GSM-Datenübertragung beim Messen und Steuern

Dr. Claus Kühnel

Der Datenübertragung kommt bei örtlich voneinander getrennt angeordneten Komponenten in der Mess-, Steuerungs- und Regelungstechnik eine wichtige Bedeutung zu. Die drahtlose Kommunikation macht es möglich, nahezu beliebige Distanzen zu überbrücken und muss bauliche Maßnahmen nur in beschränktem Maße berücksichtigen.

Im folgenden Beitrag soll gezeigt werden, wie über das Handy verschickte SMS zur drahtlosen Datenübertragung verwendet werden können.

Beispiele für diese Technik sind im Alltag stark verbreitet. Das Hochschalten der Heizung vor der Rückkehr aus den Ferien (im Winter) oder eine Mitteilung über den Restbestand im hauseigenen Öltank kann das Leben angenehmer gestalten.

1. Einige Grundbegriffe zu GSM und SMS

GSM steht für "Global System for Mobile Communications" und bezeichnet damit die gesamte Infrastruktur, die für eine Kommunikation von Mobilfunktelefonen (Handys) erforderlich ist.

Das Hauptanwendung für das GSM ist die Telefonie und zunehmend auch die Datenübertragung.

Die Entwicklung des GSM begann 1982. Die ersten beiden Phasen, die auch den Kurzmitteilungsdienst SMS (Short Message Service) umfassten, sind abgeschlossen. In der zur Zeit laufenden Phase 2+ werden die Datenfunkdienste GPRS (General Packet Radio Service) und HSCSD (High Speed Circuit Switched Data) eingeführt.

SMS steht für Short Message Service (Kurzmitteilungsdienst), der das Versenden von Kurznachrichten über das GSM-Netz bezeichnet. In der Umgangssprache wird nicht zwischen dem Service selbst und der versendeten Mitteilung unterschieden – beides wird als SMS bezeichnet.

Mit dem Kurzmitteilungsdienst SMS lassen sich Nachrichten zwischen Handys, GSM-Modems, Faxen und eMail-Systemen austauschen.

Unterschieden werden zwei SMS-Übertragungsarten:

SMS/PP (Short Message Service/Point-to-Point)	**SMSCB** (Short Message Service Cell Brodcast)
Kurzmitteilungen mit max. 160 Zeichen (zu je 7 Bit entsprechen 140 Byte) werden von einem GSM-Endgerät über die Kurzmitteilungszentrale SMS-SC (Short Message Service – Service Center) des Netzbetreibers zu einem GSM-Endgerät übertragen. Neben Textmitteilungen können ebenso 8-Bit-Binärdaten übertragen werden. Die SMS wird im SMS-SC zwischengespeichert. Die GSM-Phase 2+ sieht die Möglichkeit der Verkettung von SMS vor, so dass dadurch überlange SMS verschickt werden können.	Der SMSCB-Dienst ermöglicht das Versenden von Rundsprüchen an alle eingeschalteten Mobiltelefone in einem bestimmten Gebiet. Eine Rundspruch-nachricht umfasst max. 93 Byte. Die Einführung des SMSCB-Dienstes obliegt den GSM-Netzbetreibern und ist nicht zwingend.

Für unsere Belange hier ist nur der Dienst SMS/PP von Interesse, da mit diesem Kurzmitteilungen zwischen GSM-Endgeräten direkt ausgetauscht werden können.

2. SMS-Protokolle

Der ETSI-Standard GSM 07.05 [4] definiert die folgenden Protokolle zur Benutzung der SMS-Funktionen:

- Blockmodus
- Textmodus
- PDU-Modus

Gemäss [5] ist nur bei einigen wenigen GSM-Endgeräten der Blockmodus implementiert und soll deshalb hier nicht weiter betrachtet werden.

Beim Textmodus werden die zu übertragenden Zeichen im 7-Bit-ASCII-Code dem GSM-Endgerät übergeben. Nach vorangegangener Initialisierung kann eine SMS im Textmodus sehr einfach durch den folgenden Befehl direkt verschickt werden:

```
AT+CMGS= "+49xxxx",145<CR>Mitteilungstext<^Z>
```

Im sogenannten PDU-Modus werden die zu übertragenden Daten im Hex-Format eingegeben und (normalerweise) platzsparend zusammengeschoben. Das macht es leider etwas mühsam und unleserlich. Beim PDU-Modus sieht ein solcher Direktversand dann wesentlich kryptischer aus:

```
AT+CMGS=28
> 07919471227230332500 0C91947112325476000008D4F29C4E2FE3E9
```

Im PDU-Modus besteht das Protokoll aus einem Steuerungs- und Informationsteil sowie einem Datenteil (User Data UD).

Eine zu versendende SMS setzt sich aus den folgenden Bestandteilen zusammen:

SMS-SUBMIT-TPDU (Mobile Originated)								
SCA *)	FO	MR	DA	PID	DCS	VP	UDL	UD

Eine empfangene SMS ist ähnlich aufgebaut:

SMS-DELIVER-TPDU (Mobile Terminated)							
SCA	FO	OA	PID	DCS	SCTS	UDL	UD

Hierbei bedeuten:

SCA	Service Center Address *) teilweise optional
FO	First Octett (erstes Byte nach SCA)
MR	Message Reference (fortlaufende Zahl)
DA	Destination Address (Empfängeradresse)
PID	Protocol Identifier (Art der Nachricht)
DCS	Data Coding Scheme (Art der Datencodierung)
VP	Validity Protocol (Gültigkeitsdauer der SMS)
OA	Originator Address (Absenderadresse)
SCTS	Service Center Timestamp (Zeitstempel der Kurzmitteilungszentrale)
UDL	User Data Length (Länge der Kurzmitteilung)
UD	User Data (Daten der Kurzmitteilung)

Ein Beispiel soll das Protokollhandling verdeutlichen. Will man eine SMS mit dem Inhalt "Testmitteilung" an die Nummer +41-79-3960801 senden, dann nimmt die PDU den Wert "0001000B911467930608F100000ED4F29CDE4ED3E9E534BBEE3E03" an, der wie folgt in das Protokoll SMS-SUBMIT-TPDU umgesetzt wird. Hierbei ist vorausgesetzt, dass die Nummer des SMS-SC im GSM-Endgerät gespeichert ist und deren Angabe dadurch nicht explizit notwendig ist.

	7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	0	0	Length: Länge SCA
2									Tosca: SCA-Nummerierungstyp
3									Adress: Nummer des SMS-SC
12									
1	0	0	0	0	0	0	0	1	FO
1	0	0	0	0	0	0	0	0	MR
1	0	0	0	0	1	0	1	1	DA (max. 12 Oktette)
2	1	0	0	1	0	0	0	1	
3	0	0	0	1	0	1	0	0	
8	1	1	1	1	0	0	0	1	
1	0	0	0	0	0	0	0	0	PID

1	0	0	0	0	0	0	0	0	DCS
1									VP (0,1 oder 7 Oktette
2									
7									
1	0	0	0	0	1	1	1	0	UDL
1	1	1	0	0	0	1	0	0	UD (max. 140 Oktette)
2	1	1	1	1	0	0	1	0	
12	0	0	1	1	1	1	1	0	
13	0	0	0	0	0	0	1	1	

Um die Verwirrung noch etwas größer zu machen, wird im GSM-Jargon nicht von Bytes sondern von Oktetten gesprochen.

Unsere variablen Daten (Nummer des Empfängers und Mitteilungstext) sind unter DA und UDL/UD abgespeichert.

Empfängt man diese Mitteilung von einem anderen GSM-Endgerät, dann stellt das SMS-SC noch einen Zeitstempel bei und die empfangene PDU nimmt folgende Gestalt an: "07911467950800F0040B911467930608F10000405062615212080ED4F29CDE4ED3E9E5 34BBEE3E03".

Das Protokoll SMS-DELIVER-TPDU (für die empfangene SMS) unterscheidet sich etwas von der SMS-SUBMIT-TPDU. Die empfangenen Daten sind wieder eingetragen:

	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	
1	0	0	0	0	0	1	1	1	Length: Länge SCA
2	1	0	0	1	0	0	0	1	Tosca: SCA-Nummerierungstyp
3	0	0	0	1	0	1	0	0	Adress: Nummer des SMS-SC
8	1	1	1	1	0	0	0	0	
1	0	0	0	0	0	1	0	0	FO
1	0	0	0	0	1	0	1	1	OA (max. 12 Oktette)
2	1	0	0	1	0	0	0	1	
3	0	0	0	1	0	1	0	0	
8	1	1	1	1	0	0	0	1	
1	0	0	0	0	0	0	0	0	PID
1	0	0	0	0	0	0	0	0	DCS
1	0	1	0	0	0	0	0	0	SCTS (0,1 oder 7 Oktette
2	0	1	0	1	0	0	0	0	
7	0	0	0	0	1	0	0	0	
1	0	0	0	0	1	1	1	0	UDL
1	1	1	0	0	0	1	0	0	UD (max. 140 Oktette)
2	1	1	1	1	0	0	1	0	
12	0	0	1	1	1	1	1	0	
13	0	0	0	0	0	0	1	1	

Die Protokolle SMS-SUBMIT-TPDU (für eine zu versendende SMS) und MS-DELIVER-TPDU (für eine empfangene SMS) zeigen deutlich den erforderlichen Implementierungsaufwand für einen angeschlossenen Mikrocontroller.

Der Aufwand rührt im Wesentlichen daher, dass die im 7-Bit-ASCII-Format vorliegenden Textmitteilungen nach dem folgenden Schema in die zu versendenden Oktette zusammengeschoben werden (Abbildung 1). In [7] wird der Inhalt einer SMS im PDU-Modus hinreichend beschrieben, so dass auf komplexere Darstellungen erst mal verzichtet werden kann.

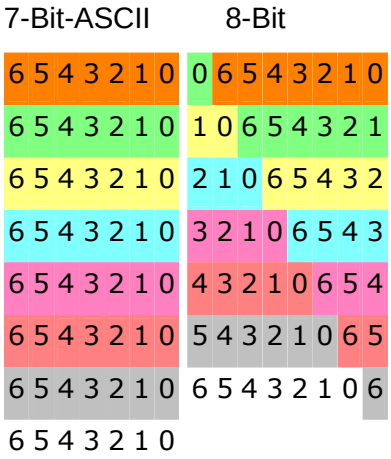

Abbildung 1 Datenkonvertierung Text-PDU

Scheut man den erforderlichen Programmieraufwand und reichen 140 zu übertragende Zeichen aus, dann kann man anstelle des standardmäßigen 7-Bit-ASCII Formats auch mit 8-Bit Daten arbeiten. Mit dem Programm PDUSpy wollen wir uns die Codierung der User Daten, also unseren zu sendenden Text noch näher ansehen.

Wie aus den beiden Beispielen am Anfang dieses Abschnitts zu erahnen war, erfolgt die Kommunikation mit dem GSM-Endgerät über sogenannte AT+C-Kommandos, eine Erweiterung der bei Modems eingesetzten AT-Kommandos. Damit das Verhalten des GSM-Endgerätes transparenter wird, wollen wir uns in einem späteren Abschnitt mit diesen Kommandos befassen.

3. PDUSpy

Das Programm PDUSpy ist ein sehr nützliches Tool, um die im GSM-Endgerät oder in der SIM-Karte gespeicherten Kurzmitteilungen auszulesen und zu decodieren. Außerdem können Kurzmitteilungen mit allen Optionen erzeugt, verschickt und im GSM-Endgerät oder in der SIM-Karte gespeichert werden.

PDUSpy weist u.a. die folgenden Merkmale auf:

- funktioniert mit allen gängigen Telefonen, die AT-Befehle unterstützen
- jedes Bit des Nachrichtenheaders kann spezifikationsgemäß beeinflusst werden
- dynamische SMSC- und Zieladressenliste
- kann Unicode-Nachrichten decodieren und verschicken
- kann Nachrichten mit UDH erzeugen und verschicken
- beherrscht fast alle EMS-Header
- erzeugt auch SMS-COMMAND PDUs

Von der Website [www.nobbi.com/download/pduspy.zip] kann das Programm heruntergeladen werden.

Da wir im Weiteren sowohl mit dem standardmäßigen 7-Bit-ASCII Format also auch mit 8-Bit Daten arbeiten wollen, sind in den folgenden Screenshots die Schritte zum Versand und zum Empfang einer SMS in diesen Formaten gegenübergestellt. Die linke Spalte zeigt das Vorgehen bei Verwendung des 7-Bit-ASCII Formats, die rechte Spalte das Vorgehen bei Verwendung der 8-Bit Daten.

Gleichzeitig wird die Arbeit mit dem Programm PDUSpy deutlich. Wir verwenden hier allerdings nur eine geringe Menge der zur Verfügung stehenden Möglichkeiten.

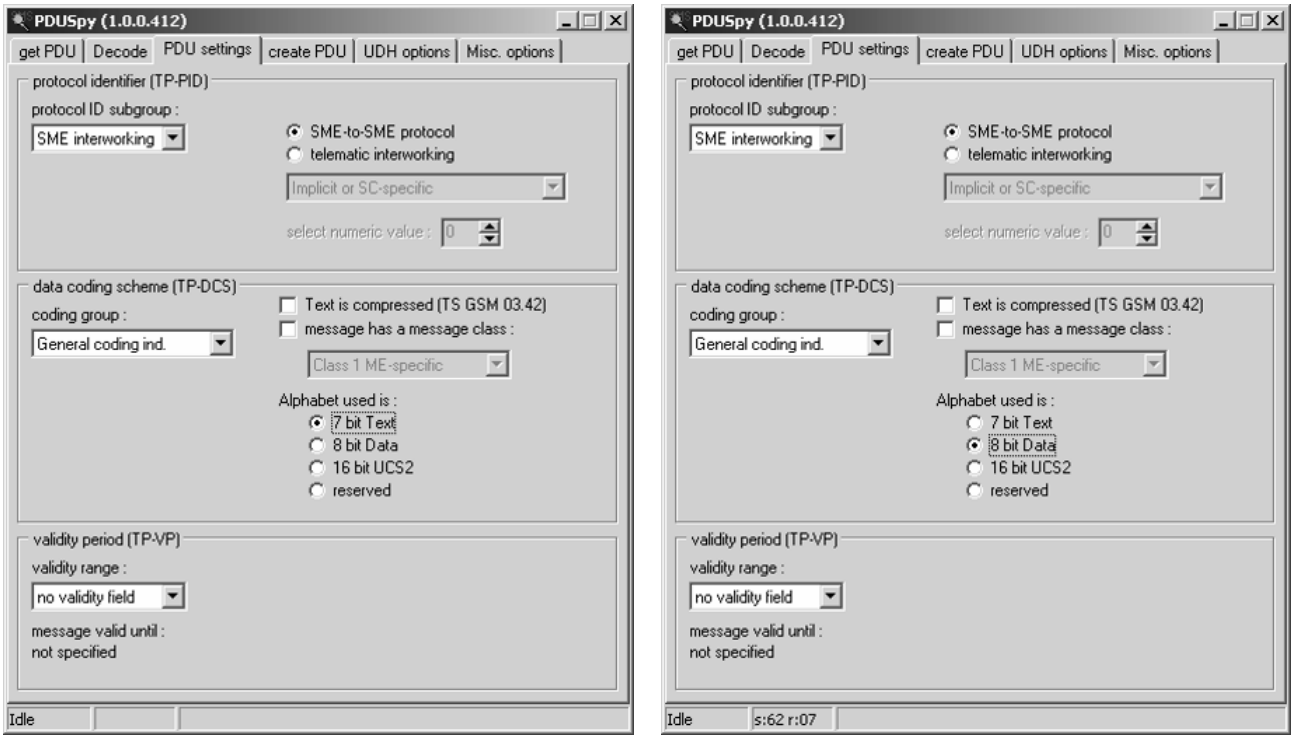

Abbildung 2 Einstellung des Datenformats in PDU Settings

Bis auf das verwendete Datenformat werden die Standardeinstellungen beibehalten. Eine Gültigkeitsdauer der SMS (TP-VP) wird nicht angegeben.

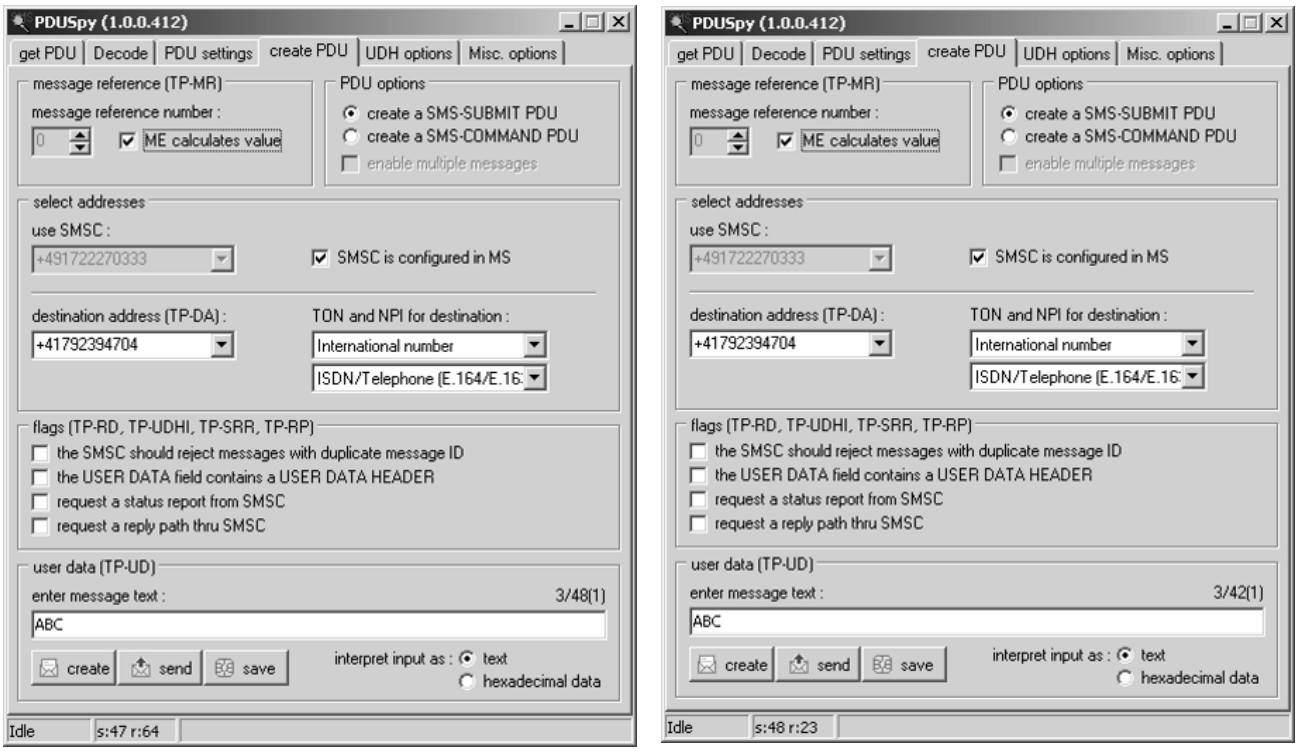

Abbildung 3 Erzeugen der PDU nach Eingabe von Telefonnummer und Mitteilungstext

Die Telefonnummer des SMS-SC ist im GSM-Endgerät gespeichert. Eingegeben werden nur die Empfängernummer in internationalem Format und die zu übertragende Mitteilung (hier "ABC").

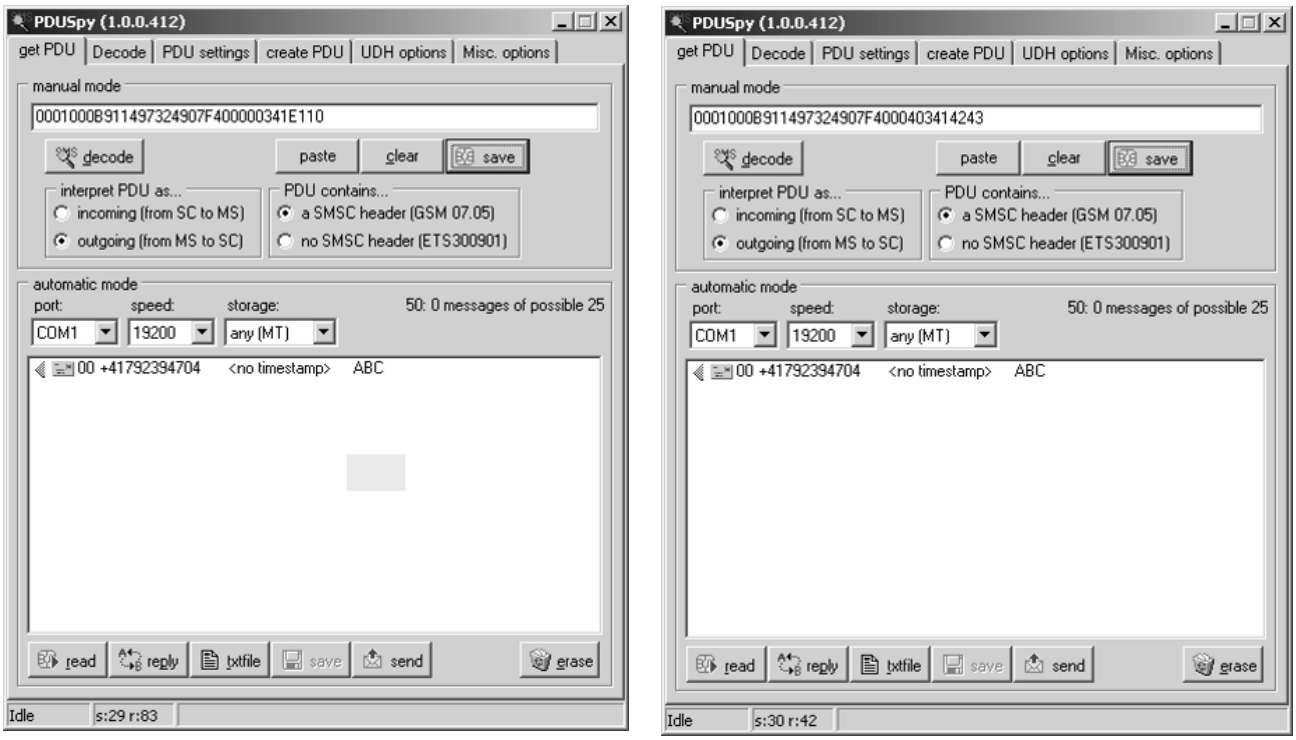

Abbildung 4 Speichern der erzeugten SMS und anschließendes Versenden

Die erzeugte SMS wird an der Position 00 im Speicher abgelegt und kann anschließend versendet werden. Die PDU ist im Feld "Manual Mode" dargestellt.

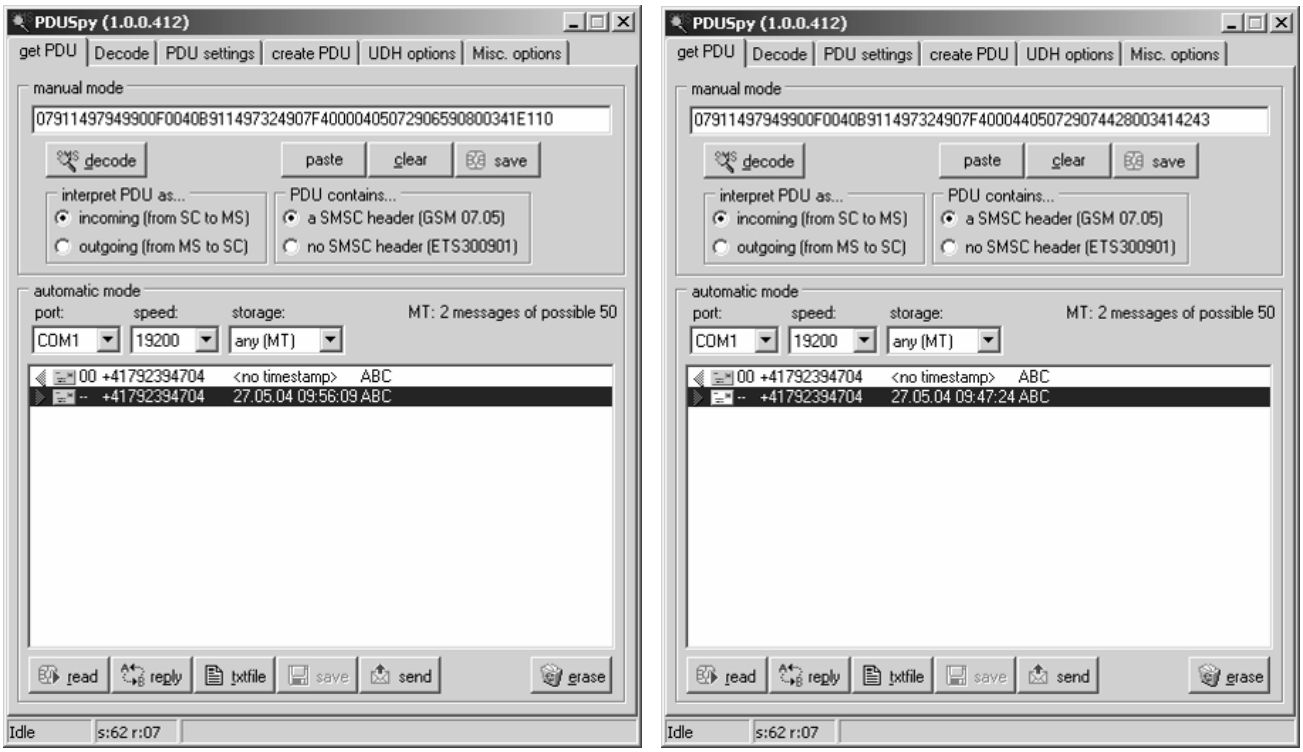

Abbildung 5 Empfang der SMS

Da die Empfangsnummer identisch zur Sendenummer war, wird die versendete SMS nach kurzer Zeit (mit einem Zeitstempel der SMS-SC versehen) wieder empfangen. Die erweiterte PDU ist deutlich erkennbar.

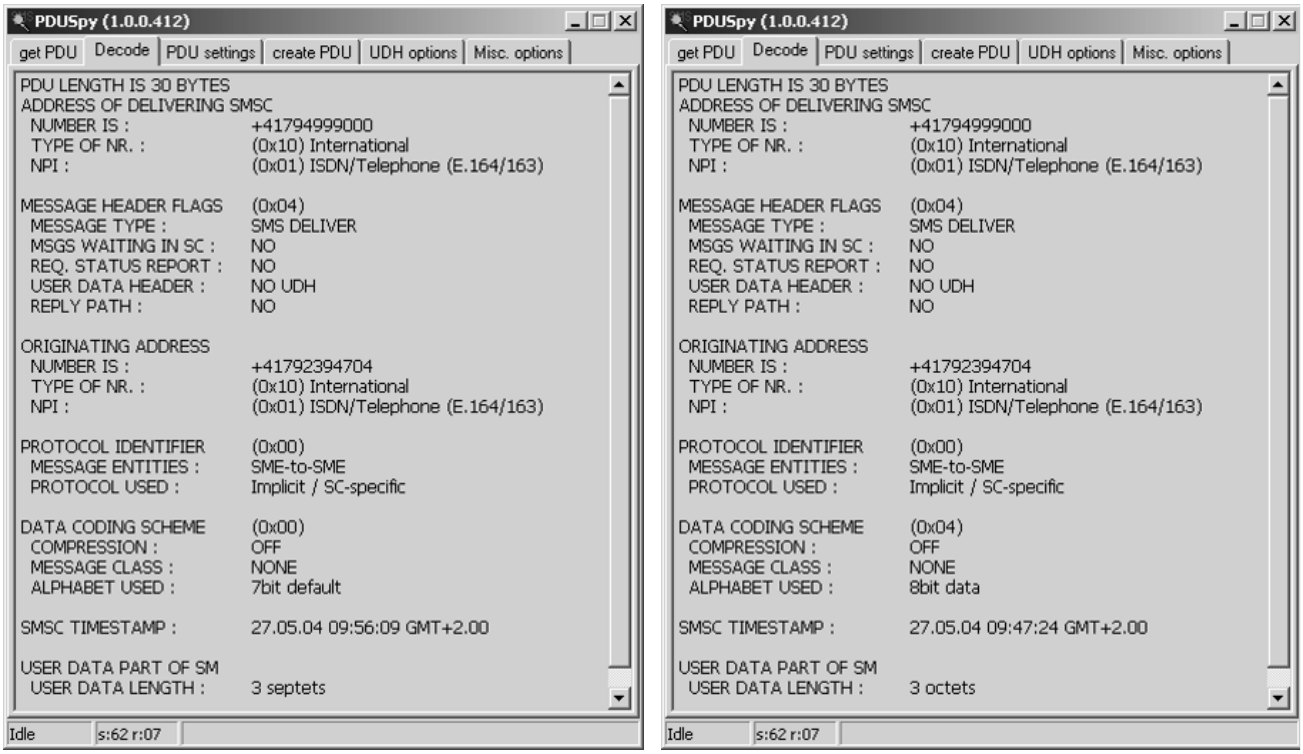

Abbildung 6 Decodierung der empfangenen SMS

4. Kommunikation über AT+C-Kommandos

Nachdem mit Hilfe des Programms PDUSpy geklärt werden konnte, wie die zu übertragende Mitteilung verpackt wird, betrachten wir die Kommunikation mit dem GSM-Endgerät. Wir verwenden hier ein Mobiltelefon Siemens S45 und verbinden es über ein passendes Datenkabel mit einem freien COM-Port des Entwicklungs-PCs.

Leider verhalten sich die Geräte unterschiedlicher Hersteller nicht in jedem Fall identisch, so dass ein Nachschlagen in der Herstellerdokumentation in jedem Fall hilfreich ist. Unter [5] kann man Informationen zu den von verschiedenen GSM-Endgeräten unterstützten AT+C-Kommandos finden. Eine Zusammenstellung der wichtigsten AT+C-Kommandos ist am Ende des Beitrags zu finden.

Das Terminalprogramm ist auf die Parameter des GSM-Modems (vor allem die Baudrate) einzustellen. Das Senden der Zeichenfolge "AT" gefolgt von CR sollte nun vom GSM-Modem mit OK beantwortet werden.

Für den Datenaustausch mit einem GSM-Endgerät gibt es drei verschiedene Befehlsformen, die am Beispiel der Pinnummer erläutert werden:

1. Parametereingabe AT+CPIN=“1234“
2. Abfrage der Einstellung AT+CPIN? mit Antwort +CPIN: READY
3. Abfrage möglicher Einstellungen AT+CPIN=? mit der Antwort +CPIN: OK

Die in Abbildung 7 dargestellten AT+C-Kommandos verwenden diese drei Befehlsformen.

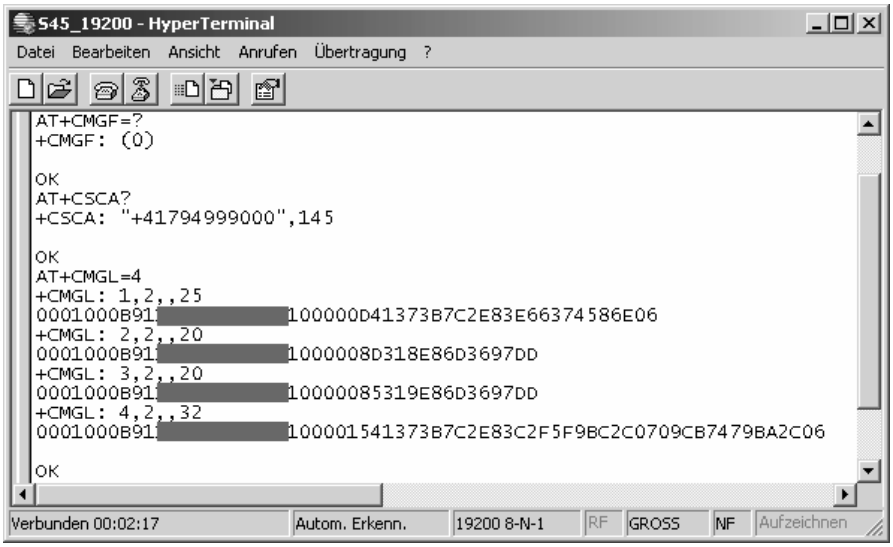

Abbildung 7 Befehlsformen der AT+C-Kommandos

$Die Tabelle erläutert, was geschehen ist.

Befehl	*Bedeutung*	*Antwort*
AT+CMGF=?	Abfrage der möglichen SMS-Formate	+CMGF: (0)
AT+CSCA?	Abfrage der Telefonnummer des SMS-Servicecenters	+CSCA: „+41794999000“,145
AT+CMGL=4	Ausgabe aller SMS	+CMGL: 1,2,,22....

Der Befehl AT+CMGF=? ermittelt die beim S45 möglichen SMS-Formate. Die Antwort war ernüchternd, denn dieses zumindest zum Zeitpunkt der Markteinführung doch recht komfortable Handy nötigt seinem Besitzer den PDU-Modus ab. Den einfachen Textmodus versteht es nicht.

Der Befehl AT+CSCA? fragt nach der Telefonnummer des SMS-Servicecenters. Gelistet wurde hier die Nummer des (Schweizer) Swisscom-Servicecenters im internationalen Format (ausgewiesen durch den Parameter 145). Beim Betrieb eines Handys sollte man ohnehin vom internationalen Format der Telefonnummern Gebrauch machen, also stellt das keine Besonderheit dar.

Mit dem Befehl AT+CMGL=4 rufen wir nun alle abgespeicherten SMS vom GSM-Endgerät ab.

Anhand eines Hyperterminal-Mitschnitts wollen wir uns die Antworten des Siemens S45 auf verschiedene AT+C-Kommandos zur Parameterabfrage verdeutlichen (Abbildung 8).

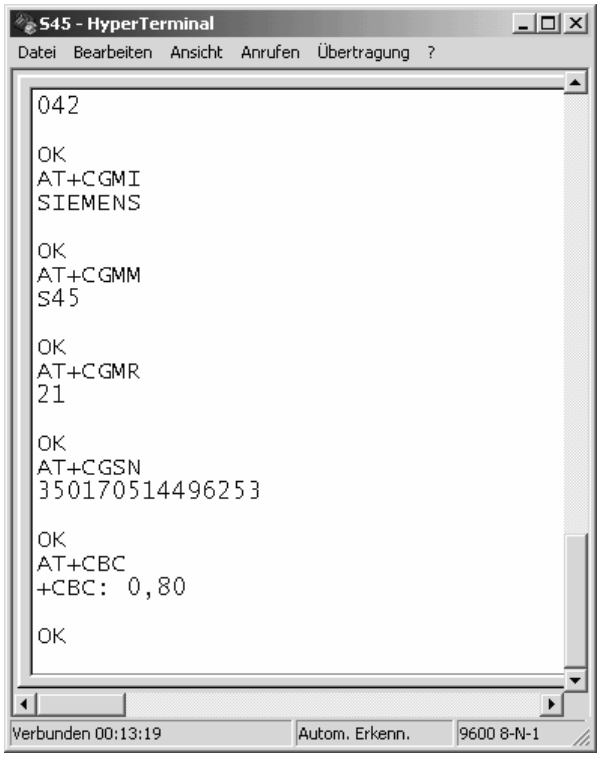

Abbildung 8 Parameterabfrage

Die folgende Tabelle listet die Abfragen und ihre Bedeutung:

Befehl	*Bedeutung*	*Antwort*
AT+CGMI	Abfrage des Herstellers	SIEMENS
AT+CGMM	Abfrage des Modells	S45
AT+CGMR	Abfrage der Softwareversion	21
AT+CGSN	Abfrage der IMEI	350170514496253
AT+CBC	Abfrage des Batteriezustands	0,80

Wie wir feststellen mussten, arbeitet das hier eingesetzte Siemens S45 nur im umständlichen PDU-Modus.

Welche Möglichkeiten es zum Versenden oder Empfangen von SMS gibt, ohne die umständliche Codierung der PDU in einem Anwendungsprogramm vornehmen zu müssen, zeigen die nächsten Beispiele.

4.1. Programmierung über die Handy-Tastatur

Bei Verwendung eine Handys als GSM-Endgerät kann man die zu übermittelnden SMS über die Tastatur eintippen und im internen Speicher ablegen. Dann ist über ein AT+C-Kommando nur noch die gewünschte Mitteilung zu selektieren und an den gewünschten Empfänger zu versenden. Für den Empfang können die zugelassenen Mitteilungen auf die gleiche Weise erzeugt werden. Die empfangene SMS wird dann mit den abgespeicherten Mustern vergleichen.

Bevor wir mit Hilfe der AT+C-Kommandos Mitteilungen versenden bzw. empfangen, wollen wir noch einen Blick auf die verschiedenen Speicher des GSM-Endgeräts werfen. Unterschieden werden der Speicher im Mobilteil (ME) und der Speicher auf der SIM-Karte

(SM). Die Kombination aus ME und SM wird als MT bezeichnet. Der vorhandene Speicher wird in drei Bereiche organisiert: aus mem1 werden Mitteilungen gelesen und gelöscht, in mem2 werden Mitteilungen geschrieben und daraus versendet und in mem3 werden Nachrichten abgelegt, wenn keine Umleitung zum Host gesetzt ist.

Mit dem Kommando AT+CPMS=? kann ausgelesen werden, welche Speicher unterstützt werden.

AT+CPMS=?

+CPMS: ("MT","SM","ME"),("MT","SM","ME"),("MT","SM","ME")

OK

Beim Siemens S45 kann auf jeden der Bereiche zugegriffen werden. Wir wollen den Speicher so zuweisen, dass aus dem gesamten Speicher (MT) gelesen und in den Speicher der SIM-Karte (SM) geschrieben wird.

AT+CPMS="MT","SM"

+CPMS: 4,50,3,25,3,25

OK

Die Antwort des Siemens S45 zeigt, dass der gesamte Speicher 50 Mitteilungen aufnehmen kann und vier Mitteilungen im Speicher vorhanden sind. Auf der SIM-Karte steht eine Kapazität von 25 Mitteilungen zur Verfügung, wovon 3 belegt sind.

Das Programm PDUSpy bringt auch hier Licht in das Dunkel (Abbildung 9).

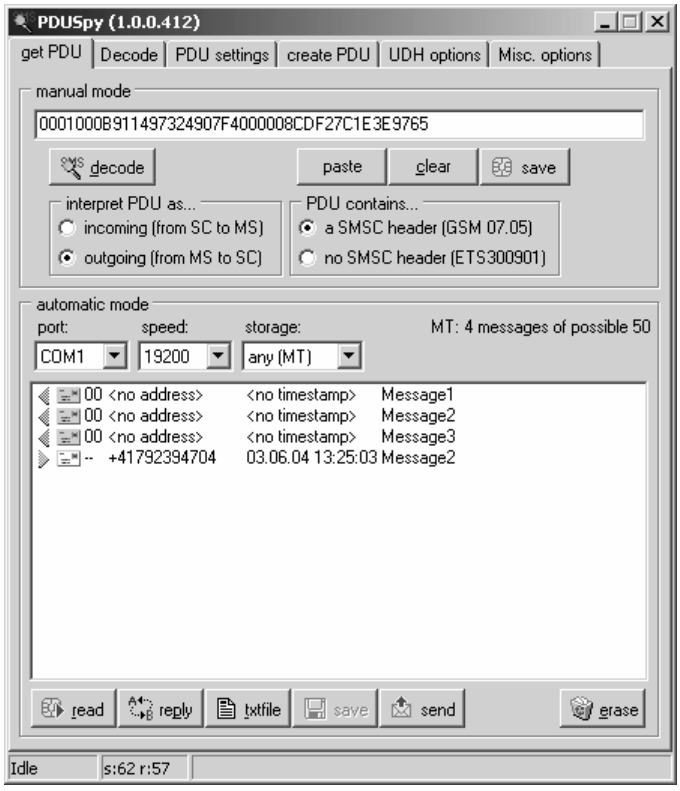

Abbildung 9 Abgespeicherte Mitteilungen

Wie Abbildung 9 ausweist befinden sich im Speicher MT insgesamt 4 Mitteilungen. Die ersten drei Meldungen wurden über die Tastatur eingegeben und auf der SIM-Karte abgespeichert. Während die vierte Meldung empfangen wurde und im Speicher ME abgelegt ist. Diese Feststellung wollen wir überprüfen.

Hierzu wählen wir zuerst den Speicher der SIM-Karte und lesen anschliessend alle enthaltenen Mitteilungen aus:

AT+CPMS="SM"

+CPMS: 3,25,3,25,3,25

OK

AT+CMGL=4

+CMGL: 1,2,,15

07911497949900F0110000810000A908CDF27C1E3E9763

+CMGL: 2,2,,15

07911497949900F0110000810000A908CDF27C1E3E9765

+CMGL: 3,2,,15

07911497949900F0110000810000A908CDF27C1E3E9767

OK

Wir finden drei Mitteilungen mit nahezu identischem Dateninhalt. Der einzige Unterschied bestand in der dem Text "Message" folgenden Ziffer.

Nach Umschalten auf den Speicher ME können wir wieder alle dort vorhandenen Mitteilungen abfragen.

AT+CPMS="ME"

+CPMS: 1,25,3,25,3,25

OK

AT+CMGL=4

+CMGL: 1,1,,26

07911497949900F0040B911497324907F400004060303152308008CDF27C1E3E9765

OK

Erwartungsgemäß finden wir eine Mitteilung, die vom Dateninhalt her der Mitteilung "Message2" entspricht.

Mit den beschriebenen AT+C-Kommandos haben wir Speicher zugeordnet und das Vorhandensein von Mitteilungen überprüft. Das Versenden einer abgespeicherten Mitteilung ist nun denkbar einfach. Es reichen die Angabe der betreffenden Speicherstelle und der Empfängernummer und die Mitteilung wird versandt. Im folgenden Beispiel wird die Mitteilung "Message3" an die im internationalen Format angegebene Nummer versendet. Der Parameter 145 verweist auf das internationale Format der Telefonnummer.

AT+CMSS=3,"+41792394704",145

+CMSS: 224

OK

Durch die Rückgabe einer Message Reference (verschieden von ERROR) wird angezeigt, dass der Versand der SMS erfolgreich war.

Beim Empfang von Kurzmitteilungen kann entschieden werden, wie dies dem angeschlossenen Host mitgeteilt wird. Wir wollen hier von der Default-Einstellung des Siemens S45 ausgehen und eine empfangene SMS in jedem Fall zwischenspeichern (Mode=0) und nach dem Lesen löschen. Die hierfür erforderliche Einstellung kann folgendermassen abgefragt werden.

AT+CNMI?

+CNMI: 0,0,0,0,1

OK

Nun geht es bei der Überwachung des Empfangs von Kurzmitteilungen nur noch darum, den Speicher ME in geeigneten Abständen abzufragen, die Mitteilung(en) zu lesen und anschliessend zu löschen.

Hier erfolgt nun nicht mehr die Abfrage nach allen abgespeicherten Mitteilungen (AT+CMGL=4) sondern nur noch nach den bislang ungelesenen.

AT+CMGL=0

+CMGL: 1,0,,26

07911497949900F0040B911497324907F4000040603051005480 08CDF27C1E3E9767

OK

Die angegebene Abfrage zeigt, dass auf Position 1 eine ungelesene Mitteilung wartet. Der Status "ungelesen" (0) wird mit dieser Abfrage aber sofort auf "gelesen" (1) geändert. Wir verifizieren das mit der allgemeinen Abfrage (4).

AT+CMGL=4

+CMGL: 1,1,,26

07911497949900F0040B911497324907F4000040603051005480 08CDF27C1E3E9767

OK

Nun kann die empfangene Mitteilung noch vom Host gelesen (AT+CMGR=index) und anschliessend gelöscht (AT+CMGD=index) werden.

AT+CMGR=1

+CMGR: 1,,26

07911497949900F0040B911497324907F4000040603051005480 08CDF27C1E3E9767

OK

AT+CMGD=1

OK

Eine erneute Abfrage des Speichers ME bestätigt nun, dass keine Mitteilungen mehr im Speicher vorhanden sind.

AT+CMGL=4

OK

Durch Vergleich des Dateninhalt der empfangenen Mitteilung mit vorgegebenen Mustern kann auf das Decodieren der PDU verzichtet werden.

Gerade bei weniger leistungsfähigen Mikrocontrollern als Host kann diese Möglichkeit sehr willkommen sein [3]. Natürlich bleibt der beschrieben Modus auf das Versenden von Festtexten beschränkt.

4.2. PDU im 8-Bit Datenformat

Wollen wir nun die Einschränkung auf Festtexte fallen lassen und trotzdem die PDU-Codierung vermeiden, dann hilft uns das 8-Bit Datenformat anstelle der 7-Bit ASCII Daten weiter.

Abbildung 2 hatte uns schon die Vorkehrungen im Programm PDUSpy vorgestellt. Wir wollen den Aufbau einer zu versendenden Mitteilung anhand des folgenden Beispiels studieren. Aufgebaut werden die Mitteilungen ABC, ABCD und ABCDE, die jeweils an die Telefonnummer +41792394704 versendet werden sollen:

Mitteilung	*PDU*
ABC	0001000B911497324907F40004**03414243**
ABCD	0001000B911497324907F40004**0441424344**
ABCDE	0001000B911497324907F40004**054142434445**

Am Ende der erzeugten PDUs können wir nun den Mitteilungstext in hexadezimaler Darstellung erkennen. Eröffnet wird durch die Angabe eines Längenbytes gefolgt von den Zeichen der Mitteilung.

Erinnern wir uns an die Ausführungen in Abschnitt 2 und nehmen nun die PDU auseinander.

Die Adresse des Servicecenters SCA ist im GSM-Endgerät gespeichert, weshalb der Länge der SCA mit Null angegeben wird. Wir wollen eine Kurzmitteilung versenden, also ist der Nachrichtentyp SMS-SUBMIT und das Byte FO nimmt den Wert 01 an. Die Message Reference Number wird durch das GSM-Endgerät selbst erzeugt und wird deshalb im Byte MR mit 00 vorgegeben. Abbildung 10 zeigt die weiteren Parameter des Message Headers.

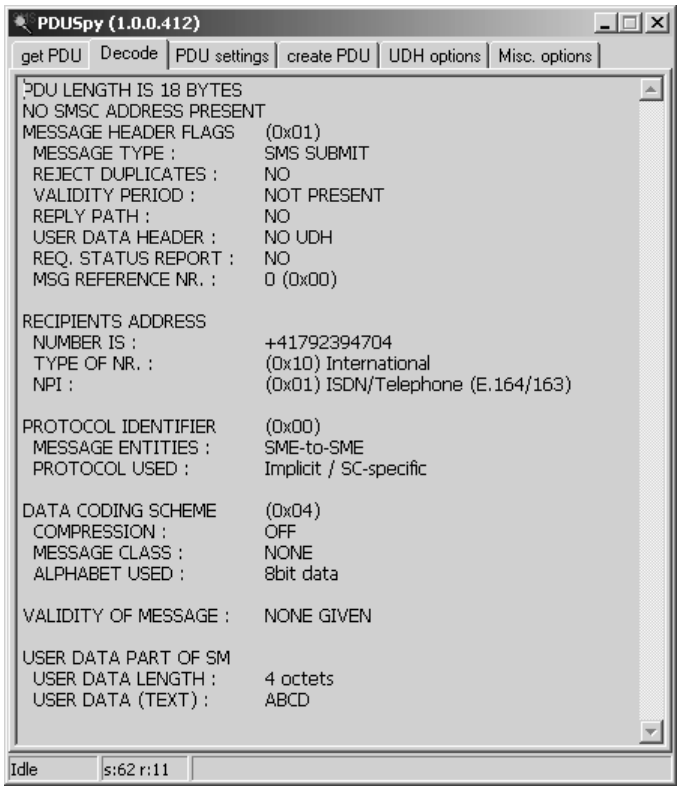

Abbildung 10 Zu versendende Mitteilung decodiert

Nun folgt die Angabe der Telefonnummer des Empfängers nach dem bereits beschriebenen Mechanismus (Adresslängenfeld (hier 0B), Adresstypfeld (hier 91), Telefonnummer (Halboktette vertauscht)).

An die letzte Stelle der Telefonnummer schliesst sich der Protokollidentifier PID an. Für den Austausch von SMS wird hier der Wert 00 stehen.

In welcher Form die Mittelungsdaten folgen ist im Data Coding Scheme DCS angegeben. Der Wert 04 deklariert die folgenden Daten als 8-Bit codiert.

Weiter folgen nun nur noch die Zeichen des zu übermittelnden Textes durch ein Längenbyte angeführt.

Die folgende, tabellarische Zusammenstellung zeigt die erläuterten Bestandteile der zu versendenden Kurzmitteilung in kompakter Form.

HEX		*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	
00	*1*	0	0	0	0	0	0	0	0	Length: Länge SCA
01	*1*	0	0	0	0	0	0	0	1	FO
00	*1*	0	0	0	0	0	0	0	0	MR
0B	*1*	0	0	0	0	1	0	1	1	DA (max. 12 Oktette)
91	*2*	1	0	0	1	0	0	0	1	
14	*3*	0	0	0	1	0	1	0	0	
F4	*8*	1	1	1	1	1	0	0	0	
00	*1*	0	0	0	0	0	0	0	0	PID
04	*1*	0	0	0	0	0	1	0	0	DCS
04	*1*	0	0	0	0	0	4	0	0	UDL
41	*1*	1	0	0	0	0	0	0	1	UD (max. 140 Oktette)
42	*2*	1	0	0	0	0	0	1	0	
43	*3*	1	0	0	0	0	0	1	1	
44	*4*	1	0	0	0	0	1	0	0	

Die zu versendende Mitteilung kann nun im Speicher des GSM-Endgerätes abgelegt und versendet werden.

Ausserdem ist ein Direktversand über das Kommando AT+CMGS möglich. Dem Kommando AT+CMGS folgt eine mit CR abgeschlossene Längenangabe. Meldet sich das GSM-Endgerät mit dem Prompt ">" dann kann die TPDU eingegeben werden, die mit Ctrl-Z ($1A_H$) abzuschliessen ist.

Nach erfolgreicher Datenübernahme meldet das GSM-Endgerät entweder +CMGS: xxx xxx bezeichnet die Mitteilungsnummer). Anderenfalls mit +CMS ERROR: yyy (yyy bezeichnet einen Fehlercode)

AT+CMGS=17<CR>

> 0001000B911497324907F400040441424344<Ctrl-Z>

+CMGS: 229

OK

Wurde die erzeugte Kurzmitteilung (an die eigene Nummer) versandt, dann dauert es nicht lange und der Empfang der Mitteilung wird signalisiert. Abbildung 11 zeigt die beiden Mittelungen im Speicher des GSM-Endgerätes. Aus Abschnitt 4.1 wissen wir, dass eine empfangene Mitteilung u.a. auch einen Zeitstempel des SMS-Servicecenter trägt und deshalb etwas umfangreicher ausfällt, als die ursprünglich versandte Mitteilung. Abbildung 12 zeigt die empfangene Mitteilung in decodierter Form.

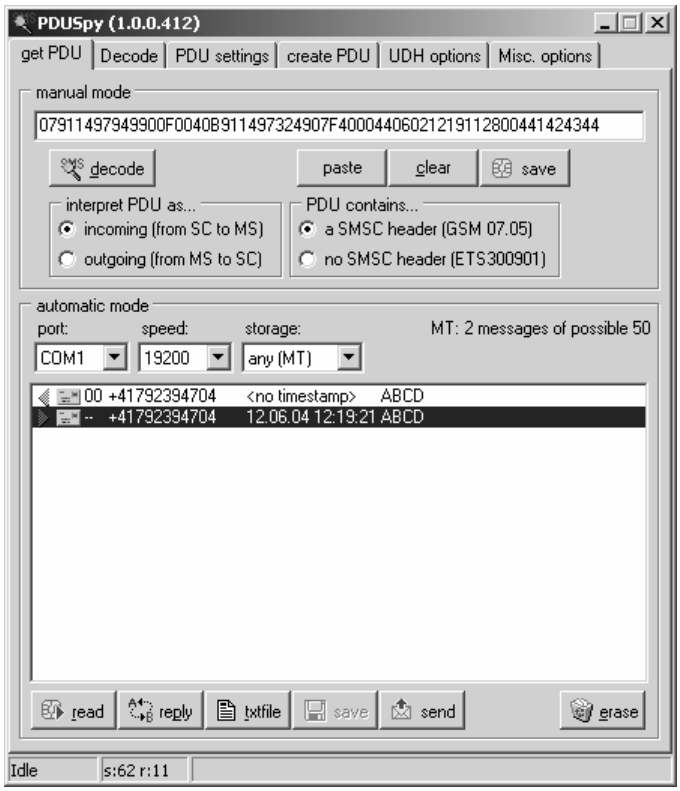

Abbildung 11 Mitteilungen im Speicher des GSM-Endgerätes

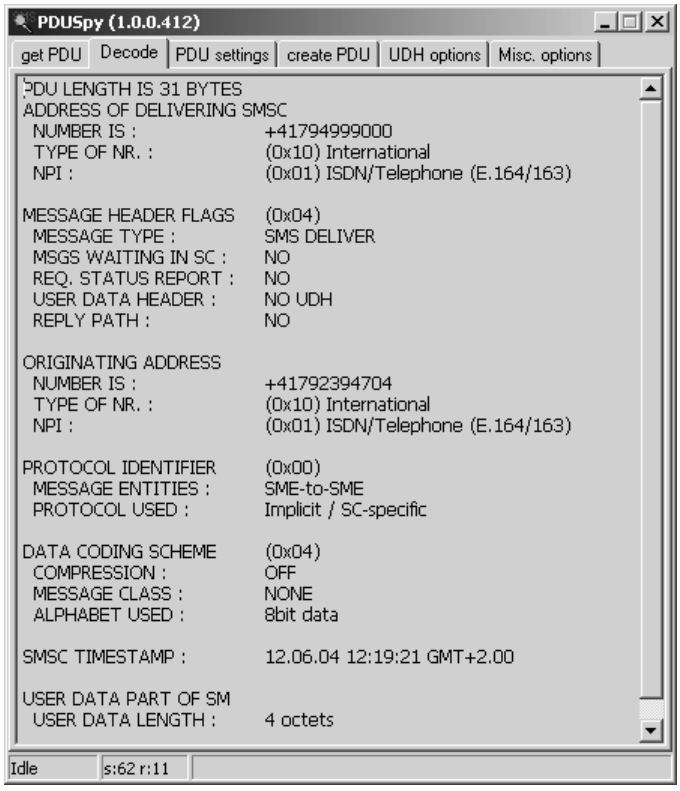

Abbildung 12 Empfangene Mitteilung decodiert

Um eine Vorlage für die spätere Decodierung im Anwendungsprogramm eines Mikrocontrollers zu schaffen, werden wir die empfangene Mitteilung noch detaillierter aufschlüsseln.

Die empfangene Mitteilung beginnt mit der Nummer der SMS-Servicecenters (Länge 7, internationales Format der Telefonnummer, Telefonnummer des SMS-SC +41794999000).

Das FO weist den Wert 04 auf, was darauf hinweist, dass keine weiteren Mitteilungen im SMS-SC warten.

Dem FO folgt die Nummer des Empfängers wieder im Format Adresslängenfeld, Adresstypenfeld und Adressfeld. Der Wert im Adresslängenfeld beschreibt die Anzahl der Ziffern der Telefonnummer (hier $0B_H$). Der Adresstyp 91_H weist wieder auf die das internationale Format der Telefonnummer hin.

Die Oktette PID und DCS sind identisch zu versendeten Mitteilung. Ihnen folgt der Zeitstempel des SMS-SCs im folgenden Format (jeweils zwei Ziffern vertauscht).

Jahr	Monat	Tag	Stunde	Minute	Sekunde	Zeitzone
04	06	12	12	19	21	GMT+8/4

Die Zeitzone gibt die Differenz der Lokalzeit zu GMT in Viertelstunden an (der Wert 8 bedeutet also GMT+2 = MESZ).

Als letztes folgenden zu übermittelnden Daten im gleichen Format, wie schon bei der zu versendenden Kurzmitteilung.

Die folgende, tabellarische Zusammenstellung zeigt die erläuterten Bestandteile der zu versendenden Kurzmitteilung in kompakter Form.

HEX		*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	
07	*1*	0	0	0	0	0	1	1	1	Length: Länge SCA
91	*2*	1	0	0	1	0	0	0	1	Tosca: SCA-Nummerierungstyp
14	*3*	0	0	0	1	0	1	0	0	Adress: Nummer des SMS-SC
F0	*8*	1	1	1	1	0	0	0	0	
04	*1*	0	0	0	0	0	1	0	0	FO
0B	*1*	0	0	0	0	1	0	1	1	OA (max. 12 Oktette)
91	*2*	1	0	0	1	0	0	0	1	
14	*3*	0	0	0	1	0	1	0	0	
F4	*8*	1	1	1	1	0	1	0	0	
00	*1*	0	0	0	0	0	0	0	0	PID
04	*1*	0	0	0	0	0	0	0	0	DCS
40	*1*	0	1	0	0	0	0	0	0	SCTS (0,1 oder 7 Oktette
60	*2*	0	1	0	1	0	0	0	0	
80	*7*	1	0	0	0	0	0	0	0	
04	*1*	0	0	0	0	1	1	1	0	UDL
41	*1*	1	1	0	0	0	1	0	0	UD (max. 140 Oktette)
42	*2*	1	1	1	1	0	0	1	0	
43	*3*	0	0	1	1	1	1	1	0	
44	*4*	0	0	0	0	0	0	1	1	

Mit diesen Kenntnissen ausgerüstet können wir nun einen Mikrocontroller in die Lage versetzen, ausgelöst durch bestimmte Ereignisse Kurzmitteilungen zu versenden oder auf empfangene Kurzmitteilungen mit bestimmten Aktionen zu reagieren.

5. SMS in der Steuerungs- und Messtechnik

Im ersten Teil dieses Beitrag waren die allgemeinen Grundlagen, die SMS-Protokolle und die AT+C-Kommandos zur Kommunikation mit einem Datenendgerät vorgestellt worden. Jetzt wollen wir konkret werden und einen AVR-Mikrocontroller ins Spiel bringen.

5.1. Entwicklungsumgebung

Als Entwicklungsumgebung dient uns das STK500 von Atmel. Wir werden den ATmega8 für unsere Versuche einsetzen. Das STK500 unterstützt aber nahezu alle AVR-Mikrocontroller, so dass hier keine wesentlichen Einschränkungen zu erwarten sind. Abbildung 13 zeigt das STK500 mit den verwendeten Komponenten.

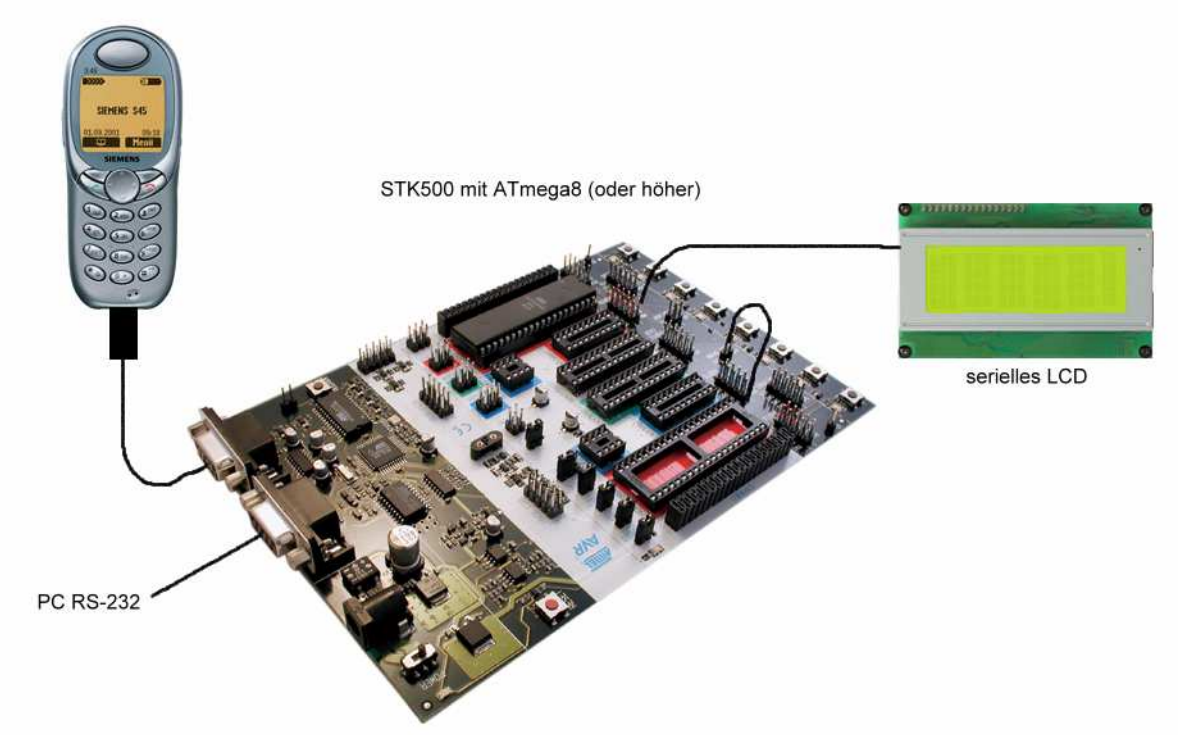

Abbildung 13 Entwicklungsumgebung mit STK500

Das STK500 weist zwei serielle Schnittstellen auf. Über die Schnittstelle RS232 CTRL wird der Entwicklungs-PC zum Programmdownload angeschlossen. An die Schnittstelle RS232 SPARE wird das verwendete Mobiltelefon angeschlossen. Beim hier verwendeten Siemens S45 einschließlich Original-Datenkabel war das Zwischenschalten eines Nullmodem-Kabel mit beidseitigem DSUB9-Stecker erforderlich. Zum Betrieb der Schnittstelle RS232 SPARE sind auf dem STK500 PD0 mit TX0 RS232 SPARE und PD1 mit RX0 RS232 SPARE zu verbinden. Ein serielles LCD wird an PB0 angeschlossen, um Ausgaben des betreffenden Programms darstellen zu können.

Für die Hardware gibt es praktisch keine Einschränkungen. Wichtig sind die serielle Schnittstelle zum Mobiltelefon und je nach Anwendung einige freie I/O-Leitungen.

Die im folgenden vorzustellenden Programmbeispiele wurden mit Hilfe von BASCOM-AVR programmiert. Details zu dieser Programmierumgebung sind auf der Website des Autors [www.ckuehnel.ch] und unter [1] zu finden.

5.2. Zustandsabhängiges Versenden von SMS

Im ersten Programmbeispiel sollen abhängig von bestimmten Zuständen der Überwachungsschaltung SMS versendet werden. Jeder SMS kann eine eigene Telefonnummer zugeordnet werden.

Vier mit Tastern versehene digitale Eingänge bestimmen hier die unterschiedlichen Zustände. In einer Überwachungsanlage werden die Zustände durch Kontakte, Stromschleifen oder das Überschreiten von Grenzwerten beeinflusst.

Abbildung 14 zeigt das resultierende Schaltbild unserer Überwachungsschaltung.

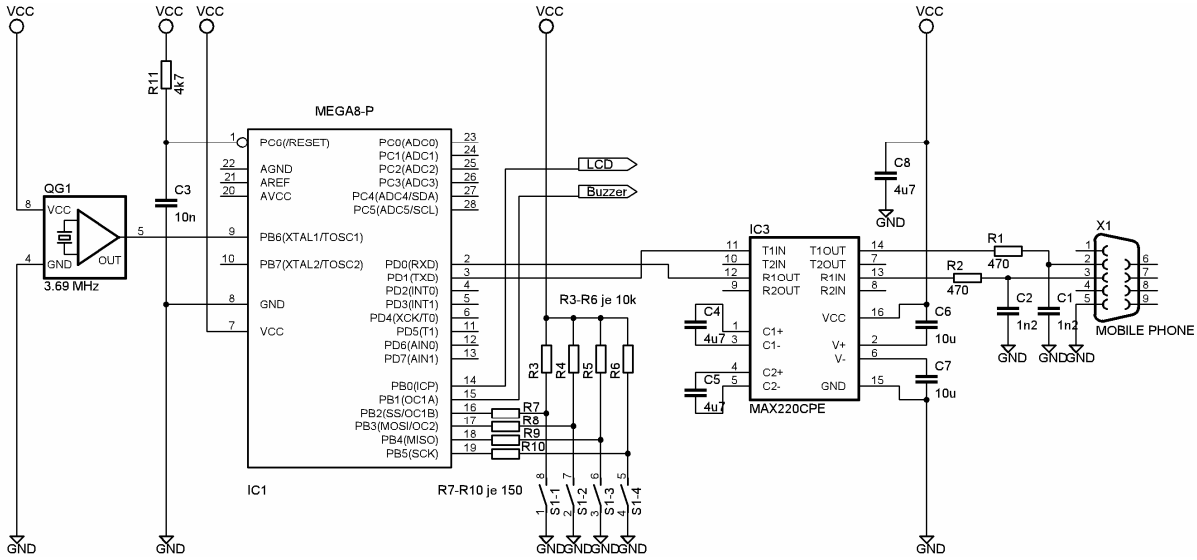

Abbildung 14 Überwachungsschaltung mit ATmega8

Die Überwachungsschaltung wird an einer Betriebsspannung VCC = 5 V betrieben. Der Schnittstellentreiber MAX220 erzeugt die Spannungspegel der RS232-Schnittstelle (± 12 V) mit Hilfe der integrierten Ladungspumpen, so dass der Datenaustausch mit dem Mobiltelefon zuverlässig funktioniert. Das RC-Glied am I/O-Pin PC6 bewirkt einen Reset beim Zuschalten der Betriebsspannung (Power-On Reset). Ein externer Oszillator erzeugt stabil den Takt mit 3,69 MHz. Die I/O-Pins von PortB bedienen das serielle LCD (PB0), einen Buzzer (PB1) zur Tonausgabe bzw. dienen der Abfrage der vier Taster (PB2 bis PB5). Da im Programm die internen PullUp-Widerstände aktiviert werden, könnte auf die Widerstandsbeschaltung der I/O-Pins PB2 bis PB5 verzichtet werden.

Listing 1 zeigt den Quelltext des Programmbeispiels send_sms.bas. Im Sekundentakt werden die vier Tasten nach einer veränderten Eingangsbelegung abgefragt und bei Zustandsänderung eine zugeordnete SMS verschickt.

```
$regfile = "m8def.dat"                    ' ATmega8
$crystal = 3690000                        ' für STK500
$baud = 19200
```

```
Const Instr = 254
Const Setcursor = 71                    ' [x] [y]
Const Cursorhome = 72
Const Cleardisplay = 88

Const Prebytes = "000100"
Const Toa = "91"
Const Pid = "00"
Const Dcs = "04"                        ' 8 bit data

Dim Num As String * 15
Dim Msg As String * 60
Dim Sms As String * 160
Dim Key As Byte                         ' number of key pressed
Dim Old_key As Byte
Dim Retval As Byte                      ' return value
Dim Status As Byte
Dim I As Byte                           ' index

Old_key = 0                             ' all buttons open

Open "Comb.0:19200,8,N,1,inverted" For Output As #1   ' PB0 ist serieller Ausgang
Wait 1
Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "Initialization"

Config Pinb.1 = Output                  ' Buzzer
Config Pinb.2 = Input                   ' S0
Config Pinb.3 = Input                   ' S1
Config Pinb.4 = Input                   ' S2
Config Pinb.5 = Input                   ' S3
Portb = 255                             ' PullUp aktiv

Buzzer Alias Portb.1

Config Serialin = Buffered , Size = 40  ' Konfig. der ser. Eingabe
Enable Interrupts

Declare Function Query_key() As Byte
Declare Sub Event(byval Number As Byte)
Declare Function Read_gsm(byval Mask As String) As Byte
Declare Function Dial(num_ As String , Msg_ As String ) As Byte
Declare Function Txt2pdu(byval Spn As String , Byval Stxt As String) As String
Declare Function Sorted_pn(byval Spn As String) As String

Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "GSM-Modem ";

Print "AT"                             ' check connection to GSM modem
Status = Read_gsm( "OK")
If Status <> 0 Then
   Print #1 , "ready" ;
   Wait 1
Else
   Print #1 , "not ready" ;
   Gosub Error
   Print #1 , Chr(instr) ; Chr(cleardisplay);
   Print #1 , "Activation";
   Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(2);
   Print #1 , "not possible.";
```

```
   Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
   Print #1 , "Program stopped.";
   End
End If

Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "Activated..."
Wait 1

Do
  Key = Query_key()                    ' query the keys
  If Key <> Old_key Then               ' any changes?
     Old_key = Key
     If Key = 0 Then                   ' all keys open
       Print #1 , Chr(instr) ; Chr(cleardisplay);
       Print #1 , "Activated...";
     Else
       Select Case Key
         Case 4 : Event 1
         Case 8 : Event 2
         Case 16 : Event 3
         Case 32 : Event 4
         Otherwise:
       End Select
     End If
  End If
  Wait 1                               ' wait until next query
Loop

End

Error:
  Do
    Reset Buzzer
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(4);
    Print #1 , "* * E r r o r * *" ;
    Waitms 100
    Set Buzzer
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(4);
    Print #1 , "                 ";
    Waitms 500
    If Pinb.2 = 0 Then Exit Do         ' leave the loop
  Loop
Return

Function Query_key() As Byte
  Query_key = Not Pinb
  Query_key = Query_key And &B00111100  ' mask input pins
End Function

Sub Event(byval Number As Byte)
  Select Case Number
    Case 1 : Restore Event1
    Case 2 : Restore Event2
    Case 3 : Restore Event3
    Case 4 : Restore Event4
  End Select
  Read Num : Read Msg
  Retval = Dial(num , Msg)             ' dial number and send message
  If Retval = 0 Then Gosub Error
```

```
End Sub

Function Read_gsm(byval Mask As String) As Byte
   Local In$ As String * 20
   Local C As Byte

   In$ = ""
   Wait 1
   While Ischarwaiting() <> 0
      C = Inkey()
      In$ = In$ + Chr(c)
   Wend
   Read_gsm = Instr(in$ , Mask)
End Function

Function Dial(num_ As String , Msg_ As String ) As Byte
  Local X As Byte

  Print #1 , Chr(instr) ; Chr(cleardisplay);
  Print #1 , "Send"
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(2);
  Print #1 , Left(msg , 20)
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
  Print #1 , "to";
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(4);
  Print #1 , Num
  Sms = Txt2pdu(num , Msg)
  X = Len(sms) / 2 : Decr X
  Print "AT+CMGS=" ; Str(x)
  Dial = Read_gsm( ">")
  If Dial <> 0 Then
    Print Sms + Chr(26)                ' send pdu and Ctrl-Z
    Dial = Read_gsm( "ERROR")
    If Dial = 0 Then Dial = 1
  End If
End Function

Function Txt2pdu(byval Spn As String , Byval Stxt As String) As String
   Dim Tmp As String * 16

   Sms = Prebytes
   Sms = Sms + Hex(len(spn))
   Sms = Sms + Toa
   Tmp = Sorted_pn(spn)
   Sms = Sms + Tmp
   Sms = Sms + Pid
   Sms = Sms + Dcs
   Sms = Sms + Hex(len(stxt))
   For I = 1 To Len(stxt)
     Tmp = Mid(stxt , I , 1)
     Sms = Sms + Hex(asc(tmp))
   Next
   Txt2pdu = Sms
End Function

Function Sorted_pn(byval Spn As String) As String
   Dim J As Byte
   Dim X As Byte
   Dim Y As Byte
   Dim S As String * 12
```

```
   Dim Ss As String * 2

   S = Spn
   X = Len(s)
   Y = X Mod 2

   If Y <> 0 Then S = S + "F"

   Sorted_pn = ""
   For I = 2 To Len(s) Step 2
       Sorted_pn = Sorted_pn + Mid(s , I , 1)
       J = I - 1
       Sorted_pn = Sorted_pn + Mid(s , J , 1)
   Next
End Function

'=================================================================
Event1:
Data "4179xxx4711" , "Alarm - Line #1 open"
Event2:
Data "4179xxx4711" , "Alarm - Line #2 open"
Event3:
Data "4179xxx4711" , "Alarm - Line #3 open"
Event4:
Data "4179xxx4711" , "Alarm - Line #4 open"
'=================================================================
```

Listing 1 Versenden von SMS (send_sms.bas)

Am Ende des Quelltextes steht eine Tabelle mit Telefonnummern und zugeordneten Mitteilungstexten für die vier hier vorgesehenen Alarmierungszustände der Überwachungsschaltung.

Die Telefonnummer wird im internationalen Format, also beginnend mit der Länderkennzahl (49 für Deutschland, 43 für Österreich, 41 für die Schweiz u.s.w.), eingetragen. Der Telefonnummer folgt der Mitteilungstext.

Die in Listing 1 stehenden Telefonnummern wurden nachträglich verändert und müssen in jedem Fall durch gültige Telefonnummern ersetzt werden.

Zu Beginn des Programms werden Konstanten und Variablen definiert und letztere auch teilweise initialisiert.

Die Ansteuerung des seriellen LCDs erfolgt über einen dem Anschluss PB0 zugeordneten Software-UART mit 9600 Baud im invertierten Mode. Ein Treiber ist deshalb nicht erforderlich. Der weitere Programmablauf wird durch entsprechende Textausgaben am LCD dokumentiert.

Im nächsten Schritt erfolgt die Initialisierung der restlichen I/O-Pins PB1 bis PB5, bevor die serielle Schnittstelle zum Mobiltelefon initialisiert wird. Um keine Zeichen vom Mobiltelefon zu versäumen wird mit einer interruptgesteuerten seriellen Eingabe mit Zwischenspeicherung in einem Ringbuffer gearbeitet. Der Speicher wurde mit 40 Bytes vereinbart. Hier können bei Bedarf auch Anpassungen gemacht werden. Nicht vergessen werden darf die Freigabe der Interrupts, da sonst keine Zeichen vom UART in den Ringbuffer geschrieben werden.

Im nächsten Schritt wird die Kommunikation mit dem Mobiltelefon durch Senden des Kommandos "AT" überprüft. Ist die Kommunikation in Ordnung, dann wird sich das Mobiltelefon mit "OK" melden, anderenfalls möglicherweise überhaupt nicht. Dieses gesendete "OK" signalisiert einen fehlerfreien Datenaustausch. Ist dieser nicht gegeben bzw.

gar kein Mobiltelefon angeschlossen, dann springt das Programm in eine Fehlerroutine, die den Kommunikationsfehler optisch und akustisch signalisiert. Diese Fehlerroutine kann durch Drücken der an PB2 angeschlossenen Taste verlassen werden. Ob man diese für die Inbetriebnahme sicher sinnvolle Möglichkeit später in der Anwendung beibehält, muss die konkrete Anwendungssituation entscheiden.

Verläuft dieser Kommunikationsversuch mit dem Mobiltelefon ohne Beanstandung, dann ist die Überwachungsschaltung aktiviert und es schließt sich eine Abfrage der Taster im Sekundentakt an. Aus den möglichen Eingangskombinationen werden hier vier Zustände decodiert, die das Versenden einer Kurzmitteilung zur Folge haben sollen.

Die Subroutine `Event(Number)` verwaltet den Zugriff auf die Tabelle mit den Einträgen der Telefonnummern und Kurzmitteilungen. Aus dieser Tabelle werden die betreffende Telefonnummer und der zugehörige Kurzmitteilungstext gelesen und der Funktion `Dial(Num , Msg)` übergeben. Wie die Namensgebung vermuten lässt, wählt diese Funktion die angegebene Telefonnummer und versendet die übergebene Kurzmitteilung. In den Funktionen `Txt2pdu()` und `Sorted_pn()` wird die dem Mobiltelefon zu übergebende PDU gemäss den Angaben im ersten Teil des Beitrags zusammengestellt. Die Konvertierung in der Funktion Txt2pdu() gestaltet sich bei Verwendung von 8-Bit Daten sehr einfach. Voraussetzung dafür ist allerdings, dass das Byte DCS entsprechend gesetzt wird (DCS=4).

In unserem Programmbeispiel (Listing 1) stehen die zu versendenden Kurzmitteilungen festcodiert im Programmbereich. Zur Übergabe von numerischen Variablen sind diese nur in einen String zu konvertieren und können dann in den zu versendenden Text aufgenommen werden.

5.3. Empfangen von SMS

Für das Empfangen von SMS kann die mit Abbildung 14 gegeben Überwachungsschaltung verwendet werden.

Auch hier wollen wir die umständliche PDU-Konvertierung vermeiden. Wir können allerdings nicht auf das 8-Bit Datenformat zurückgreifen, denn beim Empfang ist nicht bekannt, in welchem Datenformat die empfangene Kurzmitteilung vorliegt. Erst die Decodierung des DCS-Feldes zeigt an, ob es sich um 7-Bit oder 8-Bit-Daten handelt.

Nach den Erläuterungen im ersten Teil des Beitrags kann mit Hilfe des Kommandos AT+CMGL=4 der Empfangsspeicher auf empfangene Kurzmitteilungen abgefragt werden. Sind Mitteilungen vorhanden, dann werden diese anschließend auch gelistet. Der im Folgenden (nachträglich kommentierte) Hyperterminal-Mitschnitt zeigt die einzelnen Schritte.

```
AT+CPMS="ME"                    ; Umschalten auf Speicher ME
+CPMS: 1,25,1,25,1,25

OK
AT+CMGL=4                       ; Lesen vorhandener Mitteilungen
+CMGL: 1,0,,22                  ; ungelesene Mitteilung (#1) vorhanden
07911497949900F0040B911497324907F400004080718014628003C17018

OK
AT+CMGL=4                       ; Lesen vorhandener Mitteilungen
+CMGL: 1,1,,22                  ; gelesene Mitteilung (#1) vorhanden
07911497949900F0040B911497324907F400004080718014628003C17018
+CMGL: 2,0,,23                  ; ungelesene Mitteilung (#2) vorhanden
07911497949900F0040B911497324907F400004080718054318004C170380C

OK
AT+CMGD=1                       ; Löschen von Mitteilung #1
OK
AT+CMGL=4                       ; Lesen vorhandener Mitteilungen
+CMGL: 2,1,,23                  ; gelesene Mitteilung (#2) vorhanden
07911497949900F0040B911497324907F400004080718054318004C170380C

OK
AT+CMGD=2                       ; Löschen Mitteilung #2

OK

AT+CMGL=4                       ; Lesen vorhandener Mitteilungen

OK
```

Nach dem Umschalten auf den Empfangsspeicher werden alle dort bereitstehenden Kurzmitteilungen abgefragt. Bei der ersten Abfrage war eine Kurzmitteilung empfangen worden. Zum Zeitpunkt der zweiten Abfrage war eine weitere Kurzmitteilung vorhanden. Die bei der ersten Abfrage bereits gelistete Kurzmitteilung erscheint bei der zweiten Abfrage folglich auch als gelesen (0), während die zweite Kurzmitteilung noch als ungelesen (1) gilt.

Über die Kommandos AT+CMGR=*i* kann eine selektierte Mitteilung gelesen und über AT+CMGD=*i* gelöscht werden. Sind keine Kurzmitteilungen mehr im Empfangsspeicher mehr vorhanden, dann ist die Antwort auf das Kommando AT+CMGL=4 nur noch OK.

Wir wollen im folgenden Programmbeispiel die in der empfangenen Kurzmitteilung enthaltene Daten (User Data UD) mit hinterlegten Mustern vergleichen und aus entsprechende Aktionen auslösen.

In Tabelle 1 sind beispielhaft die Userdaten und die zugehörige Längenangabe (User Data Lenght UDL) für einige willkürliche, für den Test benutzt Kurzmitteilungen angegeben. Dass

die empfangenen Kurzmitteilungen insgesamt etwas umfangreicher waren, hatten wir bereits gesehen.

SMS	*UDL&UD*
A	...0141
Aa	...02C130
Aaa	...03C17018
Aaaa	...04C170380C
Aaaaa	...05C170381C06
Aaaaaa	...06C170381C0E03
Aaaaaaa	...07C170381C0E8701
Aaaaaaaa	...08C170381C0E87C3
Aaaaaaaaa	...09C170381C0E87C361

Tabelle 1 Userdaten einer SMS (7-Bit-Mode)

Listing 2 zeigt den Quelltext des Programmbeispiels receive_sms2.bas. Nach Überprüfung der Verbindung zum eingesetzten Mobiltelefon und der Initialisierung der zum Vergleich heranzuziehenden Muster (Pattern) tritt das Programm in eine Endlosschleife und arbeitet die bereits weiter oben erläuterten Schritte ab. Jeder Schritt wird durch Ausgaben am angeschlossenen LCD kommentiert.

Wird durch das Kommando AT+CMGL=4 eine oder mehrere Mitteilungen im Mobiltelefon festgestellt, dann wird eine Mitteilungsnummer *i* mit Hilfe der Funktion `message_number()` festgestellt. Diese Mitteilung wird dann mit dem Kommando AT+CMGR=*i* gelesen und mit Hilfe der Subroutine `decode_message()` decodiert. Kern dieser Subroutine ist das Reduzieren der Kurzmitteilung auf die Userdaten und der anschließende Vergleich mit den hinterlegten Mustern. In Abhängigkeit vom Vergleichergebnis werden Subroutines aufgerufen, die die gewünschte Aktion (Event) auslösen.

```
$regfile = "m8def.dat"                  ' ATmega8
$crystal = 3690000                      ' für STK500
$baud = 19200

Const Instr = 254
Const Setcursor = 71                    ' [x] [y]
Const Cursorhome = 72
Const Cleardisplay = 88

Const Prebytes = "000100"
Const Toa = "91"
Const Pid = "00"
Const Dcs = "04"

Const Entries = 4                       ' maximum number of events

Dim Msg As String * 160
Dim Sms As String * 160
Dim Pattern(entries) As String * 10

Dim A1 As Byte
Dim A2 As Byte
```

```
Dim S1 As String * 1
Dim S As String * 10

Dim Retval As Byte                      ' return value
Dim Status As Byte
Dim I As Byte                           ' index

Open "Comb.0:19200,8,N,1,inverted" For Output As #1   ' PB0 is serial output
Wait 1
Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "Initialization"

Config Pinb.1 = Output                  ' Buzzer
Buzzer Alias Portb.1
Config Pinb.2 = Input
Portb.2 = 1                             ' PullUp active

Config Pinb.4 = Output
Config Pinb.5 = Output

Declare Function Read_gsm(byval Mask As String) As Byte
Declare Function Message_number() As Byte
Declare Sub Decode_message()

Config Serialin = Buffered , Size = 160 ' configuration of serial input
Enable Interrupts

Pattern(1) = "41"                       ' SMS=A
Pattern(2) = "C130"                     ' SMS=Aa
Pattern(3) = "C17018"                   ' SMS=Aaa
Pattern(4) = "C170380C"                 ' SMS=Aaaa

Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "GSM-Modem ";

Print "AT"                              ' check connection to GSM modem
Status = Read_gsm( "OK")
If Status <> 0 Then
   Print #1 , "ready" ;
   Wait 1
Else
   Print #1 , "not ready" ;
   Gosub Error
   Print #1 , Chr(instr) ; Chr(cleardisplay);
   Print #1 , "Activation";
   Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(2);
   Print #1 , "not possible.";
   Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
   Print #1 , "Program stopped.";
   End
End If

Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "Activated..."
Wait 1

Print "AT+CPMS=" ; Chr(34) ; "ME" ; Chr(34) ' switch to ME
Status = Read_gsm( "+CPMS:")
If Status = 0 Then Gosub Error
```

```
Do
  Do                                    ' look for received messages
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
    Print #1 , "Wait for SMS...";
    Print "AT+CMGL=4"
    Status = Read_gsm( "+CMGL:")
    Wait 1
  Loop Until Status <> 0
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
  Print #1 , "SMS received     ";
  Wait 1

  I = Message_number()
  Print "AT+CMGR=" ; Str(i)             ' read message # i
  Status = Read_gsm( "OK")
  If Status = 0 Then Gosub Error
  Call Decode_message()                 ' decode message and compare with saved patterns
  Print "AT+CMGD=" ; Str(i)             ' delete message # i
  Status = Read_gsm( "OK")
  If Status = 0 Then Gosub Error
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
  Print #1 , "SMS deleted     ";
  Wait 1
Loop

End

Error:
  Do
    Reset Buzzer
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(4);
    Print #1 , "* * E r r o r * *" ;
    Waitms 100
    Set Buzzer
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(4);
    Print #1 , "                  ";
    Waitms 500
    If Pinb.2 = 0 Then Exit Do          ' leave the loop
  Loop
Return

Function Read_gsm(byval Mask As String) As Byte
   Dim In$ As String * 160
   Dim C As Byte

   In$ = ""
   Wait 2                               ' wait for an answer from GSM
   While Ischarwaiting() <> 0
      C = Inkey()
      In$ = In$ + Chr(c)
   Wend
   Msg = In$
   Read_gsm = Instr(msg , Mask)
End Function

Function Message_number() As Byte
  A1 = Instr(msg , "+CMGL: ")           ' decode message number
  A1 = A1 + 6 : S = "" : S1 = ""
  Do
    Incr A1
```

```
    S = S + S1
    S1 = Mid(msg , A1 , 1)
  Loop Until S1 = ","
  Message_number = Val(s)
End Function

Sub Decode_message()
  A1 = Instr(msg , ",,")
  Incr A1 : S = "" : S1 = ""
  While S1 <> Chr(10)
    Incr A1
    S1 = Mid(msg , A1 , 1)
    S = S + S1
  Wend
  A1 = Val(s) * 2                       ' TPDU
  A2 = Len(msg) - 8
  Msg = Left(msg , A2)                  ' clear OK etc.
'  Print #1 , Chr(instr) ; Chr(cleardisplay);
'  Print #1 , Msg ;
'  Wait 4
  Msg = Right(msg , A1)                 ' User Data
'  Print #1 , Chr(instr) ; Chr(cleardisplay);
'  Print #1 , Msg ;
'  Wait 4
  A2 = Len(msg) - 2
  Msg = Right(msg , A2)                 ' clear DCS
'  Print #1 , Chr(instr) ; Chr(cleardisplay);
'  Print #1 , Msg ;
'  Wait 4
  S = Left(msg , 2)
  A1 = Hexval(s)
  A2 = A1 Mod 2
  If A2 <> 0 Then Incr A1
  A1 = A1 + 4
  A2 = Len(msg) - A1
  Msg = Right(msg , A2)                 ' clear phone number
'  Print #1 , Chr(instr) ; Chr(cleardisplay);
'  Print #1 , Msg ;
'  Wait 4
  A2 = Len(msg) - 20                    ' clear date
  Msg = Right(msg , A2)
'  Print #1 , Chr(instr) ; Chr(cleardisplay);
'  Print #1 , Msg ;
'  Wait 4
  If Msg = Pattern(1) Then Gosub Event1
  If Msg = Pattern(2) Then Gosub Event2
  If Msg = Pattern(3) Then Gosub Event3
  If Msg = Pattern(4) Then Gosub Event4
End Sub

'=============================================================
Event1:                                 ' define action
  Set Portb.4
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
  Print #1 , "Event1          ";
  Wait 1
Return

Event2:
  Reset Portb.4
```

```
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
  Print #1 , "Event2          ";
  Wait 1
Return

Event3:
  Set Portb.5
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
  Print #1 , "Event3         ";
  Wait 1
Return

Event4:
  Reset Portb.5
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
  Print #1 , "Event4          ";
  Wait 1
Return
'============================================================
```

Listing 2 Empfang von SMS (receive_sms2.bas)

Die feste Codierung der Vergleichsmuster im Programm ist für den Test in Ordnung, sicher aber keine sehr praktikable Lösung. Ein Lernmode kann die Zuordnung Event – SMS flexibler gestalten.

Für die Eingabe der Vergleichsmuster kann das Mobiltelefon selbst herangezogen werden. Die SMS werden über die Tastatur so eingetippt, wie sie auch später erwartet werden aber nur abgespeichert. So stehen sie im Ausgangsspeicher und können wieder vom Mikrocontroller gelesen und decodiert werden. In Tabelle 2 sind zwei identische SMS in gespeicherter und empfangener Form gegenübergestellt.

SMS	*Telegramm*
Aaa (gesichert)	07911497949900F01100008100 00A9 03C17018
Aaa (empfangen)	07911497949900F0040B911497324907F400004080619054848003C17018

Tabelle 2 Telegrammaufbau SMS-SUBMIT und SMS-DELIVER

Wichtig für das folgende Programmbeispiel sind die identischen Userdaten und der unterschiedliche Telegrammaufbau (SMS-SUBMIT, SMS-DELIVER).

Listing 3 zeigt den Quelltext des Programmbeispiels receive_sms1.bas. Der Programmaufbau ist weitgehend identisch zu Listing 2. Nach der Überprüfung der Verbindung zum eingesetzten Mobiltelefon erfolgt die dialoggeführte Eingabe der den einzelnen Events zugeordneten SMS (Vergleichsmuster, Pattern).

Nach Aufforderung eine SMS einzugeben, wartet der Mikrocontroller bis eine Kurzmitteilung im Sendespeicher (SM) steht. Wurde eine Mitteilung gefunden, dann erfolgt deren Decodierung und Abspeicherung durch die Subroutine `save_pattern(i)`. Nach der Decodierung wird die Kurzmitteilung gelöscht und zur Eingabe der nächsten aufgefordert. Jeder Schritt wird wieder durch Ausgaben am angeschlossenen LCD kommentiert.

```
$regfile = "m8def.dat"                    ' ATmega8
$crystal = 3690000                        ' für STK500
$baud = 19200
```

```
Const Instr = 254
Const Setcursor = 71                    ' [x] [y]
Const Cursorhome = 72
Const Cleardisplay = 88

Const Prebytes = "000100"
Const Toa = "91"
Const Pid = "00"
Const Dcs = "04"

Const Entries = 4                       ' maximum number of events

Const Test = 1

Dim Msg As String * 160
Dim Sms As String * 160
Dim Pattern(entries) As String * 10

Dim A1 As Byte
Dim A2 As Byte
Dim S1 As String * 1
Dim S As String * 10
Dim Retval As Byte                      ' return value
Dim Status As Byte
Dim I As Byte                           ' index
Dim Learning As Bit                     ' flag for learning mode

Open "Comb.0:19200,8,N,1,inverted" For Output As #1       ' PB0 is serial output
Wait 1
Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "Initialization"

Config Pinb.1 = Output                  ' Buzzer
Buzzer Alias Portb.1
Config Pinb.2 = Input
Portb.2 = 1                             ' PullUp active

Config Pinb.4 = Output
Config Pinb.5 = Output

Declare Function Read_gsm(byval Mask As String) As Byte
Declare Sub Save_pattern(byval N As Byte)

Config Serialin = Buffered , Size = 160 ' configuration serial input
Enable Interrupts

Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "GSM-Modem ";

If Pinb.2 = 0 Then Learning = 1

Print "AT"                             ' check connection to GSM modem
Status = Read_gsm( "OK")
If Status <> 0 Then
   Print #1 , "ready" ;
   Wait 1
Else
   Print #1 , "not ready" ;
   Gosub Error
```

```
   Print #1 , Chr(instr) ; Chr(cleardisplay);
   Print #1 , "Activation";
   Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(2);
   Print #1 , "not possible.";
   Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
   Print #1 , "Program stopped.";
   End
End If

If Learning = 1 Then
  Print "AT+CPMS=" ; Chr(34) ; "SM" ; Chr(34)       ' switch to SIM
  Status = Read_gsm( "+CPMS:")
  If Status = 0 Then Gosub Error
  For I = 1 To Entries
    Print #1 , Chr(instr) ; Chr(cleardisplay);
    Print #1 , "Learning Mode";
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(2);
    Print #1 , "Type SMS for Event " ; I;
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(3);
    Print #1 , "Wait for SMS..." ;
    Do
      Print "AT+CMGL=2"                 ' read unsent messages
      Status = Read_gsm( "+CMGL:")
      Wait 1
    Loop Until Status <> 0
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(4);
    Print #1 , "SMS received";
    Call Save_pattern(i)
    Print "AT+CMGD=1"                  ' delete message # 1
    Status = Read_gsm( "OK")
    If Status = 0 Then Gosub Error
  Next
End If

Print #1 , Chr(instr) ; Chr(cleardisplay);
Print #1 , "Saved pattern:";
Wait 2
Print #1 , Chr(instr) ; Chr(cleardisplay);

For I = 1 To Entries
  Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(i);
  Print #1 , I ; ": " ; Pattern(i);
Next

End

Error:
  Do
    Reset Buzzer
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(4);
    Print #1 , "* * E r r o r * *" ;
    Waitms 100
    Set Buzzer
    Print #1 , Chr(instr) ; Chr(setcursor) ; Chr(1) ; Chr(4);
    Print #1 , "                ";
    Waitms 500
    If Pinb.2 = 0 Then Exit Do          ' leave the loop
  Loop
Return
```

```
Function Read_gsm(byval Mask As String) As Byte
   Dim In$ As String * 160
   Dim C As Byte

   In$ = ""
   Wait 2
   While Ischarwaiting() <> 0
      C = Inkey()
      In$ = In$ + Chr(c)
   Wend
   Msg = In$
   Read_gsm = Instr(msg , Mask)
End Function

Sub Save_pattern(byval N As Byte)
  A1 = Instr(msg , ",,")
  Incr A1 : S = "" : S1 = ""
  While S1 <> Chr(10)
    Incr A1
    S1 = Mid(msg , A1 , 1)
    S = S + S1
  Wend
  A1 = Val(s) * 2                    ' length of TPDU
  A2 = Len(msg) - 8
  Msg = Left(msg , A2)               ' clear OK etc.
  Msg = Right(msg , A1)              ' TPDU
  A1 = Len(msg) - 16
  Msg = Right(msg , A1)              ' User Data
  Pattern(n) = Msg
End Sub
```

Listing 3 Abspeichern von Vergleichmustern (receive_sms1.bas)

Sind schließlich die hier vereinbarten vier SMS eingetippt worden, dann erfolgt die Darstellung der abgespeicherten Userdaten auf dem LCD zur Kontrolle. Abbildung 15 zeigt die Userdaten der bereits verwendeten SMS (A, Aa, Aaaa, Aaaaa)., die zu den in Tabelle 1 abgegebenen Daten identisch sind.

Abbildung 15 Ausgabe der Vergleichsmuster am LCD

Anhang

Wichtige AT+C-Kommandos

Befehl	Funktion
AT+CPIN	Eingabe der Pinnummer
AT+COPS	Netzbetreiber auswählen; AT+COPS=0 wählt das Heimnetz
AT+CREG	Registrierungszustand anzeigen
AT+CSQ	Signalqualität ausgeben
AT+IPR	Fixieren der Baudrate der seriellen Schnitttstelle; AT+IPR=0 bedeutet Autobauding
AT+CGMI	Herstellerdaten abfragen
AT+CGMM	Modellkenndaten abfragen
AT+CGMR	Softwareversion abfragen
AT+CGSN	Abfrage der IMEI (International Mobile Equipment Identity)
AT+CBC	Batterieladung abfragen
AT+CPBS	Telefonbuchspeicher auswählen und verändern
AT+CPBW	Telefonbucheinträge schreiben und löschen
AT+CPBR	Aktuelle Telefonbucheinträge lesen
AT+CPMS	SMS-Speicher auswählen
AT+CSMS	Short message Service auswählen
AT+CMGF	SMS-Format auswählen
AT+CSCA	Adresse des SMS Service Center eingeben
AT+CMGL	SMS aus ausgewähltem Speicher auflisten
AT+CMGR	Einzelne SMS lesen
AT+CMGW	SMS in SMS-Speicher schreiben
AT+CMGD	SMS löschen
AT+CMGS	SMS direkt senden
AT+CMSS	SMS aus dem SMS-Speicher senden
AT+CNMI	Anzeige neu eingegangener SMS

Werkseinstellungen über AT&F beim Siemens S45 [2]

ATE1 (only in case of RCCP mode)
ATQ0
ATV1
AT+CCWA=0
AT+CREG=0
AT+CLIP=0
AT+COLP=0
AT+CRC=0
AT+CAOC=0
AT+CMEE=0
AT+CPBS=SM (if available)
AT+COPS=0
AT+VTS=1
AT+CSCS="GSM"
AT+CSSN=0,0

AT^SCKS=0

Reset pending locks (Phone Pin/Puk, Pin2/Puk2 ...)

which are given as answer to AT+CPIN?

AT+CSMS=0

AT+CNMI=0,0,0,0,1

AT^SMGO=0

AT+CSCB=0

6. Literatur

[1] Zogg, Jean-Marie:
Telemetrie mit GSM/SMS und GPS-Einführung.
Poing: Franzis' Verlag, 2002

[2] Siemens Mobile Phones - Reference Manual
AT command set for S45 Siemens mobile phones and modems
Document ID: A30880-A10-A001-3-D376
Release/Version: 1.8 - 30.11.2001

[3] Lanconoelli, C.; Ricci Bitti, A.:
Tiny Planet — A One-Bit Wireless I/O Port
Circuit Cellar, Issue 142, May 2002

[4] ETS 300 585
Digital cellular telecommunications system (Phase 2);
Use of Data Terminal Equipment - Data Circuit terminating Equipment (DTE - DCE) interface for Short Message Service (SMS) and Cell Broadcast Service (CBS)
(GSM 07.05)

[5] Comparison chart of AT+C commands of GSM devices
http://gatling.ikk.sztaki.hu/~kissg/gsm/at+c.html

[6] PDUSpy? PDUSpy.
http://www.nobbi.com/pduspy.htm

[7] Kurznachrichten auf die perverse Art
http://www.nobbi.com/sms_pdu.htm

[8] Kühnel, C.:
Programmieren der AVR RISC Mikrocontroller mit BASCOM-AVR.
2. Auflage
Altendorf: Skript Verlag Kühnel, 2004
ISBN 3-907857-04-6