Herwig Feichtinger

Mädchen für alles

Was hier im folgenden vorgestellt wird, ist ein fest zu programmierender, sehr preiswerter Mikrocomputer, der sich zum Beispiel als Drucker-Interface, intelligentes Bedienteil für Meßgeräte, Frequenzgenerator, Schaltuhr, Codeumsetzer oder für tausend andere Zwecke einsetzen läßt. Die Programme für ihn lassen sich mit preiswerten Tischcomputern auf 6502-Basis wie Apple, PET, CBM, AIM-65, PC-100 oder KIM-1 entwickeln; Beispiele dafür folgen in den nächsten Heften.

Wenn man von Computern spricht, meint man meist Geräte, die sich frei programmieren lassen, mit denen man eigene Programme entwickeln und testen kann und die über eine Tastatur sowie über einen Bildschirm oder wenigstens ein einfaches Display verfügen. Solche Computer bekommt man heute schon für weniger als 1000 DM.
Hier wird aber etwas ganz anderes vorgestellt, nämlich ein Mikrocomputer, der nur einmal und vor allem fest programmiert und dann für einen ganz bestimmten Verwendungszweck eingesetzt wird (*Bild 1*). Er ist also in keiner Weise dafür konstruiert, Programme mit ihm zu entwickeln, als Lehr- und Lerncomputer zu dienen oder später mit zusätzlichem Speicherplatz, ja vielleicht sogar mit einem Basic-Interpreter erweitert zu werden.

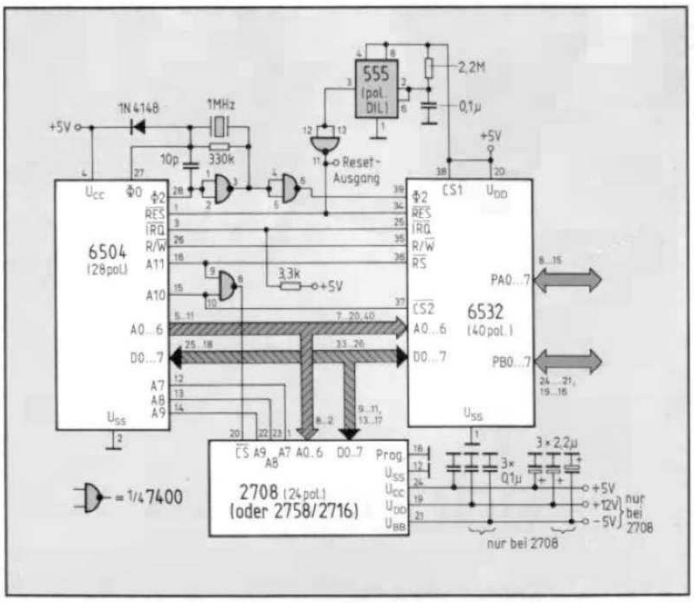

Bild 1. Gesamtschaltung des 6504-Computers mit 1 KByte ROM, 128 Byte RAM, einem programmierbaren Interrupt-Timer und 16 I/O-Leitungen

Ein Computer für weniger als hundert Mark

Unser Computerchen ist also dafür gedacht, überall dort eingesetzt zu werden, wo es im Grunde nur als Ersatz für eine vielleicht recht umfangreiche, undurchsichtige Digitalschaltung dient. So etwa in einer numerischen Steuerung, in einer Schaltuhr, in einem rechnenden Meßgerät usw., wo der Benutzer nicht selbst programmiert.
Dieses Konzept gestattet es, einen Mikrocomputer als Minimalkonfiguration mit absichtlichem Verzicht auf spätere Erweiterbarkeit und gleichzeitig als äußerst preiswerte Schaltung aufzubauen. Natürlich gibt es für diesen Zweck auch Ein-Chip-Mikrocomputer, z. T. sogar mit UV-löschbaren EPROMs – aber: ein Entwicklungssystem für einen solchen Computer kostet leider -zigtausend Mark. Bei geringen Stückzahlen treten daher enorme Kostenbelastungen auf, die die Verwendung der Ein-Chip-Mikrocomputer wieder oft als fraglich erscheinen lassen.
Unser Mikrocomputer arbeitet daher mit einer CPU, die es zuläßt, die benötigten Programme mit preiswerten Tischcomputern zu entwickeln, so etwa mit CBM, PET, AIM-65, Apple-II usw., die alle mit dem Mikroprozessor 6502 arbeiten. Bei der Übertragung des Programms auf das EPROM, das in unser Computerchen gesteckt wird, brauchen dann lediglich noch einige Adressen geändert zu werden. Zum Beispiel diejenigen für die I/O-Ports. Verwendet man einen Assembler für die Programmentwicklung, so braucht man das nicht einmal einzeln von Hand zu tun.

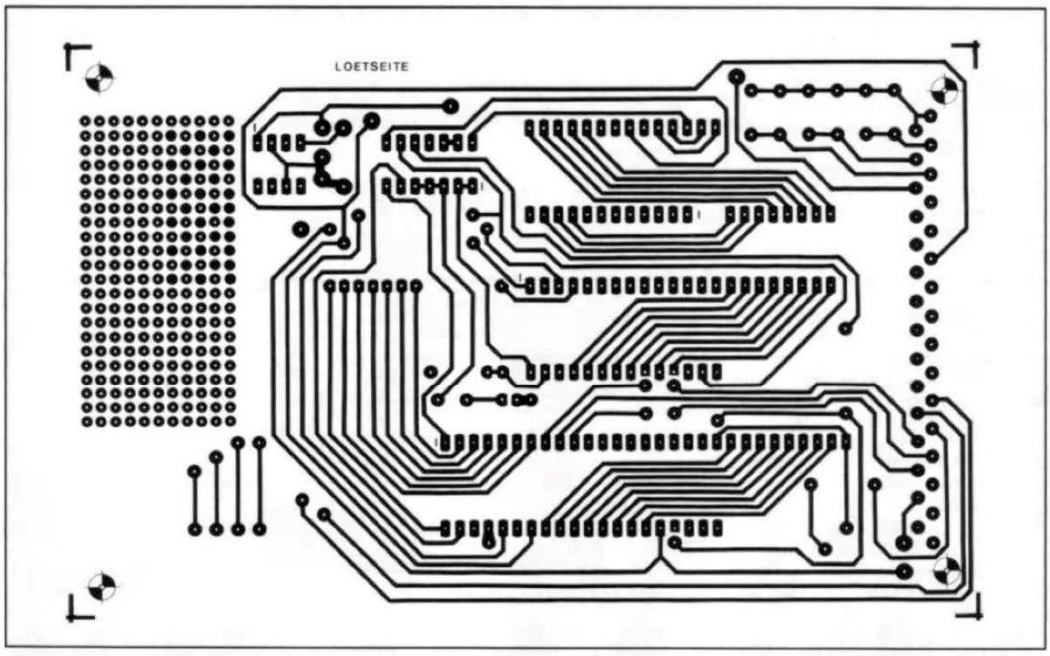

Bild 2. Lötseite der Platine. Sie enthält ein Lochraster-Feld, das vom Anwender für besondere Aufgaben frei verdrahtet werden kann, z. B. für die Nachrüstung eines D/A-Wandlers

Die Drei-Chip-Lösung hat es in sich

Wegen der Verbreitung des Prozessors 6502 bei den preiswerteren Tischcomputern wurde eine CPU aus dieser Familie gewählt, nämlich der Typ 6504. Er unterscheidet sich von der „Mutter" 6502 dadurch, daß er statt 16 nur 12 Adressenleitungen besitzt, nur einen Interrupt-Eingang herausführt ($\overline{IRQ}$), in einem 28-Pin-Gehäuse untergebracht ist (6502: 40 Pins) und nicht zuletzt deshalb auch preiswerter ist.
Wie der geneigte Leser weiß, braucht man in einem Mikrocomputer neben der CPU noch drei Dinge, nämlich einen Arbeitsspeicher (RAM), einen Eingabe/Ausgabe-Baustein (I/O), über den die Verbindung zur Außenwelt hergestellt wird und der somit dafür sorgt, daß der Computer kein Selbstzweck ist, sowie einen Programmspeicher, der hier gemäß dem Verwendungszweck als Festwertspeicher ausgeführt ist.
Um die Chip-Anzahl gering zu halten, findet hier ein Baustein namens 6532 Verwendung, der nicht nur zwei 8-Bit-I/O-Ports sowie 128 Byte RAM enthält, sondern auch einen für mancherlei Zwecke äußerst nützlich programmierbaren Interrupt-Timer, der Zeiten bis zu 261 ms liefern kann. Dazu wird nun nur noch ein EPROM benötigt, das das Betriebsprogramm enthält – in unserem Fall z. B. ein 1-KByte-Typ namens 2758, der ebenfalls schon recht preiswert zu haben ist.

Reicht denn das wirklich aus?

Wenn hier von kläglichen 128 Byte RAM und 1 KByte EPROM die Rede ist, wird manch Tischcomputer-Benutzer sagen, was soll ich damit schon anfangen? Für einen Basic-Computer wäre das tatsächlich viel zu wenig, denn allein ein Basic-Interpreter belegt ja schon rund 4...12 KByte ROM bzw. EPROM. Da Basic aber für die meisten Steuerungszwecke und für zeitkritische Aufgaben völlig ungeeignet ist, wird unser Mikro-Mikrocomputer in der Maschinensprache des verwendeten Prozessors programmiert.

Hier sei gleich vermerkt, daß der 6504 genau den gleichen Befehlssatz wie sein großer Bruder 6502 besitzt und somit zumindest softwaremäßig keinerlei Einschränkungen unterliegt. Und in 1 KByte bringt man z. B. schon ein kleines Schachprogramm unter, ein Programm zur Ansteuerung einer Schreibmaschine über eine serielle Schnittstelle, die Software zum Betrieb eines „dummen" Matrixdruckers oder vieles andere mehr. Übrigens sitzt solch ein 6504-Prozessor auch in der Floppy-Disk-Einheit CBM-3032 von Commodore – auch das ist eine Steueraufgabe, die mit einer Mikrocomputer-Minimalkonfiguration wunderbar zu lösen ist. Also keine Angst vor zu wenig Speicherplatz!

Adressierungs-Kniffe müssen sein

6502-Kenner wissen, daß dieser Prozessor zwei besondere Speicherbereiche besitzt, die beide vorhanden sein müssen, aber hardwaremäßig in ihrer Adressenlage leider mehr als 128 Bytes auseinanderliegen. Unsere 128 Byte zusammen-

hängendes RAM würden dafür nicht ausreichen: Wir brauchen einen Bereich in der „Zero Page" (0000...00FF), die nützliche Adressierungsarten bei vielen Maschinensprache-Befehlen des 6502 und die Verwendung speichersparender 2-Byte-Befehle ermöglicht, und einen weiteren in der Page 1 (0100...01FF), der die für Unterprogrammsprünge erforderlichen Rücksprungadressen speichert und gemeinhin als Stack bezeichnet wird.
Dieses Problem wurde hier aber auf eine listige Art umgangen: nämlich mit der sonst mit Recht verpönten Technik, den Adressenbus nicht vollständig zu decodieren und dadurch Speicherplätze scheinbar an mehreren Adressen gleichzeitig erscheinen zu lassen. Und so erscheinen unsere 128 Byte RAM nicht nur an den Zero-Page-Adressen 0000...007F, sondern – mit dem gleichen Speicherinhalt – bei 0180...01FF, also im Stack-Bereich.
Dabei muß man nur bedenken, daß das Schreiben z. B. an die Adresse 01FE den Inhalt bei 007E gleichermaßen verändert. Man muß sich also beim Programmieren überlegen, wieviel Platz man für Unterprogrammsprünge in Stack und wieviele Bytes man in der Zero Page benötigt. Die Verteilung der 128 Bytes RAM könnte dann typischerweise so aussehen, daß 01F0...01FF als Stack dient, um maximal sechs Unterprogrammebenen plus eine Interrupt-Ebene zuzulassen, und 0000...006F als frei verwendbarer Zero-Page-Bereich.
Der anderswo große Nachteil, daß eine Systemerweiterung wegen der unvollständigen Adressendecodierung schwierig ist, wurde hier im Interesse möglichst geringer Hardware-Kosten bewußt in-kauf genommen.
Die restliche Adressenbelegung entstand ebenfalls unter diesem Aspekt; es ist nur noch ein einziges TTL-IC nötig, um die Decodierung der Adressen vorzunehmen. Die genaue Zuordnung geht aus *Tabelle 1* hervor.
Die Eigenschaft der Adressenduplizierung kann u. U. auch einen Vorteil darstellen. Denn nicht immer steht in dem Tischcomputer, der zur Entwicklung des Programms Verwendung findet, derjenige Adressenbereich zur Verfügung, in dem der EPROM-Bereich unseres kleinen Systems eigentlich liegt. Möglicherweise besitzt der Tischcomputer aber einen Speicherbereich, der identisch mit einem duplizierten Bereich des EPROM ist. Eine Adressenanpassung ist dann nicht mehr nötig. Dies gilt selbstverständlich auch für die Zero-Page- und Stack-Bereiche.

Tabelle 1: Adressenbelegung des 6504-Computers

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

(A = gültiges Adressen-Bit, X = ignoriertes Adressenbit)
6532-Adressen: 800 = Port A, 801 = Port-A-Richtungsregister, 802 = Port B, 803 = Port-B-Richtungsregister; 814 = Timer 1 µs, 815 = Timer 8 µs, 816 = Timer 64 µs, 817 = Timer 1024 µs; 81C...81F wie 814...817, jedoch mit Interrupt bei abgelaufener Zeit. Timer auslesen: 816; Timer testen: 817 (N-Flag).

Die Inbetriebnahme des Systems

Nehmen wir an, wir hätten ein EPROM mit dem nötigen Betriebsprogramm für unseren individuellen Verwendungszweck programmiert. Dann können wir alle Bauelemente auf die doppelseitige durchkontaktierte Epoxy-Platine löten (Bilder 2 bis 4; beziehbar u. a. bei Fa. Walter, Am Starzenbach 9, 8069 Wolnzach), wobei es sich dringend empfiehlt, für die drei LSI-ICs 6504, 6532 und 2758 Fassungen und eine 31polige Steckerleiste (*Tabelle 2*) zu verwenden. Einen Bausatz liefert die Firma Elektronikladen, Wilhelm-Mellies-Str. 88, 4930 Detmold 1.
Beim Anschalten der 5-V-Versorgungsspannung (Netzteil-Belastbarkeit min. 200 mA) erfolgt über das auf der Platine befindliche Monoflop automatisch ein Reset, so daß der Prozessor mit dem Abarbeiten des Programms beginnt, dessen Startadresse in den Zellen FFFC (niederwertiges Byte) und FFFD (höherwertiges Byte) abgelegt ist. Diese Adressen gibt es in unserem System natürlich nicht wirklich; sie finden sich aber dupliziert am oberen Ende des EPROM-Bereichs bei 0FFC und 0FFD.
Im nächsten Heft werden wir ausführlich auf die Programmierung von Anwenderprogrammen für den 6504-Mikrocomputer eingehen und häufig benötigte Routinen für Tastaturabfrage, Display-Ansteuerung und Timer-Verwendung vorstellen.

Tabelle 2: Steckerbelegung

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

Literatur

[1] R 6532 Data Sheet.
Rockwell Doc. Nr. 29 000 D42.
[2] R 650X Data Sheet.
Rockwell Doc. Nr. 29 000 D39.
[3] R 6500/6532 Timer Interrupt Precautions.
Rockwell Doc. Nr. R 6500 N02.
[4] EMUF-Programmiertips. mc 1981, Heft 2.
[5] Bits und Bytes: 6502-Programmierung.
Sonderheft „Hobbycomputer 2", Franzis-Verlag.

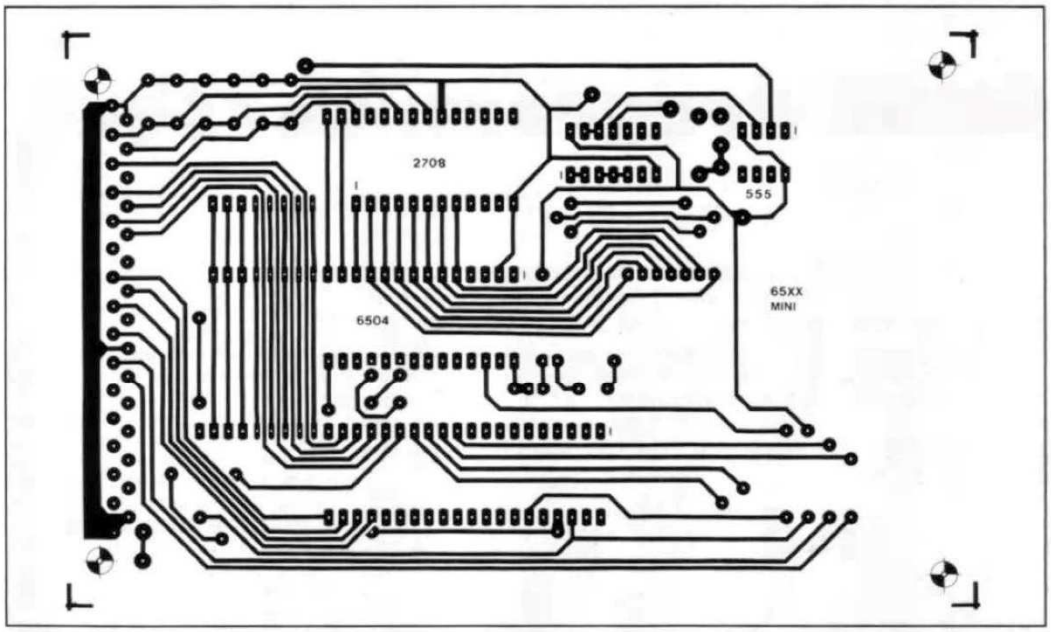

Bild 3. Bestückungsseitige Leiterbahnen der (doppelseitigen, durchkontaktierten) Platine. Die 31polige Steckerleiste ist später auf diese Seite zu löten

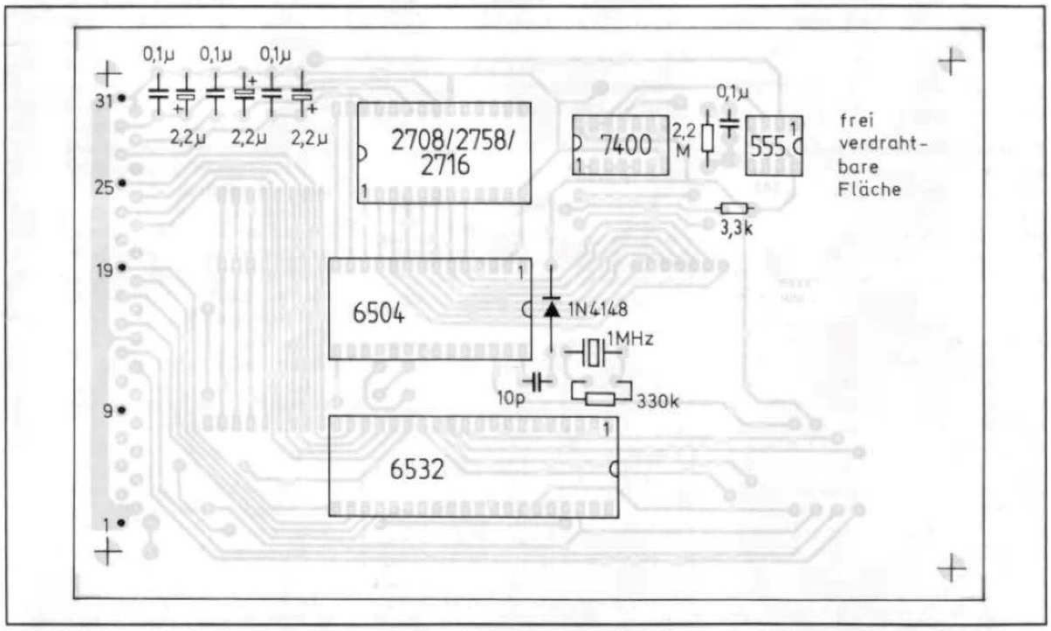

Bild 4. Bestückungsplan des 6504-Computers. Es sei erwähnt, daß das 2-KByte-EPROM 2716 z. T. schon preiswerter angeboten wird als der 5-V-/1-KByte-Typ 2758. Beim 2716 kann man entweder eine Hälfte „verschenken" oder auch mit einem Schalter zwischen zwei 1-KByte-Betriebsprogrammen wählen

Herwig Feichtinger

EMUF-Programmiertips

Wer ist „EMUF"? Nun, der in diesem Heft beschriebene Einplatinen-Mikrocomputer wurde unter dieser redaktionsinternen Bezeichnung ursprünglich für einen ganz anderen Verwendungszweck entwickelt, nämlich zur Steuerung eines Kurzwellenempfängers, eines „**E**mpfängers **m**it **u**nzulässigen **F**requenzbereichen". Inzwischen ist es allerdings besser, EMUF als „**E**inplatinen-**M**ikrocomputer für **u**niverselle **F**estprogramm-Anwendung" zu interpretieren. Hier nun einige Tips, wie man Programme für den EMUF entwickelt.

Der Einplatinen-Mikrocomputer „EMUF" arbeitet mit der CPU 6504, die exakt über den gleichen Befehlssatz wie ihre größere Schwester 6502 verfügt. Dies ermöglicht die Programmentwicklung auf praktisch allen 6502-Tischcomputern wie PET, CBM, PC-100, Apple II usw.

Erster Schritt: I/O-Festlegung

Bevor das EMUF-Maschinenprogramm geschrieben wird, muß man sich zunächst über die Hardware-Voraussetzungen klar werden. Üblicherweise werden an den EMUF verschiedene Ein- und Ausgänge, Schalter, Leuchtdioden-Treibertransistoren usw. angeschlossen. Dafür stehen zwei Ports (PA und PB) mit je acht Leitungen zur Verfügung. Für jeden Port enthält der Baustein 6532 ein Datenrichtungsregister (PAD bzw. PBD), das es erlaubt, einzelne Leitungen (also Bits) eines Ports wahlweise als Ein- oder Ausgang zu deklarieren (0 = Eingang, 1 = Ausgang). Nach einem Reset und damit nach dem Einschalten des EMUF sind alle Ports zunächst als Eingang geschaltet.
Nicht beschaltete Eingänge liegen (wie offene TTL-Gatter-Eingänge) auf log. 1; legt man an eine Portleitung also einen Schalter nach Masse, so ist ein Pull-Up-Widerstand nicht unbedingt nötig, wenn auch empfehlenswert, um die Empfindlichkeit gegenüber Störeinflüssen zu reduzieren. *Bild 1* zeigt, wie man an drei Leitungen des Ports PA einen Schalter und zwei Leuchtdioden wahlweise über einen Transistor oder einen TTL-Inverter (ev. mit offenem Kollektor, also 7404 oder 7406) als Treiber anschließt, nachdem der Port selbst nicht genügend Strom für eine LED zur Verfügung stellen kann.
Sobald jedem Ein- und Ausgang der Anwenderschaltung eine Portleitung zugeordnet ist, kann man sich der Software-Erstellung zuwenden.

Zweiter Schritt: Programmtest mit Tischrechner

Für die Bewältigung der gestellten Aufgabe sollte man das Programm nicht aufs Geratewohl schreiben und sofort in ein EPROM für den EMUF schießen. Vielmehr ist es sinnvoll, es zunächst auf dem Tischrechner zu testen, der auch der Programmentwicklung dient, und eventuell zu korrigieren.
Dies stellt an den Tischrechner aber bestimmte Anforderungen. Er sollte möglichst über die gleichen Ein- und Ausgabemöglichkeiten verfügen, die man später beim EMUF benutzen möchte, und auch einen Timer gleicher Struktur wie im 6532 des EMUF besitzen. Die Computer AIM-65 (Rockwell) und PC-100 (Siemens) sind hier ideal geeignet, da auch sie zwei User-Ports und einen 6532-Timer enthalten. Ferner ist bei ihnen der gesamte benutzbare Adressenbereich des EMUF im Arbeitsspeicher als RAM vorhanden, so daß z. B. das später im EPROM stehende EMUF-Programm sofort im endgültigen Adressenbereich 0C00...0FFF entwickelt werden kann.
An Änderungen vor dem Programmieren eines EPROM ist dann lediglich noch die Anpassung der Timer- und Portadressen nötig.
Kommt man mit nur einem Port aus und verzichtet auch auf die Möglichkeit, den 6532-Timer noch im Entwicklungssystem zu testen, ist das Erstellen der Software auch mit Rechnern wie PET und CBM möglich. Hierbei kann es auch nützlich sein, den 6532-Timer während der Entwicklungsphase durch den VIA-Timer im 6522 des CBM zu simulieren, was aber eine größere Softwareänderung erforderlich macht.

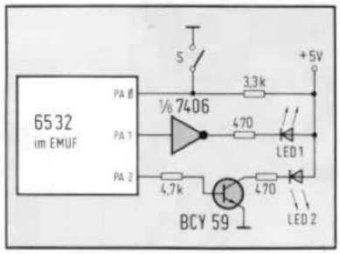

Bild 1. Will man einen Schalter an eine I/O-Leitung anschließen, empfiehlt sich ein Pull-Up-Widerstand (hier 3,3 kΩ). Leuchtdioden können über eine Transistor- oder TTL-Treiberschaltung angesteuert werden

Gleich sind wir fertig...

Hat man das Maschinenprogramm auf dem Entwicklungssystem (AIM, PC-100, CBM usw.) zum Laufen gebracht, so muß man vor dem Programmieren eines EPROM noch beachten, daß manche Dinge, die sonst das Monitorprogramm des Systems erledigt, für den EMUF explizit programmiert werden müssen.
Dies gilt speziell für die Befehlsfolge, die nach einem Reset nötig ist, um die CPU in einen definierten Zustand zu bringen. Normalerweise sieht der Programmanfang deshalb etwa so aus wie in *Bild 2*: Der Stackpointer wird auf den Anfangs-

wert FF gesetzt, das Dezimal-Flag wird gelöscht (dies ist normalerweise sinnvoll, um die hexadezimale Arbeitsweise bei ADC- und SBC-Befehlen sicherzustellen). Dann kann man die Port-Datenrichtungsregister initialisieren, um bestimmte Portleitungen als Ein- bzw. Ausgang zu deklarieren.
Wenn schließlich noch alle Port- und Timer-Adressen an den EMUF angepaßt sind und das Programm auch im EPROM-Bereich ab 0C00 lauffähig ist, kann man ein EPROM programmieren und auf die EMUF-Platine stecken.

Sehr nützlich: ein Assembler

Das Umschreiben des mühsam entwickelten Programms vom Entwicklungssystem auf die veränderte Adressenlage des EMUF von Hand wäre unvertretbar zeitraubend und fehlerträchtig, speziell bei Programmen von mehr als einem halben KByte. Es ist deshalb sinnvoll, für die Programmerstellung einen symbolischen Assembler zu verwenden, dem man die Adressenlage nur einmal am Anfang des Quellentext-Programms mitzuteilen braucht und der bei der Umsetzung in den Objektcode, wie er ins EPROM muß, alle nötigen Anpassungen selbständig vornimmt. Im Quellentext selbst steht nämlich z. B. für den Port A nicht eine bestimmte hexadezimale Adresse, sondern nur der symbolische Name PA, und überall, wo der Assembler auf PA trifft, setzt er dafür jene Adresse ein, die der Programmierer am Anfang dafür einmal definiert hat.
Für solche Assembler-Listings werden Sie in mc noch mehrfach typische Beispiele finden.

Zwei Programme im EPROM

Da der EMUF immer nur ein Kilobyte im EPROM-Bereich 0C00..0FFF adressiert, was genau dem Speicherplatz eines EPROM 2708 bzw. 2758 entspricht und für die meisten Anwendungsfälle auch vollkommen ausreicht, ergibt sich die Möglichkeit, bei Verwendung eines 2-KByte-EPROM (2716) an dessen höchstwertige (hier unbeschaltete) Adressenleitung einen Schalter zu legen.
Damit ist es möglich, im „unteren" und „oberen" Kilobyte zwei voneinander unabhängige Programme von je maximal 1 KByte Länge unterzubringen und mit einem Schalter auszuwählen, welche Aufgabe der EMUF gerade zu erfüllen hat. Nach der Schalterbetätigung ist natürlich stets ein Reset (Betriebsspannungsunterbrechung oder Reset-Taste) notwendig.
Die Programmentwicklung hat für beide EPROM-Hälften so zu geschehen, daß die Adressenbelegung stets dem Bereich 0C00...0FFF oder einem seiner Speicherbereichs-Duplikate (z. B. 6C00...6FFF) entspricht. Wenn im Entwicklungssystem genügend Speicherplatz zur Verfügung steht, kann man deshalb das „untere" Programm von 0C00...0FFF und das „obere" von 1C00...1FFF entwickeln, ohne daß im EMUF später Schwierigkeiten auftreten.
Wie man das Programm dann mit AIM-65 oder PC-100 in ein EPROM schießt, ist sehr ausführlich in dem von Siemens erhältlichen „Applikationsbuch PC-100" beschrieben; an Hardware ist dafür nur eine Diode und eine EPROM-Fassung erforderlich.

Bild 2. Typisches Assemblerlisting zur Initialisierung des EMUF nach einem Reset. Die CPU holt sich die Reset-Adresse aus 0FFC und 0FFD. Dann wird der Stackpointer auf FF gesetzt, die Port-Datenrichtungsregister werden mit dem jeweiligen Bitmuster geladen, und das Dezimal-Flag wird gelöscht. Das Listing entstand mit dem 2-Pass-Assembler von AIM-65 und PC-100

```
0C0E          PASS 2
0000
0000          ;EMUF-INITIALISIERUNG
0000 PA             =$800
0000 PAD            =$801
0000 PB             =$802
0000 PBD            =$803
0000                *=$FFC        ;RESET
0FFC          0000  .WOR RESET    ;VECTOR
0FFE                *=$C00
0C00 RESET    A2FF  LDX #$FF      ;STACK-
0C02          9A    TXS           ;POINTER
0C03          A9F8  LDA #%11111000
0C05          8D0180 STA PAD      ;PORT-
0C08          A93E  LDA #%00111110
0C0A          8D0380 STA PBD      ;RICHTUNG
0C0D          D8    CLD
0C0E          ;WEITER IM PROGRAMM
0C0E                .END
```

Spruch des Monats

Die Mathematiker, die nichts als Mathematiker sind, haben einen klaren Verstand, vorausgesetzt, daß man ihnen alles durch Definitionen und Prinzipien erklärt, sonst sind sie wirr und unerträglich, denn sie denken nur richtig an Hand deutlich gemachter Prinzipien.
Und die Feinsinnigen, die nichts als feinsinnig sind, sind unfähig, die Geduld aufzubringen, bis zu den ersten Prinzipien der Spekulation und Abstraktion vorzudringen, denen sie in der Welt niemals begegnet sind und die man dort nie braucht.

Blaise Pascal