GSM-Modul mit SIM800L (TTGO-Version)

Schon für weniger als 5 € kann man inzwischen GSM/GPRS-Module (inkl. Porto) erwerben. Die Preise variieren allerdings sehr stark je nach benutztem IC und den zusätzlichen Komponenten auf dem Modul. Ich habe mich für eine spartanische Version entschieden (vgl. Abb. 1). Im Wesentlichen sind hier nur die wichtigsten I/O-Anschlüsse des SIM800L-Bausteins herausgeführt; daneben gibt es einen Anschluss für eine Antenne sowie eine Kontroll-LED (dazu später mehr). Die Antenne war übrigens bei meinem Modul mit im Preis inbegriffen.

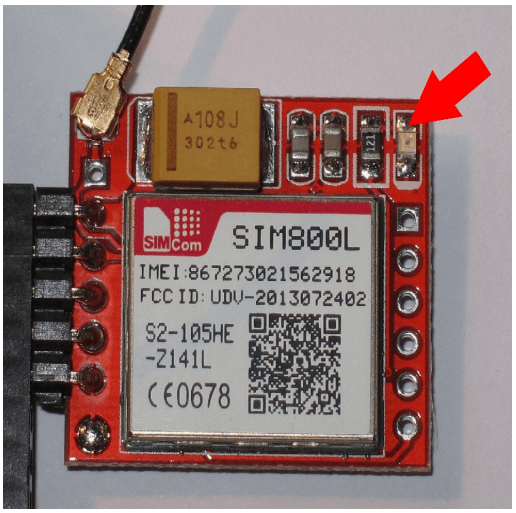

Abb.1: Das SIM800L-Modul

Achtung: Die Versorgungsspannung des SIM800L darf 4,4 V nicht überschreiten; vorgesehen sind 3,4 V - 4,4 V. Ansonsten kann der Baustein zerstört werden. Die Signalpegel dürfen aber auch 5 V betragen.

In diesem Beitrag werden wir ausführlich darlegen, wie man über das SIM800L-Modul eine SMS senden oder empfangen kann; damit kann z. B. mittels eines Mikrocontrollers (z. B. dem TTGO) eine Meldung an ein Handy gesendet werden, wenn ein Sensor ein entsprechendes Signal liefert. Abschließend zeigen wir auch, wie man mit dem SIM800L telefonieren kann.

Für erste Erfahrungen schließen wir das SIM800L-Modul an einen USB-COM-Wandler FTDI232RL an:

FTDI232	**SIM800L**
3,3 V über USB (Jumper setzen)	
GND (über USB)	GND, auch mit Minus-Pol des Netzgeräts (s.u) verbinden
RXD	TXD
TXD	RXD
	VCC → Plus-Pol eines externen Netzgeräts (4,0 V)

Da der SIM800L im Extremfall bis zu 2A Strom benötigt, reicht der Strom, den der FTDI232-Baustein zur Verfügung stellen kann, in der Regel nicht aus. Wir benutzen hier ein gutes geregeltes Netzgerät; alternativ kann auch ein entsprechender Akku eingesetzt werden.

1 Einführung

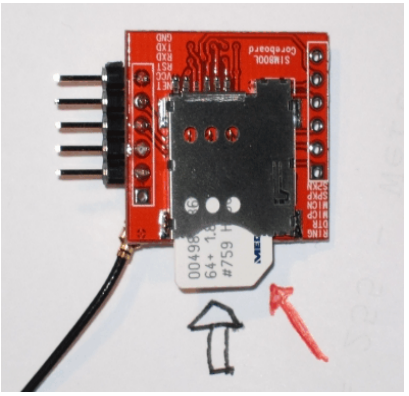

Abb: 2: Einschieben der SIM-Karte

Bevor es losgeht, müssen wir noch unsere Mikro-SIM-Karte in die entsprechende Halterung auf der Rückseite des SIM800L hineinschieben. Achtung: Die Kontakte der SIM-Karte müssen dabei nach unten weisen und die Aussparung der SIM-Karte (roter Pfeil) muss beim Einschieben auf Südosten liegen (vgl. Abb. 2).

Wenige Sekunden nachdem das Modul mit Strom versorgt worden ist, fängt die Kontroll-LED (s. Pfeil in Abb. 1) an zu blinken (800 ms Blinkrhythmus).

Das SIM800L-Modul wird über sogenannte **AT-Commands**, wie sie manchen vielleicht noch von den Modems bekannt sind, gesteuert und kontrolliert. Diese werden nach dem UART-Standard übertragen. Über diese können die Kommandos und die Antworten des SIM800L übermittelt werden. Dazu setzen wir hier das (frei erhältliche) Terminal-Programm **HTerm** ein. Nach dem Start dieses Programms geben wir zunächst als **Baudrate** den Wert 56000 ein; dann wählen wir die vom FTDI-Baustein benutzte **COM-Nummer**. Anschließend aktivieren wir die Verbindung mit dem **Connect**-Button.

Übrigens können auch andere Baudraten benutzt werden: Der SIM800L-Baustein erkennt die benutzte Baudrate nämlich automatisch. Manchmal gelingt das bei dem allerersten Befehl noch nicht korrekt; geben Sie dann das Kommando noch ein weiteres mal ein.

Für die Ausgabe der von Hterm empfangenen Daten (**received data**) stellen wir ein: ASCII aktivieren (Häkchen) und "Newline at LF". Alle Kommandos werden im **Input-Control-Bereich** (am unteren Rand des Formulars) eingegeben. Passende Einstellungen sind: Häkchen bei ASCII und "CR" bei "Send on enter".

Als ersten Test geben wir nun das Kommando `AT` ("Attention") in der Eingabe-Zeile ein und schließen die Eingabe mit der Enter-Taste. Dadurch wird die Zeichenkette `AT\r` an das SIM800L-Modul übertragen; das Steuerzeichen `\r` steht hier für CR. Das Modul antwortet immer mit einem **Echo** (in diesem Fall `AT\r`, gefolgt von den Steuerzeichen `\r\n` (\r für CR und \n für LF bzw. New Line) und - wenn alles in Ordnung ist - mit einem `OK\r\n`).

Mit dem Kommando `AT+CPIN="xxxx"` loggen wir uns ins GSM-Netz ein; statt `xxxx` geben Sie natürlich die PIN Ihres Handys ein. Das SIM800L-Modul gibt zunächst das Echo des eingegebenen Kommandos aus; nach kurzer Pause sendet es dann folgende Zeilen:

```
+CPIN: READY
Call Ready
SMS Ready
```

Damit ist unser SIM800L-Modul ins GSM-Netz eingeloggt. Dies können wir mit dem Kommando `AT+CREG?` überprüfen: Die Zahl 1 in der Antwort

```
+CREG: 0,1
```

zeigt an, dass das Modul eingeloggt ist. Mit dem Kommando `AT+COPS?` können wir auch den Namen unseres GSM-Netzes anzeigen lassen.

Wenn das Modul eingeloggt ist, wird dies auch durch die Kontroll-LED angezeigt: Diese blinkt nun mit einer Periodendauer von etwa 3s.

2 Versenden einer SMS mit HTerm

Der folgende Screenshot zeigt, wie man eine SMS versenden kann; dabei wurden die PIN und die Telefonnummer des Adressaten unkenntlich gemacht.

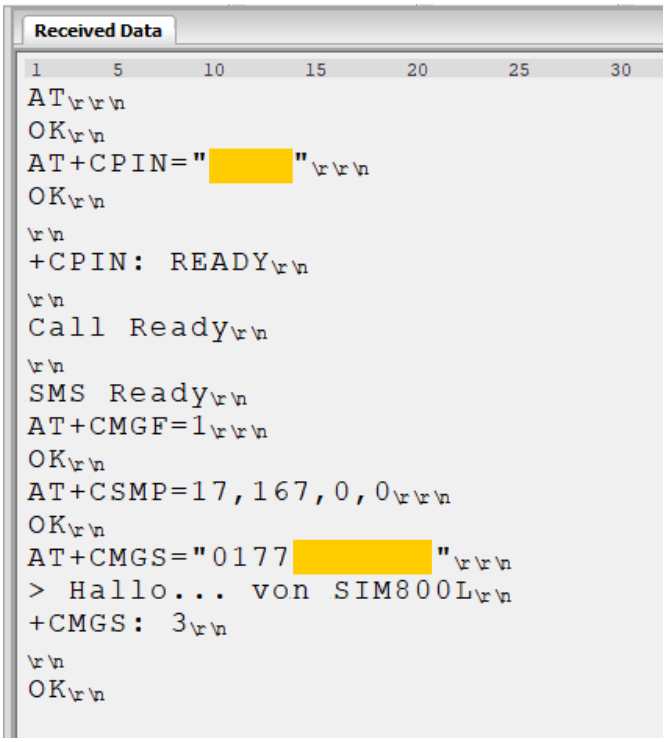

Abb. 3

Das Einloggen erfolgt wieder wie oben bereits beschrieben. Mit dem Kommando `AT+CMGF=1` wird jetzt das SMS-Format festgelegt:

Select SMS Message Format
Für eine SMS stehen zwei verschieden Formate zur Verfügung: Text-Mode (`CMGF=1`) und PDU-Mode (`CMGF=0`). Während der Text-Mode für (reine) Texte benutzt wird, können im PDU-Mode auch binäre Daten, z. B. komprimierte Daten, versendet werden.

Das folgende Kommando `AT+CSMP=17,167,0,0` legt einige Parameter für das Text-Format fest:

Set SMS Text Mode Parameters
Bis zu 4 Parameter vom Type Byte können hier gesetzt werden. Dabei kann die Bedeutung einzelner Bits manchmal auch davon abhängen, wie die anderen Bits gesetzt sind. Hier sollen nur die für uns entscheidenden Bits erläutert werden:

1. Parameter (fo)
fo.0=0 und fo.1=0 bedeutet SMS-DELIVER (Versand von Basis-Station an Handy); fo.0=1 und fo.1=0 bedeutet SMS-SUBMIT (Versand von Handy an Basis-Station).
Fo.3=0 und fo.4=1 aktiviert eine Validitätsperiode, wenn ein vp-Wert vorliegt, s. u.

2. Parameter (vp)
Unser Wert von 167 bedeutet eine Periode von 24 Stunden; für diesen Zeitraum wird die SMS zwischengespeichert, wenn der Adressat nicht erreichbar ist. Danach verfällt die SMS.

Mit dem Kommando `AT+CMGS="xxxxxxxxxxx"` wird die Telefonnummer des Handys festgelegt, an das die SMS gesendet werden soll. Es erscheint dann in der nächsten Zeile das **Eingabe-Prompt** ">". Unsere Nachricht (ohne Sonderzeichen wie Ä, Ö, ß, ...) wird nicht mit CR, sondern mit Ctrl-Z abgeschlossen. Deswegen wählen wir jetzt bei **Set on enter** zunächst die **Option None** und senden anschließend unseren Text, z. B. *Hallo... von SIM800L* (ohne Anführungszeichen!). Um jetzt noch das abschließende **Steuerzeichen Ctrl-Z** zu senden, wählen wir für **Type** die **Option DEC** und senden die Zahl 26.

Kurze Zeit später erhalten wir auf unserem Handy die folgende Nachricht:

Abb. 4

3 TTGO sendet eine SMS mit SIM800L

Im Kapitel 2 habe ich dargestellt, wie man mithilfe eines Terminalprogramms wie HTerm eine SMS über das SIM800L-Modul versenden kann. Nun möchte ich zeigen, wie eine SMS von unserem TTGO verschickt werden kann. Dies kann man z. B. ausnutzen, um sich eine Alarm-SMS zukommen zu lassen, wenn z. B. eine Bewegung mit einem PIR-Bewegungsmelder registriert wird (s. dazu auch Kapitel 4).

Es ist auch recht einfach, den TTGO die benötigten AT-Commands (s.o.) an das SIM800L-Modul schicken zu lassen. Dazu benutzen wir eine Instanz der UART-Klasse:

```
from machine import UART
uart1 = UART(1, baudrate = 56000, tx = 12, rx = 13)
```

Um ein Kommando an den SIM800L-Baustein zu übertragen, benutzen wir die Methode `send_command`:

```
def send_command(uart, cmd):   # cmd ist vom Typ String, ohne \r
    c = cmd + '\r'             # c ist Vom Typ String
    anzahl_der_gesendeten_zeichen = uart.write(c)
    print(anzahl_der_gesendeten_zeichen, 'Zeichen gesendet:', c)
```

Das AT-Kommando wird damit z. B. so gesendet:

```
send_command(uart1, b'AT')
```

Deutlich komplexer ist die Aufgabe, die vom SIM800L gegebenen Antworten zu analysieren. Die Analysen sind erforderlich, weil diese anzeigen, wann und ob der SIM800L einen Befehl erfolgreich ausgeführt hat; erst dann ist es sinnvoll, die nächsten Schritte durchzuführen. Bei dieser Analyse durchsucht der TTGO nach jedem empfangenen Byte die bislang vorliegenden Byte-Folge nach einem signifikanten Suchwort, welche das Ende der Botschaft kennzeichnet. Bei dem `AT`-Kommando benutzen wir als Suchwort z. B. "`OK\r\n`", bei dem Kommando `AT+CPIN="xxxx"` wählen wir als Suchwort z. B. "`SMS READY\r\n`".

Die folgende Funktion `read_resonse` durchsucht die vom SIM800L empfangene Antwort nach dem Suchwort (`search_word`). Wird diese innerhalb der Time-Out-Zeit (`timeout` in ms) gefunden, wird die empfangene Antwort zurückgegeben, ansonsten wird das Programm abgebrochen. Dafür benötigen wir zusätzlich die folgenden Importe (Die Methode `ticks_ms` liefert einen Zeitstempel in ms; mit `ticks_diff` bestimmt man den Zeitabstand zwischen zwei Zeitstempeln.):

```
from time import sleep, ticks_ms, ticks_diff
from sys import exit
```

```
def read_response(uart, search_word, timeout):
    received = b''
    t0 = ticks_ms()
    while not (search_word in received):
        c = uart.read(1) # nur 1 Byte lesen
        if c != None:
            # print(c)
            received = received + c
            # print(received)
        t1 = ticks_ms()
        deltat = ticks_diff(t1, t0)
        if deltat > timeout:
            print('Timeout: Suchwort nicht in Anwort enthalten')
            print('Abbruch...')
            exit()
    return str(received, 'UTF-8')
```

Unser Programm führt die einzelnen Senden-Empfangen-Schritte im Prinzip genauso der Reihe nach durch, wie das im letzten Kapitel beschrieben worden ist. Es tauchen nur zwei Zusätze auf:

1. Nach dem einleitenden `AT`-Kommando senden wir (optional) noch ein `ATZ`-Kommando; dieses sorgt beim SIM800L-Baustein für einen RESET.

2. Wenn das SIM800L-Modul in einem vorangegangen erfolgreichen Programmdurchlauf noch im GSM-Netz eingeloggt ist, dann soll das Kommando `AT+CPIN="xxxx"` nicht ein weiteres Mal ausgeführt werden. Ob unser Modul noch eingeloggt ist, können wir mit dem Kommando

```
AT+CREG?
```

kontrollieren; ist das Modul eingeloggt, liefert es als Antwort `+CREG 0,1`, ansonsten `+CREG 0,0`.

Das gesamte Programm `send_sms.py` finden Sie im Anhang.

Bemerkung: In meinem Beitrag `https://g-heinrichs.de/attiny/module/SIM800L.pdf` habe ich ein Programm für den Attiny2313 vorgestellt, welches ganz ähnlich arbeitet. Allerdings war in diesem Fall die Empfangs-Prozedur deutlich aufwendiger, insbesondere die Suchwort-Suche: Diese wurde in einer Interrupt-Routine mithilfe eines Software-Ringspeichers durchgeführt.

4 TTGO sendet eine Alarm-SMS mit SIM800L

Im Kapitel 3 habe ich dargestellt, wie man mithilfe eines Terminals eine SMS über das SIM800L-Modul versenden kann. Nun möchte ich zeigen, wie eine Alarm-SMS von unserem TTGO verschickt werden kann, wenn z. B. eine Bewegung mit einem PIR-Bewegungsmelder registriert wird.

Im Folgenden soll der Alarm der Einfachheit halber durch Betätigen des Tasters `Ta0` des TTGO ausgelöst werden. Es dürfte keine größeren Probleme bereiten, das Programm so abzuändern, dass der Alarm auch durch andere Sensoren ausgelöst werden kann.

Das Programm baut im Wesentlichen auf dem Programm des letzten Kapitels auf. Damit es auch ohne Entwicklungsumgebung eingesetzt werden kann, werden jetzt die wichtigsten Informationen für den Anwender auch auf dem Display angezeigt.

Außerdem wird berücksichtigt, dass die Verbindung zwischen dem SIM800L-Modul und dem GSM-Netz nicht unbedingt schon beim ersten Versuch zustande kommt oder auch nach einem Einloggen wieder abbrechen kann; mit diesem Problem war ich manchmal konfrontiert, weil in meinem Arbeitszimmer das benutzte GSM-Netz recht schwach ist. So kann es vorkommen, dass das vom SIM800L-Modul gesendete Signal nicht von dem benutzten Netz registriert wird oder umgekehrt die vom Funkmast abgestrahlten Antworten nicht oder nur teilweise von meinem Modul empfangen werden konnten. In solchen Fällen wird in der `response`-Funktion die globale Variable `time-out-flag` auf `True` gesetzt; dann wird erneut versucht, die Alarm-SMS zu senden. Wie viele Versuche maximal durchgeführt werden, wird in der Variable `max_count` festgelegt.

Wird der Alarm z. B. durch einen Bewegungs/Näherungs-Sensor ausgelöst, muss beachtet werden, dass ggf. in rascher Folge Alarm-Signale beim TTGO ankommen. Damit nun keine riesige Flut von Nachrichten an das Handy gesendet wird, werden nach einem Alarm-Ereignis für eine gewisse Zeitspanne `delay_after_alarm` keine SMS gesendet.

Weitere Hinweise zur Funktionsweise des Programms finden Sie in den Kommentaren des folgenden Listings.

```
#################################################################################
#                              Import und Instanziierungen                      #
#################################################################################

from time import sleep, ticks_ms, ticks_diff

from machine import UART, Pin, SPI
# UART und Taster instanziieren;  SIM800L hat autobauding von 1200 bis 57600
uart1 = UART(1, baudrate = 56000, tx = 12, rx = 13)
taster0 = Pin(0, Pin.IN, Pin.PULL_UP)
```

```
import vga1_8x16 as font1
import vga1_bold_16x32 as font2
import st7789

# display instanziieren und konfigurieren...
spi = SPI(1, baudrate=20000000, polarity=1, sck=Pin(18), mosi=Pin(19))
display = st7789.ST7789(spi, 135, 240,  reset=Pin(23, Pin.OUT), cs=Pin(5, Pin.OUT),
dc=Pin(16, Pin.OUT), backlight=Pin(4, Pin.OUT), rotation=3)
display.init()


################################################################################
#     Funktionen u. globale Variable zur Kommunikation mit dem SIM800L-Modul       #
################################################################################

pin = b'xxxx'
phone_number = b'0177xxxxxxx'
timeout = 40000 # in ms
message = b'Alarm: Taster0 wurde betaetigt'
delay_after_alarm = 120 # in Sekunden (Ein erneuter Alarm kann erst nach
                                                dieser Zeitspanne gesendet werden.)
time_out_flag = False
count = 0 # zählt Anzahl der Versuche (Wenn ein Time-Out auftritt,
                        wird ein erneuter Sende-Versuch gestartet; max. 3 Versuche)
max_count = 3


def send_command(uart, cmd):       # cmd ist vom Typ Bytes, ohne \r
    c = cmd + '\r'       # c ist vom Typ Bytes
    anzahl_der_gesendeten_zeichen = uart.write(c)
    print(anzahl_der_gesendeten_zeichen, 'Zeichen gesendet:', c)

def read_response(uart, search_word, timeout):
    global time_out_flag
    received = b''
    t0 = ticks_ms()
    while not (search_word in received) or ('ERROR' in received):
        c = uart.read(1) # nur 1 Byte lesen (oder keins)
        if c != None:
            # print(c)
            received = received + c
            # print('received:', received) # zum Testen
        t1 = ticks_ms()
        deltat = ticks_diff(t1, t0)
        if deltat > timeout:
            time_out_flag = True
            break # Abbruch, wenn timeout ms überschritten
            # print('Time-Out-Flag', time_out_flag) # zum Testen
    print('Test', received)
    return str(received, 'UTF-8')
```

```
###############################################################################
#                                 Alarm senden                                #
###############################################################################

def one_alarm(message, uart):
    global time_out_flag
    # Attention...
    cmd = b'AT'
    send_command(uart, cmd)
    response = read_response(uart1, b'OK\r\n', timeout)
    print(response) # zum Testen
    sleep(2)

    '''
    # SIM800L resetten, ggf. zum Auslassen auskommentieren
    if not time_out_flag:
        cmd = b'ATZ'
        send_command(uart, cmd)
        response = read_response(uart1, b'OK\r\n', timeout)
        print(response)
        sleep(2)
    '''

    # eingeloggt?
    if not time_out_flag:
        cmd = b'AT+CREG?'
        send_command(uart, cmd)
        response = read_response(uart1, b'OK\r\n', timeout)
        print(response)
        logged = '0,1'
        if logged in response:
            print('SIM bereits im Netz eingeloggt')
            print()
        else: # einloggen...
            cmd = b'AT+CPIN="'+pin+b'"'
            send_command(uart, cmd)
            response = read_response(uart1, b'SMS Ready\r\n', timeout)
            print(response)
        sleep(2)

    # Message Format: Textmode...
    if not time_out_flag:
        cmd = b'AT+CMGF=1'
        send_command(uart, cmd)
        response = read_response(uart1, b'OK\r\n', timeout)
        print(response)
        sleep(2)

    # Text Mode: SMS-SUBMIT, Validity 1 day
    if not time_out_flag:
        cmd = b'AT+CSMP=17,167,0,0'
        send_command(uart, cmd)
```

```
        response = read_response(uart1, b'OK\r\n', timeout)
        print(response)
        sleep(2)

    # Adressat (Handy-Nr.)
    if not time_out_flag:
        cmd = b'AT+CMGS="'+phone_number+b'"'
        send_command(uart, cmd)
        response = read_response(uart1, b'>', timeout)
        print(response)
        sleep(2)

    # Nachricht
    if not time_out_flag:
        cmd = message + b'\x1A'  # \x1A = Ctrl-Z
        send_command(uart, cmd)
        response = read_response(uart1, b'OK\r\n', timeout)
        print(response)
        sleep(2)

def alarm(message, uart):
    global count
    global max_count
    global time_out_flag
    while count < max_count:
        print('count:', count) # zum Testen
        one_alarm(message, uart)
        if not time_out_flag: # wenn kein Time-Out vorliegt, Schleife abbrechen
            print('Alarm-SMS wurde an', phone_number, 'gesendet!')
            print()
            display.text(font1, 'Alarm-SMS gesendet', 5, 90)
            break
        else:
            display.text(font1, 'Alarm-SMS: Error', 5, 90)
            display.text(font1, 'Neuer Versuch in '
                                + str(delay_after_alarm) + 's  ', 5, 110)
            count =+ 1 # sonst count inkrementieren und neuen Versuch starten
            time_out_flag = False


################################################################################
#                                 Hauptprogramm                                #
################################################################################

display.fill(0)
print('SMS-Alarm-Programm bereit!')
print('Alarm wird mit Taster0 ausgelöst...')
display.text(font2, 'SMS-Alarm', 40, 5)
display.text(font1, 'Alarm -> Taster Ta0', 5, 50)

while True:
    if (taster0.value() == 0): # wenn Taster Ta0 betätigt
```

```
print('Alarm!!!')
display.text(font1, 'Alarm ausgeloest...', 5, 70)
alarm(message      , uart1)
print('Erneuter Alarm nach', str(delay_after_alarm), 's möglich...')
display.text(font1, 'Alarm moeglich in ', 5, 110)
for i in range(delay_after_alarm): # Countdown der Wartezeit
    display.text(font1, str(delay_after_alarm-i) + ' s', 150, 110)
    sleep(1)
print()
display.fill(0)
print('SMS-Alarm-Programm bereit!')
print('Alarm wird mit Taster0 ausgelöst...')
display.text(font2, 'SMS-Alarm', 40, 5)
display.text(font1, 'Alarm -> Taster Ta0', 5, 50)
```

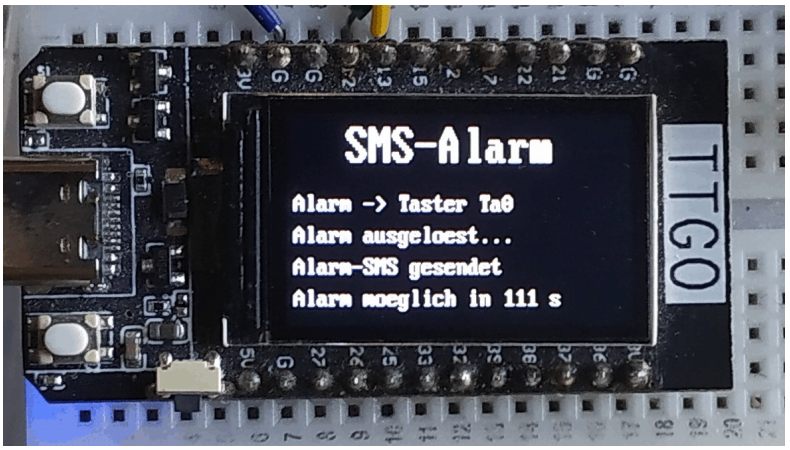

Abb. 5: TTGO hat Alarm-SMS gesendet

5 SMS empfangen

Zugegeben: Die Überschrift ist eigentlich etwas irreführend; denn zum eigentlichen Empfang einer SMS gibt es nicht viel zu sagen: Sobald ein Handy, insbesondere auch unser SIM800L mit dem GSM-Netz verbunden ist, nimmt es automatisch jede SMS an, die an es gerichtet ist. Die Frage ist also eher: Wie kann man eine empfangene SMS über ein Terminal zur Anzeige bringen?

Wie üblich verbinden wir zunächst unser SIM800L-Modul mit dem GSM-Netz, indem wir unsere PIN benutzen; die Eingaben bzw. deren Echos schreiben wir im Folgenden fett:

```
AT+CPIN="xxxx"
OK

AT +CMGF=1
OK
```

Wenn Sie das Programm `sms_terminal.py` benutzen, werden diese Schritte nach dem Programmstart automatisch ausgeführt. Eingehende SMS werden automatisch auf der SIM-Karte oder einem anderen Speicher abgelegt. Die möglichen Speicher kann man mit

`AT+CPMS=?` (Suchwort: OK)

abfragen. Unser SIM800L-Modul speichert die SMS auf der SIM-Karte in der Rubrik “SM”. Geben wir den Befehl

`AT+CPMS="SM"` (Suchwort: OK)

ein, erhalte ich als Antwort bei meiner SIM-Karte:

```
+CPMS: 0,10,0,10,0,10
```

Nur die ersten beiden Parameter sind hier relevant. Sie besagen, dass im Speicher der SIM-Karte 0 von maximal 10 möglichen Botschaften vorliegen.

Wenn wir nun eine SMS an unser SIM800L-Modul senden, erhält es die Meldung

```
+CMTI: "SM",1
```

Dabei zeigen die Parameter an, dass die Botschaft auf der SIM-Karte unter der Nummer 1 (in der Rubrik “SM”) abgespeichert wurde. Um diese Meldung brauchen wir uns aber nicht weiter zu kümmern. Wir fragen jetzt lieber direkt die Informationen für die SMS mit der Nummer 1

auf dem SIM-Karten-Speicher mit dem Kommando

`AT+CMGR=1` (Suchwort: OK)

ab und erhalten als Antwort (Suchwort: OK):

```
+CMGR: "REC READ","+49177xxxxxxx","","15/12/17,16:17:45+04"
Test-SMS an SIM800L
OK
```

Der erste Parameter zeigt an, dass die SMS durch das Anzeigen bzw. Lesen den Zustand `"REC READ"` (received read messages, d. h. gelesene Nachrichten) erhalten hat; der zweite Parameter gibt die Telefonnummer des Handys an, von dem die SMS verschickt wurde; der dritte und der vierte Parameter geben das Datum und die Uhrzeit der SMS an. Als letzter Parameter wird der Text der Botschaft angegeben.

Gibt man bei dem Befehl `AT+CMGR=nr` für `nr` eine Speichernummer an, in der keine Botschaft abgespeichert worden ist, ist die Antwort lediglich `OK`.

Das Kommado

`AT+CPMS="SM"` (Suchwort: OK)

liefert jetzt übrigens - wie erwartet - das Ergebnis:

```
+CPMS: 1,10,1,10,1,10
```

Man kann auch sämtliche vorhandenen Botschaften auflisten lassen; der Befehl dazu lautet:

`AT+CMGL="ALL"` (Suchwort: OK)

Nun werden alle SMS der Reihe nach inklusive ihrer Speicherplatznummer zur Anzeige gebracht; im Gegensatz zum `CGMR`-Kommando wird hierdurch aber nicht der Lese-Zustand von `REC UNREAD` zu `REC READ` geändert.

Irgendwann ist der Speicher der SIM-Karte voll. Spätestens dann ist es an der Zeit, einzelne Nachrichten zu löschen. Mit dem Befehl

`AT+CMGD=3` (Suchwort: OK)

wird zum Beispiel die SMS mit der Nummer 3 gelöscht.

6 Telefonieren mit dem SIM800L

Bislang haben wir den SIM800L-Baustein nur benutzt, um digitale Daten auszutauschen. Kann man mit diesem Modul aber auch wie mit einem Handy telefonieren?

Ja, man kann! Tatsächlich findet man auf der Rückseite der Platine links oben Abkürzungen, die auf diese Möglichkeit hinweisen: MICP, MICN, SPKP und SPKN. Diese Bezeichnungen machen deutlich, dass an den darunter befindlichen Kontakten ein Mikrofon (MIC) und ein kleiner Lautsprecher (SPK = Speaker) angeschlossen werden können (Abb. 6).

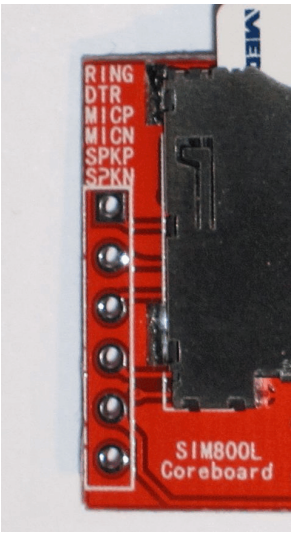

Abb. 6

Bei EBAY habe ich mir (vor einiger Zeit) kleine 8-Ohm-Lautsprecher (5 Stück für ca. 2 Euro) und Elektret-Mikrofone (10 Stück für ca. 1,60 Euro) besorgt.

Die Lautsprecher habe ich an die Pins SPKP (Speaker Positive) und SPKN (Speaker Negative) angeschlossen. Die Bezeichnungen Positive und Negative weisen auf einen Differenzverstärker hin; für das Anschließen des Speakers spielt die Polung keine Rolle. Der Hersteller des SIM800L rät zwar zu einer zusätzlichen Beschaltung mit Kondensatoren. Ein Test zeigte aber, dass diese nicht unbedingt erforderlich ist.

Beim Anschließen der Elektretmikrofone kann man leider nicht auf eine solche Beschaltung verzichten. Die Empfehlung des Herstellers ist folgende Schaltung:

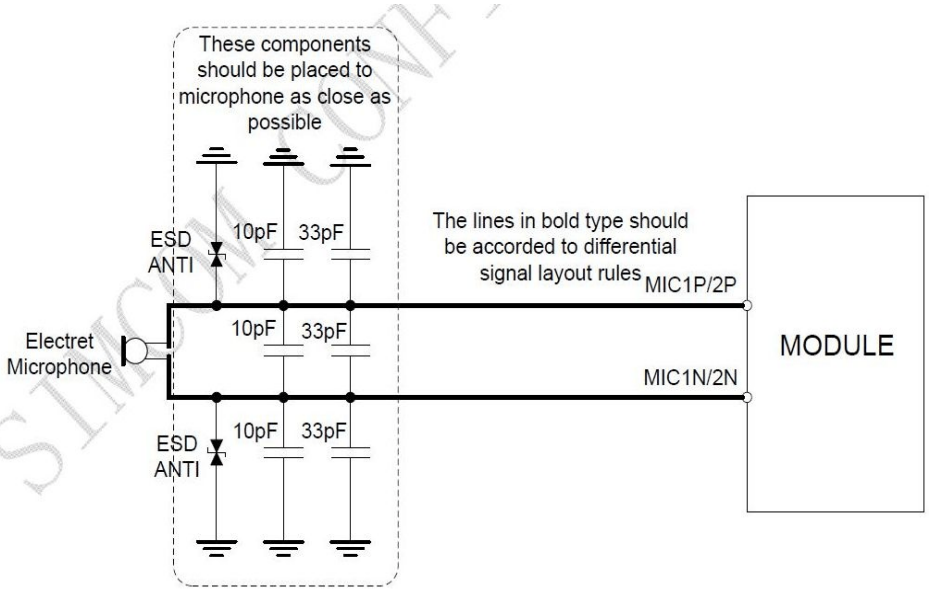

Abb. 7: Quelle: sim800h_hardware_design_v1.00.pdf des Herstellers

Der Einfachheit halber habe ich die parallel geschalteten Kondensatoren jeweils durch einen einzigen Kondensator von 15 pF ersetzt. Die Abschirmung des Kabels habe ich über einen solchen Kondensator mit der Masse des SIM800L verbunden. Auf den ESD-Schutz habe ich verzichtet. Trotz der Einsparungen bei der Beschaltung konnte so eine akzeptable Tonqualität erzielt werden.

Welche Kommandos sind nun für ein Gespräch erforderlich? Dies möchte ich für den Fall, dass das SIM800L-Modul angerufen wird, erläutern. Der Einfachheit halber wird das SIM800L-Modul hier über einen FTDI232-Baustein mithilfe des Programms Hterm gesteuert (vgl. Kapitel 1).

Wie üblich muss das Modul zunächst im Netz mit Hilfe der PIN eingeloggt werden:

```
AT+CPIN="xxxx"
```

Wenn das Modul im Netz eingeloggt ist, dann blinkt die LED im 3000 ms - Rhythmus, und das Modul meldet:

```
OK

+CPIN: READY

Call Ready

SMS Ready
```

Damit ist das SIM800L-Modul in Bereitschaft versetzt. Nun rufen wir es an; dieser Anruf wird auf dem Terminal durch die beiden Meldungen

```
RING

+CLIP: "0177xxxxxxxxx",129,"",0,"",0
```

angezeigt.

Diese Meldung wird fortwährend wiederholt - ähnlich dem Klingeln eines angerufenen Telefons; die Meldungen hören auf, wenn man am Terminal den Befehl

```
ATA
```

eingibt (`ATA` = **AT**tention **A**nswer). Dadurch wird das Gespräch entgegengenommen; auf dem Terminal erhalten wir ein `OK`.

Will man das Gespräch beenden, gibt man den Befehl

```
ATH
```

ein (**H** = hang up = Hörer auflegen). Das Terminal meldet dann

```
NO CARRIER
```

Auf genau dieselbe Weise erfolgt übrigens auch die Kommunkation zwischen (Hayes-kompatiblen) Modems.