VDIRECT
Spracherkennung für MSR

Ich kann mich noch gut daran erinnern: Als der Frühjahrskatalog ’99 von Conrad Elektronik kam, fiel mein Blick auf das neue Spracherkennungsmodul VDIRECT. Die Abbildung mit der Dame, die das Zimmer betritt und per Sprechblase ‚Licht ein‘ sagt, war genau das, was in einer erfolgreichen Palette der Steuerelektronik fehlte. Gesprochene Befehle erkennen und dann per Software reagieren – das wär’s.

Spracherkennung per Computer ist ein lang gehegter Wunsch des Menschen. Jeder möchte wohl gerne dem Computer etwas sagen und der versteht ihn aufs Wort. Schon 1983 hatten wir eine Erkennungssoftware im Apple - Computer. Das Vokabular bestand aus 192 Wörtern. Wollte man mehr erkennen, so mußte man den Speicher mit einem anderen Vokabular laden. Prinzipiell funktionierte die Erkennung – manchmal -. Doch jede kleinere Änderung des Umweltgeräusches ließ die Erkennung versagen: Man mußte dann, wenn man die Erkennung an einem anderen Ort starten wollte, alles neu trainieren.

Natürlich war damals der Rechner bedeutend langsamer als die heutigen Pentiumrechner. Auch der Speicherplatz war sehr beschränkt. Doch die notwendigen Algorithmen zur Spracherkennung (Mustervergleich) waren damals in ihrer Grundfunktion bereits implementiert.

Lange Jahre passierte relativ wenig. Die Erkennung per Computer blieb eine Geheimwissenschaft. Anfang der neunziger dann wagten sich einige Firmen in das Alltagsgeschäft. Die Firma IBM brachte damals einen Erkenner auf den Markt, der für das eingeschränkte Vokabular von Befundärzten in der Radiologie gedacht war. Es ist klar: je weiter das Vokabular von der Alltagssprache entfernt ist, desto größer ist die Chance für die Spracherkennung.

Probleme gab es verständlicherweise bei ähnlich klingenden Wörtern wie Wein, Bein, sein, mein, klein, fein, Rhein usw. Für die Radiologen gibt es Insuffizienz, Fraktur oder Thorax usw. – Wörter, die gut auseinanderzuhalten sind.

Ich konnte einmal die Befundärzte im Klinikum in Aachen beobachten, wie sie die etwa 150 Röntgenaufnahmen eines Vormittags befundeten. Nur einige Ärzte vertrauten dem neuen Medium, der Rest des Ärztepersonals diktierte noch in ein Diktiergerät, dessen Inhalt später abgetippt wurde.

Der Vorteil der direkten Spracheingabe liegt auf der Hand: die entstandenen digitalen Dateien konnten direkt in das Kliniknetz eingespielt werden, so daß der behandelnde Arzt in der Station über die Befundwerte verfügte.

Die abgetippten Ergebnisse waren dagegen meist erst am nächsten Tag verfügbar.

Das, was den Profis vorbehalten war, wanderte dann ab Mitte der neunziger Jahre in die normalen Personalcomputer und sogar als reine Softwarelösung. Plötzlich konnte jedermann seinen Computer per Sprache bedienen; manche schafften es sogar ganze Diktate per Sprache in den Computer zu bringen.

Zufriedenstellend ist es jedoch bis heute noch nicht – allzuviele Fehler werden erkannt, wenn der Wortschatz zu breit gefächert ist. Erstaunlich ist es dennoch, daß es überhaupt so gut funktioniert, denn die Software dazu muß einiges leisten.

Selbst wenn die Erkennung perfekt funktionieren würde, so wäre diese Methode sicherlich keine Alternative für den Einsatz in MSR – Lösungen. Wer würde hierfür schon einen kompletten Pentium mit einigen Megabyte opfern, nur um das Licht ein- oder auszuschalten. Insofern war diese Anzeige im Katalog vielversprechend: ein einzelner Chip, kaum Stromverbrauch, kaum Platzbedarf, billiger als ein Computer – das ist die Lösung.

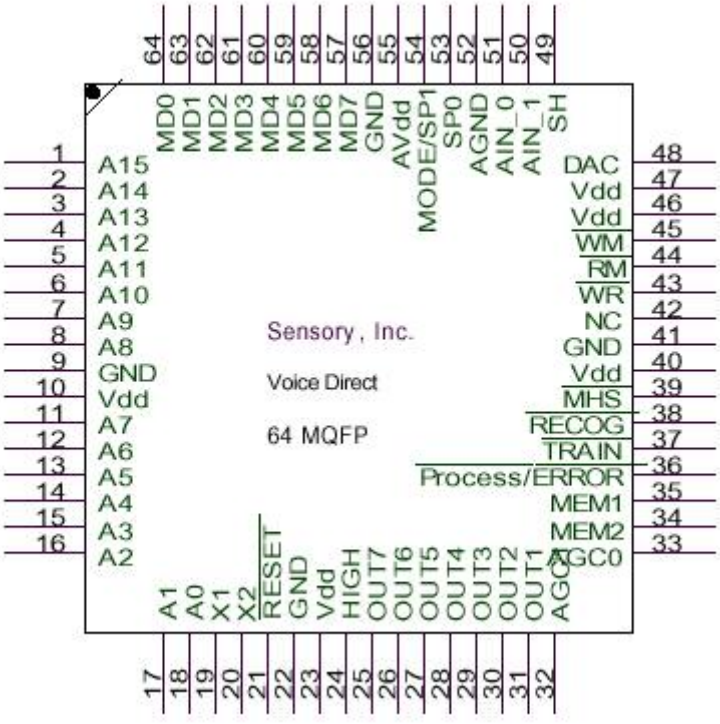
Ich mußte 14 Tage auf die Elektronik warten, bevor ich die ersten Tests starten konnte. Die Nachfrage muß sehr hoch gewesen sein.

Der Aufbau der kleinen Schaltung aus dem beigefügten Datenblatt war schnell realisiert.
Doch schon schnell fühlte ich mich von der Anzeige im Katalog getäuscht.

Was ist das für ein Spracherkenner, der zuerst mit einem Taster gestartet werden muß, damit man danach ein Wort sprechen kann. Es stellte sich auch bald heraus, daß dies nur ein Experimentierkit war, um die Funktionen zu verstehen. Das Internet half dann weiter. Bei sensory.inc.com lud ich das zum Vdirect – Chip gehörende Databook (CD: **Lallus\ Datenblätter\vdirdatabook.pdf**).

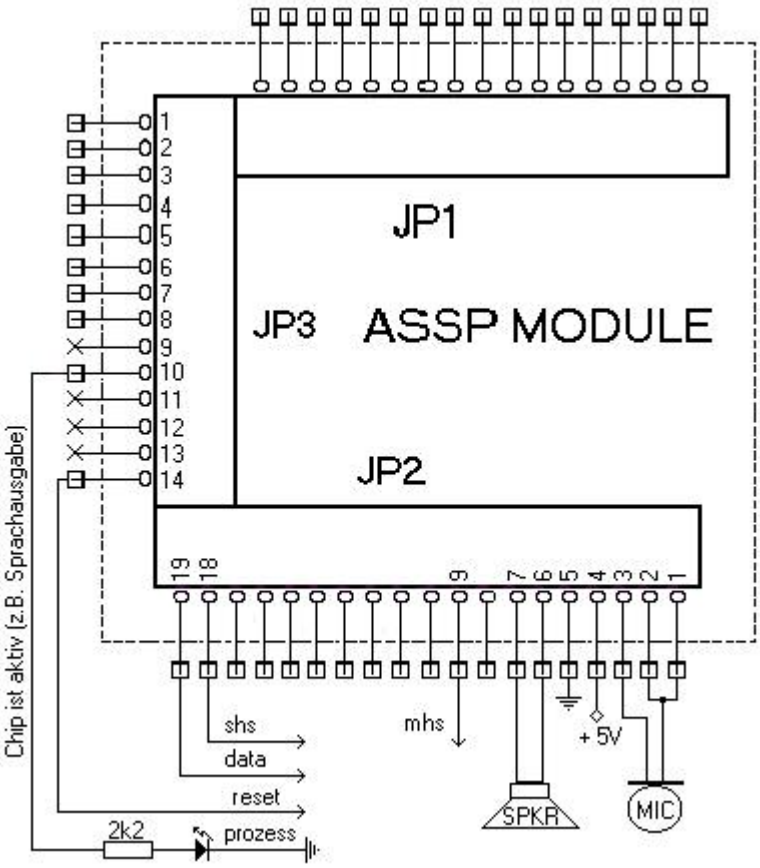
Hier ist genau beschrieben, wie man den Chip im Hostmodus betreiben kann.<Nur so macht er eigentlich Sinn. Plötzlich hat man dann auch aus den 15 Segmenten zur Erkennung 60 zur Verfügung. Die CControl eignet sich hier hervorragend, um den Hostbetrieb zu realisieren.

Die Hardware



Der 64 – pin Chip mit seinen Anschlüssen.

Zum Glück muss man den SMD – Chip nicht selbst auflöten. Dies könnte manchem Schwierigkeiten bereiten. Zum Glück auch muss man nicht alle Pins selbst beschalten. Zum Einsatz des Chips mit einem Microcontroller als Hostcomputer sind nur wenige Strippen zu ziehen.



Nur wenige Leitungen müssen angeschlossen werden.

Der Vdirect – Bus

In dem beigefügten Beispielprogramm werde ich Schritt für Schritt die einzelnen verfügbaren Modi beschreiben. Das Programm **vdirtest.bas** auf der CD zeigt dann menügesteuert die einzelnen Funktionen genau an.

Die Kommunikation mit dem Chip läuft über einen Dreidrahtbus, der so ein paar seltsame Spezialitäten hat, die sich beim Programmieren etwas quer stellen.

Drei digitale Ports müssen mindestens von der CControl zur Verfügung gestellt werden:
Will man den Chip softwaremäßig resetten können, so muß man noch einen vierten Port opfern.

Die CControl fungiert als Master.
Der Vdirect – Chip ist der Slave.

data – hierüber werden bitweise die Daten geschickt.
mhs – **M**aster **H**and**s**hake kontrolliert den Datenfluß vom Master zum Slave
shs – **S**lave **H**and**s**hake kontrolliert den Datenfluß vom Slave zum Master.

Die Kommunikation findet jeweils über fest definierte Datenpakete statt.

Byte	Value	Notes
0	FFh	Sync Field, 8 or more consecutive 1 bits
1	05h	Packet length, count of bytes to follow
2	01h	First data byte
3	02h	Second data byte
4	03h	Third data byte
5	04h	Forth data byte
6	0Fh	Checksum, (5+1+2+3+4)

Zum Teil ist die Kommunikation etwas umständlich, da immer wieder Synchronbytes gesendet werden müssen. In der obigen Tabelle erkennt man das Format eines Datenpaketes. Zunächst einmal muss immer (Byte 0) ein gefülltes Byte (255) bitweise gesendet werden. Die 8 Bit mit 1 werden offenbar vom Chip zur Synchronisation benötigt. Damit ist klar: Byte 0 muß immer mit 255 gefüllt werden.

Byte 1 vermittelt dann dem Vdirect-Chip die Information, wie viele Bytes nachgeschickt werden. Der Slave kann sich dann darauf einstellen. Im obigen Fall in der Tabelle wird ein Byte mit dem Wert 5 bitweise geschickt. Der Slave weiß nun, daß nach 5 folgenden Bytes das Datenpaket beendet ist.

Jetzt erst folgt das erste Datenbyte. In der Tabelle steht hier 01 hex, also eine 1.
Dann das zweite (02 hex), dritte (03 hex) und vierte (04 hex) Datenbyte.

Wichtig ist jetzt die Addition aller vorher gesendeten Bytewerte inclusive dem Längenbyte (Byte 1). Hier wird die sogenannte Checksum gespeichert. Sie muß genau stimmen, sonst wird die Anforderung an den Slave verworfen.

In dem obigen Fall müssen wir also die 5 aus dem Längenbyte mit der 1,2,3 und 4 aus den Datenbytes addieren. Gesendet wird daher als letztes Byte die errechnete Checksum 15 - natürlich wieder bitweise über die Dataleitung unter Kontrolle des MHS – Masterhandshake.

Das obige Beispiel ist kein typisches Beispiel zur Ansteuerung des Chips, da eine solche Kommandoabfolge zu nichts führt. Wir setzen jetzt einmal ein echtes Kommando ein.

Command 01h - (Get Version)
Arguments: ~
Returns: **Version String (56h, 44h, 49h, 01h, 01h)**
Responses: **00, 21**

This command returns a five-byte sequence indicating the Sensory product model and software version number. The first three bytes will always be 56h, 44h, and 49h (ASCII 'VDI') for the Voice Direct. The last two bytes are, respectively, the major and minor version numbers in binary format.

Das Kommando Get Version liefert als Antwort 5 Bytes vom Slave an den Master. Das Protokoll für die Darstellung der Antwort werden wird dann anschließend beschrieben. Zunächst einmal soll die Anforderung gesendet werden: Liefere mir die Versionsnummer des Vdirect – Chips..

Zunächst wieder das Synchronbyte Byte 0 mit FF hex oder 255 dezimal gefüllt.

Das Kommando **Get Version** verfügt über keine Argumente wie andere Befehle.

Gesendet wird also nur ein Datenbyte 01h. Da das Checksumbyte aber bei der Berechnung der zu sendenen Bytes (Längenbyte) berücksichtigt werden muss, folgen also Datenbyte und Checksumbyte – also 02h oder 2 dezimal. Danach kommt das Datenbyte und schließlich das Checksumbyte.

Bit 0 = 255
Bit 1 = 2
Bit 2 = 1
Bit 3 = 3

Das Kommando wurde jetzt komplett versandt und man kann vom Chip jetzt die Antwort abholen.

Da wir die Bytewerte einzeln als Bit versenden müssen und den Inhalt über die quasi als Clockleitung zu bezeichnende Handshakeleitung **mhs** dem Chip mitteilen müssen, müssen die einzelnen Bytes in ihre Bitwertigkeit aufgelöst werden. Aus anderen Applikationen kennen Sie dieses Verfahren vielleicht.

Das Unterprogramm **schreiben** wurde so realisiert:

```
#schreiben
for i = 7 to 0 step - 1
if wert and 1 shl i then data =on else data = off
wait shs
mhs=off
wait not shs
mhs = on
wait shs
next i
print wert; "  ";
return
```

Das Byte wird über die FOR i -Schleife in 8 Bits aufgelöst. Sie erinnern sich an den Basicbefehl **shl**. Er bedeutet 1 shl 7 =128. 1 shl 6 = 64 usw. bis 1 shl 0 =1.

Die Variable **wert** wird so in 1en und 0en aufgelöst.
Bei der **if** – Abfrage wird nun je nach Bitwert die Dataleitung auf 1 oder auf 0 gesetzt.
Jetzt wird die Kommunikation über die Handshakeleitungen eingeleitet. Das Verfahren wird im Datenblatt so beschrieben:

Data is transferred one bit at a time with full handshaking as described below.

1. When the MCPU has data to transmit to the VDI, the MCPU sets DATA to the data value, verifies that SHS (Slave Handshake) is in the *high* state, then sets MHS (Master Handshake) to the *low* state to request a transfer.

2. The VDI senses the *low* state of MHS and reads DATA, which then sets SHS to the *low* state to acknowledge the DATA.

3. The MCPU senses the *low* state of SHS, and sets MHS to the *high* state to indicate that DATA is no longer valid, and at the same time sets DATA high (releasing it).

4. The VDI then sets SHS to the *high* state to indicate that the cycle is complete. Both devices are now ready to transfer the next data bit.

Schritt 1:
Wenn der Master Daten zum Slave schickt, so werden die Daten bitweise angelegt, wie dies oben gezeigt wurde.

```
 #schreiben
for i = 7 to 0 step - 1
if wert and 1 shl i then data =on else data = off
```

Hier wird das Byte, das gesendet werden soll bitweise zerlegt. Danach versichert sich der Master, daß **shs** high ist: **wait shs.**

Der nächste Schritt: Der Master fordert eine Datenübertragung an. Die Handshakeleitung **mhs** wird auf 0 gesetzt – das Signal für den Slave, daß jetzt etwas passiert: **mhs = off**.

Schritt 2.
Der Slave (Vdirect – Chip) entdeckt den Potenzialwechsel auf der Leitung und übernimmt das gesetzte Datenbit **data.** Danach wird **shs** vom Slave auf 0 gesetzt**,** um zu signalisieren: ich habe verstanden, das Bit ist bei mir angekommen.

Schritt 3.
Für den Master bedeutet dies: Die Leitung wird auf den Potenzialwechsel überprüft.
wait not shs.
Ist der Wechsel erkannt, so wird vom Master die Handshakeleitung **mhs** wieder auf 1 gelegt.
mhs = on. Dies bedeutet, daß **data** jetzt nicht mehr gültig ist und das Bit für den Master übertragen ist.

Schritt 4.
Der Slave setzt jetzt **shs** auf 1, um zu signalisieren, daß er alles verstanden hat und das nächste Bit gesendet werden kann. Der Master wartet so lange, bis er **shs = on** entdeckt hat. **wait shs** ist der Befehl dazu.

Das nächste Bit kann übertragen werden.
Dann folgt nach 8 Bits das nächste Byte.

Damit ist das Übertragen von Kommandos erklärt. Was muss jetzt getan werden, damit man die entsprechenden Reaktionen aus dem Chip empfangen kann. Es ist genau umgekehrt.

Oben wurde am Anfang **mhs** aktiv. Jetzt ist es der Chip, der über **shs** die Aktion einleitet.

Damit gilt: der Prozessor, der seine Handshakeleitung zuerst auf 0 setzt, ist der Sender, der andere Prozessor ist der Empfänger.
Der VDI – Prozessor wird jedoch niemals aktiv, wenn der Master die Daten nicht angefordert hat.

Voice Direct remains busy (SHS = low state) after receiving the final bit of a command packet, until after that command has been completed and Voice Direct is ready to send a response. During this time, a time-consuming command can be interrupted by the MCPU with a low pulse on the MHS line. When Voice Direct has data to transmit to the MCPU, it follows the same procedure by setting SHS to the low state. (See Figures 2, 3, and 4.) The protocol is completely symmetrical. The first processor to set its HS signal to the low state is the transmitter; the other processor is the receiver.

Hier noch einmal die Routine **#schreiben**

```
#schreiben
for i = 7 to 0 step - 1
if wert and 1 shl i then data =on else data = off
wait shs
mhs=off
wait not shs
mhs = on
wait shs
next i
print wert; "  ";
return
```

Die Routine **#lesen** ist hier abgebildet.

```
#lesen
wert=0
deact data
for i = 7 to 0 step -1
wait not shs
if data  then wert = wert + (1 shl i)
mhs = off
wait shs
mhs = on
next i
print wert;"  ";
return
```

Diese Leseroutine muß sicherlich nicht genauer erklärt werden, da sie analog zur Schreibroutine aufgebaut ist. Unterschiedlich ist hier nur, dass der digitale Port **data** jetzt als Eingang definiert wird, damit die Daten auf der Leitung abgeholt werden können.

Noch einmal eine kurze Zusammenfassung der Übertragung eines Kommandos an den Slave, den VDI-Chip.

1. Synchronbyte aus 8 * 1 = 255 oder FFhex.
2. Es folgt das Längenbyte, bestehend aus der Anzahl der Folgebytes.
3. Danach das Datenbyte oder die Datenbytes, die das Kommando ausmachen.
4. Das Checksumbyte.

Kommen wir jetzt noch einmal zurück zu dem Kommando **Get Version** von oben. Das Kommando lautet : 01h.

```
Wert = 255 : gosub schreiben
Wert = 2   : gosub schreiben
Wert = 1   : gosub schreiben
Wert = 3   : gosub schreiben
```

Damit ist eine Anforderung an den Chip gesendet. Wir holen jetzt die Antwort ab.

```
Gosub lesen
print wert
```

Es werden 8 Bits mit einer 1 zurückgesendet. In **`wert`** steht jetzt also 255.
Wir lesen das nächste Byte, das geschickt wird. Hier steht nun der Anzahl der folgenden Bytes , das Längenbyte. Diesen Wert merken wir uns in der Variblen **`anzahl`** denn das ist genau die Anzahl Bytes, die wir lesen müssen.

```
gosub lesen
print wert
Anzahl = Wert

For i = 1 to anzahl
Gosub lesen
Print wert
Next i
```

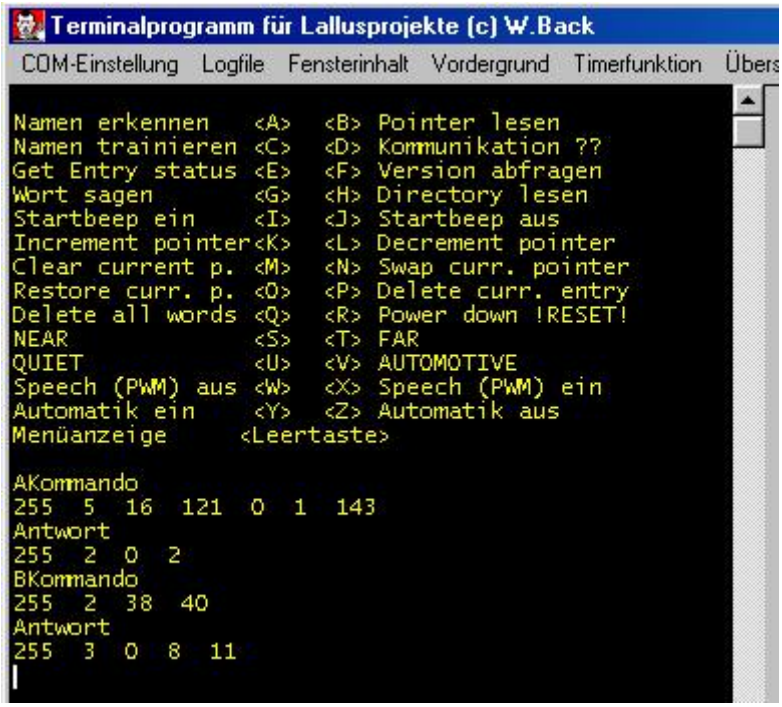
Ein Ausschnitt aus dem Terminalfenster zeigt das Ergebnis der Aktion. Mit der Eingabe des Buchstabens F wird die Version abgefragt. Das Kommando 255 2 1 3 wird, wie oben beschrieben, zum VDI gesendet.

Die Antwort ist entsprechend. Das erste Byte ist immer 255. Das Längenbyte 7 zeigt an, daß 7 Bytes gesendet werden. Das nächste Byte entspricht einer 0; es bedeutet, daß kein Fehler aufgetreten ist. Der ASCII-Wert 86 entspricht einem „V“, 68 einem „D“ und 73 einem „I“. Danach folgt die Versionsnummer 2.1. Das letzte Byte –237- ist das bekannte Checksumbyte, das sich aus 7+0+86+68+73+2+1 zusammensetzt.

Das Checksumbyte kann im Programm benutzt werden, um die Datenübertragung zu überprüfen. Zählt man alle gesendeten Bytes zusammen und kommt am Ende eine andere Checksum heraus, so ist irgend etwas schief gelaufen.

Jetzt soll es etwas interessanter werden. Der Chip soll eine Spracherkennung vornehmen. Vorab hatte ich die Zahlen 1 bis 20 trainiert und bei der Aufforderung des Chips **‚say a word'** – also ‚sage ein Wort zur Erkennung' neun in das Mikrofon gesprochen.

Schauen wir uns das Ergebnis im Bild an.



Im Menü wurde ein ‚A' ausgewählt, das für ‚Namen erkennen' steht. Das Kommando ‚**recognize word**' wird mit dem Wert 10hex oder 16 dezimal eingeleitet. Danach lesen wir unter Arguments, wie sich das komplette Kommando zusammensetzt. ‚Prompt' bedeutet hier ein Wort zur Sprachausgabe, das in dem Chip bereits abgelegt ist. Es gibt hier sogar zwei Listen mit vordefinierten Wörtern: das sog. Mandatory – Vokabular und die Optional Prompt List. In dem ersten Vokabular sind 208 Segmente, in dem zweiten Vokabular sind 240 Segmente abgespeichert. Im Datenblatt ist dies genau zu sehen.

Die Mandatory – Liste spricht man mit 0, die Optional Prompt Liste mit 1 an. Nach der 16 kommt also die 121 aus der 0 – Liste (Mandatory). Dieses entspricht ‚**say a word**'. In der Argumentenliste finden wir noch den Eintrag **‚tries'**, die Anzahl der Versuche, die zur Erkennung des gesprochenen Wortes unternommen werden sollen. Hier steht der Wert auf 1 = nur ein Versuch. Das Checksumbyte ist bekannt.

Command 10h - (Recognize Word)	
Arguments:	**Prompt, Source, Tries**
Returns:	~
Responses:	**00, 01, 02, 03, 04, 05, 06, 07, 08, 10, 21, FF**

Nach der Erkennung kommt die Antwort. 255, Längenbyte 2, Fehler 0 (erfolgreiche Erkennung), Checksum 2.

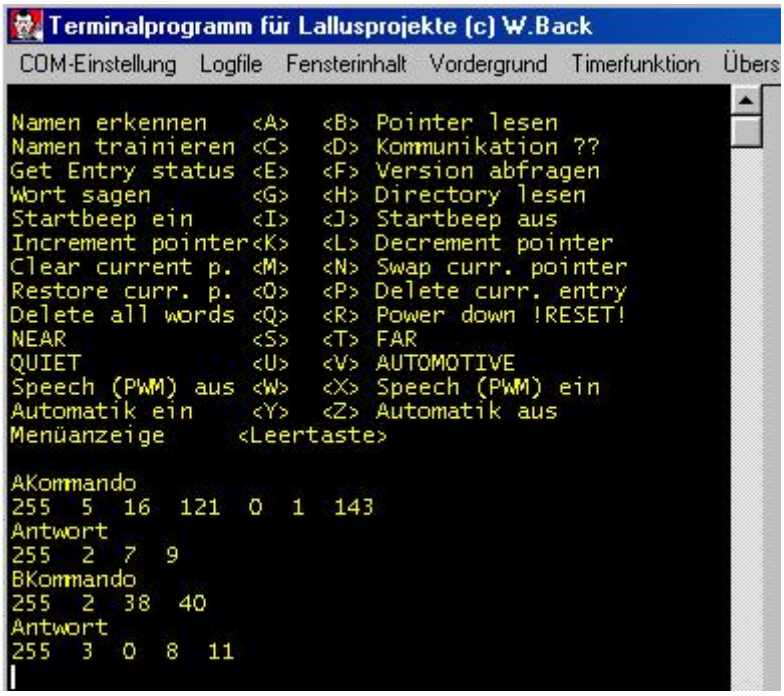
Es folgt nun ein zweites Kommando. Bei erfolgreicher Erkennung wird der interne Zeiger im Chip auf das erkannte Segment gestellt. Diesen **‚Current Pointer'** kann man jetzt abfragen, um zu erfahren, was erkannt wurde. Das erste Segment beginnt mit 0, so dass wir bei der gesprochenen neun 9 - 1 = 8 lesen müssen.

Das Kommando **‚Return Current Pointer'** wird mit 26h oder 38 dezimal eingeleitet. Es hat keine weiteren Argumente. Wir können jetzt das Kommando auslesen. 255, Längenbyte 2, Kommando 38, Checksum 40.

Die Antwort ist nun auch zu interpretieren. Synchronbyte 255, Längenbyte 3, 0 = kein Fehler, 8 ist die Zeigerposition, Checksum 11. Jetzt sind wir immerhin schon soweit, daß wir ein gesprochenes Wort auswerten können.

Wenn wir den Fehler 0 (bei der Antwort auf das erste Kommando) betrachten und den zugehörigen Zeiger (bei der Abfrage des ‚Current Pointer') auswerten, so könnten wir damit schon einen Port nach draussen bedienen und meinetwegen die Berieselungsanlage einschalten.

Ein zweites Beispiel soll hier noch gezeigt werden, bei dem die Erkennung nicht geklappt hat. Ich sagte in das Mikrofon ‚Hallo', ein Wort, das gar nicht gespeichert ist.



Wir sehen hier bei der Antwort auf das erste Kommando 255 2 7 9. Diese 7 ist eine Fehlermeldung des Chips, die gleichbedeutend ist mit: nicht erkannt.

Bei der zweiten Antwort sehen wir, daß der Pointer (die 8) nicht bewegt wurde.

So können alle Kommandos des Datenbuchs auf der CD nacheinander ausprobiert werde. Kommando senden, Antwort holen usw. Das Basicprogramm vdirtest.bas ist hier abgebildet. Es würde zu weit führen, jedes einzelne Kommando ausführlich zu beschreiben.

```
define reset port[8]
define shs   port[7]
define data  port[6]
define mhs   port[5]


define wert    byte
define anzahl  byte
define check   byte
define i       byte
define j       byte
define offset  byte
define taste   byte
define segment byte
define wort    byte
define manda   byte
define erkannt bit[192]
define speek   bit[191]
define sagen   bit[190]

#init
deact shs : reset =off : reset =on
pause 30

#anzeige
print
print "Namen erkennen   <A>  <B> Pointer lesen"
print "Namen trainieren <C>  <D> Kommunikation ??"
print "Get Entry status <E>  <F> Version abfragen"
print "Wort sagen       <G>  <H> Directory lesen"
print "Startbeep ein    <I>  <J> Startbeep aus"
print "Increment pointer<K>  <L> Decrement pointer"
print "Clear current p. <M>  <N> Swap curr. pointer"
print "Restore curr. p. <O>  <P> Delete curr. entry"
print "Delete all words <Q>  <R> Power down !RESET!"
print "NEAR             <S>  <T> FAR"
print "QUIET            <U>  <V> AUTOMOTIVE"
print "Speech (PWM) aus <W>  <X> Speech (PWM) ein"
print "Automatik ein    <Y>  <Z> Automatik aus"
print "Menüanzeige     <Leertaste>"
print

#main
taste=0
if not rxd then goto main
get taste
if taste=32 then goto anzeige
if taste <65 then goto main
if taste >96 then taste = taste-32
offset = (taste-65)*5
if offset = 120 then goto automatik
if offset=30 then sagen =on
if sagen=on then goto sprechen else goto ausführen
#sprechen
print "Mandatory prompt list  <0> "
print "Optional  prompt list  <1> "
```

```
input manda
if manda>1 then goto sprechen
if manda = 0 then print "Das gewünschte Wort (1 -208) eingeben."
if manda = 1 then print "Das gewünschte Wort (1 -240) eingeben."
input wort

#ausführen
print "Kommando"
looktab command,offset,anzahl
wert=255 : gosub schreiben
wert = anzahl + 1 : gosub schreiben
check = wert
for j = 1 to anzahl
looktab command,j+offset,wert
if sagen=on and j=2 then wert=wort
if sagen=on and j=3 then wert=manda
check = check + wert
gosub schreiben
next j
sagen=off
manda=0
wert=check : gosub schreiben
print

#antwort
print "Antwort"
gosub lesen : gosub lesen
anzahl=wert
for j = 1 to anzahl
gosub lesen
next j
print
goto main

#lesen
wert=0
deact data
for i = 7 to 0 step -1
wait not shs
if data  then wert = wert + (1 shl i)
mhs = off : wait shs : mhs = on
next i
print wert;"  ";
return

#schreiben
for i = 7 to 0 step - 1
if wert and 1 shl i then data =on else data = off
wait shs
mhs = off
wait not shs
mhs = on
wait shs
next i
print wert; "  ";
return
```

```
#automatik
offset = 0
if rxd then get taste
if taste=90 or taste=122 then goto anzeige
if taste >96 then taste=taste-32
if taste =91 then goto anzeige
print "Kommando"
looktab command,offset,anzahl
wert=255 : gosub schreiben
wert = anzahl + 1 : gosub schreiben
check = wert
for j = 1 to anzahl
looktab command,j+offset,wert
check = check + wert
gosub schreiben
next j
wert=check : gosub schreiben
print
print "Antwort"
gosub lesen : gosub lesen
anzahl=wert
for j = 1 to anzahl
gosub lesen
if j=1  and wert=0 then erkannt=on
next j
print
if erkannt=on then goto erkannt
goto automatik

#erkannt
erkannt=off
print "Erkennung"
wert=255 : gosub schreiben
wert=2   : gosub schreiben
wert=38  : gosub schreiben
wert=40  : gosub schreiben
print
print "Antwort"
gosub lesen : gosub lesen
anzahl=wert
for j = 1 to anzahl
gosub lesen
if j=2 then segment=wert+1
next j
print
print "Erkennung"
wert=255 : gosub schreiben
wert=5   : gosub schreiben
wert=48  : gosub schreiben
wert=90+segment:gosub schreiben
wert=1   : gosub schreiben
wert=1   : gosub schreiben
wert=5+48+90+segment+1+1: gosub schreiben
print
print "Antwort"
gosub lesen : gosub lesen
anzahl=wert
```

```
for j = 1 to anzahl
gosub lesen
next j
print
goto automatik



table command
4 16 121 0 1 ' recognize word
1 38 0 0 0   ' return pointer
4 2 121 0 1  ' train a name
1 0 0 0 0    ' no operation
1 42 0 0 0   ' get entry status
1 1 0 0 0    ' get version
3 48 7 0 0   ' say a prompt
1 43 0 0 0   ' query directory
3 65 8 1 0   ' beep ein
3 65 8 0 0   ' beep aus
1 33 0 0 0   ' increment pointer
1 34 0 0 0   ' decrement pointer
1 32 0 0 0   ' clear current pointer
1 37 0 0 0   ' swap current pointer
1 36 0 0 0   ' restore current pointer
1 39 0 0 0   ' delete current entry
2 40 85 0 0  ' delete all stored words
1 64 0 0 0   ' power down
3 65 0 254 0 ' environment near
3 65 0 255 0 ' environment far
3 65 0 243 0 ' environment quiet
3 65 0 12 0  ' environment automotive
3 65 7 1 0   ' speech pwm aus
3 65 7 2 0   ' speech pwm ein
tabend
```

Um die Tabelle zu verstehen, seien hier noch ein paar Erklärungen gegeben. Alle Tabelleneinträge bestehen aus 5 Bytes, um hier ein einheitliches Format zu haben. Das Synchronbyte 255 wird im Programm selbst erzeugt; ebenso das Checksumbyte. Die erste Ziffer (z.B. 4) bedeutet, es folgen 4 Kommandobytes, danach das Kommando und die Parameter.

Assembler für VDIRECT

Natürlich macht es auch hier Sinn, die Software für die Voice Direct in Assembler zu schreiben. Das folgende Assemblerprogramm geht davon aus, dass man für die Steueraufgabe eine kontinuierliche Erkennung wünscht, der Mikrofoneingang wird ständig abgescannt. Wird ein Segment nicht erkannt, so wird die Erkennung innerhalb des Assemblers erneut gestartet. Wird dagegen ein Segment erkannt, so wird der entsprechende Pointereintrag gelesen und nach Basic ausgegeben. Damit wäre es z.B. dann leicht möglich eine Erkennung in Basic zu programmieren:

```
If ausgang = 1 then motor = on
If ausgang = 2 then motor = off
```

Das Assemblerprogramm enthält einige Spezialitäten, die nach dem Listing erklärt werden sollen.

```
* ****************************************
* Assemblerprogramm zum Ansteuern
* der VDIRECT - Unit. Es wird eine staendige
* Erkennung eingeschaltet. Das Assemblerpro-
* gramm springt nur nach Basic zurueck, wenn
* ein Segment erkannt wurde.
* Programmname: VOICEASM.ASM
* Compilat:     VOICEASM.S19
* Basic:        define ausgabe byte
*               #main
*               sys &H0101
*               print ausgabe+1
*               goto main
*               syscode "voiceasm.s19"
* (c) Wolfgang Back, 1999
*****************************************
bport   equ   $01          ; Portbyte (1-8)
bpdir   equ   $05          ; Richtungsbyte

mhs     equ   4            ; master hand shake
dat     equ   5            ; daten
shs     equ   6            ; slave hand shake
res     equ   7            ; reset

merker  equ   $91          ; Hilfsvariable
helper  equ   $93          ; Hilfsvariable
anzahl  equ   $95          ; Hilfsvariable
kennung equ   $97          ; Hilfsvariable

ausgabe equ   $A1          ; nach Basic
*****************************************************
        org   $101         ; Startadresse

        bset  res,bpdir    ; Reset als Ausgang
        bset  mhs,bpdir    ; mhs als Ausgang
        bclr  shs,bpdir    ; shs als Eingang


start    lda  #0           ; lade 0
         sta  anzahl       ; Tabellenzugriff

comwrite ldx  anzahl       ; lade Indexwert
         lda  recordtab,x  ; lese die Tabelle mit Index
         jsr  write        ; schicke es dem Chip
         lda  anzahl       ; lade die Anzahl
         cmp  #6           ; sind es 6?
         beq  comread      ; wenn ja, nach comread
         inca              ; erhoehe accu
         sta  anzahl       ; speichern
         bra  comwrite     ; verzweige nach comwrite

comread  jsr  read         ; lese 1. Antwortbyte (255)
         jsr  read         ; lese Anzahl Bytes (2)
         jsr  read         ; lese Erfolg (nur bei 0)
         lda  helper       ; wie sieht helper aus?
         sta  kennung      ; in kennung speichern
```

```
         jsr  read          ; lese Checksum
         lda  kennung       ; welche Wert?
         bne  start         ; wenn nicht 0, dann start

pointer  lda  #255          ; schreibe das Kommando
         jsr  write         ; fuer get current pointer
         lda  #2            ; 2 Bytes folgen
         jsr  write
         lda  #38           ; Kommando f□r pointer lesen
         jsr  write
         lda  #40           ; Checksum
         jsr  write

         jsr  read          ; Antwort einholen (255)
         jsr  read          ; Anzahl Bytes (2)
         jsr  read          ; Template
         jsr  read          ; pointer (1. Segment = 0)
         lda  helper        ; lade den Lesewert
         sta  ausgabe       ; uebergebe ihn nach Basic
         jsr  read          ; lese die Checksum
ende     rts                ; zurueck

*****************************************************

write    bset  dat,bpdir     ; dat als Ausgang
         ldx   #8            ; Schleifenzaehler
loop     bclr  dat,bport     ; dat auf 0
         lsla                ; shifte accu links
         bcc   w0            ; verzweige wenn 0
         bset  dat,bport     ; setze dat
w0       brclr shs,bport,w0  ; wait shs
         bclr  mhs,bport     ; mhs=off
w1       brset shs,bport,w1  ; wait not shs
         bset  mhs,bport     ; mhs=on
w2       brclr shs,bport,w2  ; wait shs
         decx                ; Schleife -1
         bne loop            ; verzweige bis 0
         rts                 ; nach Aufruf zurueck

*************************************************

read     bclr  dat,bpdir     ; dat als Eingang
         lda   #0            ; setze helper auf 0
         sta   helper
         ldx   #8            ; Schleifenzaehler
         lda   #128          ; Bit 7 zum Shiften
         sta   merker        ;
w3       brset shs,bport,w3  ; wait not shs
         brclr dat,bport,w5  ; 0 ? - dann nach weiter
         lda   helper        ; lade Hilfsregister
         add   merker        ; addiere merker  128,64,32 ...
         sta   helper        ; helper speichern
w5       bclr  mhs,bport     ; mhs=off

w4       brclr shs,bport,w4  ; wait shs
         bset mhs,bport      ; mhs = on
```

```
        lda merker          ; merker laden
        lsra                ; rechts shiften
        sta merker          ; merker speichern
        decx                ; Schleife -1
        bne w3              ; verzweige bis 0
        rts                 ; zurueck nach Aufruf


recordtab  fcb 255,5,16,121,0,1,143
```

Die 160 Bytes Assembler entsprechen den Basicroutinen. Es wird hier gezeigt, wie man den 6 Byte langen Kommandostring für recognize word aus einer Tabelle entnimmt. Die Tabelle `recordtab` liefert zunächst mit 255 das Synchronbyte. Danach kommt die Anzahl der zu sendenen Bytes (5), 16 (10h) ist das Kommando für recognize word, 121 ist der Eintrag say a word in der Mandatoryliste (0). Alles zusammengenommen ergibt die Checksum 143.

Später wird man wahrscheinlich eine andere Tabelle einfügen, um dem ständigen Gequatsche zu entgehen – auch um die kontinuierliche Erkennung zu beschleunigen. Ersetzt man die obige Tabelle mit:

```
recordtab  fcb 255,5,16,229,1,1,252
```

so hat man in der Optional List (1) das Segment 229 (0 msec) eingeschaltet, d.h. dass die Erkennung sich nicht mit der Sprachausgabe say a word aufhält und sofort zurückspringt. Die Fehlerausgaben (*you spoke too soon, please talk louder, word not recognized* usw.) sind damit jedoch noch nicht unterdrückt. Hier kann man, wie in Basic gezeigt, die Sprachausgabe komplett abschalten. Dies kann man entweder in Basic selbst vornehmen, oder man fügt die Kommandofolge noch am Anfang des Assemblers ein.