8052 AH-BASIC

COLLABORATORS

	TITLE : 8052 AH-BASIC	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 31, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

8052 AH-BASIC

1.1 8052 AH-BASIC Entwicklungs-Paket

```
                        8052 AH-BASIC Entwicklungs-Paket

             von Andreas Heinrich

        Erstveröffentlicht am  1. August  1994
        1. Update          am 21. Februar 1995


Was ist Neu ?
       Inhaltsverzeichnis


Copyright und Vertrieb

Warnung ! Bitte lesen.

Einleitung
 und
Was ist 8052 AH-BASIC ?

Die Hardware / Prozessorboard

Die Hardware / IN_OUT-Board

Die Hardware / Meßwertgeber

Bauanleitung

8052 AH-Basic-Terminal 2.0

BASIC - Befehle

Demoprogramme

Tips zur EPROMprogrammierung

Adresse des Autors
```

1.2 Was ist Neu ?

Das Terminal-Programm wurde komplett neu in Blitz Basic 2 ↩
geschrieben.
Eine einfache Umsetzung von Amos war nicht möglich, da sich diese
Sprachen doch zu sehr unterscheiden.
Herausgekommen ist nun ein etwas systemkonformeres Programm, daß
auch unter Kick/WB 1.3 laufen sollte.
Außerdem wurde ich gebeten, daß das Amos-Listing zur Version 1.1
veröffentlicht werden soll. Leider kann ich das nicht machen, da
ich mit der Procedurorgie NCommand das Programm geschrieben habe
und dieses Lizensware ist und somit einem Copyright unterliegt.
Im Verzeichnis Old_Stuff ist die Version 1.0, die Freeware war und
nach belieben verändert werden kann. Das Listing ist kommentiert
und es dürfte keine Probleme beim umschreiben und erweitern geben.

Auf der IN_OUT-Platine wurden zwei zusätzliche Schaltungen integriert:

1. Eine WatchDog-Schaltung

2. Eine Zusatzschaltung die ein Rechtecksignal erzeugt.
   (Notwendig zum decodieren des DCF-Funkuhr-Signals.)

   Dazu folgendes:

   Eigentlich sollte eine RTC (Real-Time-Clock) mit integriert werden.
   Leider mußten die Uhrenchips mit denen ich Versuche gemacht habe,
   alle mit einer Assembler-Routine angesprochen werden, die die
   ganze einfache Programmierung in Basic hinfällig machen würde.
   Außerdem hätte das komplette Platinenlayout neu entworfen werden
   müssen. Darum habe ich mich für die DCF-Funkuhr entschlossen, die
   immer die genaue Uhrzeit ins System holt und sich auch beim
   Neustart von alleine setzt.

Weiteres zu den Hardwareergänzungen:
Die Hardware / IN_OUT-Board

1.3 Copyright und Vertrieb

Die Bauanleitungen, Schaltpläne, Platinenlayouts und Bauteillisten
wurden aus Standardaplikationen zusammengetragen, verbessert,
optimiert und erweitert.
Soweit mir bekannt ist lagen für die Standardaplikationen keine
Copyrights vor.

Die Firma Intel hat das Copyright auf den MCS-52-Basic-Interpreter.

Deshalb müssen die Copyrightbestimmungen beim kopieren des Interpreters in ein EPROM beachtet werden.

Dieses Entwicklungspaket ist Shareware und kann frei kopiert und weitergegeben werden. Sollte das Paket in einer FD-Serie aufgenommen werden, darf der Preis pro Einzeldisk nicht höher als 10.- DM sein.

Bei Gefallen und Benutzung des Pakets ist allerdings eine Shareware-gebühr von 25.- DM an mich zu zahlen. Registrierte Anwender bekommen dann von mir das Platinenlayout und einen Bestückungsplan zugesandt. Außerdem biete ich diesen Anwendern einmalig die fertig geätzten Platinen (Prozessorplatine/IN_OUT-Platine) zum Preis von je 15.- DM an. Die Meßwertgeberplatinen in SMD oder Standard kosten je 7,50 DM. Auf den Platinen sind die Meßwertgeber 8 mal vorhanden. (Alle Platinen müssen allerdings noch gebohrt bzw. gesägt werden.)

Durch die Zahlung der Shareware ist der gewerbliche Nachbau in Serie trotzdem nicht erlaubt.

Siehe auch:
Adresse des Autors
.

1.4 Einleitung

Was beim C 64 der Userport war, ist beim Amiga der .?.?.?...

Der Amiga hat zwar verschiedene Schnittstellen zur Außenwelt, die sich vom Rechnertyp mehr oder weniger unterscheiden, aber meistens lassen sich diese nur schwer programmieren und / oder der Hardwareaufwand wird zu aufwendig.

Wie die Steuerung über den Parallelport geht, veröffentlichte ich mit dem Schaltungsaufbau des IN_OUT_BOARDS. Leider hat der Amiga (meistens) nur einen Parallelport und dieser ist dann auch noch mit dem Drucker belegt. Es bleibt einem dann also nichts anderes übrig, als ewig hinten am Rechner die Stecker umzutauschen oder einen Dataswitch zubenutzen.

Bei dieser anderen Lösung mit dem 8052 AH-BASIC-Prozessor wird nun der Seriellport benutzt. Sofern man kein Modem oder etwas anderes am Serialport hat, kann nun die Verbindung "dauerhaft" hergestellt werden. Da der Serial-port auch weniger empfindlich ist, lassen sich nun Verbindungen von etlichen Metern in zweiadrigen abgeschirmten Kabel herstellen.
Mit RS 232 Treiber sind sogar mehrere hundert Meter möglich.
Auch ist die Hardware mit anderen Rechnern zu betreiben. Nur ein serieller Port (RS 232) und eine Terminalsoftware muß vorhanden sein.

Da mit dieser Hardware ein intelligentes System zum Regeln und Steuern zu realisieren ist, sind bestimmt auch die abenteuerlichsten Sachen zu verwirklichen. Zwar wird nun der Hardwareaufwand ein wenig aufwendiger, aber dafür hat man ein eigenständiges System, wobei der Amiga nicht ständig eingeschaltet sein braucht.

Die
Demoprogramme
verdeutlichen diesen Vorteil.

Hier nun ein Beispiel anhand einer Wetterstation:

Das externe System sammelt zu verschiedenen Tageszeiten die Meßdaten von Gebern, die die Temperatur und Helligkeit messen. Einmal am Tag oder in der Woche (kommt auf die Größe der Daten an) holt man sich über die serielle Schnittstelle die gesammelten Daten und wertet diese aus. Oder das externe System schaltet selbst den Amiga an und das Programm, das in der User-Startup eingefügt wurde wird ausgeführt.

Oder noch ein Beispiel:

Steuerung der Vorlauftemperatur bei einem Heizkessel. Aus der Außentemperatur wird die optimale Vorlauftemperatur ausgerechnet. Die Temperatur wird durch Ein und Ausschalten mit Relais des elektrischen Magnetventils geregelt. Zuerst wird die optimale Temperaturkurve auf dem Amiga ausgerechnet und in ein Programm an das externe System übertragen. Hier kann nun die Software über einen längeren Zeitraum getestet werden und wenn alles zur Zufriedenheit läuft in das EPROM gebrannt werden. Danach kann die serielle Verbindung unterbrochen werden und das System arbeitet völlig selbstständig.

Diese beiden Beispiele sollen eigentlich nur Denkanstöße für eigene Entwicklungen geben und die Möglichkeiten aufzeigen.

Siehe auch:
Was ist 8052 AH-BASIC ?

1.5 Was ist 8052 AH-BASIC ?

Der 8052 AH-BASIC ist ein Basicprogrammierbarer Mikrokontroller ↩
der dem
Mikrokontroller des 8052 von Intel ähnlich ist. Der Unterschied liegt darin, daß dieser in einem 8 KB großen ROM einen Basicinterpreter integriert hat. Mußten vorher Mikrokontroller mühsam mit Hilfe aufwendiger und kostspieliger Entwicklungssysteme in Maschinensprache programmiert werden, kann man nun dieses in einem speziellen Dialekt der Hochsprache Basic tun. Das Basic-Programm wird in einem externen RAM oder EPROM gespeichert und startet auf Wunsch automatisch beim Einschalten der Betriebsspannung bzw. eines Resets. Auch ist es möglich den Interpreter auszulesen und diesen in das EPROM zu brennen. So kann statt des 8052 AH-BASIC der wesentlich billigere 80C32 verwendet werden, der stromsparender und höher taktbar ist (max. 32 MHZ).
Wie schon erwähnt, wird der Kontroller in einer speziell für diesen Zweck zugeschnittenen Variante der Sprache Basic programmiert; der Anwender braucht sich nicht in die Abgründe der Maschinensprache zu begeben, um ihn als programmierbare elektronische Steuerung einzusetzen.
Ich kann in dieser Anleitung eigentlich nur das Wichtigste vermitteln. Wer mehr über dieses Basic erfahren möchte und mehr aus dem Prozessor herausholen will, empfehle ich das Buch 8052 AH-BASIC aus dem Elektor-Verlag. (ISBN 3-921608-72-4) Preis zur Zeit 48.- DM.

Siehe auch:
Basic-Befehle

1.6 Warnung

Achtung!!!Achtung!!!Achtung!!!Achtung!!!Achtung!!!Achtung!!!Achtung

Obwohl der Aufbau der Hardware eigentlich keine Schwierigkeiten bereiten sollte, muß ich allerdings noch dieses schreiben:

Der Aufbau und die Inbetriebnahme der Hard und Software erfolgt auf eigene Gefahr und für etwaige Schäden wird keine Haftung übernommen.

(Das soll aber keine Abschreckung sein.)

1.7 Die Hardware / Prozessorboard

Um den einfachen Nachbau zu verwirklichen, wird das Prozessorboard ↩
auf
einer einseitigen Europakarte 100*160 mm aufgebaut. Der Prozessor läßt sich, wie schon erwähnt in Basic programmieren. Aber auch Assembler oder beides ist möglich.
Das System ist mit 32 KB-EPROM und 32 KB-RAM aufgebaut.
Der RAM-Bereich erstreckt sich von Adresse 0000 Hex - 7FFF Hex.
Der Speicherinhalt des RAM's kann durch eine Back-Up-Batterie gesichert werden. Allerdings ist dann eine Programmieranweisung im EPROM notwendig. Denn bei einem Neueinschalten bzw. Reset macht der Interpreter einen Selbsttest, wobei im RAM stehende Programme überschrieben werden.
Siehe auch unter Basicbefehle >
PROG4
<.
Der ROM (EPROM) - Bereich ist in zwei 16 KB große Blöcke geteilt.
Im oberen Bereich von 8000 Hex - BFFF Hex können eigene Basicprogramme gebrannt werden. Im unteren Bereich von 0000 Hex - 1FFF Hex kann der Interpreter kopiert werden. Allerdings ist aufgrund des Hardwareaufbaus zum programmieren ein Adaptersockel notwendig.
Siehe auch unter:
Tips zur EPROMprogrammierung
Die EPROM-Programmierhardware ist auf der Platine integriert.
Die Schaltung braucht eine Versorgungsspannung von mindestens 16 Volt, damit eine Programmierspannung von 12,5 Volt erzeugt werden kann.
Wird der EPROMer nicht benötigt kann die Spannung zwischen 8 - 12 Volt liegen. Die Stromaufnahme mit dem Original-Prozessor ist ca. 400 mA.
Mit der CMOS-Version 80C32 ungefähr die Hälfte. Der Original-Prozessor kann bis ca. 15 MHZ getaktet werden. Aus Sicherheitsgründen (Überhitzung) sollten nur 12 MHZ Quarze verwendet werden. Anders sieht es bei der CMOS Variante aus; hier sind 24 MHZ und mehr möglich. Allerdings schaffen nur einige Examplare 32 MHZ. Um Schwingungen zu vermeiden, muß ab 20 MHZ eine

Festinduktivität von 1,5 $\mathrm{\mu}$H an den Quarz gelötet werden.
Diese Festinduktivität sollte allerdings nur beim 80C32 verwendet werden.
(Beim Original-Prozessor schwingt sonst der Quarz nicht an.)
Der Prozessor hat einen bidirektionalen Port -> Anschlüsse P1.0-P1.7.
Die Ausgänge davon sind mit Schutzwiderständen von 2,7 K bestückt. Diese
begrenzen bei Programmierfehlern den Strom und der Prozessor ist somit
geschützt. Über diesen Port kann die Ein und Ausgabe erfolgen z.B. Relais,
Leuchtdioden angesteuert oder eine Tastatur / Schalter abgefragt werden.
Mit diesen einem Ein-Ausgabeport läßt sich schon viel anfangen. Wenn das
aber nicht ausreicht, kann mit 18 pol.IC-Steckverbindern und Flachbandkabel
die Verbindung zur IN_OUT-Platine hergestellt werden.

Siehe auch:
Hardware / IN_OUT-Board

1.8 Die Hardware / IN_OUT-Board

Auch hier wurde um den einfachen Nachbau zu verwirklichen, das IN_OUT-Board
auf einer einseitigen Europakarte 100*160 mm aufgebaut.
Auf dieser befinden sich zusätzlich 4 8-Bit-Ausgabe-Ports, 2 8-Bit-Eingabe-
Ports, ein achtfacher Multiplex - Frequenzeingang und eine Hilfsschaltung
für ein LCD-Display. Theoretisch läßt sich das LCD-Display auch direkt
über den Datenbus ansteuern. Leider gab es manchmal Timing-Probleme,
so daß ich für die Ansteuerung zwei Ausgabe-Ports heranziehe.
Das kommt auch der Betriebssicherheit zu gute.
Mit einem anderen Ausgabe-Port wird ein 8-Fach Multiplexer angesteuert.
An diesen können Meßwertgeber für Temperatur / Lichtmessung usw.
angeschlossen werden.

Weiteres dazu bei den
Demoprogrammen
und unter
Die Hardware / Meßwertgeber
Werden nicht alle Ports gebraucht, oder kann z.B. auf das LCD- Display ect.
verzichtet werden, kann das entsprechende IC / Bauteil entfallen.
Der IN / OUT Bereich erstreckt sich von Adresse C000 Hex - FFFF Hex.
In Anspruch genommen durch das IN_OUT-Board werden nur die Adressen
C000 Hex - C005 Hex. Also bleibt für eigene Erweiterungen genügend freier
Adressraum.

Neues:

Auf der Platine wurde noch der
WatchDog
und eine Zusatzschaltung für ein
625 Hertz Rechtecksignal integriert. Dieses 625 Hertz Signal ist zum
decodieren des
DCF
- Funkuhr-Signals notwendig. Das DCF-Funkuhr-Empfangs-
modul gibt ein Signal mit den Impulslängen von 100 ms bzw. 200 ms ab, das
an den Interrupt Eingang INT1 gelegt wird. Entweder auf der Prozessor-
oder der IN_OUT-Platine. Das 625 Hertz Signal kommt an den Frequenzein-

gang 8 oder direkt an T1, wenn gar keine IN_OUT-Platine angeschlossen ist. Geeicht auf 625 Hertz wird mit einem Frequenzzähler oder dem Programm Multiplex. Hier dauert allerdings ein kompletter Durchlauf 8 Sekunden, so daß man immer eine kleine Pause zwischen einem Abgleich machen muß.

Ein passendes Empfangsmodul kostet ca. 30 DM und ist zum Beispiel bei Conrad electronic oder Westfalia Technica erhältlich.

1.9 WatchDog - Schaltung

Die WatchDog - Schaltung:

Die Resetschaltung, im Fachjargon auch WatchDog genannt, wirkt auf den Resetanschluß des Prozessors. Solange am Resetanschluß 0 Volt anliegt, ist die CPU inaktiv und führt keine Programmbefehle aus. Wechselt das Potential von 0 auf 5 Volt, so startet der Prozessor den Programmlauf ab der Adresse 0000H. Durch kurze (oder auch längere) negative Impulse kann man somit hardwaremäßig den Prozessor zwingen, an der Adresse 0000H die Programmausführung neu zu beginnen.
Die Resetschaltung dient als Einschalt - Reset bei Spannungs-einbrüchen oder -ausfällen, wozu auch das Ein- und Ausschalten des Gerätes zählt. Sobald die Versorgungsspannung einen kritischen Wert unterschreitet (oder beim Einschalten die notwendige Spannung noch nicht erreicht ist), wird die Resetleitung auf 0 Volt gelegt.
Zum zweiten wird der Prozessor durch die Resetschaltung auf einwand-freie Programmbearbeitung überwacht. Mittels regelmäßiger Impulse, die - gleichgerichtet - einen Kondensator aufladen, muß der Prozessor anzeigen, das er normal arbeitet. Bei fehlenden Impulsen entlädt sich der Kondensator und die Resetschaltung gibt einen negativen Impuls ab, worauf der Prozessor zur Adresse 0000H springt und das Programm an definierter Stelle neu anläuft. Diese zweite Funktion ist der eigent-liche "Watch-Dog", wogegen das Erste ein Einschalt-Reset ist.
Besonders einfach und elegant läßt sich diese Resetschaltung mit dem IC TL 7705 verwirklichen. Die entsprechende Schaltung ist in der "Bildergalerie" zu sehen. Das IC hat intern bereits eine Spannungs-überwachung die unterhalb 4,55 Volt den Reset-Ausgang nullt. Der Resin-Eingang hat seine Umschaltschwelle bei 0,8 Volt, unterhalb dieser Spannung wird ein Reset ausgegeben. Durch die Rückkopplung mittels dem 100 K-Widerstand von Resin + nach Resin - wird erreicht, daß sich die Schaltung kurzfristig selbst zurücksetzt und somit dem Prozessor Zeit gibt, das Programm zu starten, denn bei einem Reset wird Reset + positiv und lädt damit den 4,7 $\mathrm{\mu}$F-Kondensator auf, welcher wiederum bewirkt, daß die Resetleitung auf "High" schaltet. Somit kann der Prozessor sein Programm starten und gibt in regelmäßigen Abständen eine Rechteckspannung an den 0,1 $\mathrm{\mu}$F-Kondensator. Diese wechselnde Spannung wird mittels der Diode gleichgerichtet und lädt so den Kondensator am Resin - auf. Dabei werden ca. 2 Volt erreicht. Der 750 k Widerstand, sowie das Trimmpoti 2,5 M sorgen für eine stetige Entladung so daß der Prozessor darauf achten muß, daß der Kondensator stets auf-geladen ist. Zur Ansteuerung eignet sich z.B. D7 (Adresse 0C001H) auf der IN_OUT-Platine. Mit diesem Bit wird dann das LCD-Display und der WatchDog gleichzeitig bedient.

Programmierbeispiel:
Man verschaltet den WatchDog wie in der Bidergalerie angegeben auf der IN_OUT-Platine und lädt das Programm Uhrzeit_3.bas in den Interpreter. Der Kippschalter sollte natürlich geöffnet sein, sonst gibt es dauernd einen Reset. Danach startet man das Programm und dreht an dem Trimmpoti bis die LED nicht mehr blinkt. Jetzt kann der Kippschalter geschlossen werden und der WatchDog ist hiermit scharf. Das kann durch Programmabbruch gleich ausprobiert werden. Also Ctrl-C hält das laufende Programm an, der Impuls bleibt aus und der erwartete Reset wird ausgeführt. Natürlich ist nun das gerade unterbrochene Programm aus dem RAM verschwunden und demzufolge wird es in regelmäßigen Abständen einen Reset geben. Somit ist der erste Test schon bestanden.
Der nächste Schritt wäre es sein Programm ins Eprom zu brennen "PROG" und dann mit "PROG2" eine Kennung ins Eprom zu legen, daß das erste im Eprom stehende Programm gestartet werden soll. Gleichzeitig wird mit dieser Kennung die Baudrateninformation abgelegt, so daß beim Neustart kein Leerzeichen gesendet werden muß. Wird nun mit scharfen WatchDog ein Reset erzwungen (Programmabbruch = Ctrl-C), startet das ins Eprom gebrannte Programm. Eventuell muß das Trimmpoti noch etwas nachreguliert werden, da der Selbsttest des Interpreters ca. 2 - 3 Sekunden dauert.

Zurück zur
Hardware / IN_OUT-Board

1.10 Die DCF77-Codierung

In der Physikalischen-Technischen Bundesanstalt (PTB) in ↩ Braunschweig
wird unsere Gesetzliche Zeit "hergestellt". Die dort erzeugte Zeitskala wird codiert und vom Langwellensender DCF77 per Funksignal ausgestrahlt. Dieser Sender gehört der Deutschen Bundespost und wird von der PTB nur gemietet. Der zentrale Standort in Mainflingen bei Frankfurt am Main und die gewählte Frequenz von 77,5 KHZ im Längstwellenbereich garantieren eine Reichweite von gut 1500 km.
Das Schema der codierten Zeitübertragung ist ganz simpel:
Die Trägerfrequenz von 77,5 KHZ wird kontinuierlich im Dauerstrich ausgestrahlt und in jeder Sekunde wird die Amplitude kurzzeitig auf 25 % des Normalwertes abgesenkt. Die dadurch entstehenden Sekundenmarken können entweder kurz (0,1 Sek.) oder lang sein (0,2 Sek.); die kurzen entsprechen dem digitalen LOW-, die langen einem HIGH-Bit.
Durch "Einsammeln" der Bits und geeignetes Decodieren kommt man an die verschlüsselte Datum- und Zeitinformation wieder heran.
Um in den endlosen Strom der ankommenden Bits Ordnung zu bringen, braucht man einen Bezugspunkt zur Synchronisation. Bei DCF77 erhält man den dadurch, daß pro Minute eine Sekundenmarke ausbleibt, dort also keine Trägerabsenkung erfolgt.
Mit der auf diese Lücke folgenden Marke beginnt jeweils eine neue Minute. In diesem Augenblick setzt die Auswertesoftware ihren Sekundenzähler auf Null. Bei jeder folgenden Sekundenmarke wird dieser Zähler um Eins erhöht. Der Zählerstand läßt also jederzeit erkennen, um welchen Sekundenpuls es sich gerade handelt.
Auf diese Weise hat man pro Minute 59 Sekundenmarken zur Informations-

übertragung zur Verfügung. Diese Bitfolge von 59 zusammengehörenden Bits nennt man auch Telegramm, in diesem Fall das Zeittelegramm, weil darin Uhrzeit, Datum und Wochentag übermittelt werden.
Folgende Informationen sind darin enthalten:
Minuten (Einer und Zehner)
Stunden (Einer und Zehner)
Kalendertage (Einer und Zehner)
Monat (Einer und Zehner)
Jahr (Einer und Zehner)
Wochentag als Zahl (Montag=1)
Übertragen wird immer die Uhrzeit der nächstfolgenden Minute, bei deren Beginn (Sekundenmarke Null) sie in eine Anzeige überschrieben wird. Sekunden brauchen nicht mit im Telegramm übertragen zu werden; sie ergeben sich stets aus dem Stand des Sekundenzählers.
Jede der elf Ziffern des Telegramms (Minuten-Einer bis Jahres-Zehner) wird im BCD-Format dargestellt. Das ist ein Vier-Bit-Binärcode, bei dem jedes aufsteigende Bit die doppelte Wertigkeit des vorhergehenden hat (8-4-2-1-Code). Um damit beispielsweise die Ziffer 9 darzustellen, müssen das erste Bit (mit der Wertigkeit 1) und das vierte (mit der Wertigkeit 8) gesetzt sein (auf HIGH).
Es sind aber nicht für jede Stelle vier Bits erforderlich. Der Tages-Zehner z.B. kann maximal eine 3 sein, so daß man für dessen Darstellung nur zwei Bits braucht (mit den Wertigkeiten 1 und 2).

Hier nun eine Tabelle des Codierungsschemas:

Nr.	Bedeutung	Wert
0	Anfang	X
1	....	
14	nicht belegt	
Sonderbits:		
15	R	X
16	A1	X
17	Z1	X
18	Z2	X
19	A2	X
20	Start	X
Minuten:		
21	Einer	1
22		2
23		4
24		8
25	Zehner	1
26		2
27		4
28	Prüfbit 1	X
Stunden:		
29	Einer	1
30		2
31		4
32		8
33	Zehner	1
34		2
35	Prüfbit 2	X
Kalendertag:		
36	Einer	1
37		2

```
38              4
39              8
40   Zehner     1
41              2
Wochentag:
42   Nummer     1
43              2
44              4
Monat:
45   Einer      1
46              2
47              4
48              8
49   Zehner     1
Jahr:
50   Einer      1
51              2
52              4
53              8
54   Zehner     1
55              2
56              4
57              8
58   Prüfbit 3  X
59   fehlt
```

Der "richtige" Informationsgehalt beginnt bei Sekunde Nr.21 mit dem Minuten-Einer. Davor liegen 14 Bits ohne Informationsinhalt (Nr.1 bis 14) sowie weitere sechs Sonderbits (Nr.15 bis 20;s.u.). Die drei Prüfbits dienen einer Auswerte-Software zur Kontrolle. Sie ergänzen die davor übertragenen Datenbits auf gerade Parität, d.h. einschließlich Prüfbit muß im jeweiligen Abschnitt des Telegramms eine gerade Anzahl von Einsen (HIGH-Bits) vorhanden sein.
Auch die Sonderbits dienen u.a. dazu, die Auswerte-Software zu unterstützen und vor Fehlschlüssen zu bewahren. Bit 15 (R) ist HIGH, wenn die Reserve-Antenne des Senders aktiv ist.
Die Ankündigungsbits (Nr.16 und 19) leiten die Zeitumstellung ein (A1) bzw. kündigen das Einfügen einer Schaltsekunde an (A2). Mit den Zeitzonen-Bits (Nr.17 und 18) wird die Sommerzeit gekennzeichnet (Z1=HIGH während MESZ). Bit Nr.20 ist immer ein "langes", das den Beginn der eigentlichen Zeitcodierung markiert.
Wie lange braucht nun die Software, bis sie nach dem Start die DCF77-Zeit anzeigt ?
Mindestens eine volle Minute und zwar von der 59. Sekunde der einen bis zur 59. Sekunde der nächsten Minute. Wegen der fehlenden Synchronisation zählt eine beim Start gerade laufende "angebrochene" Minute nicht.
Die Auswertesoftware wertet aber zwei aufeinanderfolgende Zeittelegramme aus, die sich nur um eine Minute Differenz unterscheiden dürfen. Erst dann wird der korrekte Empfang der Zeitinformation bestätigt. Also nach spätestens 3 Minuten sollte die Uhr synchron laufen, vorrausgesetzt der Empfang ist einwandfrei.

Zurück zur
Hardware / IN_OUT-Board

1.11 Die Hardware / Meßwertgeber

Um nicht teure AD-Wandler verwenden zu müssen, werden die ↩ Meßwertgeber
mit den Standard-Timer-IC's 555 in CMOS-Ausführung aufgebaut.
Diese Meßwertgeber arbeiten genügend präzise und sind kaum störanfällig.
Kabellängen von 20 Metern oder mehr sind möglich, so daß Fehlmessungen
eigentlich nicht zu erwarten sind.

Zur Schaltung:

Die Schaltung setzt die von einem NTC-Widerstand gemessene Temperatur in
ein digitales Signal um. Der Widerstandswert des NTC fällt mit steigender
Umgebungstemperatur. Er steuert die Frequenz eines Oszillators, der mit
der CMOS-Version des Timers 555 aufgebaut ist. Die Schaltung ist so
dimensioniert, daß die Oszillatorfrequenz bei 20 \textdegree{}C ↩ Umgebungstemperatur
etwa 2000 Hertz beträgt und sich bei steigender Temperatur erhöht.
Der nichtlinare Zusammenhang zwischen Temperatur und Oszillatorfrequenz
ist kein Problem, da der 8052er-Mikrokontroller mit Hilfe einer einfachen
Tabelle eine rechnerische Linearisierung vornehmen kann. Ausgehend von
drei gemessenen Temperaturpunkten mit zugehöriger Oszillatorfrequenz
können die Zwischenwerte gefunden werden. Also die ganze Eichung der
Meßwertgeber wird per Software erledigt.
Statt des NTC-Widerstand kann auch ein Fotowiderstand genommen werden
und dann die Helligkeit gemessen werden. Aber auch Fühler für Luftfeuchtig-
keit oder Luftdruck sind bestimmt möglich. Diese Sensoren allerdings
teuer und die Eichung wird auch nicht einfach sein.

Angeschlossen wird der Meßwertgeber an den externen Timereingang T1 bzw.
die Meßwertgeber auf der IN_OUT-Platine am Multiplexereingang.
Das Multiplex-IC wird durch den Ausgangsport 4 angesteuert und es können
dann die einzelnen Eingänge abgefragt werden. Der 8052er-Mikrokontroller
fungiert mit Hilfe der Software als Frequenzzähler und die gewonnenen
Daten können entweder gleich verwertet oder diese von der Amiga-Software
mit benutzt werden.

Siehe auch bei den
Demoprogrammen
.

1.12 Bauanleitung

In der Bildergalerie sind die Platinenlayouts des Prozessorboards, der
IN_OUT-Platine und der Meßwertgeber als IFF-Bilder abgelegt. Aus Platz-
gründen sind die Leiterbahnen, Bauteile und Bauteilbezeichnungen überein-
ander gelegt. Die Schaltpläne des WatchDogs und des 625 Hertz Rechteck-
Signals sind für die registrierten Anwender gedacht, damit sie diese bei
Bedarf optional auf einer Punktrasterplatine aufbauen können.
Auch sind bei dieser Veröffentlichung die Bauteillisten mit dabei.
Es gab nämlich Unklarheiten mit der Erhältlichkeit der Bauteilen und deren
Kosten. So kann sich jeder überzeugen, daß nur Standardbauteile verwendet

werden.

Ich hoffe das trotzdem die Sharewaregebühr an mich bezahlt, denn der Aufbau ohne die Layouts oder der Platinen wäre bestimmt sehr mühsam.

Aber nun zum Aufbau des Prozessorboards:

Wichtig ist, daß die Drahtbrücken zuerst eingelötet werden, da einige unter IC-Fassungen liegen. Br.1 und Br.2 sollten allerdings ganz zum Schluß mit isolierten Schaltdraht gesetzt werden. Danach können die IC-Fassungen, Widerstände, Kondensatoren, Elkos, Dioden und Transistoren eingelötet werden. IC 4 (LM 317) wird auf den passenden Kühlkörper geschraubt und dieser auf der Platine mit Schrauben befestigt.
Erst dann wird das IC verlötet. Nachdem man noch einmal den Aufbau kontrolliert hat und auch die Leiterbahnseite auf evtl. Kurzschlüsse untersucht hat, kann man Spannung auf die Platine geben. Mit einem Meßgerät wird kontrolliert, ob der Spannungsregler korrekt funktioniert und an den IC-Fassungen 5 Volt Versorgungsspannung anliegt. Ist dieser erste Test erfolgreich verlaufen, wird die Versorgungsspannung unterbrochen und die IC's können eingesetzt werden. Das EPROM braucht noch nicht eingesetzt werden. Vom seriellen Anschluß der Platine (TxD, RxD, Minus) kann nun die Verbindung zum Rechner hergestellt werden.
Achtung: TxD und RxD werden gekreuzt = Nullmodemschaltung. Siehe auch in der Bildergalerie.
Nun kann das Terminal-Programm gestartet und die Platine unter Spannung gesetzt werden. Nachdem man die Leertaste betätigt hat, sollte sich der Prozessor melden.

Das müßte so aussehen:

```
*MCS-51(tm)BASIC V1.1*
READY
>
```

Das Prozessorboard ist damit betriebsbereit.

Der Aufbau der IN_OUT-Platine ist im Prinzip gleich:

Zuerst sollten die Drahtbrücken eingesetzt und verlötet werden. Da auf der IN_OUT-Platine die Drahtbrücken teilweise sehr eng beieinander liegen, sollte auf Kurzschlüsse geachtet werden. Dann können die Fassungen, die passiven und dann die aktiven Bauteile eingesetzt werden. Die Verbindung mit dem Prozessorboard erfolgt durch zwei Flachbandkabel die 18 polige IC-Stecker haben. (Siehe Bildergalerie)
Zum Testen der Ports können die Hilfsmittel und die Demosoftware benutzt werden.

Zum Aufbau der Meßwertgeber ist nicht viel zu sagen:

Wem der Aufbau in SMD-Technik nicht liegt, kann die "größere" konventionelle Variante nehmen.
Angeschlossen wird der Geber an T1 auf dem Prozessorboard, bzw. die Geber

auf der IN_OUT-Platine.

1.13 8052 AH-BASIC Terminal 2.0

```
         8052 AH-Basic-Terminal Version 2.0 vom 21.Februar 1995

    Achtung! Dieses Programm ist Shareware und darf nur komplett mit dem
                Entwicklungspaket weitergegeben werden.
```

Verbunden wird das Prozessorboard über die serielle Schnittstelle mit einer sogenannten Nullmodem-Schaltung. Über die integrierte serielle Schnittstelle kann der Mikrokontroller vollduplex (d.h. gleichzeitig sendend und empfangend) Daten mit dem Rechner austauschen. Nach dem Systemstart oder Reset arbeitet die serielle Schnittstelle im asynchronen Modus mit folgenden Datenformat: 1 Startbit, 8 Datenbit, 1 Stopbit, kein Paritätsbit. Außerdem erwartet der Mikrokontroller ein Leerzeichen (Spacetaste). Eingestellter Defaultwert des Terminalprogramms ist 9600 Baud. In der Praxis hat sich dieser Wert als ausreichend erwiesen.

Zum Programm selbst:

Im 8052 AH-Basic ist ein einfacher Zeileneditor integriert, mit dem die Basic-Programme eingegeben werden können.
Gibt man z.B. ein verkehrtes Zeichen ein, ist dieses mit der Del-Taste zu löschen.
Wurde hingegen schon die Eingabetaste betätigt, ist das Löschen der Zeile nur noch mit Eingabe der Zeilennummer möglich.
(Der Interpreter ist zeilenorientiert, wie in guten alten Zeiten).

Hier ein immer wieder beliebtes Programmbeispiel:

```
10 REM  Ein Super-Beispiel
20 FOR X=1 TO 10
30 PRINT "HELLO WORLD"
40 NEXT X
```

Wird dieses schöne Programm durch die Eingabe von RUN gestartet, erscheint der allzeit beliebte Text 10 mal.

Möchte man nun die 1. Zeile löschen, reicht es aus, 10 einzugeben und die Eingabetaste zu drücken.

Gestartet wird das Programm wie bereits erwähnt durch RUN.

Unterbrechen lassen sich die Programme mit Ctrl-C.

Das Terminalprogramm hat Defaultwerte mit dem der Serialport geöffnet wird, sowie eine Funktionstastenbelegung die wie folgt aussieht:

```
F 1  LIST     0 - 99
F 2  LIST   100 - 999
F 3  LIST  1000 - 9999
F 4  LIST 10000 - 49000
F 5  LIST     0 - 65535
F 6  RUN
```

```
F 7  CONT
F 8  RAM
F 9  ROM
F10  XFER
```

Die LIST-Befehle auf den Funktionstasten F1 - F5 bedürfen eigentlich keiner Erklärung. Nur das F5 das gesamte Programm listet (0-65535) und das 65535 die maximale Zeilenzahl ist die der Interpreter verwalten kann.

Taste F 6 startet das Programm.

Taste F 7 führt das Programm an der Stelle fort, wo es mit Ctrl-C abgebrochen wurde.

Taste F 8 schaltet ins RAM.

Taste F 9 schaltet ins ROM.

Taste F10 kopiert dem ROM-Inhalt ins RAM.

Natürlich lassen sich alle diese Funktionstasten auch anders belegen.

Über die Menüleiste Einstellungen/F-Tasten oder der Tastenkombination Rechts-Amiga-F kommt man in das Konfigurationsfenster der F-Tasten. Hier können bis zu 64 Zeichen lange Befehlszeilen eingegeben werden.

Nach verlassen des Konfigurationsfensters und wählen des Menüpunktes Einstellungen/Konf. sichern bzw. Rechts-Amiga-S werden die Einstellungen dauerhaft im Konfigurationsfile 8052AH-BASIC.config festgehalten. Dieses File sollte im Verzeichnis SYS:S/ stehen, da die eigenen Einstellungen jedesmal beim Start des Terminalprogramms hieraus geholt werden. Ist allerdings das File nicht vorhanden, werden die Defaultwerte gesetzt. Hat man nun einige Einstellungen verändert und möchte das Terminalprogramm zurücksetzten, kann durch den Menüpunkt Einstellungen/Konf. laden (bzw. Rechts-Amiga-L) das vorher abgespeicherte Konfig.-File geladen werden. Aber auch mehrere verschiedene Konfigurationsfiles lassen sich anlegen. Nur sollte ein abgewandelter Name für das File verwendet werden. Wie z.B. 8052AH-BASIC2.config oder Meine.config.

Hat man nun alles verstellt und möchte die Grundeinstellung wiederhaben, hilft der Menüpunkt Einstellungen/Defaultwerte (bzw. Rechts-Amiga-D) weiter.

Weitere Einstellungen können mit Menüpunkt Einstellungen/Terminal bzw. Rechts-Amiga-T gemacht werden.

Hier die Defaultwerte für die serielle Schnittstelle:

```
Baud          9600
Buffer       16384
Delay           15
Device      serial.device
Unit             0
Timeout        250
Taskpriorität   -1
```

Baud:

Hier lassen sich vier verschiedene Baudraten (2400, 4800, 9600 oder 19200) einstellen.

Buffer:
Hier kann die Buffergröße geändert werden. (4096, 8192, 16384, 32768)

Delay:
Hier läßt sich die Verzögerungszeit zu den verschiedenen Baudraten einstellen. (0-63)
Dazu folgendes:
Die Übertragung des Listings an den Mikrokontroller muß in einem bestimmten Format ablaufen. Jedesmal wenn die Eingabe einer Programmzeile in das 8052-AH-Basic-System mit einem Carriage Return (0D Hex) abgeschlossen wird, benötigt der Interpreter gewisse Zeit, um die Zeile intern in Kurzzeichen (Token) umzusetzen und zu speichern. Erst danach darf die nächste Programmzeile eingegeben werden; anderenfalls gehen die ersten Zeichen der nächsten Zeile verloren. Um dem Interpreter die notwendige Zeit zu lassen, wird das Basic-Prompt ">" ausgewertet. (Software-Handshake). Danach muß noch ein gewisser Zeitraum gewartet werden, damit der Eingabepuffer wirklich leer ist und die nächste Zeile gesendet werden kann. Die Defaultwerte sind meine Erfahrungswerte auf einem A 4000/030. Eventuell müssen diese auf einem Rechner mit höheren Prozessor höher bzw. einen Standard-Amiga niedriger gesetzt werden.

Device & Unit:
Hier ist das serial.device mit dem Unit 0 vorgegeben.
Hat man nun eine Multiseriell-Karte im Amiga, kann das Terminalprogramm darauf umgeleitet werden.

Timeout:
Tritt bei der Übertragung ein Fehler auf, erscheint ein Info-Requester und die Übertragung wird beendet. Defaultwert ist 250 = 5 Sekunden.

Taskpriorität:
Voreingestellt ist -1. Eventuell ist hier überhaupt keine Änderung notwendig. Sollte allerdings das Multitasking gestört werden, kann ein Wert zwischen -5 bis 5 gesetzt werden. Der Filetransfer erfolgt sicherheitshalber immer mit der Priorität 3.

Eine noch nicht erwähnte Funktion im Menüpunkt Einstellungen ist Umlaute (Tastenkombination Rechts-Amiga-U).
Der Basic-Interpreter schluckt nämlich keine Umlaute wie ö ä ü oder das ß. Ist die Funktion aktiviert und wird z.B. ä eingegeben, erfolgt eine Ausgabe als ae. Damit auch Leute die kein deutsches Keyboard haben nicht durcheinander kommen ist die Wandelung abschaltbar.

Alle diese Einstellungen werden auch im vorher erwähnten Konfigurationsfile abgespeichert, so daß man seine einmal eingestellte Konfiguration gleich beim Start des Terminalprogramms geladen hat.

Noch nicht erwähnte Menüpunkte:

Info/Ende:

Info (bzw. Rechts-Amiga-I) bringt ein Infofenster auf den Screen mit den aktuellen Daten von:
Datum, Uhrzeit, Uhrzeit beim Start des Programms, Prozessor, Execversion,

AGA oder kein AGA, PAL oder NTSC, freies Chip-Mem, freies Fast-Mem, größter freier Speicherblock und Taskpriorität.

Quit (bzw. Rechts-Amiga-Q) beendet das Programm.

Filetransfer:

File übertragen (bzw. Rechts-Amiga-Ü) überträgt das Basiclisting in den Interpreter.

Listing holen (bzw. Rechts-Amiga-H) holt das Listing und sichert es.

Sonstiges:

CLS (bzw. Rechts-Amiga-C) löscht das Ausgabefenster (CLS).
F-Taste 1 - 10 = Funktionstasten.

Zum Terminalprogramm noch folgendes:

Das Carriage Return an Texten im Amiga-Format muß noch angehängt, bzw. beim Sichern von Listings herausgefiltert werden.
Das übernimmt alles dieses Terminal-Programm.
Das Schreiben von längeren Programmen ist einfacher, wenn man dieses in einem Textprogramm macht, einschließlich Zeilennummern und dieses ASCII-Textfile anschließend in das 8052-System lädt. Auch ist es möglich, im Hintergrund das Textprogramm laufen zu lassen und mit der Tastenkombination Links-Amiga-M hin und herzuschalten.

****

Hier noch einmal die Funktionsübersicht der Tasten:

```
F1      LIST     0 - 99
F2      LIST   100 - 999
F3      LIST  1000 - 9999
F4      LIST 10000 - 49000
F5      LIST     0 - 65535
F6      RUN
F7      CONT
F8      RAM
F9      ROM
F10     XFER

Del     Zeichen löschen
Ctrl-C  Programmabbruch

Rechts-Amiga-I    Infofenster
Rechts-Amiga-Q    Programm beenden
Rechts-Amiga-Ü    File übertragen
Rechts-Amiga-H    Listing holen
Rechts-Amiga-T    Terminal
Rechts-Amiga-F    Funktionstasten
Rechts-Amiga-U    Umlaute
Rechts-Amiga-S    Konfiguration sichern
```

```
Rechts-Amiga-L    Konfiguration laden
Rechts-Amiga-D    Defaults nehmen
Rechts-Amiga-C    CLS (ClearScreen)

Links-Amiga-M Zur Workbench


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


Und die Menüleiste:

Info/Ende       File-Transfer       Einstellungen       Sonstiges

Info            File übertragen     Terminal            CLS
Quit            Listing holen       F-Tasten            F-Taste  1
                                    Umlaute             F-Taste  2
                                    Konfig. sichern     F-Taste  3
                                    Konfig. laden       F-Taste  4
                                    Defaultwerte        F-Taste  5
                                                        F-Taste  6
                                                        F-Taste  7
                                                        F-Taste  8
                                                        F-Taste  9
                                                        F-Taste 10


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


Achtung:

Das Serial.Device sollte im Verzeichnis Devs stehen.
Gerade bei Rechner die mit Diskette gebootet werden, wird oftmals das
Serial.Device aus Platzgründen weggelassen.


Und noch das übliche zum Schluß:

Für etwaige Schäden, die wohl nicht vorkommen dürften bzw. Datenverlust
durch Bedienungsfehler wird keine Haftung übernommen.
Die Benutzung geschieht auf eigene Gefahr.
```

1.14 BASIC-Befehle

```
                Die folgende Übersicht über Kommandos, Anweisungen und Operatoren ↩
                   wurde
dem Original-Datenblatt von Intel entnommen.

Wer ausführlichere Angaben und die deutschen Erklärungen haben möchte,
verweise ich auf das in
                 Was ist 8052 AH-BASIC ?
                 aufgeführte Buch.
```

```
RUN
DO
UI0
RCAP2

CONT
UNTIL
UO1
T2CON

LIST
WHILE
UO0
TCON

LIST#
END
ST@
TMOD

LIST@
FOR-TO
LD@
TIME

NEW
NEXT
IDLE
TIMER0
```

```
NULL

GOSUB

RROM

TIMER1

RAM

RETURN

+

TIMER2

ROM

GOTO

/

PI

XFER

ON GOTO

**

PROG

ON GOSUB

*

PROG1

IF-THEN

-

PROG2

INPUT

.AND.

PROG3
```

```
LET

.OR.

PROG4

ONERR

.XOR.

PROG5

ONTIME

ABS()

PROG6

ONEX1

NOT()

FPROG

PRINT

INT()

FPROG1

PRINT#

SGN()

FPROG2

PH0.

SQR()

FPROG3

PH1.

RND

FPROG4

PH0.#
```

LOG()

FPROG5

PH1.#

EXP()

FPROG6

PRINT@

SIN()

BAUD

PH0.@

COS()

CALL

PH1.@

TAN()

CLEAR

PGM

ATN()

CLEARS

PUSH

CBY()

CLEARI

POP

DBY()

CLOCK1

PWM

XBY()

CLOCK0

REM

GET

DATA

RETI

IE

READ

STOP

IP

RESTORE

STRING

PORT1

DIM

UI1

PCON

1.15 Basic-Befehl RUN

COMMAND FUNCTION EXAMPLE(↩
S)

RUN Execute a program RUN

Zurück zu den
Basic-Befehlen.

1.16 Basic-Befehl CONT

COMMAND	FUNCTION	EXAMPLE(↩ S)
CONT	CONTinue after a STOP or Control-C	CONT

Zurück zu den Basic-Befehlen.

1.17 Basic-Befehl LIST

COMMAND	FUNCTION	EXAMPLE(↩ S)
LIST	LIST program to the console device	LIST LIST 10-50

Zurück zu den Basic-Befehlen.

1.18 Basic-Befehl LIST#

COMMAND	FUNCTION	EXAMPLE(↩ S)
LIST#	LIST program to serial printer	LIST# LIST# 50

Zurück zu den Basic-Befehlen.

1.19 Basic-Befehl LIST@

COMMAND	FUNCTION	EXAMPLE(↩ S)
LIST@	LIST program to user driver	LIST@ LIST@ 50

Zurück zu den
Basic-Befehlen.

1.20 Basic-Befehl NEW

COMMAND	FUNCTION	EXAMPLE(↩ S)
NEW	erase the program stored in RAM	NEW

Zurück zu den
Basic-Befehlen.

1.21 Basic-Befehl NULL

COMMAND	FUNCTION	EXAMPLE(↩ S)
NULL	set NULL count after carriage return-line feed	NULL NULL 4

Zurück zu den
Basic-Befehlen.

1.22 Basic-Befehl RAM

COMMAND	FUNCTION	EXAMPLE(↩ S)

RAM evoke RAM mode, current program in RAM READ/WRITE memory RAM

Zurück zu den Basic-Befehlen.

1.23 Basic-Befehl ROM

COMMAND FUNCTION EXAMPLE(↩ S)

ROM evoke ROM mode, current program in ROM ROM/EPROM memory ROM 3

Zurück zu den Basic-Befehlen.

1.24 Basic-Befehl XFER

COMMAND FUNCTION EXAMPLE(↩ S)

XFER transfer a program from ROM/EPROM to RAM XFER

Zurück zu den Basic-Befehlen.

1.25 Basic-Befehl PROG

COMMAND FUNCTION EXAMPLE(↩ S)

COMMAND	FUNCTION	EXAMPLE(S)
PROG	save the current program in EPROM	PROG

Zurück zu den Basic-Befehlen.

1.26 Basic-Befehl PROG1

COMMAND	FUNCTION	EXAMPLE(S)
PROG1	save baud rate information in EPROM	PROG1

Zurück zu den Basic-Befehlen.

1.27 Basic-Befehl PROG2

COMMAND	FUNCTION	EXAMPLE(S)
PROG2	save baud rate information in EPROM and execute program after RESET	PROG2

Zurück zu den Basic-Befehlen.

1.28 Basic-Befehl PROG3

COMMAND	FUNCTION	EXAMPLE(S)
PROG3	save baud rate and MTOP information in EPROM	PROG3

Zurück zu den
Basic-Befehlen.

1.29 Basic-Befehl PROG4

COMMAND	FUNCTION	EXAMPLE(↩ S)
PROG4	save baud rate and MTOP information in EPROM and execute program after RESET	PROG4

Zurück zu den
Basic-Befehlen.

1.30 Basic-Befehl PROG5

COMMAND	FUNCTION	EXAMPLE(↩ S)
PROG5	same as PROG4 except that external RAM is not cleared on RESET or power up if external RAM contains a 0A5H in location 5EH	PROG5

Zurück zu den
Basic-Befehlen.

1.31 Basic-Befehl PROG6

COMMAND	FUNCTION	EXAMPLE(↩ S)
PROG6	same as PROG6 except that external code location 4039H is CALLED after RESET	PROG6

Zurück zu den
Basic-Befehlen.

1.32 Basic-Befehl FPROG

COMMAND	FUNCTION	EXAMPLE(↩ S)
FPROG	save the current program in EPROM using the INTELligent algorithm	FPROG

Zurück zu den
Basic-Befehlen.

1.33 Basic-Befehl FPROG1

COMMAND	FUNCTION	EXAMPLE(↩ S)
FPROG1	save baud rate information in EPROM using the INTELligent algorithm	FPROG1

Zurück zu den
Basic-Befehlen.

1.34 Basic-Befehl FPROG2

COMMAND	FUNCTION	EXAMPLE(↩ S)
FPROG2	save baud rate information in EPROM and execute program after RESET, use INTELligent algorithm	FPROG2

Zurück zu den
Basic-Befehlen.

1.35 Basic-Befehl FPROG3

COMMAND	FUNCTION	EXAMPLE(↩ S)
FPROG3	same as PROG3, except INTELligent programming algorithm is used	FPROG3

Zurück zu den
Basic-Befehlen.

1.36 Basic-Befehl FPROG4

COMMAND	FUNCTION	EXAMPLE(↩ S)
FPROG4	same as PROG4, except INTELligent programming algorithm is used	FPROG4

Zurück zu den
Basic-Befehlen.

1.37 Basic-Befehl FPROG5

COMMAND	FUNCTION	EXAMPLE(↩ S)
FPROG5	same as PROG5, except INTELligent programming algorithm is used	FPROG5

Zurück zu den
Basic-Befehlen.

1.38 Basic-Befehl FPROG6

COMMAND	FUNCTION	EXAMPLE(↩ S)
FPROG6	same as PROG6, except INTELligent programming algorithm is used	FPROG6

Zurück zu den
Basic-Befehlen.

1.39 Basic-Befehl BAUD

COMMAND	FUNCTION	EXAMPLE(↩ S)
BAUD	set baud rate for line printer port	BAUD 1200

Zurück zu den
Basic-Befehlen.

1.40 Basic-Befehl CALL

COMMAND	FUNCTION	EXAMPLE(↩ S)
CALL	CALL assembly language program	CALL 9000H

Zurück zu den
Basic-Befehlen.

1.41 Basic-Befehl CLEAR

COMMAND	FUNCTION	EXAMPLE(↩ S)
CLEAR	CLEAR variables, interrupts and Strings	CLEAR

Zurück zu den Basic-Befehlen.

1.42 Basic-Befehl CLEARS

COMMAND	FUNCTION	EXAMPLE(↩ S)
CLEARS	CLEARS Stacks	CLEARS

Zurück zu den Basic-Befehlen.

1.43 Basic-Befehl CLEARI

COMMAND	FUNCTION	EXAMPLE(↩ S)
CLEARI	CLEAR interrupts	CLEARI

Zurück zu den Basic-Befehlen.

1.44 Basic-Befehl CLOCK1

```
                    COMMAND      FUNCTION                                  EXAMPLE( ↩
                       S)

CLOCK1        enable REAL TIME CLOCK                 CLOCK1


Zurück zu den
            Basic-Befehlen.
```

1.45 Basic-Befehl CLOCK0

```
                    COMMAND      FUNCTION                                  EXAMPLE( ↩
                       S)

CLOCK0        disable REAL TIME CLOCK                CLOCK0


Zurück zu den
            Basic-Befehlen.
```

1.46 Basic-Befehl DATA

```
                    COMMAND      FUNCTION                                  EXAMPLE( ↩
                       S)

DATA          DATA to be read by READ statement      DATA 100


Zurück zu den
            Basic-Befehlen.
```

1.47 Basic-Befehl READ

```
                    COMMAND      FUNCTION                                  EXAMPLE( ↩
                       S)
```

READ	READ data in DATA statement	READ A

Zurück zu den
Basic-Befehlen.

1.48 Basic-Befehl RESTORE

COMMAND	FUNCTION	EXAMPLE(↩ S)
RESTORE	RESTORE READ pointer	RESTORE

Zurück zu den
Basic-Befehlen.

1.49 Basic-Befehl DIM

COMMAND	FUNCTION	EXAMPLE(↩ S)
DIM	allocate memory for arrayed variables	DIM A(20)

Zurück zu den
Basic-Befehlen.

1.50 Basic-Befehl DO

COMMAND	FUNCTION	EXAMPLE(↩ S)
DO	set up loop for WHILE or UNTIL	DO

Zurück zu den
Basic-Befehlen.

1.51 Basic-Befehl UNTIL

COMMAND	FUNCTION	EXAMPLE(↩ S)
UNTIL	test DO loop condition (loop if false)	UNTIL A=10

Zurück zu den
Basic-Befehlen.

1.52 Basic-Befehl WHILE

COMMAND	FUNCTION	EXAMPLE(↩ S)
WHILE	test DO loop condition (loop if true)	WHILE A=B

Zurück zu den
Basic-Befehlen.

1.53 Basic-Befehl END

COMMAND	FUNCTION	EXAMPLE(↩ S)
END	terminate program execution	END

Zurück zu den
Basic-Befehlen.

1.54 Basic-Befehl FOR-TO

COMMAND	FUNCTION	EXAMPLE(↩ S)
FOR-TO	set up FOR-NEXT loop	FOR A=1 TO 5

Zurück zu den Basic-Befehlen.

1.55 Basic-Befehl NEXT

COMMAND	FUNCTION	EXAMPLE(↩ S)
NEXT	test FOR-NEXT loop condition	NEXT A

Zurück zu den Basic-Befehlen.

1.56 Basic-Befehl GOSUB

COMMAND	FUNCTION	EXAMPLE(↩ S)
GOSUB	execute subroutine	GOSUB 1000

Zurück zu den Basic-Befehlen.

1.57 Basic-Befehl RETURN

COMMAND	FUNCTION	EXAMPLE(↩ S)

RETURN RETURN from subroutine RETURN

Zurück zu den
Basic-Befehlen.

1.58 Basic-Befehl GOTO

COMMAND FUNCTION EXAMPLE(↩
S)

GOTO GOTO program line number GOTO 500

Zurück zu den
Basic-Befehlen.

1.59 Basic-Befehl ON GOTO

COMMAND FUNCTION EXAMPLE(↩
S)

ON GOTO conditional GOTO ON A GOTO 5,20

Zurück zu den
Basic-Befehlen.

1.60 Basic-Befehl ON GOSUB

COMMAND FUNCTION EXAMPLE(↩
S)

ON GOSUB conditional GOSUB ON A GOSUB 2,6

Zurück zu den
Basic-Befehlen.

1.61 Basic-Befehl IF-THEN-ELSE

COMMAND	FUNCTION	EXAMPLE(↩ S)
IF-THEN ELSE	conditional test	IF A<B THEN A=0

Zurück zu den
Basic-Befehlen.

1.62 Basic-Befehl INPUT

COMMAND	FUNCTION	EXAMPLE(↩ S)
INPUT	INPUT a string or variable	INPUT A

Zurück zu den
Basic-Befehlen.

1.63 Basic-Befehl LET

COMMAND	FUNCTION	EXAMPLE(↩ S)
LET	assign a variable or string a value (LET is optional)	LET A=10

Zurück zu den

Basic-Befehlen.

1.64 Basic-Befehl ONERR

COMMAND	FUNCTION	EXAMPLE(↩ S)
ONERR	ONERRor GOTO line number	ONERR 1000

Zurück zu den Basic-Befehlen.

1.65 Basic-Befehl ONTIME

COMMAND	FUNCTION	EXAMPLE(↩ S)
ONTIME	generate an interrupt when TIME is equal to or greater than ONTIME argument-line number is after comma	ONTIME 10,1000

Zurück zu den Basic-Befehlen.

1.66 Basic-Befehl ONEX1

COMMAND	FUNCTION	EXAMPLE(↩ S)
ONEX1	GOSUB to line number following ONEX1 when INT1 pin is pulled low	ONEX1 1000

Zurück zu den Basic-Befehlen.

1.67 Basic-Befehl PRINT

COMMAND	FUNCTION	EXAMPLE(↩ S)
PRINT	PRINT variables, strings or literals P. is shorthand for PRINT	PRINT A

Zurück zu den
Basic-Befehlen.

1.68 Basic-Befehl PRINT#

COMMAND	FUNCTION	EXAMPLE(↩ S)
PRINT#	PRINT to software serial port	PRINT# A

Zurück zu den
Basic-Befehlen.

1.69 Basic-Befehl PH0.

COMMAND	FUNCTION	EXAMPLE(↩ S)
PH0.	PRINT HEX mode with zero suppression	PH0. A

Zurück zu den
Basic-Befehlen.

1.70 Basic-Befehl PH1.

COMMAND	FUNCTION	EXAMPLE(↩ S)
PH1.	PRINT HEX mode with no zero suppression	PH1. A

Zurück zu den Basic-Befehlen.

1.71 Basic-Befehl PH0.#

COMMAND	FUNCTION	EXAMPLE(↩ S)
PH0.#	PH0. to line printer	PH0.# A

Zurück zu den Basic-Befehlen.

1.72 Basic-Befehl PH1.#

COMMAND	FUNCTION	EXAMPLE(↩ S)
PH1.#	PH1.# to line printer	PH1.# A

Zurück zu den Basic-Befehlen.

1.73 Basic-Befehl PRINT@

COMMAND	FUNCTION	EXAMPLE(↩ S)

COMMAND	FUNCTION	EXAMPLE(S)
PRINT@	PRINT to user defined driver	PRINT@ 5*5

Zurück zu den Basic-Befehlen.

1.74 Basic-Befehl PH0.@

COMMAND	FUNCTION	EXAMPLE(S)
PH0.@	PH0. to user defined driver	PH0.@ XBY(5EH)

Zurück zu den Basic-Befehlen.

1.75 Basic-Befehl PH1.@

COMMAND	FUNCTION	EXAMPLE(S)
PH1.@	PH1. to user defined driver	PH1.@ A

Zurück zu den Basic-Befehlen.

1.76 Basic-Befehl PGM

COMMAND	FUNCTION	EXAMPLE(S)
PGM	Programm an EPROM	PGM

Zurück zu den
Basic-Befehlen.

1.77 Basic-Befehl PUSH

COMMAND	FUNCTION	EXAMPLE(↩ S)
PUSH	PUSH expressions on argument stack	PUSH 10,A

Zurück zu den
Basic-Befehlen.

1.78 Basic-Befehl POP

COMMAND	FUNCTION	EXAMPLE(↩ S)
POP	POP argument stack to variables	POP A,B,C

Zurück zu den
Basic-Befehlen.

1.79 Basic-Befehl PWM

COMMAND	FUNCTION	EXAMPLE(↩ S)
PWM	PULSE WIDTH MODULATION	PWM 50,50,100

Zurück zu den
Basic-Befehlen.

1.80 Basic-Befehl REM

```
                    COMMAND      FUNCTION                                EXAMPLE( ↩
                       S)

REM           REMark                                REM DONE
```

Zurück zu den
Basic-Befehlen.

1.81 Basic-Befehl RETI

```
                    COMMAND      FUNCTION                                EXAMPLE( ↩
                       S)

RETI          RETurn from interrupt                 RETI
```

Zurück zu den
Basic-Befehlen.

1.82 Basic-Befehl STOP

```
                    COMMAND      FUNCTION                                EXAMPLE( ↩
                       S)

STOP          break program execution               STOP
```

Zurück zu den
Basic-Befehlen.

1.83 Basic-Befehl STRING

```
                    COMMAND      FUNCTION                                EXAMPLE( ↩
                       S)
```

STRING	allocate memory for STRINGs	STRING 50,10

Zurück zu den
Basic-Befehlen.

1.84 Basic-Befehl UI1

COMMAND	FUNCTION	EXAMPLE(↩ S)
UI1	evoke User console Input routine	UI1

Zurück zu den
Basic-Befehlen.

1.85 Basic-Befehl UI0

COMMAND	FUNCTION	EXAMPLE(↩ S)
UI0	evoke BASIC console Input routine	UI0

Zurück zu den
Basic-Befehlen.

1.86 Basic-Befehl UO1

COMMAND	FUNCTION	EXAMPLE(↩ S)
UO1	evoke User console Output routine	UO1

Zurück zu den
Basic-Befehlen.

1.87 Basic-Befehl UO0

COMMAND	FUNCTION	EXAMPLE(↩ S)
UO0	evoke BASIC console Output routine	UO0

Zurück zu den
Basic-Befehlen.

1.88 Basic-Befehl ST@

COMMAND	FUNCTION	EXAMPLE(↩ S)
ST@	store top of stack at user specified location	ST@ 1000H ST@ A

Zurück zu den
Basic-Befehlen.

1.89 Basic-Befehl LD@

COMMAND	FUNCTION	EXAMPLE(↩ S)
LD@	load top of stack from user specified location	LD@ 1000H LD@ A

Zurück zu den

Basic-Befehlen.

1.90 Basic-Befehl IDLE

COMMAND	FUNCTION	EXAMPLE(S)
IDLE	wait for interrupt	IDLE

Zurück zu den Basic-Befehlen.

1.91 Basic-Befehl RROM

COMMAND	FUNCTION	EXAMPLE(S)
RROM	run a program in EP(ROM)	RROM 3

Zurück zu den Basic-Befehlen.

1.92 Basic-Befehl +

COMMAND	FUNCTION	EXAMPLE(S)
+	ADDITION	1+1

Zurück zu den Basic-Befehlen.

1.93 Basic-Befehl /

COMMAND	FUNCTION	EXAMPLE(↩ S)
/	DIVISION	10/2

Zurück zu den Basic-Befehlen.

1.94 Basic-Befehl **

COMMAND	FUNCTION	EXAMPLE(↩ S)
**	EXPONENTATION	2**4

Zurück zu den Basic-Befehlen.

1.95 Basic-Befehl *

COMMAND	FUNCTION	EXAMPLE(↩ S)
*	MULTIPLICATION	4*4

Zurück zu den Basic-Befehlen.

1.96 Basic-Befehl -

COMMAND	FUNCTION	EXAMPLE(↩ S)

-	SUBTRACTION	8-4

Zurück zu den
Basic-Befehlen.

1.97 Basic-Befehl .AND.

COMMAND	FUNCTION	EXAMPLE(↩ S)
.AND.	LOGICAL AND	10.AND.5

Zurück zu den
Basic-Befehlen.

1.98 Basic-Befehl .OR.

COMMAND	FUNCTION	EXAMPLE(↩ S)
.OR.	LOGICAL OR	2.OR.1

Zurück zu den
Basic-Befehlen.

1.99 Basic-Befehl .XOR.

COMMAND	FUNCTION	EXAMPLE(↩ S)
.XOR.	LOGICAL EXCLUSIVE OR	3.XOR.2

Zurück zu den
Basic-Befehlen.

1.100 Basic-Befehl ABS()

COMMAND	FUNCTION	EXAMPLE(↩ S)
ABS()	ABSOLUTE VALUE	ABS(-3)

Zurück zu den
Basic-Befehlen.

1.101 Basic-Befehl NOT()

COMMAND	FUNCTION	EXAMPLE(↩ S)
NOT()	ONES COMPLEMENT	NOT(0)

Zurück zu den
Basic-Befehlen.

1.102 Basic-Befehl INT()

COMMAND	FUNCTION	EXAMPLE(↩ S)
INT()	INTEGER	INT(3.2)

Zurück zu den
Basic-Befehlen.

1.103 Basic-Befehl SGN()

COMMAND	FUNCTION	EXAMPLE(↩ S)
SGN()	SIGN	SGN(-5)

Zurück zu den Basic-Befehlen.

1.104 Basic-Befehl SQR()

COMMAND	FUNCTION	EXAMPLE(↩ S)
SQR()	SQUARE ROOT	SQR(100)

Zurück zu den Basic-Befehlen.

1.105 Basic-Befehl RND

COMMAND	FUNCTION	EXAMPLE(↩ S)
RND	RANDOM NUMBER	RND

Zurück zu den Basic-Befehlen.

1.106 Basic-Befehl LOG()

COMMAND	FUNCTION	EXAMPLE(↩ S)

LOG() NATURAL LOG LOG(10)

Zurück zu den
Basic-Befehlen.

1.107 Basic-Befehl EXP()

COMMAND FUNCTION EXAMPLE(↩
S)

EXP() "e"(2.7182818) TO THE X EXP(10)

Zurück zu den
Basic-Befehlen.

1.108 Basic-Befehl SIN()

COMMAND FUNCTION EXAMPLE(↩
S)

SIN() RETURNS THE SINE OF ARGUMENT SIN(3.14)

Zurück zu den
Basic-Befehlen.

1.109 Basic-Befehl COS()

COMMAND FUNCTION EXAMPLE(↩
S)

COS() RETURNS THE COSINE OF ARGUMENT COS(0)

Zurück zu den
Basic-Befehlen.

1.110 Basic-Befehl TAN()

COMMAND	FUNCTION	EXAMPLE(S)
TAN()	RETURNS THE TANGENT OF ARGUMENT	TAN(.707)

Zurück zu den
Basic-Befehlen.

1.111 Basic-Befehl ATN()

COMMAND	FUNCTION	EXAMPLE(S)
ATN()	RETURNS ARCTANGENT OF ARGUMENT	ATN(1)

Zurück zu den
Basic-Befehlen.

1.112 Basic-Befehl CBY()

COMMAND	FUNCTION	EXAMPLE(S)
CBY()	READ PROGRAM MEMORY	P.CBY(4000)

Zurück zu den
Basic-Befehlen.

1.113 Basic-Befehl DBY()

COMMAND	FUNCTION	EXAMPLE(↩ S)
DBY()	READ/ASSIGN INTERNAL DATA MEMORY	DBY(99)=10

Zurück zu den Basic-Befehlen.

1.114 Basic-Befehl XBY()

COMMAND	FUNCTION	EXAMPLE(↩ S)
XBY()	READ/ASSIGN EXTERNAL DATA MEMORY	P.XBY(10)

Zurück zu den Basic-Befehlen.

1.115 Basic-Befehl GET

COMMAND	FUNCTION	EXAMPLE(↩ S)
GET	READ CONSOLE	P.GET

Zurück zu den Basic-Befehlen.

1.116 Basic-Befehl IE

COMMAND	FUNCTION	EXAMPLE(↩ S)

IE	READ/ASSIGN IE REGISTER	IE=82H

Zurück zu den
Basic-Befehlen.

1.117 Basic-Befehl IP

COMMAND	FUNCTION	EXAMPLE(↩ S)
IP	READ/ASSIGN IP REGISTER	IP=0

Zurück zu den
Basic-Befehlen.

1.118 Basic-Befehl PORT1

COMMAND	FUNCTION	EXAMPLE(↩ S)
PORT1	READ/ASSIGN I/O PORT 1 (P1)	PORT1=0FFH

Zurück zu den
Basic-Befehlen.

1.119 Basic-Befehl PCON

COMMAND	FUNCTION	EXAMPLE(↩ S)
PCON	READ/ASSIGN PCON REGISTER	PCON=0

Zurück zu den
Basic-Befehlen.

1.120 Basic-Befehl RCAP2

COMMAND	FUNCTION	EXAMPLE(↩ S)
RCAP2	READ/ASSIGN RCAP2 (RCAP2H:RCAP2L)	RCAP2=100

Zurück zu den
Basic-Befehlen.

1.121 Basic-Befehl T2CON

COMMAND	FUNCTION	EXAMPLE(↩ S)
T2CON	READ/ASSIGN T2CON REGISTER	P.T2CON

Zurück zu den
Basic-Befehlen.

1.122 Basic-Befehl TCON

COMMAND	FUNCTION	EXAMPLE(↩ S)
TCON	READ/ASSIGN TCON REGISTER	TCON=10H

Zurück zu den
Basic-Befehlen.

1.123 Basic-Befehl TMOD

COMMAND	FUNCTION	EXAMPLE(↩ S)
TMOD	READ/ASSIGN TMOD REGISTER	P.TMOD

Zurück zu den Basic-Befehlen.

1.124 Basic-Befehl TIME

COMMAND	FUNCTION	EXAMPLE(↩ S)
TIME	READ/ASSIGN THE REAL TIME CLOCK	P.TIME

Zurück zu den Basic-Befehlen.

1.125 Basic-Befehl TIMER0

COMMAND	FUNCTION	EXAMPLE(↩ S)
TIMER0	READ/ASSIGN TIMER0 (TH0: TL0)	TIMER0=0

Zurück zu den Basic-Befehlen.

1.126 Basic-Befehl TIMER1

COMMAND	FUNCTION	EXAMPLE(↩ S)

TIMER1	READ/ASSIGN TIMER1 (TH1: TL)	P.TIMER1

Zurück zu den
Basic-Befehlen.

1.127 Basic-Befehl TIMER2

COMMAND	FUNCTION	EXAMPLE(↩ S)
TIMER2	READ/ASSIGN TIMER2 (TH2: TL2)	TIMER2=0FFH

Zurück zu den
Basic-Befehlen.

1.128 Basic-Befehl PI

COMMAND	FUNCTION	EXAMPLE(↩ S)
PI	PI=3.1415926	PI

Zurück zu den
Basic-Befehlen.

1.129 Demoprogramme

Hier nun eine kleine Sammlung von 8052er Basic-Programmen.

Alle Programme könnten zwar eleganter geschrieben sein, aber um auch
Anwendern die wenig Basic-Kenntnisse haben den Einstieg zu ermöglichen
wurde auf die Optimierung verzichtet.

Programm

Lese_Port
Programm
Multiplex
Programm
Portausgangstest
Programm
ROMkopie
Programm
Temperaturmessung
Programm
Test_Multiplex
Programm
Uhrzeit
Programm
Uhrzeit_2
Programm
Meßwertgeberaufnahme
Programm
DCF_Decoder

1.130 Programm Lese_Port

Programm Lese_Port

Nach dem Start des Programms, kann der Wert der am Eingangsport Adresse 0C004 Hex anliegt gelesen und auf dem Monitor gebracht werden.

Die Hauptprogrammschleife erstreckt sich von Zeile 100 - 200. In Zeile 110 wird durch den GET-Befehl der serielle Eingang laufend abgefragt. Kommt nun das Dollarzeichen "$" an, wird zum Unterprogramm Zeile 9000 verzweigt. Hier wird der Wert der aus Adresse 0C004 Hex gelesen wurde, in die Variable MESS abgelegt und durch den PRINT-Befehl angezeigt.

Anmerkung zum Befehl XBY:

Dieser entspricht im Prinzip dem normalen Basic Peek oder Poke - Befehl. Ob nun gelesen oder geschrieben werden soll hängt davon ab, wo XBY steht. Ist dieser an zweiter Stelle wie z.B. MESS=XBY(Adresse) wird gelesen. Schreiben an eine Adresse sieht demzufolge so aus: XBY(Adresse)=Wert

Zurück zum Inhaltsverzeichnis
Demoprogramme
.

1.131 Programm Multiplex

Programm Multiplex

Die 8 Frequenzeingänge werden durchgescannt und der gelesene Wert kann mit den Tasten 1-8 ausgelesen werden.

In Zeile 515 wird der Ausgangsport Adresse 0C003 Hex angesteuert und die Datenbits D0, D1, D2 selektieren die die Eingangsadressen des Multiplex-IC. Um ein Inverter-IC auf der IN_OUT-Platine zu sparen, muß die Ansteuerung wie in den Zeilen 1000 - 1070 aussehen. Soll z.B. der erste Eingang ausgewählt werden, muß ein Wert zwischen 0-31 an Adresse 0C003 Hex geschrieben werden. Um die Sache nicht kompliziert zu machen, wird in Zeile 510 einfach 32 dazu addiert und somit kann der 5. Eingang gelesen werden usw... usw. .

Zurück zum Inhaltsverzeichnis
Demoprogramme
.

1.132 Programm Portausgangstest

Programm Portausgangstest

Nach dem Start des Programms können durch die Eingabe auf den Tasten 1-10 die folgenden Ausgaben gemacht werden:
Taste 1=Bit 1 (D0) usw. / Taste 9 gibt die Werte binär hochzählend von 0-255 aus. Taste 0=0

In Zeile 80 wird die Adresse des Ausgangsport festgelegt. (Hier Port 3.) Ist die optische Ausgangskontrolle (Bildergalerie/Hilfsmittel) angeschlossen kann die Ausgabe kontrolliert werden. Gleichzeitig erfolgt auch die Ausgabe des entsprechenden Wertes auf dem Monitor.
Achtung: Die LED's zeigen den Wert invertiert an. Also beim Wert 0 leuchten alle Leuchtdioden.

Zurück zum Inhaltsverzeichnis
Demoprogramme
.

1.133 Programm ROMkopie

Programm ROMkopie

Benötigt wird der Adaptersockel wie in
Tips zur EPROMprogrammierung
beschrieben. Das Programm selbst hat eine Benutzerführung.

Zurück zum Inhaltsverzeichnis
Demoprogramme
.

1.134 Programm Temperaturmessung

```
                Programm Temperaturmessung

Liest den Wert der an T1 bzw. Multiplexeingang anliegt und gibt diesen
auf das LCD und dem Monitor aus.
Zwischen den Zeilen 9000-9290 liegen die Unterprogramme für die Ansteuerung
des LCD-Displays. Nach dem Start folgt zuerst die Initialisierung des LCD,
wie in Zeile 20 (Gosub 9200). Danach können entweder Daten oder Kommandos
an das LCD gesendet werden.

Zurück zum Inhaltsverzeichnis
                 Demoprogramme
                .
```

1.135 Programm Test_Multiplex

```
                Programm Test_Multiplex

Die 8 Frequenzeingänge werden durchgescannt und der gelesene Wert auf
dem Monitor ausgegeben.
Das Programm ist ähnlich wie das Programm
                 Multiplex.
                Hier aber erfolgt die Ausgabe kontinuierlich auf dem Monitor im  ↩
                   Format
" Auf Eingang Nr. X ist XXXX Hertz ".

Zurück zum Inhaltsverzeichnis
                 Demoprogramme
                .
```

1.136 Programm Uhrzeit

```
                Programm Uhrzeit

Nach den Programmstart wird man aufgefordert das aktuelle Datum sowie die
Uhrzeit einzugeben. Die Ausgabe erfolgt dann auf das LCD und dem Monitor.
Ab Zeile 60000 fängt das Unterprogramm für die Berechnung der Uhrzeit und
des Datums an. Zwischen den Zeilen 9000-9290 liegen die Unterprogramme für
die Ansteuerung des LCD's. In Zeile 10 wird die Systemvariable XTAL mit der
entsprechenden Quarzfrequenz gesetzt. Indem man an dieser Systemvariablen
etwas herumdoktort kann die Softwareuhr geeicht werden. Geht die Uhr z.B.
1 Minute am Tag nach, kann der Wert entsprechend optimiert werden.

Zurück zum Inhaltsverzeichnis
                 Demoprogramme
                .
```

1.137 Programm Uhrzeit_2

Programm Uhrzeit_2

Nach dem Start des Programms kann die Uhrzeit mit Shift/S gesetzt und mit Shift/T geholt werden. Shift/4 ($) gibt den Wert der an T1 anliegt aus. Uhrzeit, Datum und Meßwert werden auch auf dem LCD angezeigt.
Eigentlich ist dieses Programm nur eine Erweiterung des Programms Uhrzeit.

Zurück zum Inhaltsverzeichnis
Demoprogramme
.

1.138 Programm Meßwertgeberaufnahme

Programm Meßwertgeberaufnahme

Meßwertgeberaufnahme ist nur ein kleines Demoprogramm, das in BlitzBasic 2 geschrieben wurde und das zeigen soll, wie die Software auf Amigaseite aussehen könnte.
Zuerst ist es nötig, das Programm Multiplex zum 8052er System zu übertragen und zu starten. Danach muß das Terminalprogramm beendet werden. Es ist nämlich nicht möglich, daß zwei Programme den Seriellport teilen. Das Demoprogramm sendet die Tastaturcodes und gibt die empfangenen Werte aus.

Zurück zum Inhaltsverzeichnis
Demoprogramme
.

1.139 Programm DCF_Decoder

Programm DCF_Decoder

Das Listing "DCF77-Dekoder" aus dem 8052 AH-Basic Buch wurde hier als Grundlage genommen und entsprechend der anderen Hardware angepaßt. Da auch hier nur die wichtigsten Funktionen übernommen wurden, verkleinerte sich das Programm wesentlich.
Die REM-Zeilen 1 bis 51 zeigen die gebrauchten Variablen und Strings an. Bei Platzproblemen können diese entfernt werden.
Wichtig ist, daß der Gebereingang 8 an dem die 625 Hertz Rechteckspannung anliegt auch eingestellt wird (Zeile 100 XBY(0C003H)=255).

Da im Prinzip die Uhr nur beim Neustart gestellt werden braucht und danach evtl. einmal die Stunde (oder am Tag) synchron sein braucht (der Timer des Prozessors ist verhältnismäßig genau), kann mit einer TMOD-Anweisung der externe Interrupt abgeschaltet werden. So ist genügend Prozessorzeit für andere Routinen übrig und es können keine Interrupts das Programm unterbrechen.

Beim Start des Programms wird das LCD-Display zurückgesetzt und die Uhrzeit mit 00:00:00 ausgegeben. Das Datum ist 1. Januar '00 und vor der Uhrzeit erscheint ein "-" -Zeichen, das anzeigt das keine Synchronisation besteht.

```
Nach spätestens 3 Minuten (einwandfreier Empfang vorrausgesetzt) sollte
die aktuelle Uhrzeit im Display erscheinen. Außerdem wechselt das
"-" -Zeichen zum "*", das einwandfreien Empfang signalisiert.
Durch Eingabe von "$" wird die aktuelle Uhrzeit, sowie das Datum seriell
ausgegeben.

Shift/T holt die Daten im Format:
Wochentag, Monatstag, Monat, Jahr, Stunde, Minuten, Sekunden, Synch. und
Sommerzeit.
Zum Beispiel steht 1 bei Wochentag für den Montag.
Bei Synch. steht 42, wenn einwandfreier Empfang war, bzw. 45 wenn dieser
gestört ist.
( 42 ist der Dezimale ASCII-Code des Zeichens "*", 45 das Zeichen "-" )
Bei Sommerzeit wird eine 2, Normalzeit eine 1 ausgegeben.

Shift/L gibt die Uhrzeit aus, wann die letzte Synchronisation bestand.

Mit Shift/S kann die Uhrzeit auch manuell über die Tastatur gesetzt werden
im Format: Wochentag, Tag, Monat, Jahr, Stunden, Minuten, Sekunden, jeweils
getrennt durch ein Komma.


Zurück zum Inhaltsverzeichnis
                 Demoprogramme
                .
```

1.140 Tips zur EPROMprogrammierung

```
                Da zur EPROMprogrammierung die Portausgänge 1.3 und 1.4 benutzt ↩
                   werden,
sollte hier nichts angeschlossen sein, wie z.B. Relais oder LED's, da
der Pegel evtl. heruntergezogen werden kann und die EPROMprogrammierung
dann nicht funktioniert.
Also hier die beste Vorgehensweise:
Bei ausgeschalteter Prozessorplatine das EPROM einsetzen. Da das EPROM
ein CMOS-IC ist, sollte statische Aufladung vermieden bzw. abgeleitet
werden. Eine evtl. Verbindung von den Portausgängen 1.3 und 1.4 zu
externen Ein oder Ausgabebausteinen lösen. Da die Portausgänge 1.0 - 1.7
sowieso auf eine Stiftleiste geführt und normalerweise ein Stecker dafür
verwendet werden sollte, braucht dieser nur abgezogen werden.
Schalter S2 und S3 können geschlossen und das 8052er System eingeschaltet
werden. Nach dem Betätigen der Leertaste, müßte sich der Interpreter wie
gewohnt melden. Nun, nachdem man sein Programm übertragen hat, wird S1
geschlossen und die Programmierspannung steht dem EPROM zur Verfügung.
Durch die PROG-Anweisung wird das im Speicher stehende Programm ins
EPROM gebrannt. In der Programmierphase leuchtet die LED und auf dem
Terminal wird das PROG z.B. 1 angezeigt. Es ist nämlich möglich mehrere
Programme hintereinander ins EPROM zu brennen und der Interpreter
nummeriert diese dann. Für die 8 KB große Interpreterkopie braucht
der Prozessor ca. 7 Minuten. Nachdem die LED ausgegangen ist, muß Schalter
S1 geöffnet werden und die EPROMprogrammierung ist damit beendet.
Schalter S2 und S3 können auch wieder geöffnet werden und es kann ein
kleiner Test erfolgen. Durch den Befehl
                 ROM
                 wird ins EPROM geschaltet
```

und mit

LIST

kann das gerade gebrannte Programm auf's Terminal gebracht werden.
Der Interpreter stellt auch einen INTELligenten Programmieralgorithmus zur Verfügung, aber dieser wird bei diesem Hardwareaufbau nicht unterstützt. Also die ganzen FPROG-Programmieranweisungen funktionieren nicht und deshalb sollten nur die PROG(1-6)-Befehle verwendet werden.
Das Kopieren des Interpreters ins EPROM erfordert zusätzlich noch einen Adaptersockel. Die Programme, die normalerweise durch PROG gebrannt werden, liegen in der zweiten Hälfte des EPROMs. Beim Systemstart sollte der Prozessor den Interpreter ab Adresse 0000 Hex im EPROM finden. Um nun die erste Hälfte im EPROM anzusprechen und den Interpreter dorthin zu kopieren, wird eine Adressleitung auf Masse gelegt.
Also man nehme:
Zwei 28 polige IC-Fassungen, biege von einer Fassung Pin 27 (A14) um und löte einen kurzen Draht an. Dieser Draht wird mit Pin 14 (Masse) verbunden und angelötet. Diese Fassung wird auf die andere gesteckt und zusammen-gedrückt. Dabei ist zu beachten, daß der umgebogene Pin 27 (A14) keinen Kontakt mit dem darunterliegenden hat (evtl. etwas Isolierband nehmen). Der Massepin 14 bleibt natürlich durchgehend. Zweck der Sache ist, daß die Adreßleitung A14 zum EPROM unterbrochen und der Anschlußpin am EPROM an Masse gelegt wird. Dadurch wird die Adressierung "umgebogen". Auf die gestapelten Fassungen wird das EPROM eingesetzt und dieses ganze Gebilde in den EPROMsockel auf dem Prozessorboard gesteckt. Nach der Kopie des Interpreters in das EPROM muß der Adaptersockel wieder entfernt werden. Nach dem Austausch des 8052 AH-Basic-Prozessors gegen einen 80C32, muß nur noch das interne ROM des 80C32 abgeschaltet werden. Dieses geschieht, indem man eine Drahtbrücke neben dem Prozessor oberhalb C7 (wird mit Jumper bezeichnet) einlötet.

1.141 Adresse des Autors

Bitte schicken Sie das ausgefüllte Registrierungsformular an:

Andreas Heinrich
Ellerstraße 34
33615 Bielefeld

Wenn Ihnen ein Fehler am Programm oder der Hardware auffallen sollte, scheuen Sie sich bitte nicht, mir diesen mitzuteilen.

Ich bedanke mich im vorraus.