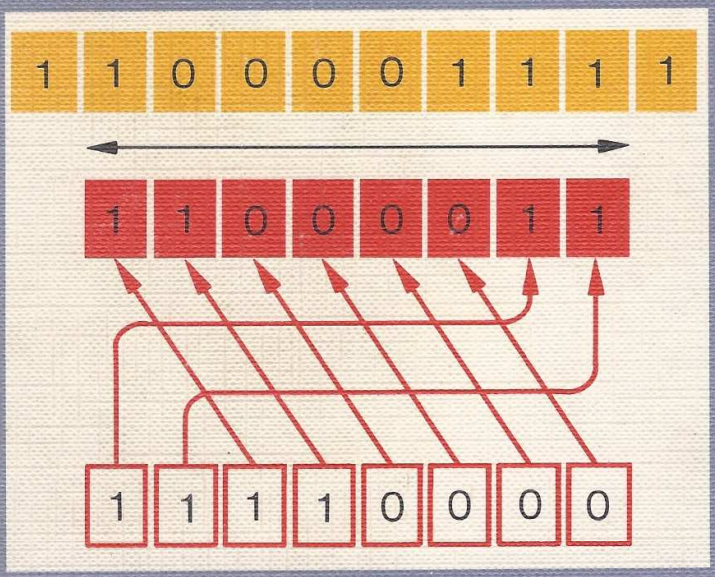
Franzis
Computer-Praxis

Feichtinger
Mit Computern steuern

Aufbau und Anwendung von Einplatinen-Mikrocomputern



Steuern und Regeln
Automatisierung
Arbeitsweise
eines Mikrocomputers
8- oder 16-Bit-Mikrocomputer?
Peripherie-Bausteine
Sensoren und Aktuatoren
Wie entsteht ein Programm?
Steuerung einer Relaisfunkstelle
Ein einfacher
Einplatinen-Computer
Erstellen eines Flußdiagramms
Der Assembler
Das Programm
kommt ins EPROM
Hardware-Fehlersuche
„Sichere" Mikrocomputer
Universeller 6502-Computer
Sukzessive Approximation
Kreuzkorrelation
Funkfernschreib-Empfänger

Franzis'

Inhalt

1 Einleitung ... 9
1.1 Steuern und Regeln ... 9
1.2 Automatisierung ... 10
1.3 Etwas Geschichte ... 12
1.4 Analogtechnik, Digitaltechnik, programmgesteuerte Logik ... 15
1.5 Arbeitsweise eines Mikrocomputers ... 18

2 Bauelemente eines Mikrocomputer-Systems ... 21
2.1 Auswahl eines geeigneten Mikroprozessors ... 21
2.2 8- oder 16-Bit-Mikrocomputer? ... 25
2.3 Peripherie-Bausteine ... 26
2.4 Speicher-Bausteine ... 30
2.5 Einchip-Mikrocomputer ... 32
2.6 Verbindung mit der Außenwelt ... 38
2.7 Sensoren und Aktuatoren ... 45

3 Vorgehensweise bei der Realisation einer Anwendung ... 50
3.1 Wie entsteht ein Programm? ... 50
3.2 Musterbeispiel: Steuerung einer Relaisfunkstelle ... 59
3.3 Ein einfacher Einplatinen-Computer ... 61
3.4 Erstellen eines Flußdiagramms ... 68
3.5 Eingabe des Assembler-Programms ... 72
3.6 Der Assembler ... 81
3.7 Test der Software auf dem Entwicklungssystem ... 86
3.8 Das Programm kommt ins EPROM ... 89
3.9 Aufgaben des Monitorprogramms ... 92
3.10 Hardware-Fehlersuche ... 95

4 Was sonst noch dazugehört ... 97
4.1 Stromversorgung des Einplatinencomputers ... 97
4.2 Bussysteme ... 100
4.3 „Sichere" Mikrocomputer ... 106

5 Weitere Schaltungen von Einplatinencomputern ... 108
5.1 Eine Z80-Platine ... 108
5.2 Universeller 6502-Computer ... 110
5.3 Einfache 6809-Schaltung ... 113

6 Häufig gebrauchte Routinen ... 116
6.1 Zeitmessung ... 116
6.2 Tonerzeugung ... 120
6.3 Frequenzmessung ... 121
6.4 Sukzessive Approximation ... 122
6.5 Interrupt-Uhr ... 123
6.6 D/A-Wandler ... 125

6.7 Erzeugung beliebiger Schwingungsformen ... 126
6.8 Autokorrelation ... 128
6.9 Kreuzkorrelation ... 129
6.10 Parallele Ausgabe ... 132
6.11 Serielle Ausgabe ... 133
6.12 Serielle Eingabe ... 134
6.13 Tastenfeldabfrage ... 135
6.14 Arithmetik-Routinen ... 137

7 Vollständige Anwendungsprogramme ... 141
7.1 Tonrufempfänger ... 141
7.2 Die melodische Eieruhr ... 149
7.3 Funkfernschreib-Decoder ... 155
7.4 IEC-Bus/V.24-Interface ... 161

8 Tabellenanhang ... 170
8.1 Baustein-Bezeichnungen ... 170
8.2 Technologien ... 171
8.3 Hersteller von Mikrocomputer-Bauelementen ... 171
8.4 Hersteller von Tischcomputern und Peripherie-Geräten ... 172
8.5 Hersteller von Einplatinen-Computern ... 173

9 Literatur ... 175

Sachverzeichnis ... 177

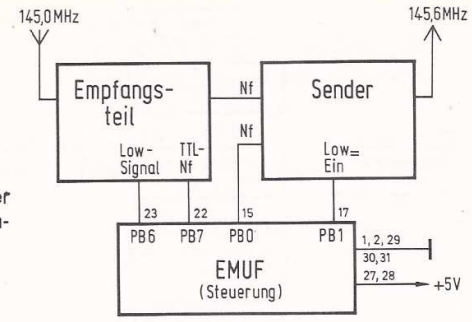


Abb. 3.2.1 Blockschaltbild einer Relaisfunkstelle mit Mikrocomputer-Steuerung

3.2 Musterbeispiel: Steuerung einer Relaisfunkstelle

Nehmen wir an, jemand hätte uns beauftragt, eine Relaisfunkstelle mit einer Steuerung auszurüsten. Die Tatsache, daß es sich dabei um ein ziemlich spezielles Problem handelt, ist typisch für die meisten Mikrocomputer-Anwendungen: Wegen der relativ geringen erreichbaren Stückzahl lohnt es sich nämlich nicht, eigens eine integrierte Schaltung zu entwickeln, die auf einem Chip die nötigen Funktionen vereint. Ein Mikrocomputer, der die geringen Kosten einer Standard-Hardware mit der Flexibilität der Programmierbarkeit kombiniert, ist für solche Spezialaufgaben ideal geeignet.

Welche Aufgaben hat die Steuerung in einer Relaisfunkstelle zu erfüllen? Relaisfunkstellen dienen dazu, die Reichweite von Funkgeräten zu erhöhen, die im Kraftfahrzeug oder als Handgerät betrieben werden. Sie werden an exponiert gelegenen Standorten installiert, z. B. auf hohen Bergen.

Eine Relaisfunkstelle besteht aus drei Funktionseinheiten *(Abb. 3.2.1)*: einem Empfänger, einem Sender und der nötigen Steuerung. Der Sender arbeitet natürlich auf einer anderen Frequenz als der Empfänger, damit er diesen nicht stört. Die Steuerung hat nun folgende Aufgaben: Da man die Relaisfunkstelle ferngesteuert mit einem 1750-Hz-Tonruf einschalten kann, muß sie diesen Ton erkennen und sofort den Sender einschalten. Dieser überträgt dann alles verstärkt, was im Empfänger ankommt. Wird jedoch eine Zeitlang kein Signal mehr empfangen, so muß die Steuerung mit einer bestimmten Verzögerungszeit den Sender wieder ausschalten. Für diesen Zweck liefert der Empfänger ein sogenanntes Rauschsperren-Signal an die Steuerung. Es ist z. B. so lange auf logisch 1, wie etwas empfangen wird.

Die für die Lizenzvergabe zuständige Fernmeldebehörde erteilt der Relaisfunkstelle ein Rufzeichen, das diese nach dem Einschalten sowie bei eingeschaltetem Sender etwa alle drei Minuten in Morsetelegrafie ausstrahlen muß. Die Steuerung muß sich also um die Erzeugung dieses Rufzeichens sowie um seine regelmäßige Wiederholung kümmern. Manchmal kann es zusätzlich nützlich sein, wenn die Benutzer der Relaisfunkstelle mittels Tonruf nähere Einzelheiten z. B. deren Standort abrufen können. Die Ausstrahlung dieser Daten kann ebenfalls (als vorprogrammierter Text) in Morsetelegrafie erfolgen.

Relaisfunkstellen werden im sogenannten Wechselsprechbetrieb benutzt. Das bedeutet, daß die beiden Stationen, die die Relaisfunkstelle verwenden, abwechselnd spre-

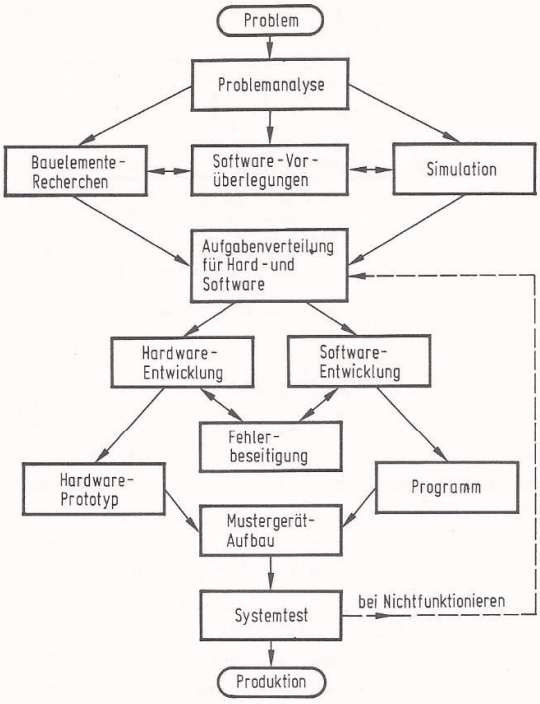


Abb. 3.2.2 Ein langer Weg: von der Problemanalyse zum fertigen System

chen. Nun hat es sich eingebürgert, daß die Relaisfunkstelle ein kurzes Tonsignal erzeugt, sobald eine Station zu senden aufhört, damit die andere weiß, daß sie nun dran ist. Natürlich muß die Steuerung auch diese Aufgabe wahrnehmen. Statt eines einfachen Tons kann dabei auch das Morsezeichen „K" ausgesendet werden.

Bei der Entwicklung der Steuerung muß man sich zunächst überlegen, welche Aufgaben überhaupt vom Mikrocomputer wahrgenommen werden können *(Abb. 3.2.2)*: Beginnen wir gleich mit der Erkennung des 1750-Hz-Tonrufs. Weiter oben wurden bereits drei Möglichkeiten zur Tonerkennung per Programm besprochen: Periodendauer-Messung, Autokorrelation und Kreuzkorrelation. Da man bei einer Relaisfunkstelle nicht immer mit ganz störungsfreiem Empfang rechnen kann, scheidet die Methode der Periodendauermessung eigentlich aus. Andererseits werden keine allzu hohen Anforderungen an die Ansprechempfindlichkeit gestellt, da es ja keinen Sinn hat, das Tonrufsignal einer Station auch dann noch zu erkennen, wenn das Sprachsignal ohnehin nicht mehr verständlich ankommt. Als Auswertungsmethode wählt man daher sinnvollerweise die Autokorrelation, deren Programmieraufwand im Vergleich zur Kreuzkorrelation geringer ist.

Um das Rauschsperren-Signal des Empfängers abzufragen, genügt ein normaler Eingang eines Peripherie-Bausteins. Ebenso ist es kein Problem, über einen Schalttransistor mit demselben Peripheriebaustein den Sender ein- und auszuschalten.

Die Aufgaben des Mikrocomputers als Relaisfunkstellen-Steuerung wären damit also festgelegt. Wie sich später noch zeigen wird, ist der Programmaufwand dafür nicht einmal besonders groß. Aus diesem Grunde genügt auch ein Mikrocomputer mit relativ wenig Speicherplatz. Fragt sich nur noch, welchen Mikroprozessortyp man verwenden soll. In unserem Fall fiel die Entscheidung für die CPU-Familie 6502, weil mit dem AIM-65 ein preisgünstiges Entwicklungssystem zur Verfügung stand. Ferner war es nicht erforderlich, erst einen Mikrocomputer zu entwickeln, weil unter der Bezeichnung „EMUF" (Einplatinen-Mikrocomputer für universelle Festprogramm-Anwendung) ein geeigneter Bausatz mit der CPU 6504, die zur CPU 6502 software-kompatibel ist, für weniger als 100 DM erhältlich ist[1]). Er wurde im Juli 1981 erstmals von der Zeitschrift „mc" als Applikationsschaltung vorgestellt [8] und wird mittlerweile von mehreren Firmen vertrieben *(s. Kap. 8.5)*. Dabei kann man sich entscheiden, ob man nur die (doppelseitige, durchkontaktierte) Platine im Europa-Format 100 × 160 mm², einen Bausatz mit allen Teilen oder auch den fertig bestückten und getesteten EMUF haben möchte. Für den EMUF wurde inzwischen schon eine Vielzahl an Applikations-Software veröffentlicht. Sehen wir uns also am besten seine Schaltung etwas näher an, weil sie für kleine Steuerungscomputer typisch ist.

3.3 Ein einfacher Einplatinen-Computer

Wenn wir uns die EMUF-Platine *(Abb. 3.3.1)* ansehen, so stellen wir fest, daß sie insgesamt fünf integrierte Schaltungen umfaßt: den Mikroprozessor 6504; ein IC namens 6532, das 128 Byte RAM, einen programmierbaren Zeitgeber (Timer) sowie 16

[1]) Der EMUF wird als Bausatz ohne EPROM für 89 DM angeboten (Stand: Juli 1983). Ein ähnlich preisgünstiger Computer mit dem Z80 als CPU wurde in mc 1983, Heft 4, veröffentlicht.

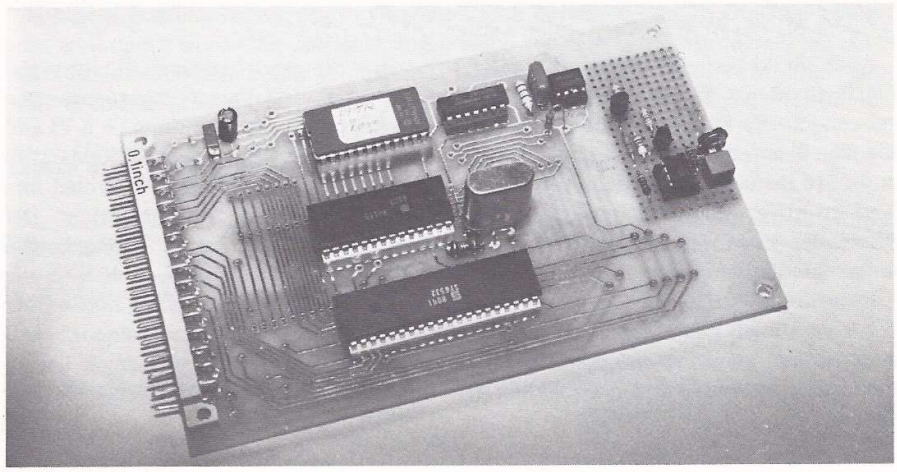


Abb. 3.3.1 Der „EMUF" ist ein kleiner Computer im Eurokarten-Format

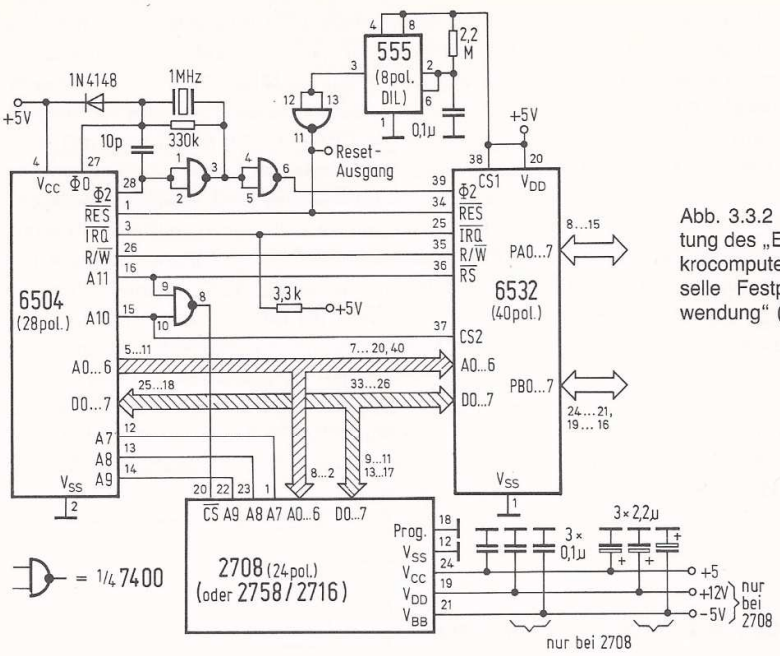


Abb. 3.3.2 Gesamtschaltung des „Einplatinen-Mikrocomputers für universelle Festprogramm-Anwendung" (EMUF)

I/O-Leitungen enthält; ein EPROM, in dem das jeweilige Anwenderprogramm steht (in unserem Beispiel das Betriebsprogramm für die Relaisfunkstelle); einen Zeitgeber (NE555), der beim Einschalten der Versorgungsspannung einen Reset-Impuls mit einigen 100 ms Länge erzeugt, um das Programm zu starten; und schließlich ein TTL-IC 7400 mit vier NAND-Gattern, von denen drei als Inverter und eines als Adressen-Decoder für das EPROM verwendet werden *(Abb. 3.3.2)*.

Als EPROM lassen sich unterschiedliche Typen einsetzen. Ursprünglich wurde der EMUF für den 1-KByte-Typ 2708 konzipiert, der noch drei Versorgungsspannungen verwendet. Die Typen 2758 oder 2508 benötigen nur noch eine Spannung (+ 5 V) und besitzen ebenfalls 1 KByte Kapazität. Die untereinander gleichwertigen EPROMs 2716 und 2516 lassen sich ebenfalls verwenden: Sie besitzen 2 KByte Kapazität, wobei ihre höchstwertige Adressenleitung A10 dem 12-V-Anschluß des 2708 entspricht. Der Anwender kann dann das 2-KByte-EPROM mit zwei je maximal 1 KByte langen unterschiedlichen Programmen laden und zwischen diesen wählen, indem er die Adressenleitung A10 (d. h. den 12-V-Anschluß des EMUF) wahlweise mit Masse oder + 5 V verbindet. Bei unserer Relaisfunkstelle wäre es somit möglich, mit einem Schalter zwischen zwei Programmversionen zu wählen, z. B. mit unterschiedlich langen Verzögerungszeiten zum Ausschalten des Senders oder unterschiedlichen Morse-Texten.

Es ist dem Leser vielleicht noch im Gedächtnis, daß der Mikroprozessor 6502 (und ebenso der 6504) beim Einschalten, also bei einem Reset, erst in den beiden Speicher-

zellen hex FFFC und FFFD nachsieht, wo er mit der Abarbeitung eines Programms beginnen soll. Demzufolge muß der EPROM-Bereich am oberen Ende des Adressenbereichs 0000...FFFF liegen. Andererseits lassen sich beim 6502 zahlreiche Befehlsarten speichersparend auf die sogenannte „Zero Page" (0000...00FF) anwenden, so daß der Arbeitsspeicher (RAM) am unteren Speicherende (ab 0000) liegen sollte.

Dazu kommt noch, daß die Mikroprozessor-Familie 6502 einen speziellen Speicherbereich von 01FF abwärts bis 0100 dazu verwendet, um sich die Rücksprungadressen bei Unterprogrammsprüngen zu merken (Stackbereich). Also muß auch dort irgendwo RAM zur Verfügung stehen.

Oben wurde schon erwähnt, daß im Baustein 6532 unter anderem 128 Bytes RAM enthalten sind. Wie will man nun mit 128 Bytes sowohl oberhalb 0000 als auch unterhalb 01FF Speicherplatz zur Verfügung stellen? Dies geschieht mit einem Adressierungskniff. Weil nämlich zur Ansteuerung des RAM die Adressenleitungen A0...A6 sowie A11 verwendet werden, nicht aber die übrigen, dupliziert sich das RAM mehrfach in unterschiedliche Bereiche. Das bedeutet, daß zum Beispiel der Inhalt der Speicherzelle 0000 auch an den Adressen 0080, 0100, 0180, 0200 usw. bis 0380 erscheint. Die RAM-Speicherzellen erscheinen also an folgenden Adressen: 0000...007F; 0080...00FF; 0100...017F; 0180...01FF usw. Darüber hinaus werden die Adressenleitungen, die höherwertig als A11 sind, überhaupt nicht decodiert, also ignoriert. Das führt dazu, daß die Adressen 0000...0FFF gleichwertig sind mit 1000...1FFF, 2000...2FFF usw. bis F000...FFFF. Letzteres ist sehr wichtig, weil damit auch der Reset-Vektor FFFC...FFFD unter anderem in den Speicherzellen 0FFC und 0FFD wiederzufinden ist *(Tab. 3.3.1)*.

Weil nun aber das Entwicklungssystem AIM-65 einen Arbeitsspeicher besitzt, der 4 KByte groß ist und von 0000 bis 0FFF reicht, ist es möglich, den Adressenbereich des EMUF-EPROMs von 0C00...0FFF zu legen und das Programm in gleicher Adressenlage auf dem AIM-65 zu entwickeln, bevor es in das EPROM geladen wird.

Solche Adressenduplizierungen vermeidet man normalerweise in größeren Computern, weil damit wertvoller Adressierungsbereich verloren geht. Würde man beispielsweise bei einem Tischcomputer, dessen CPU 16 Adressenleitungen besitzt, eine davon nicht decodieren, so würde sich der ansprechbare Speicherbereich auf die Hälfte, also

Tabelle 3.3.1: Adressenbelegung des Einplatinencomputers EMUF

Adressenbits	Inhalt	Adressenbereiche
00XX XAAA AAAA	128 Byte RAM im 6532	000...07F; 080...0FF; 100...17F; 180...1FF; 200...27F; 280...2FF; 300...37F; 380...3FF
10XX XXXA AAAA	I/O-Ports und Timer im 6532	800...81F u.a. (32mal dupliziert bis BFF)
11AA AAAA AAAA	EPROM (1 KByte)	C00...FFF
01AA AAAA AAAA	Expansion (1 KByte)	400...7FF

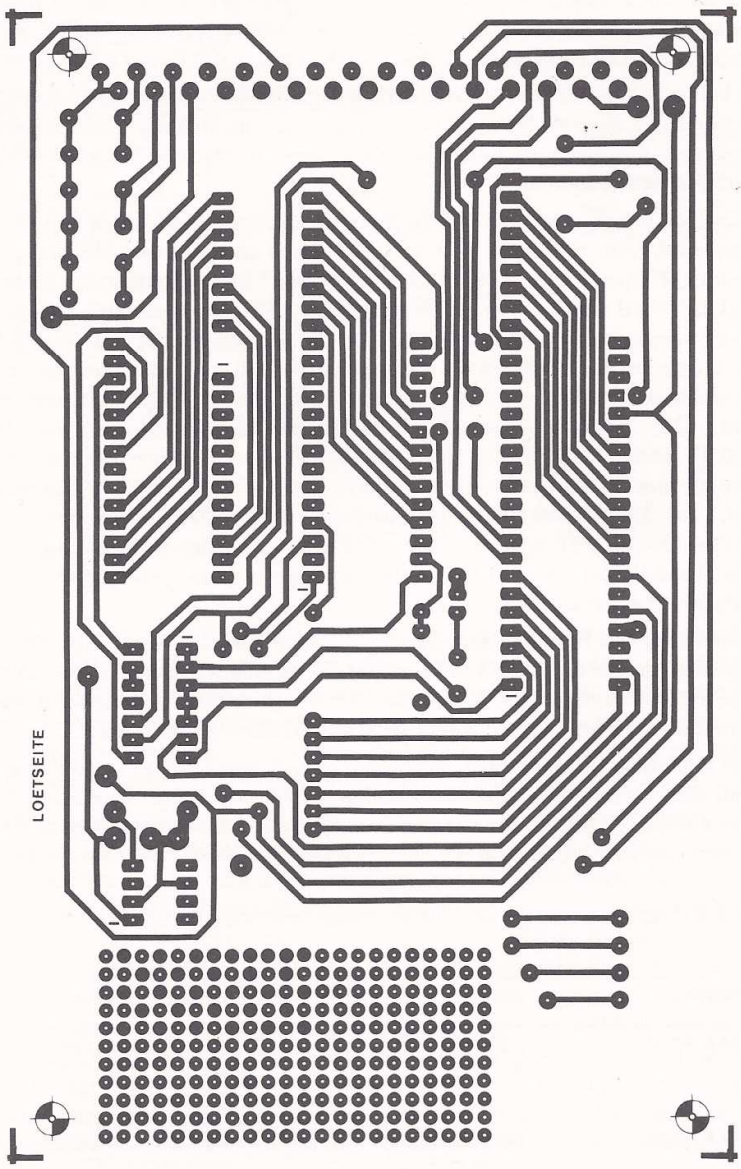


Abb. 3.3.3 Lötseite der EMUF-Platine

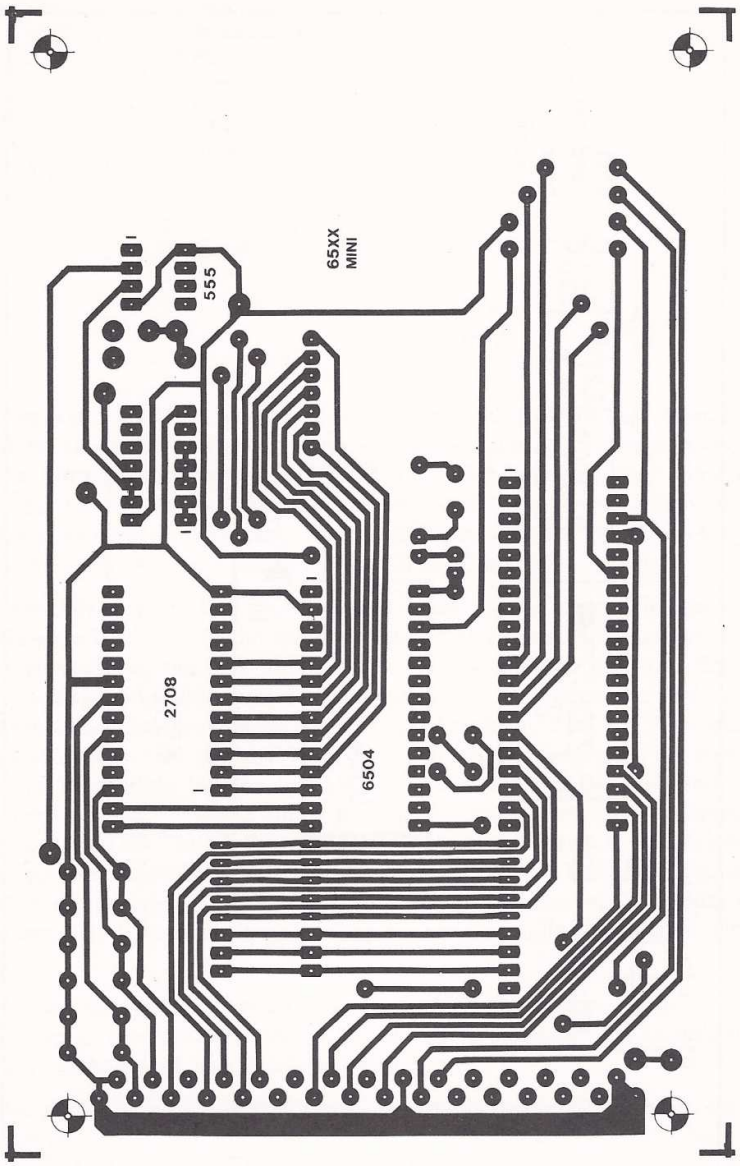


Abb. 3.3.4 Bestückungsseitige Leiterbahnen der EMUF-Platine

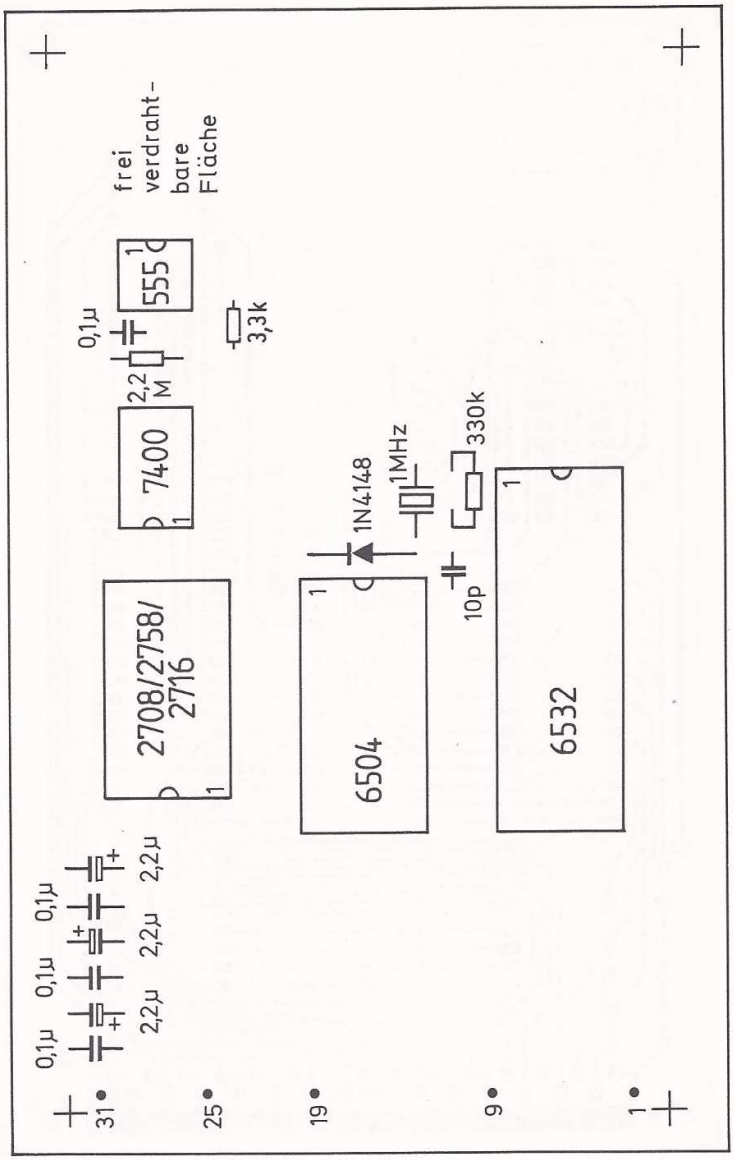


Abb. 3.3.5 Bestückungsplan des Einplatinen-Mikrocomputers EMUF

Tabelle 3.3.2: Stiftbelegung der 31poligen EMUF-Steckleiste

1 Masse	18 PB2
2 Masse	19 PB3
4 IRQ	21 Reset-Ausgang
6 PA0	22 PB7
7 PA1	23 PB6
8 PA2	24 PB5
9 PA7	25 PB4
10 PA6	26 Reset-Eingang
11 PA5	27 + 5 V
12 PA4	28 – 5 V (bei 2716 + 5 V)
13 PA3	29 + 12 V (bei 2716 Masse)
14 Masse	30 Masse
15 PB0	31 Masse
17 PB1	

auf 32 KByte statt 64 KByte, verringern. Bei unserem Einplatinen-Mikrocomputer spielt dies jedoch keine Rolle, weil ein so großer Adressenbereich ohnehin nicht benötigt wird. Die Adressenduplizierung wird hier absichtlich verwendet, um die Programmerstellung auf dem Entwicklungssystem zu vereinfachen, indem sie gleich im endgültigen Adressenbereich erfolgen kann, weil der EPROM-Adressenbereich des Einplatinen-Computers gleich dem Arbeitsspeicherbereich des Entwicklungssystems ist.

Im Gegensatz zum Mikroprozessor 6502 kann die in einem kleineren Gehäuse untergebrachte Version 6504 mit ihren nur 13 statt 16 Adressenleitungen statt 64 KByte nur 8 KByte adressieren. Sie unterscheidet sich von ihrem größeren Bruder auch dadurch, daß sie einige Steueranschlüsse weniger besitzt, nicht zuletzt deshalb, weil sie statt 40 nur 28 Gehäusepins besitzt. Die CPU 6504 verbraucht bei 5 V Versorgungsspannung typisch knapp über 100 mA. Wem das zu viel ist, der kann unter der Bezeichnung 65C04 eine kompatible Version in CMOS-Technik erhalten. Der Prozessor wird mit 1 MHz Taktfrequenz betrieben; eine 8-Bit-Addition kann er in 2 µs durchführen. Eine zuverlässige Funktion ist im Spannungsbereich von 4,75...5,25 V gewährleistet. Dies gilt auch für den Baustein 6532, so daß es notwendig ist, die Versorgungsspannung für den EMUF mit geeigneten Maßnahmen, z. B. einem integrierten Spannungsregler, zu stabilisieren. Die Belastbarkeit dieser Stromversorgung sollte dabei etwa 300 mA betragen.

Für Leute, die fertige Layouts der Eigenentwicklung von Platinen vorziehen: *Abb. 3.3.3* zeigt die Lötseite, *Abb. 3.3.4* die bestückungsseitigen Leiterbahnen der EMUF-Europakarte, und *Abb. 3.3.5* die Bestückung.

So viel zunächst also zur Hardware, der Mikrocomputer-Schaltung. Für unsere spezifische Aufgabe, hier die Steuerung einer Relaisfunkstelle, wird die universell verwendbare Mikrocomputer-Platine aber erst durch die Erstellung des Betriebsprogrammes im EPROM zugeschnitten. Wenden wir uns also den einzelnen Schritten dieser Programmerstellung zu.

```
FF00          PASS 1
0D0C          PASS 2
0000
0000          ;REPEATER (DC1YB)
0000          ;PB7=AF INPUT (TTL)
0000          ;PB6=SQUELCH INPUT
0000          ;PB1=TX OFF OUTPUT
0000          ;PB0=AF OUTPUT
0000 TIL            *=*+1
0001 TIM            *=*+1
0002 TIH            *=*+1
0003 THLD           *=*+1
0004 SUM            *=*+1
0005 SPL1           *=*+1
0006 SPL2           *=*+1
0007 PB             =$802
0007 PBD            =$803
0007                *=$FFC
0FFC          000C  .WOR RES
0FFE                *=$C00
0C00 RES      A2FF  LDX £$FF
0C02          9A    TXS
0C03          D8    CLD
0C04          78    SEI
0C05          A903  LDA £3
0C07          8D0308 STA PBD
0C0A          ;TX OFF
0C0A OFF      AD0208 LDA PB
0C0D          0902  ORA £2
0C0F          8D0208 STA PB
0C12          ;WAIT FOR 1750 HH
0C12          A264  LDX £100
0C14 OFF1     20670C JSR AK
0C17          10F1  BPL OFF
0C19          CA    DEX
0C1A          D0F8  BNE OFF1
0C1C          ;TX ON
0C1C          AD0208 LDA PB
0C1F          29FD  AND £$FD
0C21          8D0208 STA PB
0C24
0C24          ;SEND CW CALL
0C24 CA       A200  LDX £0
0C26          20A40C JSR CW
0C29          A902  LDA £2
0C2B          8502  STA TIH
0C2D ON       A9FE  LDA £$FE
0C2F          8503  STA THLD
0C31          ;COUNT DOWN TIMER
0C31 ON3      C600  DEC TIL
0C33          D020  BNE ON2
0C35          C601  DEC TIM
0C37          D004  BNE ON1
0C39          C602  DEC TIH
0C3B          F0E7  BEQ CA
0C3D ON1      2C0208 BIT PB
0C40          50EB  BVC ON
0C42          E603  INC THLD
0C44          F008  BEQ BEEP
0C46          A90A  LDA £10
0C48          C503  CMP THLD
0C4A          30BE  BMI OFF
0C4C          1007  BPL ON2
0C4E
0C4E          ;SEND ROGER BEEP
0C4E BEEP     A209  LDX £K-CALL
0C50          8606  STX SPL2
0C52          20AA0C JSR NXT
0C55          ;TEXT REQUEST?
0C55 ON2      A264  LDX £100
0C57 REQ      20670C JSR AK
0C5A          10D5  BPL ON3
0C5C          CA    DEX
0C5D          D0F8  BNE REQ
0C5F          A20B  LDX £TXT-CALL
0C61          20A40C JSR CW
0C64          4C310C JMP ON3
0C67
0C67          ;AUTOCORRELATION
0C67          ;1750HZ,A/X/Y SAVED
0C67 AK       48    PHA
0C68          98    TYA
0C69          48    PHA
0C6A          8A    TXA
0C6B          48    PHA
0C6C          A900  LDA £0
0C6E          8504  STA SUM
0C70          A010  LDY £16
0C72 AK1      A20A  LDX £10
0C74 AK2      CA    DEX
0C75          D0FD  BNE AK2
0C77          AD0208 LDA PB
0C7A          0A    ASL A
0C7B          6605  ROR SPL1
0C7D          6606  ROR SPL2
0C7F          88    DEY
0C80          D0F0  BNE AK1
0C82          A505  LDA SPL1
0C84          F010  BEQ AK5
0C86          C9FF  CMP £$FF
0C88          F00C  BEQ AK5
0C8A          4506  EOR SPL2
0C8C          A007  LDY £7
0C8E AK3      6A    ROR A
0C8F          B002  BCS AK4
0C91          C604  DEC SUM
0C93 AK4      88    DEY
0C94          10F8  BPL AK3
0C96 AK5      A504  LDA SUM
0C98          6905  ADC £5
```

zu Abb. 3.6.3

```
0C9A        8504    STA SUM
0C9C        68      PLA
0C9D        AA      TAX
0C9E        68      PLA
0C9F        A8      TAY
0CA0        68      PLA
0CA1        2404    BIT SUM
0CA3 RDY    60      RTS
0CA4        ;SEND CW CODE
0CA4 CW     8606    STX SPL2
0CA6        A21E    LDX £30
0CA8        D002    BNE *+4
0CAA NXT    A202    LDX £2
0CAC SPC    18      CLC
0CAD        20D80C  JSR ELEM
0CB0        A606    LDX SPL2
0CB2        E606    INC SPL2
0CB4        BDF30C  LDA CALL,X
0CB7        F0EA    BEQ RDY
0CB9        8505    STA SPL1
0CBB        C9FF    CMP £$FF
0CBD        D004    BNE SHFT
0CBF        A207    LDX £7
0CC1        D0E9    BNE SPC
0CC3 SHFT   0605    ASL SPL1
0CC5        F0E3    BEQ NXT
0CC7        A201    LDX £1
0CC9        9002    BCC DOT
0CCB        A203    LDX £3
0CCD DOT    38      SEC
0CCE        20D80C  JSR ELEM
0CD1        18      CLC
0CD2        E8      INX
0CD3        20D80C  JSR ELEM
0CD6        F0EB    BEQ SHFT
0CD8 ELEM   A096    LDY £150
0CDA ELEM1  A932    LDA £50
0CDC        8504    STA SUM
0CDE ELEM2  C604    DEC SUM
0CE0        D0FC    BNE ELEM2
0CE2        9008    BCC QUIET
0CE4        AD0208  LDA PB
0CE7        4901    EOR £1
0CE9        8D0208  STA PB
0CEC QUIET  88      DEY
0CED        D0EB    BNE ELEM1
0CEF        CA      DEX
0CF0        D0E6    BNE ELEM
0CF2        60      RTS
0CF3        ;CW CALL 'DC1YB'
0CF3 CALL   90      .BYT $90,$A8
0CF4        A8
0CF5        7C      .BYT $7C,$B8
0CF6        B8
0CF7        88      .BYT $88,0
0CF8        00
0CF9        FF      .BYT $FF,$FF,$FF
0CFA        FF
0CFB        FF
0CFC        ;ROGER BEEP 'K'
0CFC K      B0      .BYT $B0,0
0CFD        00
0CFE        ;CW TEXT
0CFE        ;'TEST DE DC1YB'
0CFE TXT    C0      .BYT $C0,$40
0CFF        40
0D00        10      .BYT $10,$C0
0D01        C0
0D02        FF      .BYT $FF,$90
0D03        90
0D04        40      .BYT $40,$FF
0D05        FF
0D06        90      .BYT $90,$A8
0D07        A8
0D08        7C      .BYT $7C,$B8
0D09        B8
0D0A        88      .BYT $88,0
0D0B        00
0D0C                .END
0D0C         ERRORS= 0000
```

Abb. 3.6.3 Assembler-Listing des Relaisfunkstellen-Programms

5 Weitere Schaltungen von Einplatinen-Computern

Der EMUF, auf den bisher Bezug genommen wurde, ist die denkbar preiswerteste Lösung zur Realisation einfacher Steuerungsaufgaben. Im folgenden wollen wir uns drei weitere Einplatinen-Computer ansehen, deren Konzept etwas aufwendiger ist, und die sich nicht zuletzt wegen des größeren zur Verfügung stehenden Speicherplatzes auch für anspruchvollere Applikationen eignen. Die drei Computer [12, 13, 14] arbeiten mit den Mikroprozessoren Z80, 6502 unnd 6809.

5.1 Eine Z80-Platine

Mit 16 ICs arbeitet der in *Abb. 5.1* vorgestellte Z80-Computer. Er kann mit bis zu 2 KByte EPROM (1 × 2716) sowie 2 KByte RAM (4 × 2114) bestückt werden. Zwei Peripherie-Bausteine vom Typ 8255 stellen insgesamt 48 Ein-/Ausgabeleitungen zur Verfügung, zwei UART-Bausteine 8251 erlauben die serielle Ein- und Ausgabe von Daten (Universal Asynchronous Receiver/Transmitter). Damit kann man z. B. einen Drucker mit RS-232-Schnittstelle steuern oder ein Terminal anschließen. Im Gegensatz zum EMUF besitzt der Z80-Computer allerdings keinen eingebauten Timer; eventuell benötigte Verzögerungszeiten sind also durch Programmieren einer Zeitschleife zu realisieren. Wie funktioniert nun die Schaltung?

Beim Einschalten der Versorgungsspannung wird mit R 8 und C 7 ein Reset-Impuls generiert. Der Interrupt-Eingang INT ist über C 10 mit dem Taktgenerator für das UART (Baud-Rate-Generator) verbunden, so daß die CPU alle 256 Taktzyklen ein Interrupt-Signal erhält. Dies kann z. B. dafür benutzt werden, um im „Hintergrund" eine Software-Uhr mitlaufen zu lassen. Das laufende Hauptprogramm würde dann alle 256 Zyklen unterbrechen, und das Interrupt-Programm stellt dann die zur Uhrzeitspeicherung verwendeten Arbeitsspeicherzellen entsprechend weiter. Z 13 arbeitet als Adressendecoder für den Speicher und kann bis zu 8 Blöcke mit je 2 KByte adressieren. Damit ergibt sich folgende Speicheraufteilung:

2400... FFFF frei
2000... 23FF RAM 2 (Z 2, Z 3, 1 KByte)
1400... 1FFF frei
1000... 13FF RAM 1 (Z 4, Z 5, 1 KByte)
0800... 0FFF frei
0000... 07FF EPROM (Z 10, 2 KByte)

Bei dieser Adressenaufteilung ist es auch möglich, statt dem 2-KByte-EPROM 2716 einen 4-KByte-Typ (2732) zu verwenden. Dazu ist Pin 21 des 2732 an die Adressenleitung A 11 zu legen.

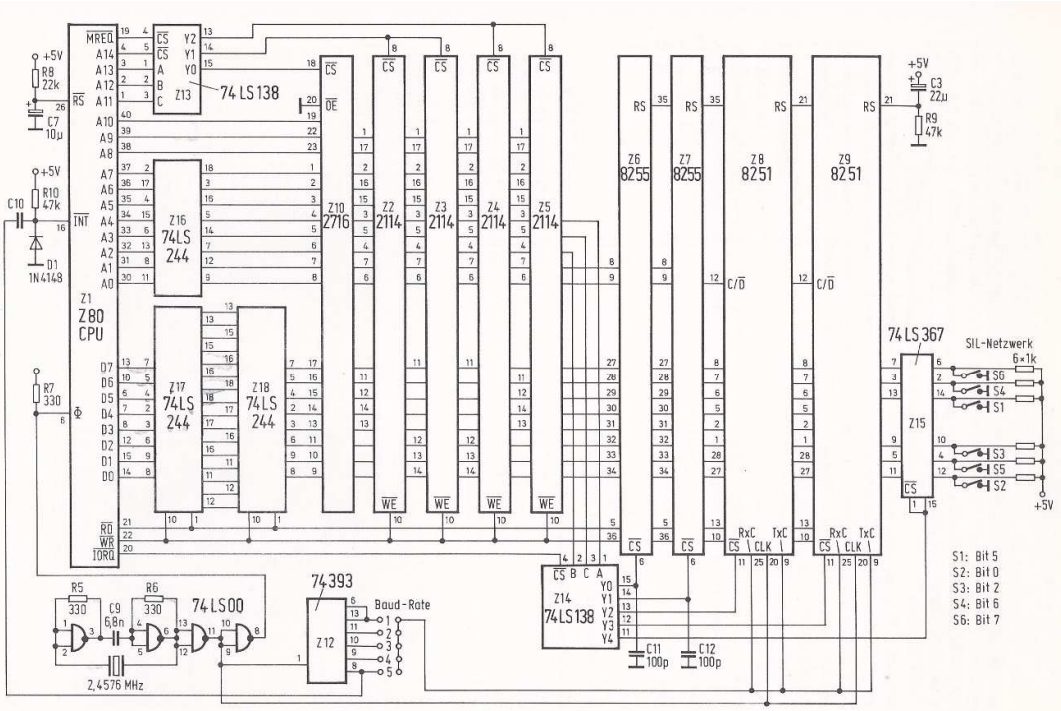


Abb. 5.1 Schaltung eines Einplatinen-Computers mit der CPU Z80

Tabelle 3.7.1: Adressenvergleich von EMUF, AIM-65, PET-2001, CBM-3001, KIM-1

	EMUF	AIM	CBM	PET	KIM	
PA	0800	A00F	E84F	E84F	1700	Port A
PAD	0801	A003	E843	E843	1701	Port-A-Richtung
PB	0802	A000	——	——	1702	Port B
PBD	0803	A002	——	——	1703	Port-B-Richtung
T1	0814	A494	——	——	1704	Timer 1 µs
T8	0815	A495	——	——	1705	Timer 8 µs
T64	0816	A496	——	——	1706	Timer 64 µs
TK	0817	A497	——	——	1707	Timer 1024 µs
T1I	081C	A49C	——	——	170C	T1 mit Interrupt
T8I	081D	A49D	——	——	170D	T8 mit Interrupt
T64I	081E	A49E	——	——	170E	T64 mit Interrupt
TKI	081F	A49F	——	——	170F	TK mit Interrupt
IRQL	0FFE	A404	0090	0219	17FE	IRQ-Vektor Low
IRQH	0FFF	A405	0091	021A	17FF	IRQ-Vektor High

Die Taktfrequenz der Schaltung beträgt 2,4576 MHz; diese Frequenz wurde gewählt, um auf einfache Weise die für den Baud-Rate-Generator benötigten Frequenzen zu erhalten. Zusammen mit dem per Programm zusätzlich einstellbaren Teilerfaktor der UART-Bausteine (1, 16 oder 64) kann man damit Geschwindigkeiten von 150, 300, 600, 1200, 2400, 4800 und 9600 Baud einstellen.

Der Baustein Z 15 erlaubt die Abfrage von 6 Schaltern per Programm. Diese Schalter können z. B. dazu dienen, unterschiedliche Varianten einer Ablaufsteuerung zuzulassen, ohne den EPROM-Baustein wechseln zu müssen. Die drei Bausteine Z 16, Z 17 und Z 18 dienen dazu, die Datenbusleitungen und die niederwertigen acht Adressenleitungen zu puffern, da die Belastbarkeit der CPU-Anschlüsse für die zahlreichen Peripherie-Bausteine nicht ausreichen würde.

Wie beim EMUF genügt für den Z80-Einplatinen-Computer eine Betriebsspannung von 5 V. Der Stromverbrauch ist allerdings etwa doppelt so hoch wie der des EMUF, was auf die größere Anzahl von ICs in der Schaltung zurückzuführen ist. Als Ausgleich bekommt man dafür aber auch eine größere Anzahl von Ein- und Ausgabeleitungen und mehr Speicherplatz zur Verfügung gestellt.

5.2 Universeller 6502-Computer

Wenden wir uns nun einer Platine zu, die etwa den gleichen Schaltungsaufwand beinhaltet, aber mit der CPU 6502 arbeitet (*Abb. 5.2.1*). Eine Besonderheit ist bei ihr erwähnenswert: Sie läßt sich nicht nur als eigenständiger Mikrocomputer, sondern auch als Erweiterungsplatine für 6502-Systeme anwenden, etwa für den AIM-65 oder den KIM-1. Zu diesem Zweck kann ihre Adressenbelegung mit Schaltern oder Draht-

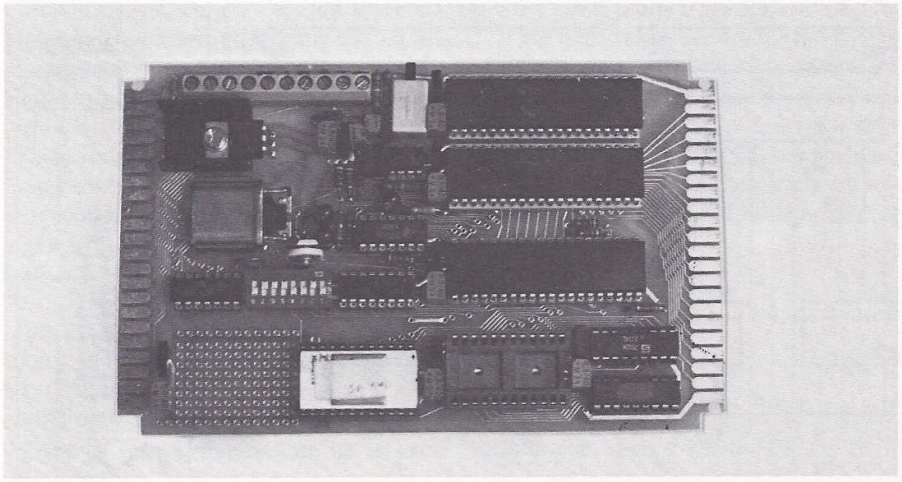


Abb. 5.2.1 Relativ dicht geht es auf der 6502-Platine zu

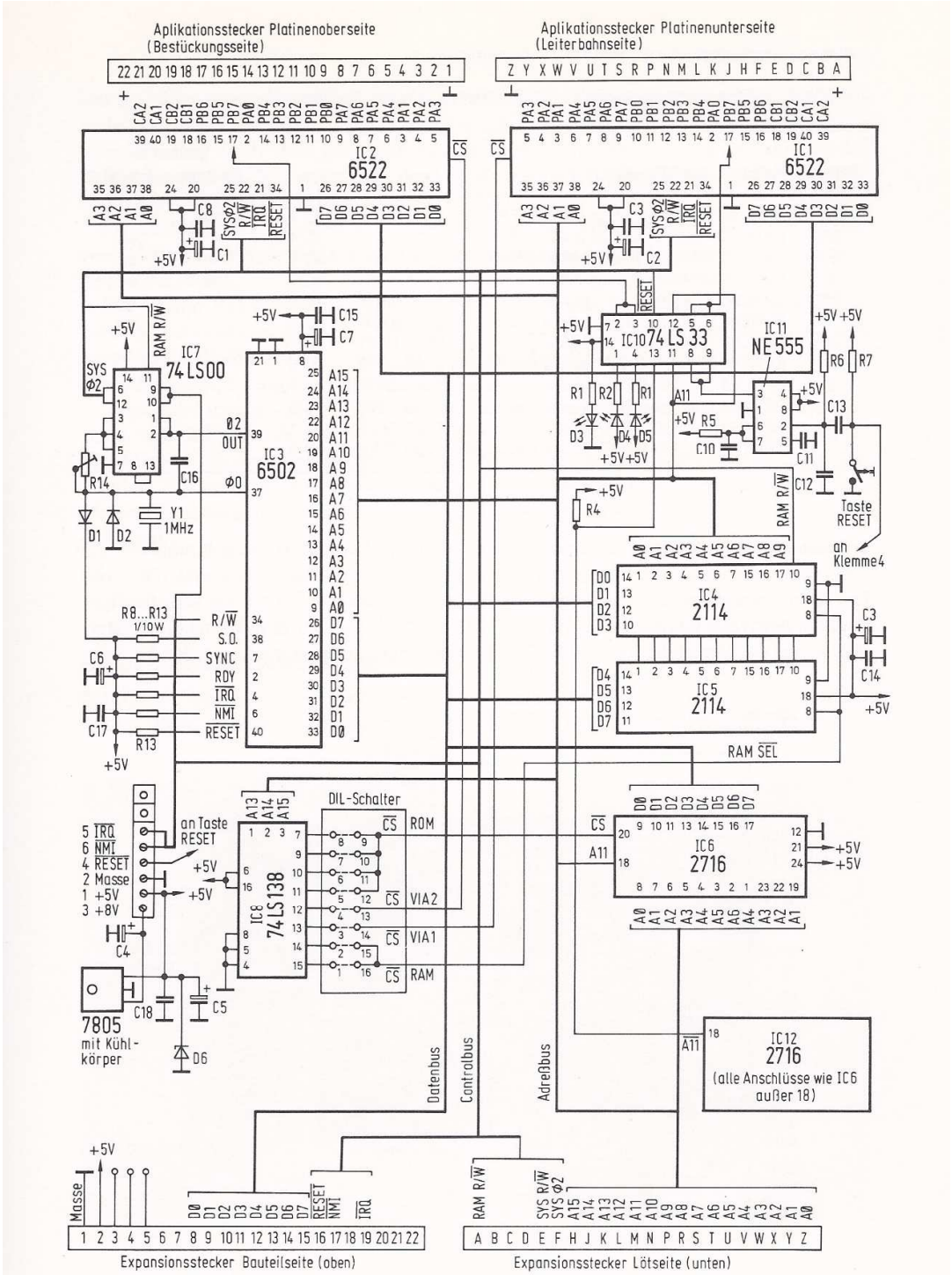


Abb. 5.2.2 Schaltung der 6502-Platine

Tabelle 5.2: Adressenbelegung der 6502-Platine je nach Betriebsart

Schalter	Funktion, Adreßbereich			Betrieb als Rechner	Betrieb als Erweit.
S1	RAM, 0000-03FF (belegt 0000-1FFF)		IC 4,5	ein	
S2	RAM, 2000-23FF (belegt 2000-3FFF)		IC 4,5		ein
S3	VIA 1, 4000-400F (belegt 4000-5FFF)		IC 1	ein	ein
S4	VIA 2, 6000-600F (belegt 6000-7FFF)		IC 2	ein	ein
S5	EPROM 1, 8800-8FFF	sowie gleichzeitig 9800-9FFF	IC 12		ein
	EPROM 2, 8000-87FF	sowie gleichzeitig 9000-97FF	IC 6		ein
S6	EPROM 1, A800-AFFF	sowie gleichzeitig B800-BFFF	IC 12		
	EPROM 2, A000-A7FF	sowie gleichzeitig B000-B7FF	IC 6		
S7	EPROM 1, C800-CFFF	sowie gleichzeitig D800-DFFF	IC 12		
	EPROM 2, C000-C7FF	sowie gleichzeitig D000-D7FF	IC 6		
S8	EPROM 1, E800-EFFF	sowie gleichzeitig F800-FFFF	IC 12	ein	
	EPROM 2, E000-E7FF	sowie gleichzeitig F000-F7FF	IC 6	ein	

brücken an den jeweiligen Rechner angepaßt werden. Dadurch ist es auch möglich, das Programm zunächst mit einem „Mutterrechner" zu entwickeln und erst dann die Erweiterungsplatine mit einer eigenen CPU zu versehen, so daß sie als selbständiger Rechner betrieben werden kann. Wie der EMUF enthält auch sie eine kleine Lochrasterfläche, auf der anwenderspezifische Zusatzschaltungen untergebracht werden können.

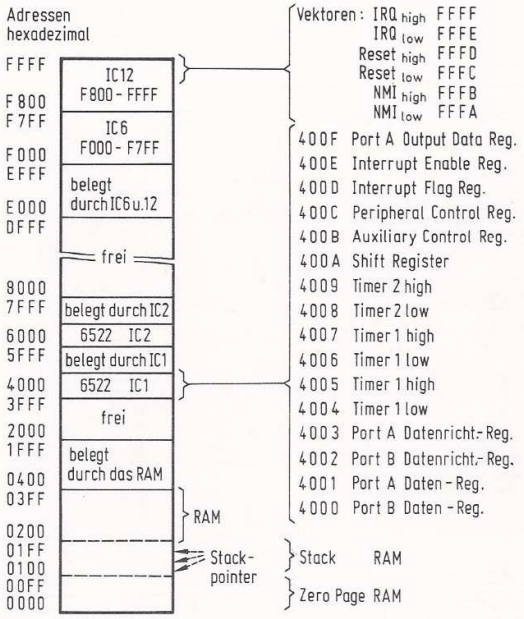


Abb. 5.2.3 Adressenbelegung des 6502-Einplatinencomputers

Ein 5-V-Spannungsregler-IC ist bereits auf der Platine untergebracht, so daß sie wahlweise mit 5 V oder mit unstabilisierten 8...12 V versorgt werden kann. Zu Kontrollzwecken sind auch drei Leuchtdioden untergebracht: D 3 leuchtet bei anliegender Betriebsspannung, D 4, wenn PB 7 des VIA-Bausteins 1 auf High-Pegel ist, D 5 leuchtet, wenn PB 7 des VIA-Bausteins 2 auf High-Pegel ist.

Zwei EPROM-ICs vom Typ 2716 ergeben zusammen eine Festwertspeicher-Kapazität von 4 KByte. 1 KByte RAM wurde mit zwei Bausteinen 2114 realisiert (*Tab 5.2* und *Abb. 5.2.2, 5.2.3*).

Die beiden VIA-Bausteine 6522 enthalten zusammen vier programmierbare Timer, zwei Schieberegister (z. B. für serielle Ein-/Ausgabe), vier 8-Bit-Ports und zusätzlich insgesamt 8 Handshake-Leitungen, die bedarfsweise der Datenübernahmesteuerung für die Ein-/Ausgabe-Bausteine dienen (CA, CB).

Verwendet man die Platine nur zur Erweiterung eines Rechners, so entfallen die CPU 6502, der Reset-Timer NE 555 sowie das IC 7400. Bei IC 10 (7433) muß Pin 10 abgeklemmt werden. Schließlich müssen ihre Bausteine auf Adressenbereiche gelegt werden, die vom Mutterrechner selbst nicht verwendet werden. Das kann z. B. mit kleinen Schaltergruppen geschehen, die in IC-ähnlichen Gehäusen lieferbar sind (DIL-Schalter).

5.3 Einfache 6809-Schaltung

Schließlich betrachten wir noch eine Platine, die mit dem Prozessor 6809 arbeitet (*Abb. 5.3.1*). Dieser Prozessor ist insofern interessant, da er intern weitgehend mit 16-Bit-

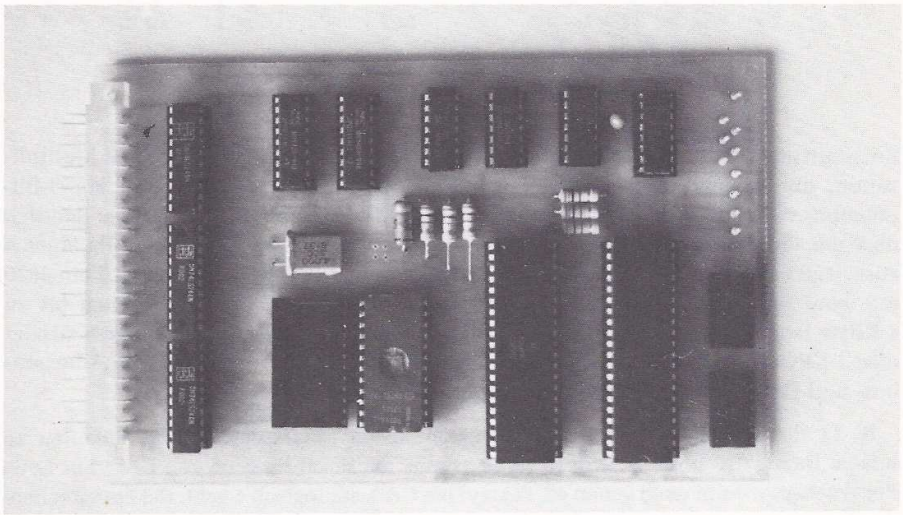


Abb. 5.3.1 So sieht die 6809-Europakarte aus

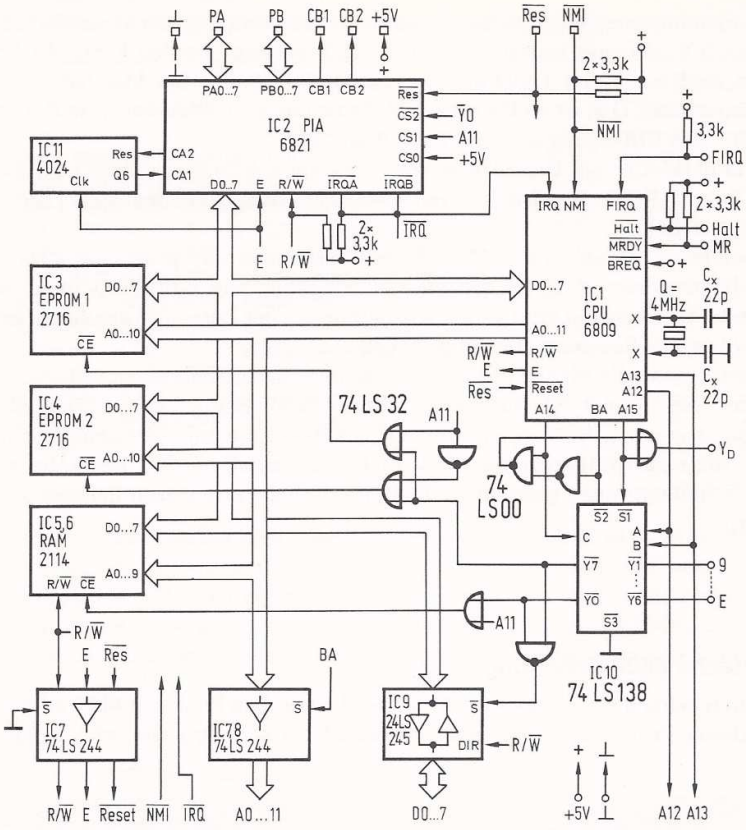


Abb. 5.3.2 Schaltung des 6809-Einplatinencomputers

Registern arbeitet und auch zahlreiche 16-Bit-Operationen, z. B. Addition und Multiplikation, durchführen kann. Seine Leistungsfähigkeit erreicht zwar, weil ein 8-Bit-Datenbus verwendet wird, nicht die von „echten" 16-Bit-Computern, liegt aber deutlich über der der meisten anderen 8-Bit-CPUs. Die Schaltung ist verhältnismäßig einfach ausgelegt; es wird nur ein einziger Ein-/Ausgabe-Baustein verwendet, nämlich ein 6820 mit zwei 8-Bit-Ports (*Abb. 5.3.2*). Zwei EPROM-Bausteine 2716 ermöglichen bis zu 4 KByte lange Anwenderprogramme. Die beiden RAM-Bausteine 2114 besitzen zusammen 1 KByte Arbeitsspeicher-Kapazität. Für externe Erweiterungen sind die Adressenbus- und Datenbusleitungen über Puffer-ICs (IC 7, 8, 9) nach außen geführt.

IC 11 dient dazu, für Testzwecke Programme im Einzelschrittbetrieb ablaufen zu lassen. Dazu wird über CA 2 des PIA-Bausteins der Reset-Eingang von IC 11 auf Low-Pegel gelegt, worauf nach genau 32 Taktzyklen CA 1 auf logisch 1 geht. Bei entsprechender Programmierung der PIA löst diese Low-High-Flanke einen Interrupt über IRQA

aus, wodurch sämtliche CPU-Register auf den Stack gerettet werden und der momentane Befehl unterbrochen wird. Die Interrupt-Routine kann man nun so programmieren, daß zunächst CA 2 auf high gesetzt wird und dann z. B. alle Registerinhalte auf einem Terminal oder Display angezeigt werden. Wird dann CA 2 wieder auf Low gesetzt, so verbleiben wiederum 32 Taktzyklen bis zum nächsten Interrupt. Durch Verzögern mit einigen Befehlen (NOP = No Operation) kann man erreichen, daß nach der Rückkehr aus der Interrupt-Routine gerade der nächste Befehl des Hauptprogramms begonnen wird, das vom Interrupt unterbrochen wurde. Dieser Befehl wird aber sofort wieder unterbrochen, worauf sich der gesamte Ablauf wiederholt.

Die einzelnen Bausteine auf der 6809-Platine sind folgenden Adressenbereichen zugeordnet:

EPROM 2	F800...FFFF
EPROM 1	F000...F7FF
RAM	8000...83FF
PIA	8800...8803

Da beim RAM keine vollständige Adressendecodierung stattfindet, spiegelt es sich auch in den Bereich 8400...87FF.

6 Häufig gebrauchte Routinen

Im folgenden werden einige Programmstücke vorgestellt, die in zahlreichen unterschiedlichen Anwendungen nützlich sind. Sie werden hier im Assembler-Format abgedruckt und sind für den Einplatinen-Computer EMUF ausgelegt. Da dessen Adressenbelegung bereits ausführlich beschrieben wurde, ist es kein Problem, diese Routinen auf anderen Computern mit den CPU-Typen 6502 oder 6504 laufen zu lassen. Das Umschreiben auf Computer mit anderen Mikroprozessoren ist schon schwieriger, ist aber durchaus machbar, wenn man sich etwas mit dem Befehlssatz des 6502 vertraut gemacht hat. Dabei werden sich natürlich durch die anderen Befehlsausführungszeiten auch die für einzelne Routinen angegebenen Laufzeiten ändern, und es sind Korrekturen in jenen Formeln erforderlich, die zeit- und frequenzabhängige Parameter enthalten.

6.1 Zeitmessung

Ein häufig wiederkehrendes Programmierproblem ist es, festzustellen, wie lange ein bestimmter Eingang des Peripherie-Bausteines auf logisch 1 oder 0 bleibt (*Abb. 6.1.1*). Diese Zeitmessung kann, je nachdem, ob der Computer über einen programmierbaren Zeitgeber verfügt, entweder mit diesem oder, wenn dies nicht der Fall ist, mit einer Programmschleife geschehen.

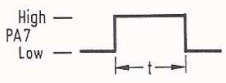


Abb. 6.1.1 Aufgabenstellung: Die Impulsdauer t soll gemessen werden

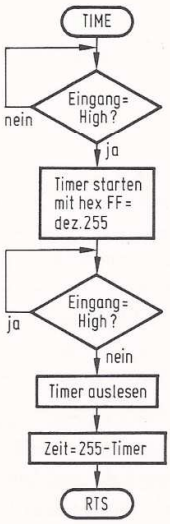


Abb. 6.1.2 Flußdiagramm zur Zeitmessung mit Timer

Verwendet man einen programmierbaren Zeitgeber (Timer), so ist der Software-Aufwand am geringsten (*Abb. 6.1.2*). Es muß lediglich der Timer gestartet werden, und das Programm wartet so lange, bis die Peripherieleitung den erwarteten Zustand angenommen hat. Dann wird das Timer-Register ausgelesen, und, weil der Zeitgeber stets abwärts zählt, komplementiert. Im Akkumulator der CPU steht dann der gesuchte Zeitwert in Vielfachen von 1024 Mikrosekunden. Es wäre natürlich auch möglich gewesen, den Timer im 6532 nicht für Schritte von 1024 Mikrosekunden, sondern für 1-, für 8- oder für 64-Mikrosekunden-Schritte zu initialisieren. Das hängt in erster Linie davon ab, in welchem Bereich die zu messenden Zeitwerte liegen. *Abb. 6.1.3* zeigt die EMUF-Systemadressen; sie benötigt der Assembler auch für die anderen hier vorgestellten Programmstücke. Und aus *Abb. 6.1.4* geht hervor, wie das Assemblerlisting zur Zeitmessung mit Timer aussieht.

Will man Zeiten mit Hilfe einer Programmierschleife messen, so zählt man zweckmäßig ein Register der CPU, von Null beginnend, hoch. Will man Zeiten von mehreren Millisekunden oder noch mehr erfassen, so genügt dafür ein 8-Bit-Register nicht, weil es wegen der relativ geringen Befehlsausführungszeiten innerhalb der Schleife recht schnell „überlaufen" würde.

```
0000          ;EMUF-ADRESSEN
0000                *=$800
0800 PA             *=*+1
0801 PAD            *=*+1
0802 PB             *=*+1
0803 PBD            *=*+$11
0814 T1             *=*+1
0815 T8             *=*+1
0816 T64            *=*+1
0817 TK             *=*+5
081C TI1            *=*+1
081D TI8            *=*+1
081E TI64           *=*+1
081F TIK
```

Abb. 6.1.3 Die EMUF-Systemadressen als Assemblerlisting. Auf diese Adressen beziehen sich die folgenden Routinen

```
081F                *=$840        ;BEISP-ADR.
0840
0840          ;ZEITMESSUNG M.TIMER
0840          ;EINGANG = PA7
0840 TIME     2C0008 BIT PA       ;PA7 HIGH?
0843          10FB   BPL TIME     ;NEIN
0845          A9FF   LDA £$FF     ;TIMER
0847          8D1708 STA TK       ;STARTEN
084A TIME1    2C0008 BIT PA       ;PA7 HIGH?
084D          30FB   BMI TIME1    ;JA
084F          AD1608 LDA T64      ;TIMER LESEN
0852          49FF   EOR £$FF     ;KOMPLEMENT
0854          60     RTS          ;ZEIT IN AKKU
```

Abb. 6.1.4 Assemblerlisting zur Impulsdauer-Messung mit Timer

Dann ist es erforderlich, mehrere Register oder Speicherzellen zu verwenden (*Abb. 6.1.5, 6.1.6*), wobei die höherwertige Zelle immer dann um 1 weitergezählt wird, wenn die nächstniederwertige von hexadezimal FF auf 00 springt.

Will man den Zeitwert dezimal zählen, um sich später die Umwandlung des Wertes in das Dezimalformat z. B. für eine Digitalanzeige zu sparen, so ist das bei der CPU 6502/6504 besonders einfach mit dem Befehl SED (Set Decimal Mode) möglich. Die Operationen ADC (Addition) und SBC (Subtraktion) erfolgen dann im dezimalen BCD-Format (Binary Coded Decimal). Die Befehlsfolge müßte dann folgendermaßen aussehen:

```
TIME    BIT PA      ;Warten, bis
        BPL TIME    ;PA7 = High
        SED         ;Dezimalmodus
        LDA £0      ;Akku = 0
        TAX         ;X-Reg. = 0
TIME1   CLC         ;Carry löschen
        BIT PA      ;PA7 = High?
        BPL TIME2   ;Nicht mehr
        ADC £1      ;Akku hochzählen
        BCC TIME1   ;Kein Übertrag
        INX         ;Übertrag in X
        BNE TIME1   ;Kein Überlauf
TIME2   RTS         ;LSB = Akku, MSB = X[1])
```

```
0855        ;ZEITMESSG.M.SCHLEIFE
0855        ;EINGANG=PA7
0855 TIMS   2C0008 BIT PA      ;PA7 HIGH?
0858        10E6   BPL TIME    ;NEIN
085A        A000   LDY £0      ;REGISTER
085C        A200   LDX £0      ;LOESCHEN
085E TIMS1  2C0008 BIT PA      ;PA7 HIGH?
0861        1006   BPL TIMS2   ;NICHT MEHR
0863        E8     INX
0864        D0F8   BNE TIMS1   ;ZAEHL-
0866        C8     INY         ;SCHLEIFE
0867        D0F5   BNE TIMS1   ;ZEIT:LSB IN X,
0869 TIMS2  60     RTS         ;MSB IN Y
```

Abb. 6.1.6 Assemblerlisting zur Zeitmessung mit Zählschleifen

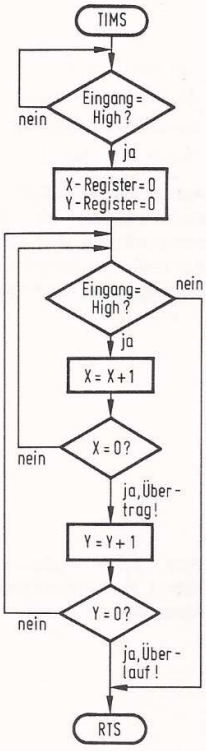


Abb. 6.1.5 Flußdiagramm zur Zeitmessung mit Zählschleifen

[1]) LSB = Least Significant Byte, niederwertiges Byte;
MSB = Most Significant Byte, höherwertiges Byte.
Beide ergeben zusammen einen 16-Bit-Wert als Ergebnis.

Da das höherwertige Byte (MSB in X) normal inkrementiert wird, ist dafür die automatische Dezimalkorrektur durch SED nicht wirksam. Deshalb arbeitet die Routine für MSB-Werte ab 09 nicht mehr korrekt dezimal (der nächste Wert wäre 0A). Will man das vermeiden, so muß man irgendeine Speicherzelle statt X verwenden und diese unter Verwendung des Akkus (dessen Inhalt „gerettet" werden muß, um das LSB nicht zu verlieren) mit dem ADC-Befehl um 1 erhöht werden, sobald beim Hochzählen des LSB ein Übertrag auftritt.

Zum Vergleich eine kleine Gegenüberstellung, wie ab der „kritischen Zahl" 09 binär und im BCD-Format weitergezählt wird. Betrachtet man das ganze Byte, das eine BCD-Zahl enthält, als 8-Bit-Dezimalzahl, so ergeben sich völlig irreführende Werte. Vielmehr muß man sich das Byte in zwei 4-Bit-Hälften (Nibbles) aufgespalten denken, von denen jede eine Dezimalziffer enthält.

Binär-Zählweise	Dezimal-zahl	Hexa-dezimal	BCD-Zählweise	BCD-Zahl	Byte als Dezimalzahl
0000 1000	8	08	0000 1000	08	8
0000 1001	9	09	0000 1001	09	9
0000 1010	10	0A	0001 0000	10	16
0000 1011	11	0B	0001 0001	11	17
0000 1100	12	0C	0001 0010	12	18
0000 1101	13	0D	0001 0011	13	19
0000 1110	14	0E	0001 0100	14	20
0000 1111	15	0F	0001 0101	15	21
0001 0000	16	10	0001 0110	16	22
.					
. usw.					
.					
0110 0011	99	63	1001 1001	99	153
0110 0100	100	64	0000 0000	00	0

Beim Sprung von 99 auf 100, d. h. wenn die BCD-Zahl 99 durch dezimale Addition von 1 erhöht wird, wird der Akkuinhalt wieder Null, allerdings wird das Überlauf-Flag in der CPU (Carry-Flag) gesetzt. Bei anderen CPU-Typen läßt sich eine BCD-Addition erreichen, indem man dem Additionsbefehl den Befehl DAC (Decimal Adjust) nachstellt, z. B. bei 8080, 8085 und Z80.

Selbstverständlich funktioniert das dezimale Zählen mit dem ADC-Befehl deutlich langsamer als das binäre Zählen durch einfaches Inkrementieren von Registern. In unserem Beispiellisting, in dem ja binär gezählt wird, ergibt sich als 16-Bit-Ergebnis in Y und X etwa:

$$y,x = \frac{1}{15{,}016\ \mu s}$$

Die Konstante 15,016 entsteht dadurch, daß man mit Hilfe einer Befehlszyklen-Tabelle, wie sie im 6502-Programmierhandbuch zu finden ist, die Ausführungszeiten der Befehle in den zwei verschachtelten Schleifen bestimmt und addiert. Da die Y-Schleife nur alle 256 Mal ausgeführt wird und vier Zyklen benötigt, während die X-Schleife immer ausgeführt wird und 15 Zyklen beansprucht, ergibt sich als Konstante der Wert $15\ \mu s + {}^{1}/_{256}\ \mu s = 15{,}016\ \mu s$.

6.2 Tonerzeugung

Will man einen im Hörbereich liegenden Ton ohne großen Hardware-Aufwand erzeugen, so läßt sich dies sehr einfach mit einer Programmschleife machen (*Abb. 6.2.1*). Dabei wird an einem beliebigen Ausgangsport ein symmetrisches Rechtecksignal erzeugt. Seine Frequenz läßt sich in einem weiten Bereich durch Ändern einer einzigen Speicherzelle einstellen. Unser kleines Beispielprogramm erzeugt immer dann die gewünschte Ausgangsfrequenz am Port PA6, wenn der Port PA7 auf High-Pegel liegt. Sobald PA7 auf Low-Pegel geht, ist das Programm zu Ende, und es erfolgt ein Rücksprung aus der Routine mit RTS.

Es ist leicht einzusehen, daß sich die einfache Routine in *Abb. 6.2.2* nicht für sehr niedrige Frequenzen eignet, da nur ein einziges Byte als Schleifenzähler verwendet wird. Dieser Zähler dient der Verzögerung um eine halbe Periodendauer des Ausgangssignals. Wollte man noch tiefere Töne erzeugen, so wäre es erforderlich, entweder die Ausführungszeit der Schleife z. B. durch Einfügen von NOP-Befehlen zu verlängern oder aber zwei ineinander verschachtelte Zählschleifen zu verwenden, indem man noch das Y-Register hinzuzieht.

Schließlich sei noch auf die Möglichkeit verwiesen, den programmierbaren Zeitgeber (Timer) im Baustein 6532 zu verwenden. Dann kann man auf umständliche Zählschleifen verzichten und sogar eine Periodendauer bis zu rund einer halben Sekunde erreichen. Es liegt im Prinzip der Software-Tonerzeugung begründet, daß es in Wahrheit nicht möglich ist, jede beliebige Frequenz zu erzeugen. Denn eine Änderung des frequenzbestimmenden Bytes um nur einen Schritt hat ja bereits eine Frequenzänderung von mehreren Hz zur Folge, und die dazwischen liegenden Frequenzwerte lassen

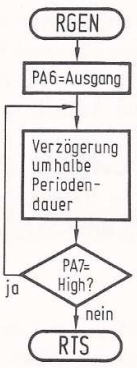


Abb. 6.2.1 Flußdiagramm zum Rechteck-Generator

```
081F                    *=$C80      ;BEISP-ADR
0C80
0C80            ;RECHTECKGENERATOR
0C80            ;PA7=HIGH:EIN
0C80            ;PA7=LOW:AUS
0C80            ;AUSGANG: PA6
0C80 FREQ               =180        ;BEISP-FREQ
0C80 MASKE              =%01000000  ;PA6
0C80 RGEN   A940        LDA £MASKE
0C82        8D0108      STA PAD
0C85 RGEN1  A2B4        LDX £FREQ   ;HALBE
0C87 RGEN2  CA          DEX         ;PERIODEN-
0C88        D0FD        BNE RGEN2   ;DAUER
0C8A        AD0008      LDA PA      ;AUSGANG
0C8D        4940        EOR £MASKE  ;UMSCHALTEN
0C8F        8D0008      STA PA
0C92        2C0008      BIT PA      ;PA7=HIGH?
0C95        30EE        BMI RGEN1   ;JA
0C97        60          RTS         ;NEIN
```

Abb. 6.2.2 Rechteckgenerator-Assemblerlisting

sich nicht programmieren. Diesen kleinen Nachteil muß man bei der Software-Tonerzeugung jedoch hinnehmen. Die Formel für die erzeugte Frequenz f lautet:

$$f = \frac{10^6}{54 + 4\,FREQ + 6\,(FREQ - 1)}\ Hz$$

6.3 Frequenzmessung

Wenn man keine allzu hohen Anforderungen an die Auflösung stellt, so ist es auch mit einfachen Mitteln möglich, eine Frequenzmessung per Software zu realisieren (*Abb. 6.3.1*). Dazu muß man lediglich die als Rechtecksignal vorliegende Eingangsspannung dem Port PA7 zuleiten, auf den unser Beispielprogramm in *Abb. 6.3.2* ausgelegt ist. Es verwendet den Timer im 6532-Baustein und liefert als Ergebnis einen der Eingangsfrequenz proportionalen 16-Bit-Wert in den Registern Y (höherwertiges Byte) und X (niederwertiges Byte).

Das Programm arbeitet folgendermaßen: Zunächst werden die Zählregister X und Y auf Null rückgesetzt. Dann schreibt man in den Timer die Meßdauer von 100 Millise-

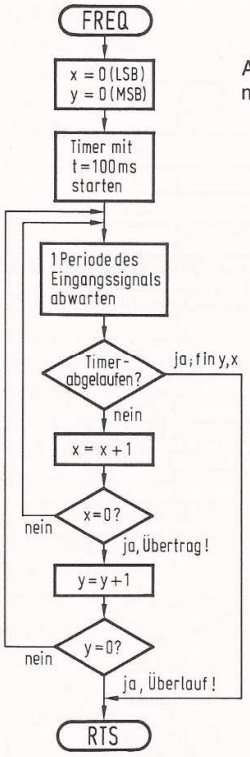


Abb. 6.3.1 Flußdiagramm des Programms zur Frequenzmessung

```
081F                    *=$C80      ;BEISP-ADR
0C80
0C80            ;FREQUENZMESSUNG
0C80            ;EINGANG=PA7
0C80 FREQ   A200    LDX £0      ;X=LSB
0C82        A000    LDY £0      ;Y=MSB
0C84        A962    LDA £98     ;TIMER:
0C86        8D1708  STA TK      ;100 MS
0C89 FREQ1  2C0008  BIT PA      ;WARTEN
0C8C        10FB    BPL FREQ1   ;AUF HIGH
0C8E FREQ2  2C0008  BIT PA      ;WARTEN
0C91        30FB    BMI FREQ2   ;AUF LOW
0C93        2C1708  BIT TK      ;100 MS
0C96        3006    BMI FREQ3   ;VORBEI
0C98        E8      INX         ;PERIODEN
0C99        D0EE    BNE FREQ1   ;ZAEHLEN
0C9B        C8      INY         ;UEBERLAUF:
0C9C        D0EB    BNE FREQ1   ;X=0,Y=0
0C9E FREQ3  60      RTS         ;F IN Y,X
```

Abb. 6.3.2 Messung der Frequenz am Eingang PA7

kunden. In zwei verschalteten Schleifen werden nun X und Y nach jeder vollständigen Periode des Eingangsignals um 1 hochgezählt. Dieser Vorgang wiederholt sich so lange, bis die vorprogrammierte Zeit von 100 Millisekunden vorbei ist. Da man als Frequenzeinheit (Hz) die Zahl der Perioden pro Sekunde verwendet, stellt der 16-Bit-Wert in Y und X dann $\frac{1}{10}$ der gemessenen Frequenz in binärer Form dar.

6.4 Sukzessive Approximation

Da ein Analog/Digital-Wandler im Vergleich zu einem Digital/Analog-Wandler relativ teuer ist, bevorzugt man in vielen Anwendungsfällen das Verfahren der sukzessiven Approximation, um einen D/A-Wandler per Programm zu einem A/D-Wandler umzufunktionieren (*Abb. 6.4.1, 6.4.2*). Dieses Verfahren wurde schon weiter vorn beschrieben, und hier folgt nun die softwaremäßige Realisation. Ein 8-Bit-Port des Mikrocomputers wird dabei für den Anschluß des 8-Bit-D/A-Wandlers benötigt, und eine weitere Portleitung, hier PB3, dient dazu, um dem Programm in *Abb. 6.4.3* mitzuteilen, ob die vom D/A-Wandler erzeugte Spannung größer oder kleiner als die zu messende Eingangsspannung ist. Das Programm probiert dann alle 8 Bits durch, um die D/A-Wandler-Ausgangsspannung möglichst gut an die Eingangsspannung anzunähern (Approximation), und benötigt dafür rund 0,3 ms.

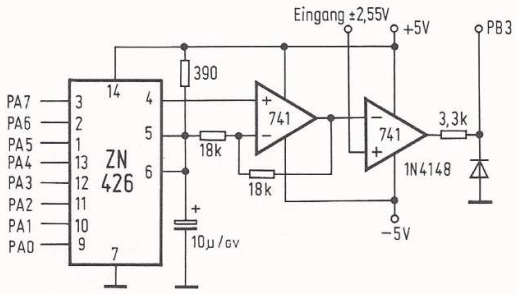


Abb. 6.4.1 Schaltung zur sukzessiven Approximation

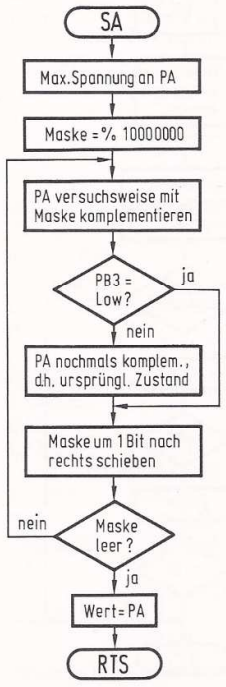


Abb. 6.4.2 Flußdiagramm zur A/D-Wandlung

```
0C98          ;SUKZESSIVE APPROXIMATION
0C98 SA   A9FF    LDA £$FF     ;PA=AUSG
0C9A      8D0108  STA PAD      ;START MIT
0C9D      8D0008  STA PA       ;2.55 V
0CA0      A940    LDA £%1000000   ;MASKE
0CA2 SA1  AA      TAX          ;RETTEN
0CA3      4D0008  EOR PA       ;VERSUCHSW.
0CA6      8D0008  STA PA       ;1 BIT KOMPL.
0CA9      A908    LDA £8       ;PB3=EING
0CAB      2C0208  BIT PB       ;KOMPL.OK?
0CAE      F007    BEQ SA2      ;NEIN
0CB0      8A      TXA          ;MASKE
0CB1      4D0008  EOR PA
0CB4      8D0008  STA PA       ;KOMPL.
0CB7 SA2  8A      TXA
0CB8      4A      LSR A
0CB9      90E7    BCC SA1
0CBB      AD0008  LDA PA       ;WERT
0CBE      60      RTS          ;IN AKKU
```

Abb. 6.4.3 Assemblerlisting für die sukzessiven Approximation

Unsere Beispielschaltung ist für einen Meßbereich von −2,55...+2,55 V ausgelegt. Die Ausgangsspannung des D/A-Wandlers beträgt 0...2,55 V. Der erste Operationsverstärker (741) dient dazu, um den Spannungshub des A/D-Wandlers zu verdoppeln, und der zweite dient als Komparator für die Wandler- und Eingangsspannung. Da die beiden Operationsverstärker mit einer symmetrischen Spannung (+5 V und −5 V) betrieben werden, ist eine Diode erforderlich, um den Porteingang PB 3 vor Beschädigungen durch eine negative Spannung zu schützen.

6.5 Interrupt-Uhr

Zahlreiche Anwendungen erfordern es, daß der Mikrocomputer jederzeit auf die aktuelle Zeit zugreifen kann. Dazu gibt es zwei Realisationsmöglichkeiten. Die erste ist, ein spezielles Uhren-IC zu verwenden und es über einen Peripherie-Baustein mit dem Mikrocomputer zu verbinden. Diese Lösung hat den Vorteil, daß das Uhren-IC mit relativ wenig Stromverbrauch auch bei ausgeschaltetem Mikrocomputer weiterläuft, so daß beim Einschalten des Computers die Uhrzeit nicht neu eingegeben werden muß. Der Nachteil dieser Lösung ist natürlich der nötige Hardware-Aufwand.

Hier wird eine andere Lösung vorgestellt, die keinerlei zusätzliche Hardware verwendet. Statt dessen unterbricht der Mikrocomputer im Abstand von ¼ Sekunde sein gerade laufendes Hauptprogramm und springt, vom Interrupt-Timer des 6532-Bausteins gesteuert, in ein Interrupt-Programm. Diese Routine stellt die interne Uhr, die aus einigen Speicherzellen besteht, dann um genau ¼ Sekunde weiter. Wenn das geschehen ist, kehrt der Prozessor in sein normales Hauptprogramm zurück, als wäre nichts geschehen (*Abb. 6.5.1*).

Die Routine in *Abb. 6.5.2* ist so ausgelegt, daß für die Sekunden (0...59), die Minuten (0...59), die Stunden (0...23) und die Tage (0...100) je eine Speicherzelle in dezimalem

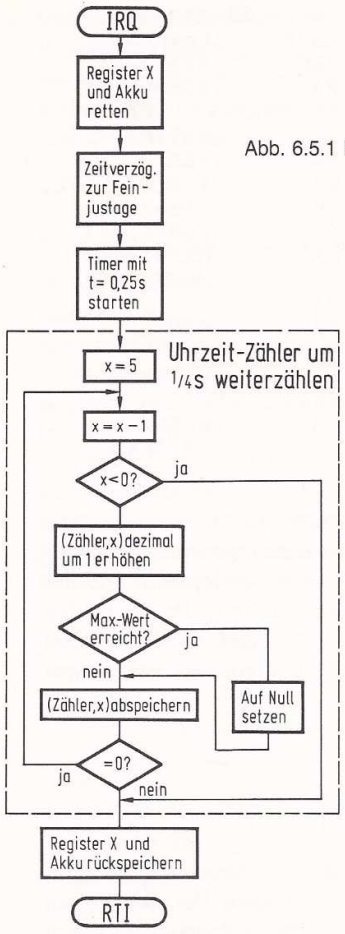


Abb. 6.5.1 Flußdiagramm der Interrupt-Routine für die Software-Uhr

```
                        *=0             ;ZERO-PAGE
0000 TAG                *=*+1           ;ZAEHLER
0001 STD                *=*+1
0002 MIN                *=*+1
0003 SEK                *=*+1
0004 VSC                *=*+1           ;1/4 SEK
0005                    *=$C80          ;BEISP-ADR
0C80
0C80        ;INTERRUPT-UHR
0C80 IRQ    48      PHA                 ;REGISTER
0C81        8A      TXA                 ;RETTEN
0C82        48      PHA
0C83        A210    LDX £16             ;FEIN-
0C85 IRQ1   CA      DEX                 ;JUSTAGE
0C86        D0FD    BNE IRQ1
0C88        A9F4    LDA £$F4            ;TIMER
0C8A        8D1F08  STA TIK             ;STARTEN
0C8D        F8      SED                 ;DEZIMAL-
0C8E        A205    LDX £5              ;MODUS
0C90 IRQ2   CA      DEX
0C91        3010    BMI IRQ4
0C93        B500    LDA TAG,X           ;ZAEHLER
0C95        18      CLC                 ;JEW.UM 1
0C96        6901    ADC £1              ;HOCH-
0C98        DDA80C  CMP TAB,X           ;ZAEHLEN
0C9B        9002    BCC IRQ3            ;BEI MAX-WERT
0C9D        A900    LDA £0              ;AUF 0 SETZEN
0C9F IRQ3   9500    STA TAG,X
0CA1        F0ED    BEQ IRQ2
0CA3 IRQ4   D8      CLD
0CA4        68      PLA                 ;REGISTER
0CA5        AA      TAX                 ;RUECK-
0CA6        68      PLA                 ;SPEICHERN
0CA7        40      RTI
0CA8 TAB    FF      .BYT $FF            ;TAG-MAX
0CA9        24      .BYT $24            ;STD-MAX
0CAA        60      .BYT $60            ;MIN-MAX
0CAB        60      .BYT $60            ;SEK-MAX
0CAC        04      .BYT 4              ;VSC-MAX
0CAD                *=$FFE              ;IRQ-VEKTOR
0FFE        800C    .WOR IRQ
```

Abb. 6.5.2 Listing der Software-Uhr-Interruptroutine

Format (BCD) zur Verfügung steht. Eine zusätzliche Speicherzelle dient zum Abzählen der Viertelsekunden. Der Programmierer hat noch dafür zu sorgen, daß am Anfang eines Hauptprogramms die Routine ein erstes Mal angesprungen wird, denn erst dann kann sie sich selbst immer wieder neu starten. Dieses erste Anspringen ist z. B. durch folgende Befehlsfolge zu erreichen:

```
LDA #1          ;Erster Interrupt nach 1 ms
STA TIK         ;Timer-Start
CLI             ;Interrupt-Eingang IRQ der CPU freigeben
...             ;Weiter im Hauptprogramm
```

Ebenso muß natürlich dafür gesorgt werden, daß beim Einschalten des Mikrocomputers der Benutzer Gelegenheit erhält, die aktuelle Uhrzeit selbst einzugeben. Das kann beispielsweise über ein Tastenfeld mit Dezimalziffern geschehen. Wenn gewünscht wird, daß die Tage nicht bis 99 gezählt werden, sondern nur bis 6, etwa zur Anzeige des Wochentags, so ist das Byte an der Adresse 0CA8 (FF) durch 07 zu ersetzen.

6.6 D/A-Wandler

Wie die gerade beschriebene Interrupt-Uhr arbeitet auch der D/A-Wandler, der nach dem Prinzip der Impulsdauer-Variation arbeitet, mit dem Interrupt-Timer im 6532. Wie bereits weiter vorn beschrieben, ist bei diesem Verfahren lediglich noch ein aus einem Widerstand und einem Kondensator bestehender Tiefpaß erforderlich (*Abb. 6.6.1*). Ein bei unserem Beispielprogramm in der Speicherzelle 0000 (symbolischer Name U) stehender 8-Bit-Wert stellt die auszugebende Spannung dar. Für Werte von 0...255 wird

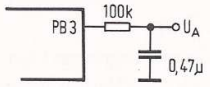


Abb. 6.6.1 Externe Beschaltung für den Software-D/A-Wandler

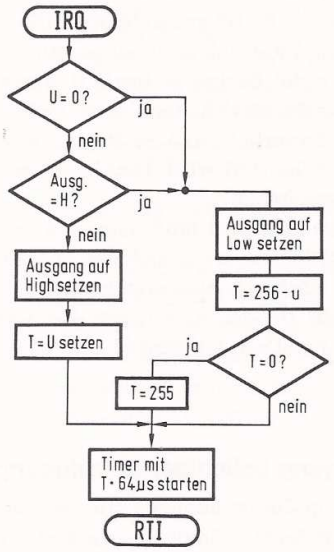


Abb. 6.6.2 Flußdiagramm der D/A-Interrupt-Routine

```
081F          ;D/A-WANDLER
081F          ;PB3=AUSGANG
081F                *=0          ;ZERO PAGE
0000 U              *=$C80       ;SP.-WERT
0C80 IRQ    48      PHA
0C81        A500    LDA U        ;U=0?
0C83        F013    BEQ HI       ;JA
0C85        AD0208  LDA PB
0C88        2908    AND £8       ;PB3=H?
0C8A        D00C    BNE HI       ;JA
0C8C        AD0208  LDA PB
0C8F        0908    ORA £8       ;NEIN,AUF H
0C91        8D0208  STA PB       ;SETZEN
0C94        A500    LDA U        ;T=T1
0C96        D011    BNE TI       ;SPRUNG
0C98 HI     AD0208  LDA PB
0C9B        29F7    AND £$F7     ;PB3=L
0C9D        8D0208  STA PB
0CA0        A900    LDA £0       ;T2=256
0CA2        38      SEC          ;T=
0CA3        E500    SBC U        ;T2-T1
0CA5        D002    BNE TI
0CA7        A9FF    LDA £$FF     ;VERMEIDE
0CA9 TI     8D1E08  STA TI64     ;T=0
0CAC        68      PLA
0CAD        40      RTI
0CAE                *=$FFE       ;IRQ-VEKTOR
0FFE        800C    .WOR IRQ
```

Abb. 6.6.3 Assemblerlisting der Interrupt-Routine zur Pulsbreiten-Variation

proportional eine Ausgangsspannung von 0...5 V erzeugt. Dies geschieht dadurch, daß am Ausgang PB 3 ein Rechtecksignal mit veränderlicher Impulsdauer und einer Frequenz von rund 60 Hz generiert wird (*Abb. 6.6.2, 6.6.3*).

Die Interrupt-Routine läuft sozusagen (wie das Uhrenprogramm) im „Hintergrund", während das gleichzeitig laufende Hauptprogramm sich lediglich darum zu kümmern hat, die auszugebende Spannung in die Speicherzelle U zu schreiben. Auch hier ist es natürlich erforderlich, daß zu Beginn des Hauptprogramms der Interrupt-Timer ein erstes Mal initialisiert wird. Dies kann mit der gleichen Befehlsfolge wie beim Uhrenprogramm geschehen.

Den Programmablauf muß man sich so vorstellen, daß der Timer immer dann ein Interrupt produziert, wenn sich der Ausgangszustand an PB3 ändern soll, also bei jeder Flanke des Rechteck-Ausgangssignals. Die Interrupt-Routine schaltet dann PB3 von logisch 1 auf 0 oder umgekehrt. Da dies sehr schnell geschieht, erfolgt die D/A-Umsetzung praktisch unbemerkt vom gleichzeitig laufenden Hauptprogramm.

6.7 Erzeugung beliebiger Schwingungsformen

Bisher war im Zusammenhang mit der Erzeugung von tonfrequenten Ausgangssignalen stets nur von Rechteckimpulsen die Rede. Mit Hilfe eines integrierten D/A-Wandlers ist

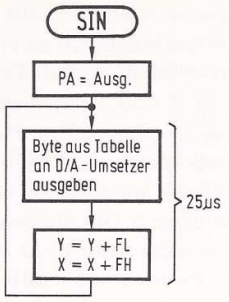


Abb. 6.7.1 Flußdiagramm zur Sinus-Ausgabe mit einstellbarer Frequenz

```
081F                    *=0             ;ZERO PAGE
0000 FL                 *=*+1           ;FREQ-LSB
0001 FH                 *=$C40          ;FREQ-MSB
0C40
0C40                ;SINUSERZEUGUNG
0C40 SIN    A9FF    LDA £$FF
0C42        8D0108  STA PAD
0C45 SIN1   BD000D  LDA TAB,X
0C48        8D0008  STA PA
0C4B        98      TYA
0C4C        6500    ADC FL
0C4E        A8      TAY
0C4F        8A      TXA
0C50        6501    ADC FH
0C52        AA      TAX
0C53        4C450C  JMP SIN1
0C56 TAB            =$D00
```

Abb. 6.7.2 Assemblerlisting für die Ausgabe einer Funktion, deren 256 Werte an den Adressen 0D00...0DFF stehen

```
(_)=0D00 FF FF FF FF FE FE FE FD FD FC FB FA FA F9 F8 F6
(_) 0D10 F5 F4 F3 F1 F0 EE ED EB EA E8 E6 E4 E2 E0 DE DC
( ) 0D20 DA D7 D5 D3 D0 CE CB C9 C6 C4 C1 BE BC B9 B6 B3
( ) 0D30 B0 AD AA A7 A5 A2 9E 9B 98 95 92 8F 8C 89 86 83
( ) 0D40 7F 7C 79 76 73 70 6D 6A 67 64 61 5D 5A 58 55 52
( ) 0D50 4F 4C 49 46 43 41 3E 3B 39 36 34 31 2F 2C 2A 28
( ) 0D60 25 23 21 1F 1D 1B 19 17 15 14 12 11 0F 0E 0C 0B
( ) 0D70 0A 09 07 06 05 05 04 03 02 02 01 01 01 00 00 00
( ) 0D80 00 00 00 00 01 01 01 02 02 03 04 05 05 06 07 09
( ) 0D90 0A 0B 0C 0E 0F 11 12 14 15 17 19 1B 1D 1F 21 23
( ) 0DA0 25 28 2A 2C 2F 31 34 36 39 3B 3E 41 43 46 49 4C
( ) 0DB0 4F 52 55 58 5A 5D 61 64 67 6A 6D 70 73 76 79 7C
( ) 0DC0 7F 83 86 89 8C 8F 92 95 98 9B 9E A2 A5 A7 AA AD
( ) 0DD0 B0 B3 B6 B9 BC BE C1 C4 C6 C9 CB CE D0 D3 D5 D7
( ) 0DE0 DA DC DE E0 E2 E4 E6 E8 EA EB ED EE F0 F1 F3 F4
( ) 0DF0 F5 F6 F8 F9 FA FA FB FC FD FD FE FE FE FF FF FF
```

Abb. 6.7.3 Wertetabelle für eine komplette Sinus-Periode

es jedoch möglich, praktisch jede beliebige Schwingungsform per Software zu erzeugen (*Abb. 6.7.1*).

Dazu ist es notwendig, daß die auszugebende Schwingung als Amplitudentabelle abgespeichert ist und vom Mikrocomputer zyklisch an den D/A-Wandler ausgegeben wird. Das Beispielprogramm in *Abb. 6.7.2* ermöglicht es, eine beliebige Funktion mit einer Frequenz bis zu einigen Kilohertz an einen 8-Bit-D/A-Wandler, der an den Port PA angeschlossen ist, auszugeben. Die in *Abb. 6.7.3* abgedruckte Amplitudentabelle enthält 256 Bytes mit den Sinuswerten genau einer Periode. Die sich ergebende Ausgangsfrequenz läßt sich wie folgt berechnen:

$$f = \frac{F}{25\ \mu s \cdot 256^2} = F \cdot 0{,}61035\ Hz$$

Das Programm funktioniert folgendermaßen: Die Sollfrequenz wird durch eine 16-Bit-Zahl repräsentiert, deren niederwertiges Byte in das Y-Register und das höherwertige in das X-Register übernommen wird. Letzteres wird als Index für die Funktionswerttabelle verwendet. Bei jedem Schleifendurchlauf wird die 16-Bit-Zahl in X und Y um den 16-Bit-Betrag F erhöht, der in den beiden frequenzbestimmenden Speicherzellen FL und FH steht. Bei entsprechend hohen Frequenzen wird die Tabelle dann nicht mehr Byte für Byte abgefragt, sondern es werden um so mehr Tabellenbytes ausgelassen, je höher die gewünschte Frequenz ist. Das führt natürlich bei Frequenzen von einigen Kilohertz dazu, daß die tatsächliche Amplitudenauflösung am Ausgang nicht mehr 8 Bit beträgt, und auf einem angeschlossenen Oszilloskop wird das in Form von deutlichen Stufen im Ausgangssignal sichtbar. Bei niedrigen Frequenzen ergibt sich jedoch die volle Amplitudenauflösung von 8 Bit. Übrigens steht am Ausgang PA7 zusätzlich ein Rechtecksignal zur Verfügung, mit dem man beispielsweise ein Oszilloskop triggern kann.

6.8 Autokorrelation

Die Wirkungsweise von Verfahren zur Tonerkennung, namentlich die Autokorrelation und die Kreuzkorrelation, wurde weiter vorn bereits beschrieben. Die Autokorrelation läßt sich mit einem vergleichsweise geringen Software-Aufwand realisieren. Das Beispielprogramm in *Abb. 6.8* arbeitet mit dem Port PB7 als Eingang. Insgesamt werden zunächst 16 Datenbits seriell eingelesen, wobei eine mit dem X-Register programmierte Verzögerungsschleife die Frequenz bestimmt. Das an der Adresse 0C84 bestehende Byte errechnet sich mit folgender Formel:

$$(0C84) = \frac{10^6}{(f - 176\ \text{Hz}) \cdot 40}$$

In unserem Fall wurde der Dezimalwert 10 (hexadezimal 0A) für eine Frequenz von 1750 Hz programmiert.

Wenn die richtige Frequenz anliegt, steht in den beiden Speicherzellen SPL1 und SPL2 je eine Periode des digitalisierten Eingangssignals. Der zweite Teil der Routine vergleicht nun diese beiden Bytes bitweise auf Übereinstimmung. Im Idealfall sind natürlich die 8 Bits in SPL1 gleich den 8 Bits in SPL2, was acht Übereinstimmungen in der Zelle SUM ergibt. Diese Zelle würde dann um genau 8 erhöht. Wenn sie vor dem Anspringen der Autokorrelations-Routine auf 0 rückgesetzt worden war, steht in ihr die Zahl der übereinstimmenden Bits von SPL1 und SPL2. In der Praxis wird man bei störungsfreien Signalen etwa 7 Übereinstimmungen erzielen, bei verrauschten oder gestörten Signalen dagegen etwa 5 oder 6. Da bei weißem Rauschen als Ergebnis etwa 3...4 auftritt, ist eine sichere Erkennung auch relativ stark gestörter Signale kein Problem. Allerdings hat das Verfahren der Autokorrelation den Nachteil, auch auf Vielfache der Sollfrequenz hereinzufallen, und deshalb gibt man in manchen Fällen der

```
OCA9          PASS 1
OCA9          PASS 2
0000
0000          ;AUTOKORRELATION
0000 PB             =$802         ;PB7=EING.
0000                *=0           ;ZERO PAGE
0000 SUM            *=*+1         ;ERGEBNIS
0001 SPL1           *=*+1         ;BIT-
0002 SPL2           *=$C80        ;MUSTER
OC80
OC80          ;BITMUSTER LESEN
OC80 RD       A010    LDY £16     ;2X8BITS
OC82          78      SEI
OC83 RD1      A20A    LDX £10     ;1750HZ
OC85 RDO      CA      DEX
OC86          DOFD    BNE RDO
OC88          AD0208  LDA PB      ;PB7=EING.
OC8B          OA      ASL A
OC8C          6601    ROR SPL1    ;MUSTER
OC8E          6602    ROR SPL2    ;SPEICHERN
OC90          88      DEY
OC91          DOFO    BNE RD1
OC93          58      CLI
OC94          ;MUSTER VERGLEICHEN
OC94          A501    LDA SPL1    ;MUSTER 1
OC96          FO10    BEQ ERR
OC98          C9FF    CMP £$FF    ;KEIN SIG.
OC9A          FOOC    BEQ ERR
OC9C          4502    EOR SPL2    ;MUSTER 2
OC9E          AOO7    LDY £7      ;8 BITS
OCAO CHK1     6A      ROR A       ;GLEICHE
OCA1          BOO2    BCS CHK2    ;BITS
OCA3          E600    INC SUM     ;ZAEHLEN
OCA5 CHK2     88      DEY
OCA6          10F8    BPL CHK1    ;Z-FLG=1:
OCA8 ERR      60      RTS         ;KEIN SIG.
OCA9                  .END
OCA9           ERRORS= 0000
```

Abb. 6.8 Autokorrelation, hier für 1750 Hz dimensioniert

Kreuzkorrelation den Vorzug. Demgegenüber ist als Vorteil der Autokorrelation zu nennen, daß sie sich für Frequenzen bis zu einigen kHz eignet (in unserem Beispiel von 96 Hz...4,63 kHz).

6.9 Kreuzkorrelation

Bei der Autokorrelation wird praktisch das Eingangssignal um genau eine Periodendauer verzögert mit sich selbst verglichen. Wenn dieses Signal nun sehr stark gestört oder verrauscht ist, wird der Vergleich problematisch, und die Autokorrelation liefert keine zuverlässige Aussage mehr. Demgegenüber arbeitet die Kreuzkorrelation mit

```
                081F        ;KORRELATION MIT
                081F        ;REFERENZ-BITMUSTER
                081F        ;PB7=EINGANG
                081F PER            =32          ;PERIODENZAHL
                081F FREQ           =2           ;1750 HZ
                081F REF            =$FO         ;REFERENZ
                081F LEN            =4           ;SAMPLE-LAENGE
                081F                *=0          ;ZERO PAGE
                0000 REFP           *=*+1
                0001 SUM            *=*+1
                0002 CNT            *=*+1        ;ZAEHLER
                0003 CNT1           *=*+1
                0004 MAX            *=*+1        ;MAX.SUM
                0005 ERGL           *=*+1        ;ERGEBNIS
                0006 ERGH           *=*+1
                0007 SMPL           *=$C80       ;SAMPLES
                0C80 CCOR   A900    LDA £O
                0C82        8505    STA ERGL     ;ALLES
                0C84        8506    STA ERGH     ;LOESCHEN
                0C86        A920    LDA £PER
                0C88        8503    STA CNT1
                0C8A        A9FO    LDA £REF     ;REFERENZ
                0C8C        8500    STA REFP     ;SETZEN
                0C8E        ;BITMUSTER LESEN
                0C8E REC    A904    LDA £LEN
                0C90        OA      ASL A        ;LEN X 8
                0C91        OA      ASL A        ;ERGIBT
                0C92        OA      ASL A        ;BIT-
                0C93        8502    STA CNT      ;ANZAHL
                0C95 REC1   A204    LDX £LEN
                0C97        AD0208  LDA PB
                0C9A        2A      ROL A
                0C9B REC2   3606    ROL SMPL-1,X
                0C9D        CA      DEX
                0C9E        DOFB    BNE REC2
                0CAO        AOO2    LDY £FREQ    ;VER-
                0CA2 REC3   88      DEY          ;ZOEGERUNG
                0CA3        DOFD    BNE REC3
                0CA5        C602    DEC CNT
                0CA7        DOEC    BNE REC1
                0CA9        ;MUSTERVERGLEICH
                0CA9        A908    LDA £8       ;REF. 8MAL
                0CAB        8502    STA CNT      ;SCHIEBEN
                0CAD        4A      LSR A        ;MAX=4
                0CAE        8504    STA MAX
                0CBO COMO   A500    LDA REFP     ;REF.
                0CB2        2A      ROL A        ;ROTIEREN
                0CB3        2600    ROL REFP
                0CB5        A900    LDA £O
                0CB7        8501    STA SUM      ;SAMPLES
                0CB9        A204    LDX £LEN     ;MIT
                0CBB COM1   AOO8    LDY £8       ;REFER.
zu Abb. 6.9     0CBD        B506    LDA SMPL-1,X    ;VERGL.
                0CBF        4500    EOR REFP
```

```
OCC1 COM2  2A     ROL A       ;GLEICHE
OCC2       B002   BCS COM3    ;BITS
OCC4       E601   INC SUM     ;ZAEHLEN
OCC6 COM3  88     DEY
OCC7       D0F8   BNE COM2
OCC9       CA     DEX
OCCA       D0EF   BNE COM1
OCCC       A501   LDA SUM     ;MAXIMAL-
OCCE       C504   CMP MAX     ;WERT
OCD0       9002   BCC COM4    ;SPEICHERN
OCD2       8504   STA MAX
OCD4 COM4  C602   DEC CNT
OCD6       D0D8   BNE COM0
OCD8       A504   LDA MAX     ;MAX.-WERT
OCDA       18     CLC         ;ZUM ER-
OCDB       6505   ADC ERGL    ;GEBNIS
OCDD       8505   STA ERGL    ;ADDIEREN
OCDF       9002   BCC NOCY
OCE1       E606   INC ERGH
OCE3 NOCY  C603   DEC CNT1
OCE5       D0A7   BNE REC
OCE7       60     RTS
```

Abb. 6.9 Kreuzkorrelation mit einstellbarer Bandbreite und Frequenz

einem fest gespeicherten Referenz-Bitmuster, das sozusagen ungestört ist. Deshalb können mit ihr noch schwächere Eingangssignale als mit der Autokorrelation erkannt werden. Ein weiterer Vorteil ist, daß sie nicht auf Harmonische, also auf Vielfache der Sollfrequenz hereinfällt. Aus unserem Beispielprogramm in *Abb. 6.9* geht allerdings auch hervor, daß der Software-Aufwand wesentlich höher ist.

Als Referenz-Bitmuster wird hier 11110000 verwendet, hexadezimal F0. Dies entspricht genau einer digitalisierten Periode eines Rechteck- oder Sinussignals. Der Programmierer kann wählen, wie groß die Bandbreite der Tonerkennung ist; dies kann man mit LEN einstellen. Die Sollfrequenz der Kreuzkorrelations-Routine ergibt sich aus der folgenden Formel:

$$f = \frac{10^6}{8 \cdot (19 + 11\ \mathrm{LEN} + 5\ \mathrm{FREQ})}\ \mathrm{Hz}$$

Um kurze Störimpulse auszumitteln und somit eine höhere Empfindlichkeit zu erzielen, wird die als Ergebnis im ERGL und ERGH stehende 16-Bit-Bewertungszahl über mehrere Perioden des Eingangssignals aufsummiert, hier über 32. Grob vereinfacht läßt sich dabei folgende Faustformel anwenden: Um die doppelte Empfindlichkeit zu erzielen, kann man entweder die Periodenzahl verdoppeln oder die Bandbreite auf die Hälfte verringern, indem man die Programmvariable LEN verdoppelt. Tut man beides gleichzeitig, ergibt sich sogar vierfache Empfindlichkeit. LEN gibt auch an, wie viele Zero-Page-Bytes für SPL vom Programm benutzt werden.

Leider hat eine Änderung von LEN Einfluß auf die sich ergebende Sollfrequenz. Mit der hier angegebenen Dimensionierung von LEN = 4 läßt sich deshalb nur eine

maximale Sollfrequenz von rund 1,8 kHz erreichen, also deutlich weniger als bei der Autokorrelation. Diese Beschränkung gilt jedoch nur für herkömmliche Mikroprozessoren. Inzwischen kamen auch spezielle Signalprozessoren auf den Markt, die speziell an die Erfordernisse der digitalen Signalverarbeitung angepaßt wurden und die eine Kreuzkorrelation auch bei wesentlich höheren Frequenzen gestatten.

6.10 Parallele Ausgabe

Um Zeichen auf einen Drucker auszugeben, verwendet man oft die sogenannte Centronics-Schnittstelle. Sie besteht insgesamt aus neun Leitungen: Sieben davon dienen dazu, die sieben Bits, mit denen ein ASCII-Zeichen codiert ist, parallel zum Drucker zu übertragen. Eine sogenannte Strobe-Leitung teilt dem Drucker mit, wann ein neues Zeichen an den sieben Datenleitungen anliegt, und umgekehrt kann der Drucker dem Computer über eine Handshake-Leitung melden, wann er für das nächste Zeichen aufnahmebereit ist. Letzteres ist erforderlich, weil der Mikrocomputer die Zeichen normalerweise wesentlich schneller ausgeben könnte, als sie gedruckt werden können.

Die Polarität der Strobe-Leitung und der Handshake-Leitung ist bei manchen Drukker-Fabrikaten unterschiedlich. Eine Anpassung des Beispielprogramms ist jedoch leicht entweder per Software mit einigen wenigen geänderten Bytes oder per Hardware durch Hinzuschalten eines Inverters möglich.

Der in *Abb. 6.10* vorgestellten Routine wird das auszugebende Zeichen im Akku übergeben. Sie wartet dann zunächst so lange, bis der Drucker über die Handshake-Leitung seine Empfangsbereitschaft meldet. Dann zieht sie das Strobe-Signal auf logisch Null, da die Strobe-Leitung gleichzeitig dem Drucker meldet, ob die Daten gültig sind (DAV = „Data Valid"). Jetzt werden die neuen Daten aus dem Akku auf die Ausgangsleitungen geschaltet, dann erst geht die DAV- bzw. Strobe-Leitung wieder auf logisch 1. Mit dieser positiven Flanke kann der Drucker das neue Zeichen dann übernehmen und ausgeben.

```
081F              *=$C50       ;BEISP-ADR
0C50
0C50        ;CENTRONICS-AUSGABE
0C50 CENTR 2C0208 BIT PB       ;WARTEN
0C53        70FB   BVS CENTR   ;BIS RDY=0
0C55        48     PHA
0C56        AD0008 LDA PA
0C59        297F   AND £$7F    ;DAV=0
0C5B        8D0008 STA PA      ;ALTES ZCH.
0C5E        68     PLA
0C5F        297F   AND £$7F    ;BIT7=0
0C61        8D0008 STA PA      ;NEUES ZCH.
0C64        0980   ORA £$80
0C66        8D0008 STA PA      ;DAV=1
0C69        60     RTS
```

Abb. 6.10 Parallele Ausgabe an PA 0...6 mit PB3 als Handshake- und PA7 als Strobeleitung

6.11 Serielle Ausgabe

Die Schnittstellennormen TTY und RS-232 (in Europa auch als V.24 bekannt) erlauben den Datenaustausch mit einem Minimum an Leitungen (*Abb. 6.11.1, 6.11.2*). Die einzelnen Datenbits werden dabei nicht parallel (gleichzeitig), sondern seriell (nacheinander) übertragen. Die Geschwindigkeit, mit der die Bits übertragen werden, nennt man Baudrate. Ein Baud entspricht einem Bit pro Sekunde (bei einwertiger digitaler Übertragung). *Abb. 6.11.3* zeigt das serielle Datenformat, *Abb. 6.11.4* eine geeignete Sende-Routine.

Wenn die Übertragung beginnt, wird die Datenleitung im Ruhezustand auf logisch 1 gehalten. Jedem Zeichen wird ein sogenanntes Startbit vorausgestellt; es ist immer Null, und an der dadurch entstehenden 1/0-Flanke kann der Datenempfänger den Zeichenbeginn erkennen. Auf das Startbit folgen acht Datenbits und, damit beim nächsten Zeichen wieder eine 1/0-Flanke gewährleistet werden kann, ein oder zwei Bits, die den Wert 1 haben. Sie garantieren, daß die Leitung zwischen den einzelnen Zeichen für einen kurzen Augenblick in ihrem Ruhezustand (1) verbleibt.

Da bei einer hohen Baudrate beispielsweise bei der Ansteuerung eines Druckers die Übertragung der einzelnen Zeichen schneller erfolgen kann, als der Drucker sie ausgeben kann, ist auch hier die Verwendung einer Handshake-Leitung sinnvoll. Sie sorgt dafür, daß die Zeichen-Ausgabe erst dann erfolgt, wenn der Drucker wieder empfangsbereit ist (wie schon bei der Centronics-Schnittstelle besprochen).

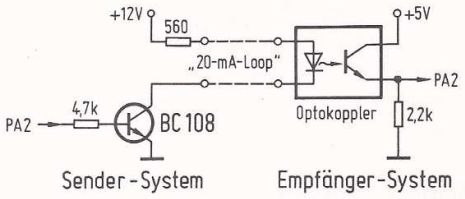


Abb. 6.11.1 Hardware einer TTY-Schnittstelle mit galvanischer Trennung durch Optokoppler

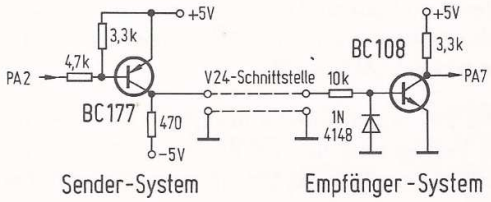


Abb. 6.11.2 Hardware einer V.24- bzw. RS-232-Schnittstelle (Handshake-Leitungen nicht gezeichnet)

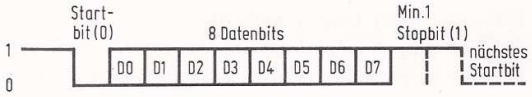


Abb. 6.11.3 Datenformat bei serieller Übertragung

```
0C6A          ;V.24-AUSGABE
0C6A          ;PA2=AUSGANG
0C6A          ;PB6=HANDSHAKE(RDY)
0C6A MSK1           =%00000100  ;PA2
0C6A MSKO           =%11111011
0C6A BD             =52         ;300 BD
0C6A OUT    2C0208  BIT PB      ;RDY?
0C6D        70FB    BVS OUT     ;NEIN
0C6F        208C0C  JSR AO      ;STARTBIT
0C72        A20A    LDX £10     ;8 DATEN- U.
0C74 NXT    38      SEC         ;2 STOPBITS
0C75        6A      ROR A       ;ZCH.SCHIEBEN
0C76        9005    BCC LO
0C78        20840C  JSR A1      ;PA2=HIGH
0C7B        3003    BMI HI
0C7D LO     208C0C  JSR AO      ;PA2=LOW
0C80 HI     CA      DEX
0C81        D0F1    BNE NXT
0C83        60      RTS
0C84 A1     48      PHA         ;1-BIT
0C85        AD0008  LDA PA
0C88        0904    ORA £MSK1
0C8A        D006    BNE A01
0C8C AO     48      PHA         ;0-BIT
0C8D        AD0008  LDA PA
0C90        29FB    AND £MSKO
0C92 A01    8D0008  STA PA
0C95        A934    LDA £BD     ;1 BIT
0C97        8D1608  STA T64     ;VERZOEG.
0C9A        68      PLA
0C9B DLY    2C1708  BIT TK
0C9E        10FB    BPL DLY
0CA0        60      RTS
```

Abb. 6.11.4 Serielle Ausgabe an PA2 mit 300 Bd

6.12 Serielle Eingabe

Hier folgt nun schließlich eine Routine, um seriell empfangene Zeichen per Software einlesen zu können. Die Handshake-Leitung ist bei diesem Programmstück allerdings noch nicht berücksichtigt; ihre Steuerung könnte vom restlichen Programm übernommen werden (*Abb. 6.12*).

Das Programm wartet zunächst so lange, bis ein Startbit erkannt wird. Dann erfolgt eine Verzögerung um 1,5 Bitlängen. Zu diesem Abfragezeitpunkt befinden wir uns genau in der Mitte des Datenbits D0. Sein Wert wird jetzt in den Akku geschoben. Im Abstand von jeweils genau einer Bitlänge folgen noch die sieben weiteren Datenbits. Nach dem letzten Datenbit erfolgt schließlich nochmals eine Verzögerung um eine Bitlänge, sonst könnte sich die Routine nämlich verhaspeln: Falls das letzte Datenbit Null ist und die Zeicheneingabe-Routine nach der Verarbeitung des zuletzt empfangenen Zeichens sofort wieder angesprungen wird, so würde dieses Datenbit fälschlich als

```
OCC3          PASS 1
OCC3          PASS 2
OCA1                .OPT LIST
OCA1          ;
OCA1          ;V.24-EINGABE
OCA1          ;PA7=EINGANG
OCA1 BDR            =52          ;300 BD
OCA1 BDH            =78
OCA1 IN       2C0008 BIT PA      ;STARTBIT?
OCA4          30FB   BMI IN      ;NEIN
OCA6          A208   LDX £8      ;8 BITS
OCA8          A04E   LDY £BDH    ;3/2 BITS
OCAA IN1      20BA0C JSR DEL     ;VERZOEG.
OCAD          A034   LDY £BDR    ;1 BIT
OCAF          18     CLC
OCB0          2C0008 BIT PA      ;EING=1?
OCB3          1001   BPL IN2     ;NEIN
OCB5          38     SEC
OCB6 IN2      6A     ROR A       ;Z.IM AKKU
OCB7          CA     DEX
OCB8          D0F0   BNE IN1     ;8 BITS,
OCBA DEL      8C1608 STY T64     ;DANN STOPBIT
OCBD DEL1     2C1708 BIT TK
OCC0          10FB   BPL DEL1
OCC2          60     RTS
OCC3           ERRORS= 0000
```

Abb. 6.12 Empfang serieller Zeichen an PA7 mit 300 Bd

Startbit des nächsten Zeichens erkannt werden. Die zusätzliche Verzögerung aber sorgt dafür, daß man statt dessen in der Mitte des Stopbits „landet".

Noch ein Hinweis zu den Stopbits: Nach dem Durchlaufen der Empfangsroutine benötigt das Programm gewöhnlich noch eine gewisse Zeit, um das gerade empfangene Zeichen weiterzuverarbeiten. Da dieser zusätzliche Zeitbedarf nicht immer unerheblich ist, ist es sinnvoll, sendeseitig zwei Stopbits zu erzeugen. Während des zweiten Stopbits hat man dann nämlich empfangsseitig genug Zeit, um z. B. das Zeichen im Speicher abzulegen oder zu decodieren.

6.13 Tastenfeldabfrage

Die folgende Routine dient dazu, ein aus sechzehn Tasten bestehendes Feld abzufragen. Um dafür nicht sechzehn Eingangsleitungen eines Peripherie-Bausteines zu belegen, sind die Tasten in einer 4×4-Matrix angeordnet (*Abb. 6.13.1*). Insgesamt sind also acht Leitungen erforderlich, die in unserem Fall an den Port PA führen. Dabei werden PA 0...3 als Ausgang, PA 4...7 als Eingang geschaltet. Die Funktionsweise des Programms in *Abb. 6.13.2* ist wie folgt:

Zunächst wird an PA0 logisch 0 ausgegeben, während PA 1, 2 und 3 an logisch 1 liegen. Jetzt prüft das Programm, ob eine der Tasten 0, 1, 2 oder 3 gedrückt ist: Dann

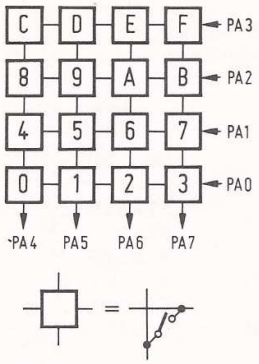


Abb. 6.13.1 Anordnung von 16 Tasten als 4×4-Matrix

```
0C56          ;TASTENABFRAGE 4X4
0C56          ;TASTENCODE IN KEY,
0C56          ;Z-FLAG=1 BEI GEDR.TASTE
0C56                *=0           ;ZERO PAGE
0000 KEY            *=$C80        ;TASTENCODE
0C80 INIT   A90F    LDA £$F       ;PA 0-3=
0C82        8D0108  STA PAD       ;AUSG.
0C85        ;
0C85 SCAN   A900    LDA £0        ;KEY=
0C87        8500    STA KEY       ;ZAEHLER
0C89        A203    LDX £3        ;F.MATRIX-
0C8B        A90E    LDA £$E       ;PUNKTE
0C8D        8D0008  STA PA
0C90 SCAN1  A004    LDY £4        ;PA 4-7
0C92        A910    LDA £$10      ;DURCH-
0C94 SCAN2  2C0008  BIT PA        ;PRUEFEN
0C97        F00F    BEQ FND
0C99        E600    INC KEY
0C9B        0A      ASL A
0C9C        88      DEY
0C9D        D0F5    BNE SCAN2     ;PA SCHIEBEN,
0C9F        0E0008  ASL PA        ;ABER PA0
0CA2        EE0008  INC PA        ;AUF HIGH
0CA5        CA      DEX
0CA6        10E8    BPL SCAN1
0CA8 FND    60      RTS
```

Abb. 6.13.2 Abfrage der Tastenmatrix per Software

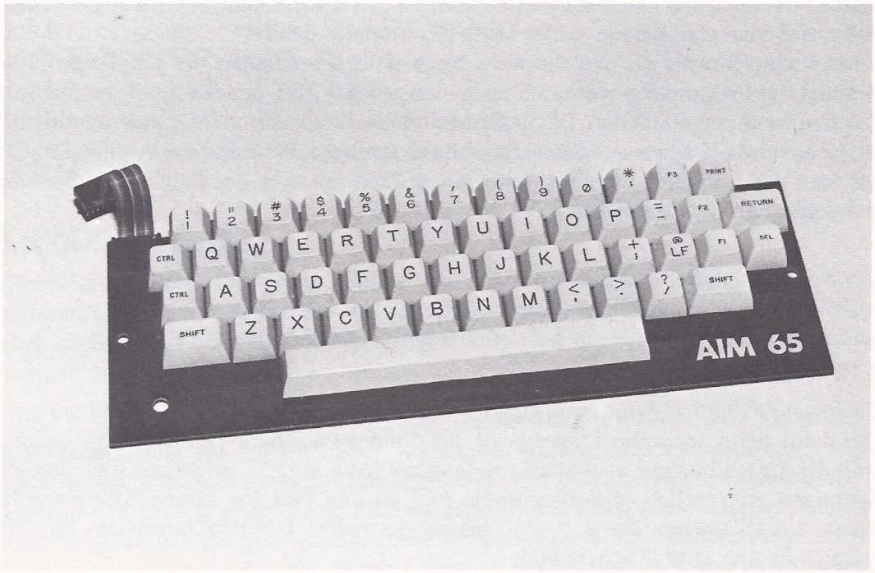


Abb. 6.13.3 Auch dieses Tastenfeld kann per Software abgefragt werden, wie das AIM-65-Monitorprogramm beweist

muß nämlich eine der Leitungen PA 4...7 Low-Pegel führen. Wenn das nicht der Fall ist, wird PA 0 wieder auf High-Pegel gelegt, PA 1 auf Low-Pegel. Falls eine der Tasten 4, 5, 6 oder 7 gedrückt ist, macht sich dies wiederum an PA 4...7 bemerkbar. So wird die logische 0 von PA 0...3 durchgeschoben, bis eine der Leitungen PA 4...7 auf Low-Pegel geht.

Das Programm liefert den Wert der gedrückten Taste (00...0F) nicht im Akku, sondern in der Speicherzelle KEY. Wenn eine Taste gedrückt war, ist das Z-Flag im Statusregister der CPU 1, andernfalls 0. Da die Routine selbst nicht wartet, bis eine Taste gedrückt ist, dies in den meisten Anwendungsfällen jedoch erforderlich wäre, kann man folgendes Programmstück verwenden:

```
LP1 JSR SCAN        ;Taste noch gedrückt?
    BEQ LP1         ;Ja
LP2 JSR SCAN        ;Neuer Tastendruck?
    BNE LP2         ;Nein
    JSR SCAN        ;Entprellung
    BNE LP2         ;Doch kein Tastendruck
    LDA KEY         ;Tastencode in Akku
```

Diese Routine wartet zunächst so lange, bis eine Taste gedrückt ist, und liefert ihren Wert dann im Akku ab. Durch mehrmaliges Anspringen des Unterprogramms SCAN wird auch ein bei den meisten Tastaturen unvermeidliches Kontaktprellen wirksam unterdrückt.

Schreibmaschinenähnliche Tastaturen wie jene in *Abb. 6.13.3*, die natürlich wesentlich mehr Tasten besitzen, lassen sich prinzipiell auf ähnliche Weise abfragen. So könnte man für eine Tastatur mit 64 Kontakten eine 8×8-Matrix mit insgesamt 16 Leitungen verwenden. Dies entspräche genau zwei 8-Bit-Ports, wovon einer als Eingang und einer als Ausgang geschaltet wäre. Eine Einsparung von Portleitungen ist aber noch dadurch möglich, daß man einen 3-zu-8-Decoder verwendet.

Die möglichen acht Ausgangskombinationen lassen sich nämlich mit nur drei Bits codieren. Eine andere Möglichkeit wäre die Verwendung eines integrierten Tastenencoders, wie er auf manchen Tastaturplatinen bereits enthalten ist. Er liefert gewöhnlich sieben Datenbits (ASCII), und eine weitere Leitung gibt an, ob gerade eine Taste gedrückt ist (Strobe).

6.14 Arithmetik-Routinen

Das hier schließlich noch beschriebene Programm-Paket in *Abb. 6.14* enthält vier Routinen, die 8-Bit-Zahlenwerte addieren, subtrahieren, multiplizieren und dividieren können [16]. Die Genauigkeit von 8 Bit reicht für zahlreiche Anwendungen aus und ermöglicht im Gegensatz zu noch genauerer Arithmetik höhere Rechengeschwindigkeiten. Solche Routinen werden in der Steuerungstechnik relativ häufig benötigt. Möchte man beispielsweise die von einem elektrischen Verbraucher benötigte Leistung anzeigen, so kann man über A/D-Wandler Spannung und Strom messen und erhält die Leistung, indem man beide Größen miteinander multipliziert.

```
0000          ;8-BIT-ARITHMETIK
0000          ;
0000          ;ARBEITSREGISTER:
0000 REG1           *=*+1
0001 REG2           *=*+1
0002 REG3
0002                *=$C90
0C90
0C90          ;
0C90          ;REG1=REG1+REG2
0C90          ;UEBERL:C=1
0C90 ADD      A500    LDA REG1
0C92          18      CLC
0C93          6501    ADC REG2
0C95          8500    STA REG1
0C97          60      RTS

0C98          ;REG1=REG1-REG2
0C98          ;UEBERL:C=0
0C98 SUB      A500    LDA REG1
0C9A          38      SEC
0C9B          E501    SBC REG2
0C9D          8500    STA REG1
0C9F          60      RTS

0CA0          ;REG1(MSB),REG2(LSB)=
0CA0          ;REG1*REG2
0CA0 MUL      A900    LDA £0
0CA2          A008    LDY £8
0CA4 MUL1     0A      ASL A
0CA5          2600    ROL REG1
0CA7          9007    BCC MUL2
0CA9          18      CLC
0CAA          6501    ADC REG2
0CAC          9002    BCC MUL2
0CAE          E600    INC REG1
0CB0 MUL2     88      DEY
0CB1          DOF1    BNE MUL1
0CB3          8501    STA REG2
0CB5          60      RTS

0CB6          ;REG1=REG1/REG2
0CB6          ;DIV.-REST IN REG2
0CB6 DIV      A008    LDY £8
0CB8          A900    LDA £0
0CBA DIV1     2600    ROL REG1
0CBC          2A      ROL A
0CBD          C501    CMP REG2
0CBF          9002    BCC DIV2
0CC1          E501    SBC REG2
0CC3 DIV2     88      DEY
0CC4          DOF4    BNE DIV1
0CC6          2600    ROL REG1
0CC8          8501    STA REG2
0CCA          60      RTS
```

```
0CCB          ;BIN/DEZ-KONVERSION
0CCB          ;ARG.IN REG1, ERGEBNIS
0CCB          ;IN REG2(LSB),REG3(MSB
0CCB CONV     F8      SED
0CCC          A008    LDY £8
0CCE          A900    LDA £0
0CD0          8502    STA REG3
0CD2          8501    STA REG2
0CD4 CONV1    0600    ASL REG1
0CD6          A501    LDA REG2
0CD8          6501    ADC REG2
0CDA          8501    STA REG2
0CDC          2602    ROL REG3
0CDE          88      DEY
0CDF          DOF3    BNE CONV1
0CE1          D8      CLD
0CE2          60      RTS
0CE3                  .END
0CE3           ERRORS= 0000
```

Abb. 6.14 Routinen für Ganzzahl-Arithmetik

Man kann sich leicht ausrechnen, daß zwei 8-Bit-Zahlen, miteinander multipliziert, ein Ergebnis mit bis zu 16 Bit Länge liefern können: 255 × 255 = 65 025. Die Multiplikations-Routine ist deshalb so ausgelegt, daß sie ein 16-Bit-Ergebnis liefern kann. Ebenso ist es manchmal nützlich, den bei einer Division übrigbleibenden Rest weiter zu verwerten. Die Divisions-Routine speichert deshalb auch den Divisionsrest ab.

Um das jeweilige Ergebnisbyte auch dezimal ausgeben zu können, wurde noch eine Binär/Dezimal-Umwandlungsroutine aufgelistet. Sie addiert Schritt für Schritt die Bitwertigkeiten, wobei als Ergebnis eine BCD-Zahl zur Verfügung steht, die zwei Bytes lang ist. Sie kann 0000...0255 sein, entsprechend den Hexadezimalzahlen 00...FF.

Nun zu den einzelnen Routinen. Die angegebenen Adressen sind nur beispielhaft aufzufassen, da die Unterprogramme selbst keine absoluten Sprungadressen enthalten und deshalb in ihrer Adressenlage frei verschieblich sind.

ADD
Binäre Addition der beiden 8-Bit-Zahlen in REG1 und REG2. Das Ergebnis wird in REG1 abgespeichert. Ist das Ergebnis größer als hex FF (Überlauf), so wird das Carry-Flag in der CPU gesetzt. Die Routine dauert (ohne RTS-Befehl) bei 1 MHz Taktfrequenz 11 µs.

SUB
Binäre Subtraktion des Inhalts von REG2 vom Wert in REG1, Ergebnis in REG1. Wird der Rechenbereich von 8 Bit überschritten, d. h. ist das Ergebnis negativ, so wird das Carry-Flag der CPU zurückgesetzt. Auch dieses Programmstück dauert 11 µs.

MUL
Der Inhalt von REG1 wird mit dem von REG2 multipliziert. Das Ergebnis wird in REG2 (niederwertiges Byte) und REG1 (höherwertiges Byte) abgespeichert. Ein Überlauf kann ohnehin nicht auftreten, da die Routine ein 16-Bit-Ergebnis liefert. Die Laufzeit beträgt je nach Operanden etwa 120...200 µs.

DIV
REG1 wird mit REG2 dividiert. Das Ergebnis steht in REG1, und in REG2 liefert die Routine den Divisionsrest. Ein Beispiel: 0A/03 = 03, Rest 1 (dezimal: 10/3 = 3, Rest 1). Für diese Routine muß man rund 150...170 µs rechnen.

CONV
Der Binärwert in REG1 wird in eine Dezimalzahl (BCD-Darstellung) umgewandelt. Das niederwertige Ergebnisbyte steht dann in REG2, das höherwertige in REG3. Beispiel: REG1 enthalte den Hex-Wert FF. In REG2 steht nach dem Durchlaufen der CONV-Routine 55, in REG3 die Zahl 02, zusammen also 0255. Die Laufzeit beträgt (ohne RTS-Befehl) 249 µs.

Die Wahl von Argument- und Ergebnis-Registern (REG 1...3) war hier rein willkürlich. Die Adressenreihenfolge dieser Speicherzellen und ihre Zuordnung in den Arithmetik-Routinen kann selbstverständlich auch anderen Erfordernissen angepaßt werden.

Andere Funktionen, von der Quadratwurzel bis zum Sinus, können stets aus den vier Grundrechenarten abgeleitet werden. Sinnvoll ist dabei meist die Verwendung von Koeffizienten-Tabellen, um die Zahl der nötigen Rechenoperationen und damit den Zeitbedarf möglichst niedrig zu halten.

7 Vollständige Anwendungsprogramme

Die im letzten Kapitel besprochenen Routinen stellen keine Komplettprogramme für Einplatinencomputer dar, sondern dienen nur als Unterprogramme für bestimmte Teilaufgaben des Systems. Die im folgenden besprochenen Programme sind dagegen jeweils die komplette Betriebssoftware zur Lösung einer bestimmten Aufgabenstellung. Sie unterscheiden sich von den bisher aufgelisteten Teilprogrammen in zwei Punkten:

1. Im Assembler-Listing wird der Reset-Vektor (0FFC im EMUF) auf die Startadresse des Applikationsprogramms gesetzt. Damit wird der richtige Programmbeginn beim Einschalten gewährleistet.
2. Am Programmanfang sind zusätzliche Befehle nötig, die
   a) den Stackpointer der CPU auf den gewünschten Anfangswert setzen (hex FF beim EMUF für die Adresse 01FF);
   b) die CPU initialisieren, z. B. das Dezimal-Flag oder das Interrupt-Disable-Bit im Statusregister der CPU setzen oder rücksetzen;
   c) durch das Schreiben bestimmter Werte in die Port-Datenrichtungsregister festlegen, welche Leitungen der Peripherie-Bausteine als Ein-/Ausgang dienen sollen.

Am Anfang des Assemblerlistings werden ferner die nötigen Adressenvereinbarungen für den Programmspeicher (EPROM) und den Arbeitsspeicher (RAM) getroffen. Einen definierten Endpunkt weisen die Anwendungsprogramme normalerweise nicht auf, weil die Betriebssoftware ein in sich abgeschlossenes System darstellt, das nur durch Abschalten der Versorgungsspannung außer Betrieb gesetzt wird.

7.1 Tonrufempfänger

Gerade die folgende Applikation zeigt, daß es mit einem Mikrocomputer möglich ist, einen sonst nötigen erheblichen Schaltungsaufwand durch nur wenige ICs zu ersetzen.

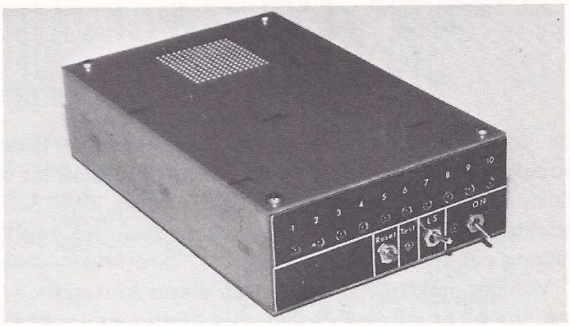


Abb. 7.1.1 Labormuster eines Eurosignal-Empfängers mit dem EMUF

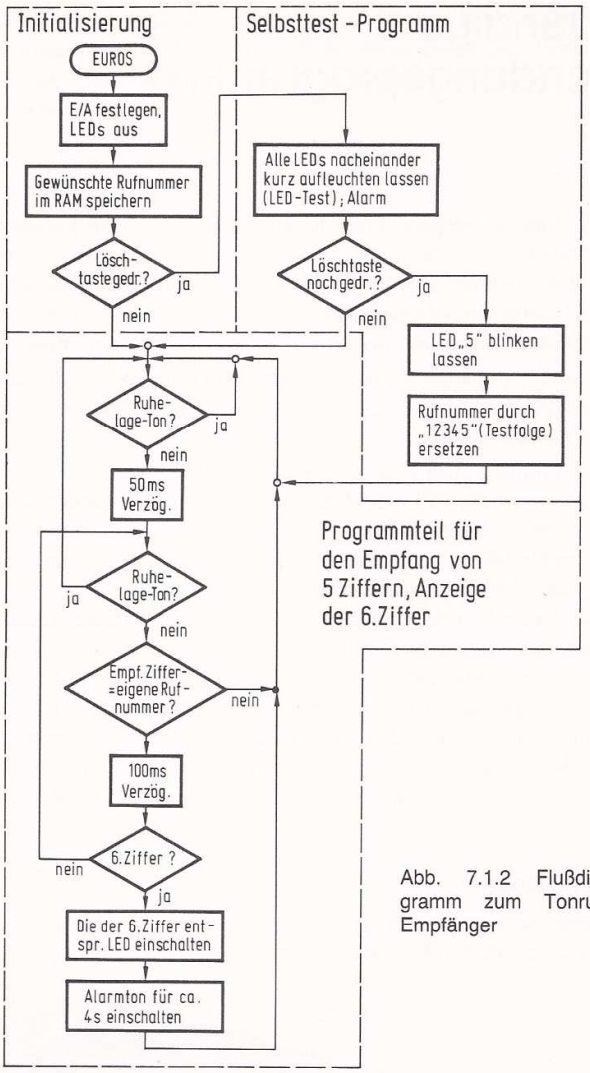
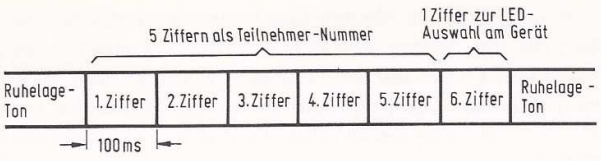


Abb. 7.1.2 Flußdiagramm zum Tonruf-Empfänger

Im Vordergrund steht dabei die softwaremäßige Erkennung einer über Funk ausgestrahlten Tonfolge. Die Programm-Parameter sind für das europaweit zu empfangende Eurosignal-Netz ausgelegt, das auf UKW um 87,36 MHz und mit 6stelligen Nummern arbeitet. *Abb. 7.1.1* zeigt einen Laboraufbau der Applikation, *Abb. 7.1.2.* das Flußdiagramm des EMUF-Programms.

Von der praktischen Realisation dieser Anwendung sei dem Leser allerdings abgeraten, um nicht mit den postalischen Bestimmungen in Konflikt zu geraten. Der Betrieb

Abb. 7.1.3 Format der Ziffernübertragung beim Eurosignal-Funkrufdienst



von Eurosignal-Empfängern ist nur mit einer Zulassung der Fernmeldebehörde gestattet und gebührenpflichtig.

Das Eurosignal-Netz dient dazu, Personen, die ziemlich viel unterwegs und daher nicht dauernd telefonisch erreichbar sind, aufzufordern, sich zu Hause, im Büro oder anderswo zu melden. Der Fernsprechteilnehmer, der einen Benutzer des europäischen Funkrufdienstes, also den Inhaber des Empfängers erreichen will, wählt zuerst die Eurosignal-Vorwahl und dann eine 6stellige Nummer. Diese Nummer wird dann europaweit, durch bestimmte Töne codiert, auf UKW übertragen (*Abb. 7.1.3*).

Die ersten fünf Ziffern der Nummer sprechen dabei das jeweilige Gerät des Teilnehmers selektiv an, während die sechste Ziffer dazu verwendet werden kann, um dem Teilnehmer klar zu machen, wo er nun anrufen soll (z. B. „1" für zu Hause, „2" für Büro usw.). Zu diesem Zweck leuchten unterschiedliche Leuchtdioden am Empfänger auf [17].

Insgesamt arbeitet das Eurosignal-Netz mit zwölf unterschiedlichen Tönen. Zehn davon codieren die Ziffern 0...9; einer dient als Ruhelage-Ton, solange keine Nummer übertragen wird. Ein weiterer Ton kennzeichnet die Wiederholung der vorhergehenden Ziffer: Wird beispielsweise die Nummer 122345 übertragen, so wird tatsächlich folgende Codierung ausgestrahlt: 1, 2, Wiederholung, 3, 4, 5. Jeder Ton ist dabei 100 ms lang.

Abb. 7.1.4 zeigt die Schaltung des Labormusters. Bei der hier vorgestellten Problemlösung sind zehn Leuchtdioden für die sechste übertragene Ziffer vorhanden, und mit

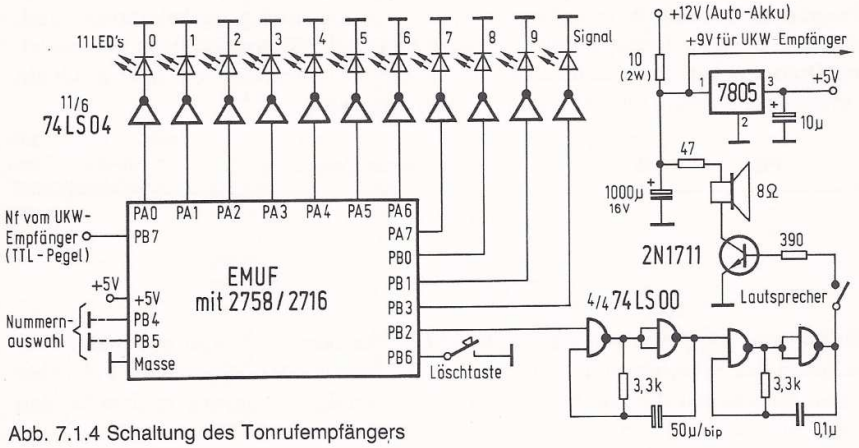


Abb. 7.1.4 Schaltung des Tonrufempfängers

zwei Drahtbrücken können vier unterschiedliche Nummernfolgen programmiert werden, die am Anfang des EPROM-Adressenbereiches gespeichert sind. Stimmen die ersten fünf Ziffern mit der Programmierung überein, so leuchtet die der sechsten Ziffer entsprechende LED auf, und etwa vier Sekunden lang ertönt ein unterbrochener Alarmton, um den Geräteinhaber auf einen empfangenen Ruf aufmerksam zu machen. Während der Lautsprecher automatisch wieder verstummt, leuchtet die LED so lange weiter, bis man die Löschtaste drückt. Treffen bis zum Drücken der Löschtaste mehrere Rufe ein, die sich in der letzten Ziffer unterscheiden, so leuchten entsprechend auch mehrere LEDs auf.

Das Programm in *Abb. 7.1.5* ermöglicht einen einfachen Selbsttest, indem man während des Einschaltens die Löschtaste gedrückt hält: Dann werden nämlich probeweise nacheinander alle LEDs kurz eingeschaltet, und auch der Alarmton ist kurz zu hören. Hält man jetzt die Löschtaste immer noch gedrückt, so ist das Gerät nicht auf die normale Rufnummer programmiert, sondern auf die Nummernfolge 12345, was durch zehnmaliges kurzes Aufblinken der Leuchtdiode „5" angezeigt wird. Damit ist es möglich, zu überprüfen, ob die Empfangsfeldstärke für eine einwandfreie Codierung ausreicht: Im Abstand von einigen Sekunden sendet das Eurosignal-Netz nämlich als Testnummernfolge die Ziffern 123455. Wenn diese nun im Selbsttest-Modus empfangen wird, gibt der eingebaute Lautsprecher den Alarmton von sich, und die LED „5" leuchtet als Bestätigung auf. Möchte man das Gerät nun auf die normale Rufnummer zurückprogrammieren, so schaltet man es kurz aus und wieder ein.

Sehen wir uns nun die Abschnitte des Assembler-Programms etwas näher an. Gleich am Anfang des EPROM-Speicherbereichs 0C00 stehen die vier vorprogrammierbaren Rufnummern 40673, 54333, 51481 und 47393. Jede Ziffer ist dabei in einem Byte codiert. Da jede Rufnummer nur aus fünf Ziffern besteht, für das Programm eine Adressierung aber in Achterschritten einfacher ist, wurden nach jeder Rufnummer drei Bytes mit irrelevanter Information aufgefüllt. Ab der Adresse 0C1D mit dem symbolischen Namen CHK findet sich schließlich noch die Testnummernfolge 12345.

Der Resetvektor bei 0FFC zeigt hier auf die Adresse 0C22. Das bedeutet, daß der Mikrocomputer beim Einschalten automatisch mit der Abarbeitung des ab 0C22 stehenden Programms beginnt. Er initialisiert zunächst die Ein- und Ausgabeleitungen und, soweit nötig, die CPU-Register. Die Leitungen PB4 und PB5 werden dazu verwendet, um mit Drahtbrücken die gewünschte Rufnummer zu programmieren. Dies geschieht nach folgendem Schema (0 = Masse, 1 = offen oder +5 V):

PB5	PB4	Nr.	Vorprogrammierte Ziffernfolge
0	0	0	40673
0	1	1	54333
1	0	2	51481
1	1	3	47393

Die Rufnummer wird nun aus dem EPROM-Speicherbereich in fünf aufeinanderfolgende Zellen des Arbeitsspeichers (hier ab der Adresse 0000) übernommen. Falls aber nach dem Einschalten die Löschtaste gedrückt ist, erfolgt ein Sprung zu dem für den Selbsttest zuständigen Programmteil.

```
FF10        PASS 1
1000        PASS 2
0000
0000        ;EUROS 1.6-EMUF
0000 PA             =$800       ;PORTS
0000 PAD            =$801
0000 PB             =$802
0000 PBD            =$803
0000 MSEC           =$817       ;TIMER
0000                *=0
0000 CALL           *=*+5       ;RAM
0005 TIL            *=*+1
0006 TIH            *=*+1
0007 FRST           *=*+1
0008 RPT            *=*+1
0009 HI             *=$C00
0C00
0C00        ;
0C00        ;EPROM
0C00 CALLO  04      .BYT 4,0,6,7,3
0C01        00
0C02        06
0C03        07
0C04        03
0C05        4E5230  .BYT 'NRO'
0C08        05      .BYT 5,4,3,3,3
0C09        04
0C0A        03
0C0B        03
0C0C        03
0C0D        4E5231  .BYT 'NR1'
0C10        05      .BYT 5,1,4,8,1
0C11        01
0C12        04
0C13        08
0C14        01
0C15        4E5232  .BYT 'NR2'
0C18        04      .BYT 4,7,3,9,3
0C19        07
0C1A        03
0C1B        09
0C1C        03
0C1D CHK    01      .BYT 1,2,3,4,5
0C1E        02
0C1F        03
0C20        04
0C21        05
0C22        ;
0C22        ;RESET SEQUENCE
0C22 RES    A2FF    LDX £$FF    ;INIT
0C24        8E0108  STX PAD     ;PORTS
0C27        9A      TXS         ;+ CPU
0C28        78      SEI
0C29        D8      CLD
0C2A        A20F    LDX £$F
```

Abb. 7.1.5 Listing des Programms zur Tonfolgeruf-Decodierung

```
0C2C          8E0308 STX PBD      ;CLEAR
0C2F          20EE0C JSR CLR      ;LEDS
0C32          AD0208 LDA PB
0C35          4A     LSR A
0C36          2918   AND £$18
0C38          6903   ADC £3       ;PB4-5
0C3A          A8     TAY          ;TO SEL.
0C3B          A204   LDX £4       ;CALL
0C3D INI      B9000C LDA CALLO,Y
0C40          9500   STA CALL,X   ;STORE
0C42          88     DEY          ;CALL TO
0C43          CA     DEX          ;Z-PAGE
0C44          10F7   BPL INI
0C46          2C0208 BIT PB       ;CLR KEY
0C49          7003   BVS ML       ;CLOSED?
0C4B          20090D JSR TEST     ;YES
0C4E
0C4E          ;
0C4E          ;MAIN LOOP
0C4E ML       A200   LDX £0
0C50          AD0208 LDA PB       ;SWITCH
0C53          0908   ORA £8       ;MARK
0C55          8D0208 STA PB       ;LED OFF
0C58          20FA0C JSR NR       ;MARK?
0C5B          D0F1   BNE ML
0C5D ST       AD0208 LDA PB       ;MARK
0C60          29F7   AND £$F7     ;LED
0C62          8D0208 STA PB       ;ON AGAIN
0C65          20FA0C JSR NR
0C68          90E4   BCC ML       ;ERROR
0C6A          F0F1   BEQ ST
0C6C          A92D   LDA £45      ;HALF TONE
0C6E          8D1708 STA MSEC     ;DELAY
0C71 WAI      20400D JSR DLY1
0C74 LP       A95F   LDA £95      ;ONE TONE
0C76          8D1708 STA MSEC     ;DELAY
0C79          20FA0C JSR NR
0C7C          90D0   BCC ML       ;ERROR
0C7E          F0CE   BEQ ML       ;ALL OVER
0C80          88     DEY
0C81          F002   BEQ SM       ;REPEAT
0C83          8408   STY RPT
0C85 SM       A408   LDY RPT
0C87          E005   CPX £5       ;6TH NR?
0C89          D021   BNE LP2      ;NO
0C8B          A9FF   LDA £$FF
0C8D          8509   STA HI
0C8F          18     CLC          ;CONVERT
0C90 LP3      2A     ROL A        ;DIGIT TO
0C91          2609   ROL HI       ;BINARY
0C93          88     DEY
0C94          D0FA   BNE LP3
0C96          2D0008 AND PA
0C99          8D0008 STA PA       ;DISPLAY
0C9C          AD0208 LDA PB
```

zu Abb. 7.1.5

```
0C9F        2509    AND HI
0CA1        0904    ORA £4       ;ALARM ON
0CA3        8D0208  STA PB
0CA6        A90A    LDA £10      ;ALARM
0CA8        8506    STA TIH      ;CA 4SEC
0CAA        D0A2    BNE ML
0CAC LP2    88      DEY          ;ADJ NR
0CAD        98      TYA          ;0=0
0CAE        D500    CMP CALL,X
0CB0        D09C    BNE ML
0CB2        E8      INX
0CB3        4C710C  JMP WAI      ;NEXT NR
0CB6        ;
0CB6        ;VERIFY PERIOD TIME
0CB6 VER    20CB0C  JSR T        ;MEASURE
0CB9        8507    STA FRST     ;2PER.
0CBB        20CB0C  JSR T
0CBE        38      SEC
0CBF        E507    SBC FRST     ;EQUAL?
0CC1        18      CLC          ;ALLOW
0CC2        6902    ADC £2       ;-2/+1
0CC4        29FC    AND £%11111100
0CC6        D0EE    BNE VER      ;NO
0CC8        A507    LDA FRST
0CCA        60      RTS          ;YES
0CCB        ;
0CCB        ;TEST PERIOD TIME
0CCB T      C605    DEC TIL      ;ALARM
0CCD        D00C    BNE TO       ;TIME
0CCF        C606    DEC TIH      ;OVER?
0CD1        D008    BNE TO
0CD3        AD0208  LDA PB
0CD6        29FB    AND £$FB     ;ALARM OFF
0CD8        8D0208  STA PB
0CDB TO     20E00C  JSR T1       ;SYNC
0CDE        A000    LDY £0
0CE0 T1     C8      INY          ;MEASURE
0CE1        2C0208  BIT PB       ;1ST HALF
0CE4        30FA    BMI T1       ;CYCLE
0CE6 T2     C8      INY          ;AND NOW
0CE7        2C0208  BIT PB       ;2ND
0CEA        10FA    BPL T2
0CEC        700A    BVS T3       ;CLR LEDS?
0CEE CLR    A9FF    LDA £$FF     ;YES
0CF0        8D0008  STA PA
0CF3        A90B    LDA £$B      ;ALARM OFF
0CF5        8D0208  STA PB
0CF8 T3     98      TYA
0CF9        60      RTS          ;T IN Y+A
0CFA        ;
0CFA        ;DETERMINE NUMBER
0CFA NR     20B60C  JSR VER
0CFD        A00B    LDY £11
0CFF T4     D9460D  CMP FQ,Y
0D02        B003    BCS RTN
```

zu Abb. 7.1.5

```
0D04          88      DEY
0D05          10F8    BPL T4        ;C=0:ERROR
0D07 RTN      98      TYA           ;A=Y=NR
0D08          60      RTS           ;Z=1:MARK
0D09          ;
0D09          ;SELF-TEST
0D09 TEST     A20B    LDX £11       ;TEST
0D0B          18      CLC           ;ALL NR-
0D0C          9004    BCC TST0      ;LEDS
0D0E TST1     203B0D  JSR DLY
0D11          38      SEC           ;AND
0D12 TST0     2E0008  ROL PA        ;ALARM
0D15          2E0208  ROL PB        ;ON/OFF
0D18          CA      DEX
0D19          D0F3    BNE TST1
0D1B          2C0208  BIT PB        ;CLR KEY
0D1E          70E7    BVS RTN       ;DOWN?
0D20          A214    LDX £20       ;YES
0D22 TST2     AD0008  LDA PA
0D25          4920    EOR £$20      ;FLASH
0D27          8D0008  STA PA        ;NR.5
0D2A          203B0D  JSR DLY       ;LED
0D2D          CA      DEX           ;10TIMES
0D2E          D0F2    BNE TST2
0D30          A204    LDX £4
0D32 TST3     BD1D0C  LDA CHK,X
0D35          9500    STA CALL,X    ;USE TEST
0D37          CA      DEX           ;SEQUENCE
0D38          10F8    BPL TST3      ;AS CALL
0D3A          60      RTS
0D3B DLY      A062    LDY £98       ;DELAY
0D3D          8C1708  STY MSEC      ;0.1SEC
0D40 DLY1     2C1708  BIT MSEC
0D43          10FB    BPL DLY1      ;WAIT
0D45          60      RTS
0D46          ;
0D46          ;FREQ VALUES
0D46 FQ       5A      .BYT 90,99,108
0D47          63
0D48          6C
0D49          74      .BYT 116,128,139,151
0D4A          80
0D4B          8B
0D4C          97
0D4D          A4      .BYT 164,178,193
0D4E          B2
0D4F          C1
0D50          D1      .BYT 209,227
0D51          E3
0D52          FF      .BYT $FF,$FF,$FF
0D53          FF
0D54          FF
0D55          4443    .BYT 'DC1YB '
0D5B          4555    .BYT 'EUROS 1.6'
0D64          FF      .BYT $FF,$FF,$FF
```

zu Abb. 7.1.5

```
0D65        FF
0D66        FF
0D67        ;
0D67              *=$FFC
0FFC
0FFC        ;RESET VECTOR
0FFC        220C    .WOR RES
0FFE        220C    .WOR RES
1000              .END
1000         ERRORS= 0000
```

zu Abb. 7.1.5

Die Decodierung der als Tonfolge empfangenen Rufnummer erfolgt nach dem Prinzip der Periodendauermessung. Dies geschieht, um eine hohe Auswertesicherheit zu erzielen, in der Routine VER sicherheitshalber zweimal nacheinander, wobei sich die beiden Meßwerte nur geringfügig voneinander unterscheiden dürfen; andernfalls wird die Messung wiederholt.

Die Zuordnung der Codierung zur Periodendauer der empfangenen Töne erfolgt über die Datentabelle FQ (0D46...0D51). Der Mikrocomputer kehrt sofort zur Hauptprogrammschleife ML zurück, wenn die empfangene Codierung nicht mit der vorprogrammierten Rufnummer übereinstimmt. Außerdem wird, solange die Ruhelage-Frequenz empfangen wird, das Vorhandensein einer ausreichenden Empfangsfeldstärke mit einer Leuchtdiode angezeigt. Sie erlischt natürlich jedesmal kurz, wenn der Sender den Ruhelageton unterbricht und eine Rufnummer ausstrahlt.

Rein prinzipiell wäre es natürlich auch möglich gewesen, den Alarmton für den Lautsprecher ebenfalls vom Mikrocomputer erzeugen zu lassen. Dies hätte aber Zeitprobleme verursacht: Solange der Computer einen Ton erzeugt, hat er kaum Zeit, gleichzeitig eine empfangene Tonfolge zu decodieren. Die Tonerzeugung wurde deshalb mit einer kleinen externen Schaltung vorgenommen, die ein unterbrochenes Alarmsignal an den Lautsprecher ausgibt. Sie wird im Bedarfsfall etwa vier Sekunden lang über eine Ausgangsleitung des Computers aktiviert.

Die Schaltung enthält nicht den für den UKW-Empfang zuständigen Teil. Dieser muß die empfangenen Töne mit TTL-Pegel an einen Eingang des Computers liefern, um die Periodendauer-Messung zu ermöglichen. Die Stromversorgungsschaltung ist für den Betrieb an einem 12-V-Kfz-Bordnetz ausgelegt. Sie verbraucht maximal etwa 0,4 A. Für den Betrieb im Auto spielt dies keine Rolle, für den bei tragbaren Geräten aber üblichen Batteriebetrieb wäre dies zuviel. Durch den Einsatz eines CMOS-Mikrocomputers ließe sich dieses Problem aber lösen.

7.2 Die melodische Eieruhr

Die nun folgende EMUF-Applikation ist eher dem Consumer-Bereich zuzuordnen. Das Programm spielt nach einer durch Tastendruck vorprogrammierbaren Zeit eine von mehreren gespeicherten Melodien. Der Aufwand für Software und Hardware ist ver-

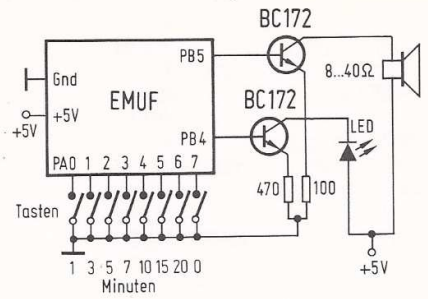


Abb. 7.2.1 Externe Beschaltung des EMUF in der „melodischen Eieruhr"

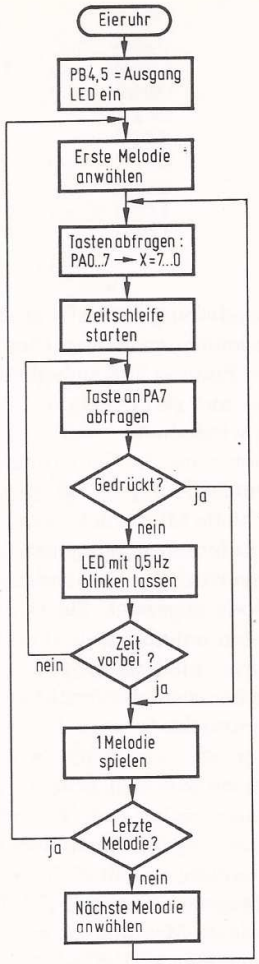


Abb. 7.2.2 Flußdiagramm der „Eieruhr"

gleichsweise gering und gut überschaubar. *Abb. 7.2.1* zeigt die nötige Beschaltung des EMUF, *Abb. 7.2.2* das Flußdiagramm der Software und *Abb. 7.2.3* das Assemblerlisting.

Zwei der EMUF-I/O-Leitungen sind als Ausgänge geschaltet: Eine steuert eine Leuchtdiode an, die blinkt, während die vorprogrammierte Zeit abläuft, und eine weitere dient als Ansteuerung für den Lautsprecher. An beiden Ausgängen ist jeweils ein Treibertransistor erforderlich, da die Belastbarkeit der Peripherie-Ports weder für einen Lautsprecher noch für eine Leuchtdiode ausreichen würde.

Die acht Leitungen des Ports PA sind als Eingänge geschaltet, an denen acht Tasten angeschlossen sind. Sieben davon dienen zum Einstellen der gewünschten Zeit, nämlich für 1, 3, 5, 7, 10, 15 oder 20 Minuten, während sich mit einer achten Taste der

```
FF00        PASS 1
0FFE        PASS 2
0000
0000        ;E I E R U H R
0000        ;PA 0-7 = TASTEN
0000        ;PB4 = LED
0000        ;PB5 = MUSIKAUSG.
0000              *=0
0000 WORK         *=*+6
0006 LIMIT        *=*+3
0009 VAL2         *=*+1
000A VAL1         *=*+1
000B TIMER        *=*+1
000C XSAV         *=$C00
0C00 MUS          =$D00        ;NOTENTAB.
0C00 PA           =$800
0C00 PB           =$802
0C00 PBD          =$803
0C00 TK           =$817
0C00 RESV         =$FFC
0C00 RES    A2FF  LDX £$FF     ;RESET
0C02        9A    TXS
0C03        A230  LDX £%00110000
0C05        8E0308 STX PBD     ;PB=AUSG
0C08        A9DF  LDA £%11011111
0C0A        8D0208 STA PB
0C0D        78    SEI
0C0E        D8    CLD
0C0F FRST   A205  LDX £5       ;ERSTE
0C11 LP1    BDCF0C LDA INIT,X  ;MEL.
0C14        9500  STA WORK,X
0C16        CA    DEX
0C17        10F8  BPL LP1
0C19 WAIT   A207  LDX £7       ;TASTE
0C1B        AD0008 LDA PA      ;GEDR.?
0C1E SRCH   6A    ROR A
0C1F        9005  BCC FND      ;JA
0C21        CA    DEX
0C22        10FA  BPL SRCH     ;NEIN
0C24        30F3  BMI WAIT
0C26 FND    BDD50C LDA TAB,X   ;ZEIT
0C29        AA    TAX
0C2A LOP    A0E6  LDY £230     ;1MIN
0C2C LOP0   A9FF  LDA £255
0C2E        8D1708 STA TK
0C31 LOP1   2C0008 BIT PA      ;PA7=L?
0C34        1013  BPL GO       ;JA
0C36        2C1708 BIT TK
0C39        10F6  BPL LOP1
0C3B        AD0208 LDA PB
0C3E        4910  EOR £$10
0C40        8D0208 STA PB
0C43        88    DEY
0C44        D0E6  BNE LOP0
0C46        CA    DEX
```

Abb. 7.2.3 Assemblerlisting der EMUF-Eieruhr

```
0C47          D0E1    BNE LOP
0C49 GO       A000    LDY £0        ;MUSIK-
0C4B          B104    LDA (WORK+4),Y
0C4D          C9FF    CMP £$FF      ;PROGR.
0C4F          F0BE    BEQ FRST
0C51          E604    INC WORK+4    ;NACH
0C53          D002    BNE *+4       ;JIM BUTTER-
0C55          E605    INC WORK+5    ;FIELD'S
0C57          C9FA    CMP £$FA      ;MUSICBOX
0C59          F0BE    BEQ WAIT
0C5B NEXT     900F    BCC NOTE
0C5D          E9FB    SBC £$FB
0C5F          AA      TAX
0C60          B104    LDA (WORK+4),Y
0C62          E604    INC WORK+4
0C64          D002    BNE *+4
0C66          E605    INC WORK+5
0C68          9500    STA WORK,X
0C6A          B0DD    BCS GO
0C6C NOTE     A600    LDX WORK
0C6E          8607    STX LIMIT+1
0C70          A601    LDX WORK+1
0C72          A8      TAY
0C73          3002    BMI OVER
0C75          A201    LDX £1
0C77 OVER     8606    STX LIMIT
0C79          297F    AND £$7F
0C7B          8509    STA VAL2
0C7D          F002    BEQ HUSH
0C7F          850A    STA VAL1
0C81 HUSH     A509    LDA VAL2
0C83          2503    AND WORK+3
0C85          F004    BEQ ON
0C87          E60A    INC VAL1
0C89          C609    DEC VAL2
0C8B ON       A609    LDX VAL2
0C8D          AD0208  LDA PB
0C90          0920    ORA £%00100000
0C92          20A60C  JSR SOUND
0C95          30B2    BMI GO
0C97          A60A    LDX VAL1
0C99          78      SEI
0C9A          AD0208  LDA PB
0C9D          29DF    AND £%11011111
0C9F          20A60C  JSR SOUND
0CA2          30A5    BMI GO
0CA4          10DB    BPL HUSH
0CA6 SOUND    A402    LDY WORK+2
0CA8          840B    STY TIMER
0CAA          860C    STX XSAV
0CAC SLOOP    E000    CPX £0          zu Abb. 7.2.3
0CAE          D008    BNE CONT
0CB0          A60C    LDX XSAV
0CB2          C60B    DEC TIMER
0CB4          D0F6    BNE SLOOP
```

```
OCB6          F016    BEQ SEX
OCB8 CONT     8D0208  STA PB
OCBB          CA      DEX
OCBC          C608    DEC LIMIT+2
OCBE          D0EC    BNE SLOOP
OCC0          C607    DEC LIMIT+1
OCC2          D0E8    BNE SLOOP
OCC4          A400    LDY WORK
OCC6          8407    STY LIMIT+1
OCC8          C606    DEC LIMIT
OCCA          D0E0    BNE SLOOP
OCCC          A9FF    LDA £$FF
OCCE SEX      60      RTS
OCCF INIT     30      .BYT 48,2,1,$FF
OCD0          02
OCD1          01
OCD2          FF
OCD3          000D    .WOR MUS
OCD5 TAB      1E      .BYT 30,20,15,10
OCD6          14
OCD7          0F
OCD8          0A
OCD9          07      .BYT 7,5,3,1
OCDA          05
OCDB          03
OCDC          01
OCDD                  *=RESV
OFFC          000C    .WOR RES
OFFE                  .END
OFFE           ERRORS= 0000
```

Notentabelle f. Musikprogramm:

```
(_)=0D00 FB 18 FE FF 44 51 E6 E6 66 5A 51 4C C4 C4 C4 D1
(¯) 0D10 BD BD BD 00 44 BD 00 44 3D 36 33 2D A8 80 80 33
( ) 0D20 44 B3 80 80 44 51 C4 80 80 5A 51 E6 80 80 FA FE
( ) 0D30 00 FB 28 5A 5A 51 48 5A 48 D1 5A 5A 51 48 DA E0
( ) 0D40 5A 5A 51 48 44 48 51 5A 60 79 6C 60 DA DA FA FE
( ) 0D50 FF 5A 5A 5A 5A 5A 5A 66 72 79 E6 E6 80 00 56 56
( ) 0D60 56 56 56 56 5A 66 F2 80 80 4C 4B 4C 4C 4C 4C 56
( ) 0D70 5A 56 4C 00 C4 44 4C 56 5A 5A 56 5A 66 56 5A 66
( ) 0D80 F2 80 FE 00 00 72 5A CC 72 5A CC 72 5A CC 80 B8
( ) 0D90 80 4C 56 5A 56 5A E6 F2 80 FA FE 00 56 52 4D AF
( ) 0DA0 4D AF 4D FC 06 AF FC 02 FE FF 2F 29 26 24 2F 29
( ) 0DB0 A4 32 A9 FC 06 AF FC 02 FE 00 56 52 4D AF 4D AF
( ) 0DC0 4D FC 06 AF FC 02 FE FF 39 40 44 39 2F A4 29 2F
( ) 0DD0 39 A9 80 80 FE 00 56 52 4D AF 4D AF 4D FC 06 AF
( ) 0DE0 FC 02 FE FF 2F 29 26 24 2F 29 A4 32 A9 AF 80 80
( ) 0DF0 2F 29 24 2F 29 A4 2F 29 2F 24 2F 29 A4 2F 29 2F
( ) 0E00 24 2F 29 A4 32 A9 AF 80 80 FA FF FF FF FF FF FF
```

zu Abb. 7.2.3

Zeitablauf jederzeit unterbrechen läßt. Diese Löschtaste führt zum sofortigen Abspielen einer der vier Melodien, danach läßt sich das Gerät wieder für eine neue Zeit programmieren. Da unbeschaltete Eingänge des Bausteins 6532 im EMUF wegen der integrierten Pull-Up-Widerstände ohnehin auf High-Pegel liegen, ist eine externe Beschaltung mit Widerständen nicht mehr erforderlich.

Nun zum eigentlichen Programm. Der Reset-Vektor bei 0FFC führt hier auf die Adresse 0C00, an der der Mikrocomputer beim Einschalten mit der Abarbeitung des Programms beginnt. Wie üblich, werden zunächst die Ausgänge am Port PB deklariert (PA ist nach einem Reset ohnehin stets als Eingang geschaltet), der Stackpointer wird auf hex FF gesetzt und die Interrupt- und Dezimal-Flags im Statusregister der CPU werden initialisiert.

Nach dem Label FRST setzt das Programm dann die als Zeiger für die Notentabelle dienenden Speicherzellen auf die erste Melodie. Dann wird solange gewartet, bis der Benutzer eine beliebige Taste drückt. Sobald das geschehen ist, liegt das zugehörige Bit im Register PA auf 0. Die Bitposition wird in der Schleife SRCH für die Weiterverwendung als Tabellenindex im X-Register umgerechnet. Dieser Index dient bei FND nun dazu, die gewünschte Zeit in Minuten in den Akku zu laden. Sie dient dann als Startwert für eine Zählschleife im X-Register, wobei ein Schleifendurchgang genau eine Minute dauert. Dafür werden zwei weitere, ineinander verschachtelte Schleifen verwendet: Einmal liefert der Timer eine Verzögerung von 255 × 1024 µs, und eine weitere mit dem Y-Register gebildete Schleife arbeitet die Timer-Schleife 230mal ab. Ist die Zeit abgelaufen oder wurde inzwischen die Löschtaste an PA7 gedrückt, so wird eine Melodie gespielt, wonach das Programm auf eine neue Eingabe wartet.

Das Musikprogramm (es beginnt beim Label GO) benutzt eine Notentabelle, die an der Adresse 0D00 beginnt. Sie enthält nicht nur die codierten Tonhöhen, sondern auch bestimmte Steuerzeichen, um Klang und Geschwindigkeit zu beeinflussen:

Steuercode	*Wirkung*
FB	Geschwindigkeit ändern: 18 = schnell, 60 = langsam, 30 = normal.
FC	Dauer langer Noten: = doppelt so lang wie kurze Note, 3 = 3mal so lang usw.
FD	Oktave setzen: 2 = Baß, 4 = tiefer Baß (normal = 01).
FE	Instrument setzen: FF = Piano, 00 = Klarinette.
FA	Ende einer Melodie.
FF	Ende aller Melodien, rücksetzen auf die erste.

Erwähnenswert ist noch, daß die Klangart „Klarinette“ am Lautsprecherausgang ein symmetrisches Rechtecksignal liefert, „Piano“ dagegen eine Rechteckschwingung mit abnehmender Impulsbreite, was die Lautstärke mit der Tondauer wie beim Klavier verringert.

Die Toncodierung selbst enthält nicht nur die Tonhöheninformation, sondern auch (im höchstwertigen Bit) eine Information über die Länge des Tons. Die in folgender Tabelle angegebene Codierung umfaßt zwei Oktaven. Dabei sind auch alle Halbtöne berücksichtigt:

Ton	tief		hoch	
	hex (kurz)	hex (lang)	hex (kurz)	hex (lang)
A	75	F5	39	B9
A#	6E	EE	35	B5
B	68	E8	32	B2
C	62	E2	2F	AF
C#	5C	DC	2C	AC
D	56	D6	29	A9
D#	52	D2	26	A6
E	4D	CD	24	A4
F	48	C8	22	A2
F#	44	C4	20	A0
G	40	C0	1E	9E
G#	3C	BC	1C	9C
Pause	00	80		

Eine solche „Eieruhr" (*Abb. 7.2.4*) ließe sich rein prinzipiell natürlich auch in herkömmlicher TTL- oder CMOS-Technik aufbauen. Beim Versuch, sie auch mit einer Musikausgabe auszurüsten, würde man aber recht schnell auf einen hohen Hardware-Aufwand kommen, wogegen sich das Gerät mit dem EMUF für kaum mehr als 100 DM realisieren läßt.

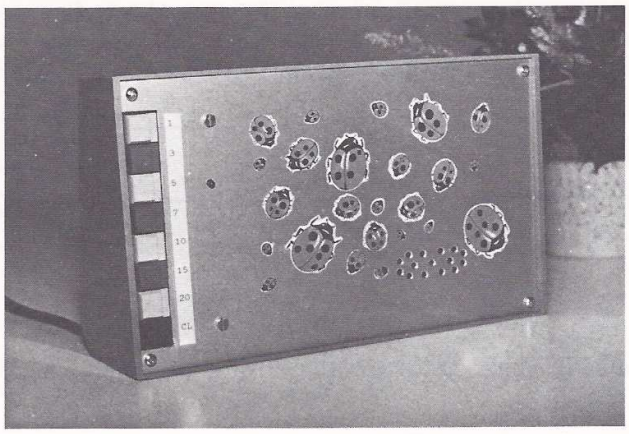


Abb. 7.2.4 Musteraufbau der Mikrocomputer-Eieruhr

7.3 Funkfernschreib-Decoder

Unsere nächste EMUF-Applikation ermöglicht es, Funkfernschreibsendungen im Baudot-Code zu empfangen und auf einem handelsüblichen Fernsehgerät mitzuschreiben. Dazu ist lediglich noch ein handelsübliches Video-Interface erforderlich, das die vom

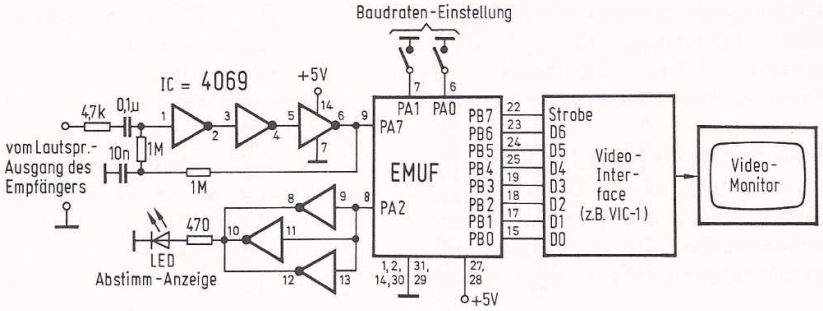


Abb. 7.3.1 Schaltung des Funkfernschreib-Empfangssystems. Als Video-Interface eignet sich jede handelsübliche Platine mit parallelem ASCII-Eingang

EMUF gelieferten parallelen ASCII-Daten in ein Videosignal umformt. Eingangsseitig wird der EMUF über eine aus drei Invertern bestehende Verstärkerschaltung mit dem Lautsprecher-Ausgang des Empfängers verbunden (*Abb. 7.3.1*). Bei Funkfernschreibsendungen, die im UKW-Bereich übertragen werden (144,6 MHz oder 145,3 MHz im 2-m-Amateurband) sind normalerweise keine weiteren Filtermaßnahmen erforderlich. Bei stark gestörtem Empfang im Kurzwellenbereich kann es aber erforderlich sein, hinter dem Lautsprecher-Ausgang noch ein Filter vorzusehen, das nur die beiden Frequenzen durchläßt, mit denen 0 und 1 codiert werden und die man im Fachjargon auch Space und Mark nennt – üblicherweise 1275 Hz und 2125 Hz.

Beim Funkfernschreiben sind unterschiedliche Übertragungsgeschwindigkeiten üblich. Sie lassen sich mit zwei Schaltern an den Ports PA1 und PA0 einstellen:

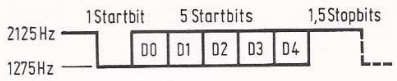
Baud-rate	Schalter an PA1	PA0
45	geschl.	geschl.
50	geschl.	offen
75	offen	geschl.
100	offen	offen

Der Einplatinencomputer EMUF übernimmt in dieser Applikation die folgenden Aufgaben:

1. Decodieren der Töne für Mark und Space durch Autokorrelation.
2. Umwandlung des empfangenen Baudot-Codes in ASCII.
3. Parallele Ausgabe der ASCII-Zeichen an ein Video-Interface.
4. Ansteuern einer Leuchtdiode, die als Abstimmhilfe für den Empfänger dient.

Doch zunächst noch ein paar Worte zum Baudot-Code (*Abb. 7.3.2*). Bei ihm wird jedes Zeichen mit nur fünf Bits codiert. Man kann sich leicht überlegen, daß damit nur $2^5 = 32$ unterschiedliche Zeichen darstellbar sind. Dies würde nicht einmal für die Menge aller Großbuchstaben, Ziffern und Satzzeichen ausreichen. Deshalb arbeiten

Abb. 7.3.2 Format der Funkfernschreib-Übertragung im Baudot-Code



Baudot-Fernschreiber in zwei „Zeichenebenen". Die erste Ebene enthält dabei alle Großbuchstaben, die zweite alle Ziffern und Satzzeichen. Die Umschaltung zwischen beiden Ebenen erfolgt mit besonderen Steuerzeichen: Die Bitfolge 11111 schaltet auf die Buchstabenebene, und die Bitfolge 1101 auf die Ziffern- und Zeichenebene. Nimmt man beide Ebenen zusammen, so ergeben sich insgesamt 64 mögliche Zeichen, von denen aber nicht alle verwertbar sind, denn die Umschaltzeichen selbst existieren in beiden Ebenen. Der Zeichenvorrat reicht also für eine gemischte Groß- und Kleinschreibung nicht aus. Baudot-Fernschreiber drucken deshalb entweder nur in Groß- oder Kleinbuchstaben. (Das hier abgedruckte EMUF-Programm codiert alle Baudot-Buchstaben in ASCII-Großbuchstaben um.)

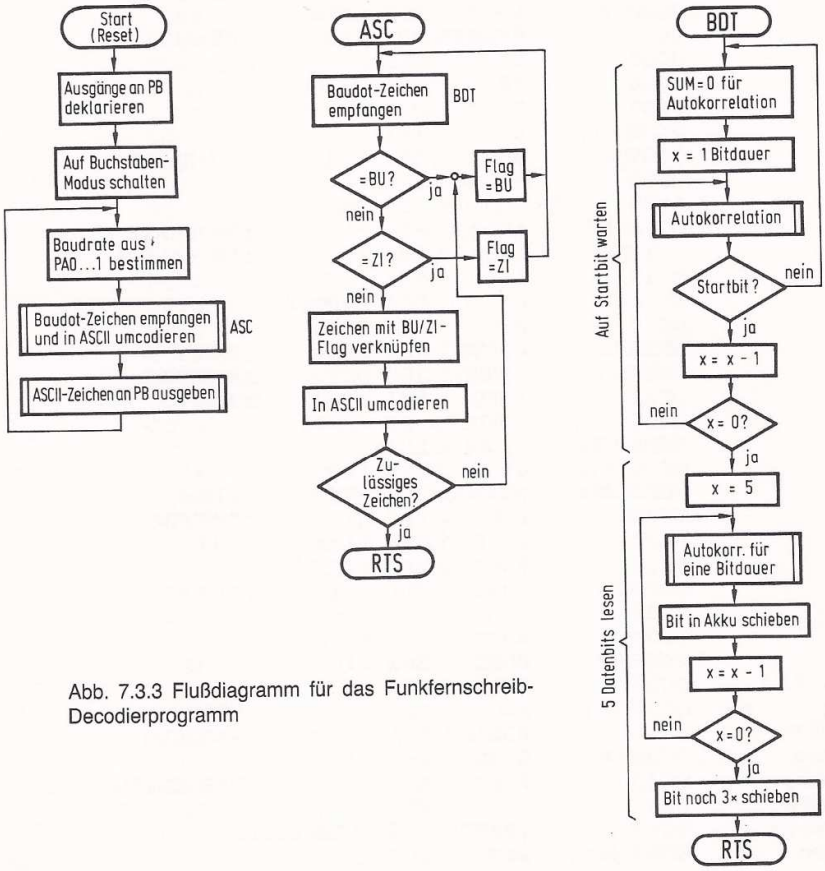


Abb. 7.3.3 Flußdiagramm für das Funkfernschreib-Decodierprogramm

```
                FF10        PASS 1
                0FFE        PASS 2
                0000
                0000        ;EMUF ALS RTTY-RX
                0000              .OPT GEN     ;F.TABELLE
                0000        ;PB=ASCII-AUSGANG
                0000        ;PB7=DAV/STROBE
                0000        ;PA 0/1=BAUDRATE
                0000        ;PA7=NF-EINGANG
                0000        ;PA2=LED-AUSGANG
                0000 PA           =$800
                0000 PAD          =$801
                0000 PB           =$802
                0000 PBD          =$803
                0000 BDR          *=*+1        ;BAUDRATE
                0001 FLG          *=*+1        ;BU/ZI
                0002 SUM          *=*+1        ;AUTOKORR.
                0003 SPL1         *=*+1        ;BIT-
                0004 SPL2         *=$C00       ;MUSTER
                0C00 RES    A2FF  LDX £$FF
                0C02        8E0308 STX PBD     ;PB=AUSG.
                0C05        9A    TXS
                0C06        78    SEI
                0C07        D8    CLD
                0C08        E8    INX
                0C09        8601  STX FLG      ;BU-MODUS
                0C0B        A904  LDA £4
                0C0D        8D0108 STA PAD
                0C10 LP     AD0008 LDA PA      ;BAUDRATE
                0C13        2903  AND £3       ;IN PA 0-1
                0C15        AA    TAX
                0C16        BD100D LDA BDRT,X
                0C19        8500  STA BDR
                0C1B        20270C JSR ASC     ;EMPFANG
                0C1E        20BB0C JSR OUT     ;AUSGABE
                0C21        30ED  BMI LP       ;JUMP
                0C23        ;BAUDOT ZU ASCII
                0C23 LTR    A900  LDA £0
                0C25 FIG    8501  STA FLG
                0C27 ASC    20430C JSR BDT     ;EIN-
                0C2A        F0F7  BEQ LTR      ;SPRUNG
                0C2C        C91F  CMP £$1F     ;BU?
                0C2E        F0F3  BEQ LTR
                0C30        C91B  CMP £$1B     ;ZI?
                0C32        D004  BNE *+6
                0C34        A920  LDA £$20
                0C36        D0ED  BNE FIG
                0C38        0501  ORA FLG
                0C3A        AA    TAX          ;CODE
                0C3B        BDD10C LDA TAB,X   ;WANDELN
                0C3E        C940  CMP £'§
                0C40        F0E1  BEQ LTR      ;UNGUELTIG
                0C42        60    RTS
                0C43        ;BAUDOT-ZEICHEN LESEN
                0C43 BDT    A200  LDX £0
```

Abb. 7.3.4 Listing des EMUF-Programms zum Empfang von Funkfernschreib-Sendungen

```
0C45          8602    STX SUM
0C47          A600    LDX BDR       ;STARTBIT-
0C49          CA      DEX           ;ZEIT
0C4A BDTO     20700C  JSR RD
0C4D          2402    BIT SUM       ;STARTBIT?
0C4F          30F2    BMI BDT       ;NEIN
0C51          CA      DEX
0C52          DOF6    BNE BDTO
0C54          A205    LDX £5        ;5BITS
0C56 BDT1     20610C  JSR BITS
0C59          6A      ROR A
0C5A          CA      DEX
0C5B          DOF9    BNE BDT1
0C5D          4A      LSR A
0C5E          4A      LSR A         ;BAUDOT-
0C5F          4A      LSR A         ;ZEICHEN
0C60          60      RTS           ;IN A
0C61          ;BAUDOT-BIT LESEN
0C61 BITS     A000    LDY £O        ;TESTE
0C63          8402    STY SUM       ;DATENBIT
0C65          A400    LDY BDR
0C67 BIT1     20700C  JSR RD
0C6A          88      DEY
0C6B          DOFA    BNE BIT1
0C6D          2602    ROL SUM       ;MARK:
0C6F          60      RTS           ;C=1
0C70          ;AUTOKORRELATION
0C70          ;T=1.1MS,F=2.19KHZ
0C70 RD       48      PHA           ;A,Y,X
0C71          98      TYA           ;RETTEN
0C72          48      PHA
0C73          8A      TXA
0C74          48      PHA
0C75          A010    LDY £16       ;2X8BITS
0C77 RD1      A207    LDX £7        ;2193HZ
0C79 RDO      CA      DEX
0C7A          DOFD    BNE RDO
0C7C          AD0008  LDA PA        ;PA7=EING.
0C7F          OA      ASL A
0C80          6603    ROR SPL1      ;MUSTER
0C82          6604    ROR SPL2      ;SPEICHERN
0C84          88      DEY
0C85          DOFO    BNE RD1
0C87          AD0008  LDA PA        ;LED AUS
0C8A          0904    ORA £4
0C8C          8D0008  STA PA
0C8F          A503    LDA SPL1      ;MUSTER 1
0C91          F01C    BEQ ERR
0C93          C9FF    CMP £$FF      ;KEIN SIG.
0C95          F018    BEQ ERR
0C97          4504    EOR SPL2      ;MUSTER 2
0C99          DOOA    BNE CHK
0C9B          A8      TAY
0C9C          AD0008  LDA PA
0C9F          29FB    AND £$FB      ;LED EIN
```

zu Abb. 7.3.4

```
OCA1          8D0008 STA PA
OCA4          98     TYA
OCA5 CHK      A007   LDY £7       ;8 BITS
OCA7 CHK1     6A     ROR A
OCA8          B002   BCS CHK2
OCAA          C602   DEC SUM
OCAC CHK2     88     DEY
OCAD          10F8   BPL CHK1
OCAF ERR      A502   LDA SUM
OCB1          6904   ADC £4       ;BEWERTUNG
OCB3          8502   STA SUM      ;ADDIEREN
OCB5          68     PLA
OCB6          AA     TAX          ;X,Y,A
OCB7          68     PLA          ;RUECK-
OCB8          A8     TAY          ;SPEICHERN
OCB9          68     PLA
OCBA          60     RTS
OCBB          ;ASCII-AUSGABE
OCBB OUT      48     PHA
OCBC          AD0208 LDA PB       ;ALTES
OCBF          297F   AND £$7F     ;ZEICHEN
OCC1          8D0208 STA PB       ;DAV=0
OCC4          68     PLA
OCC5          8D0208 STA PB       ;NEUES
OCC8          EA     NOP          ;ZEICHEN
OCC9          EA     NOP
OCCA          EA     NOP
OCCB          0980   ORA £$80
OCCD          8D0208 STA PB       ;DAV=1
OCD0          60     RTS
OCD1          ;BAUDOT/ASCII-TAB.
OCD1 TAB      4045   .BYT '§E',$A,'A SIU',$D
OCD3          0A
OCD4          4120
OCD6          534955
OCD9          0D
OCDA          4452   .BYT 'DRJNFCKTZLWHYPQOBG§'
OCDC          4A4E
OCDE          4643
OCE0          4B54
OCE2          5A4C
OCE4          5748
OCE6          5950
OCE8          514F
OCEA          424740
OCED          4D58   .BYT 'MXV§§3',$A,'- ',$27
OCEF          5640
OCF1          4033
OCF3          0A
OCF4          2D20
OCF6          27
OCF7          3837   .BYT '87',$D,'§4;'
OCF9          0D
OCFA          40343B
OCFD          2C40   .BYT ',§:(5+)2!6019?§§./='
```

```
OCFF        3A28
OD01        352B
OD03        2932
OD05        2136
OD07        3031
OD09        393F
OD0B        4040
OD0D        2E2F3D
OD10
OD10        ;BAUDRATEN-TABELLE
OD10 BDRT   14      .BYT 20      ;45 BD
OD11        12      .BYT 18      ;50 BD
OD12        0C      .BYT 12      ;75 BD
OD13        09      .BYT 9       ;100 BD    zu Abb. 7.4.3
OD14                *=$FFC       ;RESET-
OFFC        000C    .WOR RES     ;VEKTOR
OFFE                .END
```

Wie bei der asynchronen ASCII-Datenübertragung wird auch den Baudot-Datenbits ein Startbit vorangestellt, das immer 0 ist. Der Mindestabstand zwischen dem letzten Datenbit eines Zeichens und dem Startbit des nächsten Zeichens (Stopbitlänge) entspricht der 1,5fachen Dauer eines Datenbits.

Das EMUF-Programm ist modular aufgebaut, wie auch aus den Flußdiagrammen (*Abb. 7.3.3*) ersichtlich ist. Es verwendet eine Speicherzelle als „Flag", mit der sich der Computer merkt, in welcher Zeichenebene gerade gearbeitet wird. Jedes empfangene Baudot-Zeichen wird mit dem Flag logisch ODER-verknüpft, so daß schließlich ein 6-Bit-Zeichen zur Verfügung steht. Der 6-Bit-Wert dient nun als Index für eine Codeumwandlungstabelle, in der die zugehörigen ASCII-Zeichen stehen. Die parallele Ausgabe an das Video-Interface entspricht der Routine, die bereits im vorhergehenden Kapitel vorgestellt wurde (*Abb. 7.3.4*).

In die Autokorrelations-Routine, die auf die Mark-Frequenz abgestimmt ist, wurde die Ansteuerung einer Abstimmanzeige-Leuchtdiode eingebaut, die immer dann leuchtet, wenn die Mark-Frequenz anliegt. Wie man sich ausrechnen kann, ist die Mittenfrequenz der Autokorrelations-Routine 2,193 kHz, was ein wenig von der genormten Mark-Frequenz von 2,125 kHz abweicht. Wegen der genügend großen Decodierungs-Bandbreite spielt dies jedoch keine Rolle.

7.4 IEC-Bus/V.24-Interface

Der sogenannte IEC-Bus wurde bereits an anderer Stelle in diesem Buch beschrieben. Er dient dazu, um mehrere Peripherie-Geräte über eine gemeinsame Gruppe von Leitungen, eben über diesen Bus, mit einem Tischcomputer zu verbinden (*Abb. 7.4.1*). Besitzt man nun einen Computer, der über eine IEC-Busschnittstelle verfügt, aber gleichzeitig einen daran anzuschließenden Drucker, der nur eine V.24- oder RS-232-Schnittstelle

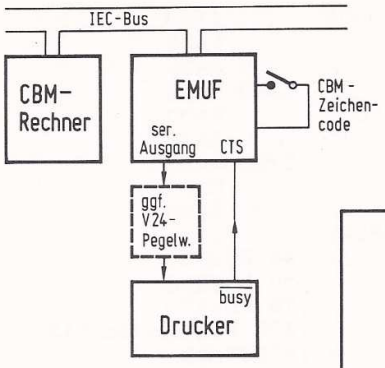


Abb. 7.4.1 Typische IEC-Bus-Konfiguration

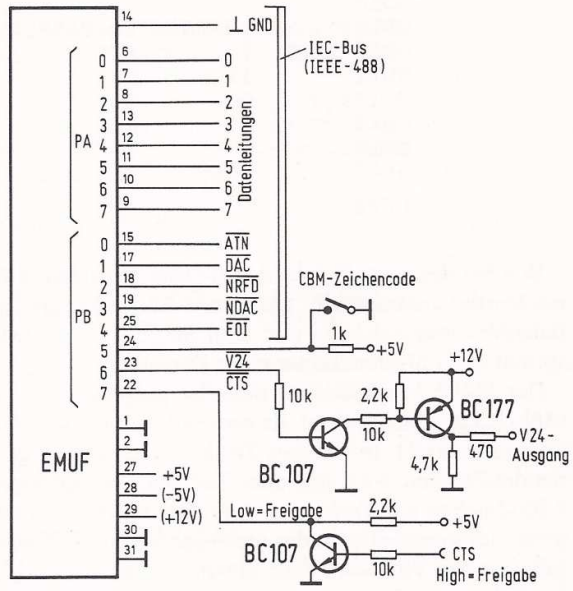


Abb. 7.4.2 Schaltung des IEC-Bus/V.24-Interface

besitzt, benötigt man eine Anpaßschaltung [18], die man „Interface" nennt. Ein solches Interface läßt sich leicht mit dem EMUF realisieren (*Abb. 7.4.2*).

Außer dem Masseanschluß führen zum Drucker zwei Leitungen: Eine Datenleitung, auf der logisch 0 mit +12 V codiert wird und logisch 1 mit −12 V, und eine Handshake-Leitung, über die der Drucker dem Interface mitteilen kann, wann er für ein neues Zeichen aufnahmebereit ist. Diese zweite Leitung heißt CTS (Clear to send); solange der Drucker noch mit der Ausgabe des letzten Zeichens beschäftigt ist, legt er sie auf logisch 0.

An PB5 ist hier ein Schalter vorgesehen, mit dem wahlweise eine Umcodierung der von den Commodore-CBM-Tischcomputern verwendeten Zeichencodes vorgenommen werden kann. Wenn dieser Schalter offen ist, werden die vom IEC-Bus kommenden Zeichen unverändert an die serielle V.24-Schnittstelle weitergegeben. Ist er dagegen geschlossen, so werden die vom IEC-Bus kommenden Codes für Klein- und Großbuchstaben an die ASCII-Norm angepaßt. *Abb. 7.4.3* zeigt das Assembler-Listing der EMUF-Software.

```
                        ;       EMUF    IECV24 INTERFACE 810814
                        ;       ROLF-DIETER KLEIN
                        ;
0050                            WIDTH 80
                        ;PORT A
                        ;    7 .. 0  IEC DATABUS
                        ;PORT B
                        ;    7      6      5      4      3      2      1      0
                        ;   -CTS   -V24   PET    -EOI  -NDAC -NRDF   -DAC   -ATN
                        ; -CTS = LOW DANN FREI
                        ; -V24 RUHEPEGEL = LOW
                        ; PET = 0 DANN PETUMWANDLUNG
                        ;
                        ;
0800                    PA      EQU     $800
0801                    PAD     EQU     $801
0802                    PB      EQU     $802
0803                    PBD     EQU     $803
                        ;
0814                    TIM1    EQU     $814
0815                    TIM8    EQU     $815
0816                    TIM64   EQU     $816
0816                    TIMIN   EQU     $816
0817                    TIMFLG  EQU     $817
                        ;
0000                    XTEMP1  EQU     $0
0008                    ZOUT    EQU     8
0009                    ZCOU    EQU     9
000A                    FLAG    EQU     $A      ;IEC MERKER ATN ..
000B                    ZEICH   EQU     $B      ;ZWSPEICHER
000C                    COUNT   EQU     $C      ;ZWSPEICHER
000D                    PADR    EQU     $D      ;FIRST TIME
000E                    MDE     EQU     $E      ;MODE 0,1,2
000F                    CNT     EQU     $F      ;ZEITSCHL.
                        ;
                        ;
                        ;
                        ; INIT ROUTINE
0FFC                            ORG     $FFC
0FFC 000C                       DW      $C00
                        ;
0C00                            ORG     $C00    ;START
0C00 A2FF               RESET:  LDX     #$FF    ;STACKPOINTER
0C02 9A                         TXS
0C03 A900                       LDA     #%00000000      ;ALL INPUT
0C05 8D0108                     STA     PAD
0C08 A94C                       LDA     #%01001100      ;SET UP
0C0A 8D0308                     STA     PBD
0C0D A900                       LDA     #%00000000      ;NOT READY NOT ACCEPT
0C0F D8                         CLD
0C10 78                         SEI
```

Abb. 7.4.3 Listing des Interface-Assemblerprogramms für den EMUF

```
0C11 A9FF           LDA     #$FF
0C13 850D           STA     PADR    ;START WERT
0C15 A903           LDA     #3      ;1200 VOREINST
0C17 20270C         JSR     BAUD
0C1A A90D           LDA     #$D     ;CR AUS TEST
0C1C 206A0C         JSR     V24OUT
0C1F A90A           LDA     #$A
0C21 206A0C         JSR     V24OUT  ;LF AUS TEST
0C24
0C24 4C440D         JMP     MAIN
                ; UPRGE
                ;
                ;
                ; V24 ROUTINEN
0C27            BAUD:           ;EINSTELLEN BAUDRATE
0C27                            ;0..6 110,300,600,1200,2400,4800,9600
0C27 C907           CMP     #7
0C29 9001           BCC     OKSK    ;FEHLERTEST
0C2B 60             RTS
0C2C 0A         OKSK:   ASL     A       ;*2 WEGEN TABELLE
0C2D A8             TAY
0C2E B9399C         LDA     TABBAU,Y
0C31 850E           STA     MDE     ;MODE 0..2
0C33 B93A0C         LDA     TABBAU+1,Y
0C36 850F           STA     CNT     ;COUNT DOWN
0C38 60             RTS
                ;
0C39            TABBAU:
0C39 080D           DB      8,13    ;110
0C3B 0398           DB      3,152   ;300
0C3D 0241           DB      2,65    ;600
0C3F 019F           DB      1,159   ;1200
0C41 014C           DB      1,76    ;2400
0C43 0122           DB      1,34    ;4800
0C45 010D           DB      1,13    ;9600
                ;
                ;
                ;
                ;
                ;
0C47 48         PETASC: PHA
0C48 AD0208         LDA     PB      ;TEST PETFLAG
0C4B 2920           AND     #%00100000
0C4D F002           BEQ     CONV    ;UMWANDELN WENN 0
0C4F 68             PLA
0C50 60             RTS
0C51            CONV:           ;WANDELN BEREICHE
0C51                            ;64..95->KLEINBU
0C51                            ;192..223->GROSSBU
0C51 68             PLA
0C52 C940           CMP     #64
```

zu Abb. 7.4.3

```
0C54 9013               BCC     CONV1
0C56 C960               CMP     #96
0C58 B004               BCS     CONV2   ;BEREICH
0C5A 18                 CLC
0C5B 6920               ADC     #32     ;IN KLEINBU
0C5D 60                 RTS
0C5E C9C0       CONV2:  CMP     #192
0C60 9007               BCC     CONV1
0C62 C9E0               CMP     #224
0C64 B003               BCS     CONV1
0C66 297F               AND     #$7F    ;RESET BIT
0C68 60                 RTS
0C69 60         CONV1:  RTS             ;OK SCHON
                ;
                ;
0C6A 48         V24OUT: PHA
0C6B AD0208     V24LP:  LDA     PB      ;CTS WARTEN BIS LOW
0C6E 2980               AND     #%10000000
0C70 D0F9               BNE     V24LP
0C72 68                 PLA
0C73 8508               STA     ZOUT    ;TEMP DATENWERT
0C75 A90B               LDA     #11     ;11 BITS
0C77 8569               STA     ZCOU
0C79 18         VLO:    CLC             ;CARRY = 0 FUER STARTBIT
0C7A 2608               ROL     ZOUT    ;IN BIT 0 ZUNAECHST
0C7C 6608       VLOP:   ROR     ZOUT    ;ZURUECK INS CARRY
0C7E B00B               BCS     V241    ;FALLS 1 DANN SPR
0C80 AD0208             LDA     PB
0C83 0940               ORA     #%01000000
0C85 8D0208             STA     PB      ;0 AUSGEBEN
0C88 4C946C             JMP     VSK
0C8B AD0208     V241:   LDA     PB
0C8E 29BF               AND     #%10111111
0C90 8D0208             STA     PB
0C93 EA                 NOP             ;ZEITANGLEICH 15YS
0C94 A69E       VSK:    LDX     MDE
0C96 A40F               LDY     CNT     ;ZEITWERTE
0C98 88         VL1:    DEY
0C99 D0FD               BNE     VL1
0C9B CA                 DEX
0C9C D0FA               BNE     VL1
0C9E 38                 SEC             ;FUER STOPP BITS
0C9F C669               DEC     ZCOU    ;SCHLEIFENZAEHLER
0CA1 D0D9               BNE     VLOP
0CA3 60                 RTS
                ;
                ; 36YS + WAI IN DEY..BNE
                ;
                ; IEC ROUTINEN
                ;
0CA4 A904       GETCHA: LDA     #%00000100      ;RDF
```

zu Abb. 7.4.3

```
0CA6 8D0208          STA   PB
0CA9 AD0208  LOPA:   LDA   PB
0CAC 2902            AND   #%00000010      ;DAV WARTEN
0CAE D0F9            BNE   LOPA
0CB0 A900            LDA   #%00000000
0CB2 8D0208          STA   PB
0CB5 AD0008          LDA   PA       ;DATA HOLEN
0CB8 49FF            EOR   #%11111111
0CBA 48              PHA   ;RETTEN
0CBB AD0208          LDA   PB
0CBE 850A            STA   FLAG
0CC0 A908            LDA   #%00001000      ;DAC
0CC2 8D0208          STA   PB
0CC5 AD0208  LOPB:   LDA   PB
0CC8 2902            AND   #%00000010
0CCA F0F9            BEQ   LOPB     ;DAV HIGH
0CCC A900            LDA   #%00000000
0CCE 8D0208          STA   PB
0CD1 68              PLA            ;DATA WERT
0CD2 60              RTS
                     ;
0CD3 AD0208  TALKON: LDA   PB
0CD6 2901            AND   #%00000001
0CD8 F0F9            BEQ   TALKON   ;WARTEN BIS ATN WEG
0CDA A912            LDA   #%00010010
0CDC 8D0208          STA   PB       ;DAV HIGH
0CDF A9FF            LDA   #%11111111
0CE1 8D0108          STA   PAD      ;DATA CHANGE
0CE4 A952            LDA   #%01010010
0CE6 8D0308          STA   PBD
0CE9 A912            LDA   #%00010010
0CEB 8D0208          STA   PB       ;SAVETY
0CEE 60              RTS
                     ;
0CEF A900    TALKOF: LDA   #%00000000
0CF1 8D0108          STA   PAD
0CF4 A900            LDA   #%00000000
0CF6 8D0208          STA   PB
0CF9 A94C            LDA   #%01001100
0CFB 8D0308          STA   PBD
0CFE A900            LDA   #%00000000
0D00 8D0208          STA   PB
0D03 60              RTS
                     ;
                     ;
0D04 48      SEND1:  PHA
0D05 202A0D  SEND:   JSR   CKATN    ;TESTER GGF
0D08 AD0208          LDA   PB
0D0B 2904            AND   #%00000100
0D0D F0F6            BEQ   SEND     ;WARTEN
0D0F 68              PLA            ;WERT HOLEN
```

zu Abb. 7.4.3

```
0D10 49FF            EOR     #%11111111
0D12 8D0008          STA     PA
0D15 A910            LDA     #%00010000      ;DAV
0D17 8D0208          STA     PB
0D1A 202A0D  CONSE:  JSR     CKATN
0D1D AD0208          LDA     PB
0D20 2908            AND     #%00001000
0D22 F0F6            BEQ     CONSE
0D24 A912            LDA     #%00010010      ;PASSIV
0D26 8D0208          STA     PB
0D29 60              RTS
                     ;
0D2A 60      CKATN:  RTS     ;Z.Z.
                     ;
0D2B 48      SENEOI: PHA
0D2C 202A0D  SENEOI: JSR     CKATN
0D2F AD0208          LDA     PB
0D32 2904            AND     #%00000100
0D34 F0F6            BEQ     SENEOI
0D36 68              PLA
0D37 49FF            EOR     #%11111111
0D39 8D0008          STA     PA
0D3C A900            LDA     #%0000000       ;EOI DAV
0D3E 8D0208          STA     PB
0D41 4C1A0D          JMP     CONSE
                     ;
0D44         MAIN:
0D44 20A40C          JSR     GETCHA          ;IEC ZEICHEN
0D47 850B            STA     ZEICH   ;RETTEN
0D49         MAIN1:
0D49 A50A            LDA     FLAG
0D4B 2901            AND     #%00000001      ;ATN
0D4D D0F5            BNE     MAIN    ;HIGH DANN NEIN
0D4F A50B            LDA     ZEICH
0D51 2960            AND     #$60    ;TEST LISTEN TALK
0D53 C920            CMP     #$20
0D55 F00A            BEQ     LISTPA
0D57 C940            CMP     #$40
0D59 F003            BEQ     TALPA
0D5B 4C440D          JMP     MAIN
                     ;
0D5E 4C440D  TALPA:  JMP     MAIN    ;KEIN TALKER
                     ;
                     ;
0D61 A50D    LISTPA: LDA     PADR    ;PRIM ADRESSE
0D63 C9FF            CMP     #$FF
0D65 D011            BNE     SK2     ;WEITER SONST
0D67 A50B            LDA     ZEICH
0D69 290F            AND     #$0F
0D6B 850D            STA     PADR    ;NEUE ADRESSE
0D6D E60D            INC     PADR    ;+1 HIERT IMMER BEI EMUF V24
```

zu Abb. 7.4.3

```
0D6F A50D            LDA   PADR
0D71 290F            AND   #$0F
0D73 850D            STA   PADR
0D75 4C800D          JMP   SK3
0D78 A50B      SK2:  LDA   ZEICH
0D7A 290F            AND   #$0F
0D7C C50D            CMP   PADR     ;VERGLEICH
0D7E D0C4            BNE   MAIN
0D80           SK3:
0D80 20A40C          JSR   GETCHA
0D83 850B            STA   ZEICH    ;TEST OB SA
0D85 A50A            LDA   FLAG
0D87 2901            AND   #%00000001  ;WENN ATN DANN SA
0D89 D00C            BNE   DATEN    ;SCHON DATEN
0D8B A50B            LDA   ZEICH
0D8D 290F            AND   #$0F     ;NUN BAUDRATE EINSTELLEN
0D8F 20270C          JSR   BAUD
0D92           LOPMAI:             ;HAUPTSCHLEIFE DATEN AUSGEBEN
0D92 20A40C          JSR   GETCHA
0D95 850B            STA   ZEICH    ;ZWSP
0D97 A50B      DATEN: LDA   ZEICH    ;AUSGEBEN
0D99 20470C          JSR   PETASC   ;UMWANDELN GGF
0D9C 206A0C          JSR   V24OUT
0D9F A50A            LDA   FLAG
0DA1 2910            AND   #%00010000       ;TEST EOI
0DA3 F009            BEQ   FINA
0DA5 A50A            LDA   FLAG
0DA7 2901            AND   #%00000001
0DA9 D0E7            BNE   LOPMAI           ;HAUPTSCHLEIFE
0DAB                                        ;ABER FEHLER AUSGABE V24
0DAB 4C490D          JMP   MAIN1            ;ATN GESETZT
0DAE 4C440D    FINA: JMP   MAIN             ;OK ENDE HIER GGF ENDROUTINE
                     ;
                     ;
                     ;
                     ;
                     ;
0000                 END
```

BAUD	0C27	MDE	000E	TALPA	0D5E
CKATN	0D2A	OKSK	0C2C	TIM1	0814
CNT	000F	PA	0800	TIM64	0816
CONSE	0D1A	PAD	0801	TIM8	0815
CONV	0C51	PADR	000D	TIMFLG	0817
CONV1	0C69	PB	0802	TIMIN	0816
CONV2	0C5E	PBD	0803	V241	0C8B
COUNT	000C	PETASC	0C47	V24LP	0C6B
DATEN	0D97	RESET	0C00	V24OUT	0C6A
FINA	0DAE	SEND	0D05	VL1	0C98
FLAG	000A	SEND1	0D04	VLO	0C79
GETCHA	0CA4	SENEO1	0D2B	VLOP	0C7C
LISTPA	0D61	SENEOI	0D2C	VSK	0C94
LOPA	0CA9	SK2	0D78	XTEMP1	0000
LOPB	0CC5	SK3	0D80	ZCOU	0009
LOPMAI	0D92	TABBAU	0C39	ZEICH	000B
MAIN	0D44	TALKOF	0CEF	ZOUT	0008
MAIN1	0D49	TALKON	0CD3		

zu Abb. 7.3.4

Normalerweise besitzen Geräte, die über eine IEC-Bus-Schnittstelle verfügen, eine Gruppe von Schaltern, mit denen man die jeweilige Bus-Adresse einstellen kann. Das hier beschriebene Interface macht dies durch einen Trick überflüssig: Es merkt sich nämlich diejenige Adresse, die zuerst nach dem Einschalten des Systems auf dem Bus auftritt, erhöht sie um 1 und ist fortan unter dieser neuen Adresse ansprechbar. Ein Beispiel: Das Floppy-Laufwerk wird bei CBM-Rechnern normalerweise über die IEC-Bus-Adresse 8 angesprochen. Gewöhnlich lädt man nach dem Einschalten des Systems zunächst irgendein Programm von der Floppy in den Tischcomputer. Dies erfolgt mit dem Befehl

LOAD „Filename", 8

wobei die Adresse 8 auf den IEC-Bus gegeben wird. Der EMUF erhöht diese Adresse intern um 1 und ist deshalb ab jetzt unter der Adresse 9 ansprechbar. Nun braucht man nur noch eine Ausgabe-Datei zu eröffnen; in der Programmiersprache Basic geschieht das so:

OPEN 5,9,3

Die 5 ist dabei irgendeine beliebige Dateinummer (0...255); 9 ist die IEC-Adresse des EMUF; und die noch folgende 3 als „Sekundäradresse" stellt den EMUF auf eine V.24-Ausgabegeschwindigkeit von 1200 Baud ein. Insgesamt sind sieben unterschiedliche Baudraten wählbar:

Sekundär-Adresse	Baud-rate
0	110
1	300
2	600
3	1200
4	2400
5	4800
6	9600

Der Ausdruck eines kleinen Prüftextes könnte dann z. B. folgendermaßen geschehen:

PRINT#5, „TEST DES INTERFACE"

Um die Datei wieder zu schließen, schreibt man in Basic einfach CLOSE 5.

Dem Leser mag aufgefallen sein, daß das Assembler-Listing in seinem Schriftbild etwas von den übrigen in diesem Buch abgedruckten Listings abweicht. Der Grund dafür ist, daß die Interface-Software auf einem Entwicklungssystem erstellt wurde, das mit dem Mikroprozessor Z80 arbeitet. Deshalb wurde ein Z80/6502-Crossassembler verwendet. Ein Test des Programms ist dann natürlich nicht auf dem Entwicklungssystem, sondern erst nach der Implementierung auf den Einplatinencomputer möglich.

Feichtinger
Mit Computern steuern

Herwig Feichtinger

Mit Einplatinen-Computern steuern und regeln, ab wann sich das lohnt und wie das zu machen ist, darum geht es in diesem Buch. Der Autor zeigt, daß der technische Aufwand dafür gering ist.

Den Schwerpunkt legte der Autor auf die Beschreibung der Bauelemente und Baugruppen, also der Mikroprozessoren- und Speichertypen, die bei einem Einplatinen-Computer gebraucht werden. Er behandelt selbstverständlich nur Typen, die auf dem Markt sind und wohl auch länger bleiben werden. Damit bekommt der Leser zusätzlich eine Anleitung in die Hand, wie er seine Probleme mit einem Einplatinen-Computer lösen kann. Er lernt auch, Typisches selbst zu programmieren.

ISBN 3-7723-7221-X

Franzis'