Bernburg
Dessau
Köthen

Hochschule Anhalt
Anhalt University of Applied Sciences

emw
Fachbereich
Elektrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen

Masterarbeit

zur Erlangung des akademischen Grades
Master of Engineering (M. Eng.)

Bing Zhang

Vorname Nachname

Elektro- und
Informationstechnik, 2010, 4052539

Studiengang, Matrikel, Matrikelnummer

Thema:

Temperaturüberwachung und Steuerung eines Heiz- und Kühlsystems mittels Daten-austauschs über das GSM-Netz

Prof. Dr.-Ing. Eduard Siemens

Vorsitzende(r) der Masterprüfungskommission

Dr.-Ing. Ingo Chmielewski

1. Prüfer(in)

Prof. Dr.-Ing. Eduard Siemens

2. Prüfer(in)

28.01.2013

Abgabe am

Selbstständigkeitserklärung

Hiermit erkläre/n ich/wir, dass die Arbeit selbstständig verfasst, in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt wurde und keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet wurden.

Köthen, 28.01.2013

Ort, Datum | Unterschrift/en der/des Studierenden

Sperrvermerk

Sperrvermerk: ja [ ] nein [X]

wenn ja: Der Inhalt der Arbeit darf Dritten ohne Genehmigung der/des (Bezeichnung des Unternehmens) nicht zugänglich gemacht werden. Dieser Sperrvermerk gilt für die Dauer von Jahren.

Köthen, 28.01.2013

Ort, Datum | Unterschrift/en der/des Studierenden

Inhaltsverzeichnis

Abkürzungsverzeichnis

GSM	Global System for Mobile Communications
SMS	Short Message Service
SoC	System on a Chip
LCD	Liquid Crystal Display
ARM	Advanced RISC Machines
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver Transmitter
SPI	Serial Peripheral Interface
GPIO	General Purpose Input/Output
MPU	Micro Processor Unit
ROHS	Restriction of Hazardous Substances
SIM	Subscriber Identity Module
EGSM	Extended Global System for Mobile
RISC	Reduced Instruction Set Computer
A/D	Analog/Digital
LSB	Least Significant Bit
MSB	Most Significant Bit
GND	Ground
VDD	Voltage Drain
ID	Identification
EIA	Electronic Industries Association
DCD	Data Carrier Detect
DTR	Data Terminal Ready
POSIX	Portable Operating System Interface based on UNIX
GPRS	General Packet Radio Service
3G	3rd Generation of Mobile Telecommunications Technology

1 Einleitung

1.1 Hintergrund und Anforderungen

In der heutigen Zeit wird das Handy immer mehr in verschiedenen Bereichen benutzt. Und die älteren Flüssigkeitsthermometer werden wegen ihrer Ungenauigkeit, Zerbrechlichkeit und Gefahr durch austretende Quecksilberdämpfe den Ansprüchen unserer modernen Technik längst nicht mehr gerecht und finden demzufolge keine Anwendung mehr. Deswegen wird im Rahmen dieser Masterarbeit ein Gerät entwickelt, welches die Temperatur messen und die Daten mit dem vorhandenen GSM-Netz verbinden und anzeigen kann. Ziel ist eine Temperaturüberwachung in bestimmten Bereichen, wie zum Beispiel in der Nutztierhaltung. Die Mitarbeiter sind nach Feierabend zu Hause und werden sofort über SMS darüber informiert, wenn sich die Temperatur in einem für die Tiere kritischen Bereich befindet. Ist dies der Fall, sendet das Gerät dem Benutzer eine SMS, und er kann sofort handeln. Diese Art der Temperaturüberwachung ist sehr vorteilhaft. Selbstverständlich kann der Benutzer dann eine Rück-SMS senden, um das gewünschte Temperaturintervall einzustellen.
Mit diesem GSM-Temperaturüberwachungssystem kann der Benutzer weltweit, überall dort, wo er Handyempfang hat, die Temperatur abfragen, zum Beispiel im Ferienhaus, im Gewächshaus, in landwirtschaftlichen Nutztieranlagen und damit sogar systemabhängig die Aktoren (Heizung) steuern. Mit diesem komfortablen GSM-Temperaturüberwachungsgerät wird das Eigenheim oder Haus sicher vor Kälteschäden, oder vor dem Risiko eines Feuerschadens geschützt. Damit könnten eingefrorene Leitungen und die daraus resultierenden erheblichen Folgekosten der Vergangenheit angehören. Damit ein solches System zuverlässig arbeiten kann, werden hohe Anforderungen an seine Entwicklung gestellt.

1.2 Zielsetzung der Arbeit

Eine Datenkommunikation unter Benutzung des standardisierten GSM-Systems ist weltweit verbreitet. Im einfachsten Fall lässt sich das zu übertragende Datum in einen SMS-Container verpacken. Auf diese Weise reduziert sich die Komplexität eines Datenprotokolls auf das Erstellen und Auswerten (Parsen) einer festzulegenden SMS-Phrase. Da das Versenden und Empfangen von SMS standardisiert ist, lässt sich einfach eine bidirektionale Datenverbindung aufbauen. Diese kann sowohl zum Senden von Umgebungsdaten, wie z.B. Temperatur, Luftdruck etc. an ein Auswertesystem verwendet werden, als auch zur Steuerung von Aktoren, wie z.B. eines Heiz- und Kühlsystems.
Ein an ein Mikrocontrollersystem, das Linux basiert arbeitet, angeschlossener Temperatursensor soll zur Überwachung der Umgebungstemperatur (-55 Grad Celsius bis +120 Grad

Celsius) dimensioniert und entwickelt werden. Die zu messende Temperatur wird periodisch vom Mikrocontrollersystem aufgenommen und verarbeitet. Aufgrund von einstellbaren Rahmenbedingungen werden die Rohdaten oder auch bereits vorverarbeitete Datensätze von diesem System mittels eines GSM-Moduls an ein Auswertesystem verschickt. Hierzu ist auf Basis des existierenden SMS-Mechanismus ein geeignetes Protokoll zu entwickeln.
Das Auswertesystem (im einfachsten Fall ein GSM-Handy) soll neben der Anzeige der empfangenden Daten (z. B. Temperatur oder auch Systemmeldungen) in der Lage sein, zu definierende Steuersequenzen an das Mikrocontrollersystem mit angeschlossenem Sensor zu verschicken. Diese Steuersequenzen sollen zum einen zum direkten Steuern (z. B. Betriebsmodus und Art der System- und Datenmeldungen) dienen und zum anderen auch die Beeinflussung von externen Aktoren, wie z. B. ein Kühl- und Heizsystem ermöglichen. [1]

2 Aufbau des Systems und das daraus resultierende Design

Im Folgenden wird ein kurzer Einblick in die verwendete Hardware und deren Aufbau gegeben. Im Anschluss daran werden die hier verwendeten Geräte eingeordnet und erläutert.

2.1 Blockschaltbild

Für das ganze System gibt es zwei Komponenten: Komponente Eins und Zwei. Komponente Eins ist das aufzubauende Gerät, das aus Temperatursensor, SoC(BeagleBone-Platine), GSM-Modem, LCD und Schaltinterfaces für die Beeinflussung von externen Aktoren besteht. Komponente Zwei ist das Auswertesystem, mit dem der Benutzer direkt SMS versenden und lesen kann.

Als Prozessor wird ein ARM Prozessor Cortex A8 [2] verwendet, der auf einer Platine BeagleBone mit notwendiger Peripherie aufgebaut ist. Der angeschlossene Tempertatursensor wird mittels eines 1-Wire Busses, siehe Abschnitt 3.1.1, mit dem SoC verbunden, und der Datenaustausch zwischen dem SoC und dem GSM-Modul wird über eine USB-Schnittstelle realisiert. Die periodisch gemessene Temperatur und Informationen des Systems, wie z.B. die Zeit und das Datum, werden mit der Steuerung des Prozessors mittels eines Daten-Busses und eines Kontroller-Busses auf dem LCD angezeigt. Das System ist außerdem in der Lage, die notwendigen Interfaces zur Steuerung von Aktoren anzubieten. In dieser Arbeit wird eine LED an eine BeagleBone-Platine mit General Purpose I/O(GPIO) angeschlossenen. Das bedeutet, dass damit das installierte Schaltinterface ein Kühl- und Heizsystem steuert.

Das Auswertungssystem, das im einfachsten Fall ein Handy oder eine intelligente Applikation in einem Smartphone ist, wird mittels des aktuellen GSM-Netzes mit dem Gerät Daten austauschen. Die Ausführlichkeit und Spezifikation der verwendeten Komponenten werden im nächsten Abschnitt vorgestellt.

Die folgende Abbildung 2.1 zeigt das konkrete Blockschaltbild des ganzen Systems.

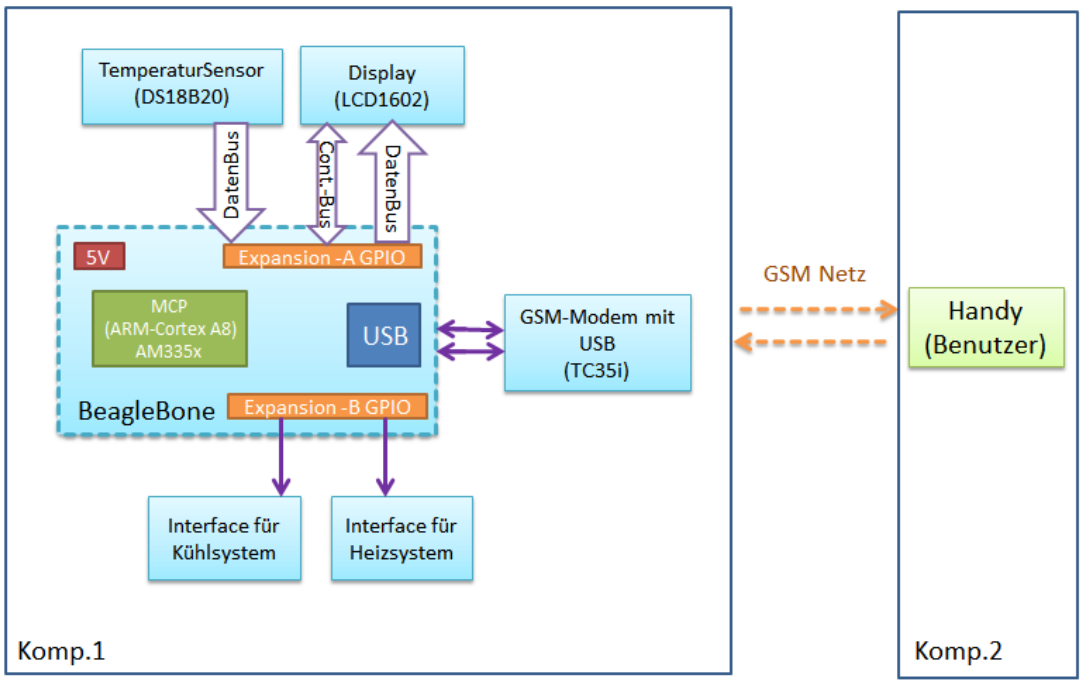


Abbildung 2.1: Blockschaltbild des Systems

2.2 Verwendete Komponenten

2.2.1 ARM Prozessor Platine „BeagleBone“

BeagleBone ist ein Entwicklerboard, das vom Chiphersteller Texas Instrument (TI)[3] hergestellt wird. Es ist ein preiswertes, hoch modernes, kompaktes Board in der Größe einer EC-Karte. Das Herzstück des BeagleBone ist ein preiswerter Sitara Am335x Chip von TI, der eine ARM Cortex A8 Struktur sowie 256 MB DDR2-RAM besitzt und mit 720MHZ getaktet ist. Abbildung 2.2 zeigt die Vorderseite einer BeagleBone-Platine.

Abbildung 2.2: BeagleBone-Platine

Das BeagleBone kann via USB oder Ethernet mit anderen Geräten kommunizieren. Ein Slot für MicroSD-Karten bietet Raum für Speicherkarten, und standardisierte 3,3V I/Os liefern zahlreiche Erweiterungsmöglichkeiten. Das Board wird mit einer 4GB fassenden MicroSD-Karte ausgeliefert, auf der bereits die Linux-Debian gespeichert wird. Nachfolgend ist ein Auszug der Spezifikationen des Boards aufgeführt. [6, Seite 17]

- Processor
  - 720MHz super-scalar ARM Cortex-A8 (armv7a)
  - 3D graphics accelerator
  - ARM Cortex-M3 for power management
  - 2x Programmable Realtime Unit 32-bit RISC CPUs
- Connectivity
  - USB client: power, debug and device
  - USB host
  - Ethernet
  - 2x46 pin headers
  - 2xI2C, 5xUART, I2S, SPI, CAN, 66x3.3V GPIO, 7xADC
- Software
  - 4GB microSD card with Debian system

2.2.2 Temperatursensor DS18B20

Der Sensor DS18B20 ist ein digitaler Temperatursensor, der von der Firma Maxim [4] (ehem. Dallas) hergestellt wird. Auf der Innenseite gibt es einen A/D-Wandler. Dabei handelt es sich um direktwandelnde Sensoren, d. h. die Temperatur wird ohne Umweg über eine analoge Zwischengröße (Spannung oder Strom) in ein digitales Signal überführt.

Die Datenkommunikation erfolgt über einen 1-Wire-Bus, wodurch man am Mikrocontroller mit nur einem einzigen I/O-Pin auskommen kann. Außerdem verfügt der Sensor über die

parasitäre Stromversorgung, d. h., man braucht für die Daten- und Stromversorgung zusammen drei Leitungen. Beim DS18B20 sind Auflösungen von 9, 10, 11 und 12 Bits konfigurierbar. Je kleiner die Auflösung, desto kürzer ist die Messungszeit. Der DS18S20 hat eine feste Auflösung von 12 Bits. In dieser Arbeit wurden 12 Bits verwendet. Er hat einen Messumfang von -55 Grad Celsius bis +120 Grad Celsius, und dessen Genauigkeit kann ±0.5°C erreichen. Der Interfaceaufbau des DS18B20 und die Kommunikation mit dem 1-Wire Bus werden unter Abschnitt 3.1 beschrieben.

2.2.3 Liquid Crystal Display 1602 (LCD 1602)

Das LCD 1602 ist eine Charakter-Flüssigkristallanzeige, auf welcher 32 Schriftzeichen gleichzeitig dargestellt werden können, 1602 bedeutet 16 Reihen und 2 Zeilen.

Es ist ein spezielles Dot-Matrix-Flüssigkristall-Modul, mit dem die Buchstaben, Zahlen und Symbole gut darauf angezeigt werden. Es besteht aus mehreren 5×7 oder 5×11 Bit-Dot-Matrix Zeichen. Auf jedem Matrixzeichen kann ein Charakter angezeigt werden. Es gibt Intervalle zwischen jedem zweiten Matrixzeichen und auch Intervalle zwischen zwei Zeilen. Deswegen ist es nicht sehr geeignet für die Anzeige einer Grafik. Um die Zeit und das Datum darstellen zu können, spielt die Anzeige einer Grafik in dieser Arbeit keine Rolle. Wegen der maximalen Ausgangsspannung der BeagleBone-Platine wird das 3,3-Volt LCD1602 ausgewählt. Nachfolgend werden die Abbildung eines LCD1602 und dessen entsprechende Spezifikationen aufgeführt. [8, Seite 3]

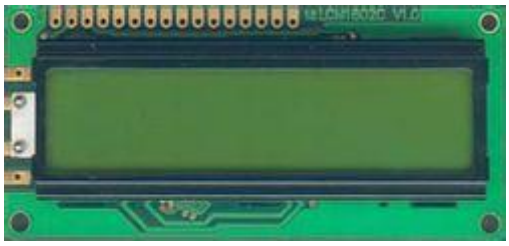


Abbildung 2.3: Liquid Crystal Display 1602

- 5×8 dots with cursor
- 16 characters ×2lines display
- 4-bit or 8-bit MPU interfaces
- Built-in controller(ST7066)
- Display Mode & Backlight Variations
- ROHS Compliant

2.2.4 GSM-Modem TC35i Terminal

Das GSM-Modem TC35i ist von der Firma Siemens hergestellt worden. Mit seinen industriellen Standardschnittstellen und einem integrierten SIM-Kartenleser ist es ein idealer Dual Band GSM-Terminal für den universellen Einsatz im heutigen Sprach- und Datenkommunikationsbereich. Damit eröffnet es die Realisierung neuer Applikationen in den Bereichen wie Telemetrie, Telematik und Telefonie. Abbildung 2.4 zeigt ein GSM-Modem mit RS-232-Schnittstelle.

Abbildung 2.4: GSM-Modem TC35i mit RS-232 Schnittstellen

Neben einer RJ12 Buchse zum Anschluss der Spannungsversorgung 5,5V ist ein Antennenanschluss und eine RS-232 Schnittstelle mit dem Max232-Chip[5] vorhanden. Gleichzeitig unterstützt TC35i Standard AT-Befehle, siehe Abschnitt 5.2. Damit kann das BeagleBone solche AT-Befehle via ein USB zu seriellem Konverter TC35i senden. Die technischen Daten sind nachfolgend aufgeführt.

- Dual-Band GSM 900/1800 MHz
- Compliant to GSM phase 2/2+
- Output power:
    - Class 4 (2 W) for EGSM900
    - Class 1 (1 W) for GSM1800
- Control via AT commands
- Supply voltage range: 3.3 ... 4.8V
- Power consumption:
    - Power down 50 µA
    - Sleep mode 3.5 mA
    - Speech mode (average) 300 m

2.2.5 Peripherie

Damit die oben genannten Elemente mit BeagleBone verbunden werden können, sind noch einige zusätzliche Elemente notwendig. Außer der Kabel zur Verbindung und der Lochrasterplatine zur Feststellung werden noch ein Potentiometer, Widerstände, mit denen der Strom begrenzt wird und ein LED, mit dem man das Alarmsignal deutlich sehen kann, benötigt. Bei der Verbindung zwischen BeagleBone und GSM-Modem ist ein USB zu serieller Konverter notwendig. Und damit das ganze System auch noch gut aussieht, wurde auch ein schwarzes Gehäuse verwendet.

2.3 Schaltplan des gesamten Systems (Anhang A)

3 Implementierung der Temperaturmessung

In diesem Kapitel werden der 1-Wire Bus, welcher zwischen dem BeagleBone und dem Temperatursensor verwendet wird, und die Implementierung der Temperaturmessung erläutert. Nachfolgend werden die aufgetretenen Probleme und die entsprechenden Lösungsansätze vorgestellt.

3.1 Ankopplung der Peripherie an die Prozessorplatine

3.1.1 1-Wire-Bus

1-Wire, auch One-Wire-Bus genannt, ist ein digitaler, serieller Bus des Herstellers Maxim (ehem. Dallas), der mit einer Datenader (DQ) und einer Masseleitung (GND) und Stromversorgung (V_{DD} optimal) auskommt. Die Bezeichnung 1-Wire leitet sich daraus ab, dass zu der ohnehin vorhandenen Masseleitung nur eine weitere Ader für die gesamte Buskommunikation erforderlich ist. Gelegentlich wird 1-Wire durch den Hersteller auch als „Single-Wire Serial Interface“ bezeichnet. Das System des 1-Wire Bus ist in der Lage, mit 1-Wire ein oder mehrere Devices zu steuern. [4]

3.1.2 Interface von DS18B20

In Kapitel 2.2 wurde ein kurzer Einblick über den Temperatursensor DS18B20 gegeben. Die Abbildung 3.1 zeigt dessen Interface.

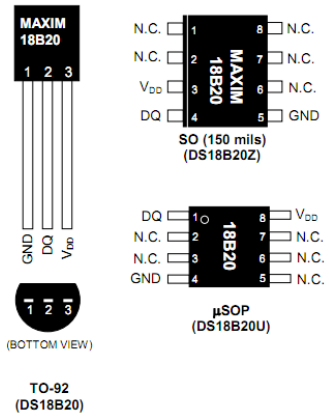


Abbildung 3.1: Interface des Temperatursensors DS18B20

Das DS18B20 benutzt das 1-Wire Protokoll, d. h., es können nahezu beliebig viele Sensoren auf einer einzelnen 1-Wire Leitung mit dem BeagleBone angeschlossen werden, da jeder Sensor mit 64-Bit-ID in ROM adressiert wird. Abbildung 3.2 zeigt schematisch den Kommunikationsmechanismus eines 1-Wire Bus Systems, mit dem das BeagleBone drei DS18B20

steuern kann. In dieser Arbeit wurde nur ein Temperatursensor in einfachster Ausführung verwendet. Die Datenader (DQ) wurde mit General Purpose Input/Output Port 8_6(P8_6) [6.Seite 58] der BeagleBone-Platine angeschlossen. Und die Masseleitung (GND) wurde mit P8_1 sowie die Stromversorgung (V_{DD}) mit P9_3 verbunden.

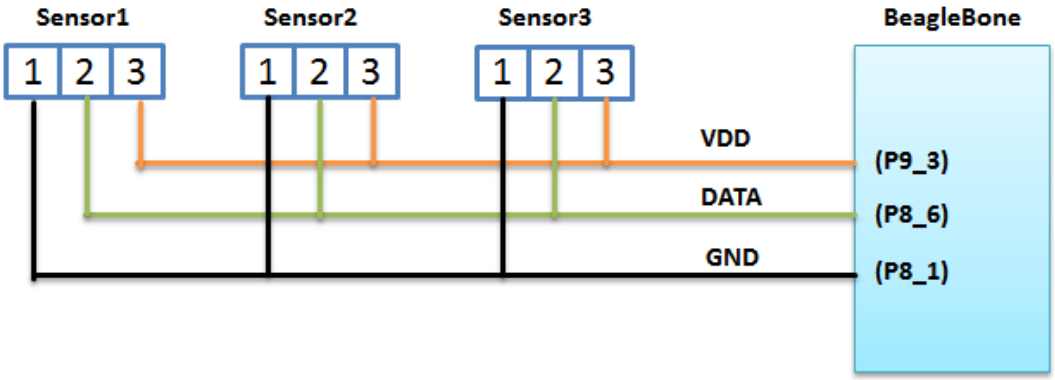


Abbildung 3.2: Schematische Verbindung zw. BeagleBone und drei Temperatursensoren

3.2 Kommunikationsprozess

Der Kommunikationsprozess wird nur mithilfe einer Ader verwirklicht, das heißt, dass das Kontrollsignal und das Datensignal sich durch unterschiedliche Zeiten unterscheiden. Dabei sind drei Zeitslots für die Kommunikation mit dem Sensor notwendig:

1. Initialisierungszeitslot
2. Schreibzeitslot
3. Lesezeitslot

Dazu wird in den anschließenden Abschnitten 3.2.1 bis 3.2.2 noch genauer eingegangen. Nach dem Protokoll des 1-Wires ist die Kommunikationssequenz für den Zugriff auf die DS18B20 wie folgt:

Schritt 1. Initialisierung

Schritt 2. ROM Befehl

Schritt 3. Funktionsbefehl

Initialisierung

Jede Kommunikation mit dem DS18B20 Sensor beginnt mit einer Initialisierungssequenz, die einen Rücksetzimpuls des Masters und einen vorhandenen Impuls des DS18B20 umfasst. Wenn der Master einen vorhandenen Impuls von DS18B20 bekommt, bedeutet es, dass der Temperatursensor bereit steht, um die Daten zu senden oder zu empfangen.

ROM Befehl

Nach dem Detektieren eines vorhandenen Impulses auf dem Bus wird ein ROM-Befehl durchgeführt. Diese Befehle arbeiten mit der einzigartigen 64-Bit-ID jedes Slave-Geräts, das in ROM gespeichert wird. Mit diesen ROM-Befehlen bestimmt der Master, wie viele und welche Arten von Slaves auf dem Bus sind. Es gibt insgesamt fünf ROM-Befehle und jeder Befehl ist 8 Bits lang. Sie heißen SEARCH ROM, READ ROM, MATCH ROM, SKIP ROM und ALARM SERCH [7, Seite 11]. In dieser Arbeit wurde nur ein Sensor verwendet, deswegen ist nur der Befehl SKIP ROM notwendig. Es bedeutet, dass der ROM-Befehl vernachlässigt wird. Nach dem SKIP ROM ist es nicht mehr notwendig, die 64-Bit-ID anzubieten.

Funktionsbefehl

Nachdem der Master einen ROM-Befehl (SKIP ROM) an DS18B20 gesendet hat, schickt dieser anschließend einen Funktionsbefehl. Mit solchen Funktionsbefehlen kann das Master-gerät die Register des Temperatursensors, wo der digitale Temperaturwert gespeichert wird, lesen und schreiben, die Temperaturwerte konvertieren und den Modus der Stromversorgung erkennen.

Die verwendeten Funktionsbefehle und der ROM-Befehl mit den entsprechenden hexadezimalen Zahlen sind in Tabelle 3-1 dargestellt.

Tabelle 3-1: Verwendete Funktionsbefehle und ROM-Befehl

Befehl	HEX	Kommentar	Typ
SKIP ROM	CCh	ROM-Befehl vernachlässigt, ohne 64-Bit-ID anbieten	ROM-Befehl
Temperatur konvertieren	44h	Temperaturwert wird digital konvertiert und in dem Register gespeichert	Funktionsbefehl
Register auslesen	BEh	Temperaturwert von RAM ausgelesen	Funktionsbefehl
Register aufschreiben	4Eh	Daten auf RAM aufgeschrieben	Funktionsbefehl

3.2.1 Initialisierungszeitslot

Der Initialisierungsprozess ist der erste Schritt der Kommunikation. Dessen Zeitslot ist in Abbildung 3.2 dargestellt. Während der Initialisierungssequenz stellt der Master den 1-Wire Bus auf Low, und die Verschiebungszeit dauert mindestens 480µs(T_X). Dann gibt der Master den Bus frei und geht in den Empfangsmodus (R_X). Wenn der Bus freigegeben ist, stellt der Widerstand den 1-Wire-Bus hoch, und der DS18B20 erkennt dieses ansteigende Signal. Nun wartet der Master noch 15µs bis 60µs. Danach überträgt DS18B20 einen Präsenzimpuls, indem er den 1-Wire-Bus auf niedrige 60µs bis 240µs stellt. Wenn der Master den Präsenzimpuls detektiert, erfolgt die Initialisierung. [7, Seite 15]

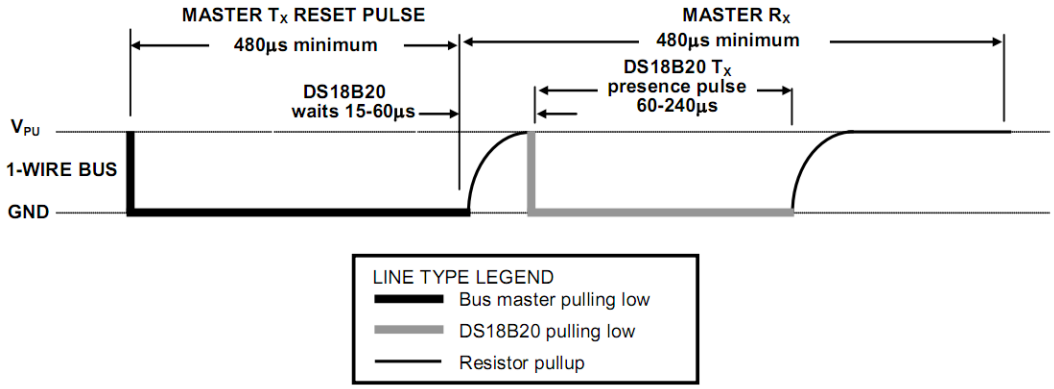


Abbildung 3.3: Initialisierungszeitslot

3.2.2 Schreibzeitslot

Der Master schreibt Daten auf den Temperatursensor des Schreibzeitslots und liest Daten aus dem DS18B20 des Lesezeitslots, siehe Abschnitt 3.1.3. Ein-Bit-Daten werden mithilfe des 1-Wire-Busses pro Zeitslot übertragen.

Es gibt zwei Operationsarten von Schreibzeitslots: „Schreib 1“ und „Schreib 0“. Der Master benutzt einen „Schreib 1“, um Logik 1 an DS18B20 zu schreiben und gegebenenfalls einen „Schreib 0“, um Logik 0 zu schreiben. Alle Schreibzeitslots müssen mindestens 60µs dauern. Übrigens muss noch eine minimale 1µs Wiederholungszeit zwischen je zwei Zeitslots vorhanden sein. Beide Operationsarten werden durch ein kurzes niedriges Signal initialisiert. Um einen „Schreib 1“ zu erzeugen muss der Master innerhalb von 15µs den Bus freigeben, nachdem dieser auf Low gezogen worden ist. Wenn der Bus frei ist, stellt der Pullup-Widerstand den Status auf High. Um einen „Schreib 0“ zu erzeugen, hält der Master den Busstatus auf Low für eine bestimmte Zeit, welche mehr als 60µs beträgt, nachdem dieser

auf Low gezogen worden ist. Der DS18B20 detektiert den 1-wire-Bus. Wenn ein Logik 1 im Bus detektiert wird, bedeutet es, dass ein Logik 1 auf DS18B20 geschrieben wird. Und gegebenenfalls bedeutet Logik 0, eine 0 auf den Temperatursensor zu schreiben. [7, Seite 16]

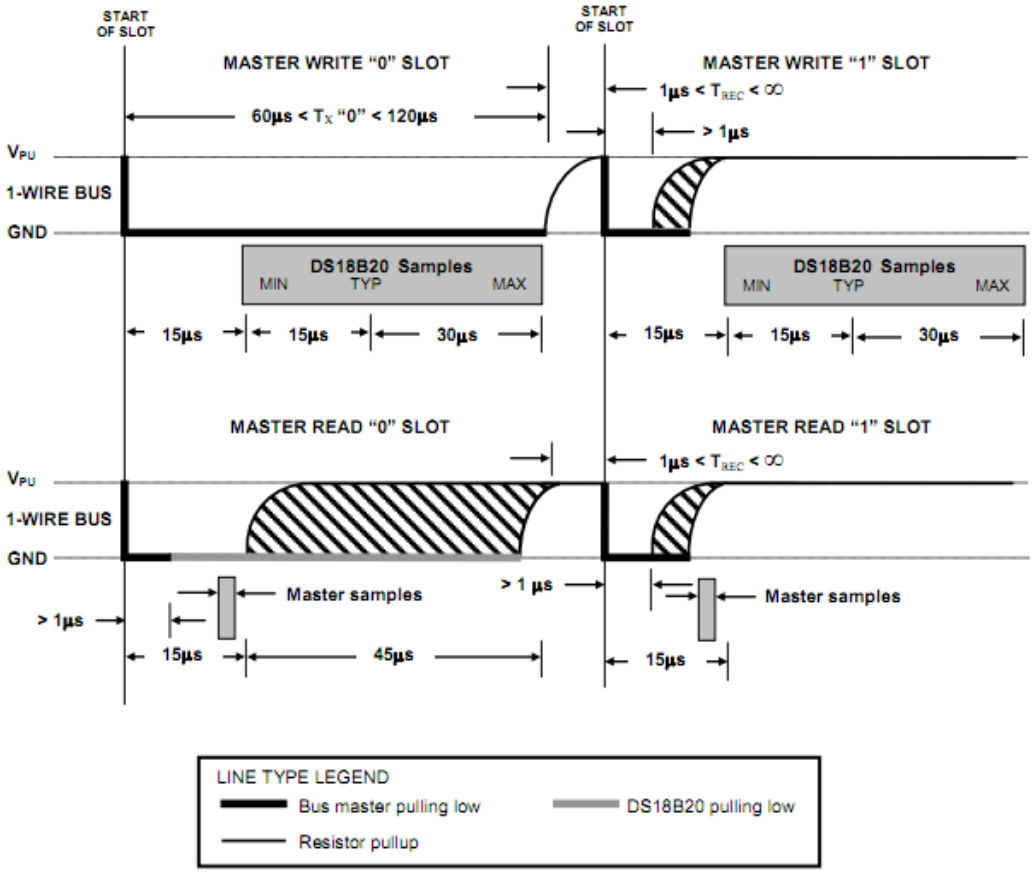


Abbildung 3.4: Schreibzeitslot und Lesezeitslot

3.2.3 Lesezeitslot

Im Vergleich zum Schreibzeitslot besteht der Lesezeitslot auch aus zwei Operationen, „Lese 1“ und „Lese 0“, die das Auslesen der zugehörigen Logik 1 und Logik 0 aus dem Temperatursensor bedeuten. Jeder Lesezeitslot dauert eine minimale Zeit von 60µs. Zwischen zwei Lesezeitslots muss eine Übergangszeit vorhanden sein, um den Bus wiederzustellen. Wie in Abbildung 3.4 dargestellt wurde, beginnt der Lesezeitslot mit einem kurzen niedrigen Initialisierungssignal, welches mindesten 1µs beträgt. Nach der Freigabe des Busses wird der DS18B20 innerhalb von 45µs Logik 1 und Logik 0 auf den Bus übertragen. Gleichzeitig muss der Master in den Eingangsmodus wechseln und die logischen Signale detektieren. [7, Seite 16]

3.2.4 Beispielsweise Verwendung

Die Kommunikationssequenzen wurden bereits vorgegeben und wie Logik „0“ und „1“ aufgeschrieben und auslesen wird, ist auch geklärt. Am nachfolgenden Beispiel wird veranschaulicht, wie die Temperatur in dieser Arbeit konvertiert und ausgelesen wird.

Der BeagleBone benötigt zwei Operationen, um die Temperaturwerte zu erhalten.

1. Konvertierung der Temperatur

   Zuvor wurde erwähnt, dass jede Operation drei Schritte haben muss, gleichfalls werden hier auch drei Schritte benötigt. Der erste Schritt ist die Initialisierung. Anschließend schreibt das BeagleBone den ROM-Befehl SKIP ROM(CCH) auf DS18B20. Nachfolgend wird der Funktionsbefehl (44H) aufgeschrieben, welcher die Konvertierung der Temperatur bedeutet. Die Abbildung 3.5 zeigt den Status des Buses für die Operationen.

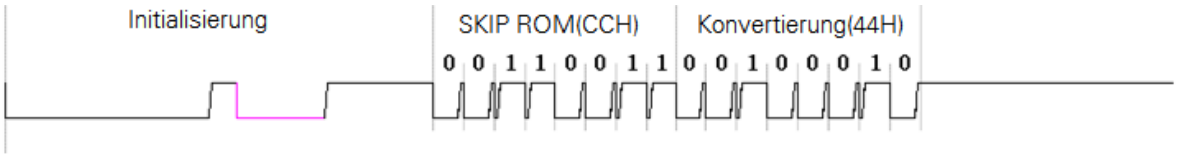


Abbildung 3.5: Operation der Temperaturkonvertierung

2. Auslesen des digitalen Wertes. Ebenso wie in Punkt 1, erfolgt die Operation auch in drei Schritten. Der Busstatus für diese Operation ist in Abbildung 3.6 dargestellt. Die roten Signale in Abb.3.5 und 3.6 werden vom Temperatursensor DS18B20 gesendet.

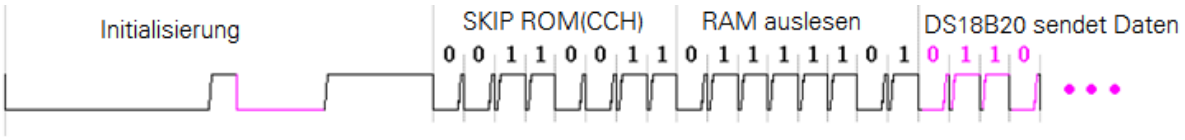


Abbildung 3.6: Operation des Auslesens des digitalen Wertes

3.3 Aufgetretene Probleme und Lösungsansätze

Für die eigentlichen Tests sind die oben genannten Zeitslots nicht geeignet. Da auf der „BeagleBone“-Platine ein Linux System läuft, und das entwickelte Programm auf der höchsten Applikationsebene arbeitet, könnte es durchaus sein, dass die laufende Applikation durch einen Interrupt oder andere Prozesse abgebrochen wird, welche in der tieferen Ebe-

ne, wie z. B. die Treiberebene oder Kernelebene arbeiten. In diesem Fall sind die Systemfunktionen *usleep()* und *nanosleep()* ungenau. Die folgende Abbildung zeigt die Ungenauigkeit dieser beiden Funktionen im Linux System auf der „BeagleBone"-Platine.

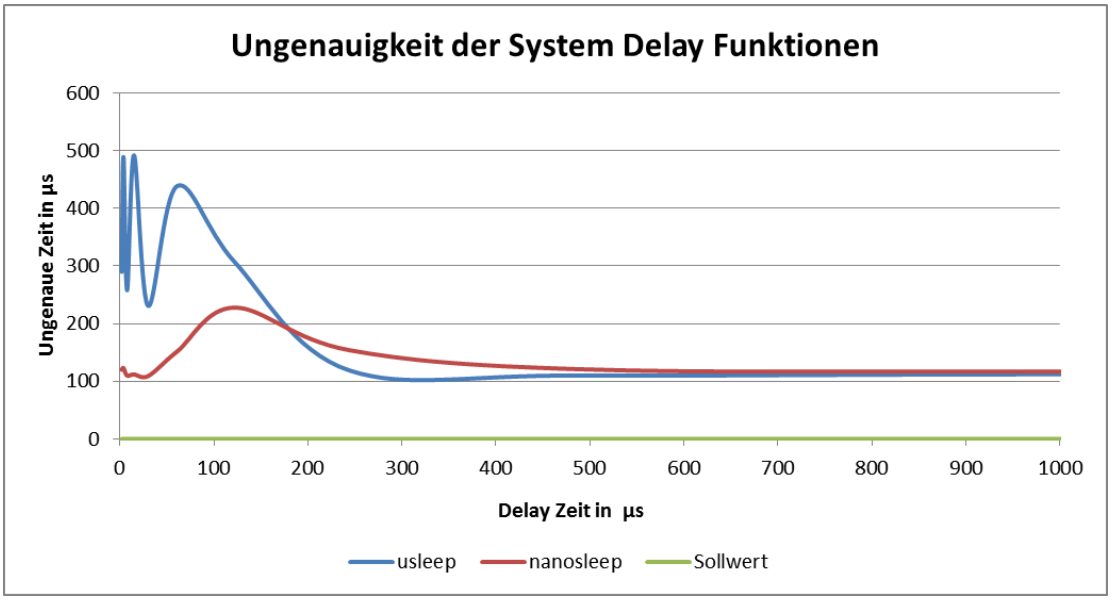


Abbildung 3.7: Ungenauigkeit der System Delay Funktionen

Anhand der Abbildung wird es deutlich, dass weder die Funktion *usleep()* noch die Funktion *nanosleep()* das Verschiebungsniveau 100µs erreichen kann. Gegebenenfalls benötigt die Kommunikation des DS18B20 mindestens ein Verschiebungsniveau von 15µs. Deswegen wurden diese vom Linux System angebotenen Funktionen *usleep()* und *nanosleep()* aufgebeben und zwei weitere benutzerdefinierte Funktionen *void delay(void)* und *unsigned int uDelay(unsigned int delaytime)* entwickelt.

Funktion uDelay ()

```
unsigned int uDelay(unsigned int delayTime)
{
   static struct timeval _tstart, _tend;  //  static struct timezone tz;
   double t,t1,t2;
   int i;
   gettimeofday(&_tstart, NULL);
   t1 =  (double)_tstart.tv_sec + (double)_tstart.tv_usec/(1000*1000);
   for(i = 0;;i++)
   {

     gettimeofday(&_tend, NULL);
     t2 =  (double)_tend.tv_sec + (double)_tend.tv_usec/(1000*1000);
     t = t2 - t1;
     if(t > (double)delayTime/(1000*1000))
         {

          return 0;
         }
     }
}
```

Das Prinzip der Funktion *Udelay()* ist nicht sehr schwer. Mithilfe der von dem Linux System angebotenen Funktion *gettimeofday()* wird eine Zeit erhalten, die aus einer Sekunde und einer Nanosekunde des Systems besteht. Dann geht das Programm zu einer Schleife, in der die Funktion *gettimeofday()* noch einmal verwendet wird, und ein neuer Zeitparameter mit Sekunde und Nanosekunde wird gemessen. Die Abweichung wird mit dem vom Benutzer definierten Parameter verglichen. Sobald die Abweichung größer als der Parameter ist, wird die Schleife unterbrochen, und die Verschiebungszeit ist erfolgt.

Funktion delay_us ()

```
void uDelay(unsigned int delayTime)
{
    for(i=0,i++,i<delayTime)
    {
      asm("nop");
    }
}
```

In der Funktion *uDelay()* werden einige Assembler verwendet, damit die CPU des BeagleBone leerläuft. Damit wird die Zeit verschoben. Gemäß der Betriebsfrequenz der CPU benötigt theoretisch jeder leere Assembler eine Verschiebungszeit von 0,0001s. Beim Versuch waren die beiden definierten Funktionen offensichtlich genauer als *usleep()* und *nanosleep()*. Aber sie konnten noch nicht die Verschiebungszeit von 15us erreichen, um mit dem Temperatursensor DS18B20 kommunizieren zu können.

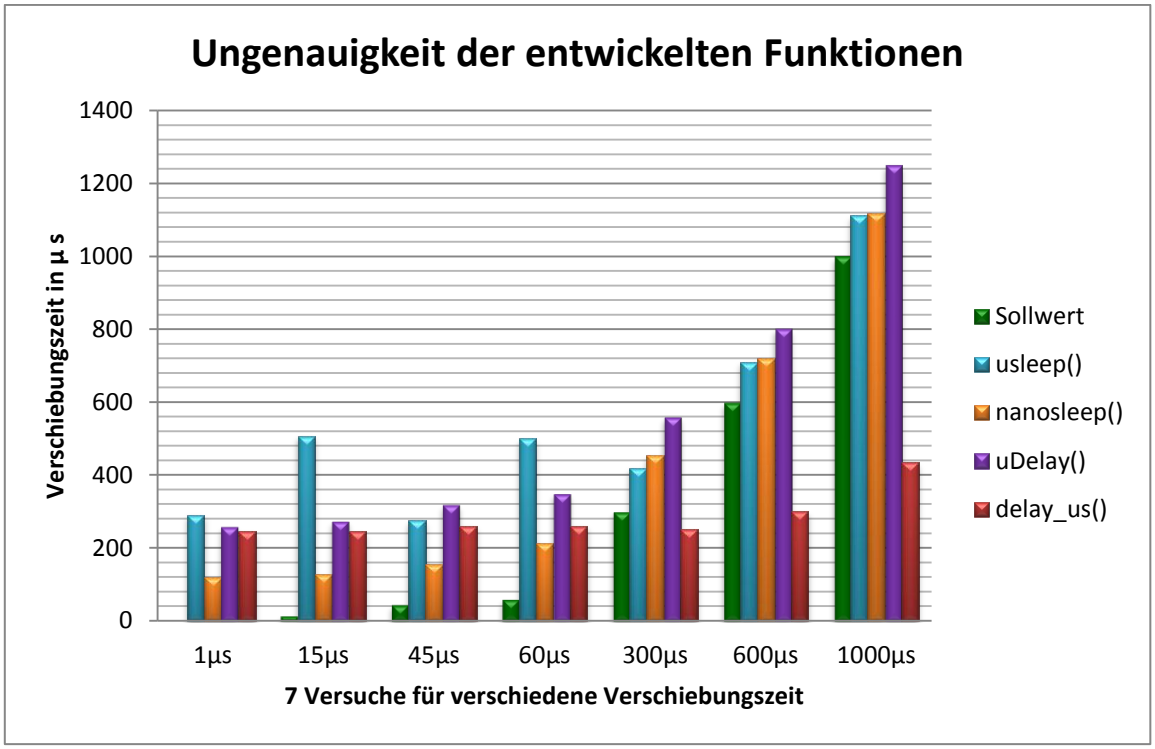


Abbildung 3.8: Ungenauigkeit der entwickelten Funktionen

Lösungsansatz

Durch die hintereinander erfolgten Versuche wurde deutlich, dass die Implementierung der Temperaturmessung auf der Ebene der Applikation schwierig zu realisieren ist. Ein Lösungsansatz dafür könnte sein, dass ein Treiber in der tieferen Ebene benutzt wird. Das heißt, im Linux System Debian arbeitet der Temperatursensor als ein Device in der BeagleBone Platine, und der Treiber bietet das Kommunikationsprotokoll an. Im Vergleich zu den Applikationen hat dieser Treiber nicht nur eine höhere Durchführungspriorität, sondern arbeitet auch mit einem Hardware-Zeitinterrupt. Sobald die Verschiebungszeit erreicht ist, wird sofort ein Interrupt durchgeführt. Während dieser Zeit arbeitet die CPU überhaupt nicht.

Die Temperaturmessung kann in diesem Fall durch ein Kommando auf der Konsole des Systems erfolgen, wie folgende Abbildung deutlich macht.

```
root@beaglebone:/# cat /sys/bus/w1/devices/28-00000274221f/w1_slave
d8 01 4b 46 7f ff 08 10 6b : crc=6b YES
d8 01 4b 46 7f ff 08 10 6b t=29500
root@beaglebone:/# cat /sys/bus/w1/devices/28-00000274221f/w1_slave
d6 01 4b 46 7f ff 0a 10 43 : crc=43 YES
d6 01 4b 46 7f ff 0a 10 43 t=29375
root@beaglebone:/#
```

Abbildung 3.9: Die vom Kommando gemessene Temperatur auf der Konsole

28-000003b78609 ist die einzige Gerätadresse (64-Bit-ID) jedes Temperatursensors. Die neunstelligen Hexadezimalzahlen stellen eine derzeitige Temperatur dar, und T=29375 ist die umgewandelte Dezimalzahl, welche 29,500 Grad Celsius beträgt.

Mit einem weiteren Kommando:

cat /sys/bus/w1/devices/28-000003b78609/w1_slave|tail -n1|awk -F= '{print $2}'

wird die Dezimalzahl verdeutlicht. Dazu erhält man einen String-Wert. Mithilfe der entwickelten Funktion *float ReadTemp(int dev,char Temp[])* [1] wird dieser String-Wert in einen Float-Wert umgewandelt und abgespeichert, um ihn dann weiter zu verwenden.[15, Seite 145]

[1] Siehe Anhang B von Code, die in CD gespeichert wird

4 Implementierung der Datenanzeige

Nachfolgend wird die Implementierung der Datenanzeige in diesem Kapitel vorgestellt. Außerdem wird die Realisierung der Programmierung erläutert.

4.1 Interface des LCD1602

Liquid Crystal Display 1602 hat insgesamt 16 Pins. Sie werden folgendermaßen aufgeteilt: Data Bus Pins, Control Pins und Power Supply Pins. In der Tabelle 4-1 werden das Symbol und die entsprechende Funktion an jedem Pin dargestellt. [8, Seite 5]

Tabelle 4-1: Fußbeschreibung des Interfaces

Pin Nr.	Symbol	Kommentar	Verbindung mit BeagleBone
1	V_{SS}	GND	GND
2	V_{DD}	+5V	+5V
3	V_0	Contrast	+5V (mit Potentiometer)
4	RS	Data/Instruction	Pin P8_3
5	R/W	Read/Write	GND
6	E	Enable	Pin P8_4
7~10	DB0~DB3	Data Bit 0~Data Bit 3	nicht angeschlossen
11	DB4	Data Bit 4	Pin P8_5
12	DB5	Data Bit 5	Pin P8_11
13	DB6	Data Bit 6	Pin P8_12
14	DB7	Data Bit 7	Pin P8_14
15	LED+	Power supply for BKL	+5V
16	LED-	Power supply for BKL	GND

Pin 1 und 2 sind Anschlüsse für die Stromversorgung. Sie müssen mit der negativen und positiven Schiene einer 3,3-Volt-Stromversorgung verbunden werden. Um eine +3.3V Stromversorgung zu erhalten, kann das Pin 1 mit P9_3 oder P9_4 [6, Seite 59] der BeagleBone-Platine verbunden werden, welche die externe Systemstromversorgung +3.3V anbietet. Und der Pin 2 kann mit irgendeiner Erdung in der BeagleBone-Platine verbunden werden. (In dieser Arbeit ist Pin 1 mit P8_1 angeschlossen und Pin 2 mit P9_3 angeschlossen)
Pin 4, 5 und 6 sind die Steuer-Pins des LCD. Dazu wird folgend genauer eingegangen.
Pin 7 bis 14 sind die Daten-Pins des LCD. Pin 7 ist das Least Significant Bit(LSB) und Pin 14 ist das Most Significant Bit(MSB) der Dateneingänge. Wenn eine Zahl oder ein Buchstabe auf dem Display angezeigt werden soll, müssen die entsprechenden „Codes“ für diesen Charakter auf diesen Stiften eingegeben werden. Diese Stifte werden auch für die Gewäh-

rung bestimmter Kontrollbefehle, wie z. B. das Löschen des Displays oder den Cursor an eine andere Stelle setzen, verwendet.

Pin 15 und 16: Die meisten LCDs haben eine Hintergrundbeleuchtung. Eine Hintergrundbeleuchtung ist ein Licht in der LCD-Anzeige, mit der man die Charaktere auf dem Bildschirm besser sieht. Hintergrundbeleuchtung ist nichts anderes als eine LED, also muss ein Widerstand in Reihe mit ihm verbunden werden, um den Strom zu begrenzen.

Abbildung 4.1 zeigt die Pins von LCD1602.

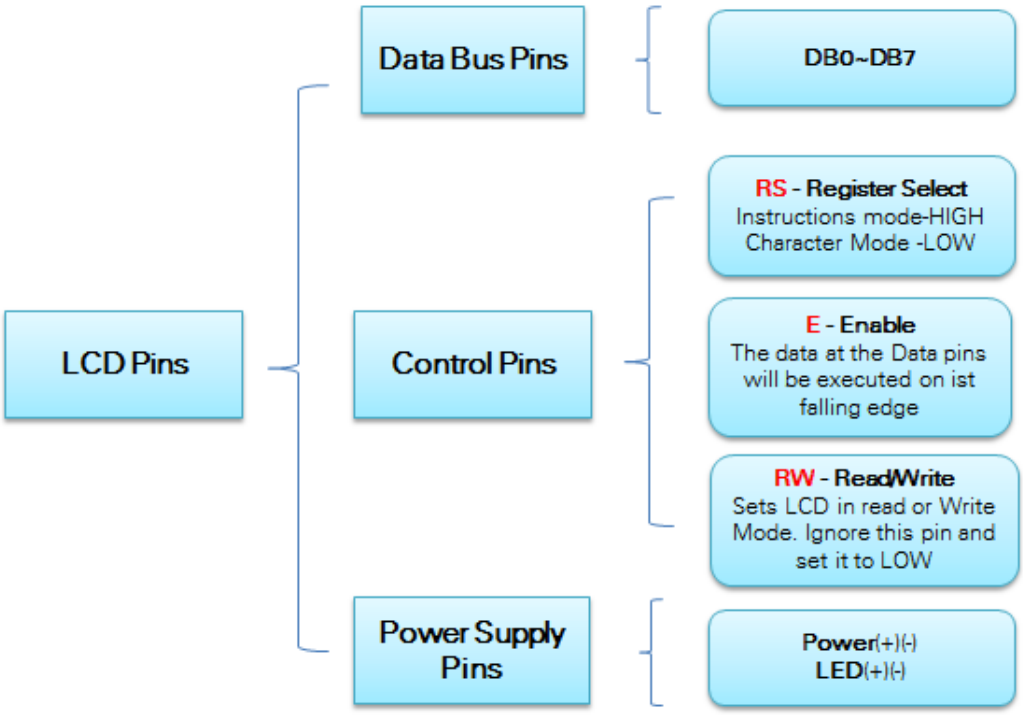


Abbildung 4.1: Verschiedene Pins für LCD1602

Der Pin RS

Das LCD hat grundsätzlich zwei Betriebsarten: den Instruction-Modus und den Character-Modus. In Abhängigkeit vom Status dieses Stiftes werden die Daten an den Datenanschlussstift entweder als Anweisung oder als Zeichendaten behandelt. Sollen einige Zeichen angezeigt werden, ist der Charakter-Modus aktiviert, andersherum über Senden einer Anweisung ist der Instruction-Modus eingestellt.

Der Pin Enable

Der Pin Enable hat eine sehr einfache Funktion. Er ist nur der Takteingang für das LCD, die Anweisungen oder die Zeichendaten werden mit dem abfallenden Signal verarbeitet.

Der Pin R/W

Normalerweise wird das LCD verwendet, um die Dinge auf dem Bildschirm anzuzeigen. Doch in einigen seltenen Fällen ist es notwendig, um das zu lesen, was das LCD anzeigt. In

solchen Fällen wird der Read/Write-Stift verwendet. Allerdings wird diese Funktion in dieser Arbeit nicht erläutert, für praktische Zwecke ist der R/W dauerhaft auf GND angeschlossen worden.

Abbildung 4.2 zeigt die Interfaceverbindung zwischen der BeagleBone-Platine und dem LCD. Die Verbindungen mit verschiedenen Farben bedeuten die unterschiedlichen Busarten: der Data Bus, der Control Bus und der Power Bus. Bei der Verbindung mit dem Data-Bus gibt es zwei Arten für den Interface Modus, nämlich den 4-Bit Modus und den 8-Bit Modus. Der 4-Bit Modus ermöglicht es, dass das LCD und BeagleBone mit vier Verbindungskabeln die 8-Bit-Daten übertragen können. Das bedeutet, LSB und MSB von 8-Bit-Daten werden getrennt mitgeteilt. In dieser Arbeit wurde der 4-Bit Mode Interface verwendet. In diesem Fall werden 4 Ports von BeagleBone gespart. Diese Modi kann man mithilfe von Kommandos einstellen. Dazu wird anschließend im Abschnitt 4.2 noch genauer eingegangen.

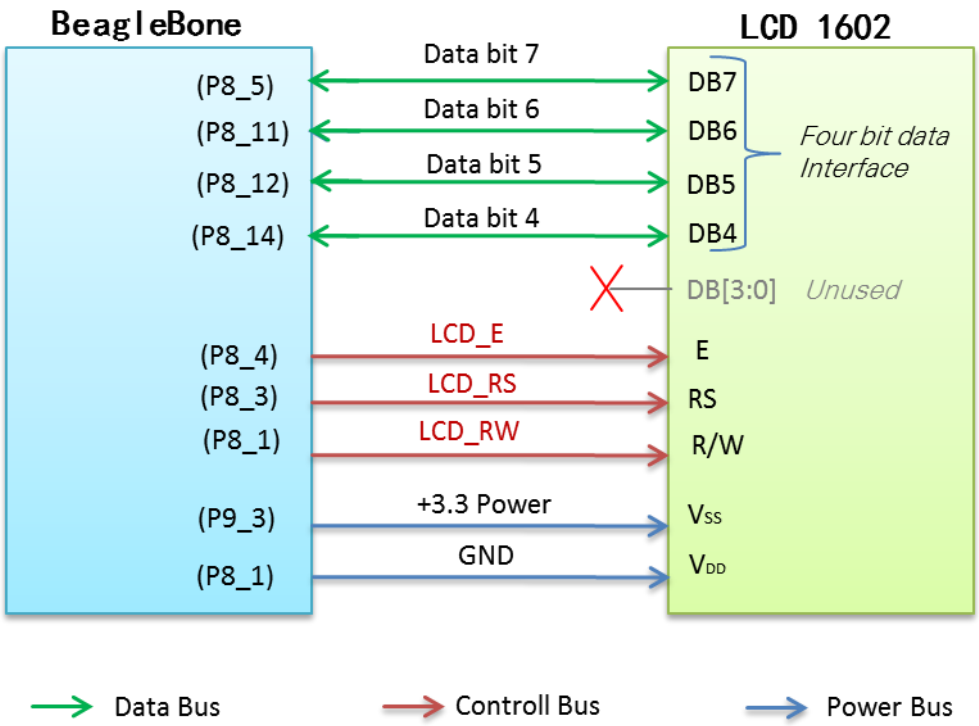


Abbildung 4.2: Interfaceverbindung zwischen BeagleBone und LCD

4.2 Steuerkommandos

Damit das LCD arbeitet, muss das BeagleBone Kommandos darauf schreiben. Mit solchen Kommandos wird deutlich, mit welchem Interface-Modus und mit welchem Display-Modus das LCD arbeiten wird, sowie auch die Art, wie auf dem LCD angezeigt und gelöscht wird. Im Folgenden sind die wichtigen Kommandoeinstellungen für das LCD aufgeführt, die in dieser Arbeit verwendet wurden. [8, Seite 10]

Interface Mode einstellen

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	-	-

Wenn DL =„High“ ist, wird 8-bit Interface Mode eingestellt, ist DL=„Low“, wird 4-Bit Interface Mode eingestellt. Wenn N=„Low“ ist, bedeutet es, dass 1-line Mode eingesetzt wird, wenn N=„High“ ist, bedeutet es, dass 2-line Mode eingesetzt wird.

Display Mode einstellen

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	D	C	B

D: Display ON/OFFC: Cursor ON/OFFB: Cursor blink ON/OFF
Ist D = „High“, wird der Charakter in RAM geschrieben und auf dem Display angezeigt. Ist D = „Low“, wird der Charakter trotzdem in RAM geschrieben und gespeichert, aber nicht auf dem Display angezeigt. B und C sind die Bitmasken für das Einschalten und das Ausschalten des Cursors und Aufblitzen des Cursors.

Clear Display

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

Mit diesem Kommando werden die Charaktere, die auf dem Display stehen, gelöscht. [8, Seite 11]

4.3 Operationszeitslot

Anschließend wird der Operationszeitslot geschrieben, dadurch kann man deutlich sehen, wie und nach welcher sequenziellen Ordnung die Kommandos mitgeteilt werden. Die Abbildung 4.3 zeigt den Operationszeitslot beim 4-Bit-Form Modus des LCDs. [8, Seite 7]

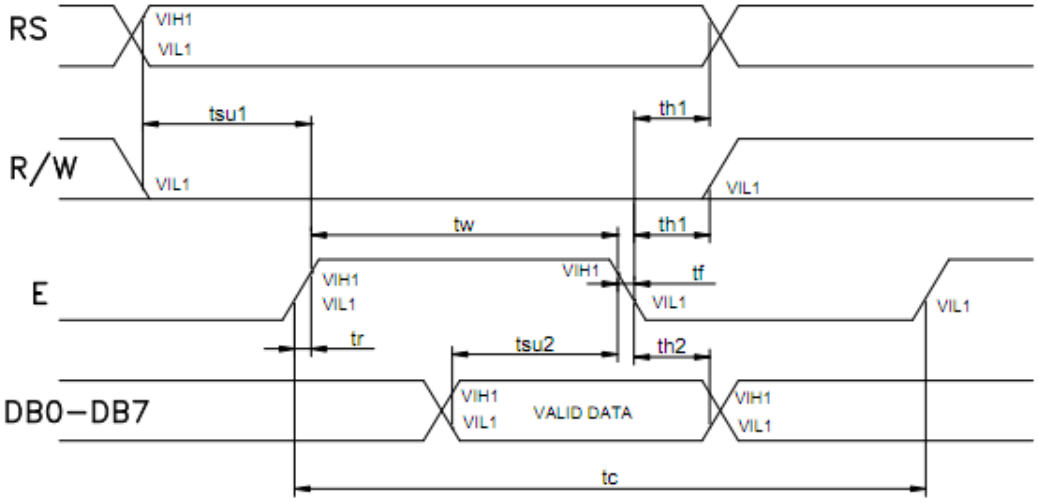


Abbildung 4.3: Diagramm für das Schreibzeitslot

Da alle Befehle und Daten im Rahmen einer 8-Bit-Form vorhanden sind, wird die Operation bei jeder Übertragung eines Befehls oder einer Datei in zwei Übertragungsvorgänge unterteilt. Die höheren 4 Bits werden zuerst mitgeteilt, danach kommen die Low 4 Bits, um die ganzen 8 Bits zu vervollständigen. Bei der Kommandoübertragung des LCDs gibt es keine ernsten Forderungen wie die Kommunikation des Temperatursensors. In diesem Fall spielt nur der Schreibzeitslot eine Rolle. Was ganz wichtig ist, dass die Signale der drei Control Pins müssen vor der Mitteilung eines 8-Bit Befehls oder einer Datei eine minimale gültige Zeit von 25ns haben. Um die Charakterdaten auf dem Display anzuzeigen, ist die Mode-Einstellung davor sehr wichtig. Hierzu wird die Initialisierung des LCDs genannt. Nach dem Kommunikationsprotokoll ist die Initialisierungssequenz wie folgt:

Schritt 1: 4-Bit Interface Mode einstellen

Schritt 2: Einschalten des Displays und Ausschalten des Cursors einstellen

Schritt 3: 2 Line Mode einstellen

Schritt 4: Display bereinigen

Nach der Initialisierung wird der Charakter auf RAM des Displays geschrieben und auf dem Bildschirm angezeigt. Die nachfolgend aufgeführte Tabelle stellt den Operationsprozess und die entsprechenden Binärzahlen dar. [8, Seite 7]

Tabelle 4-2: Operationsprozess und die entsprechenden Binärzahlen

Schritt	Instruktion	Instruktionscode										Kommentar
		R/S	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
1	Interface Mode	0	0	0	0	0	1	0	0	0	0	4-Bit-Bus Set
2	Display Mode	0	0	0	0	0	0	1	1	0	0	Display ON Cursor OFF Cursor blink OFF
3	2 Line Mode	0	0	0	0	1	0	1	0	0	0	5×7 Dot-Matrix for Charac-ter
4	Clear Dis-play	0	0	0	0	0	0	0	0	0	1	Clear Dis-play
	Charakter	Instruktionscode										Kommentar
		R/S	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
5	Write Data	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write Data to RAM

4.4 Realisierung der Programmierung

Das Interface und die Steuerkommandos sind bereits geklärt. Als nächstes werden Codes programmiert, um die Initialisierung und die Anzeige des Charakters zu realisieren. Eine wichtige Funktion ist die Initialisierungsfunktion, mit der das BeagleBone Schritt 1 bis Schritt 4 realisiert werden kann. Ein wichtiger Punkt dieser Funktion ist, dass Low 4 Bits oder High 4 Bits jedes Befehls mit der Bitmaske von R/S und Enable kombiniert werden. Danach wird die neu entstehende 6-Bit Datei auf entsprechende Ports geschrieben.

```
void initialize_Screen(struct gpioID enabled_gpio[],int selectedPins[])
{
        int delay=0;
        int nbr_selectedPins=6;
        unsigned int data_to_write;

        pinMode_multiple(enabled_gpio,selectedPins,nbr_selectedPins,"out");

        //E RS DB4 DB5 DB6 DB7 = 000100 = 4 4 bit mode
         data_to_write=4;
         digitalWrite_multiple(enabled_gpio,6,data_to_write);
         pulsePin(enabled_gpio,data_to_write,6,5,delay);
         delayms(delay);

        //enable display part 1 - Display ON/OFF & Cursor
        //E RS DB4 DB5 DB6 DB7 = 000000 = 0
          data_to_write=0;
          digitalWrite_multiple(enabled_gpio,6,data_to_write);
         pulsePin(enabled_gpio,data_to_write,6,5,delay);
          delayms(delay);

        //enable display part 2 - Display ON/OFF & Cursor
        //E RS DB4 DB5 DB6 DB7 = 000011 = 3
         data_to_write=3;
         digitalWrite_multiple(enabled_gpio,6,data_to_write);
        pulsePin(enabled_gpio,data_to_write,6,5,delay);
        delayms(delay);
        //two line mode 5x7 part 1
        //E RS DB4 DB5 DB6 DB7 = 000100 = 4
       data_to_write=4;
         digitalWrite_multiple(enabled_gpio,6,data_to_write);
        pulsePin(enabled_gpio,data_to_write,6,5,delay);
        delayms(delay);

        //two line mode 5x7 part 2 (the 0 next to the 1 specifies the 5x7)
        //E RS DB4 DB5 DB6 DB7 = 000001 = 1
    data_to_write=1;
    digitalWrite_multiple(enabled_gpio,6,data_to_write);
        pulsePin(enabled_gpio,data_to_write,6,5,delay);
        delayms(delay);
}
```

Eine andere wichtige Funktion ist *unsigned int write_character(char Character, int part*). Mit dieser Funktion wird ein 8-Bit Befehl oder eine 8-Bit-Datei in zwei Teilen getrennt. Ebenso

wie die Initialisierungsfunktion generiert jeder Teil mit der R/S-Bitmaske und R/W-Bitmaske eine neue 6-Bit Datei, die in diesen Switch-Klammern direkt in Dezimalzahlen umgewandelt wird. Abbildung 4.4 zeigt einen Ausschnitt des Codes dieser Funktion.

```
unsigned int write_character (char character, int part)
{
    assert(part==0 || part==1);

    char acB[2];
    sprintf(acB, "%x", character);

    unsigned int return_value = 0;

    switch (acB[part])
    {
        case '0': return_value = 16; break;
        case '1': return_value = 24; break;
        case '2': return_value = 20; break;
        case '3': return_value = 28; break;
        case '4': return_value = 18; break;
        case '5': return_value = 26; break;
        case '6': return_value = 22; break;
        case '7': return_value = 30; break;
        case '8': return_value = 17; break;
        case '9': return_value = 25; break;
        case 'a': return_value = 21; break;
        case 'b': return_value = 29; break;
        case 'c': return_value = 19; break;
        case 'd': return_value = 27; break;
        case 'e': return_value = 23; break;
        case 'f': return_value = 31; break;
    }

    return(return_value);
}
```

Abbildung 4.4: Funktion write_character()

5 SMS-Protokolldefinition

Für die Realisierung der Erkennung einer SMS wurde ein Programm im Rahmen dieser Arbeit prototypisch entwickelt. In diesem Kapitel werden das Design sowie die Implementierung des Protokolls beschrieben.

5.1 Beschreibung des Protokolls

Die ARM Prozessor Platine „BeagleBone“ wird mittels USB-Schnittstelle mit dem GSM-Modem verbunden und ist danach in der Lage, mit dem Benutzer zu kommunizieren. Da der existierende SMS-Mechanismus durch AT-Befehle gesteuert wird, kann das GSM-Modem auch durch solche Befehle kontrolliert werden. Hierfür wird ein Protokoll aus Sicht des Benutzers entwickelt. Dabei wird verdeutlicht, wie das System durch das Handy gesteuert wird.

Um den zu übertragenden Inhalt lesen zu können, muss der Befehl mit dem gegenwärtigen Parameter in einen SMS-Container nach einem standardisierten Format verpackt werden. Jeder Rahmen des SMS-Containers besteht aus drei Teilen: Header, End und Payload Daten. Header und End werden jeweils durch eine spitze Klammer davor „<“ und eine danach „>“ ausgedrückt. Und das Payload verteilt sich noch in zwei Unterteile, dem Command und dem Parameter. Abbildung 5.1 stellt das Format einer zu übertragenden SMS dar. Da das Versenden und Empfangen von SMS standardisiert sind, lässt sich dadurch eine bidirektionale Datenverbindung zwischen BeagleBone und Handynutzer aufbauen. [14]

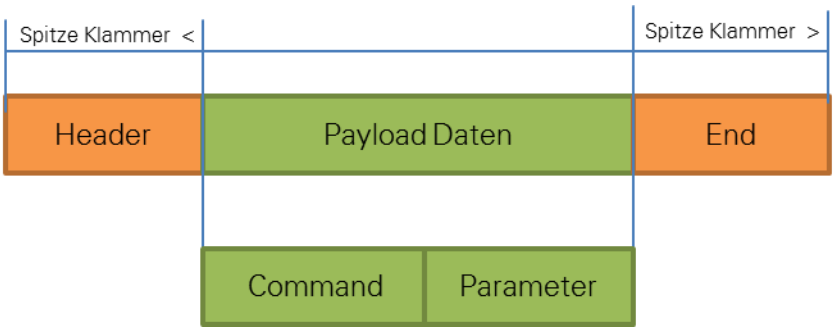


Abbildung 5.1: Format des SMS

Es gibt insgesamt fünf Kommandos, die Einstellung des Temperaturalarms, die Einstellung des Modus, das Abfragen des Status, das Einloggen des Nutzers sowie das Wechseln des Passworts.

Temperatur einstellen.

Mit diesem Kommando kann man das Temperaturintervall einstellen. Dieser Befehl beginnt mit einer Abkürzung **T** für die Temperatur. Danach folgen ein niedriger Temperaturwert, ein

Leerzeichen und ein hoher Temperaturwert. Was wichtig ist, dass der Temperaturwert eine vorzeichenbehaftete arabische Zahl mit nur einer Dezimalstelle zwischen -55 und +120 haben muss. Zum Beispiel <T-15.5 +27.5>.

Modus einstellen

Dessen Abkürzung ist M für Modus. Der Parameter ist ON oder OFF. Mit diesem kann man den Alarm der Temperatur einschalten und ausschalten. Z. B. <MOFF>. Wenn dieses Kommando durchgeführt wird, bekommt der Nutzer nicht mehr die Alarm-SMS.

Temperatur checken

C ist die Abkürzung des englischen Wortes „checken“ (überprüfen). Dieses einfache Kommando hat keinen Parameter und steht dafür, dass der Nutzer die aktuellen Temperaturen und den Status des Modus abfragen kann.

Benutzer einloggen

L bedeutet einloggen. Außer dem voreingestellten Hauptbenutzer, dessen Handynummer +4915213124133 in dieser Arbeit ist, dürfen sich auch andere Benutzer in das System einloggen. Nach L gibt man anschließend direkt das Passwort ein, welches eine vierstellige arabische Zahl muss.

Password wechseln

Das System muss auch in der Lage sein, den Wechsel des Passworts zu unterstützen. Außer dem alten Passwort müssen im Anschluss auch ein Leerzeichen und das neue Passwort angeboten werden. Und ebenso wie das alte Passwort besteht auch das neue Passwort aus einer vierstelligen Zahl.

Abbildung 5.2 zeigt grafisch die oben genannten definierten Steuerkommandos mit den Parametern und deren Aufbau.

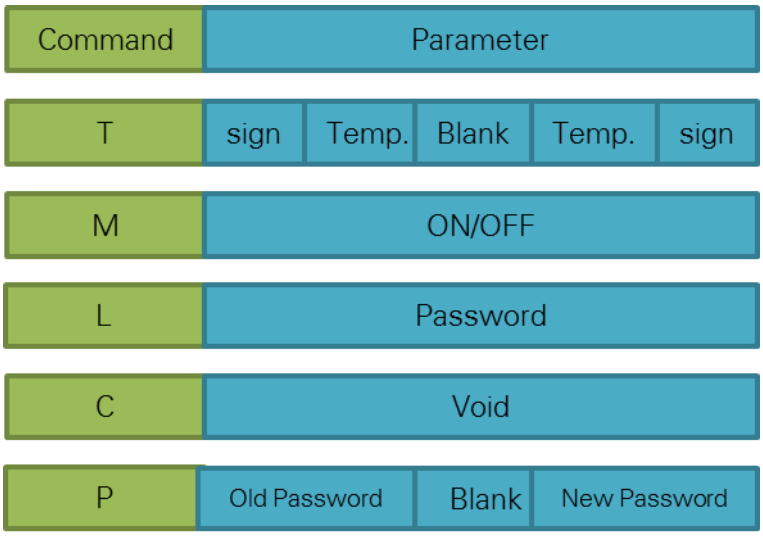


Abbildung 5.2: Definierte Kommandos mit Parameter

Tabelle 5-1 zeigt die Bedeutungen und typischen Beispiele der Steuerkommandos.

Tabelle 5-1: Prototypische Befehle aus Sicht der Nutzer

Nr.	Befehl	SMS-Format	Bedeutung	Beispiel
1	Temperaturalarm Einstellen(T)	T Low High	Umfang: -55˚C to +150 ˚C Genauigkeit: 0.5 ˚C	<T -15.5 +27.5>
2	Modus Einstellen(M)	M Nummer	1: Alarm Open 2: Alarm Close	<MON> <MOFF>
3	Temperatur Checken(C)	C	Derzeitige Temperatur anfragen	<C>
4	Benutzer einloggen(L)	L Password	Einloggen mit Passwort	<L1234>
5	Passwort Wechseln(P)	P Old New	Änderung des Passworts	<P1234 5678>

Bei der Kommunikation zwischen System und Benutzer gibt es auch eine Sequenz. Zuerst schickt das System an den Haupthandybenutzer eine Start-SMS, um ihn darüber zu informieren, dass das System bereits zur Verfügung steht. Dann kann der Haupthandybenutzer ein Steuerkommando an das System senden. Das System detektiert den Inhalt des Kommandos und schickt eine Rückmeldung, ob das gesendete Kommando erfolgreich erkannt wurde. Danach kann sich ein neuer Handynutzer mit dem L-Kommando einloggen. Selbstverständlich erhält dieser auch eine Rücknachricht. Wichtig ist, dass das System die Systemzeit nach dem ersten Erhalt der SMS aktualisiert.

Wenn die Temperatur der Umgebung nicht in dem gewünschten Intervall ist, schickt das System zweimal an alle eingeloggten Handynutzer eine Alarm-SMS. Abbildung 5.3 zeigt die teilweise Sequenz eines Ablaufs zwischen System und Handynutzer.

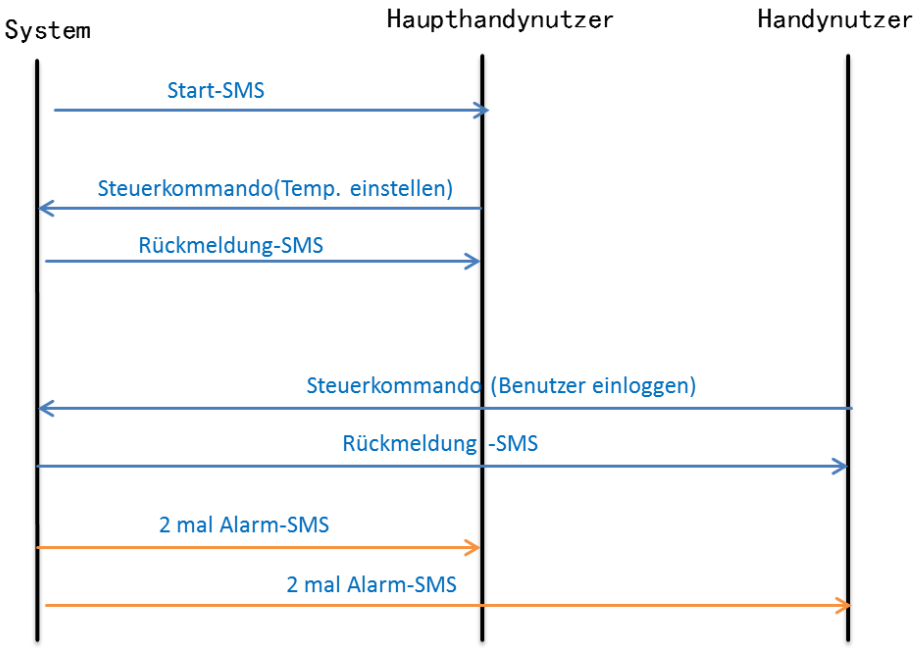


Abbildung 5.3: Teilweise Sequenz eines Ablaufs zw. System und Handynutzer

5.2 AT-Befehlssatz

Unter dem AT-Befehlssatz versteht man einen Satz, welcher ursprünglich von der Firma Hayes Communications entwickelt wurde und ein Quasi-Standard-Befehl zum Konfigurieren und Parametrieren von Modems geworden ist. Die Zeichen „AT“ stehen dabei für „Attention“ und müssen vor jedem Befehl gesendet werden. Durch Ausmessen der Länge der einzelnen Bits wird damit die Übertragungsgeschwindigkeit der Schnittstelle automatisch ermittelt. [11]

In dieser Arbeit wird jeder verwendete Befehl mit einem AT am Anfang begonnen und mit einem Enter-Zeichen beendet, wie zum Beispiel das Lesen und Empfangen der SMS durch *AT+CMGR* [12, Seite 284] und *AT+CMGS* [12, Seite 286] mit entsprechendem Parameter via seriellen Port. Natürlich werden beim Starten des Gerätes Initialisierungsbefehle *AT* benutzt. Tabelle 5-2 zeigt die wichtige verwendete AT-Kommandos in dieser Arbeit [12, Seite 274-326]

Tabelle 5-2: Wichtige verwendete AT-Kommandos in dieser Arbeit

AT	Initialization
ATE	Echo Off
AT+CMGC	Send an SMS command
AT+CMGD	Delete SMS message
AT+CMGF	Select SMS message format

AT Kommando	Funktion und Beschreibung
AT+CMGR	Read SMS message
AT+CMGS	Send SMS message
AT+CMGD	Delete SMS message

5.3 Auswertung der SMS-Daten

Wenn das System eine SMS erhält, gibt es darin außer Payload Daten natürlich auch andere Informationen, z. B. auf welcher Position des Gerätes wird die SMS gespeichert, oder von welcher Handynummer und wann wurde die SMS versendet.

Abbildung 5.4 zeigt eine SMS, welche mithilfe der RS-232-Software im Terminal eines Windows-Systems gelesen wurde. Im roten Rahmen stehen alle Informationen einer SMS. Sie zeigen an, dass diese SMS eine bisher ungelesene Nachricht um 12:52:40 Uhr, am 05.06.2012, von dem Handynutzer +4917638380094 erhalten wurde, und dessen Inhalt „hello“ war. Die Information im grünen Rahmen zeigt an, dass diese SMS auf Position 6 der SIM-Karte gespeichert wurde. Wie auf die terminalen Informationen im Linux-System zugegriffen und wie sie ausgelesen werden können, wird im nächsten Kapitel unter den Abschnitten 6.1 und 6.2 „Serielle-Kommunikation“ und „Modem-Kommunikation“ beschrieben.

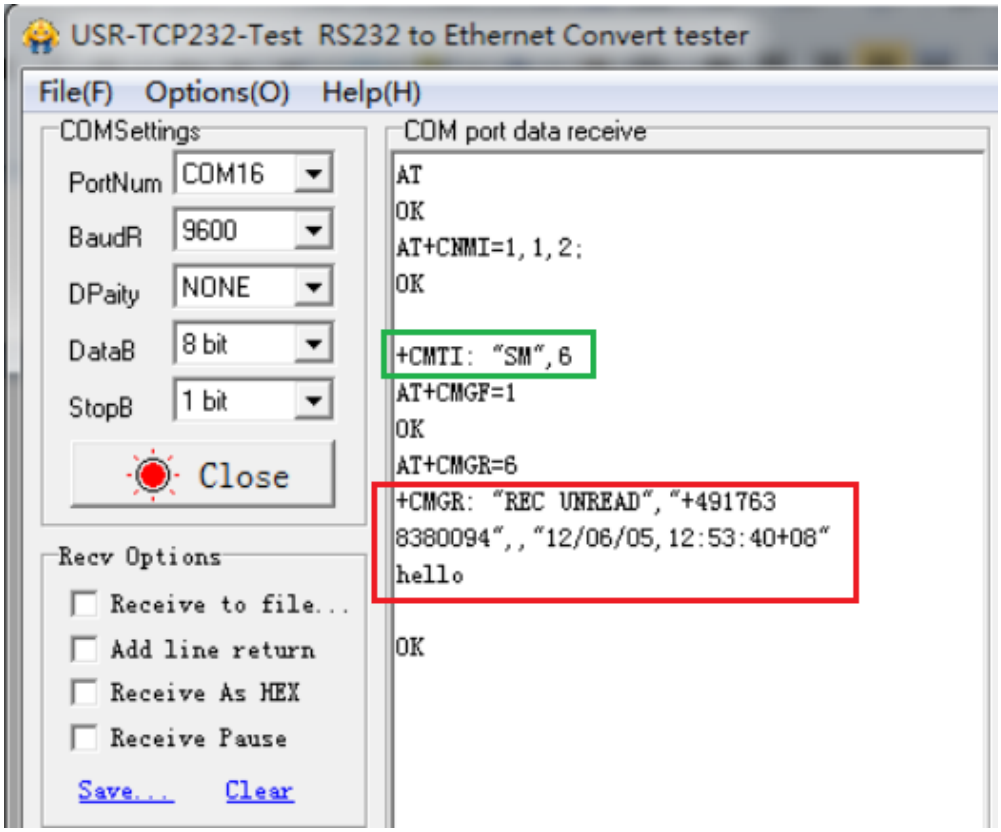


Abbildung 5.4: Terminale Information einer SMS im Windows-System

Solche wichtigen Informationen müssen mithilfe der Programmierung sortiert werden. In dieser Arbeit werden der Sender, das Datum und der Inhalt der SMS mit der Funktion *sms_save()* [2]sortiert und in einer Datenstruktur namens *sms_buffer* gespeichert. Übrigens, um die Information richtig lesen zu können, wird die erhaltene SMS immer auf der ersten Position der SIM-Karte gespeichert. Eine Methode wäre, dass die SMS nach dem Lesen direkt gelöscht wird, damit die erste Position immer leer ist.

```
struct sms_buffer
{
        char sms_sender[14];
        char sms_data[17];
        char sms_text[121];
};
```

An diesem Punkt ist der Inhalt bereits in *sms_text* gespeichert. Dann kommt die Analyse des Inhalts. Der erste Schritt ist die Entfernung der Header und End, um die Payload-Daten herauszunehmen. Anschließend werden das Steuerkommando und zum Schluss auch die nachfolgenden Parameter analysiert. Je nach den Steuerkommandos wird eine Rückmeldung gesendet.

5.4 Ablaufdiagramm

Abbildung 5.5 zeigt den Ablauf der Analysierung einer SMS beim Programmieren. [10]

[2] Siehe Anhang B Code, die in CD gespeichert wird

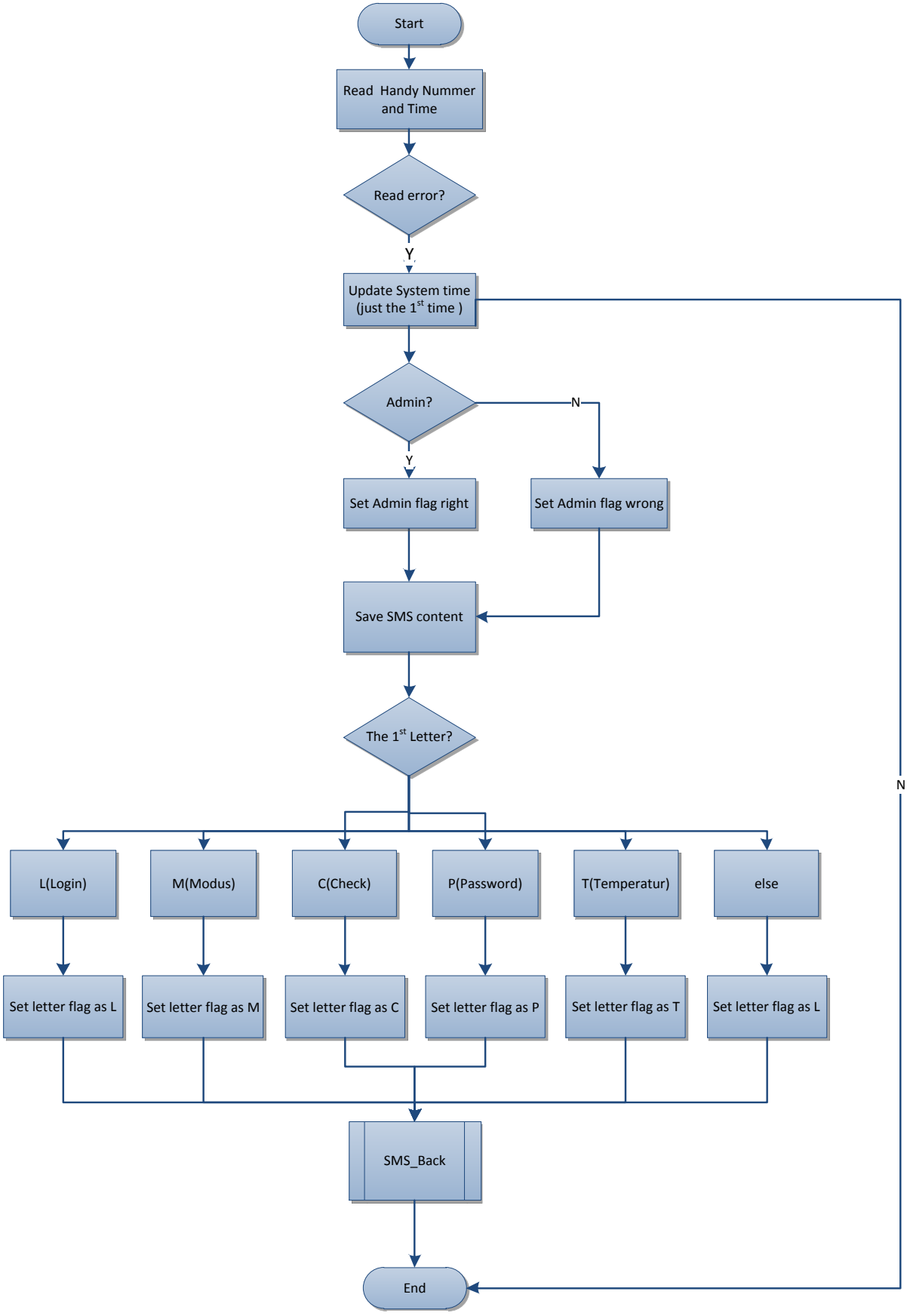


Abbildung 5.5: Flussdiagramm der Analyse der SMS

6 Ansteuerung des GSM-Moduls TC35i

In diesem Kapitel werden die serielle Kommunikation der RS-232 und deren Standards vorgestellt, die in den meisten Computern verwendet werden, sowie das in dieser Arbeit verwendete GSM-Modul TC35i. Weiterhin wird erläutert, wie im Linux-System auf eine serielle Schnittstelle zugegriffen wird, wie der entsprechende Port des Modems konfiguriert wird und wie das Senden und Empfangen einer SMS mithilfe der C-Programmierung realisiert werden.

6.1 Serielle Kommunikation

6.1.1 Grundlagen

Die serielle Kommunikation bezieht sich auf die Übertragung von Ein-Bit-Daten zu einer bestimmten Zeit. RS-232 ist ein Standard für eine bei Computern oft vorhandene serielle Schnittstelle, die in den frühen 1960ern von einem US-amerikanischen Standardisierungskomitee EIA eingeführt wurde. [9]

6.1.2 Programmierung in der Linux Umgebung

In dem Operationssystem sind die seriellen Ports mit verschiedenen Dateinamen vorhanden. In der Tabelle 6-1 werden die verschiedenen Namen der Ports für zwei typische Operationssysteme (Windows und Linux) gezeigt. In dieser Arbeit wird ein USB zu serieller Konverter in einem auf Linux basierten Gerät benutzt. Eigentlich sind die Operationen für das Device mit dem Namen /dev/ttyUSB0 entwickelt worden. [10] Tabelle 6-1zeigt Name des seriellen Ports in verschiedenen Systemen.

Tabelle 6-1: Name des seriellen Ports in verschiedenen Systemen

Operationssystem	Serieller Port 1	Serieller Port 1	USB zu Serielle Konverter 1
Windows	COM1	COM2	--
Linux	/dev/ttyS0	/dev/ttyS1	/dev/ttyUSB0

Operationen für serielle Ports

- Öffnung eines seriellen Ports

Da eine serielle Schnittstelle eine Datei ist, wird die *open()* Funktion verwendet, um darauf zugreifen zu können. Die Datei des Devices ist in der Regel von normalen Benutzern nicht zugänglich. Dazu muss man als Superuser das entwickelte Programm durchführen.

```
fd = open("/dev/ttyUSB0", ORDWR | O_NOCTTY | O_NDELAY);
```

Beim Öffnen des seriellen Anschlusses gibt es in dieser Funktion neben dem Lesen- und Schreiben-Modus noch zwei weitere Optionen. *O_NDELAY* Flag teilt dem System mit, dass es für das Programm egal ist, in welchem Zustand sich das DCD-Signal befindet.

- Schreiben und Lesen der Daten auf den Port

Das Schreiben von Daten auf den Port ist ganz einfach. Das kann mit der Funktion *write()* realisiert werden. Diese Funktion gibt die Zahl der gesendeten Bytes zurück, wenn die Daten richtig geschrieben sind, gegeben falls gibt die ein „-1" zurück, wenn ein Fehler aufgetreten ist.

```
n = write(fd, "TEST\r", 5);
if (n < 0)
  fputs("write() of 5 bytes failed!\n", stderr);
```

Der Systemaufruf *read()* kann die Daten der Ports auslesen und erhält die Zahl der gelesenen Bytes. Selbstverständlich wird der Speicherplatzt bei jedem Auslesen ausgeleert.

```
int read_serial(char *buffer,int size,int offset)
{
        int len;
        memset(buffer+offset,0,size-offset);
        if( ( len=read(fd,buffer+offset,size-offset) ) <=0 )   return FALSE;

        return len;
}
```

- Schließen eines seriellen Ports

Um den seriellen Port zu schließen, benutzt man einfach den Systemaufruf *close(fd)*. Beim Schließen der seriellen Schnittstelle wird das DTR-Signal[3] normalerweise auf „gering" eingestellt, was meistens das Ausschalten des Modems zur Folge hat.

```
close(fd);
```

Konfiguration des seriellen Ports

- Interface von POSIX Terminal

Das Linux System unterstützt die POSIX-Terminal-Schnittstelle, um Parameter wie Baudrate, Schriftgröße und so weiter ändern zu können. Die Head-Datei, welche die Dateierweiterung *<termios.h>* hat, unterstützt all diese Parameter. In dieser Head-Datei werden die Terminal Kontrollstruktur sowie die Steuerfunktionen des POSIX definiert. [10, Seite 209]
Die wichtigsten zwei POSIX Funktionen sind *tcgetattr()* und *tcsetattr().* Mit ihrer Hilfe kann man deutlich sehen, dass diese die Attribute des Terminals erhalten und einstellen können. Beim Aufruf dieser beiden Funktionen, muss man vorher eine Termios-Struktur anbieten, welche alle seriellen Optionen enthält. [10,Seite 200] Tabelle 6-2 zeigt eine Termios Struktur.

Tabelle 6-2: Termios-Struktur

Mitglied	Beschreibung
c_cflag	Control options
c_lflag	Line options
c_iflag	Input options
c_oflag	Output options
c_cc	Control characters
c_ispeed	Input baud
c_ospeed	Onput baud

Das Mitglied *c_cflag* kann die Baudrate, Schriftgröße, Parity Checking einstellen.

- Einstellung der Baudrate

Die Baudrate wird an verschiedenen Orten je nach Betriebssystem gespeichert. Für ältere Interfaces wird die Baudrate *c_cflat* mit einer der Baudratekonstanten gespeichert, während

[3] Endgerät betriebsbereit, Signal der V.24-Schnittstelle

die neuen Interfaces zwei Baudraten, *c_ispeed* und *c_ospeed,* anbieten, um die aktuelle Baudrate darin speichern zu können. Bei dem in unserer Arbeit verwendeten System Debian unterstützt es die neuen Interfaces. Die folgenden Codes sind ein sehr typisches Beispiel für die Einstellung der Baudrate auf B19200, was eine Kommunikationsgeschwindigkeit von 19200Bit pro Sekunde bedeutet.

```
struct termios options;

/* Get the current options for the port.../
tcgetattr(fd, &options);

/* Set the baud rates to 19200*/

cfsetispeed(&options, B19200);
cfsetospeed(&options, B19200);

/* Enable the receiver and set local mode...*/
options.c_cflag |= (CLOCAL | CREAD);

/* Set the new options for the port.*/
tcsetattr(fd, TCSANOW, &options);
```

- Einstellung der Schriftgröße

Anders als bei der Baudrate gibt es keine einfachen Funktionen. Stattdessen muss, um die Schriftgröße einzustellen, die Bitmaske verwendet werden.

```
options.c_flag &= ~CSIZE; /* Mask the character size bits */
options.c_flag |= CS8;    /* Select 8 data bits */
```

- Einstellung von Parity Checking

Ähnlich wie bei der Einstellung der Schriftgröße müssen die Parity Enable und Typ von Parity-Checking manuell eingesetzt werden. Der serielle Treiber unterstützt drei Varianten, also gerade (Even), ungerade (Odd) und kein Parity-Checking.[10, Seite 207]

```
options.c_cflag &= ~PARENB
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
```

```
options.c_cflag |= CS8;                 // kein Parity-Checking
```

```
options.c_cflag |= PARENB
options.c_cflag &= ~PARODD
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS7;             //Gerade Parity-Checking
```

```
options.c_cflag |= PARENB
options.c_cflag |= PARODD
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS7;             //ungerade Parity-Checking
```

6.2 Modem Kommunikation

Der erste Schritt der Kommunikation mit dem TC35i ist eigentlich die Bearbeitung für den seriellen Port namens */dev/ttyUSB0*, welcher in der Öffnung und Konfiguration des Ports besteht.

```
int             fd;
struct termios options;

/* open the port */
fd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY);
fcntl(fd, F_SETFL, 0);

/* get the current options */
tcgetattr(fd, &options);

/*set serial rate 9600*/
cfsetispeed(&serial_new_attr,B9600)<0
cfsetospeed(&serial_new_attr,B9600)<0

/* set raw input, 1 second timeout */
options.c_cflag     |= (CLOCAL | CREAD);
options.c_lflag     &= ~(ICANON | ECHO | ECHOE | ISIG);
options.c_oflag     &= ~OPOST;
options.c_cc[VMIN]  = 0;
```

```
options.c_cc[VTIME] = 10;

/* set the options */
tcsetattr(fd, TCSANOW, &options);
```

Als nächster Schritt wird eine Verbindung mit dem Modem aufgebaut. Der beste Weg ist, das Kommando „AT“ an das Modem zu senden. Wenn das GSM-Modem richtig angeschlossen und eingeschaltet ist, wird es eine Antwort mit „OK“ zurückschicken, ansonsten wird eine Fehlermeldung „Error“ zurückgeschickt.

```
char write_buffer[121],read_buffer[255];
int receipt;
        bzero(write_buffer,sizeof(write_buffer));
        sprintf(write_buffer,"AT\r");
        write_serial(write_buffer);

        if( (receipt=sms_receipt(read_buffer,sizeof(read_buffer),"OK","ERROR"))==
            RECEIPT_OK )
           {
            ;
           }
        else if( receipt==RECEIPT_ERROR )
           {
           return FALSE;
           }
        else
           {

                printf("read return receipt fail");
                return FALSE;
            }
```

Modem Konfiguration

Nachdem das BeagleBone erfolgreich mit dem Modem kommuniziert hat, werden weitere Kommandos übertragen, damit das Modem richtig konfiguriert wird und die SMS tadellos gesendet und empfangen wird.

Die Konfiguration des Modems besteht aus folgenden Schritten:

Schritt 1: Ausschalten des Echos

Schritt 2: Einstellung des Alarms für ankommende, neue SMS

Schritt 3: Einstellung des SMS-Formats

Ebenfalls, wie das oben erwähnte Kommando „AT“, sendet das Modem nach erfolgreicher Annahme solcher Kommandos eine Bestätigungsantwort „OK“ zurück.

```
int sms_init(char *sms_port)
{
char write_buffer[121],read_buffer[255];

/*ATE switch off*/
                bzero(write_buffer,sizeof(write_buffer));
                sprintf(write_buffer,"ATE0\r");
                errorhandler();

/*set alarm manner of the new SMS*/
                bzero(write_buffer,sizeof(write_buffer));
                sprintf(write_buffer,"AT+CNMI=2,1\r");
                errorhandler();

/*set SMS format*/
                bzero(write_buffer,sizeof(write_buffer));
                sprintf(write_buffer,"AT+CMGF=1\r");
                errorhandler();

return TRUE;
}
```

6.3 SMS senden und empfangen

Nach der erfolgreichen Konfiguration des Modems und der Annahme des Bestätigungskommandos „OK“ wird die SMS zunächst gesendet und empfangen. Mit dem Abstellen des Fehlermechanismus beim Programmieren haben die nachfolgenden beiden Codeschlüssel die Funktionen, die Standard-Short Message zu senden und zu empfangen. [8, Seite 274]
Um die SMS zu senden, ist ein wichtiges AT-Kommando „*AT+CMGS=*“zu verwenden. Dieses Kommando ist immer mit der zu sendenden Handynummer vorhanden. Wenn man zum Beispiel die Nachricht an die Nummer *+4915213717954* schicken möchte, muss der Befehl „*AT+CMGS=+4915213717954*“ zuerst an das Modem geschrieben werden. Nach dem Aufruf dieses Befehls wartet man, bis die Eingabeaufforderung „>“ erscheint. Dann kann man die Nachrichten schreiben und endet mit einem hexadezimalen Endzeichen „*0X1A*“, welches Enter bedeutet. Beim richtigen Versenden bekommt das GSM-Modem ein „OK“ zurück. [10] [12]

```
int sms_send(char *sms_text,char *number)
{
char write_buffer[121],read_buffer[255];
int receipt;
        bzero(write_buffer,sizeof(write_buffer));

sprintf(write_buffer,"AT+CMGS=");
        strcat(write_buffer,number);
        strcat(write_buffer,"\r");
        if( !write_serial(write_buffer) )
        {
                return FALSE;
        }
        else
        {


        if( (receipt=sms_receipt(read_buffer,sizeof(read_buffer),">","ERROR"))==REC
EIPT_ERROR | receipt==FALSE )
                return FALSE;
                else
                {
                        bzero(write_buffer,sizeof(write_buffer));
                        strcat(write_buffer,sms_text);
                        strcat(write_buffer,"\x1a");
                        if( !write_serial(write_buffer) )
                        {
                                return FALSE;
                        }
                        else
                        {

if( (receipt=sms_receipt(read_buffer,sizeof(read_buffer),"OK","ERROR"))==RE
CEIPT_OK )
                                {
                                        return RECEIPT_OK;
                                }


else if( receipt==RECEIPT_ERROR )
                                {
                                        return RECEIPT_ERROR;
                                }
                                else
                                {
                                        printf("read return receipt fail");
```

```
                                        return FALSE;
                                    }
                                }
                            }
                        }
return RECEIPT_OK;
}
```

Im Vergleich zum Senden spielt das AT-Kommando „*AT+CMGR*“ beim Empfang eine Rolle. Dieses Kommando existiert mit einem Positionsparameter, in dem die auszulesende Short Message gespeichert wird, welche im Programm *sms_id* genannt wird. Wenn man zum Beispiel die Nachricht, die auf dem ersten Speicherplatz der SIM-Karte steht, herausausnehmen möchte, muss der Befehl „*AT+CMGR=1*“ an den Port geschrieben werden. Nach Annahme des Inhalts ist der Empfangsprozess abgeschlossen. Warum man immer die erste Position lesen muss und wie man den Inhalt analysiert, ist bereits im letzten Kapitel erläutert worden.

```
int sms_read_one(char *read_buffer,int size,char *sms_id)
{
char write_buffer[121];
int receipt;

        bzero(write_buffer,sizeof(write_buffer));
        sprintf(write_buffer,"AT+CMGR=");
        strcat(write_buffer,sms_id);
        strcat(write_buffer,"\r");
        if( !write_serial(write_buffer) )
        {
                printf("send read command fail");
                return FALSE;
        }
        else
        {
        Backinformation(); //when right return RECEIPT_OK , when False return
                           RECEIPT_ERROR
return FALSE;
}
```

7 Bedienungsanleitung und Untersuchung

In diesem Kapitel wird die allgemeine Bedienungsanleitung des aufgebauten Gerätes vorgestellt, mit deren Hilfe es gesteuert werden kann. Die folgende Tabelle 7-1 zeigt die festgelegte Konfiguration des Gerätes.

Tabelle 7-1: Festgelegte Konfiguration

Default Setting	
Hauptbenutzer	+4915213124133
Temperaturintervall	0°C~100°C
Alarm	Anschalten(ON)
Systemzeit	25-12-1999

1. Starten

Die Stromversorgung des Gerätes wird in die Steckdose zuerst gesteckt. Nach quasi einer Sekunde wird die Starttaste auf der Vorderseite gedrückt. Nach etwa fünf bis sechs Sekunden werden die Temperatur und Systemzeit auf Display angezeigt. Nachdem der Hauptbenutzer die SMS mit dem Inhalt „*GSM start successful*" erhalten hat, bedeutet das, dass das Gerät richtig startet und läuft.

2. Temperatur einstellen

Wie man die Temperatur einstellen kann, siehe Abschnitt 5.1 „Beschreibung des Protokolls".

3. Benutzer anmelden

Das Gerät unterstützt, außer dem Hauptbenutzer, auch noch einen anderen Benutzer. Dieser hat nach der Anmeldung die gleiche Priorität wie der Hauptbenutzer, nämlich SMS zu senden und zu empfangen.

Hinweis: Die Handynummer muss eine vierzehnstellige Zahlengruppe sein, wie zum Beispiel +4915213124133.

4. Rückmeldung

Nach jedem Senden einer SMS wird der Benutzer eine Rück-SMS bekommen, Dadurch kann er bestätigen, ob die gesendete SMS richtig aufgenommen ist. Tabelle 7-2 zeigt die Rückmeldungen für die gesendeten SMS.

Tabelle 7-2: Rückmeldungen für die gesendeten SMS

Nr.	Befehl	Richtig	Falsch
1	Temperaturalarm Einstellen(T)	"Setting successful! Low:0,0 High:100.0"	"Setting error! Setting by default Low: 0.0 High:100"
2	Modus Einstellen(M)	"Setting successful! Alarm ON"	"Setting error! command unknown"
3	Temperatur Checken(C)	"(Temperature 27.7f) (Alarm ON) (Setting 0 to 100) (Users)"	"Setting error! command unknown"
4	Benutzer einloggen(L)	"You are now a User,and controll the GSM!"	
5	Password Wechseln(P)	"Setting successful! New Password is XXXX"	"Setting error! Password wrong!"

8 Zusammenfassung und Ausblick

In dieser Masterarbeit wurde das GSM-Temperaturüberwachungssystem erläutert. Die dafür erforderliche System-Hardware wurde ausgewählt und die entsprechende System-Software entwickelt. Dann wurden alle Elemente in einem Gehäuse aufgebaut wie ein Produkt, welches sich noch in der Entwicklungsfase befindet. Danach wurde mithilfe des vorhandenen GSM-Netzes der Versuch getestet. Bis jetzt kann das Gerät richtig betrieben werden. Die Steuerung und Überwachung mithilfe der SMS funktioniert ebenfalls vollständig. Hierfür gilt mein besonderer Dank Herrn Dr.-Ing. Chmielewski und Herrn Prof. Dr.-Ing. Siemens für die umfangreiche Betreuung und das Vertrauen in dieses wichtige Thema. Herrn Prütting danke ich recht herzlich für die Unterstützung und Bereitstellung der erforderlichen Geräte im Labor.

Mit der zunehmenden Anwendung des eingebetteten Systems im Forschungs- und Industriebereich ist davon auszugehen, dass die Fernüberwachung auch in Zukunft eine immer wichtigere Rolle spielen wird. Dieses Thema findet mehr und mehr Aufmerksamkeit in den unterschiedlichsten Bereichen, in denen es wichtig ist, die Umgebungstemperatur zu überwachen. Man kann dieses Überwachungssystem zukünftig noch erweitern, zum Beispiel, indem der BeagleBone-Platine noch weitere verschiedene Sensoren wie Luftdruck, Luftfeuchtigkeit und Konzentration des Gases hinzugefügt werden. Über die Mobilekommunikation wird das GSM-Modem durch ein GPRS-Modem oder 3G-Modem ersetzt. Diese beiden neuen Modems stellen eine neue Kommunikationsgeneration dar. Mit dieser kann eine immer schnellere und komfortablere, bidirektionale Verbindung aufgebaut werden. [10]

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Dr.-Ing. Chmielewski, Prof. Dr.-Ing. Siemens. *Aufgabenstellung der Masterarbeit.* Hochschule Anhalt, 2012

[2] ARM LTD. Home Page
URL: http://www.arm.com/ , 2012

[3] Texas Instruments Incorporated. Home Page
URL: http://www.ti.com/ , 2012

[4] Maxim Integrated. Home Page
URL: http://www.maximintegrated.com/, 2012

[5] Wikipedia: Max232-Chip
URL: http://en.wikipedia.org/wiki/MAX232, Stand: 09.01.2013

[6] Texas Instruments: BeagleBone Rev A6 System Reference Manual,
URL: http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf
Stand: 18.09.2012

[7] Maxim Integrated: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
URL: http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf, Stand: 12.12.2012

[8] Xiamen Amotec Display CO.LTD: Specifications of LCD Module.
URL: http://www.sparkfun.com/datasheets/LCD/ADM1602K-NSA-FBS-3.3v.pdf.
Stand: 12.10.2012

[9] S. Fischer, U. Walther, S. Schmidt, C. Werner. *Linux-Netzwerke*. Nicolaus Millin Verlag, 2. Aufl., 2005

[10] R Stones, N Matthew. *Linux Programmierung*. Mitp-Verlag/Bonn, 2. Aufl., 2005

[11] Wikipedia: „AT-Befehlssatz". Stand: 30.10.2012
URL: http://de.wikipedia.org/wiki/AT-Befehlssatz , Stand: 12.10.2012

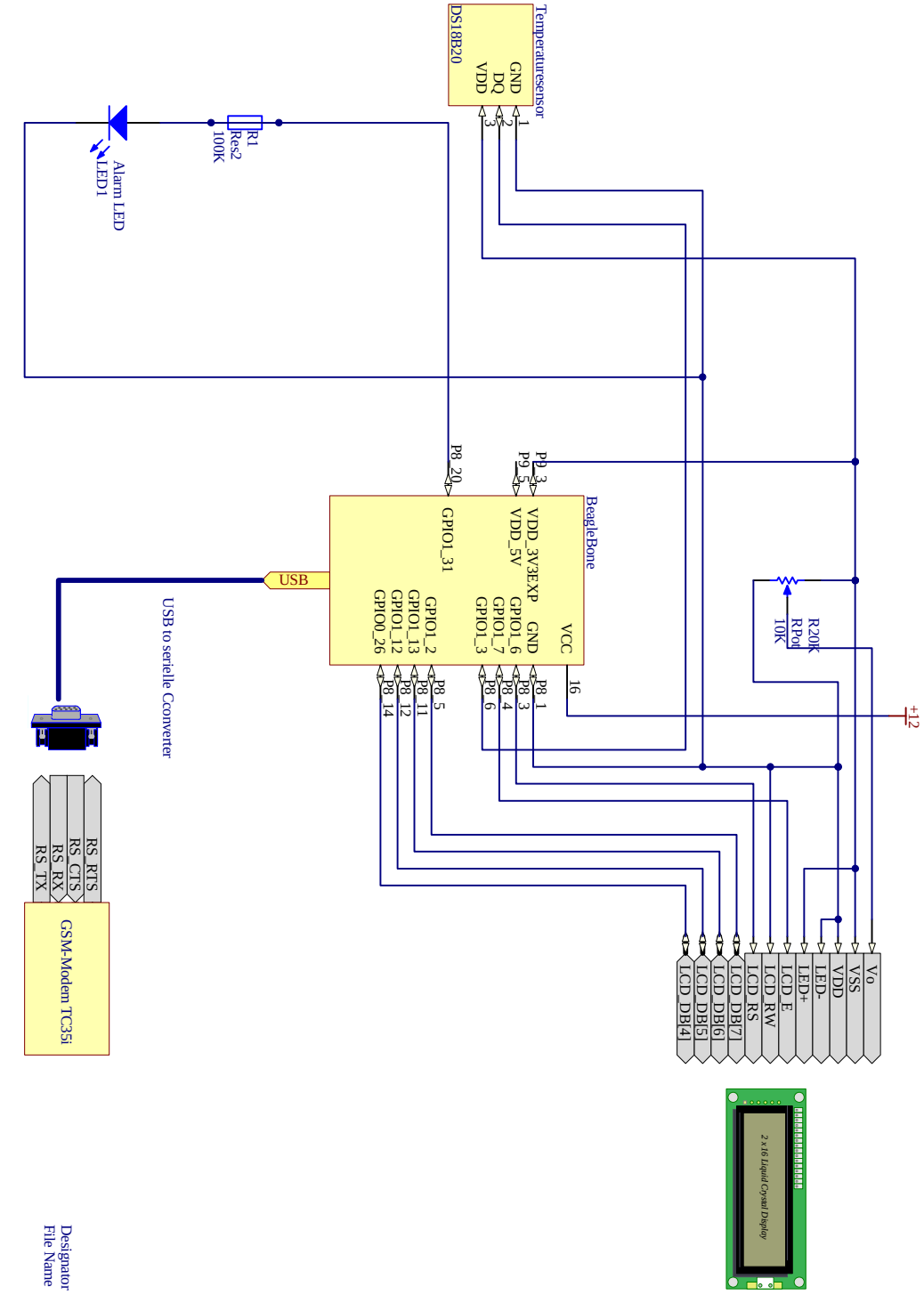
[12] Siemens Mobile: TC35i AT Command Set.
URL: http://kurser.iha.dk/eit/mic1/MICDocs/tc35i_atc_0207.pdf, Stand: 16.01.2013

[13] Siemens Mobile: TC35i Terminal Hardware Interface Description.
URL: http://www.vega.com/downloads/ext-BA-EN/Siemens-TC35i-EN.pdf,
Stand: 16.01.2013

[14] C. Lüders. *Mobilfunksysteme*. Vogel Buchverlag, 1.Aufl.,2001

[15] S Hetze, D Hohndel, M Müller, O Kirch. *Linux Anwenderhandbuch und Leitfaden für die Systemverwaltung*. LinetiX,7.Aufl., 1997

Anhang A


Temperatursensor
DS18B20
GND
DQ
VDD
Alarm LED
LED1
R1
Res2
100K
P8_20
P9_3
P9_5
BeagleBone
VDD_3V3EXP
VDD_5V
GPIO1_31
GPIO1_6
GPIO1_7
GPIO1_3
GND
VCC
USB
GPIO1_2
GPIO1_13
GPIO1_12
GPIO0_26
P8_1
P8_3
P8_4
P8_6
P8_5
P8_11
P8_12
P8_14
16
R20K
RPot
10K
+12
USB to serielle Cconverter
RS_RTS
RS_CTS
RS_RX
RS_TX
GSM-Modem TC35i
Vo
VSS
VDD
LED-
LED+
LCD_E
LCD_RW
LCD_RS
LCD_DB[7]
LCD_DB[6]
LCD_DB[5]
LCD_DB[4]
2 x 16 Liquid Crystal Display
Designator
File Name
Schaltplan des gesamten Systems
Bing Zhang