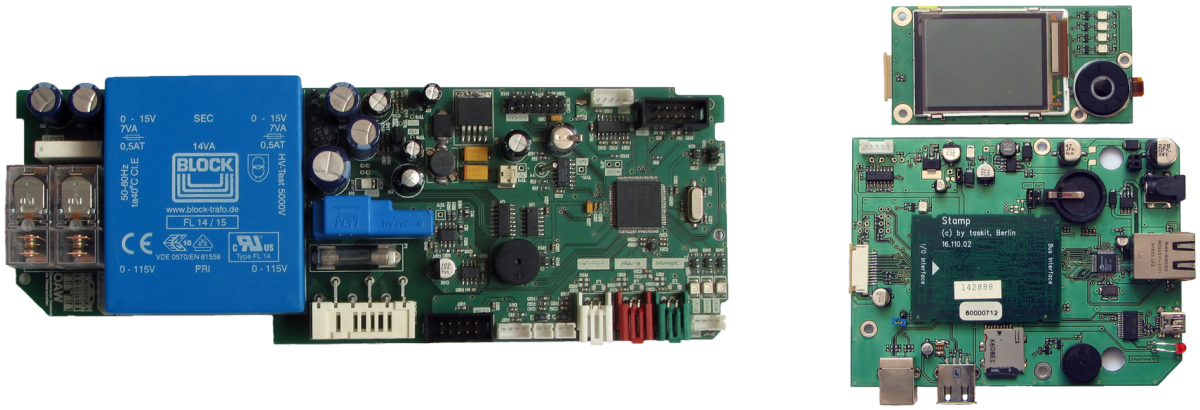

Abbildung 6.1: SmartFridge und SmartPanel Hardware

6.2 Probleme und Lösungsvorschläge

Dieser Abschnitt zeigt nun einige der aufgetretenen Probleme, aber auch Designschwächen auf und gibt Vorschläge für Verbesserungen.

Grundsätzlich kann festgehalten werden, dass die Trennung der Aufgabenstellung in zwei voneinander unabhängige Geräte für diese Arbeit eine gute Entscheidung war. Die Gründe dafür waren, dass auf diese Weise einerseits am Kühlschrank keine großen mechanischen Änderungen vorzunehmen waren und, dass andererseits eine höhere Ausfallsicherheit gewährleistet werden

kann. Das Gesamtkonzept könnte aber dahingehend verbessert werden, dass nur eine Platine verwendet wird, auf der sich beide Prozessoren befinden. Das Display könnte in diesem Fall die vorhandene Anzeige der Kühl-Gefrierkombination ersetzen.
Ein Problem, dass dadurch automatisch gelöst wäre, ist die Stromversorgung der SmartPanel-Hardware, die jetzt entweder über ein externes Netzteil oder über *Power over Ethernet* erfolgt. Prinzipiell könnte die SmartPanel-Hardware auch jetzt schon von der SmartFridge-Hardware mitversorgt werden, allerdings ist der Spannungsregler für die 5 V-Linie der SmartFridge-Hardware nicht für den zusätzlichen Leistungsverbrauch von ungefähr 720 mW ausgelegt. Da die Versorgung über das Verbindungskabel der seriellen Schnittstelle erfolgen würde, wäre zudem die Betriebsspannung der SmartFridge-Hardware von außen zugänglich. Obwohl die entsprechende Betriebsspannungs-Leitung der seriellen Verbindung vor unbeabsichtigtem Kurzschluss geschützt ist, könnte sie missbräuchlich verwendet werden, weshalb hier von der Verwendung abgesehen wird.

Weitere Schwächen der Konzeption wurden beim Gebiet der Strom- und Temperaturmessung sichtbar. Die hier verwendeten NTC-Sensoren sind alle dauerhaft mit der Betriebsspannung verbunden, wodurch es zu einer minimalen Erwärmung der Sensoren kommt. Obwohl dies auf den Betrieb und die Regelung der Kühl-Gefrierkombination keine Auswirkungen hat, könnte über eine gesteuerte An- und Abschaltung eine verbesserte *Messgenauigkeit* erreicht werden.
Neben der Problematik der Eigenerwärmung könnte die Messung mittels den NTC-Sensoren auch insofern verbessert werden, dass nur der relevante Temperaturbereich der Kennlinie auf den Spannungsbereich des ADCs von 0 bis 5 V umgesetzt wird. Dies könnte über Operationsverstärker geschehen und würde zu einer höheren *Messauflösung* führen.
Wie schon angesprochen, könnte auch die Schaltung der Strommessung weiter verbessert werden. Die aktuelle Schaltung ist für einen maximalen Eingangsstrom von 15 A ausgelegt. Die Auflösung der Messung beträgt beim verwendeten 10-Bit-ADC also ungefähr 15 mA. Hier wären mehrere Messbereiche sinnvoll, um bei kleinen Strömen eine höhere *Auflösung* zu erreichen.

Eine Funktionalität der SmartFridge-Hardware, die zwar geplant, aber aus Zeitgründen nicht integriert wurde, ist ein *Fallback-Modus*. Dieser stellt einen minimalen, reduzierten Betriebsmodus dar, der nur die Regelung auf eine definierte Temperatur implementiert – ohne Einsatz eines Echtzeitbetriebssystems und nicht unbedingt benötigter Peripherie. Gestartet werden kann dieser Modus etwa dann, wenn im Standardmodus nicht behebbare Fehler erkannt werden – alternativ auch über einen Jumper auf der SmartFridge-Platine.

Auch beim SmartPanel wurden einige Punkte erkannt, die besser gelöst werden könnten. Von der Seite der Hardware stellte sich vor allem das gewählte Gehäuse nicht als die beste Wahl heraus. Abgesehen davon, dass es für die SmartPanel-Platine von der Größe her eigentlich überdimensioniert ist, war auch die mechanische Bearbeitung der teilweise bis zu 5 mm dicken Seitenteile keine leichte Aufgabe. Vor allem die Montage des Scrollrades war problematisch.
Neben der Verwendung eines anderen Gehäuses, wäre hier der Einsatz von berührungsempfindlichen Sensoren eine Lösungsmöglichkeit. Mit dieser Sensortechnik wurde zu Beginn der Diplomarbeit experimentiert, wobei aufgrund der Empfindlichkeit und der notwendigen, experimentell durchzuführenden Anpassung der Bauteile die Entscheidung gegen den Einsatz dieser Technik fiel. Wie schon zuvor angeführt, wäre die beste Lösung aber die Integration der Hardware auf die SmartFridge-Platine. Vorstellbar wäre hierbei auch der komplette Austausch der Bedieneinheit an der Kühl-Gefrierkombination um die Funktionen von *SmartFridge* und *SmartPanel* auf ein gemeinsames Display zusammenzufassen.

Softwareseitig gesehen erfordert vor allem das Menüsystem eine Überarbeitung. Da zu Beginn

der Arbeiten noch nicht genau bekannt war welche Anforderungen gestellt werden, entstand die derzeitige Struktur vor allem durch schrittweises Verbessern der Implementierung. Hier wäre eine komplette Neuprogrammierung mit Zielvorgabe bzw. Pflichtenheft vorzunehmen. Alternativ möglich, und vermutlich auch die bessere Wahl, wäre aber der Einsatz eines leistungsfähigeren Prozessormoduls, auf dem eines der vielen frei verfügbaren Benutzinterface-Frameworks (etwa QT4) genutzt werden kann.

Natürlich bietet auch die Webseite genug Spielraum für Verbesserung. Zum einen könnten wesentlich mehr Daten dargestellt bzw. konfiguriert werden. Zum anderen wäre es auch vorstellbar, dass das SmartPanel die Daten etwa direkt in das Excel-XML Format aufbereitet und zur weiteren Analyse als Download zur Verfügung stellt.

6.3 Ausblick

Die Thematik der intelligenten Stromnetze wird in absehbarer Zeit einen immer größeren Stellenwert einnehmen. Ein Grund hierfür ist der steigende Stromverbrauch, der die Einbeziehung der Verbraucher im Haushalt als dynamische Komponente des Versorgungsnetzes erfordert [GB09]. Ziel hierbei ist die Herstellung eines Gleichgewichts zwischen der Erzeugung und dem Verbrauch der Energie und die Reduzierung von Spitzenlasten.

Eine Möglichkeit um dies zu erreichen, ist die Dezentralisierung der Energieerzeugung [GB09] und eine Einbindung von Haushalten (beispielsweise der Fotovoltaikanlage) in das Gesamtsystem. In diesem Kontext müssen auch die verschiedenen Formen der regenerativen Energien berücksichtigt werden, die teilweise eine besondere Herausforderung bei der Verteilung der Energie darstellen. So kann es beispielsweise bei Windkraftwerken zu nicht vorhersagbaren Überschüssen bei der Erzeugung kommen, die aufgrund mangelnder Nachfrage auf der Verbraucherseite nicht genutzt werden. Durch die bereits angesprochene dezentrale Verwaltung des Stromnetzes könnten in solch einem Fall geeignete Verbraucher dynamisch zugeschaltet werden. Als Beispiel seien hier Geräte genannt, welche die Energie nicht instantan benötigen, sondern in anderer Form zwischenspeichern - etwa Kühlschränke oder Heizungen.

Elementar für eine dezentrale, koordinierte Verteilung der Energie und in weitere Folge auch für die dezentrale Verwaltung ist erstens die Vernetzung der Verbraucher mit den Erzeugern und zweitens die Einbettung einer gewissen „Eigenintelligenz“ in die einzelnen Komponenten [Pal01]. In diesem Zusammenhang wird auch manchmal der Begriff *Internet der Energie* verwendet, was unter anderem auch die Rollenänderung des Stromanbieters hin zum Informationsdienstleister aufzeigen soll [Int08]. Softwareseitig könnten solche Szenarien durch Agentensysteme realisiert werden, die sich selbst organisieren und die für das gesamte Netz die beste Lösung suchen [Pal01]. Da die Kosten für leistungsfähige Prozessoren stetig sinken, wird es also vielmehr eine Frage der Definition und Normierung der Schnittstellen sein, ab wann solche Systeme für den Massenmarkt tauglich sind.

Ein Bericht der *European Smartgrids Technology Platform* – kurz ESTP – stellt bei den Zielen eines intelligenten Stromnetzes (Smart Grid) vor allem die Punkte Flexibiliät, einheitlicher Zugang, Zuverlässigkeit und vor allem auch die Wirtschaftlichkeit in den Vordergrund. Abseits der technischen Realisierbarkeit wird es also auch darauf ankommen, ob das intelligente Stromnetz nicht nur dem Energielieferanten, sondern auch dem Kunden Vorteile und somit auch Kostenersparnisse bringt.

7 Anhang

7.1 Schaltpläne SmartFridge

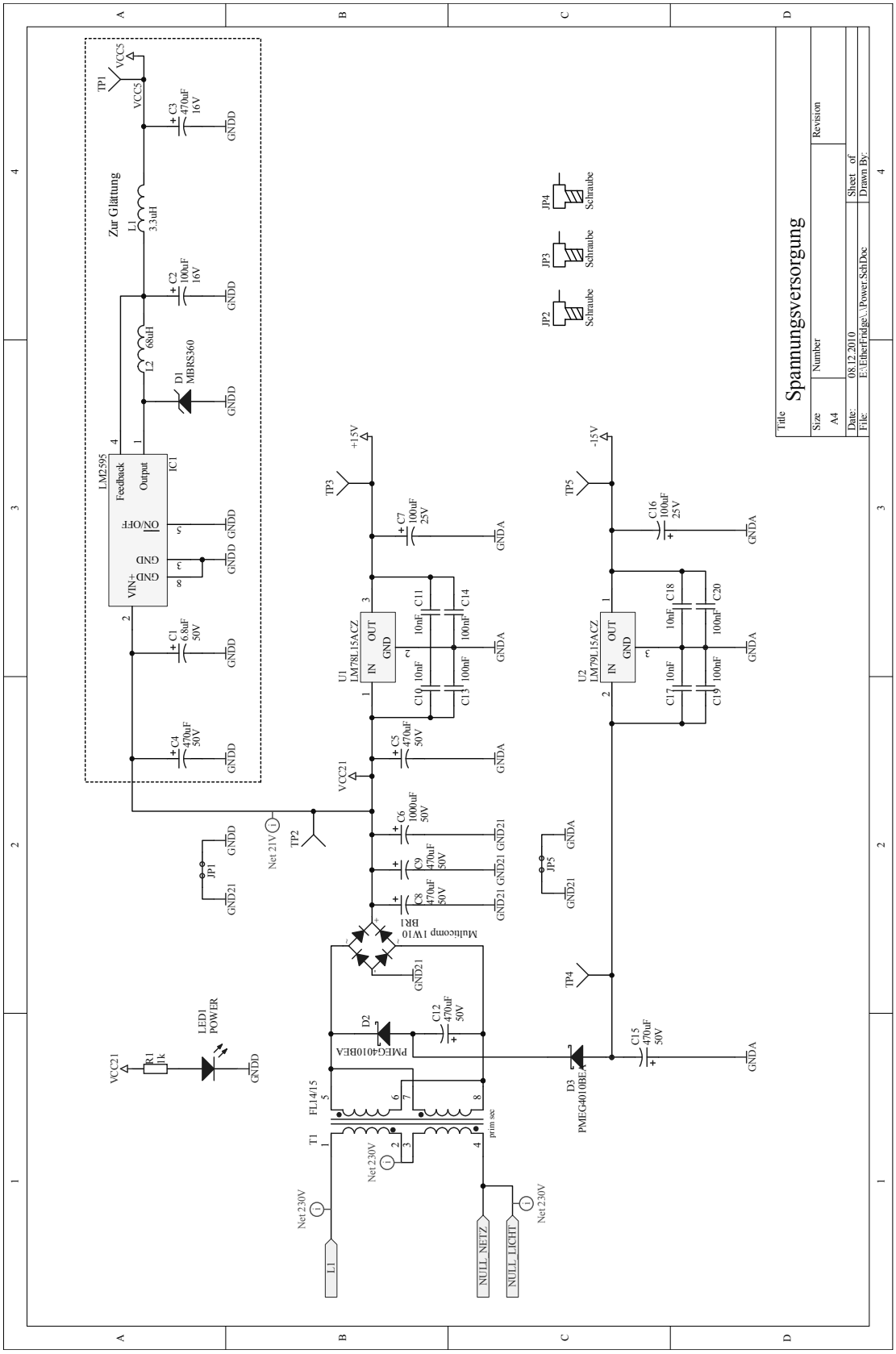

Abbildung 7.1: Schaltplan Stromversorgung SmartFridge

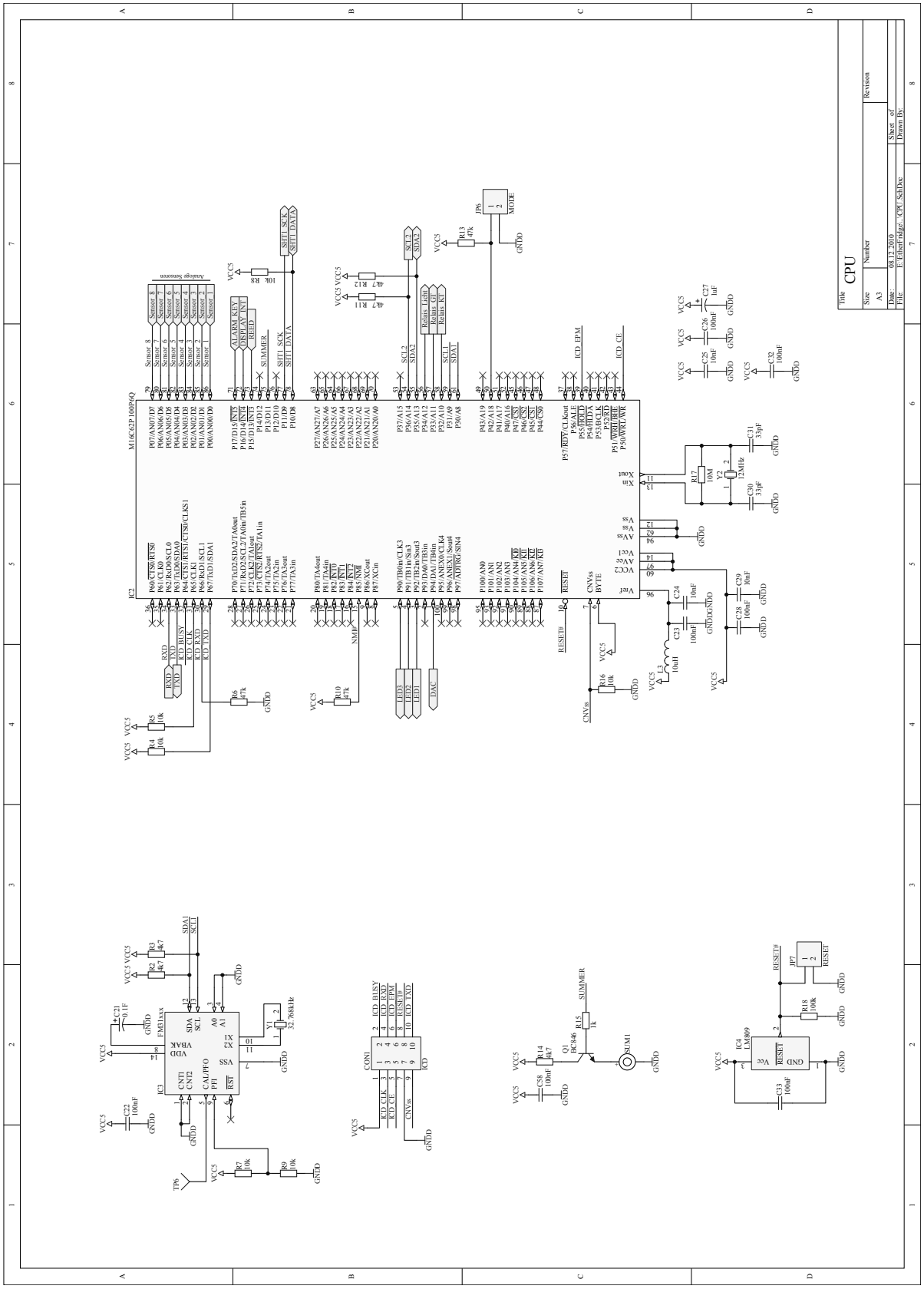

Abbildung 7.2: Schaltplan CPU SmartFridge

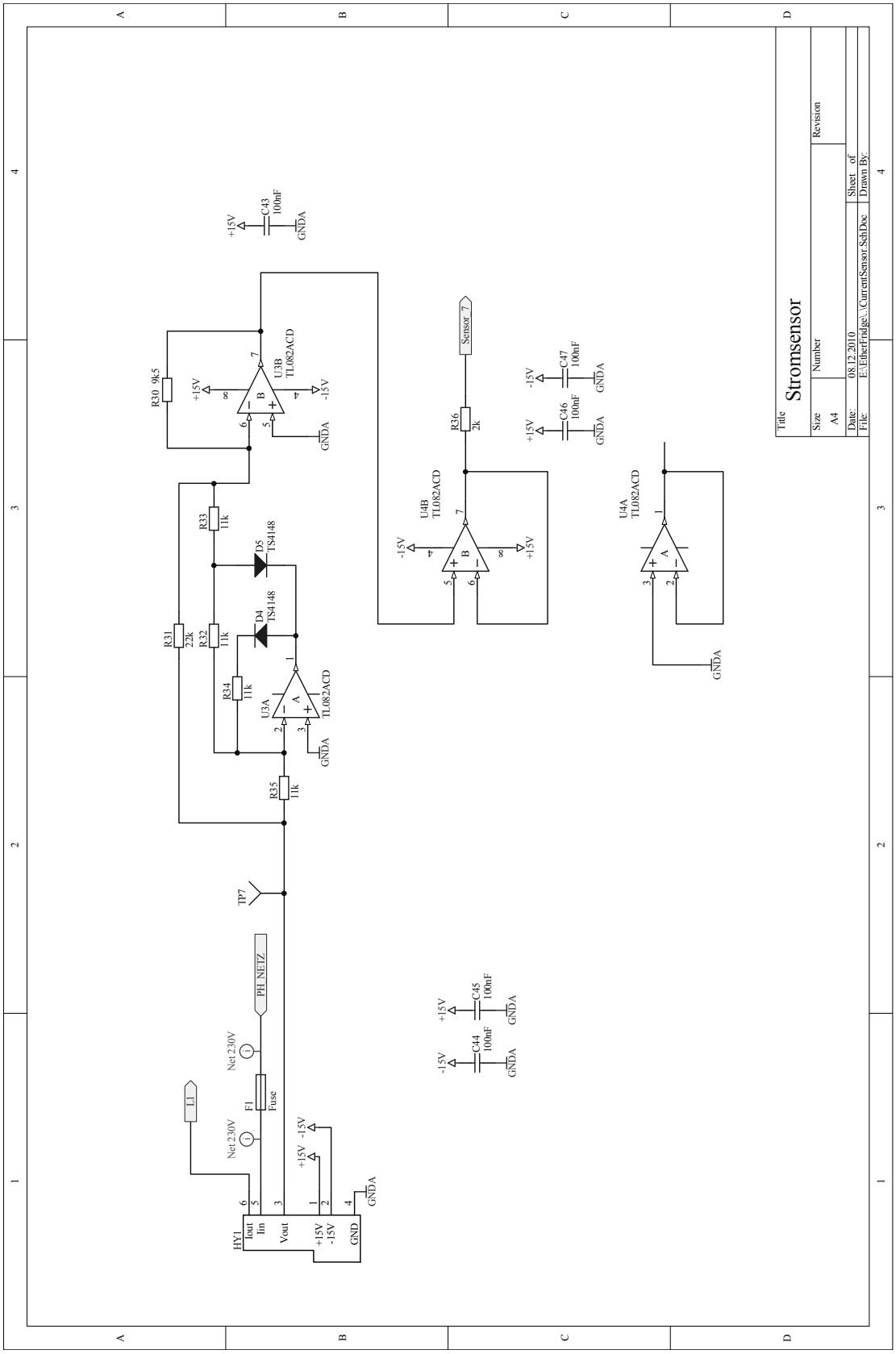

Abbildung 7.3: Schaltplan Stromsensor SmartFridge

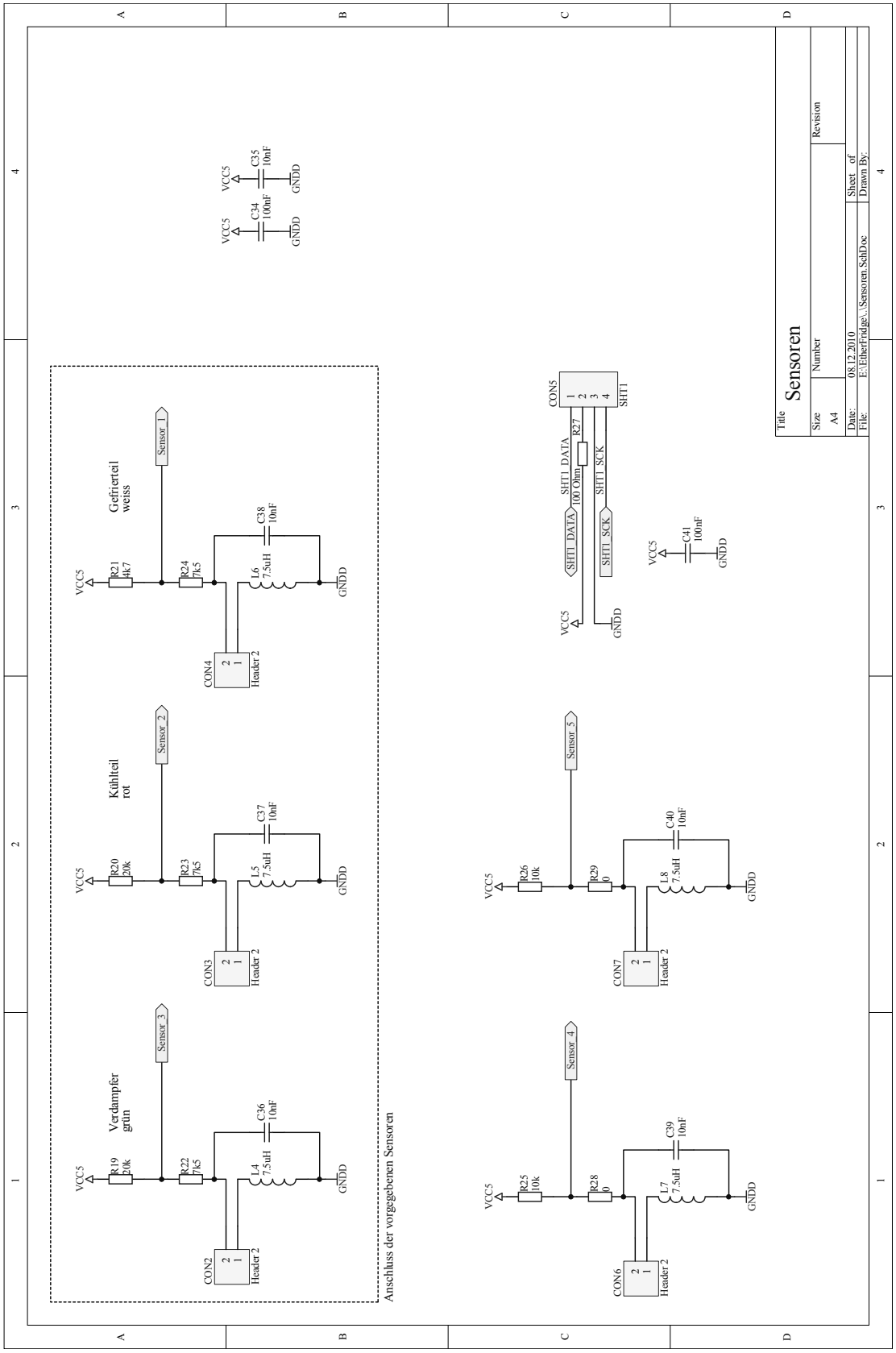

Abbildung 7.4: Schaltplan Temperatursensoren SmartFridge

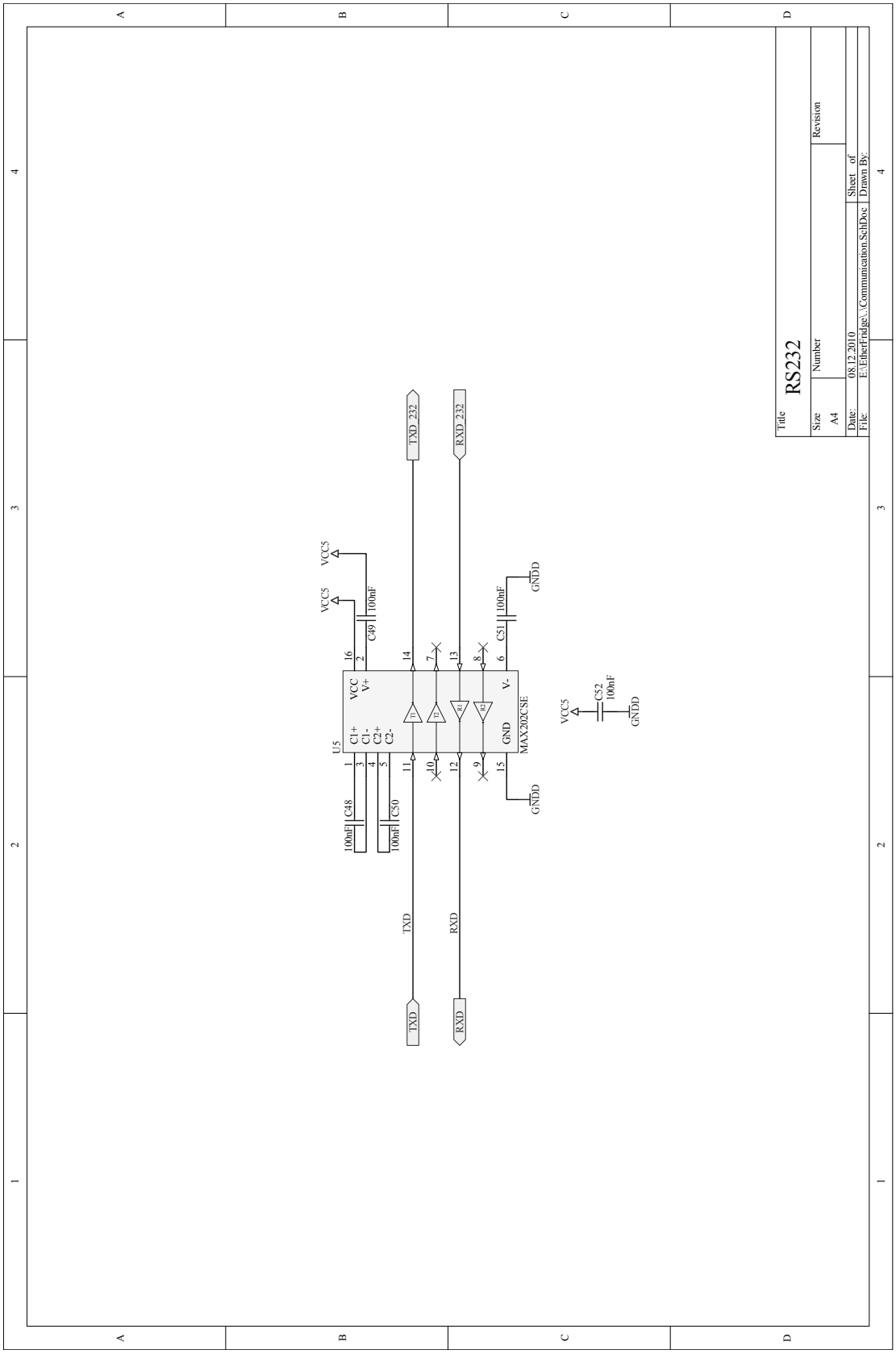

Abbildung 7.5: Schaltplan serielle RS232 Schnittstelle SmartFridge

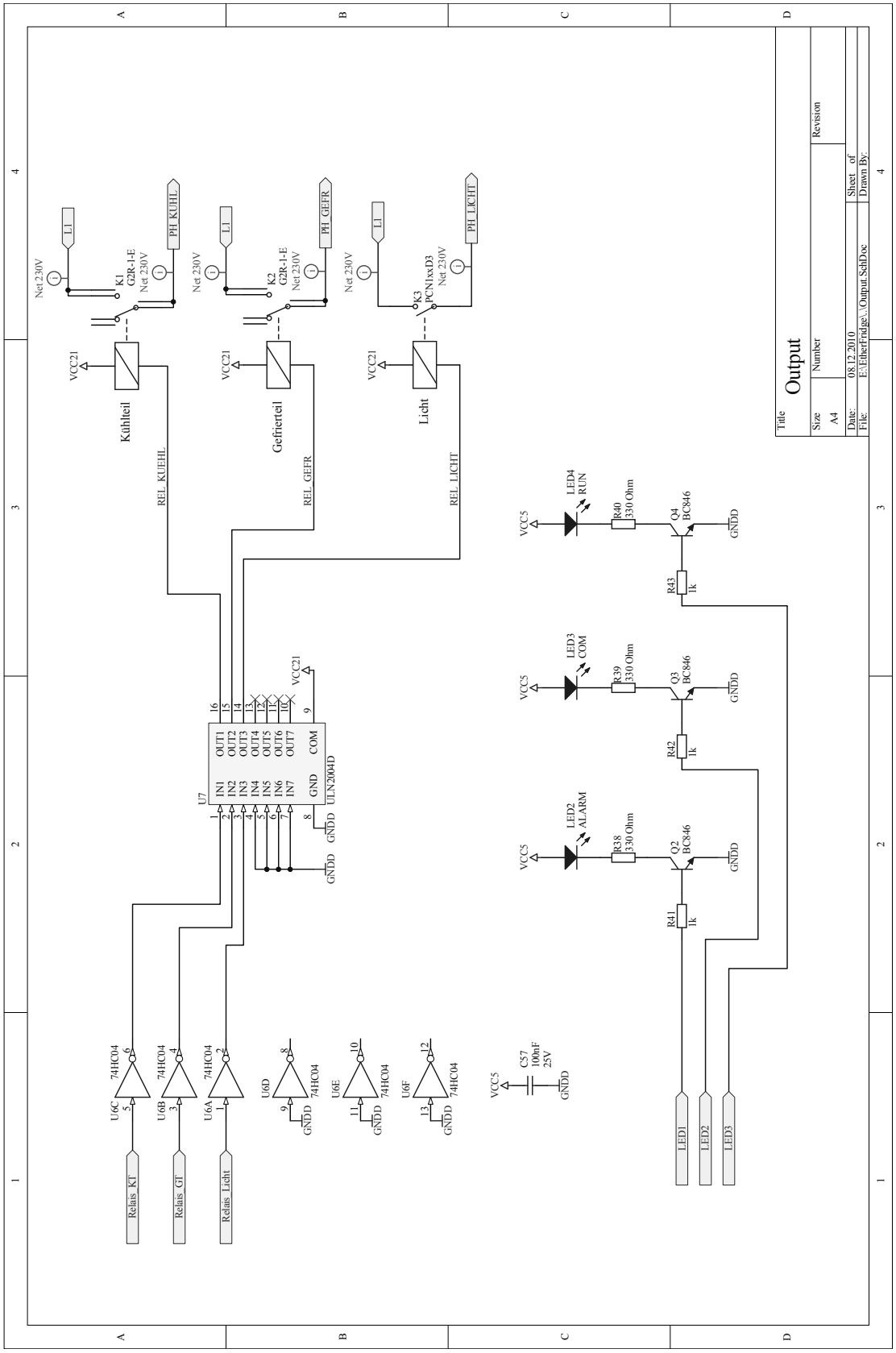

Abbildung 7.6: Schaltung Relais und LEDs SmartFridge

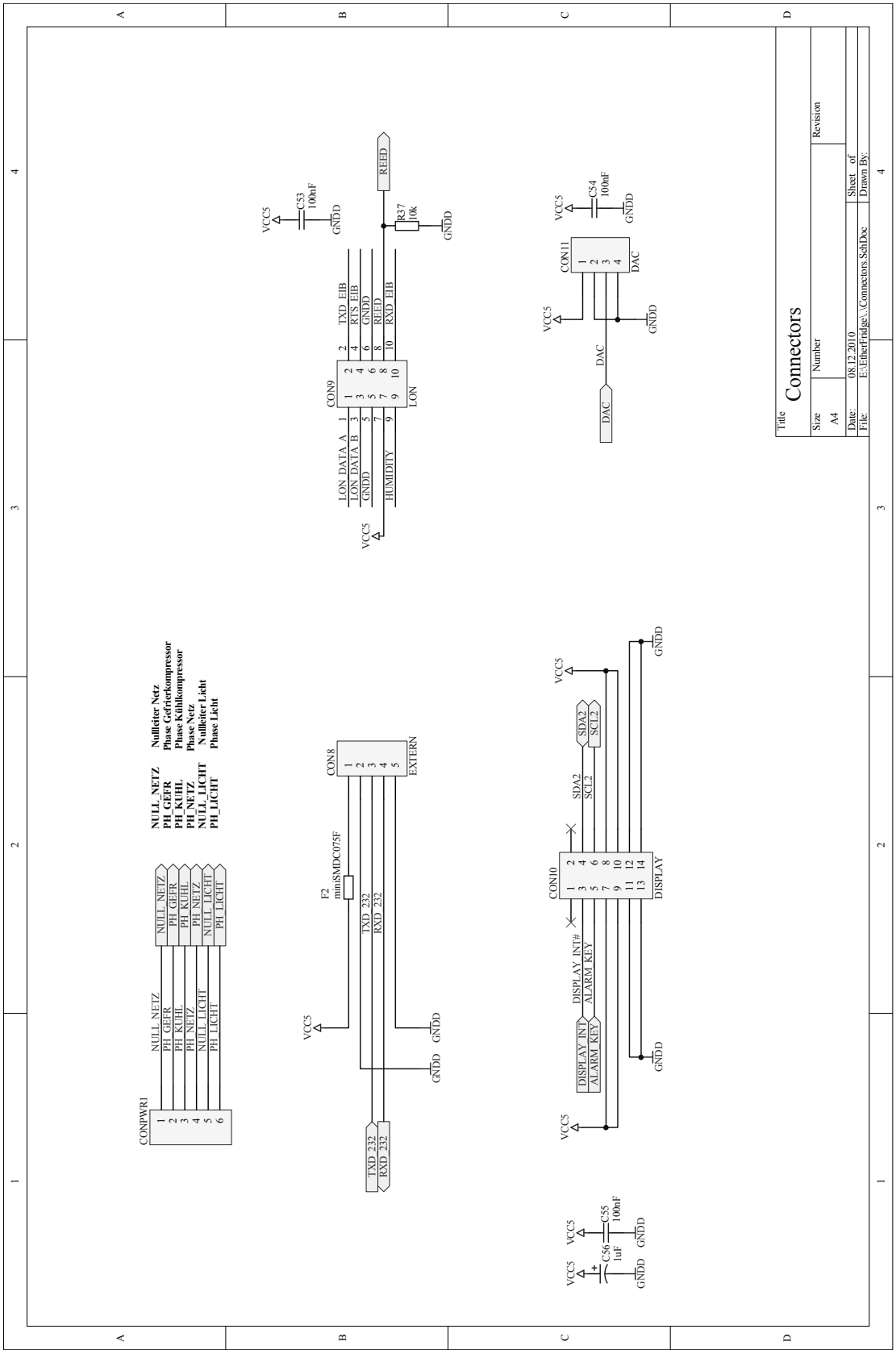

Abbildung 7.7: Schaltung Steckverbinder SmartFridge

7.2 Schaltpläne SmartPanel

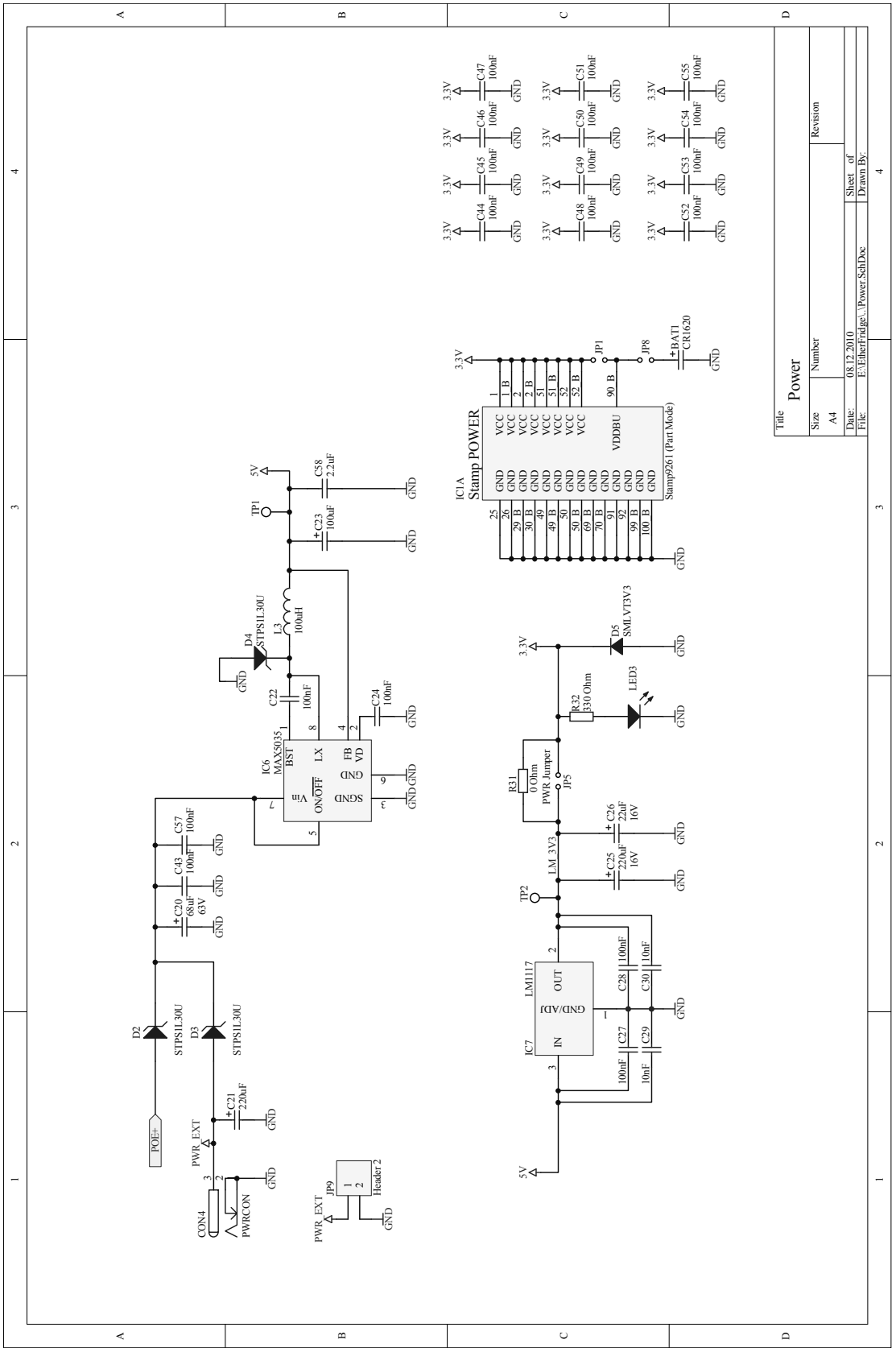

Abbildung 7.8: Schaltplan Stromversorgung SmartPanel

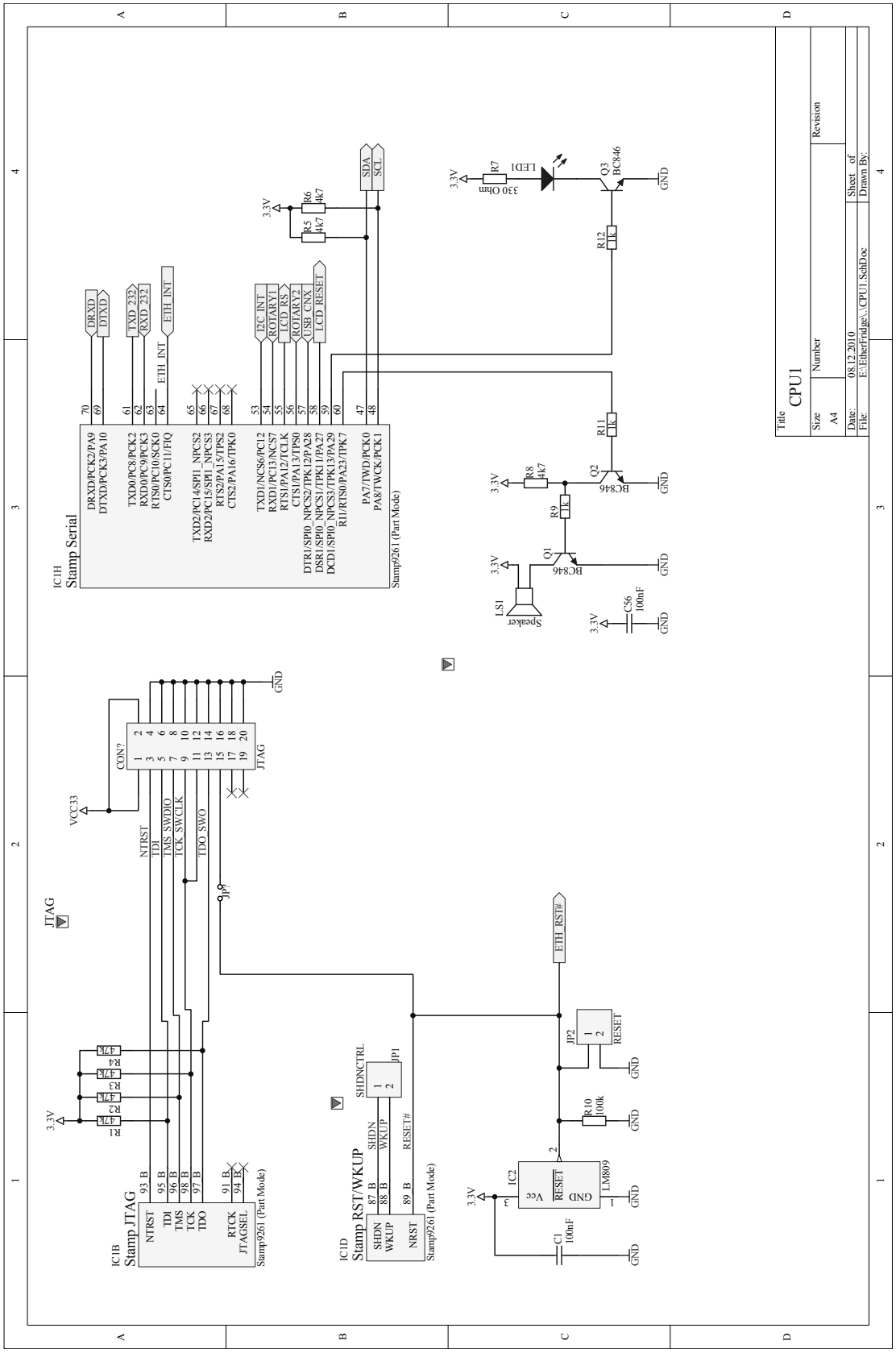

Abbildung 7.9: Schaltplan CPU SmartPanel (erster Teil)

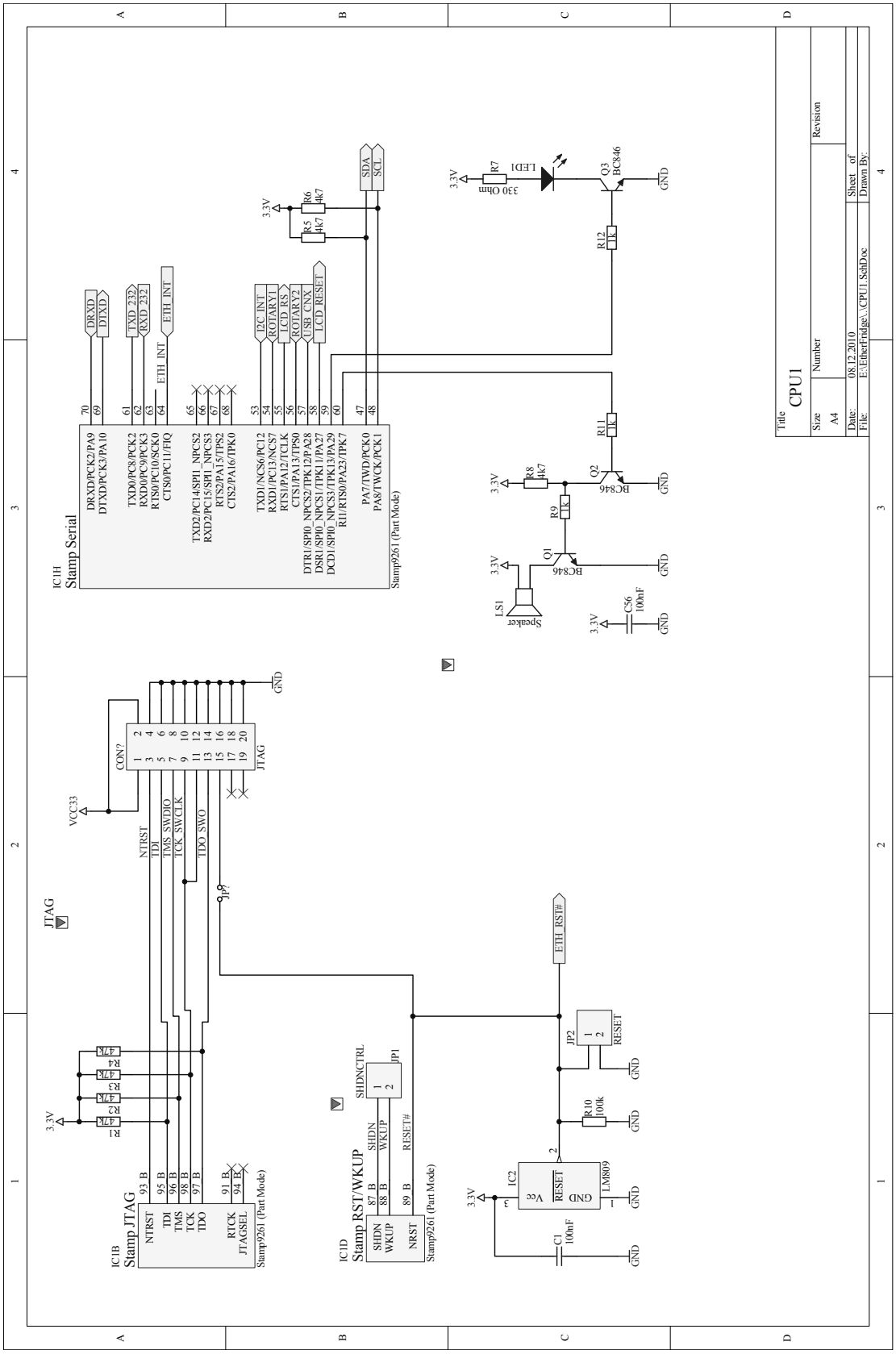

Abbildung 7.10: Schaltplan CPU SmartPanel(zweiter Teil)

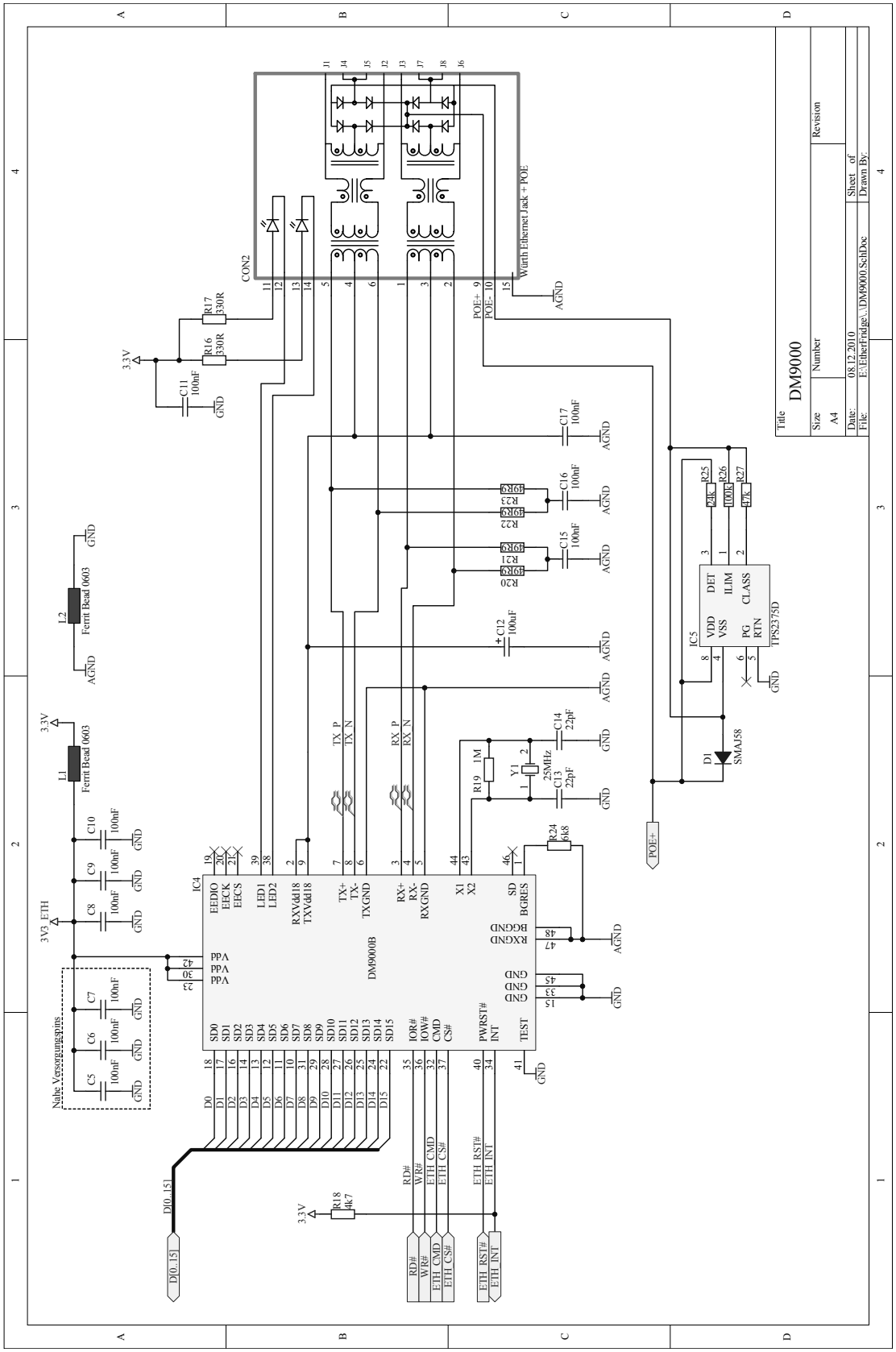

Abbildung 7.11: Schaltplan Netzwerkschnittstelle SmartPanel

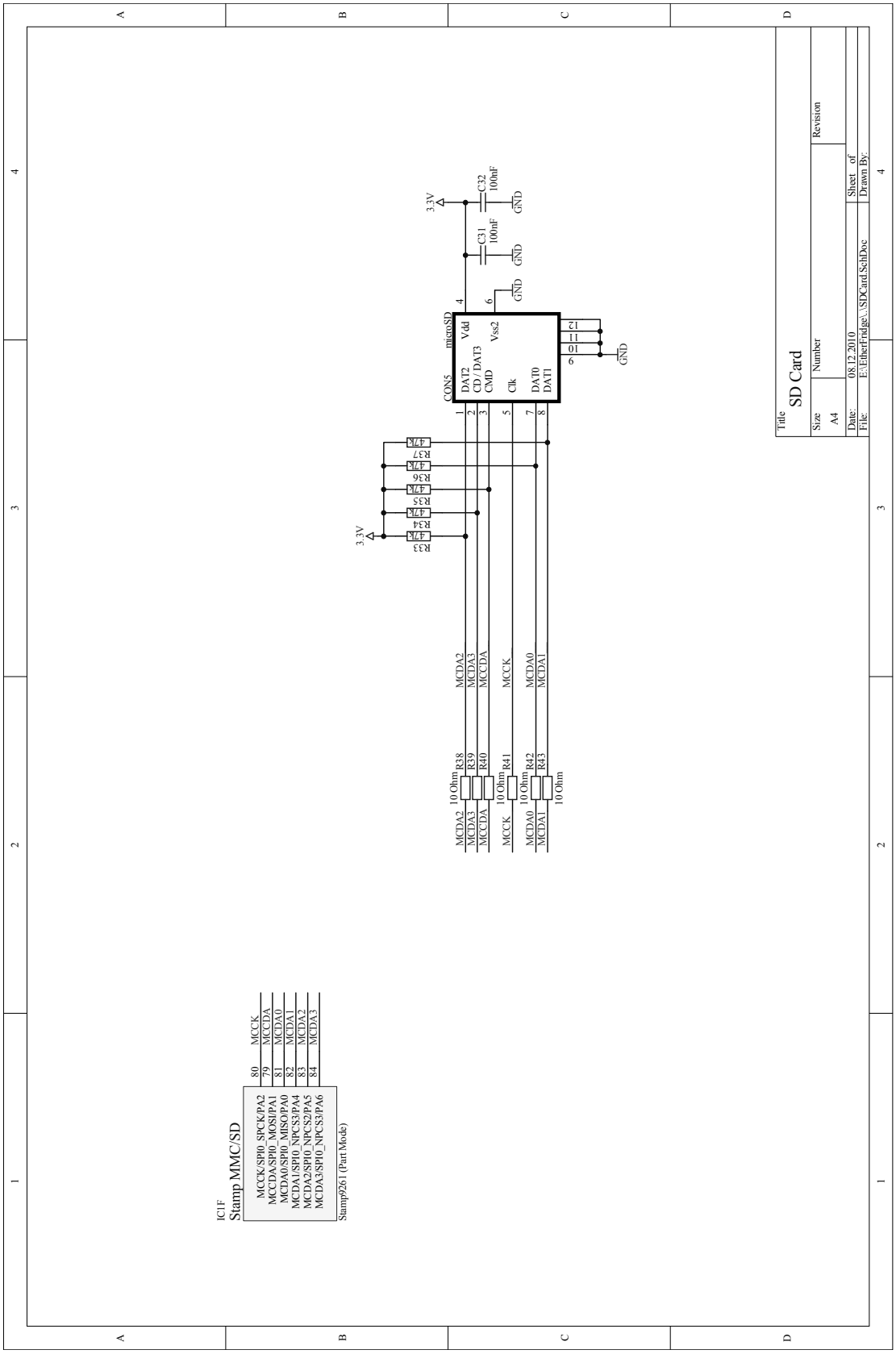

Abbildung 7.12: Schaltplan SD-Karte SmartPanel

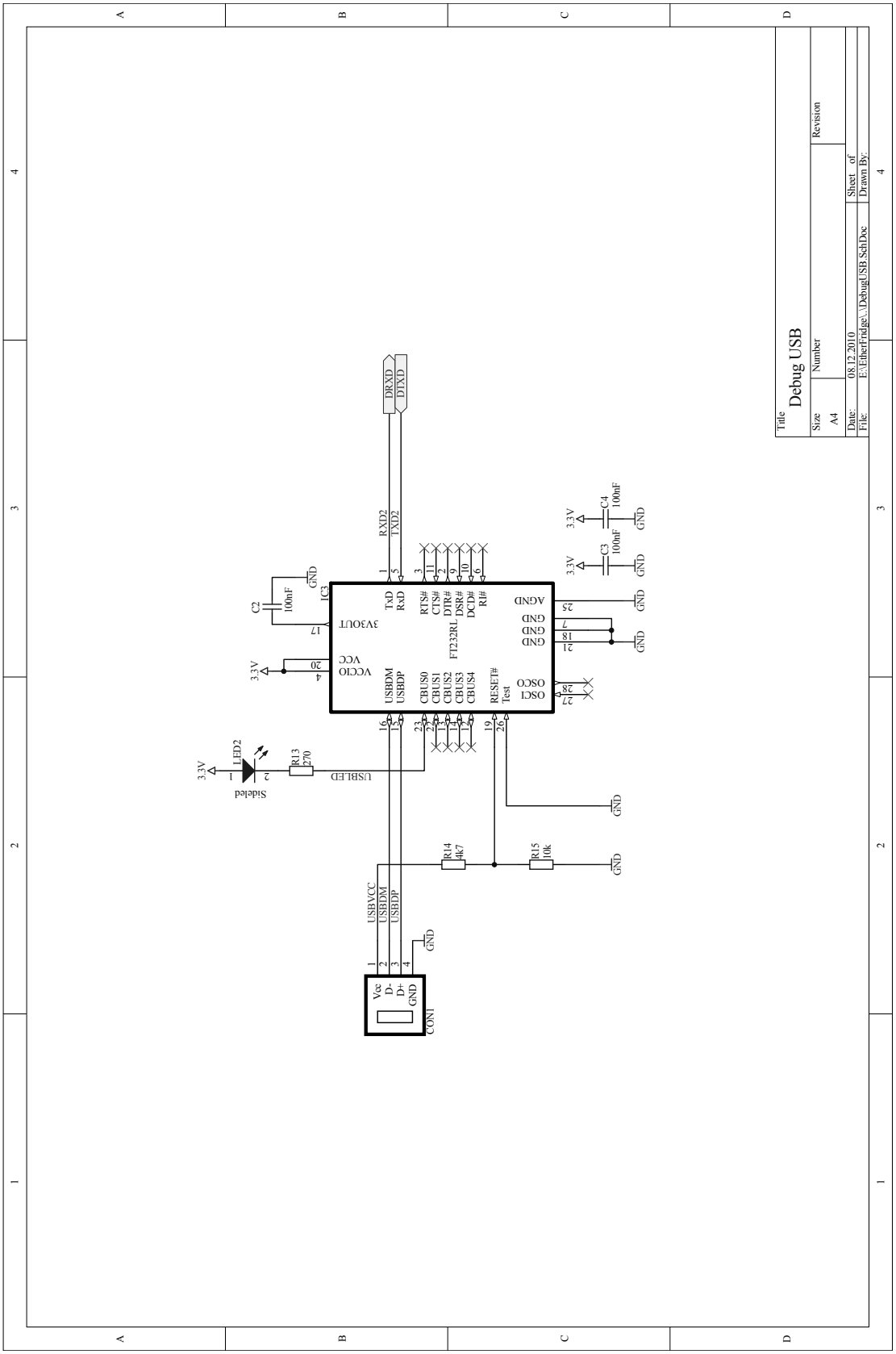

Abbildung 7.13: Schaltplan Debug Schnittstelle (USB zu RS232 Wandler) SmartPanel

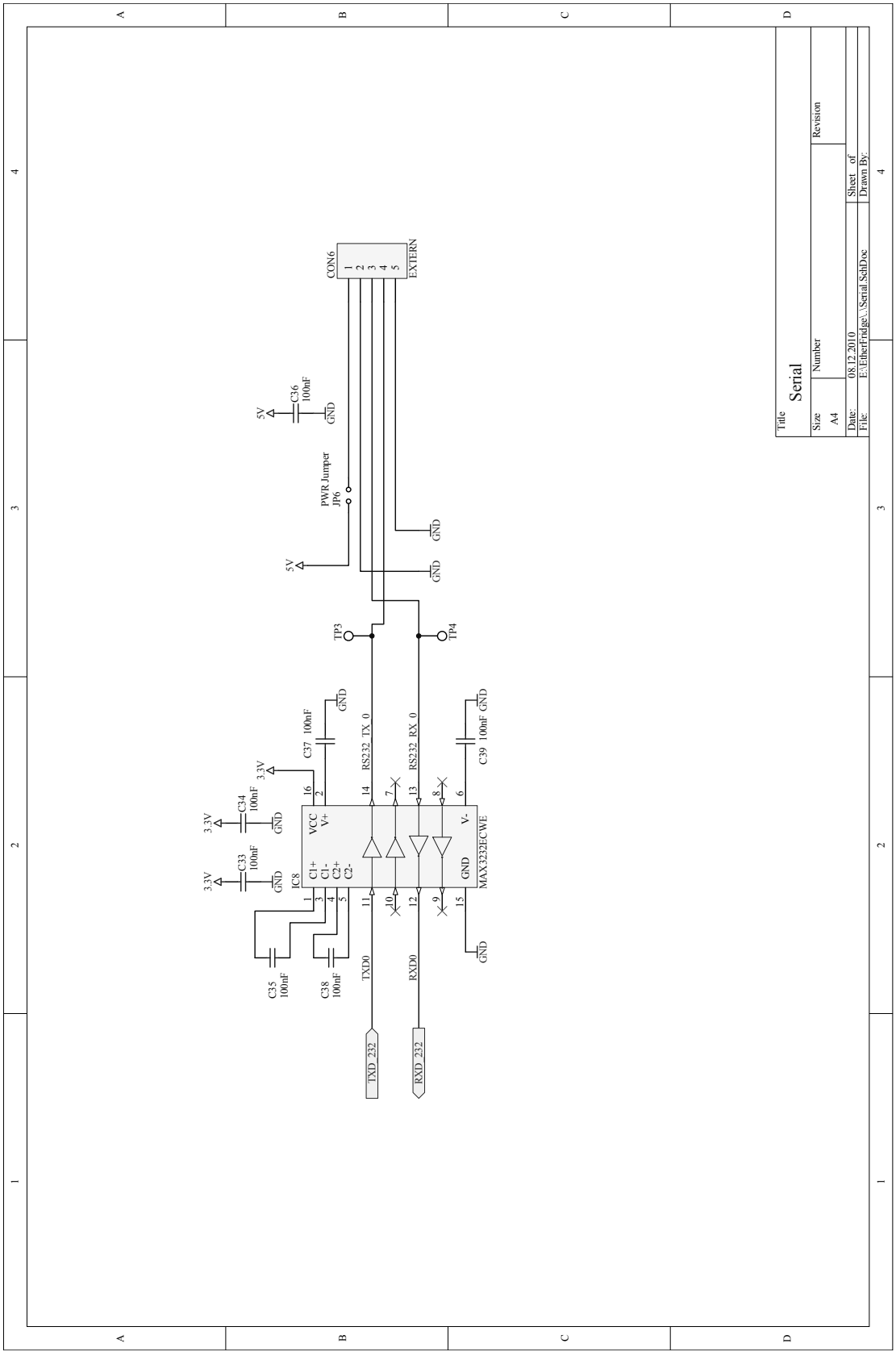

Abbildung 7.14: Schaltplan serielle Schnittstelle (RS232) SmartPanel

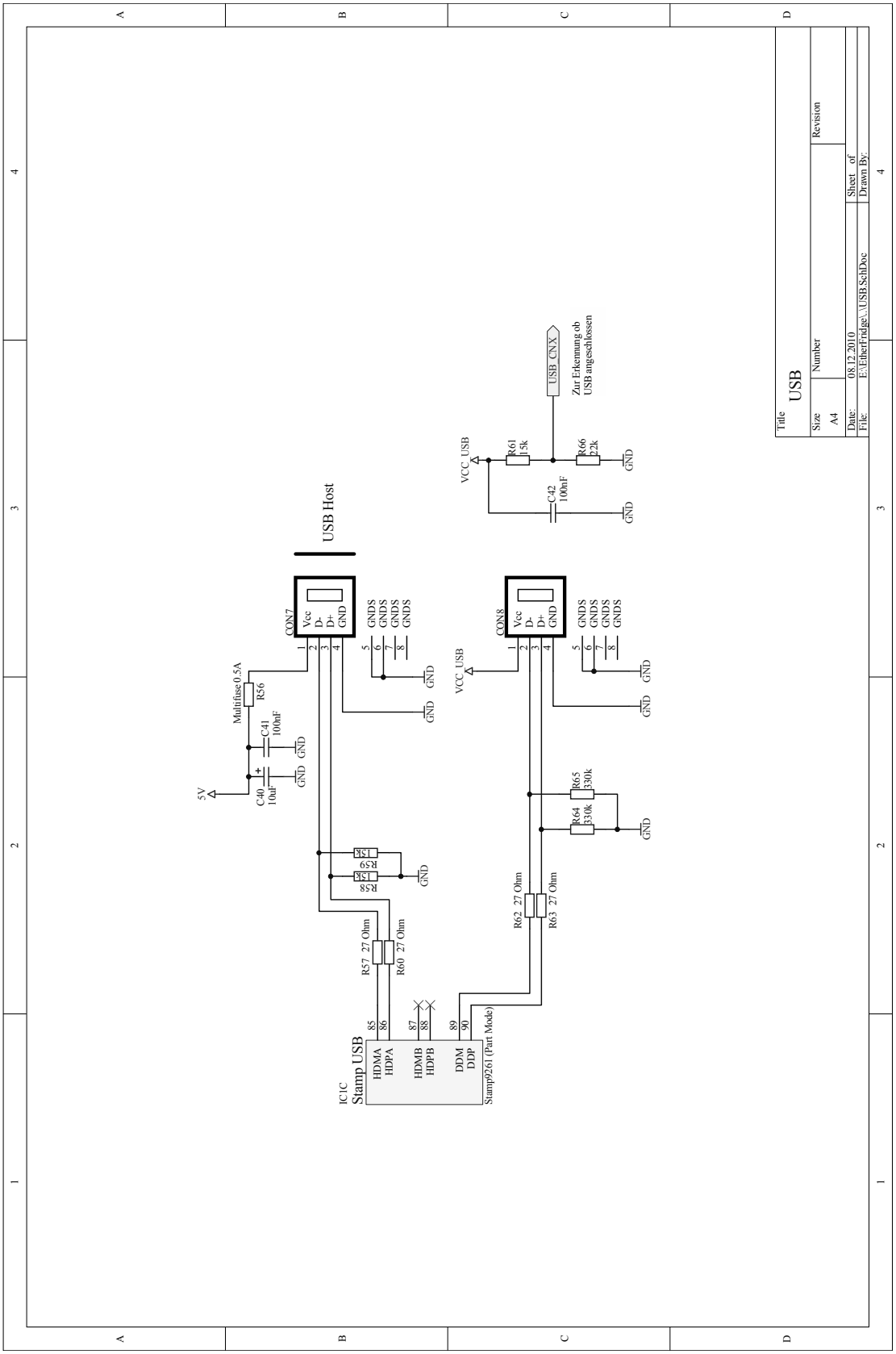

Abbildung 7.15: Schaltplan USB-Schnittstellen SmartPanel

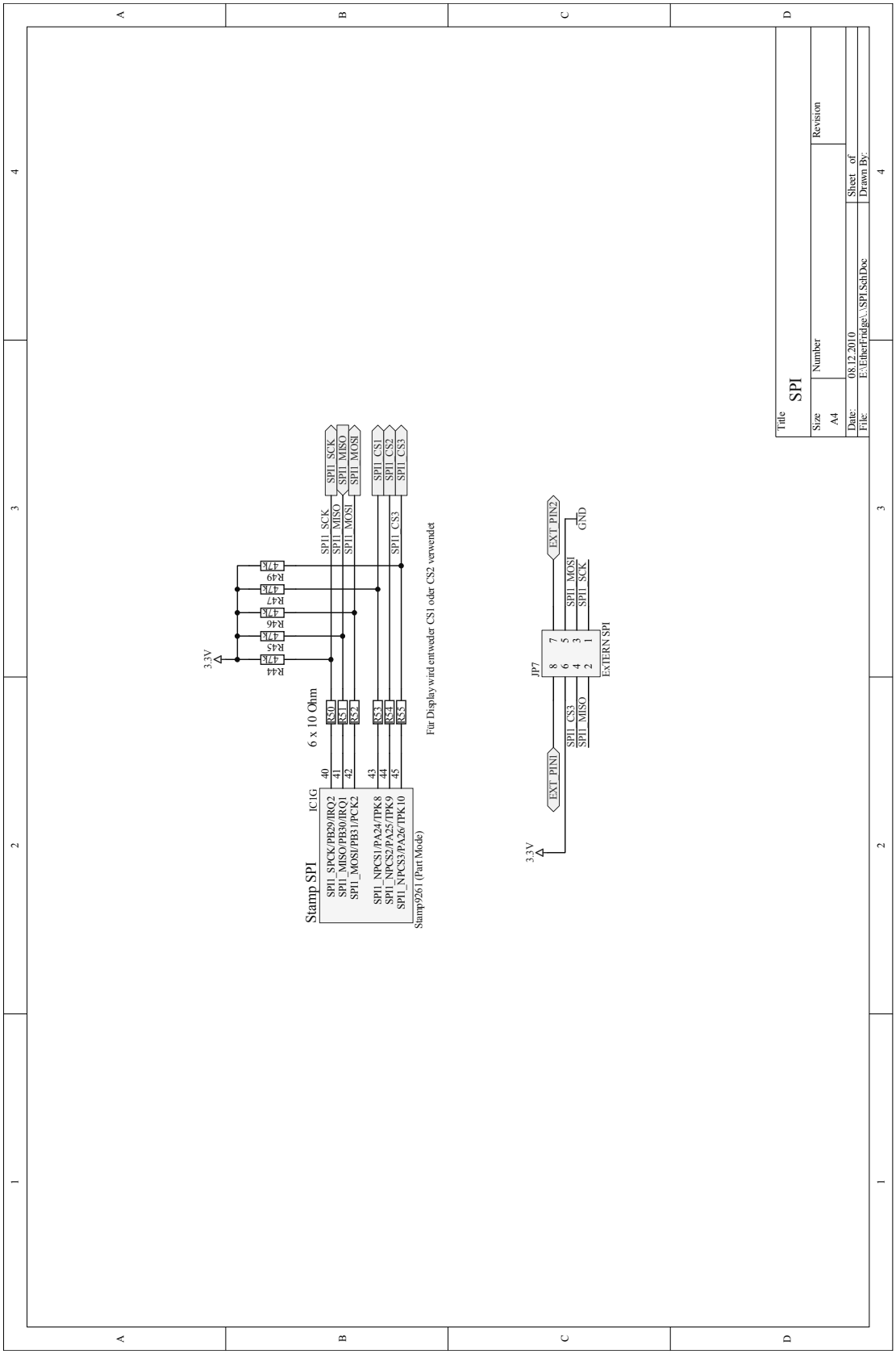

Abbildung 7.16: Schaltplan SPI Schnittstellen SmartPanel

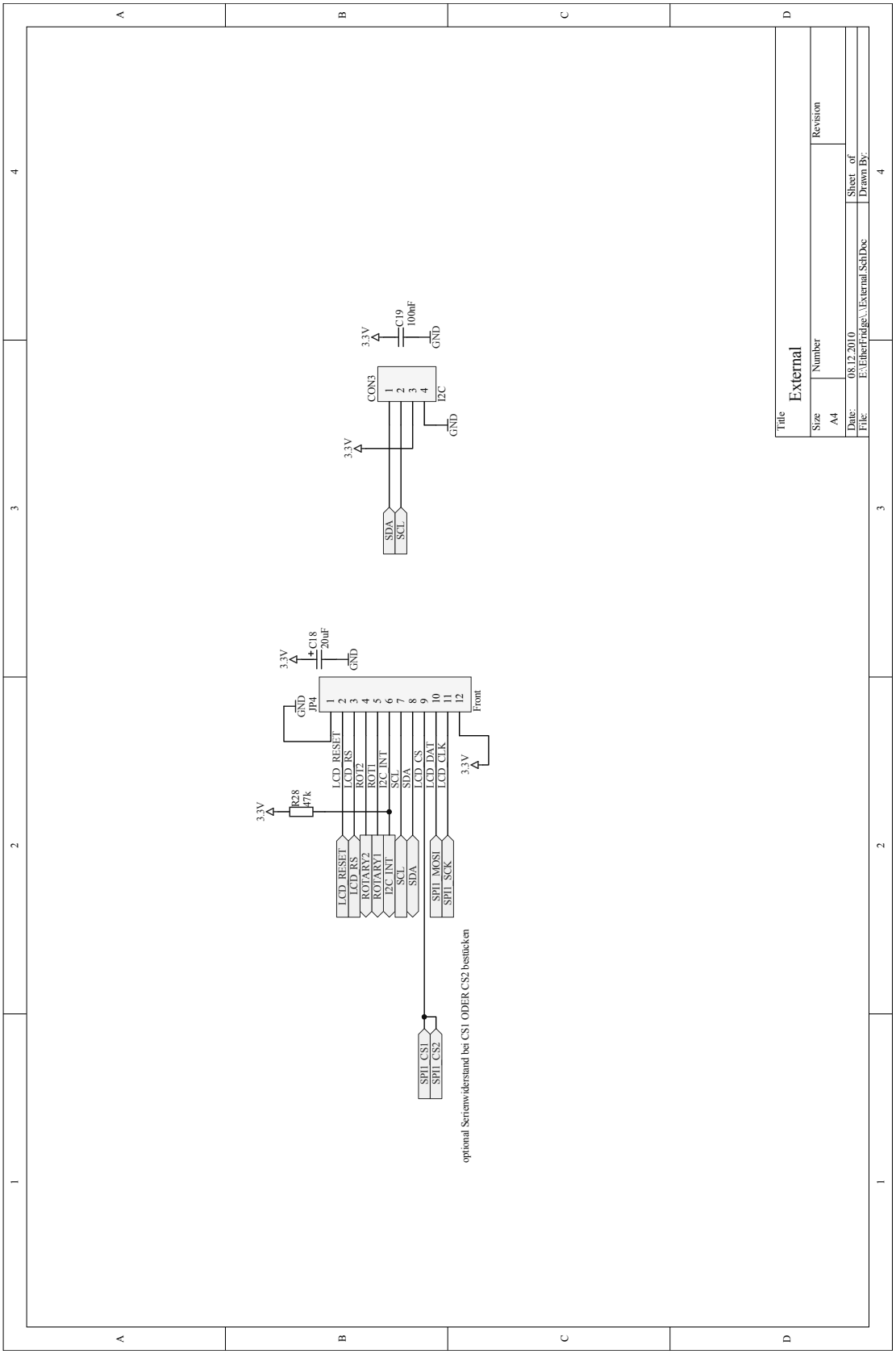

Abbildung 7.17: Schaltplan Steckverbinder Display SmartPanel

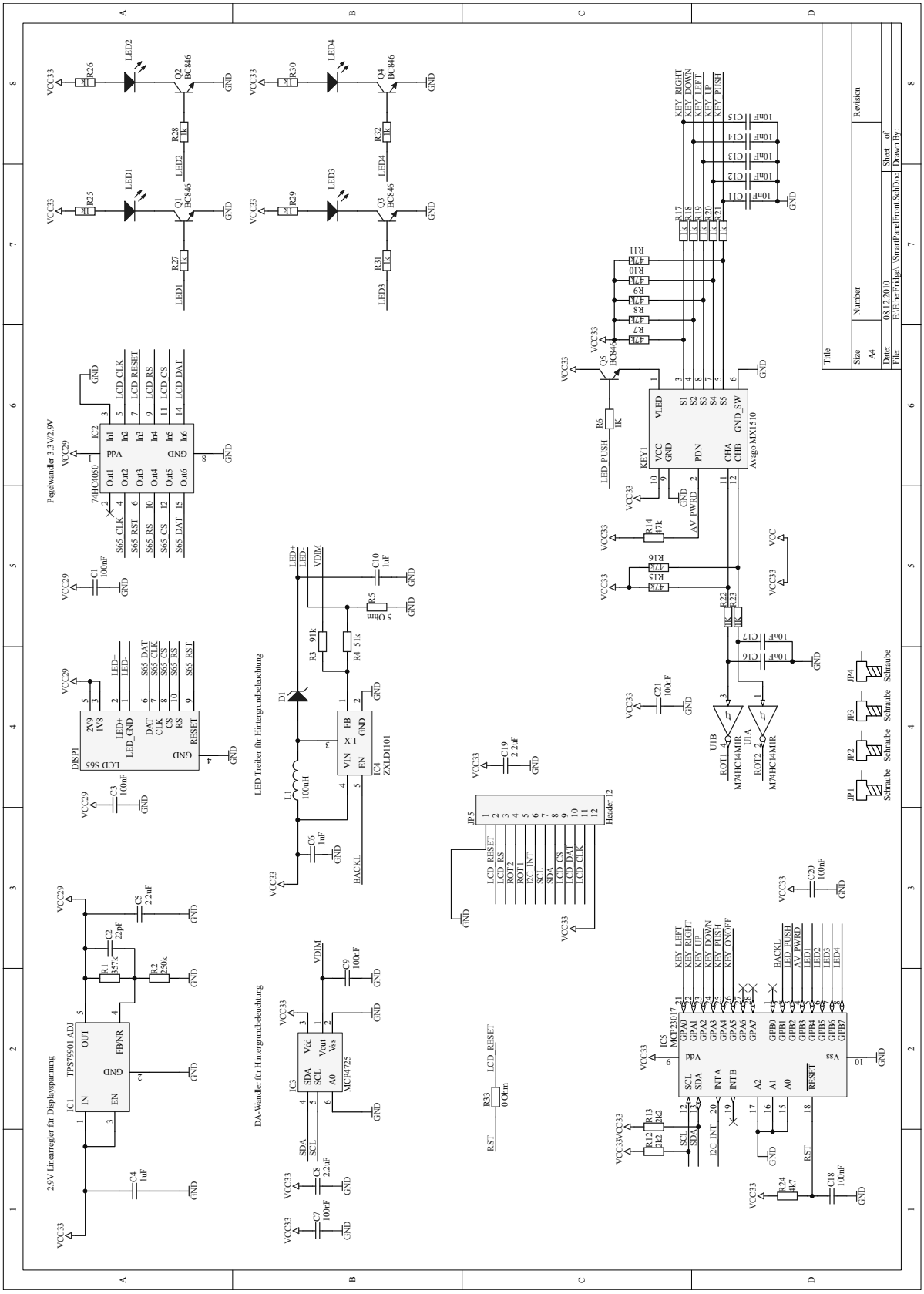

Abbildung 7.18: Schaltplan Displayansteuerung SmartPanel

7.3 Platinenlayout SmartFridge

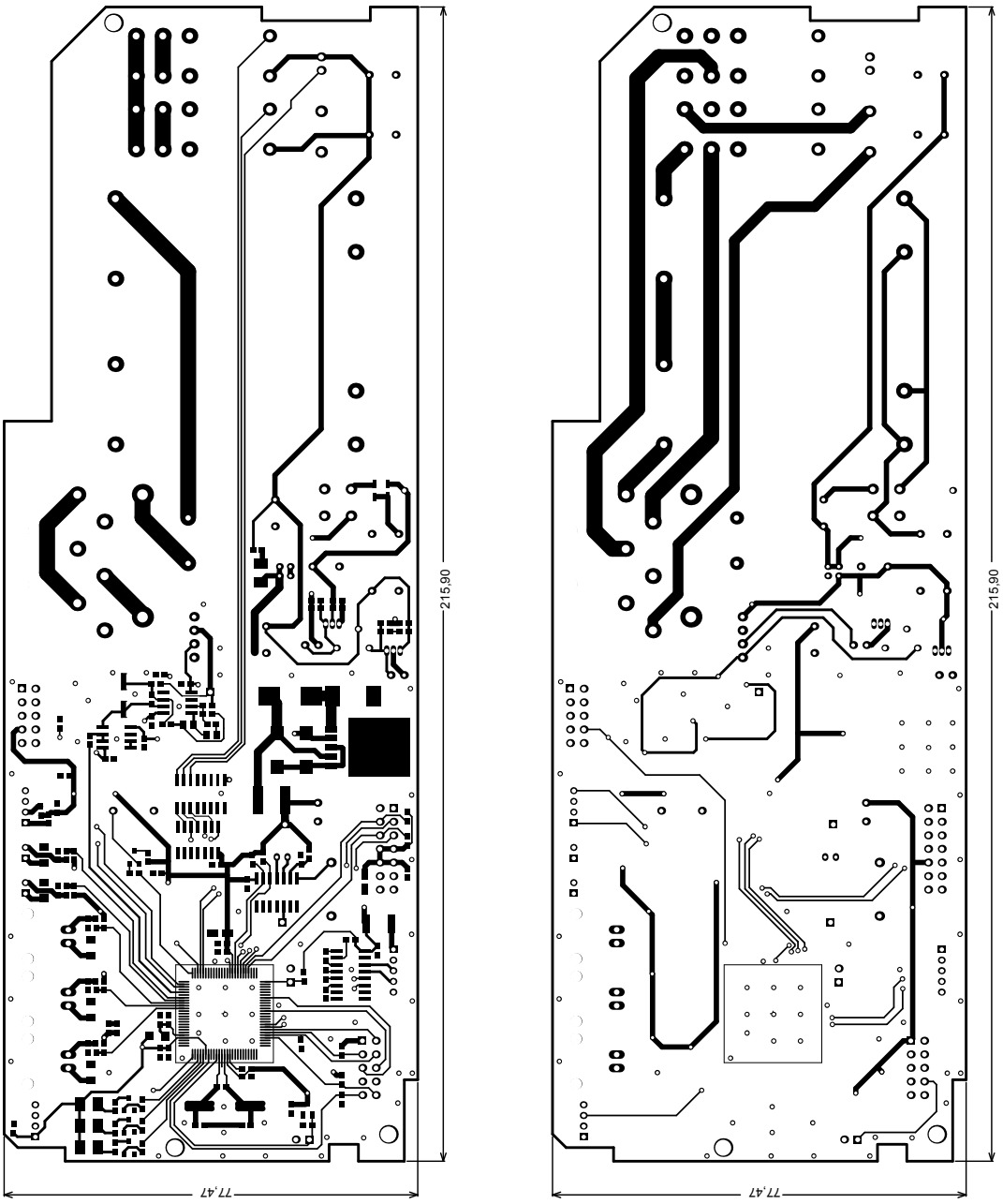

Abbildung 7.19: Layout SmartFridge: Top und Bottom

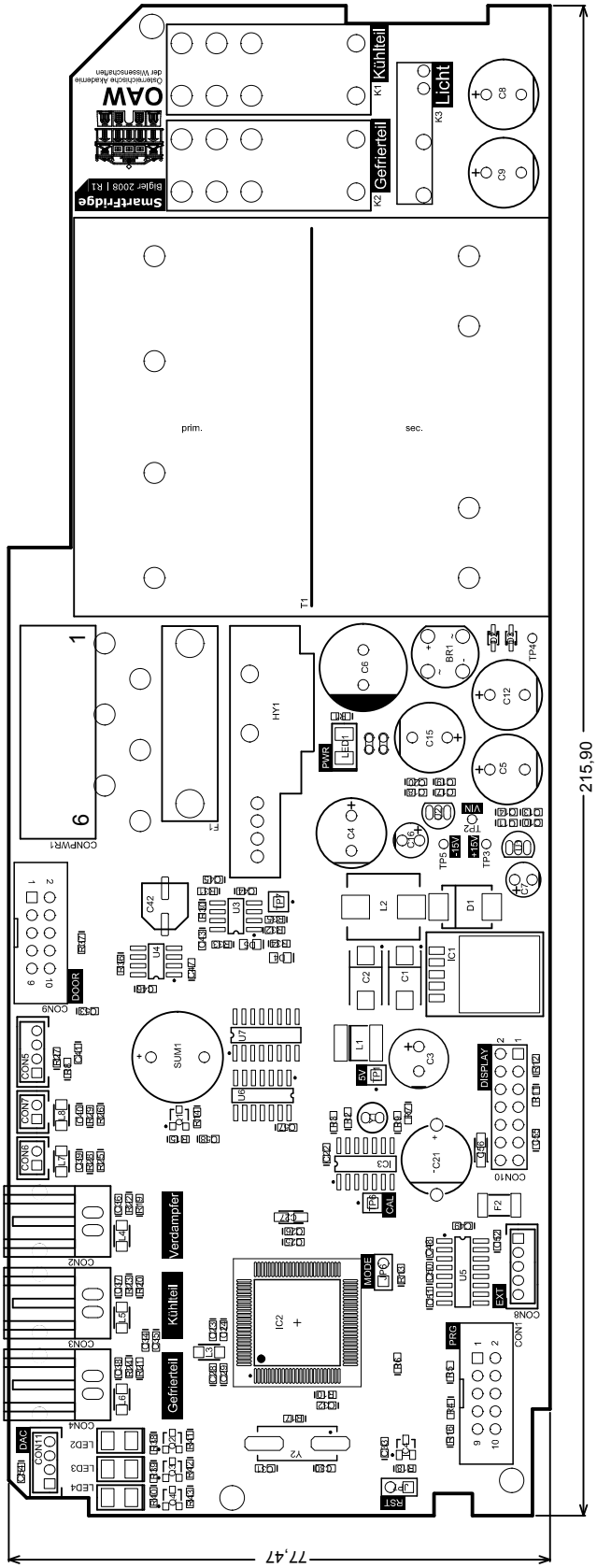

Abbildung 7.20: Layout SmartFridge: Bestückung

7.4 Platinenlayout SmartPanel

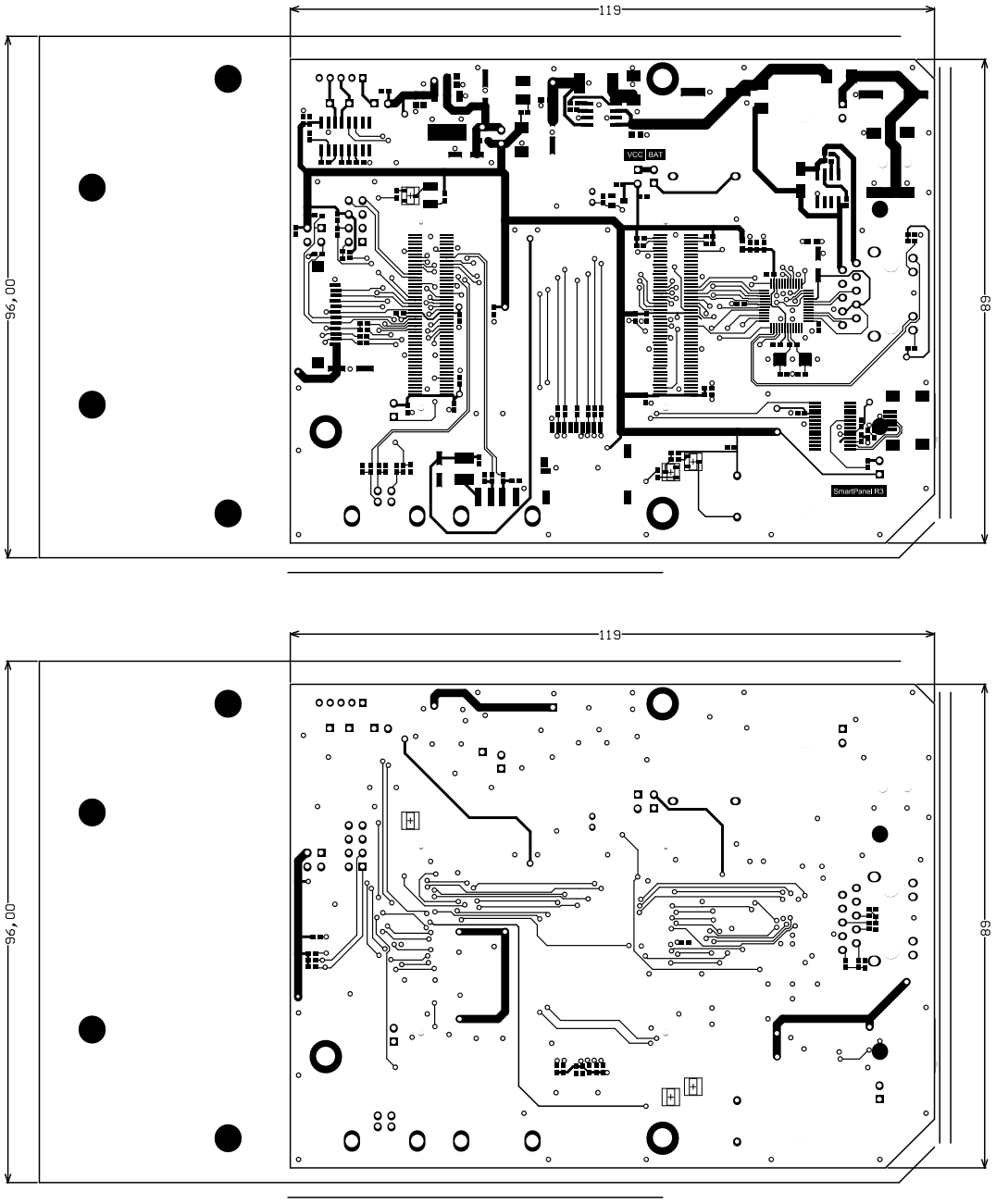

Abbildung 7.21: Layout SmartPanel: Top und Bottom

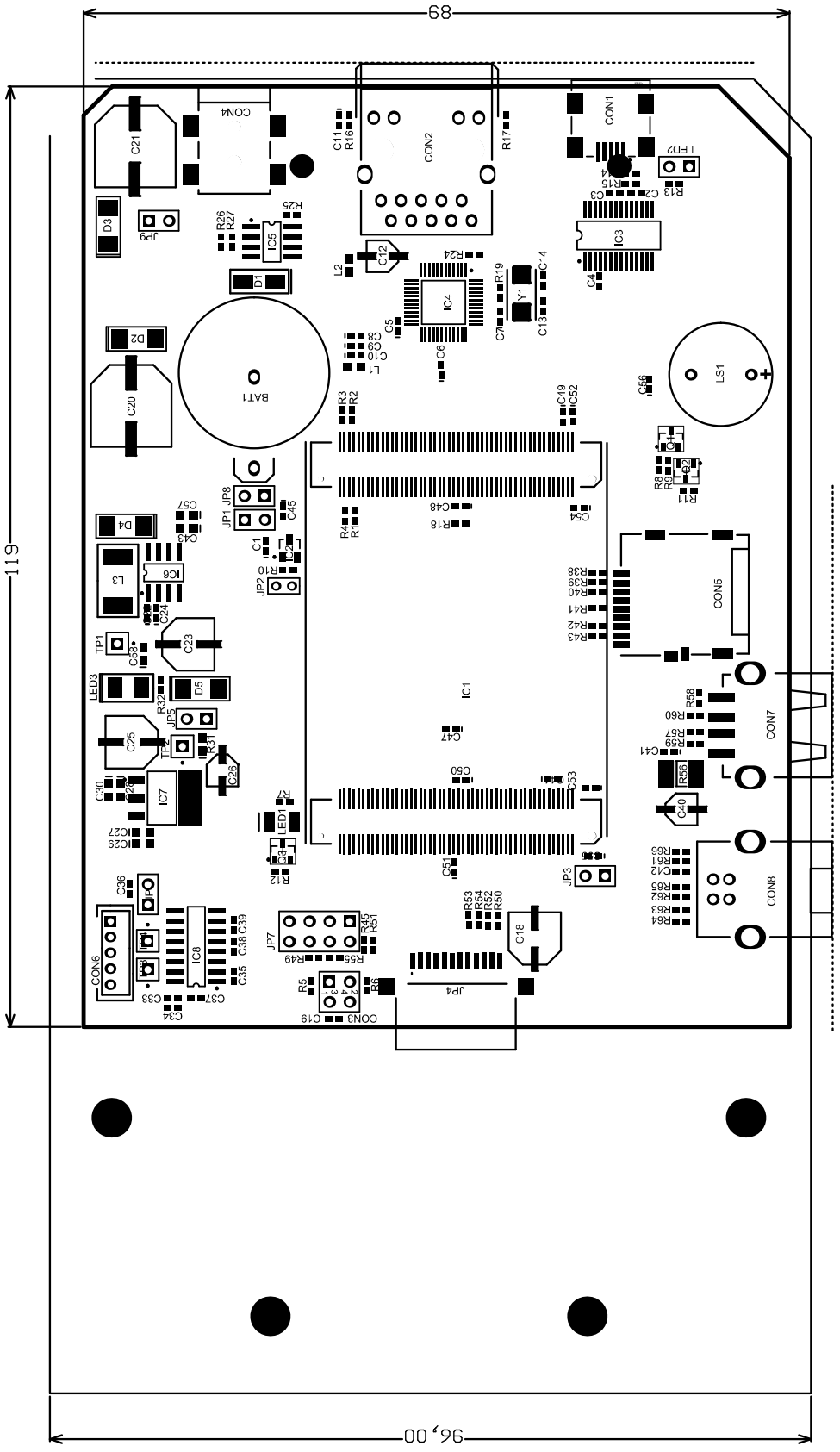

Abbildung 7.22: Layout SmartPanel: Bestückung

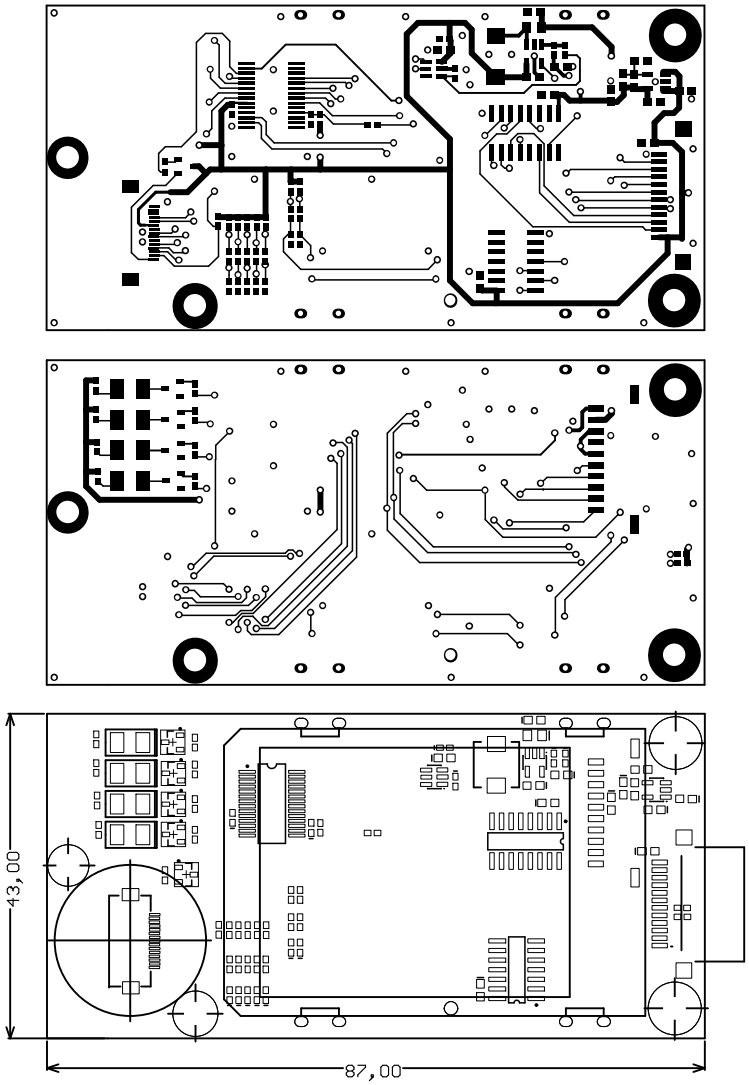

Abbildung 7.23: Layout SmartPanel Display

7.5 SmartFridge Firmware - Sourcecode Struktur

In diesem Abschnitt wird eine Übersicht zur Struktur der SmartFridge-Firmware gegeben. Abbildung 7.24 zeigt die logische Strukturierung des Quellcodes in der Entwicklungsumgebung. Die Unterteilung weicht nur in wenigen Punkten von der physikalischen Strukturierung der gespeicherten Daten ab.Die Einträge in den Rechtecken stellen die erste Ebene der Strukturierung dar. Folgende Unterteilungen wurden hier getroffen:

Main Hier befindet sich der Quellcode für das eigentliche Hauptprogramm, in dem alle anderen Module zusammengeführt werden.

Config Hier befinden sich die Konfigurationsdateien für den Prozessor und das Echtzeitbetriebssystem.

Driver In diesem Ordner befinden sich alle Treiber für die Peripherie.

Modules Hier werden die einzelnen Funktionsmodule zusammengefasst. Diese greifen im Allgemeinen auf die Treiber zu und stellen der Hauptanwendung höhere Funktionen zur Verfügung.

Utilities Hier befinden sich diverse Hilfsfunktionen.

UCOSII In diesem Ordner befinden sich die Dateien für das Echtzeitbetriebssystem ucos-II. Da diese Dateien ohne Änderung eingefügt wurden, werden sie hier nicht näher dargestellt und erläutert.

Bei der Darstellung in Abbildung 7.24 werden die C-Code- (.c) und Headerdateien (.h) in eine Zeile zusammengefasst. Dateien mit der Endung .a30 sind Assembler-Dateien, während .inc-Dateien die *include*-Dateien für diese darstellen - sie entsprechen also den Header-Dateien im C-Code. Die fett geschriebenen Einträge stellen weitere Unterordner dar.

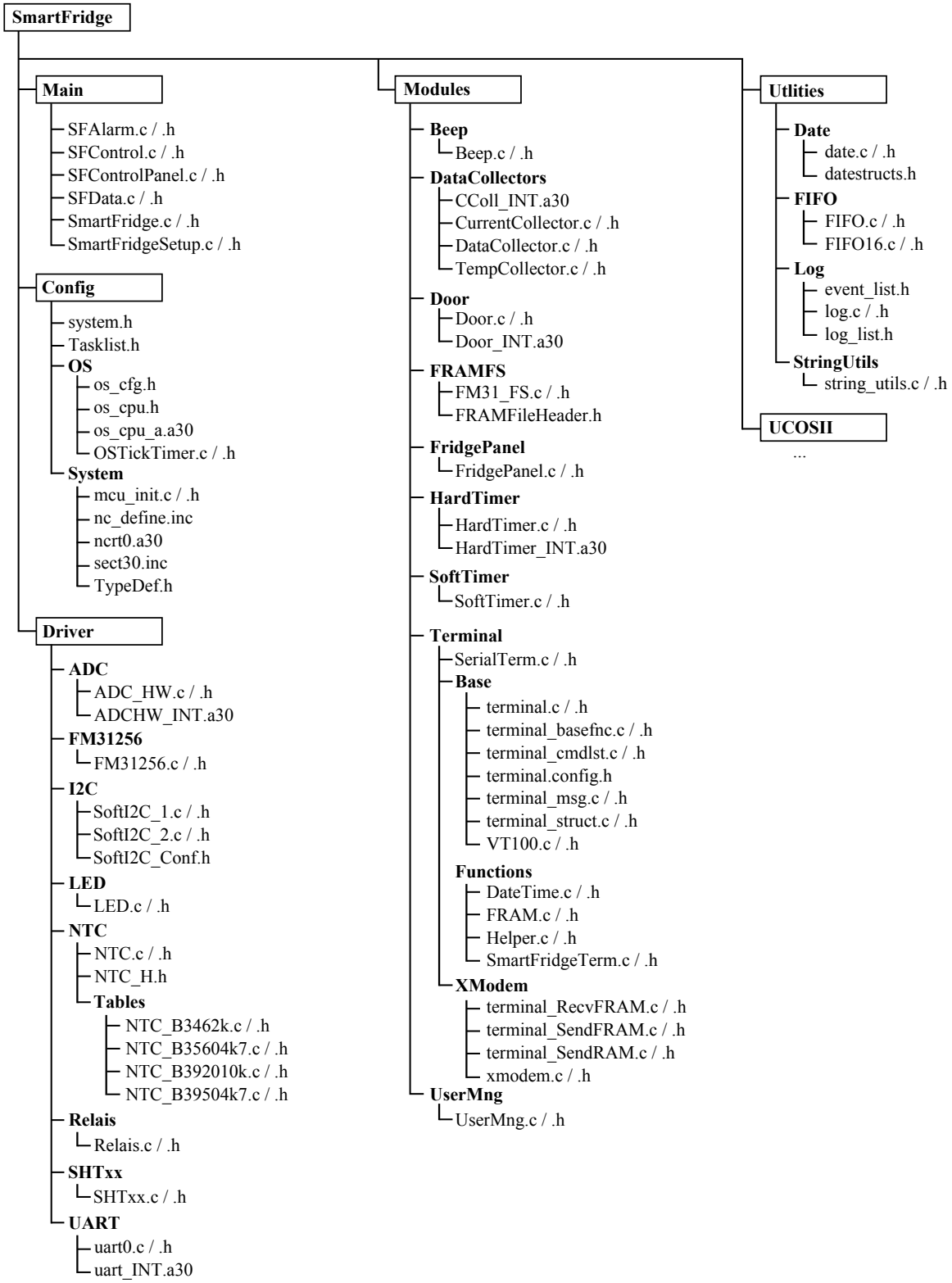

Abbildung 7.24: logische Strukturierung des Sourcecodes

7.6 SmartPanel Applikation - Sourcecode Struktur

Abbildung 7.25 ziegt die Struktur des Quellcodes für der SmartPanel-Applikationen (Menüsystem und FCGI-Interface).
Die Menüsystem-Applikation ist über Ordner logisch in mehrere Funktionsgruppen unterteilt. Nachfolgend eine Beschreibung dieser Einteilung:

SmartPanelMenu Hier befinden sich die Dateien für die Definition und Einrichtung des am SmartPanel-Display dargestellten Menüsystems. Dabei werden die vom SimpleWidgetSystem zur Verfügung gestellten Menüsystem-Elemente verwendet.

SimpleWidgetSystem In diesem Ordner befinden sich alle Dateien für den Menüsystem-Thread, die darstellbaren Container- und Control-Element, die Ansteuerung der Hintergrundbeleuchtung und der Ausgabe über den Beeper.

Database Hier befindet sich der Code für die Kommunikation mit der SQLite Datenank.

Serial In diesem Ordner befindet sich der Quellcode für die Kommunikation mit der SmartFridge über die serielle Schnittstelle.

FCGI Hier befindet sich der Code für die Verarbeitung der Kommunikation über die FCGI-Schnittstelle.

Der Quellcode für das FCGI-Interface zwischen Webserver und der Menüsystem-Applikation besteht lediglich aus drei Dateien die einen Zugriff auf die Datenbank und die Kommunikation mit der Menüsystem-Applikation ermöglichen.

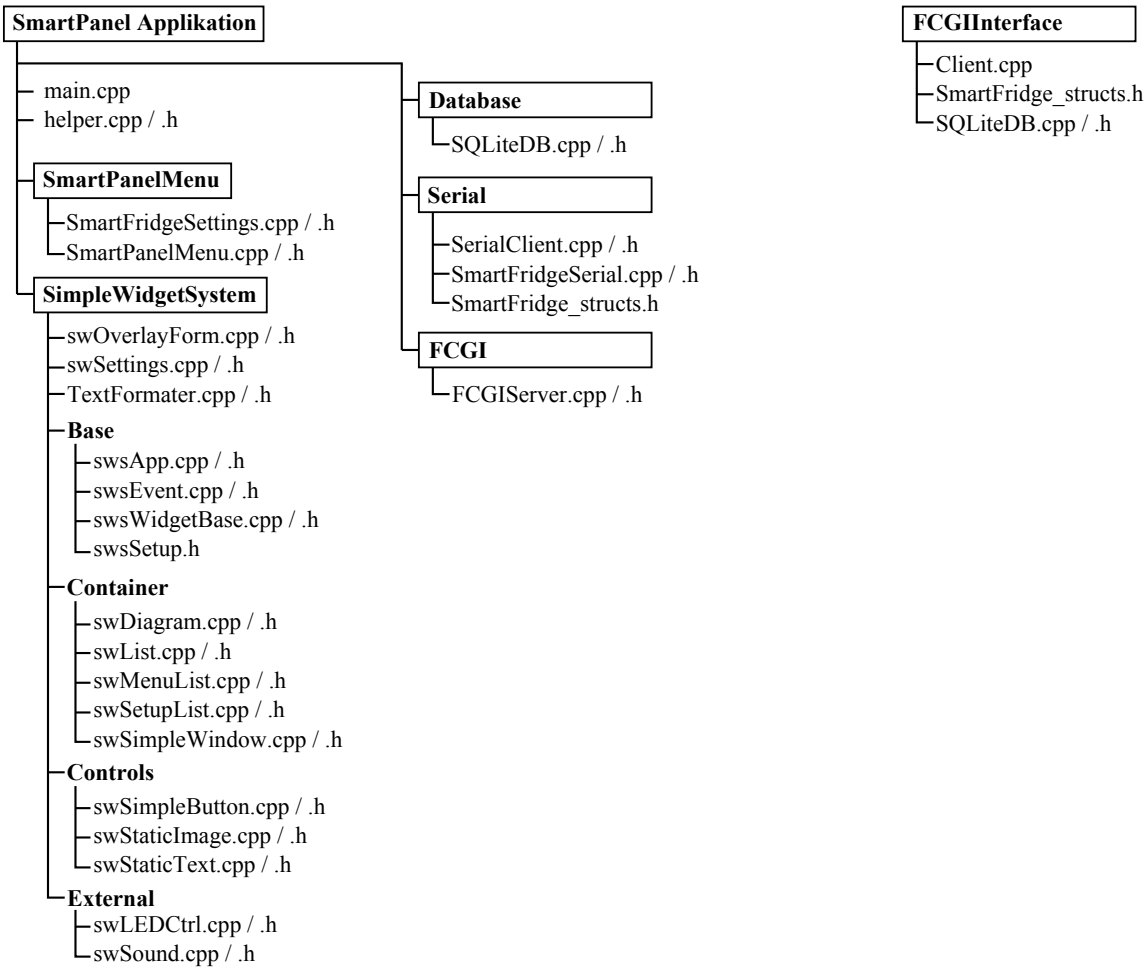

Abbildung 7.25: Strukturierung der SmartPanel Applikation

7.7 Event- und Logcodes

Die Tabellen 7.1 und 7.2 listen die mögliche Werte der Einträge im Event- und Logfile auf (siehe Abschnitt 3.2.2) auf.

Tabelle 7.1: ID und symbolische Bezeichnung der möglichen Logfile-Einträge

Fehlercode	Beschreibung
0x0000	Kein Fehler
0x0001	Maximale Anzahl an Terminal-Kommandos überschritten
0x0002	Array Größe nicht ausreichend für geforderte Operation
0x0003	Sonstiger Fehler bei Terminal
0x0004	Fehler bei Paramterübergabe, Ursache unbekannt
0x0005	Kommando ist in dieser Form nicht gültig
0x0101	FRAMFS: Datei nicht gefunden
0x0102	FRAMFS: Kein freier Dateiblock mehr
0x0103	FRAMFS: Ungültiger Block
0x0104	FRAMFS: Ungültiger/fehlender Filepointer
0x0105	FRAMFS: Datei konnte nicht angelegt werden
0x0106	FRAMFS: Datei konnte nicht geöffnet werden
0x0107	FRAMFS: Zähler Überlauf beim Ausgeben von Filedaten
0x0108	FRAMFS: Schreiben über Ende des Files entdeckt
0x0109	FRAMFS: Fehler beim Schreiben
0x010A	FRAMFS: Timeout
0x010B	FRAMFS: Undefinierter Fehler im FRAM-Dateisystem
0x0201	CCOLL: Current Collector Sampling konnte nicht gestartet werden
0x0202	DCOLL: Timeout bei ADC Messung
0x0203	CCOLL: Konflikt zwischen zwei angeforderten Strommessungen
0x0204	CCOLL: Messung konnte nicht gestartet werden
0x0205	CCOLL: Es gibt ein Problem mit den Daten
0x0206	DCOLL: Messung bei DataCollector konnte nicht gestartet werden
0x0207	DCOLL: Analyse konnte nicht durchgeführt werden
0x0208	DCOLL: Size bei Serienstrommessung musste beschränkt werden
0x0209	TCOLL: Ungültige Pointerposition
0x0301	Fridge: Ungültiger Wert
0x0302	Fridge: Invalid FridgePanel State
0x0303	SHT Sensor: CRC-Fehler beim Auslesen
0x0304	SHT Sensor: erfasster Rohwert außerhalb der Limits
0x0305	SHT Sensor: Timeout
0x0306	SHT Sensor: No ACK beim Senden des Befehls
0x0307	Fridge: Ungültige Programmposition
0x0401	SoftTimer: Kein freier Timer mehr
0x0601	I2C: Timeout einer I2C Funktion
0x0701	Relais: Gewünschter Relaisausgang nicht verfügbar

weiter auf nächster Seite

Fehlercode	Beschreibung
0x0901	TERM: XModem Protokoll hat Error geliefert
0x0902	TERM: XModem Protokoll hat Timeout geliefert
0x0903	TERM: XModem, Senden abgebrochen, size == 0
0xFE01	OS: Kein freier Eventblock mehr
0xFE02	OS: Priorität existiert schon
0xFE03	OS: Ungültige Priorität
0xFE04	OS: Keine freien TCB mehr vorhanden
0xFE05	OS: Versuch einen Task von einer ISR aus zu erzeugen
0xFE06	OS: Keine freie EventFlag Group mehr
0xFE07	OS: Fehler beim Löschen eines Tasks
0xFE08	OS: Fehler bei OSSemPend
0xFE09	OS: Unerlaubter Aufruf aus ISR
0xFE0A	OS: Null Pointer anstatt Event-Handle übergeben
0xFE0B	OS: Timeout bei Warten auf Flags
0xFE0C	OS: Ungültige Event-Flag Gruppe
0xFE0D	OS: Ungültige Optionen bei FlagGroup
0xFE0E	OS: PEVENT ist ein NULL Pointer
0xFE0F	OD: Timeout bei Pend
0xFEFF	OS: Undefinierter/Unbekannter Fehler
0xFF01	Die exit Funktion wurde aufgerufen (sollte nie auftreten)
0xFF02	StartOS() konnte nicht ausgeführt werden

Tabelle 7.2: ID und symbolische Bezeichung der möglichen Events

Eventcode	Beschreibung
0x0001	Tür wurde geöffnet
0x0002	Tür wurde geschlossen
0x0101	Alarm bei Cooler-Temperatur
0x0102	Alarm bei Freezer-Temperatur
0x0103	Alarm bei Cooler-Kompressor
0x0104	Alarm bei Freezer-Kompressor
0x0105	Bestätigung eines Alarms
0x0106	Tür zu lange geöffnet
0x0201	Alarmzustand Cooler-Temperatur zurückgesetzt
0x0202	Alarmzustand Freezer-Temperatur zurückgesetzt
0x0203	Alarmzustand Cooler-Kompressor zurückgesetzt
0x0204	Alarmzustand Freezer-Kompressor zurückgesetzt
0x020F	Alarmzustand wurde bestätigt (Beep kann deaktiviert werden)
0x0301	Warnung: Lampe scheinbar defekt
0xFFFE	Kritscher Fehler, System nicht voll funktionsfähig
0xFFFF	System wurde gestartet

7.8 Terminal Kommandos

In diesem Abschnitt werden die Kommandos, die über das serielle Terminal (siehe Abschnitt 3.2.9) bereit stehen, aufgelistet. Tabelle 7.3 fasst die Befehle zur Überwachung und Steuerung des Basissystems zusammen. In Tabelle 7.4 werden die zwei Kommandos zum Einstellen von Datum und Uhrzeit gezeigt.

Tabelle 7.3: Terminal Funktionen zur Systemsteuerung und Überwachung

Kommando		Beschreibung
version		Zeigt die Firmwareversion an
build		Zeigt das Build-Datum an
reboot		Startet das System neu
echo	{0|1}	(De)Aktiviert Terminal Echo
user	who	Zeigt den aktueller User an
	pwd {std|admin|root}	Setzt das Passwort eines Users
	lst	Zeigt die Passwörter (admin und root)
	login {std|admin|root}	Setzt neuen user (Passwortabfrage)
	lo	Zurücksetzen auf Standarduser
sysinfo		Zeigt aktuelle Systeminformationen an
tasks		Zeigt Informationen zu den aktuell laufenden Tasks an
log	{error|event}	Zeigt Einträge des Log- und Eventspeichers an
	{cerror|cevent}	Löscht Einträge aus Log- und Eventspeicher
help		Listet alle möglichen Kommandos auf

Tabelle 7.4: Terminal Funktionen für Datum und Uhrzeit

Kommando		Beschreibung
date	set {hh:mm:ss}	Setzen der Uhrzeit im Format hh:mm:ss
	get	Auslesen der aktuellen Uhrzeit
time	set {dd.MM.YY}	Setzen des Datums im Format dd.MM.YY
	get	Auslesen des aktuellen Datums

Die nachfolgende Tabelle 7.5 zeigt Unterbefehle für das Kommando zur Kontrolle des FRAM-Datenspeichers.

In Tabelle 7.6 werden die Befehle zur Einstellung der Regelparameter und des Bedienpanels aufgelistet. Tabelle 7.7 bietet verschiedene Kommandos um den Supercool/Superfrost-Modus zu aktivieren und den Zustand der Türöffnung und der Beleuchtung anzuzeigen. Weiters erfolgt mit diesem Kommando auch die Anzeige und Quitierung von Alarmen.

Tabelle 7.8 zeigt das Kommando zum Auslesen der aktuellen Sensorwerte und in Tabelle 7.9 werden die Kommandos zum Übertragen von Daten aus dem RAM über das XModem-Protokoll beschrieben.

Tabelle 7.5: Terminal Funktionen für den FRAM-Speicher und der Kalibrierung der Echtzeituhr

Kommando		Beschreibung
fram	list	Listet alle Dateien im FRAM auf
	send {filename}	Übertragung der Datei „filename“ mit XModem
	recv {filename}	Empfang der Datei „filename“ mit XModem
	stat	Zeigt aktuelle Informationen zum Speicherverbrauch
	format	Formatiert den Datenspeicher
	calon	Schaltet den FM31256-Baustein in den Kalibrierungsmodus
	caloff	Verlassen des Kalibrierungsmodus
	calset	Setzt einen Kalibrierungswert
	calget	Liest den aktuellen Kalibrierungswert aus

Tabelle 7.6: Terminal Funktionen zur Einstellung der Regelparameter

Kommando		Beschreibung
setup get	beep	Anzeigen ob Beeper aktiviert ist
	childlock	Anzeigen ob Kindersicherung aktiviert ist
	brightness	Helligkeit der Anzeige
	alarm	Anzeigen ob Alarm aktiviert ist
	{cooler|freezer} active	Zeigt an ob Kühlteil/Gefrierteil eingeschaltet
	{cooler|freezer} temp	Anzeige Solltemperatur
	{cooler|freezer} {hystp|hystn}	Hysteresewerte anzeigen
	{cooler|freezer} minoff	Minimale Ausschaltzeit
	{cooler|freezer} maxon	Maximale Einschaltzeit
	supercool timermax	Maximale Laufzeit Supercool-Modus
	superfrost timermax	Maximale Laufzeit Superfrost-Modus
setup set	beep {0|1}	(De)Aktivieren des Beepers
	childlock {0|1}	(De)Aktivieren der Kindersicherung
	brightness {0:7}	Setzen der Displayhelligkeit
	{cooler|freezer} active {0|1}	(De)Aktivieren des Kühlteils/Gefrierteils
	{cooler|freezer} temp {*value*}	Solltemperatur auf Wert *value* setzen
	{cooler|freezer} hystp {*value*}	Setzen des pos. Hysteresewerts
	{cooler|freezer} hystn {*value*}	Setzen des neg. Hysteresewerts
	{cooler|freezer} minoff {*value*}	Setzen der minimalen Ausschaltzeit
	{cooler|freezer} maxon {*value*}	Setzen der maximalen Einschaltzeit
	supercool timermax {*value*}	Setzen der maximalen Supercool-Laufzeit
	superfrost timermax {*value*}	Setzen der maximalen Superfrost-Laufzeit

Tabelle 7.7: Terminal Funktionen Supercool/Superfrost

Kommando		Beschreibung
control set	supercool active {0|1}	(De)Aktivieren von Supercool
	superfrost active {0|1}	(De)Aktivieren von Superfrost
control get	supercool active	Aktuellen Supercool Status anzeigen
	superfrost active	Aktuellen Superfrost Status anzeigen
	door	Aktuellen Status des Tür anzeigen
	light	Aktuellen Status der Beleuchtung anzeigen
control alarm	confirm	Aktuellen Alarm bestätigen
	status	Aktuellen Alarmstatus anzeigen

Tabelle 7.8: Terminal Funktionen zur Anzeige von Sensorwerten

sensor get	temp {*id*}	Anzeige des Wertes von Sensor *id*
	hum	Anzeige der aktuellen Luftfeuchtigkeit
	current	Anzeige des aktuellen Stromverbrauchs
	all	Alle aktuellen Messwerte anzeigen

Tabelle 7.9: Terminal Funktionen zum Auslesen von Datenstrukturen

data get	events	Im RAM gespeicherten Events
	errors	Im RAM gespeicherten Fehler/Logeinträge
	alarm	Datenstruktur des Alarmstatus
	ctrl	Datenstruktur der Regelung
	sensors	Datenstruktur der aktuellen Sensorwerte
	setup	Datenstruktur des aktuellen Setups
	avgsensor	Datenstruktur der geglätteten Messwerte
	memtemp	Im RAM gespeicherter Temperaturverlauf
	memcseries	Letzte Serienstrommessung

7.9 Tastenkombinationen

Tabelle 7.10: Tastenkombinationen an der Bedieneinheit im Normalmodus

Taste	kurzer Tastendruck	langer Tastendruck
Cooler Up	Solltemp. Kühlteil ++	Solltemp. Kühlteil ++ (Tastenwiederholung)
Cooler Down	Solltemp. Kühlteil --	Solltemp. Kühlteil -- (Tastenwiederholung)
Freezer Up	Solltemp. Gefrierteil ++	Solltemp. Gefrierteil ++ (Tastenwiederholung)
Freezer Down	Solltemp. Gefrierteil --	Solltemp. Gefrierteil -- (Tastenwiederholung)
Cooler On/Off	Temperatur Kühlbereich anzeigen	Kühlteil ein/aus
Freezer On/Off	Temperatur Gefrierbereich anzeigen	Gefrierteil ein/aus
Supercool		Supercool aktivieren
Superfrost		Superfrost aktivieren
Childlock & Supercool		Kindersicherung ein/aus
Alarm	Alarm bestätigen	Wechsel in Setup Modus

Tabelle 7.11: Tastenkombinationen an der Bedieneinheit im Setup Modus

Taste	einfacher Tastendruck	langer Tastendruck
Cooler Up	Auswahl Paramater ID	Auswahl Parameter ID
Cooler Down		(Tastenwiederholung)
Freezer Up	Modifikation Parameter Wert	Modifikation Parameter Wert
Freezer Down		(Tastenwiederholung)
Freezer On/Off	Wert für Parameter übernehmen	
Alarm		Setup Modus verlassen

Abbildungsverzeichnis

Tabellenverzeichnis

Wissenschaftliche Literatur

[Bau06] Baumann, Peter: *Sensorschaltungen: Simulation mit PSPICE.* Wiesbaden : Friedr. Vieweg & Sohn Verlag, 2006

[Bux09] Buxton, Stephen: *Database design: Know it all.* Amsterdam : Elsevier/Morgan Kaufmann Publishers, 2009. – ISBN 9780123746306

[CB01] Carter, Bruce ; Brown, Thomas R.: *Handbook of Operational Amplifier Applications.* Texas Instruments, Oktober 2001

[Chu07] Churcher, Clare: *Beginning Database Design.* Berkeley, CA : Clare Churcher, 2007. – ISBN 9781430203667

[CRKH05] Corbet, Jonathan ; Rubini, Alessandro ; Kroah-Hartman, Greg: *Linux device drivers.* 3. ed. Beijing : O'Reilly, 2005. – ISBN 9780596005900

[DGS09] Dillmann, Rüdiger ; Gockel, Tilo ; Schröder, Joachim: *Embedded Linux: Das Praxisbuch.* Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2009 (X.systems.press). – ISBN 9783540786207

[DPKD06] Dietrich, Dietmar ; Palensky, Peter ; von Klot, Sandrine ; Dietrich, Dorothee. *Das digitale Gebäude - Netzwerke in der Gebäudeautomation.* April 2006

[EGS03] Eckel, Christian ; Gaderer, Georg ; Sauter, Thilo. *Implementation requirements for web-enabled Appliances - a case study.* ETFA 03. Dezember 2003

[EL10] Ellerbrock, Arne ; Loviscach, Jörn. *Das Strom-Netz - IT in der Stromversorgung: Twitternde Stromzähler und abwartende Waschmaschinen.* c't Magazin, Heise Verlag. Jänner 2010

[Epc06] Epcos: *NTC Thermistors - General technical information*, März 2006

[EU06] europäische Parlament und der Rat der Europäischen Union, Das. *Richtlinine 2006/32/EG des europäischen Parlaments und des Rates über Endenergieeffizienz und Energiedienstleistungen und zur Aufhebung der Richtlinie 93/76/ EWG des Rates.* April 2006

[For88] Forsberg, Chuck. *XMODEM/YMODEM Protocol Reference - A compendium of documents describing the XMODEM and YMODEM File Transfer Protocols.* Oktober 1988

[Fri07] Friedman, Vitaly: *Praxisbuch Web 2.0 - Moderne Webseiten programmieren und gestalten.* Galileo Press, August 2007

[Gad02] Gaderer, Georg: *Entwicklung einer EIB-Anschaltung für einen intelligenten Kühlschrank - SmartFridge.EIB*, Technische Universität Wien, Diplomarbeit, Oktober 2002

[GB09] Groiss, Christoph ; Brauner, Günther. *Power Demand Side Management – Potentialabschätzung im Haushalt.* IEWT 2009. 2009

[Gmb08] GmbH, Weinzierl E. *"KNX IP only A New Class of KNX Devices.* September 2008

[GTS+02] Gourley, David ; Totty, Brian ; Sayer, Marjorie ; Reddy, Sailu ; Aggarwal, Anshu: *HTTP: The Definitive Guide.* O'Reilly, 2002

[GW09] Gu, Hanshen ; Wang, Dong. *A Content-aware Fridge Based on RFID in Smart Home for Home-Healthcare.* Februar 2009

[Haa97] Haager, Wilhelm: *Regelungstechnik.* Verlag Hölder-Pichler-Tempsky, 1997

[Hof09] Hoffmann, Manuela: *Modernes Webdesign.* Galileo Press, Dezember 2009

[HR06] Harris, Simon ; Ross, James: *Beginning algorithms.* Indianapolis, Ind. : Wiley, 2006 (Programmer to programmer). – ISBN 0764596748

[IEE03] IEEE: *802.3af - Power via Media Dependent Interface (MDI).* June 2003

[Int08] *Internet der Energie - IKT für Energiemärkte der Zukunft.* Bundesverband der Deutschen Industrie e.V. (BDI), Dezember 2008

[Kar09] Karlsson, Björn: *Beyond the C++ standard library: An introduction to Boost.* 5. print. Upper Saddle River, NJ : Addison-Wesley, 2009. – ISBN 9780321133540

[KNM95] Kawaguchi, Atsuo ; Nishioka, Shingo ; Motoda, Hiroshi. *A Flash-Memory Based File System.* 1995

[Kre10] Kreibich, Jay A.: *Using SQLite.* 1st ed. Beijing : O'Reilly, 2010. – ISBN 9780596521189

[Lab02] Labrosse, Jean J.: *MicroC/OS-II: The real-time kernel.* 2. ed. San Francisco, Calif. : CMP Books, 2002. – ISBN 1578201039

[Ler07] Lerch, Reinhard. *Elektrische Messtechnik: Analoge, digitale und computergestützte Verfahren.* 2007

[Log10] Logofatu, Doina: *Algorithmen und Problemlösungen mit C++: Von der Diskreten Mathematik zum fertigen Programm - Lern- und Arbeitsbuch für Informatiker und Mathematiker.* 2., überarb. u. erw. Aufl. Wiesbaden : Vieweg + Teubner, 2010 (Studium). – ISBN 978–3–8348–0763–2

[Lov05] Love, Robert: *Linux Kernel development.* 2. ed. Indianapolis, Ind. : Novell, 2005 (Development). – ISBN 0672327201

[LXG+08] Luo, Suhuai ; Xia, Hongfeng ; Gao, Yuan ; Jin, Jesse S. ; Athauda, Rukshan. *Smart Fridges with Multimedia Capability for Better Nutrition and Health.* 2008

[Mag04] Magerl, Gottfried. *Messtechnik*. 2004

[Nat99] National Semiconductor: *LM2595 SIMPLE SWITCHER Power Converter 150 kHz 1A Step-Down Voltage Regulator - Datasheet*. Mai 1999

[NXP07] NXP (Philips): *I2C-bus specification and user manual*. June 2007

[Owe06] Owens, Michael: *The Definitive Guide to SQLite*. Berkeley, CA : Michael Owens, 2006. – ISBN 9781430201724

[Pal01] Palensky, Dipl.-Ing. P.: *Distributed Reactive Energy Management*, Technischen Universität Wien, Diplomarbeit, Februar 2001

[Phi96] Philfky, Dr. Elliot M. *FRAM - The Ultimative Memory*. 1996

[QK04] Quade, Jürgen ; Kunst, Eva-Katharina: *Linux-Treiber entwickeln: Gerätetreiber für Kernel 2.6 systematisch eingeführt*. Heidelberg : dpunkt.verl., 2004. – ISBN 3898642380

[Qui10] Quigley, Ellie: *JavaScript by Example 2nd Edition*. Prentice Hall, Oktober 2010

[Ram10] Ramtron: *FM3104/16/64/256 - Integrated Processor Companion with Memory - Datasheet*. Juli 2010

[Ramer] Ramtron: *F-RAM Technology Brief - Technology Note*. September. – 2007

[Ren04] Renesas: *M16C/60 Software Manual*, Jänner 2004

[Ren06] Renesas: *M16C/62P Group Hardware Manual*, Jänner 2006

[RLN06] Raghavan, Pichai ; Lad, Amol ; Neelakandan, Sriram: *Embedded Linux system design and development*. Boca Raton, Fla. : Auerbach Publ., 2006. – ISBN 0849340586

[Sch03] Schliepkorte, Hans-Jürgen. *Das intelligente Heim: Einbindung von Haushaltsgeräten*. BUS Systeme Heft 3/2003. 2003

[Sen06] Sensirion: *Non-Linearity Compensation - Application Note*, Oktober 2006

[Sen07] Sensirion: *SHT1x / SHT7x - Humidity & Temperature Sensor Datasheet*, August 2007. – v3.01

[Sta10] Staub, Richard. *Ist digitalSTROM schon bald marktfähig?* April 2010

[TSG02] Tietze, Ulrich ; Schenk, Christoph ; Gamm, Eberhard: *Halbleiter-Schaltungstechnik*. Springer, 2002

[VD10] Vasseur, Jean-Philippe ; Dunkels, Adam: *Interconnecting smart objects with IP: The next Internet*. Burlington, Mass. : Morgan Kaufmann/Elsevier, 2010. – ISBN 9780123751652

[Ven09] Venkateswaran, Sreekrishnan: *Essential Linux device drivers*. 4. print. Upper Saddle River, NJ : Prentice-Hall, 2009 (Prentice Hall open source software development series). – ISBN 9780132396554

[Voi99] Voith, Rainer: *Entwicklung eines Feldbus-Interfaces für eine Kühl-Gefrier-Gerätekombination*, Technische Universität Wien, Diplomarbeit, 1999

[WC07] Wu, Hai-Ning ; Chang, Li-Pin. *A Stackable Wear-Leveling Module for Linux-Based Flash File Systems.* 2007

[Wil05] Williams, Tim: *The Circuit Designer's Companion.* Newnes / Elsevier Ltd., 2005

[Zac08] Zacher, Serge ; Reuter, Manfred (Hrsg.): *Regelungstechnik für Ingenieure: Analyse, Simulation und Entwurf von Regelkreisen.* 12., korrigierte und erweiterte Auflage. Wiesbaden : Vieweg +Teubner / GWV Fachverlage GmbH Wiesbaden, 2008

Internet Referenzen

[56] *AT91SAM Community*, November 2010. http://www.at91.com.

[57] *Avago Technologies*, November 2010. http://www.avagotech.com.

[58] *boost C++ Libraries*, November 2010. http://www.boost.org.

[59] *Boost.Asio Webseite*, November 2010. http://www.boost.org/doc/libs/1_45_0/doc/html/boost_asio.html.

[60] *CLOC Count Lines of Code*, November 2010. http://cloc.sourceforge.net.

[61] *Das U-Boot - Universal Bootloader*, November 2010. http://sourceforge.net/projects/u-boot.

[62] *Die Boost C++ Bibliotheken*, November 2010. http://www.highscore.de/cpp/boost/.

[63] *digitalSTROM Webseite*, November 2010. http://www.digitalstrom.org/.

[64] *Eclipse CDT (C/C++ Development Tooling)*, November 2010. http://www.eclipse.org/cdt/.

[65] *fastcgi++: A C++ FastCGI Library*, November 2010. http://www.nongnu.org/fastcgipp/.

[66] *flot - Attractive Javascript plotting for jQuery*, November 2010. http://code.google.com/p/flot/.

[67] *Flot Library API Reference*, November 2010. http://people.iola.dk/olau/flot/API.txt.

[68] *The FreeRTOS Project*, November 2010. http://www.freertos.org/.

[69] *Introducing JSON*, November 2010. http://www.json.org/.

[70] *jQuery - write less do more*, November 2010. http://jquery.com/.

[71] *LEM Webseite*, 2010. http://www.lem.com.

[72] *Liebherr Net@Home Webseite - Das mediale Haus*, 2010. http://www.das-mediale-haus.de/liebherr.php.

[73] *LIGHTTPD - fly light*, November 2010. `http://www.lighttpd.net/`.

[74] *The Linux USB Input Subsystem*, November 2010. `http://www.linuxjournal.com/article/6396`.

[75] M16c family features, November 2010. `http://www.renesas.com/products/mpumcu/m16c/child_folder/m16c_getst_child.jsp?title=M16C+Advantage`.

[76] M16c family (r32c/m32c/m16c), November 2010. `http://www.renesas.com/products/mpumcu/m16c/m16c_landing.jsp`.

[77] *Micrium*, November 2010. `http://micrium.com`.

[78] *Miele@Home Webseite*, November 2010. `http://www.miele.de/de/haushalt/produkte/180.htm`.

[79] *MiniGUI - A cross-operating-system graphics user interface*, November 2010. `http://www.minigui.org/`.

[80] *mySQL Webseite*, 2010. `http://www.mysql.de`.

[81] *PSplash Webseite*, November 2010. `http://labs.o-hand.com/psplash`.

[82] *Qt - Cross-platform application and UI framework*, November 2010. `http://qt.nokia.com/`.

[83] *SELFHTML 8.1.2 (HTML-Dateien selbst erstellen)*, November 2010. `http://de.selfhtml.org/`.

[84] *Signal-Slot-Konzept*, November 2010. `http://de.wikipedia.org/wiki/Signal-Slot-Konzept`.

[85] *SQLite Webseite*, November 2010. `http://www.sqlite.org`.

[86] *Taskit Embedded Systems*, November 2010. `http://www.taskit.de/`.

[87] *Usage of javascript libraries for websites*, November 2010. `http://w3techs.com/technologies/overview/javascript_library/all`.

[88] *Using the Siemens S65-Display*, November 2010. `http://www.superkranz.de/christian/S65_Display/DisplayIndex.html`.

[89] *Versorgungsqualität*, November 2010. `http://www.vattenfall.de/de/distribution/versorgungsqualitat-berlin.htm`.

[90] *YAFFS (Yet Another Flash File System)*, November 2010. `http://www.yaffs.net/`.

[91] E-Control. *Smart Metering - Die zentrale Rolle des Messwesens im Energiemarkt*, November 2010. `http://e-control.at/de/marktteilnehmer/strom/smart-metering`.

[92] iana. *MIME Media Types*, November 2010. `http://www.iana.org/assignments/media-types/`.

[93] Wikipedia. *Fluent Interface*, November 2010. `http://de.wikipedia.org/wiki/Fluent_Interface`.

[94] Wikipedia. *Gleitender Mittelwert*, November 2010. `http://de.wikipedia.org/wiki/Gleitender_Mittelwert`.

[95] Wikipedia. *Liste der Datenbankmanagementsysteme*, November 2010. `http://de.wikipedia.org/wiki/Liste_der_Datenbankmanagementsysteme`.