HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

SMS-Versand mit Raspberry Pi und Arduino über das Terminal Siemens TC35i

Stefan Brändle

Projekt angewandte Forschung

Inhaltsverzeichnis

1. Zusammenfassung

Das Ziel dieses Projektes war es, über einen Raspberry Pi bzw. Arduino SMS zu versenden und zu empfangen. Hierzu wurde ein baugleiches Modell zum Siemensmodem TC35i verwendet. Um die Kommunikation zwischen Steuerung und Modem zu ermöglichen wurden jeweils für Raspberry Pi und Arduino Platinen zur Pegelwandlung designt. Für den Testaufbau wurden diese Platinen auf einer Lochrasterplatine umgesetzt – für den späteren Einsatz wurden mit Hilfe des Programms EAGLE der Firma Autocad auch funktional erweiterte Platinen in SMD-Technik vorbereitet.

Das Kernstück der Arbeit ist ein Kommandozeilenprogramm, über das am Raspberry Pi unter Linux SMS gesendet und empfangen werden können. Da für den Arduino zum Zeitpunkt der Entwicklung bereits eine vollständig ausprogrammierte Lösung im Netz verfügbar war, wurde diese lediglich nachvollzogen und in einzelnen Punkten angepasst.

Das Projekt hat ein sehr breites Anwendungsspektrum. Grundsätzlich kann es überall dort zum Einsatz kommen, wo eine Steuerung sporadisch Statusinformationen zu senden hat oder in der Lage sein muss, Steueranweisungen zu empfangen. Der große Vorteil dieser Lösung gegenüber einem gewöhnlichen Internet-Surfstick ist, dass die Netzabdeckung in Deutschland für GSM, welches für SMS genutzt wird, gegenüber der Netzabdeckung für mobiles Internet wesentlich besser ist. So kann die Lösung zum Beispiel in einem Bienenstock eingesetzt werden, der sich mitten im Wald befindet, und von dort zuverlässig Sensordaten zum Füllstand der Waben oder zum Zustand des Bienenvolks an den Imker übermitteln. Ein weiteres Anwendungsbeispiel betrifft ein Projekt der Hochschule München. Hier befindet sich auf dem Dach eine Station, die von einem Solarpanel mit Bleiakku gespeist wird. Ist im Winter nun das Solarpanel längere Zeit mit Schnee bedeckt, so kann sich der Bleiakku tiefentladen und damit selbst zerstören. Dem könnte Abhilfe geschafft werden, indem die Batteriespannung in regelmäßigen Abständen überprüft wird und im kritischen Fall eine SMS an den Betreiber geschickt wird: „Batteriespannung niedrig – bitte Solarpanel überprüfen“.

2. Aufbau

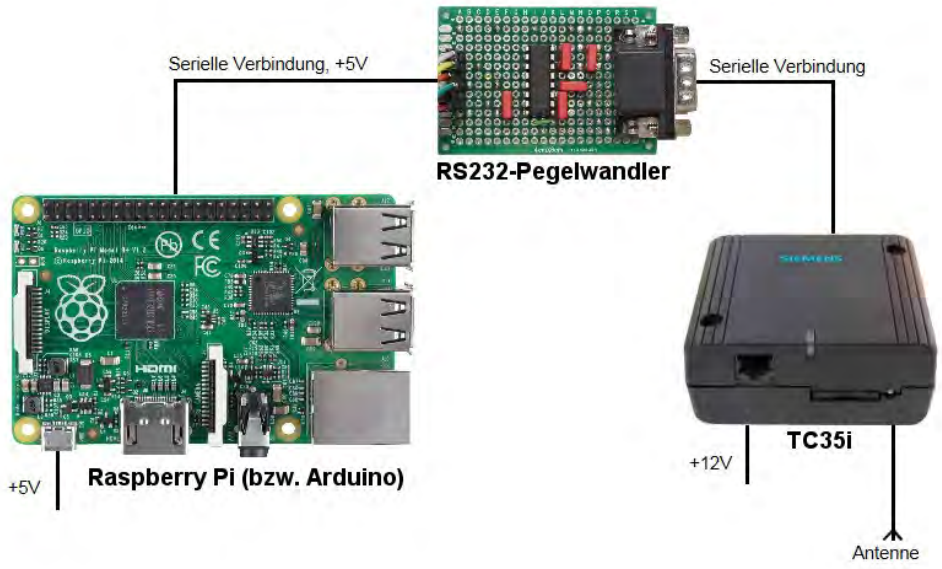

Abbildung 2.1.: Überblick über Aufbau und Verkabelung

Der Aufbau des Projektes ist nicht sehr kompliziert. Das Modem wird mit einem Netzteil über einen 6-poligen Westernstecker mit 12 V Spannung versorgt – es kann auch mit einer Spannung zwischen 8–30 V betrieben werden. In dem Modem befindet sich eine SIM-Karte in Mini-SIM-Größe (25 mm×15 mm) – diese kann auch mit einer PIN geschützt sein. Des Weiteren sind am TC35i eine Antenne und über einen 9-poligen SUB-D-Stecker die Pegelwandlerplatine angeschlossen.

Die Platine zur Umsetzung zwischen (LV)TTL-Pegel und V.24-Pegel (RS232-Pegel) wird mit einem 9-poligen seriellen Kabel an das Modem und über Patch- oder Flachbandkabel an Raspberry Pi bzw. Arduino angebunden. Die genaue Verkabelung der Leitungen für die serielle Verbindung kann den entsprechenden Stromlaufplänen entnommen werden (siehe Anhang A auf Seite 17). Das Raspberry Pi erhält seine Spannungsversorgung über eine Micro-USB-Buchse, während beim Arduino hierfür eine Hohlsteckerbuchse vorgesehen ist. Beide werden mit 5 V Versorgungsspannung betrieben.

3. Platine zur RS232-Pegelwandlung

3.1. Grundsätzliches

Das TC35i verwendet, wie schon erwähnt, eine serielle RS232-Schnittstelle zur Verbindung mit dem PC. Da sowohl Raspberry Pi als auch Arduino auf der seriellen Schnittstelle mit Logikpegeln arbeiten, ist eine Pegelanpassung notwendig. Außerdem muss eine 9-polig SUB-D-Buchse vorhanden sein, da das TC35i auch mit einer solchen Buchse ausgerüstet ist. Deren Belegung zeigt die folgende Tabelle.

Im Rahmen des Projektes entstanden drei verschiedene Platinendesigns. Für Arduino und Raspberry Pi werden unterschiedliche Umsetzer-ICs benötigt – hierfür kann jedes der drei Designs mit passenden Bauteilen bestückt werden. Konkret unterscheiden sich der IC, welcher die Spannungswandlung durchführt, und die von diesem benötigten Kondensatoren.

Die drei Designs lassen sich wie folgt charakterisieren:

- Lochrasterplatine – diese wurde als einzige für den Prototypen aufgebaut
- Ätzplatine des Prototypen – Bauteile in Durchsteckmontage
- Erweiterte Ätzplatine – Bauteile in Oberflächenmontage, weitere Leitungen der seriellen Schnittstelle wurden geroutet

Grundsätzlicher Zweck der Platinen ist die Pegelwandlung zwischen (LV)TTL-Pegeln (3.3 V bzw. 5 V) der GPIOs auf Raspberry Pi bzw. Arduino und den V.24-Pegeln (±12 V), die vom TC35i benötigt werden. Die 5 V-Spannungsversorgung der Wandlerplatine wird über den Raspberry Pi bzw. den Arduino realisiert. Die höheren Pegel werden dann über eine Ladungspumpe mit Hilfe der Pumpkondensatoren auf der Platine erzeugt.

Pin no.	Signal name	I/O	Function
1	/DCD	O	Data Carrier Detected
2	/RXD	O	Receive Data
3	/TXD	I	Transmit Data
4	/DTR	I	Data Terminal Ready Attention: The ignition of TC35i Terminal is activated via a rising edge of high potential (+5 ... +15 V)
5	GND	-	Ground
6	/DSR	O	Data Set Ready
7	/RTS	I	Request To Send
8	/CTS	O	Clear To Send
9	/RI	O	Ring Indication

Abbildung 3.1.: Belegung der RS232-Buchse am TC35i

Es ist darauf zu achten, dass bei der Variante für den Arduino, welche mit dem MAX232 betrieben wird, 1 µF-Kondensatoren verwendet werden. Da es sich um Tantal-Elektrolyt-Kondensatoren handelt, müssen diese mit der richtigen Polung eingelötet werden. Die Variante des Raspberry Pi arbeitet mit kleineren 100 nF-Kondensatoren, die als Folienkondensatoren in beliebiger Orientierung eingebaut werden können.

Grundsätzlich ist ein weiterer Stolperstein, wie RX und TX Leitungen verbunden werden. RX für Empfangen (engl. receive) und TX für Senden (engl. transmit) ist bei beiden Geräten aus Sicht der Steuerung angegeben. Das heißt am Modem ist der RX-Anschluss nicht – wie der Name suggeriert – ein Eingang. Es müssen also jeweils die RX-Pins an Steuerung und Modem miteinander verbunden werden, in gleicher Weise gilt das auch für die TX-Pins.

Schaltplan und Layout der im Folgenden beschriebenen Platinen, finden sich jeweils unter Anhang A.

3.2. Prototyp: Lochrasterplatine

Auf Grund der einfachen Herstellbarkeit, wurden die zwei Prototypen als Lochrasterplatinen aufgebaut. Vom Raspberry Pi bzw. Arduino werden die folgenden sechs Leitungen geführt:

- 5 V
- 2× Ground
- RX
- TX
- RTS

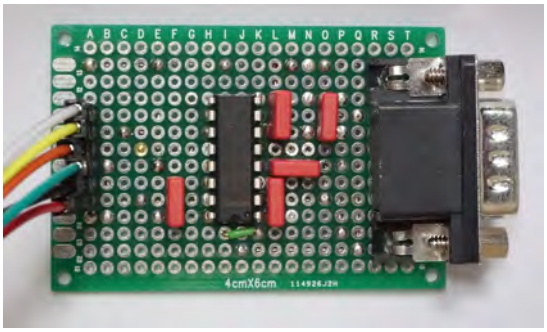

Abbildung 3.2.: Prototypenplatine in Draufsicht

In der ersten Revision der Platine wurde fälschlicherweise die DTR-Leitung anstelle der RTS-Leitung genutzt. Die DTR-Leitung dient dazu, das TC35i einzuschalten.

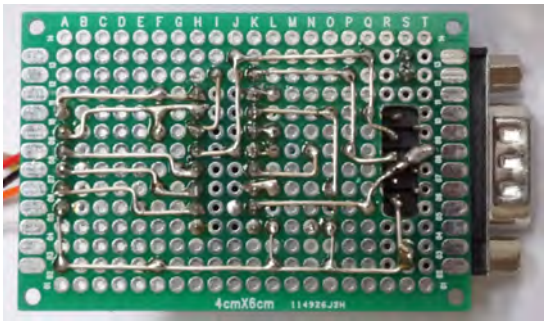

Abbildung 3.3.: Prototypenplatine in Untenansicht

Dies ist nur für spezielle Anwendungen erforderlich, bei denen ein erweitertes Energiemanagement implementiert wird (siehe den Ausblick in Kapitel 6 auf Seite 14). Die RTS-Leitung ist – zusammen mit der CTS-Leitung – Teil des Hardware-Handshakes zur Steuerung des Datenflusses. Diese Leitung muss aktiv gesetzt sein, dass das Modem AT-Befehle empfängt und beantwortet. Über den AT-Befehl `AT\Q0` kann die Steuerung des Datenflusses komplett deaktiviert werden. Ist jedoch ein Hardware-Handshake aktiviert, so kann dieser Befehl gar nicht kommuniziert werden. Es ist folglich die RTS-Leitung zwingend erforderlich. Darüber hinaus kann diese Leitung verwendet werden, um das Modem aus dem Standby-Modus aufzuwecken. Dies ist jedoch in der aktuellen Programmversion noch nicht umgesetzt (siehe Kapitel 6 auf Seite 14).

3.3. Vom Prototypen abgeleitete Ätzplatine

Diese Platine ist nahezu identisch zur Prototypen-Platine. Sie ist ebenfalls für Bauteile zur Durchsteckmontage bestimmt. Sie kann im Gegensatz zur Lochrasterplatine relativ einfach in größerer Stückzahl gefertigt bzw. bestellt werden. Mit ihr können alle Funktionen der unter Kapitel 4 auf Seite 7 und Kapitel 5 auf Seite 13 beschriebenen Programme genutzt werden. Eine kleine Änderung wurde in der Belegung der Buchsenleiste vollzogen – TX und RX sind im Vergleich zur Lochrasterplatine vertauscht. Damit ergibt sich die praktische Möglichkeit, die Platine direkt als Shield auf dem Raspberry Pi aufgesteckt zu betreiben. Ein Flachbandkabel zur Verbindung ist dann nicht mehr erforderlich.

3.4. Erweiterung: Platine in SMD-Technologie

Die erweiterte Platine nutzt Bauteile der SMD-Technologie. Sie erweitert die bisher beschriebenen Platinen um zwei weitere geführte Leitungen der seriellen Schnittstelle. Die oben schon beschriebene DTR-Leitung und die RING-Leitung wurden ebenfalls an die SUB-D-Buchse angeschlossen.

Es ist darauf zu achten, dass der zum Footprint passende IC gekauft wird – in SMD-Bauweise existieren zwei verschiedene Gehäusegrößen des Chips. Hier verwendet

wird die Baugröße SOIC mit einem Pitch von 1.27 mm.

Achtung: Da der eingesetzte IC zur Pegelwandlung in beide Richtungen nur zwei Kanäle zur Pegelwandlung bietet, wurde die DTR-Leitung direkt (am IC vorbei) geführt. Der Spannungsbereich zwischen 2 V und 5 V ist laut Datenblatt [**tc35hw**] jedoch undefiniert. Um das TC35i einzuschalten sind also mindestens 5 V Spannung erforderlich. Damit lässt sich die Funktion nur mit dem Arduino verwenden. Und auch hier muss die Zuverlässigkeit vorher getestet werden. Des Weiteren ist zu berücksichtigen, dass bei der seriellen Schnittstelle RX und TX einer inversen Logik folgen, während die Steuerleitungen positive Pegel als logische 1 interpretieren.

Die RING-Leitung wird beim Empfang einer SMS für einen kurzen Zeitraum aktiv geschaltet. In zukünftigen Anwendungsszenarien könnte dies für einen Interrupt an der Steuerung genutzt werden - damit würde ein Polling nach neu empfangenen SMS entfallen. Dies ist in der aktuellen Programmierung jedoch ebenfalls noch nicht realisiert (siehe auch Kapitel 6 auf Seite 14).

4. Raspberry Pi

4.1. Überblick über das Programm

In den folgenden Unterkapiteln sollen Aufbau, Funktionalität und Bedienung des Kommandozeilenprogramms *SMS* für den Raspberry Pi näher erklärt werden. Eine Übersicht über die einzelnen Programmmodule gibt Abb. 4.1 auf der nächsten Seite.

Für den Nutzer liegen die zwei Anwendungsfälle auf der Hand: SMS senden und SMS empfangen. Die Bedienung des Programms ist bewusst sehr einfach gehalten und auf das Wesentliche beschränkt. Daher verfügt es über keinerlei Menüstrukturen oder grafischer Oberflächen. Die gesamte Interaktion des Nutzers mit dem Programm wird über Kommandozeilenargumente geregelt, die der Nutzer dem Programm bei dessen Aufruf übergibt. Werden keine Arumente übergeben, so erscheint ein Hilfetext, der die Anwendung erklärt.

Eine Voraussetzung für die Nutzung des Programms ist, dass die wiringPi Bibliothek installiert ist [**wiringpi**]. Von dieser wird das Kommandozeilenprogramm *gpio* verwendet, da es den Zugriff auf die GPIOs des Raspberry Pi bietet, ohne root-Rechte zu erfordern.

Dem *SMS*-Programm liegt kein Installer bei. Es kann im Quellverzeichnis über den Befehl

```
# make
```

aus den Quelldateien erstellt werden. Hierfür ist das Paket `build-essential` erforderlich, welches über den Paketmanager mit

```
# sudo apt-get install build-essential
```

vorab geladen werden muss.

Das Programm kann SMS mit bis zu 160 Zeichen wahlweise als gewöhnliche SMS oder als Flash-SMS verschicken. Eine Flash-SMS wird normalerweise nicht im Speicher des empfangenden Geräts abgelegt und direkt auf dem Display angezeigt. Das senden von verketteten SMS mit mehr als 160 Zeichen ist nicht möglich.

Beim Abrufen empfangener SMS, wird die erste ungelesene SMS im Speicher auf der Standardausgabe ausgegeben und anschließend gelöscht – dies hat den Vorteil, dass die Ausgabe direkt über eine Pipe in einem anderen Kommandozeilenprogramm weiterverarbeitet werden kann. Sollte sich keine neue SMS im Speicher befinden, wird auf der Standardausgabe nichts ausgegeben, sondern nur auf dem Standarderror-Kanal. Achtung: Die erste ungelesene SMS im Speicher des TC35i muss nicht zwangsläufig die älteste ungelesene SMS sein. Befanden sich vor der Inbetriebnahme des TC35i mit dem *SMS*-Tool keine Nachrichten im Speicher, ist dies jedoch der Fall.

Die Programmmodule in Abb. 4.1 auf der nächsten Seite enthalten größtenteils auch, was deren Namen vermuten lässt:

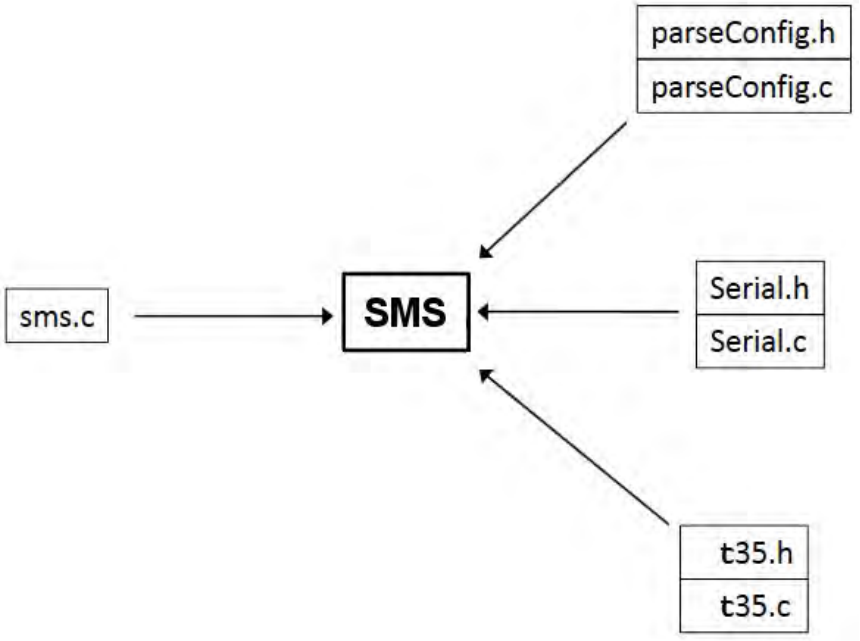

Abbildung 4.1.: Überblick über die einzelnen Module des Kommandozeilentools SMS

- sms.c – main()-Funktion und Verarbeitung von Kommandozeilenparametern und Konfiguration
- parseConfig – Funktionalität zum Einlesen einer Konfigurationsdatei
- serial – Konfiguration und Kommunikation über die serielle Schnittstelle
- tc35 – Funktionen die die Konfiguration und Kommunikation mit dem TC35i betreffen

Das *SMS*-Tool liefert als regulären Rückgabewert 0 zurück. Tritt ein Fehler auf, so wird ein Wert ungleich 0 zurückgegeben.

4.2. Kommandozeilenargumente

Zur Steuerung des Programmes sind drei verschiedene Schalter vorgesehen:

- `-r` – SMS empfangen
- `-s` [*Empfängerrufnummer*] [*SMS-Text*] – SMS senden
- `-f` – SMS als Flash-SMS senden (nur in Kombination mit -s)

Soll eine SMS gesendet werden und eine neue SMS abgerufen werden, so muss das *SMS*-Tool zwei mal separat aufgerufen werden. Werden die Argumente `-r` und `-s` im

selben Programmaufruf übergeben, so wird keine der Funktionen ausgeführt und das Programm beendet sich.

Werden dem Programm keine Parameter übergeben, so wird ein Hilfetext ausgegeben, der die Benutzung des Tools samt der Kommandozeilenparameter erklärt.

4.3. Konfiguration der seriellen Schnittstelle

Die Steuerung des TC35i erfolgt über sogenannte AT-Kommandos. Diese werden als einfache Zeichenfolge über die serielle Verbindung an das Modem gesendet. Der Name rührt daher, dass viele der Kommandos auch mit den Zeichen AT beginnen. Groß- und Kleinschreibung wird vom TC35i nicht ausgewertet. Die Kommandos müssen über ein angehängtes Carriage-Return-Zeichen \r abgeschlossen werden. Nur dann werden diese vom Modem auch ausgeführt. Die Ausführung der Befehle wird vom Modem immer quittiert.

Bevor eine Kommunikation über die serielle Schnittstelle auf dem Raspberry Pi möglich ist, sind zwei Schritte erforderlich:

1. Deaktivierung des standardmäßig auf der seriellen Schnittstelle aktivierten Loginterminals
2. Initialisieren der seriellen Schnittstelle im Programm mit den korrekten Kommunikationsparametern

Zur Deaktivierung des Login-Terminals genügte es beim vorliegenden Raspberry Pi Model B+ unter *Raspbian Jessie* die folgenden zwei Befehle über die Kommandozeile einzugeben. Um den dahinter liegenden Service zu stoppen:

```
# sudo systemctl stop serial-getty@ttyAMA0.service
```

Und um den Service dauerhaft abzuschalten:

```
# sudo systemctl disable serial-getty@ttyAMA0.service
```

Für ältere Versionen von *Raspbian* und das Modell Raspberry Pi 3, sind andere Vorgehensweisen erforderlich. Diese sind beispielsweise unter [**serialraspi**] beschrieben.

Damit die serielle Verbindung korrekt funktioniert wird diese im Programm mit folgenden Parametern initialisiert:

- 8 Datenbits
- 1 Stopbit
- Kein Paritätsbit
- Bitrate: 57 600 bps

Da das TC35i standardmäßig auf Autobauding eingestellt ist – es also die Bitrate mit der die Steuerung sendet automatisch ermittelt – sollte theoretisch jede Bitrate zur Kommunikation funktoinieren. Nach einem Neustart verwendet das TC35i automatisch die 57 600 bps, bis das erste Zeichen empfangen wird. Daher wurde gleich diese Bitrate am Raspberry Pi fix voreingestellt.

Bei Verbindungsproblemen und zum Testen von AT-Befehlen hat sich das Kommandozeilentool *minicom* als sehr praktisch erwiesen. Dieses lässt sich über den Paketmanager installieren. Eine Verbindung kann dann mit
`# sudo minicom -b 57600 -D /dev/ttyAMA0`
initiiert werden. Nun können die AT-Befehle bequem per Tastatur ans Modem geschickt werden.

Wie in Abschnitt 3.2 auf Seite 5 bereits beschrieben, ist es für eine erfolgreiche Kommunikation mit dem TC35i über die serielle Schnittstelle erforderlich, dass entweder die Datenflusskontrolle abgeschaltet wird oder der GPIO, an dem die RTS-Leitung angeschlossen ist, auf 0 gesetzt wird. Die 0 wird auf Grund der inversen Logik, die im Pegelwandler-IC repräsentiert ist, auf einen positiven Pegel gewandelt, der wiederum an den Steuerleitungen der seriellen Schnittstelle einer logischen 1 entspricht.

Weitere Steuerleitungen der seriellen Schnittstelle sind für die aktuelle Anwendung nicht nötig. In Kapitel 6 auf Seite 14 werden noch einige Anwendungsbeispiele für erweitere Funktionen beschrieben.

4.4. Grundlegende Programmkonfiguration

Die grundlegenden Programmkonfiguration kann über eine Konfigurationsdatei vorgenommen werden. Diese muss mit root-Rechten unter
`# /etc/tc35/tc35.conf`
angelegt werden. Ist diese nicht vorhanden, werden Standardwerte für die konfigurierbaren Werte übernommen. Diese sind aktuell (Standardwerte in Klammern):

- SIM-PIN (0000)
- Pfad zur seriellen Schnittstelle (`/dev/ttyAMA0`)
- GPIO Nummer der RTS-Leitung – Zählart nach wiringPi-Bibliothek (1)

Das folgende Listing zeigt ein Beispiel:

Listing 4.1: Beispiel einer Konfigurationsdatei

```
1 PIN=4711
2 TTY=/dev/ttyAMA0
3 GPIO=1
```

Der Parser für die Konfigurationsdatei ist recht einfach gehalten. Die Schlüssel und ihre zugehörigen Werte werden jeweils durch ein Gleichheitszeichen getrennt in separate Zeilen geschrieben. Es können Kommentarzeilen eingefügt werden. Hierfür muss die Zeile mit einem # beginnen, dann wird diese vom Parser komplett ignoriert.

Schlüssel und Wert werden als C-Strings in einem Array von Strukturen gespeichert. Es können vom Parser aktuell bis zu 8 Paare ausgelesen und ans Hauptprogramm übergeben werden. Dort müssen die Werte dann unter anderem in der Funktion `proccessArgs(...)` einprogrammiert werden.

Bei der grundlegenden Konfiguration des TC35i, die bei jedem Programmdurchlauf durchgeführt wird, wird die Erreichbarkeit des Modems mit dem einfachen Befehl

`AT` getestet. Sollte die Pineingabe über den Befehl `AT+CPIN=...` vom Modem mit *OK* quittiert worden sein, so läuft ein Timer von 20 Sekunden ab. Dies ist eine Sicherheitszeit, da das Modem in einem Zeitraum nach der Pin-Eingabe noch keine weiteren Anfragen verarbeitet. Des Weiteren werden über `AT+CMGF=1` der SMS-Textmodus und über `AT+CPMS=MT` die Nutzung von modeminternem- und Simkarten-Speicher aktiviert.

4.5. SMS senden

In Listing 4.2 sind die AT-Befehle aufgelistet, die zum Senden einer SMS nötig sind. Um Flash-SMS zu senden, muss der Befehl `AT+CSMP=17,167,0,16` statt dem `AT+CSMP=17,167,0,0` verwendet werden. Mit `AT+CMGS="+49..."` wird das SMS-Senden Kommando an das TC35i geschickt; nach dem =-Zeichen wird die Empfängerrufnummer in Anführungszeichen gleich mit übergeben. Nachdem dieses Kommando gewöhnlich mit \r gesendet wurde, kann der SMS-Text eingegeben werden. Erst wenn die Tastenkombination STRG+Z (dies entspricht in C \26 oder \x1A) an das TC35i geschickt worden ist, wird die SMS mit dem zuvor eingegebenen Text wirklich versendet.

Listing 4.2: Ablauf der AT-Befehle beim Senden einer SMS

```
1 AT+CMGF=1 // SMS-Textmodus einstellen
2 AT+CSMP=17,167,0,0 // Flash-SMS-Modus deaktivieren, fuer aktiven Flash-SMS-
      Modus letzte 0 durch 16 ersetzen
3 AT+CMGS="+49..." // Kommando: SMS-Senden und Empfaengerrufnummer
4 Nachricht...[STRG+Z oder \26 oder \x1A]
```

4.6. SMS empfangen

In Listing 4.3 ist der Ablauf dargestellt, wie im Programm eine ungelesene SMS aus dem Speicher des TC35i ausgelesen wird. Hierfür wird zuerst die Anzahl der im Speicher vorhandenen Nachrichten bestimmt (vgl. Listing 4.4 auf der nächsten Seite). Diese Information ist besonders von Vorteil, falls keine neue Nachricht vorhanden ist. Denn es werden nun nur über die Anzahl der belegten Speicherplätze über eine Schleife die gespeicherten Nachrichten abgefragt. Sollte sich in der Antwort des TC35i der String "REC UNREAD"befinden, wird die Nachricht extrahiert und ausgegeben. Das Listing 4.5 auf der nächsten Seite zeigt die Antwort des TC35i beim Auslesen einer bis dahin ungelesenen SMS. Aus dieser Antwort, die immer die gleiche Struktur hat, wird der SMS-Text mittels C-String-Funktionen ausgeschnitten.

Listing 4.3: Pseudocode mit AT-Befehlen zum Auslesen und extrahieren der ersten ungelesenen Nachricht im TC35i

```
1 AT+CPMS="MT" //Wichtige Einstellung, damit sowohl der interne als auch der SIM
      Speicher ausgelesen wird
2 AT+CPMS?
3 Finde erstes Komma in Antwort, -> Zeiger
4 Finde zweites Komma in Antwort, -> Zeiger
```

```
5  Extrahiere den String zwischen den beiden Zeigern und konvertiere zum Integer
       -> Anzahl i der Nachrichten im Speicher
6  //Die Anzahl der Nachrichten wird genutzt, um nicht ueber den ganzen Speicher
       iterieren zu muessen
7
8  Schleife ueber alle Nachrichten von 1 bis i mit AT+CMGR=i
9  In Antwort suche jeweils den String "REC UNREAD"
10   wenn nicht gefunden -> naechsten Speicherslot auslesen
11   wenn gefunden -> suche naechstes <lf>-Zeichen -> Zeiger
12   Finde Sequenz von <cr><lf><cr><lf>OK<cr><lf> -> Zeiger
13   Gib die Zeichen zwischen den Zeigern aus -> entspricht erster ungelesener SMS
         im Speicher
14 AT+CMGD=i //Loesche die ausgegebene SMS aus dem Speicher
```

Listing 4.4: Antwort des TC35i auf die Anfrage zum Speicherstatus

```
1  AT+CPMS?
2  +CPMS: "MT",1,45,"ME",1,25,"MT",1,45
3
4  OK
```

Listing 4.5: Antwort des TC35i auf den Lesebefehl eines Speicherslots, in dem eine ungelesene SMS gespeichert ist

```
1  AT+CMGR=1
2  +CMGR: "REC UNREAD","+491607650765",,"17/08/10,17:49:00+08"
3  Test 1,2,3
4
5  OK
```

Ein Vorteil dieser Vorgehensweise ist es, dass alle bisher vorhandenen und als gelesen markierten Nachrichten im Speicher belassen werden. Wird beispielsweise zu Testzwecken eine SIM-Karte eingesetzt, auf der private Nachrichten gespeichert sind, so bleiben diese unversehrt – nur die neue, gerade ausgelesene SMS wird gelöscht. Die extrahierte SMS wird nun auf die Standardausgabe geschrieben und kann beispielsweise über eine Pipe direkt in einem anderen Kommandozeilenprogramm weiterverarbeitet werden. Sollte keine neue SMS vorhanden sein, wird nichts auf der Standardausgabe ausgegeben sondern eine Meldung auf der Standardfehlerausgabe. Sollten mehrere neue Nachrichten vorhanden sein, so muss das *SMS*-Tool mehrfach aufgerufen werden, um diese auszulesen.

5. Arduino

Da für den Arduino bereits ein funktionierendes Beispiel im Internet gefunden werden konnte ([**2015**][**2015a**]), wurde dieses lediglich nachvollzogen und mit kleinen Änderungen versehen. Der Beispielcode lief auf dem verwendeten Arduino Uno sofort einwandfrei.

Die folgenden kleinen Änderungen wurden an dem Beispielcode vollzogen:

- PIN4 wird statt PIN8 dauerhaft auf 1 geschaltet – dies entspricht in unserem Fall der RTS-Leitung, statt des ign-PIN zum Einschalten.
- Senden von AT+CPIN=1690 zum Entsperren der verwendeten SIM-Karte.
- Abrufen von Nachrichten aus dem internen Speicher und von der SIM-Karte, statt nur von der SIM-Karte mit: AT+CPMS=MT statt AT+CPMS=SM.
- Um die Funktionalität des Codes überprüfen zu können, wurde die Onboard-LED geschaltet.

Der vollständige, leicht angepasste Beispielcode aus [**2015a**] findet sich in Listing D.1 auf Seite 24.

6. Ausblick

Verschiedene Ideen und Erweiterungen sind im *SMS*-Programm noch nicht realisiert. Diese sollen hier kurz aufgelistet werden.

1. Die Datenflusskontrolle an der seriellen Schnittstelle kann komplett deaktiviert werden, dann ist es nicht mehr erforderlich, dass die RTS-Leitung geschaltet wird. Dies lässt sich mit dem Befehl `AT\Q0` realisieren

2. Eine SMS wird bis jetzt im Textmodus versendet. Zum übermitteln von Binärdaten existiert auch ein Datenmodus, der sogenannte PDU-Modus.

3. Sollen in der Grundkonfiguration beliebige AT-Befehle gesendet werden können, so könnte dies über eine Datei geschehen, die ebenfalls geparset wird. Im Anschluss werden alle AT-Befehle vor dem eigentlichen Programm abgearbeitet.

4. Mit dem Befehl AT+CSCA=*Service-Center-Nummer* kann die Telefonnummer des SMS-Service-Centers geändert werden; wenn sich dies als relevant herausstellt, könnte es in die Konfigurationsdatei mit aufgenommen werden

5. Wird die erweiterte Platine verwendet, so könnte der Empfang einer SMS über die RING-Leitung einen Interrupt am Raspberry-Pi auslösen und das Abrufen der SMS direkt hervorrufen. So würde ein Polling-Verfahren vermieden.

6. Das TC35i arbeitet mit dem GSM-Zeichensatz. Bei bestimmten Zeichen führt dies zu Darstellungs- bzw. Übertragungsfehlern. Es könnte beispielsweise ein Modul ergänzt werden, das den eingegebenen SMS-Text auf GSM-Codierung übersetzt. Eine wohl einfachere – aber nicht getestete – Variante ist es, das TC35i intern auf den UCS2 Zeichensatz umzustellen – dieser ist eng verwandt mit UTF-16. Der Befehl hierzu lautet: `AT+CSCS="UCS2"` .

7. Einen Installer für das *SMS*-Tool schreiben, der per
   `# make install`
   das tool korrekt installiert

8. Wenn der Aufbau zum Beispiel in einem Bienenstock im Wald eingesetzt werden soll, so ist ein Energiemanagement sicher sinnvoll. Das TC35i bietet dafür verschiedene Standby-Modi. Über `AT+CFUN=0` lässt sich beispielsweise die serielle Schnittstelle komplett deaktivieren – über die RTS-Leitung kann das Modem dann wieder aufgeweckt werden. Eine weitere Möglichkeit wäre es, das TC35i mit `AT^SMSO` ganz herunterzufahren und über die DTR-Leitung dann wieder einzuschalten. Es ist zu beachten, dass das TC35i nicht mit 5 V betrieben

werden kann – es benötigt mindestens 8 V. Es wäre hier also auch ein DC-DC-Wandler nötig. Man müsste ausprobieren, ob eine zentrale Versorgung mit 5 V (Raspberry Pi oder Arduino) und Aufwärtswandlung auf 8–9 V (TC35i) oder eine zentrale Speisung mit 9 V (TC35i) und Abwärtswandlung auf 5 V weniger Leistung verbraucht.

9. Zuletzt ist es für den realen Einsatz des Aufbaus sicher ebenfalls sinnvoll, ein Gehäuse vorzusehen, in dem der Gesamtaufbau Platz findet und vor Nässe geschützt ist.

Anhang

A. Schaltpläne und Layouts der verschiedenen Platinen

A.1. Lochrasterplatine: Prototyp

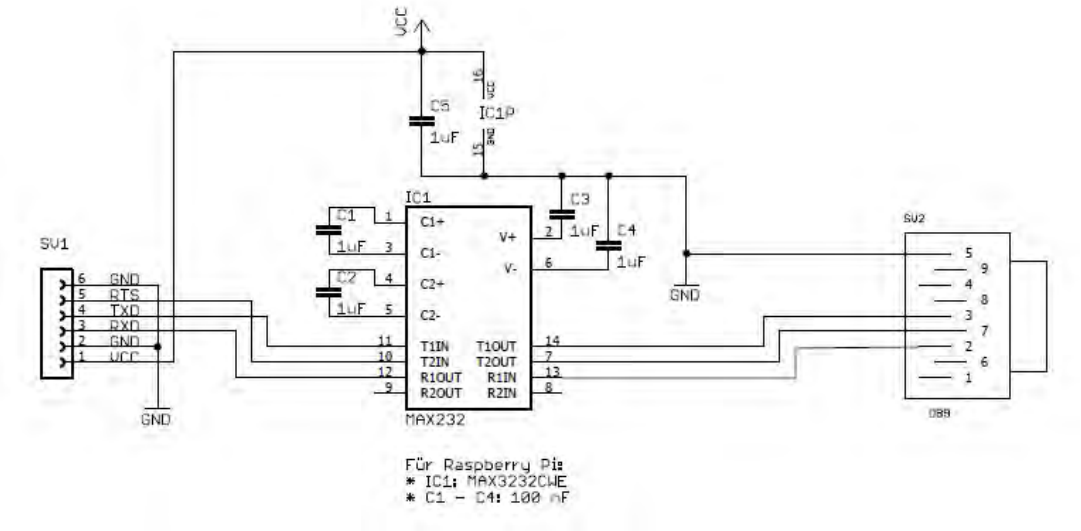

Abbildung A.1.: Schaltplan der Lochrasterplatine

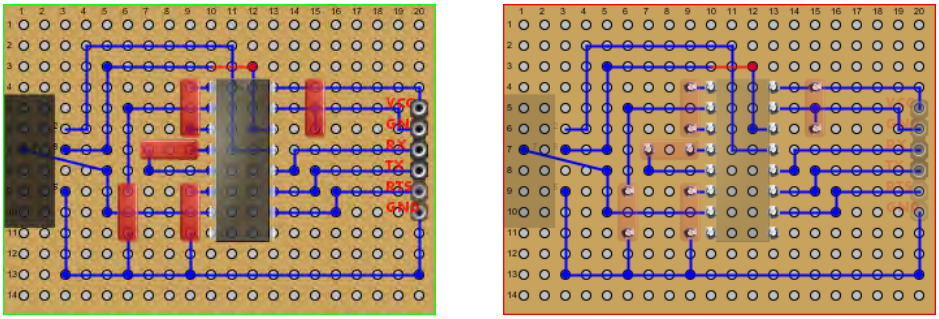

Abbildung A.2.: Layout der Lochrasterplatine in Draufsicht und Untenansicht (nicht maßstäblich)

A.2. Vom Prototypen abgeleitete Ätzplatine

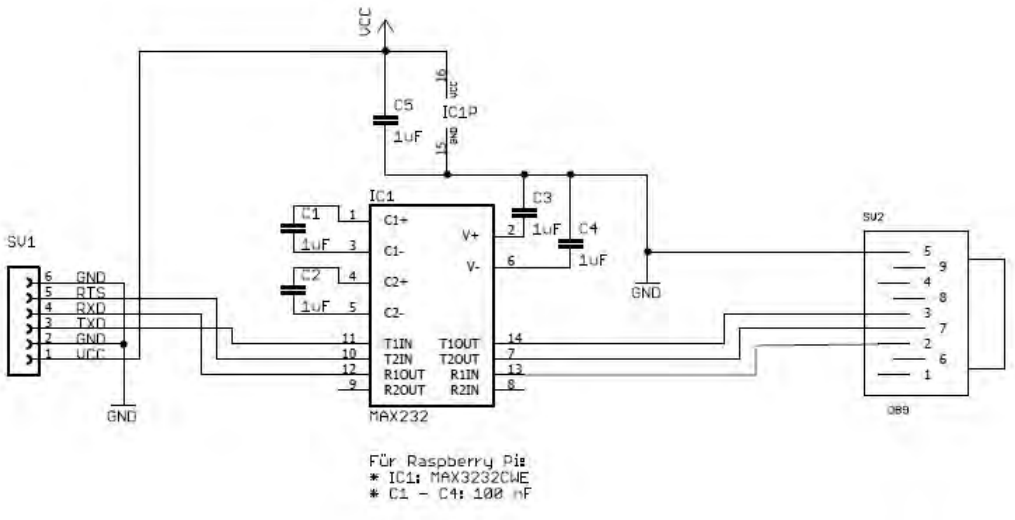

Abbildung A.3.: Schaltplan der Standardplatine

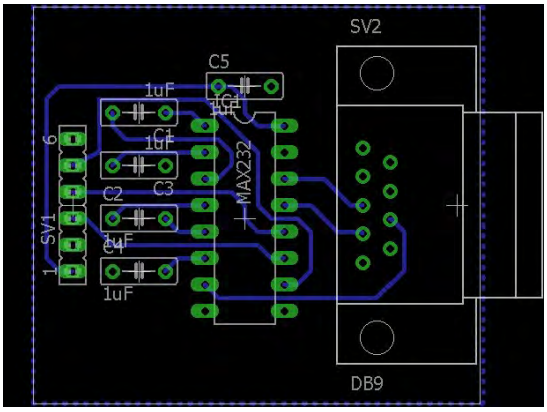

Abbildung A.4.: Layout der Standardplatine (nicht maßstäblich)

A.3. Erweiterung: Platine in SMD-Technologie

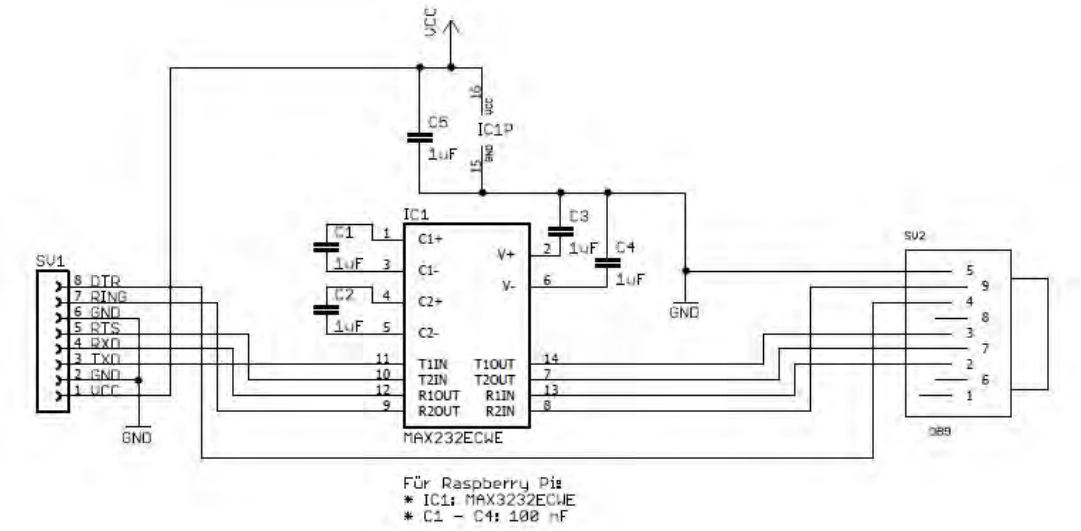

Abbildung A.5.: Schaltplan der SMD-Platine

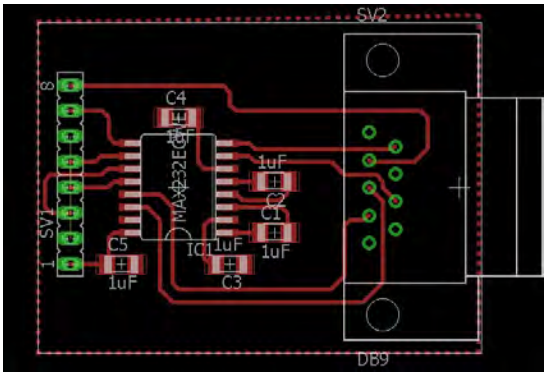

Abbildung A.6.: Layout der SMD-Platine (nicht maßstäblich)

B. Hinweise zum TC35i

B.1. Stromversorgung des TC35i

Die Stromversorgung des TC35i-Terminals erfolgt durch eine Spannungsquelle (Steckernetzteil) mit 8 bis 30 V Ausgangsspannung. Ein Spitzenstrom (gepulste 577 ms bei T = 4,615 ms) von etwa 1,1 A bei 12 V sorgt während der aktiven Übertragung einen Burst mit starker Welligkeit (Drop) auf der Versorgungsleitung. Der Drop der Spannung sollte nicht mehr als 1 V betragen. Die absolute Mindestspannung muss in jedem Fall größer als 7,6 V sein.

Der Eingang ist vor Verpolung und Überspannung geschützt. Eine interne, nicht austauschbare Sicherung sorgt für elektrische Sicherheit nach EN 60950. Zusätzlich wird als Schutz vor Eingangsspannungen über 30 V eine zusätzliche 1.25-A-Sicherung an Pin 1 des 6-poliger Westernsteckers empfohlen. Beachten Sie die EN 60950-Richtlinien.

Ein Schaltregler regelt die Eingangsspannung für die interne Versorgung. In POWER-DOWN-Modus wird der Schaltregler durch die Ein-Aus-Logik ausgeschaltet. Ein separater Spannungsregler liefert den Strom für die Echtzeituhr im GSM-System. Fällt die Stromversorgung für mehr als 1ms aus, wird das TC35i-Terminal zurückgesetzt oder es schaltet und ist für min. 7s nicht ansprechbar.

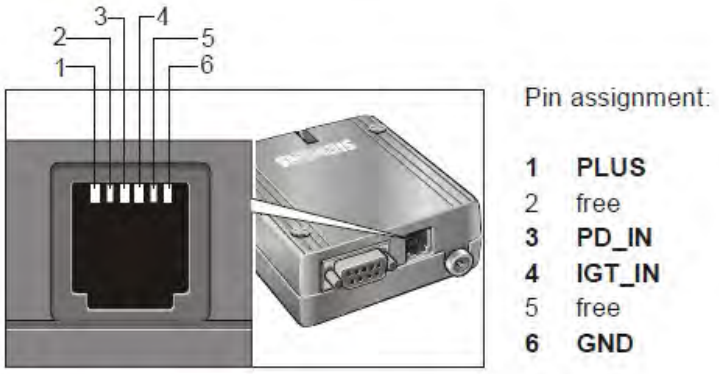

Abbildung B.1.: Belegung der Netzteilbuchse

Der mitgelieferte 12-V-Netzadapter ist mit einem 6-poligen Western-Stecker ausgestattet, bei dem eine interne Verbindung zwischen dem IGT_IN-Pin und dem PLUS-Pin für das automatische Einschalten besteht.

Pin	Signal	Zweck	Werte
1	PLUS	Stromversorgung	8 – 30 V
2	frei	–	–
3	PD_IN	Power-Down-Modus	Normalbetrieb: $U_{IH} < 2V$ Ausschalten: $U_{IH} > 5V$ für min. 3,5 s
4	IGT_IN	Ignition (s. unten)	Ein: $U_{IH} > 5V$
5	frei	–	–
6	GND	Masse	0 V

- *Ignition* wird nur durch eine steigende Flanke aktiviert. Die Anstiegszeit muss weniger als 20 ms betragen. Das IGT_IN-Signal schaltet das Terminal ein (es wechselt vom Abschaltzustand zur Netzsuche).

- Not-Ausschalten: Im Falle von Software-Hang-ups etc. kann das TC35i-Terminal durch Anlegen einer Spannung von mehr als 5 V an Pin 3 (PD_IN) für eine Dauer von mehr als 3,5 s abgeschaltet werden. Um wieder einzuschalten, haben Sie zwei Möglichkeiten: Den Pin 4 (IGT_IN) aktivieren oder die RS-232 DTR-Leitung einschalten, während der PD_IN-Pin nicht aktiv ist (Pin 3, Spannung kleiner 2 V). Das Signal PD_IN schaltet das Terminal aus. Alle internen Versorgungsspannungen sind ausgeschaltet, außer der Speisung der Echtzeituhr. Achtung: Die Aktivierung von PD_IN löscht alle Daten im NV-RAM, daher sollte dieses Methode nur im äußersten Notfall verwendet werden.

- Bei allen anderen Betriebsarten muss das Signal PD_IN kleiner 2 V sein.

- Wenn das TC35i-Terminal ausgeschaltet ist oder in den Power-Down-Modus eintritt, z. B. nach dem Befehl AT$\hat{\text{S}}$MSO oder das PD_IN-Signal aktiviert, sind alle RS-232-Leitungen während des internen Powerdown-Prozesses undefiniert. Dies kann zum Senden undefinierter Zeichen führen, die ignoriert werden können.

- Um das TC35i-Terminal ordnungsgemäß herunterzufahren, muss man nach dem Senden von AT$\hat{\text{S}}$MSO vor dem Ausschalten der Stromversorgung am Pin PLUS mindestens 10 s warten. Diese Zeit benötigt das Terminal zum sicheren Abmelden aus dem Netzwerk und Beenden des Speichern im internen Speicher.

Jedes Mal, wenn das TC35i-Terminal heruntergefahren wird, werden Daten aus dem NV-RAM in den Flash-Speicher geschrieben. Die garantierte maximale Anzahl von Schreibzyklen ist auf 100.000 begrenzt.

B.2. Einsetzen der SIM-Karte

Zum Betrieb des Terminals benötigen Sie eine Mini-SIM-Karte (3 Volt, entsprechend der Richtlinie GSM 11.12). Die SIM-Karte muss in den Halter („Schublade“auf der Rückseite des TC35i eingesetzt werden:

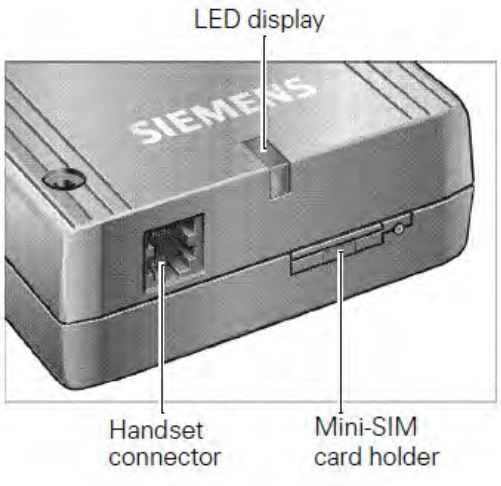

Abbildung B.2.: SIM-Karten-Halter

- Die SIM-Karte darf nur bei ausgeschaltetem GSM-/GPRS-Modem eingesetzt und/oder gewechselt werden. Der Wechsel bei anliegender Versorgungsspannung kann zu irreversibler Beschadigung der SIM-Karte und des GSM-/GPRS-Modem fuhren.
- Drücken Sie z. B. mit einem Kugelschreiber auf den kleinen gelben Knopf neben der Schublade des Halters. Die Schublade wird hierdurch ein kleines Stuck herausgeschoben.
- Ziehen Sie die Schublade heraus und setzen Sie Ihre SIM-Karte ein. Achten Sie auf die richtige Lage: Im Schubfach befindet sich eine kleine abgeschrägte Ecke, so dass die SIM-Karte nur in einer definierten Position eingesetzt werden kann.
- Schieben Sie die Schublade vorsichtig zuruck in das Terminal. Die Schublade muss sich leicht einschieben lassen und darf nicht verkanten oder verklemmen! In den Schlitz der Schublade dürfen keine Fremdkörper eingeführt werden! Nach dem Entnehmen der SIM-Karte setzen Sie die leere Schublade wieder ein, damit keine Fremdkörper ins Gerät gelangen können!

C. SMS-Zeichensatz

Basiszeichensatz

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
0x00	@	Δ	SP[4]	0	¡	P	¿	p
0x01	£	_	!	1	A	Q	a	q
0x02	$	Φ	"	2	B	R	b	r
0x03	¥	Γ	#	3	C	S	c	s
0x04	è	Λ	¤	4	D	T	d	t
0x05	é	Ω	%	5	E	U	e	u
0x06	ù	Π	&	6	F	V	f	v
0x07	ì	Ψ	'	7	G	W	g	w
0x08	ò	Σ	(	8	H	X	h	x
0x09	Ç	Θ	)	9	I	Y	i	y
0x0A	LF[1]	Ξ	*	:	J	Z	j	z
0x0B	Ø	ESC[3]	+	;	K	Ä	k	ä
0x0C	ø	Æ	,	<	L	Ö	l	ö
0x0D	CR[2]	æ	-	=	M	Ñ	m	ñ
0x0E	Å	ß	.	>	N	Ü	n	ü
0x0F	å	É	/	?	O	§	o	à

[1] ist ein Zeilenvorschub (LF, Linefeed)
[2] ist ein Wagenrücklauf (CR, Carriage Return)
[3] ist ein Escape-Zeichen (ESC)
[4] ist ein Leerzeichen (SP, Space)

Standardzeichensatzerweiterung

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
0x00					|			
0x01								
0x02								
0x03								
0x04		^						
0x05							€	
0x06								
0x07								
0x08			{					
0x09			}					
0x0A	FF[1]							
0x0B		SS2[2]						
0x0C				[				
0x0D				~				
0x0E				]				
0x0F			\					

[1] ist ein Seitenumbruch (FF, Form Feed oder Page Break)
[2] ist ein weiteres Single-Shift-Escape-Zeichen, reserviert für zukünftige Erweiterungen

Abbildung C.1.: Zeichensatz gemäß Standard GSM 03.38, „GSM-Alphabet“

D. Beispielcode für den Arduino

Listing D.1: Angepasster Arduino Beispielcode

```
#include <SoftwareSerial.h>
#define rxPin 2
#define txPin 3
#define statusPin LED_BUILTIN
#define RTSpin 4

String accessPin("1690");
String cmdOff("OFF");
String cmdOn("ON");
String cmdStatus("STATUS");

SoftwareSerial gsmSerial(rxPin, txPin);

String smsNumber;
String smsText;
String cmdMessage;
String cmdResponse;

int nMessages;
char c;
int state;
int switchStatus;

// ------------------------------------------------------------
void setup()
{
  Serial.begin(9600);
  while(!Serial) {};
  gsmSerial.begin(9600);

  pinMode(statusPin, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  switchStatus = false;

  //RTS dauerhaft setzen --> PIN 4 LOW
  pinMode(RTSpin, OUTPUT);
  digitalWrite(RTSpin, LOW);

  //PIN uebermitteln
  SendCmd("AT+CPIN=1690");
  delay(10000);

  // Format Text fuer SMS einstellen
  SendCmd("AT+CMGF=1");

  CountMessages();
  DeleteAllMessages();

  Serial.println("Ready");
}
```

```
51
52  // ---------------------------------------------------------------
53  void loop()
54  {
55    if (nMessages > 0)
56    {
57      ReadSms(nMessages);
58      Serial.println("Incoming: [" + smsNumber + "] [" + smsText + "]");
59      DeleteMessage(nMessages);
60      nMessages--;
61      if (smsText.length() >= accessPin.length() && smsText.startsWith(accessPin)
            )
62      {
63        smsText.remove(0, accessPin.length());
64        if (smsText == cmdOn)
65        {
66          digitalWrite(LED_BUILTIN, HIGH);
67          switchStatus = true;
68        }
69        else if (smsText == cmdOff)
70        {
71          digitalWrite(LED_BUILTIN, LOW);
72          switchStatus = false;
73        }
74        else if (smsText == cmdStatus)
75        {
76          smsText = "Switch is ";
77          if (switchStatus == true)
78          {
79            smsText += cmdOn;
80          }
81          else
82          {
83            smsText += cmdOff;
84          }
85          SendMessage();
86        }
87        else
88        {
89          smsText = "Available commands: " + cmdOn + ", " + cmdOff + " und " +
                cmdStatus;
90          SendMessage();
91        }
92      }
93    }
94    else
95    {
96      delay(5000);
97    }
98    CountMessages();
99  }
100
101 // ---------------------------------------------------------------
102 void SendCmd(String cmd)
103 {
104   gsmSerial.println(cmd);
105   delay(1000);
106   while(gsmSerial.available() > 0)
107   {
108     c = gsmSerial.read();
109   }
110   delay(100);
```

```
111 }
112
113 // ------------------------------------------------------------
114 void CountMessages()
115 {
116   nMessages = 0;
117   cmdMessage = "AT+CPMS=\"MT\"";
118   gsmSerial.println(cmdMessage);
119   delay(1000);
120
121   cmdResponse = "";
122   state = 0;
123
124   String num = "";
125   while(gsmSerial.available() > 0)
126   {
127     c = gsmSerial.read();
128     if (c != 13 && c != 10)
129     {
130       cmdResponse += c;
131     }
132
133     if (c == ':' && state == 0)
134     {
135       // Leerzeichen vor der ersten Zahl
136       c = gsmSerial.read();
137       state = 1;
138     }
139     if (c != ' ' && state == 1)
140     {
141       // Anfang der ersten Zahl
142       state = 2;
143     }
144     if (c == ',' && state == 2)
145     {
146       // Ende der ersten Zahl
147       state = 3;
148     }
149     if (state == 2)
150     {
151       num += c;
152     }
153   }
154   nMessages = num.toInt();
155 }
156
157 // ------------------------------------------------------------
158 void DeleteAllMessages()
159 {
160   while(nMessages > 0)
161   {
162     DeleteMessage(nMessages);
163     nMessages--;
164   }
165 }
166
167 // ------------------------------------------------------------
168 void DeleteMessage(int smsIndex)
169 {
170   cmdMessage = "";
171   cmdMessage = smsIndex + cmdMessage;
172   cmdMessage = "AT+CMGD=" + cmdMessage;
```

```
173   SendCmd(cmdMessage);
174 }
175
176 // ----------------------------------------------------------
177 void ReadSms(int smsIndex)
178 {
179   cmdMessage = "";
180   cmdMessage = smsIndex + cmdMessage;
181   cmdMessage = "AT+CMGR=" + cmdMessage;
182   gsmSerial.println(cmdMessage);
183   delay(1000);
184
185   smsNumber = "";
186   smsText = "";
187   cmdResponse = "";
188
189   state = 0;
190   // Format: +CMGR: "REC READ","+336xxxxxxxx",,"13/06/10,22:18:35+08"<CR><LF>
          Message<CR>
191   while(gsmSerial.available() > 0)
192   {
193     c = char(gsmSerial.read());
194     if (c != 13 && c != 10)
195     {
196       cmdResponse += c;
197     }
198
199     if (c == ',' && state == 0)
200     {
201       // Anfang der Telefonnummer
202       c = char(gsmSerial.read());  // " ueberlesen
203       c = char(gsmSerial.read());
204       state = 1;
205     }
206     if (c == '"' && state == 1)
207     {
208       // Ende der Telefonnummer
209       state = 2;
210     }
211     if (c == 13 && state == 2)
212     {
213       // Anfang Text der SMS
214       c = char(gsmSerial.read());  // <LF> ueberlesen
215       state = 3;
216     }
217     if (c == 13 && state == 3)
218     {
219       // Ende Text der SMS
220       state = 4;
221     }
222
223     switch(state)
224     {
225       case 1:
226         smsNumber += c;
227         break;
228       case 3:
229         smsText += c;
230         break;
231     }
232   }
233
```

```
234   smsText.toUpperCase();
235   smsText.trim();
236 }
237
238 // ------------------------------------------------------------
239 void SendMessage()
240 {
241   if (smsText.length() > 0)
242   {
243     Serial.println("Outgoing: [" + smsText + "] an " + smsNumber);
244
245     cmdMessage = "AT+CMGS=\"" + smsNumber + "\"\r";
246     gsmSerial.print(cmdMessage);
247     delay(1000);
248     gsmSerial.print(smsText);
249     delay(1000);
250     gsmSerial.println((char)26);
251     delay(2000);
252     while(gsmSerial.available() > 0)
253     {
254       c = gsmSerial.read();
255     }
256   }
257 }
```

Abbildungsverzeichnis

Listings