B. Kainka

C-Control-Hardware-Erweiterungen

- Tipps und Tricks zum Conrad-Steuercomputer -

Mit dem C-Control-Steuercomputer sind schon zahlreiche Aufgaben gelöst worden. Dieser vielseitige und preiswerte Einplatinencomputer ist für viele Anwender zu einem unverzichtbaren Hilfsmittel geworden. Und trotzdem gibt es immer noch etwas zu verbessern, zu erweitern und zu entwickeln. Mit dieser CD-ROM möchte ich Ihnen eine Reihe von Problemlösungen vorstellen, die sich vor allem aus Fragen von Lesern meiner beiden Bücher über C-Control entwickelt haben.

Diese CD ist für erfahrene C-Control-Anwender gedacht. Hier geht es nun um die speziellen Probleme, also um die Grenzfälle, in denen die C-Control-Unit auf den ersten Blick nicht mehr ausreicht. Es wird gezeigt, wie man das System in vielfacher Hinsicht "aufbohren", also erweitern und verbessern kann.

Was machen Sie, wenn Sie mehr als acht analoge Eingänge benötigen? Oder wie sind Aufgaben zu lösen, die sehr zeitkritisch sind? Wie schließt man zusätzliche Hardware an das System an? Was ist zu tun, wenn der Speicherplatz im 8K-EEPROM nicht mehr ausreicht? Wie kann das System mit eigenen PC-Programmen angesprochen werden? Wie weit kann der Stromverbrauch für Batteriebetrieb reduziert werden? Und was kann man tun, um C-Control komplett von einem PC aus zu steuern? Wenn Sie über diese Fragen schon gestolpert sind, dann sollten Sie hier nach den Lösungen suchen.

Dabei wünsche ich viel Erfolg!

Burkhard Kainka

Hinweise zum Start der Software:

Die Informationen auf der CD sind im HTML-Format aufbereitet. Sie können daher mit jedem Browser betrachtet werden. Auf der CD befindet sich der Franzis-Offline-Browser. Es wird direkt von der CD gestartet und hinterläßt keinerlei "Rückstände" auf der Festplatte Ihres PCs.

Wenn auf Ihrem Computer Windows 95/98 oder Windows NT installiert ist, erscheint nach dem Einlegen der CD nach einer Weile normalerweise automatisch der Startbildschirm der CD. Sollte das nicht der Fall sein, ist die Autostartfunktion ausgeschaltet, und Sie müssen über den Arbeitsplatz das CD-Laufwerk aufrufen und dort das Programm Browser98 durch Doppelklick starten. Haben Sie auf Ihrem Computer Windows 3.11 installiert, so müssen Sie das Programm Browser.exe starten.

Natürlich können Sie auch den Browser verwenden, der bereits auf Ihrem Rechner installiert ist. Allerdings müssen Sie dann wahrscheinlich auf einige Annehmlichkeiten verzichten, wie z.B. auf den direkten Aufruf von Programmen. Sie öffnen auf der CD das Verzeichnis HTML, und dort die Datei index.htm.

Bisher sind Sie bei der Entwicklung von Programmen wahrscheinlich folgenden Umgang mit der C-Control-Unit gewohnt:

1. Sie laden ein Programm aus der Entwicklungsumgebung (Ccew32d.exe). Während des Ladevorgangs leuchtet die rote LED.
2. Sie betätigen den gelben Start-Taster, um das Programm zu starten. Nun leuchtet die gelbe LED, die rote leuchtet oder flackert.
3. Sie testen Ihr Programm.
4. Sie brechen es mit dem roten Reset-Taster ab (beide LEDs erlöschen) und gehen eventuell wieder zu 1.

Mit dieser CD können Sie auch anders arbeiten. Sie testen nacheinander mehrere Programme direkt von der CD, ohne die Entwicklungsumgebung zu benutzen. Voraussetzung ist, dass Ihre CC-Unit an COM2 (=Default) oder an COM1 angeschlossen und bereit ist. Auf der CD wird ein eigenes Ladeprogramm eingesetzt, das in Kap. 10 auch im Quelltext vorgestellt wird.

Die meisten Programme können Sie sogar starten ohne die CC-Unit zu berühren. Hier wird nämlich eine kleine Erweiterung des CC-Systems verwendet, das es erlaubt, Programme durch ein Kommando vom PC zu starten. Programme können sozusagen ferngesteuert gestartet und abgebrochen werden. Dazu müssen Sie zunächst die Systemerweiterung laden.

Damit Sie nicht nur die Programme auf der CD verwenden können, sondern auch eigene entwickeln, werden hier die wichtigsten Werkzeuge zusammengestellt.

1.1 Das Ladeprogramm

Hier laden Sie ein kleines Zusatz-Betriebssystem, das es Ihnen erlaubt, die CC-Unit komplett vom PC aus zu steuern.

Der Ladevorgang dauert mit ca. 40 Sekunden recht lange. Verlieren Sie nicht die Geduld. Nun drücken Sie einmal die Start-Taste. Die rote und die gelbe LED leuchten nun. Trotzdem ist das System bereit zum Laden des eigentlichen Anwenderprogramms. Zum Laden eines Programms brauchen Sie die Tasten nicht mehr. Die Unit darf also nun in einer abgeschlossenen Glasvitrine stehen. Zum Ausprobieren der Programme auf der CD brauchen Sie nun nur noch einen Arbeitsschritt pro Test.

1. Sie klicken ein zu ladendes Programm an. Es wird nun automatisch geladen und gestartet. Der Ladevorgang ist daran erkennbar, dass nur die rote LED leuchtet. Wenn die gelbe LED leuchtet, ist das Programm gestartet.
2. Wenn Sie genug getestet haben, gehen Sie zurück zu Schritt 1 und laden ein neues Programm. Das laufende Programm wird dabei zuerst automatisch beendet.

Hier nun einige Programme, die Sie nacheinander aufrufen können.

```
'************************************
'
' C-Control/BASIC       Test1.BAS
'
' Aufgabe:
'
' - Schalten des digitalen Ausgangs 1
' - Blinker: ein Impuls pro Sekunde
' - Beenden nach 5 Impulsen
'
'************************************
' --- Definitionen --------------------

define Ausgang port[1]
define n Byte

' --- Programmoperationen -------------

For n= 1 to 5
  Ausgang = 1          'Einschalten
  pause 25             '0,5 s warten
  Ausgang = 0          'Ausschalten
  pause 25             '0,5 s warten
Next N                 'Schleife
end
```

Dieses Programm beendet sich nach fünf Sekunden selbst. Danach ist automatisch wieder das Betriebssystem aktiv, ohne dass Sie auf eine Taste drücken mussten.

Das zweite Programm wird durch ein beliebiges Byte über die RS232 unterbrochen. Im Programm selbst wird mit der Funktion RXD ein empfangenes Byte erkannt.

```
'************************************
'
' C-Control/BASIC       Test2.BAS
'
' Aufgabe:
'
' - Schalten des digitalen Ausgangs 1
' - Blinker: ein Impuls pro Sekunde
' - Beenden durch Byte über RS232
'
'************************************
' --- Definitionen --------------------

define Ausgang port[1]

' --- Programmoperationen -------------

#Loop
  Ausgang = 1          'Einschalten
  pause 25             '0,5 s warten
  Ausgang = 0          'Ausschalten
  pause 25             '0,5 s warten
  if rxd then end      'Abbruch RS232
goto Loop              'Schleife
end
```

Das Programm kann auch durch ein neues Download selbst unterbrochen werden. Allein mit einer Zeile "if rxd then end" erreichen Sie also, dass Ihr Programm beliebig vom PC aus gesteuert wird.

Nur an eines müssen Sie denken: Sobald Sie die Reset-Taste betätigen, ist auch die Systemerweiterung gestoppt. Sie können Sie aber erneut laden. Solange Sie die Unit mit Strom versorgen, bleibt die Systemerweiterung aktiv.

Nehmen wir an, Sie wollen die CC-Unit tatsächlich in einer Vitrine oder in einem Schaufenster arbeiten lassen. Nachts soll der Strom abgeschaltet werden. Am nächsten Tag wollen Sie wieder weiterarbeiten, ohne erst mühsam alles aufzuschließen. Quizfrage: Wie geht das?

1.2 Die CC-Unit am Draht

Also wie kann man vermeiden, am nächsten Tag wieder eine Taste der CC-Unit zu drücken, wenn nachts der Strom abgestellt wird?

Lösung: An der CC-Unit wird nach dem ersten Laden der Systemerweiterung der Autostart-Jumper AST gesetzt. Vor dem Schlafengehen müssen Sie nur als letztes Programm die Systemerweiterung erneut laden und dann alles ausschalten. Am nächsten Morgen erwacht auch die CC-Unit wieder mit ihrem neuen System. Sie können also weiterhin alles fernsteuern. Allerdings gibt es leider ein paar mögliche Programmfehler, die den ewigen Kreislauf dann doch beenden. Verwenden Sie also am besten nur gut getestete Programme für solche Anwendungen der CC-Unit am Draht. Auch auf dieser CD finden sich Programme, die keinen Nachfolger dulden und erst mit Reset verdrängt werden können. Ob Sie wohl herausfinden, welche und warum?

Sie können natürlich die Systemerweiterung selbst mehrmals durch sich selbst nachladen lassen. Aber nur fünf mal! Dann geht nichts mehr. Warum, das hat was mit Rekursion zu tun und wird weiter unten erläutert. Wenn Sie allerdings die Erweiterung laden und danach einen Neustart einleiten, dann geht es beliebig oft. Der Neustart kann durch Einschalten der Betriebsspannung mit gesetztem AST-Junper ausgeführt werden.

1.2 Werkzeuge

Wenn Sie selbst Programme entwickeln wollen, verwenden Sie wahrscheinlich schon immer die Windows-Programmierumgebung CCEW32D. Falls nicht, von der CD können Sie das Programm installieren.

Programme, die Sie selbst entwickeln und dann automatisch laden und starten wollen, müssen in der compilierten Form als DAT-Dateien vorliegen. Leider bietet das Windows-Programm CCEW32D keine Ausgabe des DAT-Formats. Man muss deshalb auf das DOS-Programm CCBAS ausweichen. Es wird z.B. mit "CCBAS TEST.BAS" aufgerufen und erzeugt die Datei TEST.DAT.

Unter Windows kann es zu Problemen mit DOS-Programmen kommen. Mit CCBAS.EXE wurde die Erfahrung gemacht, dass es zumindest auf manchen PCs unter Win98 nicht problemlos zu starten war. Dabei machte es keinen Unterschied, ob ein DOS-Fenster oder der DOS-Modus gewählt wurde. Die Lösung ist aber einfach:

Kopieren Sie CCBAS.EXE und Ihr BASIC-Programm auf eine Diskette und starten Sie es von dort. Nun funktioniert alles reibungslos. Der Unterschied dürfte in der Verwendung des Dateisystems FAT32 auf der Festplatte liegen.

Viele Anwendungen auf dieser CD verwenden Assemblerprogramme, um die Eigenschaften der CC-Unit zu erweitern. Der "offizielle" Assembler für die CC-Unit heißt AS5.EXE. Die Arbeit mit diesem Programm wurde in [2] vorgestellt. Von der CD können Sie das Programm kopieren.

Mir selbst liegt die Arbeit mit TASM besser, weil die Fehlermeldungen hilfreicher sind und weil man ein Listfile erhält. Wie man TASM verwendet, stand schon in [1]. Alle neuen Programme auf dieser CD wurden mit TASM übersetzt. Man erkennt dies auch an der Dateiendung „.OBJ“ statt des Motorola-typischen „.S19“.

2 Einfache Sensoren

Für den C-Control-Steuercomputer gibt es eine große Anzahl fertiger Sensoren. Vieles kann jedoch auch mit einfachen Mitteln selbst hergestellt werden.

Unter den einfachen Sensoren kann man drei Gruppen unterscheiden: Sensoren, die ihren Widerstand bzw. ihre Leitfähigkeit verändern, Sensoren, die eine Spannung abgeben und aktive Sensoren mit Transistoren. Sensoren, die eine hohe Verstärkung erfordern und nicht ohne Operationsverstärker auskommen, sollen hier nicht beschrieben werden. Es geht vielmehr darum, möglichst einfache Schaltungen direkt an die CC-Unit anzuschließen oder mit einem kleinen Stück Experimentierplatine auszukommen.

Hinweis: Diese Zusammenstellung einfacher Sensoren wurde einer Veröffentlichung in der Zeitschrift CompaS (Computer Anwendungen in der Schule) entnommen, die auch als online-Version zugänglich ist (www.modul.bus.de).

2.1 Widerstands-Sensoren

Einer der einfachsten Sensoren ist der LDR-Widerstand. Er verändert seinen Widerstand zwischen etwa 1 MΩ bei Dunkelheit und 1 kΩ bei hellem Licht. Zum Anschluß an den Spannungseingang eines Interfaces oder Meßgeräts verwendet man einen Spannungsteiler. Der Festwiderstand kann an den Meßbereich angepaßt werden. Wenn vorwiegend bei geringer Helligkeit gemessen werden soll, kann statt 10 kΩ ein Widerstand mit 100 kΩ oder mehr eingesetzt werden.

Genauso einfach ist der NTC-Widerstand einzusetzen. Ein NTC mit 10 kΩ bei 25 °C sollte mit einem Festwiderstand von ebenfalls 10 kΩ verwendet werden. Dann ergibt sich die beste Empfindlichkeit und eine angenähert gerade Übertragungskennlinie im Bereich der normalen Lufttemperatur. Wer einen NTC mit 4,7 kΩ verwendet, sollte entsprechend einen Festwiderstand mit 4,7 kΩ nehmen.

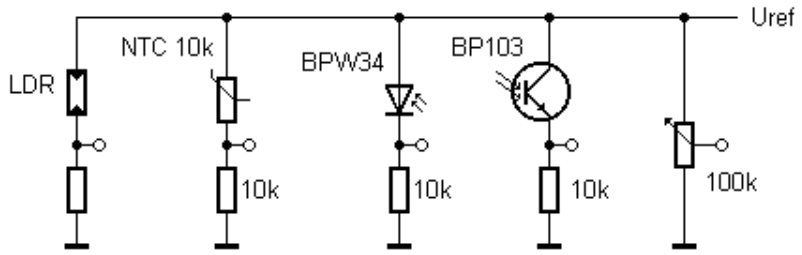

Abb. 2.1 Sensoren in Spannungsteilerschaltung ((Sensor1.BMP))

Natürlich kann auch ein einfaches Potentiometer bereits als Sensor eingesetzt werden. Es eignet sich zur Erfassung von Winkeln oder Strecken. Mit einer Seilscheibe läßt sich die Achse über ein Seil drehen. Interessante Anwendungen ergeben sich bei Schwingungsmessungen.

Die Fotodiode ist kein Widerstandssensor, sondern ein Stromsensor, wenn sie in Sperrichtung betriebene wird. Der Sperrstrom ist über weite Bereiche streng proportional zur Lichtintensität. Mit einem einfachen Arbeitswiderstand erhält man einen linearen Lichtsensor. Mit 10 kΩ ist der Sensor für große Helligkeiten ausgelegt. Die Fotodiode BPW34 hat eine große bestrahlungsempfindliche Fläche und damit eine gute Empfindlichkeit. Statt einer echten Fotodiode kann man auch mache Zenerdioden einsetzen, die seitlich bestrahlt werden müssen. LEDs arbeiten ebenfalls als Fotodioden. Allerdings sind die modernen, superhellen LEDs nicht mehr geeignet, wohl aber Infrarot-LEDs, die besonders empfindlich auf Licht reagieren. Sie werden übrigens in Gabel-Lichtschranken eingesetzt, wie man sie manchmal aus ausgedienten Diskettenlaufwerken ausbauen kann.

Statt der Fotodiode kann auch ein Fototransistor wie der BP103 verwendet werden. Man erreicht damit eine sehr viel höhere Empfindlichkeit. Die Übertragungskennlinie ist allerdings nicht mehr linear. Fototransistoren findet man ebenfalls in Gabel-Lichtschranken. Wenn die Anschlüsse nicht bekannt, sind, kann man sie mit der vorliegenden Schaltung leicht experimentell ermitteln.

Eine kleine Glühlampe ist ein brauchbarer Sensor für Luftströmungen, wenn man vorsichtig ihren Glaskolben entfernt. Der Glühdraht erhöht seinen Widerstand mit der Temperatur. Im Betrieb soll der Glühfaden durch einen Strom auf etwa 50-100 °C erhitzt werden. Jede kleine Luftströmung führt zu einer Abkühlung, die den Drahtwiderstand verringert und den Strom erhöht.

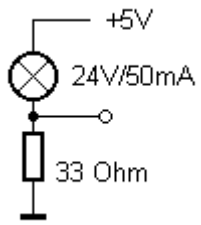

Abb. 2.2 Eine Glühlampe als Luftströmungssensor ((Sensor2.BMP))

2.2 Sensoren als Spannungsquellen

Sensoren, die eine Spannung abgeben, benötigen meist keine Hilfsspannung vom Interface. Alle Fotodioden können auch als Spannungsquellen eingesetzt werden. Sie arbeiten dann ähnlich wie eine Fotozelle. Die höchste Spannung ist etwa 0,5 V bei vollem Sonnenlicht. Durch Reihenschaltung kann man die Messspannung und die Empfindlichkeit der Messung erhöhen. Die Spannung ist nicht proportional zur Helligkeit, sondern bildet eine logarithmisches Maß der Helligkeit. Man kann daher über viele Zehnerpotenzen messen. Bei geringer Helligkeit wird die Spannungsquelle sehr hochohmig, so daß es zu Störungen durch den Messeingang kommt. Ein kleiner Kondensator verbessert die Messungen bei kleiner Helligkeit.

Jeder Gleichstrommotor eignet sich als Sensor für die Drehgeschwindigkeit. Man kann sehr einfach Sensoren für die Windgeschwindigkeit aufbauen. Besonders geeignet sind leicht laufende Motoren mit mehrpoligen Ankern, wie sie in Kassettenrecordern verwendet werden.

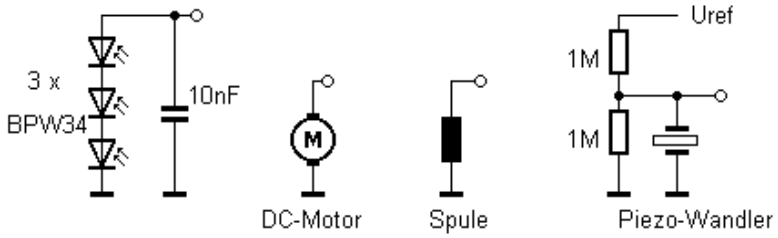

Abb. 2.3 Sensoren, die eine Spannung abgeben ((Sensor3.BMP))

Als Sensor für magnetische Wechselfelder eignet sich im Prinzip jede Spule. Man kann untersuchen, welche Störfelder an Transformatoren oder Monitoren auftreten. Für eine gute Empfindlichkeit benötigt man Spulen mit etwa 1000 Windungen und Eisenkern. Die Spule gibt eine Wechselspannung ab, von der an der CC-Unit jeweils nur die positive Halbwelle sichtbar wird.

Einfache Piezo-Schallgeber eignen sich hervorragend als Sensoren für Kräfte und Schwingungen. Die abgegebene Spannung erreicht Werte von einigen Volt, wenn man einen vorsichtigen Druck mit dem Finger ausübt. Die Gleichspannung am Sensor ist wie bei einem Kondensator unbestimmt. Für eine sinnvolle Messung muß daher mit einem hochohmigen Spannungsteiler eine Vorspannung erzeugt

2.3 Aktive Sensoren

Transistoren eignen sich auch zur Temperaturmessung. Abb. 2.4 zeigt eine Schaltung zur Messung von Differenztemperaturen. Mit dem Poti kann man zunächst die Ausgangsspannung auf den halben Messbereich einstellen. Jedes Ungleichgewicht in den Transistortemperaturen führt dann zu einer deutlichen Spannungsänderung.

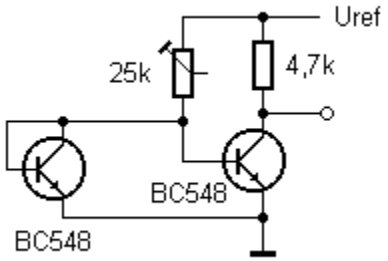

Abb. 2.4 Transistoren als Sensoren für Temperaturdifferenzen ((SENSOR4.BMP))

Ein einzelner Transistor kann als einfacher Sensorverstärker eingesetzt werden. Ein sinnvoller Einsatz ergibt sich z.B. bei der Verstärkung von Induktionsspannungen zur Magnetfeldmessung. Auch eine weniger geeignete Spule wird mit diesem Verstärker zu einem brauchbaren Sensor für magnetische Wechselfelder.

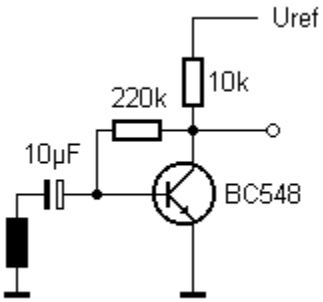

Abb. 2.5 Ein einfacher Sensorverstärker für Induktionsspannungen ((Sensor5.BMP))

Auch elektrische Felder und Ladungen lassen sich relativ einfach messen. Zwei Transistoren in Darlingtonschaltung erreichen eine Stromverstärkung von ca. 30000. Damit läßt sich ein empfindlicher Ladungsverstärker aufbauen. Ein einfacher Versuch zeigt die Aufladung durch Reibung. Man berührt den Eingang

mit dem Finger und bewegt seine Füße. Mit jedem Schritt zeigt sich dann ein Ausschlag. Die beobachtete Ladungstrennung ist abhängig vom Material der Schuhe und des Bodenbelags.

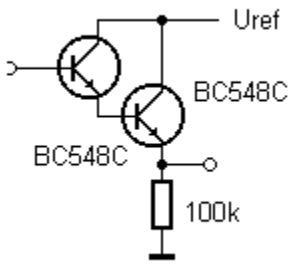

Abb. 2.6 Ein Sensorverstärker für elektrische Ladungen ((Sensor6.BMP))

3 Mehr Portanschlüsse

Der C-Control-Steuercomputer hat schon eine große Anzahl von Portanschlüssen. Und trotzdem kommt es vor, dass sie nicht ausreichen. Wenn Sie für eine Aufgabe alle 16 Digitalports verbraucht haben, und dann fehlt Ihnen gerade noch eine Leitung, was machen Sie dann? Klar, Sie schauen erst mal ins Schaltbild und entdecken doch noch ein paar Leitungen, die allerdings nicht so ohne weiteres nutzbar sind. Da wären ja noch die drei LED-Anschlüsse, die man anders nutzen könnte, und der Eingang des Starttasters, und zwei Hilfsleitungen der seriellen Schnittstelle, und und und... Aber wie kommt man da heran? Mit Assemblerprogrammen!

Assemblerprogramme wurden bisher immer internen EEPROM des Prozessors ab Adresse $101 untergebracht. Wenn es um sehr kleine Programme geht, erscheint der Aufwand relativ groß. Man muß einen Assembler-Quelltext schreiben, ihn übersetzen, des Maschinencode in das Basic-Programm einbinden und dann mit SYS aufrufen. Und wenn nur ein Bit falsch war, wechselt man wieder zum Editor, zum Assembler zum Basic ...

Aber wer hat denn eigentlich gesagt, dass Maschinenprogramme immer im internen EEPROM stehen müssen? Warum denn nicht im RAM, also im Variablenspeicher? Laut Datenblatt des Prozessors spricht nichts dagegen. Also: Ausprobieren!

3.1 Maschinencode im Variablenspeicher

Der Variablenspeicher der Control-Unit beginnt bei $A1 und hat eine Größe von nur 24 Bytes. Aber Platz ist in der kleinsten Hütte, und die interessantesten Maschinenprogramme können ganz klein sein. Oft braucht man nicht sehr viele Basic-Variablen. Da kann man leicht ein paar Bytes lockermachen.

Die Aufgabe für den ersten Test soll sein, die gelbe LED aus einem Programm heraus beliebig ein- und auszuschalten. Das Schaltbild der CC-Unit verrät: Sie liegt am Port A3. Man braucht also eigentlich nur die Assemblerbefehle bset und bclr. Ein kleines Assemblerprogramm ist schnell geschrieben. Im LST-Ausgabefile findet man den Maschinencode, den man braucht.

```
16 00       P13on   bset 3,0        ;LED gelb
81                  rts
17 00       P13off  bclr 3,0
81                  rts
```

Die insgesamt sechs Bytes müssen per Hand in den RAM-Speicher übertragen werden. Dazu muss man sich einen Adressbereich aussuchen. Mein Vorschlag: Die ersten beiden Bytes bleiben immer reserviert für den Fall, dass man einmal Daten zwischen Basic und Assemblerprogramm austauschen will. Ab Adresse $A3 beginnt der Programmcode. Insgesamt werden erst mal 12 Bytes dafür reserviert. Dazu deklariert man Variablen, deren Name gleich für Ihre Adressen steht.

```
'************************************
'
' C-Control/BASIC       RAMsys.BAS
'
' Aufgabe:
```

```
'
' - Assemblercode im Variablenspeicher
' - Schalten der Leitung PA3, LED gelb
'
'************************************
' --- Definitionen --------------------

define Data1 Byte  '&A1
define Data2 Byte  '$A2

define A3 Byte     'Platzhalter
define A4 Byte
define A5 Byte
define A6 Byte
define A7 Byte
define A8 Byte
define A9 Byte
define AA Byte
define AB Byte
define AC Byte
define AE Byte
define AF Byte

define n Byte
define Ausgang port[1]
' --- Programmoperationen -------------


A3 = &H16  'bset 3,0
A4 = &H00
A5 = &H81  'rts
A6 = &H17  'bclr 3,0
A7 = &H00
A8 = &H81  'rts

For n= 1 to 5
  Ausgang = 1          'Einschalten
  sys &HA3             'Gelb aus
  pause 25             '0,5 s warten
  Ausgang = 0          'Ausschalten
  sys &HA6             'Gelb an
  pause 25             '0,5 s warten
Next N                 'Schleife
end
```

Es funktioniert tatsächlich! Die gelbe LED blinkt. Das Programm verwendet SYS, aber nicht SYSCODE. Alles läuft im RAM. Und jetzt können Sie tatsächlich mal eben ein paar Bits ändern.

3.2 Port-A-Ausgänge

Bisher wurde nur die gelbe LED angesteuert. Es gibt aber noch mehr interessante Leistungen. Ein Blick ins Schaltbild verrät die Belegung des Ports A.

PA0: SDA zum EEPROM, nicht verändern!
PA1: SCL zum EEPROM, nicht verändern!
PA2: Ausgang rote LED: Run
PA3: Ausgang gelbe LED: Active
PA4: Ausgang grüne LED: DCF
PA5: Eingang für Start-Taste und Autostart-Jumper
PA6: Eingang RTS vom MAX232
PA7: Ausgang CTS zum MAX232

Es gibt also insgesamt vier Ausgänge, die man auch für eigene Zwecke einsetzen könnte. Jetzt sollen erst einmal alle drei LEDs angesteuert werden. Man könnte jetzt sechs kleine Programme einsetzen, die jeweils drei Bytes belegen. Mit 18 Bytes wäre das noch möglich aber schon bedenklich lang. Es geht jedoch auch kürzer: Da die Programme sich immer nur ein einem einzigen Byte unterscheiden, kann man hier sehr gut die Möglichkeit eines selbstmodifizierenden Programms nutzen. Man ändert also laufend das kleine Maschinenprogramm durch Neuzuweisung einer einzelnen Bytevariablen. Diese Chance bietet übrigens nur das RAM, im EEPROM dagegen wäre das unmöglich!

```
'**************************************
'
' C-Control/BASIC       RGG1.BAS
'
' Aufgabe:
'
' - Assemblercode im Variablenspeicher
' - Schalten der Leitungen PA4, PA3, PA2
```

```
'
'************************************
' --- Definitionen --------------------

define Data1 Byte  '&A1
define Data2 Byte  '$A2

define A3 Byte     'Platzhalter
define A4 Byte
define A5 Byte

define n Byte
' --- Programmoperationen -------------


A3 = &H16  'bset 3,0
A4 = &H00
A5 = &H81  'rts

For n= 1 to 5
  A3= &H19: sys &HA3   'bclr 4,0 Grün an
  pause 25             '0,5 s warten
  A3= &H18: sys &HA3   'bset 4,0 Grün aus
  A3= &H17: sys &HA3   'bclr 3,0 Gelb an
  pause 25             '0,5 s warten
  A3= &H16: sys &HA3   'bset 3,0 Gelb aus
  A3= &H15: sys &HA3   'bclr 2,0 Rot an
  pause 25             '0,5 s warten
  A3= &H14: sys &HA3   'bset 2,0 Rot aus
Next N                 'Schleife
end
```

Es funktioniert ... fast. Die grüne LED und die rote LED leuchten leider immer nur ganz kurz auf. Der Grund: Das Betriebssystem der Control-Unit kümmert sich auch noch um diese Leitungen. Bei der grünen LED ist es eine Interrupt-gesteuerte Timer-Routine, die sich um DCF-Signale kümmert. Im Prinzip kann man den Interrupt umleiten und die Arbeit der Routine abkürzen, damit sie sich nicht mehr um PA4 kümmern kann. Allerdings ist hierzu ein Maschinenprogramm im internen EEPROM nötig.

Die rote LED wird immer dann ausgeschaltet, wenn ein Pause-Befehl auftaucht. Nach dem Pause-Befehl wird sie vom System wieder eingeschaltet. Man braucht also nur alle Pause-Befehle zu vermeiden, um die rote LED ganz für sich zu

haben. Also verwendet man am besten statt dessen eine ganz einfache Warteschleife.

```
'************************************
'
' C-Control/BASIC       RGG2.BAS
'
' Aufgabe:
'
' - Assemblercode im Variablenspeicher
' - Schalten der Leitungen PA4, PA3, PA2
'
'************************************
' --- Definitionen --------------------

define Data1 Byte  '&A1
define Data2 Byte  '$A2

define A3 Byte     'Platzhalter
define A4 Byte
define A5 Byte

define n Byte
define t Word
' --- Programmoperationen -------------


A3 = &H16  'bset 3,0
A4 = &H00
A5 = &H81  'rts

For n= 1 to 5
  A3= &H19: sys &HA3   'bclr 4,0 Grün an
  gosub warten
  A3= &H18: sys &HA3   'bset 4,0 Grün aus
  A3= &H17: sys &HA3   'bclr 3,0 Gelb an
  gosub warten
  A3= &H16: sys &HA3   'bset 3,0 Gelb aus
  A3= &H15: sys &HA3   'bclr 2,0 Rot an
  gosub warten
  A3= &H14: sys &HA3   'bset 2,0 Rot aus
Next N                 'Schleife
end

#warten
  for t= 1 to 300
  next t
```

```
    return
```

Nun werden die gelbe und die rote LED in einem Lauflicht gesteuert, die grüne ist dagegen immer noch auch unter dem Einfluss des Betriebssystems und blinkt nur kurz auf.

3.3 Ports A und D lesen

Der Prozessor 68HC05 hat insgesamt vier vollständige Byteports. Der vierte ist Port D und wird normalerweise nur als Analogport verwendet. Digitale Ausgaben sind nicht möglich, wohl aber ganz normale digitale Eingaben. Wenn Sie also einmal mehr als 16 digitale Eingänge brauchen, verwenden Sie doch einfach die acht Analogeingänge der CC-Unit als zusätzliche Digitalports. Ein kleines Assemblerprogramm liest dazu den Inhalt des Registers PORTD.
Und weil es so einfach geht, wird gleich auch noch Port A gelesen. Dort gibt es immerhin noch zwei Leitungen, die man sinnvoll als Eingänge verwenden kann. Das folgende Programm liest beide Byteports und sendet die Ergebnisse an den PC.

```
'************************************
'
' C-Control/BASIC       PAD.BAS
'
' Aufgabe:
'
' - Assemblercode im Varablenspeicher
' - Lesen der Ports A und D
'
'************************************
' --- Definitionen --------------------

define Data1 Byte  '&A1
define Data2 Byte  '$A2

define A3 Byte     'Platzhalter
define A4 Byte
define A5 Byte
define A6 Byte
define A7 Byte
```

```
define A8 Byte
define A9 Byte
define AA Byte
define AB Byte

define n Byte
define t Word
' --- Programmoperationen -------------


A3 = &HB6  'lda PortA
A4 = &H00
A5 = &HB7  'sta A1
A6 = &HA1
A7 = &HB6  'lda PortD
A8 = &H03
A9 = &HB7  'sta A2
AA = &HA2
AB = &H81  'rts

#Loop
  sys &HA3               'Ports lesen
  print "Port A", Data1
  print "Port D", Data2
  pause 50
  if rxd then end
Goto Loop

end
```

Die Ergebnisse kann man sich mit einem kleinen Terminalprogramm ansehen, das weiter unten noch genauer vorgestellt wird. Legen Sie digitale Pegel von 0V/5V an die Analogeingänge. Der Zustand des Gesamtports D wird am Bildschirm angezeigt.

Port A enthält u.a. die Leitung PA5 mit der Starttaste und PA6 mit der Eingangsleitung CTS vom Schnittstellentreiber. Sie können Pegelwechsel an beiden Leitungen nun direkt beobachten.

Die gelbe Starttaste der CC-Unit ist eigentlich viel zu schade, um sie nur zum Starten eines Programms zu verwenden. Das folgende Programm fragt die Taste ab und bildet einen Softwarezähler. Der aktuelle Zählerstand wird bei jedem Tastendruck an den PC übertragen.

```
'************************************
'
' C-Control/BASIC       PAD2.BAS
'
' Aufgabe:
'
' - Assemblercode im Variablenspeicher
' - Auswerten von Port A.5
' - Softewarezähler mit Starttaste
'
'************************************
' --- Definitionen --------------------

define Data1 Byte  '&A1
define Data2 Byte  '$A2

define A3 Byte     'Platzhalter
define A4 Byte
define A5 Byte
define A6 Byte
define A7 Byte
define A8 Byte
define A9 Byte
define AA Byte
define AB Byte

define Count Byte
' --- Programmoperationen -------------
A3 = &HB6  'lda PortA
A4 = &H00
A5 = &HB7  'sta A1
A6 = &HA1
A7 = &HB6  'lda PortD
A8 = &H03
A9 = &HB7  'sta A2
AA = &HA2
AB = &H81  'rts

#Loop
#Lo  sys &HA3        'Ports lesen
     if (Data1 AND 32) = 0 then goto Lo
#Hi  sys &HA3        'Ports lesen
     if (Data1 AND 32) = 32 then goto Hi
     Count = Count + 1
     print Count
     if rxd then end
goto Loop
```

```
end
```

3.4 Taktsignale

Manchmal benötigt man in einer Schaltung ein Taktsignal in Rechteckform. Für Frequenzen im NF-Bereich können Sie den BEEP-Befehl nehmen. Ein Taktsignal mit 1 kHz erhält man 10 Minuten lang mit

BEEP 250, 3000,0

Höhere Taktfrequenzen kann man mit Assemblerprogrammen erzeugen. Die erreichbare Grenze liegt knapp über 100 kHz. Darf es noch etwas mehr sein? Das Datenblatt zum Prozessor verrät eine ganz besondere Eigenschaft des Ports PC2, also der Leitung Port 11 der CC-Unit. Man kann mit einem Maschinenbefehl den internen Prozessortakt von 2 MHz an diesen Ausgang legen. Dazu muss man nur im Register EEPROM an der Adresse $07 das ECLK-Bit (Bit 3) setzen.

```
'************************************
'
' C-Control/BASIC       2MHz.BAS
'
' Aufgabe:
'
' - Assemblercode im Variablenspeicher
' - Ausgabe von 2 MHz an Port 11
'
'************************************
' --- Definitionen --------------------

define Data1 Byte  '&A1
define Data2 Byte  '$A2

define A3 Byte     'Platzhalter
define A4 Byte
define A5 Byte

define n Byte
```

```
' --- Programmoperationen -------------


A3 = &H16  'bset 3,7 ; ECLK
A4 = &H07
A5 = &H81  'rts

#Loop
  A3= &H16: sys &HA3   'Takt an
  A3= &H17: sys &HA3   'Takt aus
  if rxd then end
goto Loop
end
```

Mit einem Oszilloskop können Sie nun das 2-MHz-Taktsignal am Port 11 sehen. Es erscheint mit einer Modulationsfrequenz von ca. 200 Hz, weil die Ausgabe in schneller Folge ein- und ausgeschaltet wird.

Hier überschreiten wir die Grenze zwischen Digitalelektronik und Hochfrequenztechnik. Schließen Sie einmal ein etwa 20 cm langes Stück Draht als Antenne an P11 an. Nun kann das Signal mit einem Radio empfangen werden. Die Frequenz 2 MHz liegt zwischen dem Mittelwellen- und dem Kurzwellenbereich und kann mit den meisten Radios nicht direkt abgestimmt werden. Ein symmetrisches Rechtecksignal enthält jedoch immer starke ungerade Oberwellen. Sie können die Oberwellen auf Kurzwelle bei 6 MHz, 10 MHz, 14 MHz, 18 MHz usw. empfangen. Wenn Ihr Radio nur einen Mittelwellenbereich hat, versuchen Sie es mal bei ca. 1140 kHz. Bei 1140 kHz + 2 * 460 kHz = 2000 kHz liegt die Spiegelfrequenz eines Superhets mit einer Zwischenfrequenz von 460 kHz und damit eine schwache Nebenempfangsstelle. Mit einem kurzen Stück Draht werden noch nicht gleich die EMV-Bestimmungen verletzt. Mit mehreren Metern Länge wird es dagegen kritisch. Man kann Oberwellen dann noch in größeren Entfernungen empfangen. Bedenken Sie, dass Sie nicht wissen können, welche Funkdienste Sie damit gerade stören. Also wenn Sie auch noch einen 100-W-Hochfrequenzverstärker bauen und eine wohlabgestimmte Dipolantenne anschließen, um halb Europa mit Ihrem Signal zu beglücken, dann kann ich auch nichts dafür, dass eines Tages ein Funkmesswagen vor Ihrer Tür steht, Sie in Handschellen abgeführt werden und man Ihnen die halbe Werkstatt ausräumt.

4 Mehr Analogkanäle mit Multiplexern

Die CC-Unit hat immerhin 8 analoge Eingangskanäle, mit denen man eine Menge anfangen kann. Aber manchmal benötigt man doch noch mehr Messstellen. Da hilft nur noch ein externer Analog-Multiplexer, dessen Funktion man mit der eines Umschalters beschreiben kann. Statt Schaltkontakten enthält das IC Feldeffekttransistoren, die über externe Signale geschaltet werden.

4.1 Analogschalter 4066

Ein typische Analogschalter in CMOS-Technik ist der 4016 oder der 4066. Vier interne Analogschalter können als unabhängige Schließer-Kontakte eingesetzt werden. Die logischen Eingänge können direkt von Ports der CC-Unit geschaltet werden.

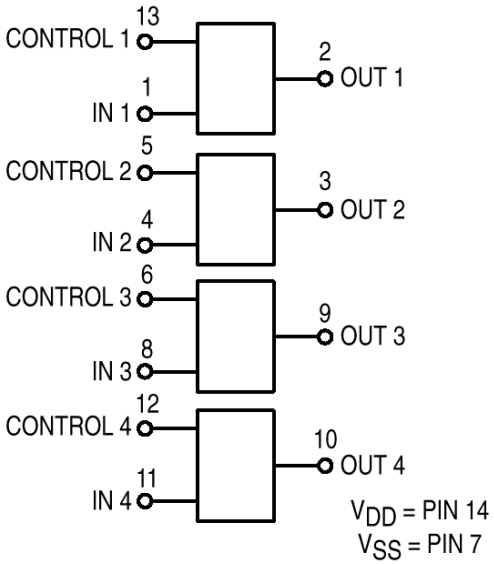

Abb. 4.1 Anschlüsse des vierfachen Analogschalters 4016/4066 ((4066B.GIF, besser: Datenblatt-Ausdruck))

Zum Anschluss an die CC-Unit verbindet man die Steuereingänge (Control 1...4) mit digitalen Ausgängen. Ein Programm schaltet jeweils einen der Eingänge ein, um den entsprechenden Schalter zu schließen. Aus einem Analogeingang werden auf diese Weise vier.

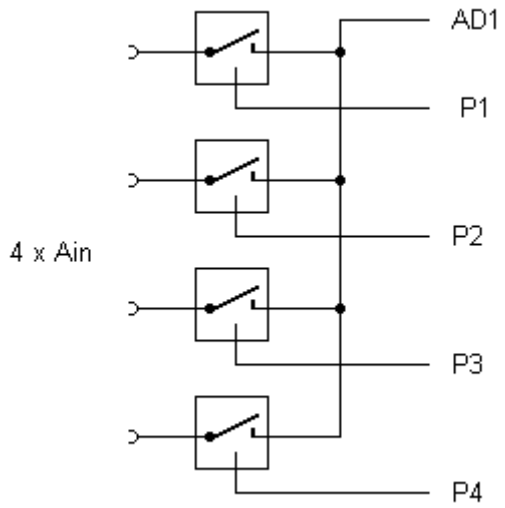

Abb. 4.2 Anschluss an die CC-Unit ((4066C.BMP))

```
'************************************
'
' C-Control/BASIC       Multi4.BAS
'
' Aufgabe:
'
' - Multiplexer mit  4066
' - Messen der Kanäle 0...3
'
'************************************
```

```
' --- Definitionen --------------------

define Analog1 AD[1]
define PA Port[1]
define PB Port[2]
define PC Port[3]
define PD Port[4]
define Kanal Byte
define Messwert Byte

' --- Programmoperationen -------------

#Loop
  PA=0: PB=0: PC=0: PD=0
  for Kanal = 0 to 3
    gosub Messung
    print Kanal, Messwert    'messen und senden
    pause 25         '0,5 s warten
  next Kanal
goto Loop          'Endlosschleife
end

#Messung
  If Kanal = 0 then PA=1
  If Kanal = 1 then PB=1
  If Kanal = 2 then PC=1
  If Kanal = 3 then PD=1
  pause 1
  Messwert = Analog1    'messen
  PA=0: PB=0: PC=0: PD=0
return
```

4.2 Analogschalter 4051

Der 4051 enthält acht Analogschalter in CMOS-Technik, die wie ein Umschalter angeordnet sind. Außerdem ist bereits eine Dekodierlogik enthalten, so dass man alle acht Schalter mit nur drei digitalen Leitungen A, B und C ansteuern kann. Ein zusätzlicher Inhibit-Eingang schaltet alle Schalter aus, wenn hier ein High-Pegel angelegt wird.

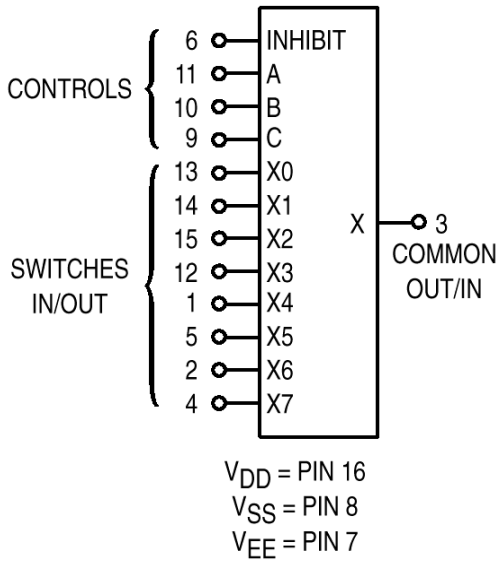

Abb. 4.3 Anschlussbelegung des 4051 ((4051B.BMP))

Der Anschluss an die CC-Unit ist einfach. Die Digitalausgänge P1 bis P3 steuern den Umschalter. Der Inhibit-Steuereingang des 4051 wird permanent an Masse gelegt, um den Baustein zu aktivieren.

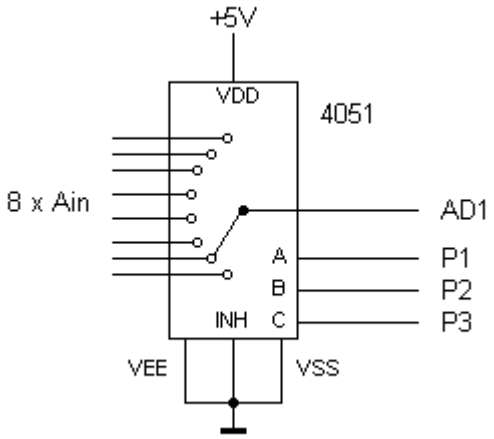

Abb. 4.4 Anschluss an die CC-Unit ((4051C.BMP))

```
'************************************
'
' C-Control/BASIC       Multi8.BAS
'
' Aufgabe:
'
' - Multiplexer mit  4051
' - Messen der Kanäle 0...7
'
'************************************
' --- Definitionen --------------------

define Analog1 AD[1]
define PA Port[1]
define PB Port[2]
define PC Port[3]
define Kanal Byte
define Adresse Byte
define Messwert Byte

' --- Programmoperationen -------------

#Loop
  PA=0: PB=0:PC=0
  for Kanal = 0 to 7
    gosub Messung
```

```
    print Kanal, Messwert    'messen und senden
    pause 25         '0,5 s warten
  next Kanal
goto Loop          'Endlosschleife
end

#Messung
  Adresse = Kanal mod 8
  PA = (Adresse and 1)
  PB = (Adresse and 2)/2
  PC = (Adresse and 4)/4
  pause 1
  Messwert = Analog1    'messen
return
```

4.3 Bis zu 64 Eingänge

Mit acht Analog-Umschaltern 4051 können Sie aus einem einzigen Analogeingang 64 Eingangskanäle machen. Die Steuereingänge A, B und C liegen parallel und werden gemeinsam von den Ports 1...3 gesteuert. Alle Analogausgänge der Schalter führen an den gleichen Eingang AD1 der CC-Unit. Zusätzlich wird aber hier jeder Inhibit-Eingang mit einem Port verbunden. Mit einem 1-Pegel wird das entsprechende IC abgeschaltet. Es darf nur immer ein IC zu einer Zeit freigegeben werden. Insgesamt kann die CC-Unit damit 64 Eingänge auswählen. Das Schaltbild zeigt zwei der insgesamt acht ICs.

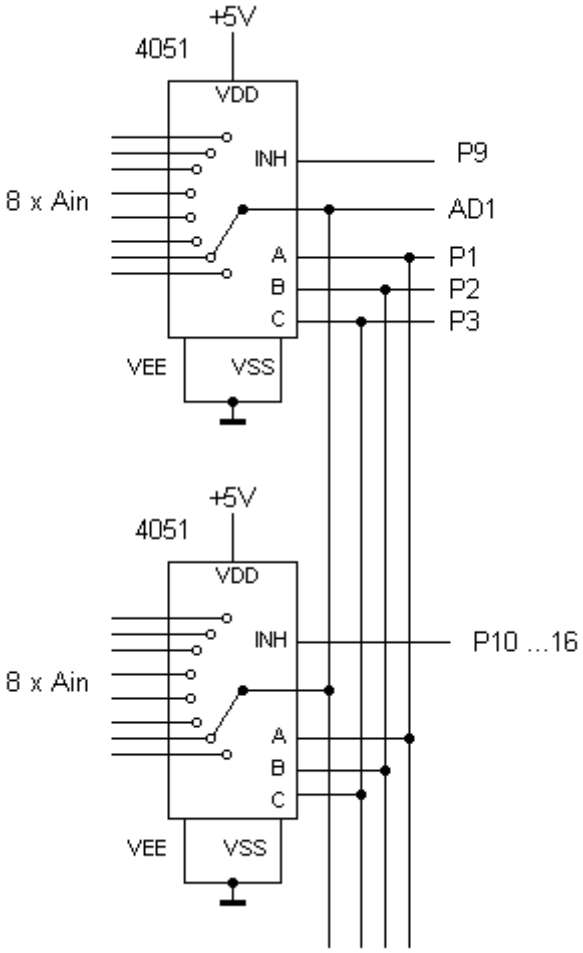

Abb. 4.5 Bis zu 64 Eingänge ((4051D.BMP))

Sie können je nach Bedarf an Messkanälen auch nur zwei oder drei ICs einsetzen. Aber natürlich können Sie auch acht mal acht ICs an die acht Analogeingänge legen. Sie hätten dann 512 Kanäle.

'**********

```
'
' C-Control/BASIC       Multi64.BAS
'
' Aufgabe:
'
' - Multiplexer mit 8 x 4051
' - Messen der Kanäle 0..63
' -
'
'************************************
' --- Definitionen --------------------

define Analog1 AD[1]
define PA Port[1]
define PB Port[2]
define PC Port[3]
define Inhibit Byteport[2]
define Kanal Byte
define Adresse Byte
define IcNr Byte
define Messwert Byte

' --- Programmoperationen -------------

#Loop
  Inhibit = &B11111111
  PA=0: PB=0:PC=0
  for Kanal = 0 to 63
    gosub Messung
    print Kanal, Messwert    'messen und senden
    pause 25         '0,5 s warten
  next Kanal
goto Loop          'Endlosschleife
end

#Messung
  IcNr = Kanal / 8
  Adresse = Kanal mod 8
  If IcNr = 0 then Inhibit = &B11111110
  If IcNr = 1 then Inhibit = &B11111101
  If IcNr = 2 then Inhibit = &B11111011
  If IcNr = 3 then Inhibit = &B11110111
  If IcNr = 4 then Inhibit = &B11101111
  If IcNr = 5 then Inhibit = &B11011111
  If IcNr = 6 then Inhibit = &B10111111
  If IcNr = 7 then Inhibit = &B01111111
  PA = (Adresse and 1)
  PB = (Adresse and 2)/2
```

```
  PC = (Adresse and 4)/4
  pause 1
  Messwert = Analog1    'messen
return
```

5 Servo-Ansteuerung

Einfache Bewegungsmodelle können mit C-Control gesteuert werden. Denkbar ist der Antrieb über Schrittmotoren oder Gleichstrommotoren. Immer dann, wenn bestimmte Positionen angefahren werden müssen, kommen auch Servos in Betracht.

Fernsteuer-Servos werden über Serien kurzer Impulse zwischen 1 ms und 2 ms gesteuert. Die Wiederholfrequenz liegt üblicherweise bei etwa 50 Hz, ist aber nicht kritisch. Mit dem Ausbleiben weiterer Impulse behält das Servo seine aktuelle Position. Auf den ersten Blick ist die Control-Unit nicht zur direkten Ansteuerung geeignet, weil das Basic einfach zu langsam ist. Es muß also ein Maschinenprogramm im internen EEPROM des Prozessors eingesetzt werden.

5.1 SERVO.ASM

Das Programm SERVO.ASM erzeugt die erforderlichen Impulse an bis zu acht Portleitungen. Es können also bis zu acht Servos gleichzeitig gesteuert werden. Vom Basic aus werden zwei Bytes für den Portanschluss und die aktuelle Sollposition des Servos übergeben. Die Impulslänge ändert sich zwischen 1 ms (Position 1) und 2 ms (Position 255).

Der Anwender kann angeben, an welchem Portanschluß der Impuls ausgegeben werden soll. Dazu wird die erste Byte-Variable des BASIC verwendet. Sie liegt an der Adresse $0A1. Hier kann man Wert zwischen 1, 2, 4 usw. bis 128 zuweisen, um die Ports 1, 2, 3 bis 8 zu verwenden. Das Maschinenprogramm schaltet den Port B (Ausgang 1...8) entsprechend dem übergebenen Bitmuster ein und nach der gewünschten Verzögerung (1...2 ms) wieder aus. Es ist also auch möglich, mehrere Servos im Gleichschritt anzusteuern. Das Bitmuster 00001111=15 steuert vier Servos an P1...P4 zugleich an.

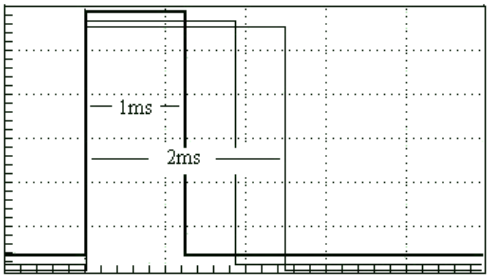

Abb. 5.1 Servo-Impulse ((Servoimp.BMP))

Das Programm Servo.ASM enthält zwei Zeitschleifen. Die erste Schleife erzeugt eine feste Verzögerung von 1 ms. Die zweite liefert eine variable Verzögerung bis zu 1 ms. Der Übergabewert wird in einer BASIC-Variablen an der Adresse $0A2 übergeben. Im Listing sind die Taktzyklen angegeben. Ein Befehlstakt dauert exakt 0,5 µs. Zusätzliche NOP-Befehle können für einen Feinabgleich verwendet werden.

```
;SERVO.ASM
portb   .equ 1       ;Daten Port B
pbdir   .equ 5       ;Datenrichtung Port B

       .org $101
        sei
loop    lda $0A1     ;Bytevariable 1
        sta portb    ;Port setzen
        ldx #230
loop1   nop          ;Loop1: 1 ms
        dex
        bne loop1
        ldx $0A2     ;Impulslänge in X laden

        inx
loop2   nop
        dex          ;X -1
        bne loop2    ;Loop2: 0...1ms

        lda #00
        sta portb    ;Port = 0
        cli
```

```
        rts
.end
```

Das folgende BASIC-Programm steuert ein Servo am Port 1. Es werden alle Positionen langsam durchfahren. Für jede Stellung zwischen 1 und 255 werden drei Impulse erzeugt.

```
'************************************
'
' C-Control/BASIC       SERVO1.BAS
'
' Aufgabe:
'
' - Servo an Port 1 (ws), +5V (rt), GND (sw)
' - Ganz langsam: 1 Schritt / 60 ms
'
'************************************
' --- Definitionen --------------------

define Bitmuster byte 'Bytevar $0A1
define Position byte    'Bytevar $0A2
Define AusPort BytePort[1]

define N byte

' --- Programmoperationen -------------

AusPort = 0
Bitmuster = 1   'Servo an P1
#loop
 for Position = 0 to 255
    sys &H0101   '1 Impuls
    pause 1      '20 ms Pause
    sys &H0101   '2. Impuls
    pause 1      '20 ms
    sys &H0101   '3. Impuls
    pause 1      '20 ms
 next
goto loop
end

syscode "Servo.OBJ"   'Code laden

Servo1.bas ausführen
```

Das Programm Servo2.bas steuert ein Servo an P1 entsprechend der Spannung am Analogeingang 1. Man kann auf diese Weise ein übergroßes Zeiger-Voltmeter realisieren.

```
'************************************
'
' C-Control/BASIC       SERVO2.BAS
'
' Aufgabe:
'
' - Servo an Port 1 (ws), +5V (rt), GND (sw)
' - Steuerung nach AD1-Spannung
'
'************************************
' --- Definitionen --------------------

define Bitmuster byte 'Bytevar $0A1
define Position byte    'Bytevar $0A2
Define AusPort BytePort[1]
Define Analog AD[1]

' --- Programmoperationen -------------

AusPort = 0
Bitmuster = 1   'Servo an P1
#loop
    Position = Analog
    sys &H0101   '1 Impuls
    pause 1      '20 ms Pause
goto loop
end

syscode "Servo.OBJ"   'Code laden
```

Das dritte Beispiel demonstriert die gleichzeitige Ansteuerung von drei Servos. Sie führen parallele Bewegungen mit einer gewissen Phasenverschiebung aus. Die drei Impulse werden ohne Verzögerung hintereinander erzeugt. Dann erst folgt eine Verzögerung von 20 ms bis zur nächsten Impulsserie.

```
'************************************
'
' C-Control/BASIC       SERVO3.BAS
'
' Aufgabe:
'
```

```
' - Servo an Port 1 (ws), +5V (rt), GND (sw)
' - 2. und 3. Servo an P2 und P3
'
'************************************
' --- Definitionen --------------------

define Bitmuster byte 'Bytevar $0A1
define Position byte    'Bytevar $0A2
Define AusPort BytePort[1]

define N byte

' --- Programmoperationen -------------

AusPort = 0
Bitmuster = 1   'Servo an P1
#loop
 for n = 0 to 255

    Bitmuster = 1 ' Servo 1
    Position = n
    sys &H0101   'Impuls

    Bitmuster = 2 'Servo 2
    Position = n+80
    sys &H0101   'Impuls

    Bitmuster = 4 ' Servo 3
    Position = n+160
    sys &H0101   'Impuls

    pause 1      '20 ms
 next
goto loop
end

syscode "Servo.OBJ"   'Code laden
```

5.2 Erweiterter Stellbereich: Servo2.ASM

Der Stellbereich von Servos ist nicht immer gleich. Viele Typen erlauben einen größeren Stellwinkel, wenn man den Impulsbereich 1...2 ms nach unten und nach oben erweitert. Mit dem Programm Servo2.ASM läßt sich zugleich eine höhere

Winkelauflösung erreichen. Hier werden zwei Bytes als Highbyte und Lowbyte übergeben. Die Zeitauflösung beträgt 3 µs, so daß eine Millisekunde in 333 Schritte aufgelöst wird. Man kann sich für jedes Servo an die tatsächlichen Aussteuerungsgrenzen heran tasten. Das Programm Servo2.ASM kann aber auch für anderen Zwecke eingesetzt werden, wenn Impulslängen bis ca. 100 ms mit hoher Präzision erzeugt werden müssen.

```
       ;SERVO2.ASM
portb  .equ 1       ;Daten Port B
pbdir  .equ 5       ;Datenrichtung Port B

       .org $101
        sei
loop    lda $0A1     ;Bytevariable 1
        sta portb    ;Port setzen
        ldx #230     ;3

        ldx $0A3     ; 3 Impulslänge-High in X laden
        beq Lowbyte  ; 3
loop1   lda #253     ; 3
        lda #253     ; 3
loopA   deca         ; 3
        bne loopA    ; 3  , 253*6 +4*3
        dex          ; 3
        bne loop1    ; 3

Lowbyte ldx $0A4     ; 3 Impulslänge-Low in X laden
        beq ende     ; 3
loop2   dex          ; 3  X -1
        bne loop2    ; 3  6*0,5µs = 3µs/Schleife

ende    lda #00
        sta portb    ;Port = 0
        cli
        rts
.end
```

Das Programm ServoB.bas demonstriert die Ansteuerung eines Servos mit hoher Auflösung. Die Übergabevariable wird hier als Word definiert, womit automatisch zwei Bytes übergeben werden.

```
    '************************************
```

```
'
' C-Control/BASIC       SERVO1.BAS
'
' Aufgabe:
'
' - Servo an Port 1 (ws), +5V (rt), GND (sw)
' - Ganz langsam: 1 Schritt / 60 ms
' _ neuer Treiber: SERVO2.OBJ mit Auflösung 3µs
'************************************
' --- Definitionen --------------------

define Bitmuster byte 'Bytevar $0A1
define Position Word  'Bytevar $0A3/A4
Define AusPort BytePort[1]

define N Word

' --- Programmoperationen -------------

AusPort = 0
Bitmuster = 1   'Servo an P1
#loop
 for N = 333 to 666  'entspricht 1 ms - 2 ms
    Position = n
    sys &H0101   '1 Impuls
    pause 1      '20 ms Pause
    sys &H0101   '2. Impuls
    pause 1      '20 ms
    sys &H0101   '3. Impuls
    pause 1      '20 ms
 next
goto loop
end

syscode "Servo2.OBJ"   'Code laden
```

6 RC5-Infrarotempfänger

Fänden Sie das nicht auch schön, vom Bett aus Lichtschalter, Kaffeemaschine, Radio und einen Ventilator mit einer Fernsteuerung zu bedienen? Viele moderne Infrarot-Fernbedienungen für TV-Geräte oder Videorecorder arbeiten nach dem

RC5-Standard, den Ihre CC-Unit jetzt auch versteht. Die könne vielleicht sogar die selbe Fernbedienung verwenden wie für das Fernsehgerät verwenden.

Man braucht eigentlich nur ein zusätzliches IC: den Infrarotempfänger SFH506. Das IC enthält bereits einen selektiven Vorverstärker und einen Demodulator für die verwendeten 33-kHz-Signale. Sie bekommen saubere 5-V-Pegel, die sofort von einem Portanschluß der CC-Unit gelesen werden können.

6.1 Das Programm RC5.ASM

Der eigentliche Dekoder für RC5-Signale kann nicht in Basic geschrieben werden, denn die Aufgabe ist zeitkritisch. Man braucht ein kleines Maschinenprogramm. Vorlage für das Programm RC5.ASM war ein Assemblerprogramm für den 8051-Controller von Olaf Kaluza (Gelsenkirchen). Es wurde praktisch Eins zu Eins in 6805-Assembler übertragen. Als Empfänger dient ein SFH506. Das empfangene Digitalsignal wird an Port C0, also am Port 9 der CC-Unit eingelesen.

```
; RC5-Unterprogramm   RC5.ASm
; nach einer Vorlage von Olaf Kaluza

PC  .equ $02
Cnt .equ $0A1
Kon .equ $0A2
Adr .equ $0A3
Tas .equ $0A4

        .org $101
        Clr Kon
        Clr Adr
        Clr Tas
rc5     brset 0,PC,rc5  ;Warten auf Startbit
        jsr w444        ;warten
        brset 0,PC,rc5  ;Stoerimpuls
        jsr w444        ;Naechsten Biphasenpegel
        jsr w444
        brclr 0,PC,rc5  ;Stoerimpuls
        jsr w444
        jsr w444
        brset 0,PC,rc5  ;Stoerimpuls
```

```
        jsr w444        ;Ok weiter
        jsr w444        ;Kontrollbit einlesen
        jsr getbit      ;Kontrollbit hier im Carry
        rol Kon
        ldx #5          ;5 Adressbits
rcadr   jsr w444
        jsr w444
        jsr getbit
        rol Adr         ;Bit merken
        dex
        bne rcadr
        ldx #6          ;6 Datenbits
rcdat   jsr w444
        jsr w444
        jsr getbit
        rol Tas         ;merken
        dex
        bne rcdat
        rts                     ;Zurueck zum Hauptprogramm
;-------------------------------------------------
;Einlesen eines Bits, Rueckgabe in Carry
getbit  lda  #255 ;Timeout festlegen
        sta  Cnt
        brset 0,PC,gethi  ;Wird es ein Lowbit?
gbl1    brset 0,PC,gbl2   ;Auf Biphasensprung synchronisieren
        dec Cnt
        bne gbl1
        bra biterr        ;Timeout
gbl2    jsr w444
        brclr 0,PC,biterr  ;Fehler
        clc                ;Lowbit
        rts
gethi   brclr 0,PC,gethi2  ;auf Phasensprung synchronisieren
        dec Cnt
        bne gethi
        bra biterr         ;Timeout
gethi2  jsr w444
        brset 0,PC,biterr  ;Fehler?
        sec                ;Highbit
        rts
biterr  Clr Kon            ;Fehler, alles löschen
        clr Adr
        clr Tas
        clr Cnt
        rts

;-------------------------------------------------
; Warteroutine fuer 1/4 eine Biphasenbits = 444us = 444T bei
```

```
12Mhz
;                        ;Aufrufcall  2T
w444    lda #147     ; 3
loop    deca         ; 3  A -1
        bne loop     ; 3  6*0,5µs = 3µ/Schleife
        rts

         .END
```

Das Programm übergibt vier Bytevariable an das aufrufende Basic-Programm:

Cnt .equ $0A1, ein Timeout-Zähler, gibt Auskunft über Lesefehler
Kon .equ $0A2, ein Kontrollbit 0/1, wechselt mit jedem neuen Tastendruck
Adr .equ $0A3, die Geräteadresse, z.B. 0 für TV
Tas .equ $0A4 , die gedrückte Taste, z.B. 0...9

Ein erstes Testprogramm in CC-Basic sendet einfach alle empfangenen Informationen an ein Terminal. Die Maschinenroutine wird mit SYS $101 aufgerufen und wartet auf ein Signal. Die vollständig empfangene Nachricht kann dann vom Basic-Programm gelesen und übertragen werden. Sie können an einem Terminal genau beobachten, welche Nachrichten Ihre Fernbedienung generiert.

```
'************************************
'
' C-Control/BASIC       RC5_1.BAS
'
' Aufgabe:
'
' - RC5-Signale an Port 9 empfangen
' - aktiv low
'
'************************************
' --- Definitionen --------------------

define Ctr byte 'Bytevar $0A1
define Kon byte 'Bytevar $0A2
define Adr byte 'Bytevar $0A3
define Tas byte 'Bytevar $0A4
Define AusPort BytePort[1]

define N byte

' --- Programmoperationen -------------
```

```
AusPort = 0
#loop
  sys &H0101
  Print Ctr, Kon, Adr, Tas
  if rxd then end
goto loop
end

   syscode "RC5.OBJ"   'Code laden
```

```
183     1       5       2
180     1       5       2
181     1       5       2
204     0       5       3
192     0       5       3
185     1       5       6
181     1       5       6
201     1       5       6
200     0       5       5
203     0       5       5
193     1       5       4
192     1       5       4
```

Abb. 6.1 Empfangene Daten mit der Geräteadresse 5 ((RC5a.BMP))

Die erste Spalte repräsentiert den Inhalt des Zählregisters Cnt beim Verlassen des Assemblerprogramms. Man kann hier ablesen, ob es Leseprobleme gegeben hat. Bei einem erkannten Lesefehler wird Cnt auf Null gesetzt. Die zweite Spalte zeigt, wie lange eine Taste gedrückt wurde. Ein Wechsel zwischen 1 und 0 bedeutet, dass eine Taste noch mal gedrückt wurde, oder dass eine neue Taste gedrückt wurde. Die dritte Spalte zeigt die Geräteadresse im Register Adr. In diesem Fall handelte es sich um die Geräteadresse 5 (TV-Geräte verwenden Adresse 0). Die letze Spalte schließlich zeigt die gedrückte Taste. Im Beispiel wurden nur Zifferntasten betätigt. Auch jede andere Taste ist einem Zahlenwert zugeordnet.

6.2 RC5-Relaissteuerung

Mit der Assemblerroutine RC5.ASM ist es nicht mehr schwierig, mehrere Relais fernzusteuern. Hier wird einfach das Starterboard mit seinen zwei Relais verwendet. Die Bedienung erfolgt über die Tasten 1, 2, 4 und 5, wobei die AN-Tasten auf der Fernbedienung über den zugehörigen AUS-Tasten liegen. Das Programm RC5_2.BAS zeigt das Prinzip. Es wird überwacht, ob Empfangsfehler vorlagen und ob die Geräteadresse 0 (TV-Gerät) ist.

```
'************************************
'
' C-Control/BASIC       RC5_2.BAS
'
' Aufgabe:
'
' - RC5-Relaissteuerung, Adresse 0
' - 2 Relais, 1,2 AN, 4,5 AUS
'
'************************************
' --- Definitionen --------------------

define Ctr byte 'Bytevar $0A1
define Kon byte 'Bytevar $0A2
define Adr byte 'Bytevar $0A3
define Tas byte 'Bytevar $0A4
Define Relais1  Port[1]
Define Relais2  Port[2]

' --- Programmoperationen -------------

#loop
  sys &H0101
  if Ctr < 100 then goto loop     'Lesefehler
  if Adr <> 0 then goto loop      'Nur Adr 5
  if Tas = 1 then Relais1 = 1
  if Tas = 2 then Relais2 = 1
  if Tas = 4 then Relais1 = 0
  if Tas = 5 then Relais2 = 0
  if rxd then end
goto loop
end

   syscode "RC5.OBJ"   'Code laden
```

7 Temperatursensor DS1820

Der Dallas-Sensor DS1820 eignet sich für problemlose und relativ einfache Temperaturmessungen mit einer Genauigkeit von ca. 0,5 Grad. Die Ansteuerung erfolgt über ein Eindraht-Interface und erfordert ein kleines Maschinenprogramm. Die hier vorgestellten Programme stammen von Herrn Thomas Grünberg (Email: Thomas_Gruenberg@compuserve.com)

7.1 DS1820.ASM

Das Übertragungsprotokoll mit nur einer Leitung ist nicht ganz einfach und erfordert eine dauernde Umschaltung der Datenrichtung auf einer Leitung . Hier wird die Leitung Port 1 (Port B1) verwendet. Das genaue Protokoll wird im Datenblatt des Sensors erläutert.

```
; DS1820.ASM, c) Thomas Grünberg

portb           .equ $01        ; Port B
pbdir           .equ $05        ; Datenrichtungsregister Port B

ds18s           .equ 0          ; Port 1
bas_1           .equ $a1        ; erste Basic Variable =>
Fehlerrückgabe
bas_2           .equ $a2        ; zweite Basic Variable => Daten
Rein&Raus
bas_3           .equ $a3        ; dritte Basic Variable => Interne
Verwendung


                .ORG $101

                bset ds18s,pbdir; Port 1 (Port_B.0) als Ausgang
loop            bclr ds18s,portb; Port 1 low
                lda  #$4b       ; 75 * 10 µs => ~750 µs
                jsr  delay      ; in Warteschleife springen
```

```
                bclr ds18s,pbdir; Port 1 high Z
                lda  #$08       ; 8 * 10 µs => ~80 µs
                jsr  delay      ; warten auf PRESENCE-PULSE-DS1820
                lda  portb      ; laden Port B
                and  #$01       ; nur bit1, alle anderen maskieren
                bne  error      ; Fehler-Flag setzen
                lda  #$2d       ; erneut 45 * ~10 µs => ~450 µs
                jsr  delay      ; warten
                lda  #$0f       ; Kein Fehler: 15 speichern
                sta  bas_1
senden          lda  #$08       ; 08 Bits
                sta  bas_3      ; sichern Anzahl
                bset ds18s,pbdir
byte_send       bclr ds18s,portb; Write 1 Slot ~5 µsec Port 1 Low
                clc             ; clear Carry bit
                ror  bas_2      ; in Carry schieben
                bcc  write0
                bcs  write1
write0          bclr ds18s,portb; o-schreiben
                bra  weiter
write1          bset ds18s,portb; 1-schreiben
weiter          lda  #$07       ; 7 * ~10 => 70 µs delay
                jsr  delay
                bset ds18s,portb; Port 1 rücksetzen
                dec  bas_3
                bne  byte_send  ; Nein => weiter
                rts
error           lda  #$ff       ; Ein Fehler => 255 speichern
                sta  bas_1
                rts             ; Fertig mit Senden und Raus !

byte_read       bset ds18s,pbdir; Port 1 als Ausgang
                lda  #$08       ; acht bits sind
                sta  bas_3      ; abzuarbeiten
                lda  #$00
                sta  bas_2

byte_read_b     bset ds18s,pbdir; Master Read 0 Slot ~2 µs P1 low
                bclr ds18s,portb; und Port 1 rücksetzen
                bclr ds18s,pbdir
                lda  #$01       ; 10 µs abwarten
                jsr  delay
                brset ds18s,portb,rot   ; Bit in C lesen
rot             ror  bas_2      ; aus C rechts schieben
                lda  #$05       ; Gesamtzykluszeit abwarten (~60µs)
                jsr  delay
                dec  bas_3      ; 8 Bits abgearbeitet ?
                bne  byte_read_b; Nein => weiter
```

```
                bra  das_wars   ; Fertig => und raus

delay           ldx  #$02       ; ca. 10 µs delay
inloop          decx
                bne  inloop
                deca
                bne  delay
                rts
das_wars        bset ds18s,pbdir
                bset ds18s,portb
                lda  #$0f       ; Rückkehrkode
                sta  bas_1      ; Fehlerkode speichern
                rts

.end
```

Das übersetzte Programm wird von einem Basic-Programm aufgerufen, um die einzelnen Übertragungsaktionen auszuführen.

```
    '************************************
    '
    ' C-Control/BASIC       DS1820_1.BAS
    '
    ' Aufgabe:
    '
    ' - Temperaturmessung mit DS1820
    ' - Anschluss an Port 1
    '
    '************************************
    ' --- Definitionen --------------------


    define  bas_1           byte
    define  bas_2           byte
    define  bas_3           byte

    define Temp_lo          byte
    define temp_hi          byte
    define temp             word

    ' --- Programmoperationen -------------


            print "testprogramm ds1820.bas"
    #main   bas_2 = &HCC    ' Reset & "Skip-Rom"
            sys &H0101      ' Sprung in Assembler
            pause 1
```

```
        bas_2 = &H44    ' "Convert-Temp" Kommando
        sys &H0120      ' Kommando senden
        pause 1
#loop   pause 20        ' 0.5 s warten
        sys &H0146      ' read DS1820
        if bas_2 = 0 then goto loop     ' busy?
        bas_2 = &HCC    ' Reset & "Skip-Rom"
        sys &H0101      ' Kommando senden
        pause 1
        bas_2 = &HBE    ' "Read-Scratchpad"
        sys &H0120      ' Kommando senden
        pause 1
        sys &H0146      ' 1 Byte lesen
        temp_lo = bas_2
        sys &H0146      ' 1 Bytes lesen
        temp_hi = bas_2
        pause 1
        temp = (10*temp_lo)/2
        if temp_hi <> 0 then temp = 2560-(10*temp_lo)/2
        print temp
        goto main
        end

syscode "ds1820.obj"
```

Das Programm liefert Temperaturwerte über die serielle Schnittstelle. Man kann die Messergebnisse an einem Terminalprogramm beobachten.

7.2 Temperatur-Datenlogger

Der folgende Temperatur-Datenlogger mit dem DS1820 misst eine Temperatur in der Minute und speichert die Daten in einem Datenfile im EEPROM ab. Die laufende Messung kann jederzeit mit einem Terminalprogramm unterbrochen werden, um alle bisher gemessenen Daten auszulesen. Das Verfahren wurde in ähnlicher Form bereits in [1] beschrieben. Die Daten lassen sich auch direkt in eine Excel-Tabelle übertragen, wobei ein kleines VBA-Makro die Rolle des Terminals übernimmt (vgl. [5]).

```
'************************************
'
' C-Control/BASIC       DS1820_2.BAS
```

```
'
' Aufgabe:
'
' - Temperaturmessung mit DS1820
' - Datenlogger
'
'************************************
' --- Definitionen --------------------


define  bas_1           byte
define  bas_2           byte
define  bas_3           byte

define Temp_lo          byte
define temp_hi          byte
define temp             word
define Kommando         byte

' --- Programmoperationen -------------

#Anfang
   open# for write
#Loop2
   gosub Messung
   gosub Warten
goto Loop2
end

#Messung
  gosub messen
  if (fileFree > 10) then print# Temp
return

#Warten
  if rxd then gosub Unterbrechung
  if Second > 0 then goto Warten
#Warten2
  if rxd then gosub Unterbrechung
  if Second = 0 then goto Warten2
return

#Unterbrechung
  get Kommando
  if Kommando = 27 then gosub Auslesen
return

#Auslesen
```

```
  close#
  open# for read
#Next
  input# Temp
  print Temp
  if not EOF then goto Next
  close#
  open# for append
return


#messen   bas_2 = &HCC    ' Reset & "Skip-Rom"
        sys &H0101      ' Sprung in Assembler
        pause 1
        bas_2 = &H44    ' "Convert-Temp" Kommando
        sys &H0120      ' Kommando senden
        pause 1
#loop   pause 20        ' 0.5 s warten
        sys &H0146      ' read DS1820
        if bas_2 = 0 then goto loop     ' busy?
        bas_2 = &HCC    ' Reset & "Skip-Rom"
        sys &H0101      ' Kommando senden
        pause 1
        bas_2 = &HBE    ' "Read-Scratchpad"
        sys &H0120      ' Kommando senden
        pause 1
        sys &H0146      ' 1 Byte lesen
        temp_lo = bas_2
        sys &H0146      ' 1 Bytes lesen
        temp_hi = bas_2
        pause 1
        temp = (10*temp_lo)/2
        if temp_hi <> 0 then temp = 2560-(10*temp_lo)/2
        print temp
return


syscode "ds1820.obj"
```

Um die gespeicherten Messwerte auszulesen, senden Sie ein Byte 27 aus dem Byte-Fenster oder drücken Sie die ESC-Taste im Textfenster des Terminals

7.3 Ein Temperaturregler

Das folgende Programm realisiert einen einfachen Temperaturregler. Die Solltemperatur ist 37 Grad (Brutkasten). Ein Heizelement wird über das Relais am Port 2 des Starterboards gesteuert.

```
'************************************
'
' C-Control/BASIC       DS1820_3.BAS
'
' Aufgabe:
'
' - Temperaturmessung mit DS1820
' - Temperatur-Regelkreis
'
'************************************
' --- Definitionen --------------------


define  bas_1           byte
define  bas_2           byte
define  bas_3           byte

define Temp_lo          byte
define temp_hi          byte
define temp             word
define Soll 270
define Relais Port[2]

' --- Programmoperationen -------------


        print "testprogramm ds1820.bas"
#main   bas_2 = &HCC    ' Reset & "Skip-Rom"
        sys &H0101      ' Sprung in Assembler
        pause 1
        bas_2 = &H44    ' "Convert-Temp" Kommando
        sys &H0120      ' Kommando senden
        pause 1
#loop   pause 20        ' 0.5 s warten
        sys &H0146      ' read DS1820
        if bas_2 = 0 then goto loop     ' busy?
        bas_2 = &HCC    ' Reset & "Skip-Rom"
        sys &H0101      ' Kommando senden
        pause 1
        bas_2 = &HBE    ' "Read-Scratchpad"
```

```
        sys &H0120      ' Kommando senden
        pause 1
        sys &H0146      ' 1 Byte lesen
        temp_lo = bas_2
        sys &H0146      ' 1 Bytes lesen
        temp_hi = bas_2
        pause 1
        temp = (10*temp_lo)/2
        if temp_hi <> 0 then temp = 2560-(10*temp_lo)/2
        print temp,
        if Temp > Soll then Relais = OFF
        if Temp < Soll then Relais = ON
goto main
end

syscode "ds1820.obj"
```

Die laufenden Messwerte können auch im Terminal beobachtet werden.

8 EEPROM-Erweiterung

Die C-Control-Unit verfügt über ein EEPROM mit 8 Kilobyte für Programm, Tabellen und ein längeres Datenfile z.B. für Messdaten. Meist kommt man damit reichlich aus, zumal auch aufwendige Programme selten länger als 2 K sind. Vor allem bei Langzeit-Messdatenerfassungen stößt man aber doch an die Grenzen des verfügbaren Speicherplatzes. Hier sollen zwei Wege gezeigt werden, den Speicher der Control-Unit zu erweitern.

8.1 Wahlfreier Speicherzugriff

Wenn Sie Datenfiles im EEPROM der CC-Unit anlegen, dann haben Sie immer nur einen sequentiellen Zugriff, d.h. Daten werden immer hintereinander als Words abgelegt, wobei jeder Speicherzugriff zwei Bytes benötigt. CC-Basic erlaubt auch nur ein Datenfile. Das ist für viele Aufgaben gut und einfach einzusetzen, aber es ist nicht sehr flexibel.

Ganz anders wäre es bei einem wahlfreien Zugriff. Sie könnten jedes der 8192 Bytes im EEPROM direkt lesen und verändern. Sie könnten mehrere Datenfiles verwalten, Sie könnten platzsparend Einzelbytes ablegen, und Sie könnten jede Menge Fehler bauen, indem Sie sich z.B. das laufende Basic-Programm überschreiben. Sie könnten? Nein Sie können!

Mit dem folgenden kleinen Assemblerprogramm haben Sie den gewünschten wahlfreien Zugriff. Insgesamt vier Bytes werden zwischen Basic und Maschinenprogramm ausgetauscht: I²C-Busadresse, Datenbyte und zwei Bytes für die interne Adresse des EEPROM. Schreibzugriffe erfordern einen Einsprung bei &H101, Lesezugriffe bei &H130. Das Programm verwendet Systemaufrufe aus dem CC-Betriebssystem, die schon in [1] erläutert wurden.

```
;Wahlfreier Zugriff auf ein EEPROM EEIO.ASM

AdrI2C  .equ $A1     ;Übergabe Busadresse
Dat     .equ $A2     ;Daten in/out
AdrHi   .equ $A3     ;interne Adresse
AdrLo   .equ $A4     ;als Word

       .org $101
;Einsprung $101, Daten schreiben
;CC-EEPROM vom Bus abmelden
        jsr $08BB    ;I2C_ReadLast
;Byte in eigenes EEPROM schreiben
        ldx AdrI2C   ;I2C-Adresse
        jsr $083C    ;I2C_Start
        ldx AdrHi    ;Adr High
        jsr $0846    ;I2C_Write
        ldx AdrLo    ;Adr low
        jsr $0846    ;I2C_Write
        ldx Dat      ;Daten
        jsr $0846    ;I2C_Write
        jsr $08E5    ;I2C_Stop
;CC-EEPROM wieder anmelden
        ldx #$0A0    ;Adresse 24C65, write
        jsr $083C    ;I2C_Start
        ldx $066     ;AdrCounter Hi
        jsr $0846    ;I2C_Write
        ldx $067     ;AdrCounter Lo
        jsr $0846    ;I2C_Write
        ldx #$0A1    ;Adresse 24C65, read
        jsr $083C    ;I2C_Start
        rts
```

```
;Einsprung $130, Daten lesen
;CC-EEPROM vom Bus abmelden
        jsr $08BB    ;I2C_ReadLast
;Byte aus eigenem EEPROM lesen
        ldx AdrI2C   ;I2C-Adresse
        jsr $083C    ;I2C_Start
        ldx AdrHi    ;Adr High
        jsr $0846    ;I2C_Write
        ldx AdrLo    ;Adr low
        jsr $0846    ;I2C_Write
        ldx $0A1     ;I2C-Adresse
        incx         ;Adr+1, Lesen
        jsr $083C    ;I2C_Start
        jsr $086F    ;I2C_Read
        sta Dat      ;Daten speichern
        jsr $08BB    ;I2C_ReadLast
;CC-EEPROM wieder anmelden
        ldx #$0A0    ;Adresse 24C65, write
        jsr $083C    ;I2C_Start
        ldx $066     ;AdrCounter Hi
        jsr $0846    ;I2C_Write
        ldx $067     ;AdrCounter Lo
        jsr $0846    ;I2C_Write
        ldx #$0A1    ;Adresse 24C65, read
        jsr $083C    ;I2C_Start
        rts
.end
```

Ein erstes kleines Programm soll den Lesezugriff demonstrieren, wobei ein kleines Basic-Programm sich selbst in Tokenform aus dem Speicher liest. Die I²C-Adresse ist hier &HA0. Das ist die Basisadresse des EEPROMs auf der CC-Unit. Haben Sie es bemerkt? Hier kann man ganz locker die Adresse eines eignen, zusätzlichen Speicherbausteins angeben.

```
'************************************
'
' C-Control/BASIC       EEio1.BAS
'
' Aufgabe:
'
' - Wahlfreier Zugriff auf ein EEPROM
' - Auslesen der CC-EEPROM-Inhalte
'
'************************************
```

```
' --- Definitionen --------------------

define AdrI2C  byte
define Dat  byte
define Adr  word

' --- Programmoperationen -------------

AdrI2C = &H0A0     'CC-EEPROM
for Adr = 0 to 40
  sys &H130        'EEPROM lesen
  print Dat,
next Adr

end

syscode "EEio.obj"
```

Das Programm hat übrigens eine Länge von 35 Bytes. Wer genau hinschaut, kann den Aufbau im EEPROM entschlüsseln. Nähere Informationen dazu findet man in [2].

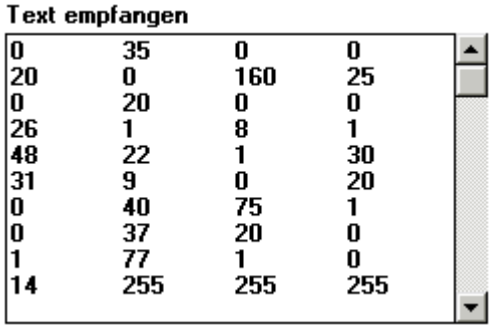

Abb. 8.1 Die ersten 40 Bytes im CC-EEPROM ((EEPRD.BMP))

Wahlfreie Schreibzugriffe auf das EEPROM der CC-Unit sind nun auch nicht schwieriger. Das folgende Programm beschreibt den Adressbereich 4000 bis 4255 mit einer aufsteigenden Zahlenfolge. Die Daten werden dann zur Kontrolle wieder zurückgelesen.

```
'************************************
'
' C-Control/BASIC       EEio2.BAS
'
' Aufgabe:
'
' - Wahlfreier Zugriff auf ein EEPROM
' - Schreiben eines Datenbereichs
'
'************************************
' --- Definitionen --------------------

define AdrI2C  byte
define Dat  byte
define Adr  word

' --- Programmoperationen -------------

AdrI2C = &H0A0     'CC-EEPROM
for Dat = 000 to 255
  adr = 4000 + Dat
  sys &H101        'EEPROM schreiben
next Dat

for Adr = 4000 to 4255
  sys &H130        'EEPROM lesen
  print Dat,
next Adr
end

'syscode "EEio.obj"
```

8.2 Das Huckepack-EEPROM

Das EEPROM 24C65 auf der C-Control-Unit verfügt über drei Adressleitungen A0, A1 und A2 zur Festlegung der I²C-Busadresse. Alle drei Leitungen liegen an Masse. Der Chip belegt damit die Adresse A0 beim Schreiben und A1 beim Lesen.

Theoretisch kann man noch sieben weitere EEPROMs der selben Art an den Bus anschließen, indem man andere Basisadressen einstellt. Das grundsätzliche

Verfahren wird hier mit einem zusätzlichen Speicherbaustein gezeigt. Der zweite 24C65 kann entweder über die nach außen geführten Anschlüsse SDA und SCL angeschlossen werden, oder man setzt ihn einen huckepack auf den ersten Baustein. Alle Leitungen bis auf A0 werden direkt angelötet. A0 wird mit VSS verbunden. Der zusätzliche Baustein belegt nun die Busadresse A2(hex) bzw. A3(hex) beim Lesen.

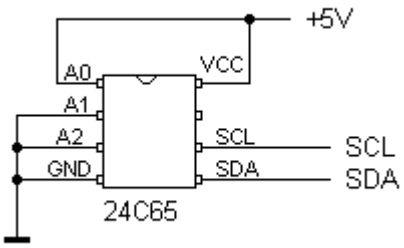

Abb. 8.1 Ein zweites EEPROM ((IC2.BMP))

Mit diesem Umbau hat man den verfügbaren Speicher verdoppelt. Allerdings wird das zusätzliche IC nicht automatisch vom Betriebssystem erkannt und verwendet. Man braucht also ein Zusatzprogramm wie EEio.ASM zur Ansteuerung. Die Adressen liegen auch wieder im Bereich 0 bis 8191, nur die I²C-Adresse ist geändert.

```
'************************************
'
' C-Control/BASIC       EEio3.BAS
'
' Aufgabe:
'
' - Zugriff auf ein zusätzliches EEPROM
' - Schreiben eines Datenbereichs
'
'************************************
' --- Definitionen --------------------

define AdrI2C  byte
define Dat  byte
define Adr  word
```

```
' --- Programmoperationen -------------

AdrI2C = &H0A2     'eigenes EEPROM
for Dat = 000 to 255
  adr = Dat
  sys &H101        'EEPROM schreiben
next Dat

for Adr = 000 to 255
  sys &H130        'EEPROM lesen
  print Dat,
next Adr
end

syscode "EEio.obj"
```

Achtung: Das Programm sollte nur gestartet werden, falls wirklich ein zusätzliches EEPROM angeschlossen wurde. Sonst wartet das CC-Betriebssystem beim ersten Speicherzugriff ganz geduldig, bis Sie fertig gelötet haben.

8.3 Das 32-K-EEPROM 24C256

Seit kurzer Zeit stellt ATMEL mit dem 24C256 ein 32-K-EEPROM her, das sich direkt in den Speichersockel der Control-Unit einsetzen läßt. Das Betriebssystem erkennt den Chip an und verwaltet nun problemlos 32 K.

Das EEPROM 24C256 von ATMEL ist noch sehr neu und nicht leicht zu erhalten. Bei Lieferproblemen können Sie sich an die Firma Modul-Bus (Email: service@modul-bus.de, Tel. 02574 8090) wenden. Diese Firma setzt das IC auch in anderen Projekten ein und kann deshalb auch Einzelstücke liefern.

Der originale 24C65 verwendet intern nur einen 13-Bit-Adresszeiger zur Verwaltung von 8 Kilobyte. Ein Blick in die Quelltexte des Betriebssystems zeigt, dass dort ein 16-Bit-Adresszeiger verwendet wird, so dass theoretisch mit bis zu 64 K gearbeitet werden kann. Ein weiteres Indiz dafür ist der beobachtete Speicherüberlauf bei zu langen Datenfiles. Das System scheint keine Obergrenze für den Speicher zu überwachen, so dass bei einem Überschreiten der Obergrenze das BASIC-Programm ab Adresse 0000 überschrieben wird, weil dann der interne 13-Bit-Zeiger des 24C65 überläuft. Diese Tatsache hat schon so

manchen Absturz und schmerzliche Datenverluste verursacht. Für den Speicheraustausch erweist sie sich jedoch als Vorteil. Alle Tests haben gezeigt: Es klappt, man kann die Control-Unit mit dem 24C256 ausrüsten. Das folgende Testprogramm schreibt 30000 Bytes in das EEPROM.

```
'************************************
'
' C-Control/BASIC       EEPROM32.BAS
'
' Aufgabe:
'
' - Test eines EEPROM 24C256
' - 15000 Worte = 30 K schreiben/lesen
'
'************************************
' --- Definitionen --------------------

define Ausgang port[1]
define n word
define data word

' --- Programmoperationen -------------

#Loop
  Ausgang = 1          'Einschalten
  pause 25             '0,5 s warten
  Ausgang = 0          'Ausschalten
  pause 25             '0,5 s warten

  open# for write
  for n= 1 to 15000
  print# (100+n)
  print (100+n)
  next n
  close#
  open# for read
  for n=1 to 16400
  input# data
  print data
  next n
  close#
end
```

Es werden 1500 Words geschrieben, aber weit mehr zurückgelesen. Man kann dabei beobachten, dass ein Adressüberlauf eintritt und man am Ende Daten liest,

die zum Basic-Programm ab Adresse 0000 gehören. Falls Sie übrigens dieses Programm mit einem normalen 8-K-EEPROM starten, wird es nie zum Auslesen kommen, weil das Basic-Programm schon vorher überschrieben wird.

Unter ganz bestimmten Voraussetzungen kann es zu Problemen mit dem EEPROM 24C256 kommen: Beim Ablegen von Daten mit print# verwendet der Basic-Interpreter in der CC-Unit den Block-Schreibmodus, bei dem beide Bytes eines Word nacheinander geschrieben und erst mit der I²C-Stoppbedingung programmiert werden. Laut Datenblatt dürfen bis zu 64 Bytes in einem Block geschrieben werden. Anders als beim 24C65 muss man dabei jedoch in einer 64-Byte-Reihe des Speicher-Arrays bleiben. Bei einem Übertrag wird die Reihenadresse nicht automatisch inkrementiert. Wenn also Words auf ungerade Adressen geschrieben werden, kommt es nach jeweils 32 Schreibzugriffen zu einem Schreibfehler. Dieser tritt also nur dann ein, wenn das Basic-Programm eine ungerade Anzahl von Bytes enthält. Die neueste Version des CC-Basic-Compilers erzeugt automatisch gerade Bytelängen. Wenn Sie eine ältere Version verwenden, sollten Sie ein Programm zur Probe einmal compilieren und dann bei Bedarf ein zusätzliches END anfügen, um auf eine gerade Programmlänge zu kommen.

9 Ein Terminalprogramm

Leider verfügt die Programmieroberfläche CCEW32D nicht über ein eingebautes Terminalprogramm zum Testen von Programmen. Man muss also auf ein zusätzliches Programm wie z.B. Hyperterminal ausweichen.
Aber auch Hyperterminal hat einen Nachteil. Es unterstützt nur den Datenaustauch im Textformat. Alle Ein- und Ausgaben mit PRINT und INPUT können damit verwendet werden. Wenn man jedoch Binärdaten, also einzelne Bytes verwenden will, braucht man noch ein weiteres Programm.
Hier soll nun ein kleines, aber vielseitiges Terminalprogramm vorgestellt werden, das beide Formate unterstützt. Das Programm CCTerm.exe wurde in VB3 geschrieben und läuft unter Win3.1, Win95 und Win98. Es liegt hier im Quelltext vor und kann als Grundlage für eigene Entwicklungen verwendet werden.

9.1 C-Control Terminalprogramm

Das Programm CCTERM wurde in VB3 geschrieben. Das Terminalprogramm besitzt vier Fenster, jeweils zwei zum Absenden von Daten und zwei zum Datenempfang. Davon ist jeweils ein Fenster für die Übertragung von Texten zuständig, das andere interpretiert Daten binär, also als Einzelbytes. Der Anwender muss entscheiden, ob er Bytes oder Text senden will. Empfangene Daten werden grundsätzlich in beiden Formen dargestellt. Man sieht dann schon, wie die Daten gemeint sind.

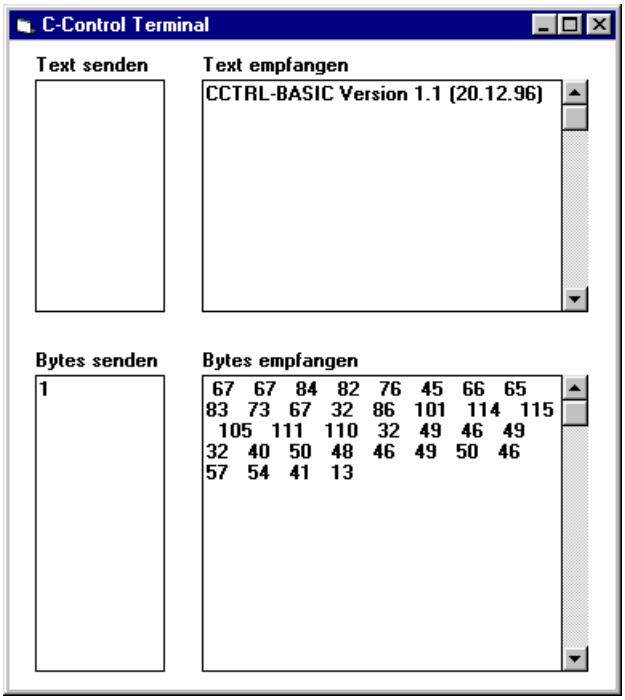

Abb. 9.1 Das Terminalprogramm ((Term.BMP))

Im Beispiel wurde nur ein einzelnes Byte "1" abgesandt. Nach einen Reset reagiert die CC-Unit darauf, indem sie einen Erkennungstext absendet: "CCTRL-BASIC Version 1.1 (20.12.96)". Der empfangene Text wird im unteren Fenster auch in binärer Form angezeigt: ASCII 67 = "C", ASCII 84= "T" usw. bis ASCII 13 = CR, Zeilenende. Bei Bedarf kann man Daten aus einem Empfangsfenster in die Zwischenablage kopieren und in anderen Programmen weiterverwenden.

Hier nun der Quelltext aus Visual Basic 3.0:

```
Dim filename As String

Function Empfang () As Integer
  t = Timer
  Do
  Loop Until ((t + .2) < Timer) Or (Comm1.InBufferCount > 0)
  If Comm1.InBufferCount > 0 Then Empfang = Asc(Comm1.Input)
     Else Empfang = 0
End Function

Sub Form_Load ()
  Comm1.CommPort = 2
  Comm1.Settings = "9600,N,8,1"
  Comm1.InputLen = 1
  Comm1.PortOpen = True
  If Comm1.PortOpen = False Then
    Comm1.CommPort = 1
    Comm1.PortOpen = True
  End If
End Sub

Sub Form_Unload (Cancel As Integer)
  Comm1.PortOpen = False
End Sub

Sub Sende (Zeichen As Integer)
  Comm1.Output = Chr$(Zeichen)
End Sub

Sub Text1_KeyPress (KeyAscii As Integer)
   Sende (KeyAscii)
End Sub

Sub Text4_Change ()
 byt = Asc(Right$(Text4.Text, 1))
 If byt = 10 Then
```

```
   Zahl$ = (Right$(Text4.Text, 5))
   A = Val(Zahl$)
   A = A And 255
   Sende (A)
   ' Text3.Text = Str$(A)

 End If
End Sub

Sub Timer2_Timer ()
  Do
  If Comm1.InBufferCount > 0 Then byt = Asc(Comm1.Input)
     Else byt = -1
  If (byt > -1) And (byt <> 13) And (byt <> 10)
     Then Text2.Text = Text2.Text + Chr$(byt)
  If byt = 13 Then
    Text2.Text = Text2.Text + Chr$(13) + Chr$(10)
    Text2.Refresh
  End If
  If (byt > -1) Then
     Text3.Text = Text3.Text + Str$(byt) + "  "
     'n = n + 1
     'If n = 5 Then n = 0: Text3.Text = Text3.Text
         + Chr$(13) + Chr$(10)
  End If
  Loop Until Comm1.InBufferCount = 0
End Sub
```

9.2 Terminal-Testprogramme

Wenn Sie Daten über die RS232-Schnittstelle austauschen wollen, muss immer entscheiden sein, ob es sich um ASCII-Text oder um binäre Daten, also um Bytes handeln soll. Das erste Beispiel verwendet das Textformat. Daten vom Terminal sollen am Byteport 1 ausgegeben werden. Dann wird Byteport 2 gelesen und das Ergebnis zurückgesandt.

```
   '************************************
   '
   ' C-Control/BASIC       TermTst1.BAS
   '
   ' Aufgabe:
```

```
'
' - Datenkommunikation im Textformat
' - Schreiben und Lesen von Portdaten
'
'*************************************
' --- Definitionen --------------------

define Ausgabeport Byteport[1]
define Eingabeport Byteport[2]
define Daten   Byte

' --- Programmoperationen -------------

print "Bitte Ausgabedaten eingeben"
#Loop
  Input Daten
  Ausgabeport = Daten
  Daten = Eingabeport
  Print Daten
goto Loop
end
```

Beachten Sie, dass die Daten als Text gesendet und empfangen werden. Vom Terminal abgesandte Daten werden in CC-Basic mit INPUT empfangen, Textdaten werden mit PRINT zurückgesandt.

Im Falle des Binärformats empfängt CC-Basic Datenbytes mit GET und sendet sie mit PUT. Das folgende Beispiel zeigt die Übertragung von Einzelbytes.

```
'*************************************
'
' C-Control/BASIC       TermTest2.BAS
'
' Aufgabe:
'
' - Datenkommunikation im Binärformat
' - Schreiben und Lesen von Portdaten
'
'*************************************
' --- Definitionen --------------------

define Ausgabeport Byteport[1]
define Eingabeport Byteport[2]
define Daten   Byte

' --- Programmoperationen -------------
```

```
#Loop
  Get Daten
  Ausgabeport = Daten
  Daten = Eingabeport
  Put Daten
goto Loop
end
```

10 Download und Upload

Haben Sie sich das auch schon einmal gefragt: Wie läuft eigentlich die Kommunikation zwischen der Control-Unit und dem PC ab? Wenn man die Protokolle kennt, kann man auch eigene Programme schreiben.
Das Betriebssystem unterstützt sogar mehr Funktionen als von der Standard-Software genutzt werden. Hier findet sich ein weites Feld für eigene Experimente.

10.1 System-Kommandos

Die Control-Unit versteht einige einfache Byte-Kommandos für Programm-Download, Upload und anderes. Das Betriebssystem ist in [2] dokumentiert. Einzelheiten erkennt man in den veröffentlichten Quelltexten.

Byte-Kommando	Funktion
1	Kennung, CC antwortet: "CCTRL-BASIC Version 1.1 (20.12.96)"
2, Hi, Lo ,n Bytes	Basic-Download ins EEPROM, Hi, Lo, n = Anzahl der Bytes, Hi, Lo und alle Programmbytes werden als Kopie zurückgesandt
3	Programm-Upload, CC sendet das komplette Programm zurück

8, n, n Bytes	SYS-Download ins SYS-EEPROM, n = Anzahl der Bytes, n und alle Programmbytes werden als Kopie zurückgesandt
9	SYS-Upload, 255 Bytes werden zurückgesandt
13	DumpFile: Ein vom Programm angelegtes Datenfile wird ausgelesen.
14, Sec, Min, Std, Wochentag, Tag, Monat, Jahr	SetTime: Die interne Uhr wird gestellt. Alle sieben Parameter werden als Byte übertragen.

10.2 Download-Programm in VB3

Manchmal ist es hilfreich, wenn man mit einem eigenen Programm den fertig entwickelten Programmcode in die CC-Unit übertragen kann. Man kann eine Download-Routine dann z.B. in größere Programme einbauen. Hier wird Visual Basic 3 verwendet. Damit schließt sich eine Lücke zwischen Windows95/98 und DOS. Wer noch einen alten PC mit Win3.1 besitzt, kann ihn nun auch für C-Control einsetzen. Es ist nicht mehr unbedingt nötig, auf die DOS-Ebene herunterzusteigen und das DOS-Programm CCDL.EXE zu verwenden. Das hier vorgestellte Programm wurde auch für den Online-Einsatz auf dieser CD geschrieben, die ja für Win3.1 und Win95/98 einsetzbar sein sollte.

CCDOWN.EXE überträgt kompilierte Programme vom Typ *.DAT. Eine BAS-Datei wird mit dem DOS-Programm CCBAS.EXE in das DAT-Format übersetzt. CCBAS kann auf neuen Win95/98-Systemen Probleme bereiten, wenn das FAT32-Dateisystem benutzt wird. Problemlösung: Kopieren Sie CCBAS.EXE und die Quelltexte auf eine Diskette und starten Sie das Programm von der Diskette. Die DAT-Dateien können dann unter Windows mit CCDOWN übertragen werden.

Das Programm CCDOWN.EXE verwendet die serielle Schnittstelle über MSCOMM.VBX. Beim Start des Programms wird standardmäßig COM2 geöffnet. Falls dies misslingt, was z.B. bei schon belegter Schnittstelle möglich ist und was zu einer entsprechenden Fehlermeldung führt, wird COM1 geöffnet.

Das Programm wertet dann den übergebenen Parameterstring als Dateinamen aus und öffnet die entsprechende DAT-Datei. Die Daten werden entsprechend dem Übertragungsprotokoll in das BASIC-EEPROM und - falls SYS-Code angehängt ist - in das interne EEPROM des Prozessors übertragen.

Programme müssen zuvor mit dem DOS-Programm CCBAS.EXE in eine DAT-Datei übersetzt werden. CCdown.EXE wertet übrigens keine Rückmeldung der Control-Unit aus und sendet auch dann, wenn keine Unit angeschlossen ist. Man muss also selbst darauf achten, dass die Unit bereit ist.

Als letztes und zusätzliches Byte sendet CCdown ein Byte 15 ab. Falls die Systemerweiterung START.ASM aktiv ist, wird damit das geladene Programm automatisch gestartet. Falls nicht, muss die gelbe Start-Taste verwendet werden.

```
Dim filename As String

Function Empfang () As Integer
  t = Timer
  Do
  Loop Until ((t + .2) < Timer) Or (Comm1.InBufferCount > 0)
  If Comm1.InBufferCount > 0 Then Empfang = Asc(Comm1.Input)
Else Empfang = 0
End Function

Sub Form_Load ()
  Comm1.CommPort = 2
  Comm1.Settings = "9600,N,8,1"
  Comm1.InputLen = 1
  Comm1.PortOpen = True
  If Comm1.PortOpen = False Then
    Comm1.CommPort = 1
    Comm1.PortOpen = True
  End If
  filename = Command
  If filename > "" Then
    msg$ = filename & " wird geladen "
    MsgBox msg$, 0
    Open (filename) For Input As #1
    Input #1, CCBASIC$
    Input #1, i
    Sende 2               'Ladekommando BAS
    Sende i \ 256         'Highbyte Anzahl
    Dummy = Empfang()
    Sende i Mod 256       'Lowbyte Anzahl
```

```
    Dummy = Empfang()
    For n = 1 To i
      Input #1, Dat
      Sende (Dat Mod 256)
      Dummy = Empfang()
    Next n
    Input #1, i
    If i > 0 Then
      Sende 8               'Ladekommando SYS
      Sende (i Mod 256)         'Anzahl SYS-Bytes
      Dummy = Empfang()
      For n = 1 To i
        Input #1, Dat
        Sende (Dat Mod 256)
        Dummy = Empfang()
    Next n
    End If
    Close #1
    Sende 15  'Nur für Autostart (neues Kommando)
  Else msg$ = "Aufruf: Download Dateiname"
  MsgBox msg$, 0
  End If
  End
End Sub

Sub Form_Unload (Cancel As Integer)
  Comm1.PortOpen = False
End Sub

Sub Sende (Zeichen As Integer)
  Comm1.Output = Chr$(Zeichen)
End Sub
```

Es wird das Programm TEST.DAT übertragen. Während der Übertragung leuchtet die rote LED der CC-Unit. Danach kann man den Starttaster betätigen. TEST-DAT lässt den Port 1 im Sekundentakt blinken.

TEST.DAT enthält ein kurzes BASIC-Programm mit 23 Bytes ohne ein SYS-Programm (0 Bytes). Wenn ein Programm auch einen SYS-Bereich verwendet, dauert die Übertragung erheblich länger, weil die Programmierung eines Bytes im internen EEPROM des Prozessors mehr Zeit beansprucht.

```
CCTRL-BASIC
23
```

```
20 0 1 13 0 20 0 25 2 20 0 0 13 0 20 0 25 2 3 0 4 255 255
0
```

10.3 Die Uhr stellen

Das Kommando 14 ermöglicht ein automatisches Stellen der Uhr auch ohne eine aktive DFF77-Antenne. Das folgend kleine VB3-Programm überträgt die Zeitinformationen des PC an die CC-Unit.

```
Sub Form_Load ()
  Comm1.CommPort = 2
  Comm1.Settings = "9600,N,8,1"
  Comm1.InputLen = 1
  Comm1.PortOpen = True
  If Comm1.PortOpen = False Then
    Comm1.CommPort = 1
    Comm1.PortOpen = True
  End If
  MsgBox "C-Control Uhr stellen", 0
  Sende 14              'Kommando Uhr stellen
  Sende Second(Now) Mod 256
  Sende Minute(Now) Mod 256
  Sende Hour(Now) Mod 256
  Sende Weekday(Now) Mod 256
  Sende Day(Now) Mod 256
  Sende Month(Now) Mod 256
  Sende (Year(Now) Mod 100) Mod 256
  End
End Sub
```

Zum Testen sollen die Zeitinformationen über die serielle Schnittstelle ausgegeben werden. Dazu benötigt man ein kleines Hilfsprogramm:

```
'************************************
'
' C-Control/BASIC       TIMEREAD.BAS
'
' Aufgabe:
'
' - Auslesen der Echtzeituhr
' - Senden an den PC
```

```
'
'************************************
' --- Definitionen --------------------

define Sekunde byte

' --- Programmoperationen -------------

#Loop
  print hour, minute, second
  print day, month, year
  #Loop2
  if Sekunde = second then goto Loop2
  Sekunde = second
goto Loop
end
```

11 C-Control als Interface

In [2] wurde eine Interface-Steuerung mit dem Maschinenprogramm CCIO.ASM vorgestellt. Der Anwender kann mit speziellen Programmen direkt auf alle Ein- und Ausgänge des C-Control zugreifen, ohne eigene Programme schreiben zu müssen. Typisch ist der Einsatz des Windows-Programms Do-It von Modul-Bus (http://www.modul-bus.de), das vor allem für die Ausbildung entwickelt wurde.

In CCIO.ASM wurde eine feste Zuordnung der Ein- und Ausgänge vorgenommen. Byteport 1 stellt acht Ausgänge, Byteport 2 wird nur als Eingang benutzt. Von den acht möglichen Analogeingängen wurden nur vier verwendet. Die C-Control-Unit als Interface kann man auch von eigenen Programmen aus ansteuern. So läßt sich z.B. eine komplexe Eisenbahnsteuerung mit C-Control als Interface zwischen PC und Modell realisieren. Manchmal möchte man jedoch mehr als acht Ausgänge nutzen. Dann stört die starre Zuordnung von Ein- und Ausgängen. Das ursprüngliche Programm wurde deshalb um weitere Kommandos erweitert, die eine flexible Zuordnung erlauben.

11.1 CCIO und Do-It

Sie können mit speziellen Programmen direkt auf alle Ein- und Ausgänge der Control-Unit zugreifen, ohne eigene Programme schreiben zu müssen. Typisch ist der Einsatz des Windows-Programms Do-It, das vor allem für die Ausbildung entwickelt wurde. (Hersteller: Modul-Bus www.modul-bus.de)

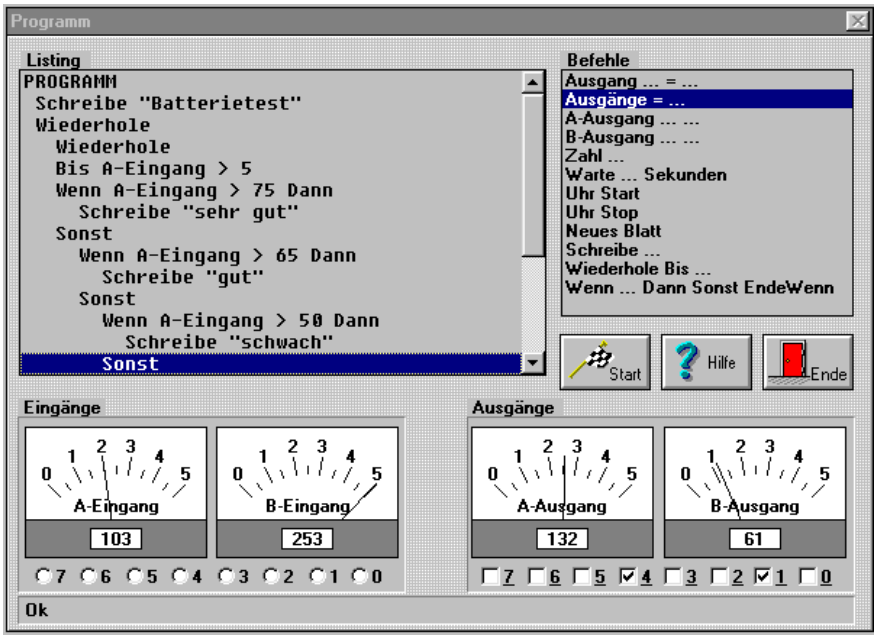

Abb. 11.1 Das Programm Do-It ((Doit.BMP))

```
****************************************
*
*  CCIO.ASM
*  Kommando-Interpreter mit 9600 Baud
*
****************************************
        opt c,l

*Ports
```

```
portb   equ $01 ;Daten Port B (Port1...8)
portc   equ $02 ;Daten Port C (Port9...16)
pbdir   equ $05 ;Datenrichtung Port B
pcdir   equ $06 ;Datenrichtung Port C

*Serielle Schnittstelle
brate   equ $0D ;SCR-Baudrate
sccr1   equ $0E ;Controlregister 1
sccr2   equ $0F ;Contrllregister 2
scsr    equ $10 ;Status-Register
scdat   equ $11 ;Datenregister

*AD-Wandler
addata  equ $08 ;Datenregister
adstat  equ $09 ;Status/Control

*DA-Wandler
misc    equ $0C ;Ausgaberate
plma    equ $0A ;PLM A
plmb    equ $0B ;PLM B

*Variablenspeicher im RAM
Kom     equ $0A1        ;Kommando
ADC     equ $0A2        ;AD-Kanal

        org $101        ;Startadresse om C-Control

init    lda #$FF        ;P1...P8 Ausgaenge
        sta pbdir
        lda #$00        ;P1...P8 = 0
        sta portb
        lda #$C0        ;19200 Baud
        sta brate
        lda #$00        ;Ohne Interrupts
        sta sccr1
        lda #$0C        ;RxD und TxD freigeben
        sta sccr2
        lda #0          ;PLM-Kanaele schnell
        sta misc

Loop    jsr RdCOM       ;Kommando empfangen
        sta Kom         ;empfangenes Kommando speichern
        lda #$01        ;Kommando 1: ID-Byte senden
        cmp Kom ;A=Kom?
        bne K16         ;nein: ueberspringen
        lda #$14        ;ID-Byte 20
        jsr WrCOM       ;senden
```

```
K16     lda #$10        ;Kommando 16, Portausgabe
        cmp Kom ;A=Kom?
        bne K32         ;nein: ueberspringen
        jsr RdCOM       ;empfangen
        sta portb       ;Port B ausgeben

K32     lda #$20        ;Kommando 32, Porteingabe
        cmp Kom ;A=Kom?
        bne K48         ;nein: ueberspringen
        lda portc       ;Port C lesen
        jsr WrCOM       ;senden

K48     lda #$30        ;Kommando 48, Analog A
        cmp Kom         ;A=Kom?
        bne K49         ;nein: ueberspringen
        lda #0          ;Kanal 0
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden

K49     lda #$31        ;Kommando 49, Analog B
        cmp Kom         ;A=Kom?
        bne K50         ;nein: ueberspringen
        lda #1          ;Kanal 1
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden

K50     lda #$32        ;Kommando 50, Analog C
        cmp Kom         ;A=Kom?
        bne K51         ;nein: ueberspringen
        lda #2          ;Kanal 2
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden

K51     lda #$33        ;Kommando 50, Analog D
        cmp Kom         ;A=Kom?
        bne K64         ;nein: ueberspringen
        lda #3          ;Kanal 3
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden

K64     lda #$40        ;Kommando 64, PLM A
        cmp Kom         ;A=Kom?
        bne K65         ;nein: ueberspringen
        jsr RdCOM       ;empfangen
```

```
        sta plma

K65     lda #$41        ;Kommando 65, PLM B
        cmp Kom         ;A=Kom?
        bne K100        ;nein: ueberspringen
        jsr RdCOM       ;empfangen
        sta plmb

K100    bra Loop        ;Endlosschleife

RdCOM   brclr   5,scsr,RdCOM    ;warten bis Empfang
        lda     scdat           ;Daten lesen
        rts

WrCOM   brclr   7,scsr,WrCOM    ;warten bis bereit
        sta     scdat           ;Daten schreiben
        rts

RdAD    lda     #$20            ;AD-Wandler aktiv
        ADD     ADC             ;plus Kanal
        sta     adstat          ;Wandlung starten
Wait    brclr   7,adstat,Wait   ;warten bis fertig
        lda     addata          ;Ergebnis lesen
        rts

        end
```

```
'************************************
'
' C-Control/BASIC       CCIO.BAS
'
' Aufgabe:
'
' - Interface-Emulation
' - Assembler-Unterprogramm CCIO
'
'************************************
' --- Definitionen --------------------

define Ausgang Byteport[1]

' --- Programmoperationen -------------

Ausgang = 0
sys &H0101            'Aufruf bei $0101
end

syscode "CCIO.S19"   'Code laden
```

11.2 CCIOn - die Erweiterung

Gegenüber der ersten Version besitzt CCIOn folgende Erweiterungen:

- Es können nun sechs Analogeingänge abgefragt werden. (Kommandos 48 ... 53)
- Die Zuordnung der Ports ist flexibel.
- Es gibt jeweils zwei Schreib- (16, 17) und Lesebefehle (32, 33) für die Byteports 1 und 2.

Die Zuordnung der Ein- und Ausgänge zu den einzelnen Ports erfolgt erst im aufrufenden BASIC-Programm. Der Anwender kann also mit wenig Aufwand bis zu 16 Eingänge oder bis zu 15 Ausgänge festlegen.

Das Programm reicht fast an die Grenze von 255 Bytes im SYS-Bereich. Deshalb war es nicht möglich alle acht Analogeingänge zu unterstützen. Das Programm wurde mit TASM übersetzt.

```
;****************************************
;*
;*  CCIO.ASM
;*  Kommando-Interpreter mit 9600 Baud
;*
;****************************************

;Ports
portb   .equ $01        ;Daten Port B (Port1...8)
portc   .equ $02        ;Daten Port C (Port9...16)
pbdir   .equ $05        ;Datenrichtung Port B
pcdir   .equ $06        ;Datenrichtung Port C

;Serielle Schnittstelle
brate   .equ $0D        ;SCR-Baudrate
sccr1   .equ $0E        ;Controlregister 1
sccr2   .equ $0F        ;Contrllregister 2
scsr    .equ $10        ;Status-Register
scdat   .equ $11        ;Datenregister
```

```
;AD-Wandler
addata  .equ $08        ;Datenregister
adstat  .equ $09        ;Status/Control

;DA-Wandler
misc    .equ $0C        ;Ausgaberate
plma    .equ $0A        ;PLM A
plmb    .equ $0B        ;PLM B

;Variablenspeicher im RAM
Kom     .equ $0A1       ;Kommando
ADC     .equ $0A2       ;AD-Kanal

        .org $101       ;Startadresse om C-Control

init    lda #$FF        ;P1...P8 Ausgaenge
        sta pbdir
        lda #$00        ;P1...P8 = 0
        sta portb
        lda #$C0        ;19200 Baud
        sta brate
        lda #$00        ;Ohne Interrupts
        sta sccr1
        lda #$0C        ;RxD und TxD freigeben
        sta sccr2
        lda #0          ;PLM-Kanaele schnell
        sta misc

Loop    jsr RdCOM       ;Kommando empfangen
        sta Kom         ;empfangenes Kommando speichern
        lda #$01        ;Kommando 1: ID-Byte senden
        cmp Kom ;A=Kom?
        bne K16         ;nein: ueberspringen
        lda #$14        ;ID-Byte 20
        jsr WrCOM       ;senden

K16     lda #$10        ;Kommando 16, Portausgabe
        cmp Kom ;A=Kom?
        bne K17         ;nein: ueberspringen
        jsr RdCOM       ;empfangen
        sta portb       ;Port B ausgeben

;neu Ausgabe über Byteport 2, im Basicprogramm wird erst
;festgelegt, welche Bit Ausgänge sind

K17     lda #$11        ;Kommando 17, Portausgabe Byteport2
        cmp Kom ;A=Kom?
        bne K32         ;nein: ueberspringen
```

```
        jsr RdCOM       ;empfangen
        sta portc       ;Port B ausgeben

K32     lda #$20        ;Kommando 32, Porteingabe
        cmp Kom ;A=Kom?
        bne K33         ;nein: ueberspringen
        lda portc       ;Port C lesen
        jsr WrCOM       ;senden

;neu Eingaben jetzt auch über Byteport 1, Festlegung in Basic

K33     lda #$21        ;Kommando 33, Porteingabe Byteport1
        cmp Kom ;A=Kom?
        bne K48         ;nein: ueberspringen
        lda portb       ;Port B lesen
        jsr WrCOM       ;senden

K48     lda #$30        ;Kommando 48, Analog A
        cmp Kom         ;A=Kom?
        bne K49         ;nein: ueberspringen
        lda #0          ;Kanal 0
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden

K49     lda #$31        ;Kommando 49, Analog B
        cmp Kom         ;A=Kom?
        bne K50         ;nein: ueberspringen
        lda #1          ;Kanal 1
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden

K50     lda #$32        ;Kommando 50, Analog C
        cmp Kom         ;A=Kom?
        bne K51         ;nein: ueberspringen
        lda #2          ;Kanal 2
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden


K51     lda #$33        ;Kommando 51, Analog D
        cmp Kom         ;A=Kom?
        bne K52         ;nein: ueberspringen
        lda #3          ;Kanal 3
        sta ADC         ;speichern
        jsr RdAD        ;messen
```

```
        jsr WrCOM       ;senden

;neu: noch zwei Analogkanäle

K52     lda #$34        ;Kommando 52, Analog 5
        cmp Kom         ;A=Kom?
        bne K53         ;nein: ueberspringen
        lda #4          ;Kanal 3
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden

K53     lda #$35        ;Kommando 53, Analog 6
        cmp Kom         ;A=Kom?
        bne K64         ;nein: ueberspringen
        lda #5          ;Kanal 3
        sta ADC         ;speichern
        jsr RdAD        ;messen
        jsr WrCOM       ;senden


K64     lda #$40        ;Kommando 64, PLM A
        cmp Kom         ;A=Kom?
        bne K65         ;nein: ueberspringen
        jsr RdCOM       ;empfangen
        sta plma

K65     lda #$41        ;Kommando 65, PLM B
        cmp Kom         ;A=Kom?
        bne K100        ;nein: ueberspringen
        jsr RdCOM       ;empfangen
        sta plmb

K100    jmp Loop        ;Endlosschleife

RdCOM   brclr   5,scsr,RdCOM    ;warten bis Empfang
        lda     scdat           ;Daten lesen
        rts

WrCOM   brclr   7,scsr,WrCOM    ;warten bis bereit
        sta     scdat           ;Daten schreiben
        rts

RdAD    lda     #$20            ;AD-Wandler aktiv
        ADD     ADC             ;plus Kanal
        sta     adstat          ;Wandlung starten
Wait    brclr   7,adstat,Wait   ;warten bis fertig
        lda     addata          ;Ergebnis lesen
```

```
        rts

        .end
```

Das Basic-Programmbeispiel zeigt die Verwendung von 10 Ausgängen und 6 Eingängen.

```
'************************************
'
' C-Control/BASIC       CCIOn.BAS
'
' Aufgabe:
'
' - Interface-Emulation, erweitert
' - Assembler-Unterprogramm
'
' - Zusätzliche Ausgänge am Byteport 2
' - werden hier zugewiesen (im Beispiel P9, P10)
' - bis maximal 16 Ausgänge oder 16 Eingänge
' - neue Kommandos: 17 (Ausgabe2) und 33 (Eingabe2)
'
' - Neue Kommandos 52-53 für 2 weitere Analogeingänge
'
'************************************
' --- Definitionen --------------------

define Ausgang Byteport[1]
define Ausgang9 Port[9]
define Ausgang10 Port[10]

' --- Programmoperationen -------------

Ausgang = 0
Ausgang9 = 0
Ausgang10 = 0
sys &H0101            'Aufruf bei $0101
end

syscode "CCIOn.obj"   'Code laden
```

12 Low-Power-Betrieb der CC-Unit

Haben Sie schon einmal daran gedacht, die Control-Unit mit Batterien zu betreiben? Auf den ersten Blick erscheint das nicht sehr wirtschaftlich, weil der Strom laut Datenblatt mit ca. 30 mA recht groß ist. Aber Sie werden sehen: Man kommt auch mit wesentlich weniger aus.

Das Datenblatt zum Prozessor 68HC05B6 gibt Anlaß zur Hoffnung, denn hier findet man Angaben zum Stromverbrauch zwischen 3,5 mA und 0,35 mA, je nach Modus und Arbeitsgeschwindigkeit. Die Frage ist also zunächst: Welche anderen Bauteile auf der CC-Unit brauchen viel Strom, und was kann man dagegen tun?

Es gibt im wesentlichen drei Bereiche, in denen man Strom sparen kann. Ein großer Verschwender ist der Leitungstreiber MAX232 der Control-Unit. Als zweites kann man den Wait-Modus des Prozessors nutzen und dabei auch das EEPROM abschalten. Und schließlich kann man sogar den Stop-Modus verwenden und damit den Verbrauch fast auf Null bringen.

12.1 Max232-Ersatz

Wieviel Strom braucht eigentlich die CC-Unit wirklich? Im normalen Betrieb, also bei laufendem Programm, misst man etwa 25 mA. Ziehen Sie den RS232-Stecker vom PC ab, sind es nur noch 16 mA. Wer hätte das gedacht: 9 mA versickern im PC, und zwar genauer in den Widerständen der RS232-Eingänge.

Entfernen Sie einmal den LED-Jumper. Sie sparen weitere 7 mA, brauchen also noch 9 mA. Und dann gibt es ja noch den RS232-Jumper, mit dem man dem MAX232 den Strom entziehen kann. Leider bringt das nicht sehr viel, Der Strom beträgt nun ca. 6,5 mA. Die Betriebsspannung des MAX232, gemessen am Pin 16, ist leider nicht Null, sondern immer noch ca. 1,5 V.

Schuld daran ist die interne Schutzschaltung der CMOS-Eingänge. Über interne Begrenzerdioden gelangt ein Signal von zwei Leitungen des Prozessors an den internen Betriebsspannungsanschluss. Die Leitung TD= (RS232-Senden) liegt

hoch und quält sich redlich, den MAX232 mit Strom zu versorgen. Deshalb wird nicht viel aus dem Stromsparen.

Nun gibt es zwei mögliche Wege, die Verschwendung des RS232-Interface zu unterbinden: Zum einen kann man die entsprechenden Leitungen low-schalten, solange man die RS232 nicht benötigt. Der zweite Weg führt über einen Ersatz des MAX232 durch eine einfache Transistorschaltung. Das RXD-Signal wird durch einen NPN-Transistor aufbereitet, das TXD-Signal durch einen PNP-Transistor. Beide sind im Ruhezustand völlig stromlos. Die CTS-Leitung am Prozessor benötigt zusätzlich einen Pullup-Widerstand, da ein völlig offener CMOS-Eingang unkontrollierte Ströme verursachen kann.

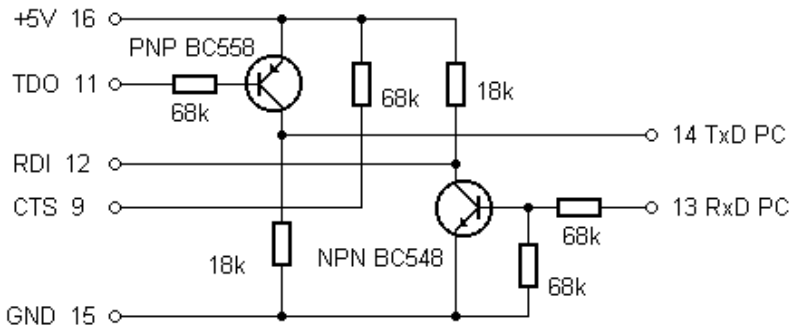

Abb. 12.1 Die stromsparende Ersatzschaltung ((RS232.BMP))

Die Schaltung lässt sich auf einer IC-Fassung aufbauen, die direkt an Stelle des MAX232 eingesetzt wird. Allein der neue Leitungstreiber bringt sehr viel. Die Control-Unit braucht nun ohne weitere Maßnahmen nur noch ca. 2 mA im Wartezustand und ca. 3 mA bei laufendem Programm.

Sie können mit dem Basic-Kommando SlowMode 1 einen stromsparenden Modus einschalten, bei dem der Prozessor mit einem 16-tel der normalen Taktfrequenz läuft. Wenn die RS232 nichts mehr verschwendet, kommen sie nun auf ca. 2 mA.

12.2 Der Wait-Modus

Laut Datenblatt des Prozessors benötigt der Wait-Modus erheblich weniger Strom, nämlich nur 0,35 mA im Slow-Modus. Er wird aber im CC-Betriebssystem ohnehin verwendet, wenn ein Pause-Befehl ausgeführt wird. Viel Pause - wenig Strom! Aber warum ist dann immer noch so viel?

Auf der Suche nach weiteren Einsparmöglichkeiten fallen die Pullup-Widerstände der Control-Unit als hungrige Energiefresser auf. Da ist zunächst der Widerstand an der RTS-Leitung (Port A6) des Prozessors. Die Leitung ist im Normalfall heruntergeschaltet, so dass nutzlose 0,5 mA durch den Pullup-Widerstand fließen. Da RTS mit dem eigenen Leitungstreiber ohnehin nicht mehr benötigt wird, kann man die Leitung besser hochschalten. Leider gibt es keinen passenden Basic-Befehl. Ein kleines Maschinenprogramm muss helfen. Es besteht praktisch nur aus einer Zeile, wenn man den Rücksprungbefehl nicht mitzählt.

```
bset 6,0 ;PA6=RTS hochschalten (-0,5mA)
```

Als nächstes fällt auf, dass das EEPROM auf der Platine während eines laufenden Programms auch dann am I²C-Bus angemeldet bleibt, wenn gerade kein Zugriff erfolgt, also z.B. während der Ausführung eines PAUSE-Befehls. Dabei ist es nicht einmal das EEPROM selbst, sondern es sind die beiden Pullup-Widerstände an den Leitungen SCL und SDA, die zusammen ein ganzes Milliampere benötigen.

Was liegt daher näher als die Pause-Routine durch eine eigene, stromsparende Maschinenroutine zu ersetzen. Das EEPROM wird dazu zunächst vom Bus abgemeldet. Dabei gehen SCL und SDA in den High-Zustand. Dann folgen mehrere WAIT-Befehle. Jeder von ihnen setzt den Prozessor in den stromsparenden WAIT-Zustand und wird erst durch den 20-ms-Interrupt des internen Timers wieder aufgeweckt. Mit 10 WAIT-Befehlen erhält man also eine Wartezeit von ca. 200 ms. Danach wird das EEPROM mit den dafür vorgesehenen Betriebsroutinen wieder ordentlich angemeldet. Die Wirkung läßt sich leicht an einem Oszilloskop erkennen: SDA und SCL liegen für ca. 0,2 s hoch und werden nur noch dann aktiv, wenn das Programm sie wirklich braucht.

Wenn man für die Übersetzung TASM einsetzt, muss über den Parameter -x3 der spezielle Befehlssatz für die CMOS-Version des 6805 angegeben werden. Nur sie kennt den WAIT-Befehl und auch den weiter unten angewandten Befehl STOP. Der Aufruf lautet nun: Tasm.exe -05 -x3 WAIT.ASM -g2

```
;WAIT.ASM
;Wait-Modus
;Tasm.exe -05 -x3 Wait.asm -g2
;Das EEPROM wird abgemeldet
.org $101
bset 6,0 ;PA6=RTS hochschalten (-0,5mA)
jsr $08BB ;I2C_ReadLast, EEPROM abmelden
Wait
Wait
Wait
Wait
Wait
Wait
Wait
Wait
Wait
Wait
ldx #$0A0 ;Adresse EEROM, write
jsr $083C ;I2C_Start
ldx $066  ;AdrCounter Hi
jsr $0846 ;I2C_Write
ldx $067  ;AdrCounter Lo
jsr $0846 ;I2C_Write
ldx #$0A1 ;Adresse EEPROM, read
jsr $083C ;I2C_Start
rts
.end
```

Das folgende kleine Testprogramm ruft die neue Warte-Routine auf. Es wurde absichtlich nicht wirklich ein Port geschaltet, um nicht den zusätzlichen Strom eines Pullup-Widerstands an einem Ausgang zu messen:

```
'************************************
'
' C-Control/BASIC       WAIT.BAS
'
' Aufgabe:
'
' - Test des Stromsparmodus
' - Ersatz des Pause-Befehls
```

```
'
'************************************
' --- Definitionen --------------------

define Ausgang port[1]
define n word
' --- Programmoperationen -------------

     Slowmode 1
     for n = 1 to 1000
     Ausgang = 1 'Einschalten
     'pause 25 '0,5 s warten
     sys &h0101
     Ausgang = 1 'Ausschalten
     sys &h0101
     'pause 25 '0,5 s warten
     next n end

syscode "WAIT.obj"
```

Jetzt fragen Sie sicher: Und wieviel bringt der ganze Aufwand? Sie werden staunen: Die Control-Unit benötigt nur noch rund 1 mA!

Und man kann noch etwas tun und die Betriebsspannung etwas reduzieren. Das Datenblatt des Prozessors erlaubt Spannungen bis hinab zu 4 V. Aber testen Sie mal: Auch mit nur 3 V läuft noch alles prima. Sie können also eine 3,6V-Lithium-Batterie einsetzen. Bei 3,6 V wurde ein Strom von nur noch 0,7 mA gemessen. Für Langzeiteinsätze ist es günstig, drei Alkali-Batterien zu verwenden. Die frischen Batterien haben dann ca. 4,5 V. Sie dürfen sich im Betrieb bis weit unter 3,5 V entladen und werden gut ausgenutzt. Rechnet man mit einer Stromaufnahme von 1 mA und einer Kapazität von 2300 mAh, dann ergibt sich eine Betriebsdauer von über 3 Monaten. Wenn es drauf ankommt, kann man mit drei Alkali-Monozellen (12 Ah) weit über ein Jahr auskommen.

Aber was ist denn mit den von Datenblatt versprochenen 0,35 mA?
Auf der Suche nach der verlorenen Energie stößt man irgendwann im Datenblatt des 68HC05 auf den AD-Wandler und die serielle Schnittstelle des Prozessors. Also müssen diese Elemente auch noch abgeschaltet werden.

```
;WAIT2.ASM
;Wait-Modus
;Tasm.exe -05 -x3 Wait2.asm -g2
;Das EEPROM wird abgemeldet
```

```
;AD und RS232 abschalten

     .org $101
     bset 6,0 ;PA6=RTS hochschalten (-0,5mA)
     bclr 5,$09; Bit 5 (ADON) im ADSTAT-Register
     bclr 6,$09; Bit 5 (ADRC) im ADSTAT-Register
     bclr 2,$0F; Bit 5 (RE) im SCCR2-Register
     bclr 3,$0F; Bit 5 (TE) im SCCR2-Register
     jsr $08BB ;I2C_ReadLast, EEPROM abmelden
     Wait
     Wait
     Wait
     Wait
     Wait
     Wait
     Wait
     Wait
     Wait
     Wait
     ;bset 5,$09; Bit 5 (ADON) im ADSTAT-Register
     ;bset 6,$09; Bit 5 (ADRC) im ADSTAT-Register
     ;bset 2,$0F; Bit 5 (RE) im SCCR2-Register
     ;bset 3,$0F; Bit 5 (TE) im SCCR2-Register
     ldx #$0A0 ;Adresse EEROM, write
     jsr $083C ;I2C_Start
     ldx $066  ;AdrCounter Hi
     jsr $0846 ;I2C_Write
     ldx $067  ;AdrCounter Lo
     jsr $0846 ;I2C_Write
     ldx #$0A1 ;Adresse EEPROM, read
     jsr $083C ;I2C_Start
     rts

     .end
```

Das Programm WAIT2.ASM schaltet AD-Wandler und serielle Schnittstelle vor dem Eintritt in den Wait-Modus ab. Nach 200 ms sollten beide Elemente wieder eingeschaltet werden, sofern man sie benötigt. Das Einschalten ist im Listing angedeutet, aber nicht aktiviert. Die Control-Unit arbeitet also nun ohne AD-Wandler und RS232.

```
'************************************
'
' C-Control/BASIC       WAIT2.BAS
'
' Aufgabe:
```

```
'
' - Test des Stromsparmodus
' - Abschalten des AD und der RS232
'
'************************************
' --- Definitionen --------------------

define Ausgang port[1]
define n word
' --- Programmoperationen -------------

     Slowmode 1
#Loop
     sys &h0101
     goto Loop
     end

syscode "WAIT2.obj"
```

Bei einer Strommessung sieht man nun einen schwankenden Stromverbrauch, weil der Prozessor dauernd zwischen dem Wait-Modus und dem Normalmodus wechselt. Im Schnitt fließt nun ein Strom von 0,6 mA.

Das ist zwar leider immer noch etwas mehr als im reinen Wait-Modus zu erwarten wäre. Aber schließlich muss die Control-Unit fünf mal in der Sekunde aufwachen und ihre Arbeit tun. Dazu gehört auch die Verwaltung der Echtzeituhr, die an den Timer-Interrupt geknüpft ist. Der genaue Strombedarf richtet sich nach dem Verhältnis der aktiven Zeit zur Wait-Zeit, ist also von der jeweiligen Anwendung abhängig. Man könnte noch etwas mehr sparen, indem man den Prozessor länger im Wait-Modus hält. Im Maschinenprogramm müsste dann eine Zeitschleife ablaufen.

12.3 Der Stop-Modus

Sind Sie mit 0,6 mA immer noch nicht zufrieden? Dann bleibt nur noch eine ganz radikale Methode: Man kann den Prozessor durch einen STOP-Befehl völlig abschalten. Er verfällt damit wie Schneewittchen in einen Tiefschlaf. Nur der Kuss eines Prinzen kann dann noch helfen, d.h. ein Reset oder ein Hardware-

Interrupt. Nach einem Interrupt ist alles wie so als wäre nie etwas gewesen. So wie Schneewittchen merkt der Prozessor nichts von seinem Schlaf, und wenn es auch hundert Jahre waren. Das Programm wird ganz normal fortgeführt.
Mit dieser Methode kann die Stromaufnahme in der Wartezeit auf weit unter 1 µA gesenkt werden. Man benötigt aber eine externe Schaltung, die den Prozessor wieder aufweckt. Es kann sich z.B. um einen CMOS-Zähler oder um einen Uhrenbaustein handeln, der einen externen Interrupt auslöst.

```
; STOP.ASM
; Test von Stop und Interrupt, geht nur für CMOS-HC05, TASM mit:
; Tasm.exe -05 -x3 Stop.asm -g2
.org $101
        bset 6,0     ;PA6=RTS hochschalten (-0,5mA)
        jsr $08BB    ;I2C_ReadLast, EEPROM abmelden

        STOP         ;Stop Modus

        ldx #$0A0    ;Adresse EEROM, write
        jsr $083C    ;I2C_Start
        ldx $066     ;AdrCounter Hi
        jsr $0846    ;I2C_Write
        ldx $067     ;AdrCounter Lo
        jsr $0846    ;I2C_Write
        ldx #$0A1    ;Adresse EEPROM, read
        jsr $083C    ;I2C_Start
        rts          ;Kein RTI weil Interrupt vom System
abgefangen!
.end
```

Das aufrufende Basic-Programm:

```
'************************************
'
' C-Control/BASIC       STOP.BAS
'
' Aufgabe:
'
' - Test des STOP-Modus
' - Weiter über Interrupt
'
'************************************
' --- Definitionen --------------------

define Ausgang port[1]
define n word
```

```
' --- Programmoperationen -------------

     for n = 1 to 10
     Ausgang = 1 'Einschalten
     'pause 25 '0,5 s warten
     sys &h0101
     Ausgang = 1 'Ausschalten
     sys &h0101
     'pause 25 '0,5 s warten
     next n end

syscode "Stop.obj"
```

Messen Sie nun einmal den Stromverbrauch. Sofort nach dem Start wird der Stop-Modus aktiviert. Mit einem Interrupt steigt der Strom für einige Millisekunden auf übliche Werte, bis der nächste Aufruf des Maschinenprogramms den Stop-Modus erneut einschaltet. Solange sich der Prozessor aber im Stop-Modus befindet, braucht er unter 1 µA.

13 Remote-Start

Haben Sie sich auch schon einmal gewünscht, Sie könnten ein Programm in der Control-Unit durch ein Kommando vom PC aus starten? Dann brauchte man nicht mehr auf den Start-Taster zu drücken und könnte die Unit in einem Gehäuse verschwinden lassen oder über ein langes Kabel fernsteuern.
Leider fehlt der Control-Unit ein Kommando für den Programmstart. Aber keine Sorge: Man kann ein solches Kommando nachrüsten.

13.1 START.ASM

Die CC-Unit startet ein kleines Maschinenprogramm im internen EEPROM des Prozessors, das einen vollständigen Ersatz der ursprünglichen Haupt-Interpreterroutine DispatchHostCommand des Betriebssystems liefert. Zusätzlich

zu den nachgebildeten Originalkommandos gibt es nun ein zusätzliches Kommando ProgrammStart mit dem Code 15. Alle Kommandos rufen die entsprechenden Systemroutinen auf. Das neue Kommando startet die Routine Run, die im Originalzustand über die Starttaste erreicht wurde.

```
;Start.ASM
;Remote-Start über Kommando 15
;Ersatz für DispatchHostCommand
;Alle alten Kommandos bleiben erhalten

;Variablenspeicher im RAM
Kom   .equ $0A1 ;Kommando

       .org $101;EEPROM-Bereich
      jmp $190

       .org $190;Startadresse in C-Control

Loop    jsr $0CB5         ;RS232_GetByte, kommt in X
        stx Kom           ;empfangenes Kommando speichern
      lda #1              ;Kommando 1: ID-Byte senden
      cmp Kom             ;A=Kom?
      bne K2              ;nein: überspringen
      jsr $0AD6 ;SendInfo
      bra Loop

K2    lda #2              ;Kommando 2: Programm lesen
      cmp Kom             ;A=Kom?
      bne K3              ;nein: überspringen
      jsr $08BB       ;EndSequentialRead
      ldx #0
      stx $066  ;AdrCounter löschen
      stx $067
      lda #0
      jsr $0AEE ;Program
      bra Loop

K3    lda #3              ;Kommando 3: Programm-Dump
      cmp Kom             ;A=Kom?
      bne K8              ;nein: überspringen
      jsr $0B42 ;DumpProg
      bra Loop

K8    lda #8              ;Kommando 8: User-Prog laden
      cmp Kom             ;A=Kom?
      bne K9              ;nein: überspringen
      jsr $0B9F ;ProgramUserCode
```

```
       bra Loop

K9     lda #9              ;Kommando 8: Dump User-Prog
       cmp Kom             ;A=Kom?
       bne K13             ;nein: überspringen
       jsr $0BCA ;DumpUserCode
       bra Loop

K13    lda #13             ;Kommando 8: File-Dump
       cmp Kom             ;A=Kom?
       bne K14             ;nein: überspringen
       jsr $1B2D ;DumpFile
       bra Loop

K14    lda #14             ;Kommando 8: Set-Time
       cmp Kom             ;A=Kom?
       bne K15             ;nein: überspringen
       jsr $0BFF ;SetTime
       bra Loop

K15    lda #15             ;Kommando 15: Programm starten
       cmp Kom             ;A=Kom?
       bne K16             ;nein: überspringen
       jsr $08EE ;Run
       bra Loop

K16             bra Loop

.end

Assemblieren mit: Tasm.exe -05 -c -0f start.asm -g2
```

Das eigentliche Programm in START.ASM beginnt nicht wie üblich bei Adresse $101 sondern bei Adresse $190. Mit einem zusätzlichen Sprungbefehl kann man aber auch SYS$101 aufrufen. Das Programm wurde aus einem wichtigen Grund an die höchste mögliche Startadresse gelegt. Nun ist es nämlich möglich, ein zusätzliches Maschinenprogramm nachzuladen, solange es selbst nicht mehr als 140 Bytes hat. Bei der Verwendung von TASM muß der -c Parameter verwendet werden, damit ein durchgehendes Ausgabefile entsteht:

```
S11B0101CC01900F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F4A
S11B01190F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F62
S11B01310F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F4A
S11B01490F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F32
S11B01610F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F1A
```

```
S11B01790F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0FCD44
S11B01910CB5BFA1A601B1A12605CD0AD620F0A602B1A12610CD08BB90
S11B01A9AE00BF66BF67A600CD0AEE20DAA603B1A12605CD0B4220CFAD
S11B01C1A608B1A12605CD0B9F20C4A609B1A12605CD0BCA20B9A60D42
S11B01D9B1A12605CD1B2D20AEA60EB1A12605CD0BFF20A3A60FB1A1D8
S10C01F12605CD08EE20982096A5
S9030000FC
```

Die Datei Start.obj

```
'************************************
'
' C-Control/BASIC       START.BAS
'
' Aufgabe:
'
' - Remote-Start für CC
' - Zusatzkommando 15
'
'************************************
' --- Definitionen --------------------

' --- Programmoperationen -------------

sys &H0101            'Aufruf bei $0101 oder $190
end

syscode "Start.OBJ"   'Code laden
```

Das Programm Start.ASM wird nur einmal geladen und noch auf die alte Art mit der Starttaste gestartet. Danach kann ganz normal ein weiteres Programm geladen und mit dem neuen Kommando 15 gestartet werden. Die gesamte Programmausführung erfolgt nun als Unterprogramm des kleinen Maschinenprogramms Start.ASM. Eine End-Anweisung im Basic-Programm führt in das Assemblerprogramm zurück. Nun kann es erneut gestartet werden, oder man lädt ein neues Programm. Sie können nun also mehrere Programme nacheinander laden und starten. Ein Basic-Programm muß allerdings immer mit END beendet werden oder ein definiertes Ende haben, das z.B. durch ein empfangenes Byte über die RS232 erreicht werden kann. Sonst hängt das neue Betriebssystem, und man muss doch wieder an den Reset- und den Starttaster.

Jeder Unterprogrammaufruf benötigt Speicherplatz im Stack-Bereich, um die Rücksprungadresse zu sichern. Das System erlaubt daher nur eine begrenzte Verschachtelungstiefe. Damit geht nun eine Stufe bereits durch den Einsatz von Start.ASM verloren. Man kann übrigens bis zu 5 mal das Programm Start.BAS rekursiv durch sich selbst nachladen lassen, dann gibt es einen Absturz durch einen Stack-Überlauf.

Wenn Sie das START-Programm geladen und gestartet haben, führen Sie einen Test mit einem kleinen Programm aus, das nun über das erweiterte System geladen wird. Bedingung ist, dass es ohne den Einsatz der RESET-Taste beendet wird. Das folgende Beispielprogramm beendet sich nach fünf Schleifen selbst.

```
'************************************
'
' C-Control/BASIC       OUTPUT5.BAS
'
' Aufgabe:
'
' - Schalten des digitalen Ausgangs 1
' - Blinker: ein Impuls pro Sekunde
'
'************************************
' --- Definitionen --------------------

define Ausgang port[1]
define n Byte

' --- Programmoperationen -------------

For n= 1 to 5
  Ausgang = 1          'Einschalten
  pause 25             '0,5 s warten
  Ausgang = 0          'Ausschalten
  pause 25             '0,5 s warten
Next N                 'Schleife
end
```

Das Programm startet automatisch mit dem Laden, weil das Download-Programm selbst das Startkommando 15 sendet. Nach fünf Schleifen führt die END-Anweisung wieder in das eigene Assemblerprogramm zurück. Nun kann

man erneut ein Programm laden oder das selbe Programm noch einmal starten. Dazu muss man nur aus dem Terminalprogramm oder mit dem folgenden kleinen Hilfsprogramm das Startkommando absenden.

Das Programm CCstart.EXE sendet nur ein Byte: 15. Sie können also das nachgeladene Programm in der CC-Unit beliebig oft vom PC aus starten, ohne es neu zu laden.

```
Function Empfang () As Integer
  t = Timer
  Do
  Loop Until ((t + .2) < Timer) Or (Comm1.InBufferCount > 0)
  If Comm1.InBufferCount > 0 Then Empfang = Asc(Comm1.Input)
      Else Empfang = 0
End Function

Sub Form_Load ()
  Comm1.CommPort = 2
  Comm1.Settings = "9600,N,8,1"
  Comm1.InputLen = 1
  Comm1.PortOpen = True
  If Comm1.PortOpen = False Then
    Comm1.CommPort = 1
    Comm1.PortOpen = True
  End If
  MsgBox "Remote-Start", 0
  Sende 15              'Kommando Remote-Start
  End
End Sub

Sub Form_Unload (Cancel As Integer)
  Comm1.PortOpen = False
End Sub

Sub Sende (Zeichen As Integer)
  Comm1.Output = Chr$(Zeichen)
End Sub
```

14 Infothek: Daten, Software, Literatur

Software:

C-Control Software: CC-Basic für Windows und DOS, CC-Plus, Assembler, Do-It, ModulLab Alle Quelltexte der CD
Datenblätter und Schaltpläne Wichtige Informationen, meist im PDF-Format. C-Control Schaltpläne, Datenblatt des 68HC05, Datenblätter zu den auf der CD verwendeten ICs, Quelltexte des C-Control Betriebssystems.
Literaturhinweise Bücher und CDs zum Thema Elektronik und Mikrocontroller

14.1 Datenblätter

Datenblätter, Schaltpläne Informationen zum Ansehen und Ausdrucken

Schaltpläne der CC-Unit und Zubehör Mehr unter **http://www.c-control.de**	Schaltplan der C-Control Unit C-Control Starterboard C-Control Applicationboard Schaltplan der M-Unit M-Unit Programmieradapter LED-Anzeige Relaisplatine C-Control Station, Teil 1 C-Control Station, Teil 2
Datenblätter einiger wichtiger ICs	Prozessor 68HC05 Analogschalter 4051 Analogschalter 4066 Infrarot-Empfänger SFH605 Temperatursensor DS1820 EEPROM 24C65 EEPROM 24C256

14.2 Software

CC-BASIC mit Unterstützung des LED-Displays und zahlreichen Programmbeispielen kann direkt von der CD auf die Festplatte installiert werden. Das Setup-Programm installiert alle Programmversionen und Beispielprogramme. Wählen Sie den Setup-Typ "Benutzer" und aktivieren Sie die Komponente "Basic (komplett)". Sie erhalten die DOS-Software und die Version für Windows95. (Version 1.33, 6.4.98) siehe auch: http://www.c-control.de	CC-BASIC für DOS und Win95 installieren (SETUP.EXE)
Letztes Update für CC-Basic unter Windows. Nur die Datei CCBAS32.DLL muss in das Programmverzeichnis CCEW32D kopiert werden. Der Compiler erzeugt dann immer gerade Codelängen. (Update 11.11.99)	CCBAS32.DLL kopieren (CCBAS32.ZIP**)**
CC/Plus mit Unterstützung des LED-Displays und zahlreichen Programmbeispielen kann direkt von der CD auf die Festplatte installiert werden. Das Setup-Programm installiert alle Programmversionen und Beispielprogramme. Wählen Sie den Setup-Typ "Benutzer" und aktivieren Sie die Komponente "Plus".	CC-Plus installieren (SETUP.EXE)
Für die C-Control-Station gibt es eine spezielle CC/Plus-Version mit eigenen Control-Elementen. Sie kann direkt von der CD installiert werden. Das Setup-Programm installiert standardmäßig die Programmversionen zur C-Control-Station. Wählen Sie den Setup-Typ "Normal" oder wählen Sie "Benutzer" und aktivieren Sie die Komponente "Plus".	CC-Plus für Station installieren (SETUP.EXE)
Der Assembler AS5.EXE kann mit seiner Dokumentation direkt von der CD auf die Festplatte kopiert werden.	AS5 kopieren (AS5.ZIP)

TASM, Table Driven Assembler, Shareware von Thomas N. Anderson	TASM kopieren (TSM.ZIP)
Alle wesentlichen Quelltexte zum Betriebssystem des C-Control-Steuercomputers befinden sich im Archiv CCSYS.ZIP. Fortgeschrittene Anwender können die Quelltexte verwenden, um zusätzliche Informationen zum System und mögliche Aufrufe von Systemroutinen zu finden.	Verzeichnis CCSYS kopieren (CCSYS.ZIP)
Das Experimentier-Programm Do-It kann als 30-Minuten-Sharewareprogramm direkt von der CD gestartet werden. Wählen Sie den Simulationsmodus für erste Versuche ohne Interface. Zur Verwendung mit der Control-Unit muß CCIO.BAS geladen und gestartet sein. Bereiten Sie zunächst Ihre Control-Unit als Interface vor und setzen Sie den Autostart-Jumper. Alle Do-It-Beispielprogramme aus dem Buch befinden sich mit im Verzeichnis Do-It. Siehe auch: http://www.modul-bus.de	Do-It starten (DOITCC30.EXE)
Die Laborsoftware ModulLAB bietet erweiterte Möglichkeiten der Meßdatenverarbeitung und der Programmierung. Das Programm kann als Demoversion ohne Hardware-Ansteuerung direkt von der CD gestartet werden.	ModulLAB starten (MLCC.EXE)
Alle Programmbeispiele der CD in CC-Basic und Assembler	Programme kopieren (PROG.ZIP)
Alle VB3-Programmebeispiele der CD	VB3-Programme kopieren (VB3.ZIP)

14.3 Literaturhinweise

[1] B. Kainka, Messen, Steuern, Regeln mit dem C-Control/BASIC-System, Franzis-Verlag 1997
[2] B.Kainka/M.Förster, C-Control-Anwendungen, Franzis-Verlag 1998
[3] B. Kainka, Messen, Steuern und Regeln über die RS232-Schnittstelle,

Franzis-Verlag, 7. Auflage 1997
[4] B.Kainka, Wie mißt und steuert man mit dem PC, Franzis-Verlag 1997
[5] H.J.Berndt/B.Kainka, Messen, Steuern und Regeln mit Word und Excel, Franzis-Verlag 1998
[6] B. Kainka, Erfolgreich Messen, Steuern, Regeln mit Mikrocontrollern, Franzis-Verlag, 2. Auflage 1998
[7] B. Kainka, Einführung in die Elektronik - eine interaktive Lern-CD, Franzis-Verlag 1998
[8] B. Kainka, Elektronik-Start mit dem PC - Eine interaktive Einführung in die Steuerungs- und Regelungstechnik mit einer Experimentierplatine zum Anschluß an den PC, Franzis-Verlag 1999

Franzis-Buchkatalog Herbst 1999

Weitere Informationen finden Sie im Internet unter:
http://home.t-online.de/home/B.Kainka

Weitere Neuheiten und Bestellungen unter
http://www.franzis.de