Ronald Zech

Forth auf einem Einchip-Mikro

Konzept für kleinere Anwendungen

Die Programmiersprache Forth hat sich, parallel zu ihrer starken Verbreitung in den letzten Jahren, ihren Weg von den großen Computern im Prozeß- und Automatisierungsbereich auch in die Welt der Mikrocomputer gebahnt. Wie sich die Sprache nun auch im Bereich der Einchip-Mikrocomputer und Minimalsysteme behauptet, zeigt der vorliegende Artikel.

Bislang war man bei der Entwicklung von Applikationen für solche kleinen Computer im allgemeinen auf Assemblerprogrammierung und auf große Entwicklungssysteme angewiesen, zwei Umstände, die ins Geld gingen. Gelegentliche Ansätze von µP-Herstellern, etwa mit Hilfe von On-Chip-Basic-Interpretern hier Abhilfe zu schaffen, waren wegen diverser Nachteile wie z. B. sehr niedriger Arbeitsgeschwindigkeit zwar hilfreich, aber doch nicht wirklich überzeugend.

Recht bald kam daher auch die neuere Sprache Forth ins Spiel. Als Operationssystem, als Interpreter, Monitor und als Compiler gleichermaßen geeignet, ist diese vielseitige und zudem erweiterbare Sprache dennoch außerordentlich kompakt. Zudem reichen die Speichergrößen typischer Einchip-Mikrocomputer bereits heute für die Unterbringung leistungsfähiger Runtime-Packages aus. Unter mehreren Firmen, die in den letzten Jahren solche Einchip-Mikrocomputer mit Forth-Kern angekündigt haben [1], ist Rockwell nun als erste mit zwei konkret verfügbaren Produkten auf dem Markt erschienen [3, 4, 5].

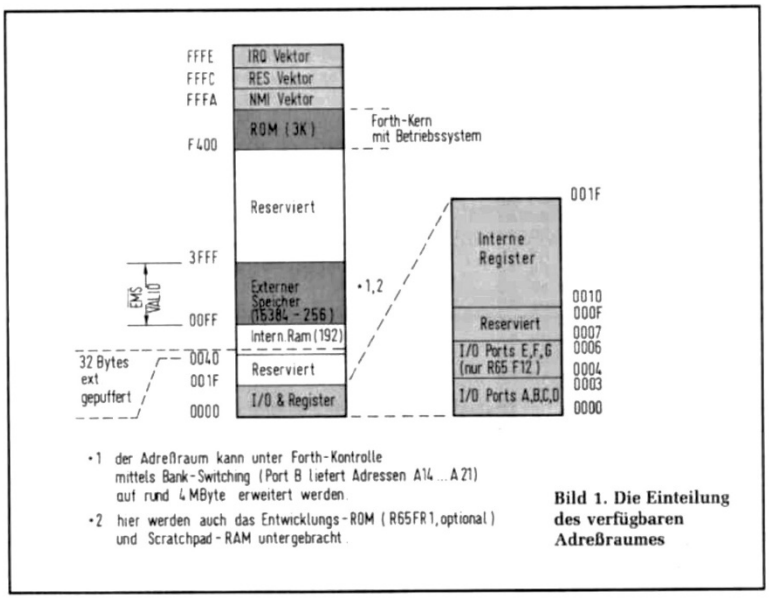


Bild 1. Die Einteilung des verfügbaren Adreßraumes

Ein kompletter Computer auf einem Chip

Der Forth-Mikrocomputer R65F11 von Rockwell ist ein Abkömmling der bekannten 6502-Familie. Der in NMOS-Technologie gefertigte Chip besteht aus den folgenden Einheiten:

- ☐ Eine konventionelle 6502-CPU mit vier zusätzlichen Bit-Manipulationsbefehlen.
- ☐ 192 Bytes RAM, davon 32 mit getrennter Spannungsversorgung, so daß mit einer Akkupufferung wichtige Daten auch bei Netzausfall oder Abschaltung erhalten bleiben.
- ☐ Zwei Parallelports von 8 Bit Breite, jedes Bit ist einzeln als Ein- oder Ausgang konfigurierbar. Einige Leitungen können Sonderfunktionen übernehmen.
- ☐ Eine serielle Schnittstelle, die in vielfältiger Weise konfigurierbar ist und insbesondere über Netzwerkfähigkeit verfügt.
- ☐ Zwei programmierbare Zähler/Timer, von denen einer ggf. die (interne) Baudrate für die serielle Schittstelle erzeugt.
- ☐ Ein interner Clock-Oszillator, der die für Entwicklungsumgebungen oder auch Master/Slave-Betrieb wichtige Option besitzt, auch ein externes Clock-Signal zu akzeptieren.
- ☐ Ein Interrupt-System, das die von den acht auf dem Chip untergebrachten Peripherieeinheiten herührenden Interruptquellen verwaltet. Vier der Parallel-Port-Leitungen können flankengetriggert mit Interruptereignissen verknüpft werden. Ebenso können die Timer Interrupts auslösen, was natürlich auch für die Funktionen des seriellen Ports gilt. Jede der verschiedenen Interruptquellen ist einzeln ein- und ausschaltbar. Daneben existiert auch ein Eingang für einen nicht maskierbaren Interrupt (NMI).
- ☐ 3 KByte ROM, das einen Forth-Kern mit über 120 Funktionen enthält (genauer: deren Runtime-Prozeduren nebst vollständigem Betriebssystem).

Die Gliederung des Adreßraumes zeigt *Bild 1.*

Der Zweite im Bunde ist der R65F12. Während der R65F11 noch mit einem 40poligen DIL-Gehäuse auskommt, ist der R65F12 in einem 64poligen Gehäuse mit recht kompakter Bauform untergebracht. Dafür bietet er bei ansonsten gleichen Funktionen drei weitere 8-Bit-Parallel-Ports. Beide Prozessoren nützen aus Platzgründen einige Leitungen doppelt.

Eine vollständige Entwicklungsumgebung

Der R65F11 wird in den Programm-Entwicklungsphasen durch ein externes ROM von 8 KByte Größe unterstützt (Develop ROM R65FR1, für spezielle Anforderungen existieren Varianten). Jedoch erlaubt bereits der Kern für sich allein interaktives Arbeiten in beschränktem Umfang, was zumindest den Service am fertigen Produkt erleichtern kann. Aus Platzgründen enthält der Forth-Kern im Prozessor aber nur die für fertig kompilierte Programme nötigen Funktionsteile. Diese liegen in Form von sogenanntem headerlosen Code vor. Es handelt sich dabei im wesentlichen um eine Sammlung applikationsnaher FIG-Forth-Befehle, allerdings mit erheblichen maschinenspezifischen Erweiterungen. So unterstützt etwa BANKEXECUTE Floppy-Disk-gestützte Overlay- oder auch Bank-Select-Techniken (Adreßraumerweiterung mit Hilfe von Port B auf max. 4 MByte).
Interpreter, Compiler, Assembler und Hilfsfunktionen sind hingegen aus Platzgründen in das Entwicklungs-ROM ausquartiert worden. Auch die komplette Dictionary-Struktur ist aus diesem Grund hier untergebracht worden.

Ergänzt man den R65F11-Mikrocomputer um dieses Entwicklungs-ROM sowie ein externes Scratchpad-RAM, so ist die Entwicklungsumgebung interaktiv arbeitsfähig und bereit, über die serielle Schnittstelle des Prozessors mit einem Datenterminal zu kommunizieren.
Einen typischen Aufbau hierzu zeigt *Bild 2.*

Für ein Zwei-Chip-Produktionssystem kann das Entwicklungs-ROM wieder entfernt und das ins Scratchpad-RAM kompilierte Applikationsprogramm in ein EPROM übertragen werden. Die Programmierung des EPROMs wird vom System direkt unterstützt.

Der Entwickler bzw. Anwender findet auch softwareseitig alles vor, um einfach und problemlos mit Floppy-Disks arbeiten zu können. Ergänzt man die Schal-

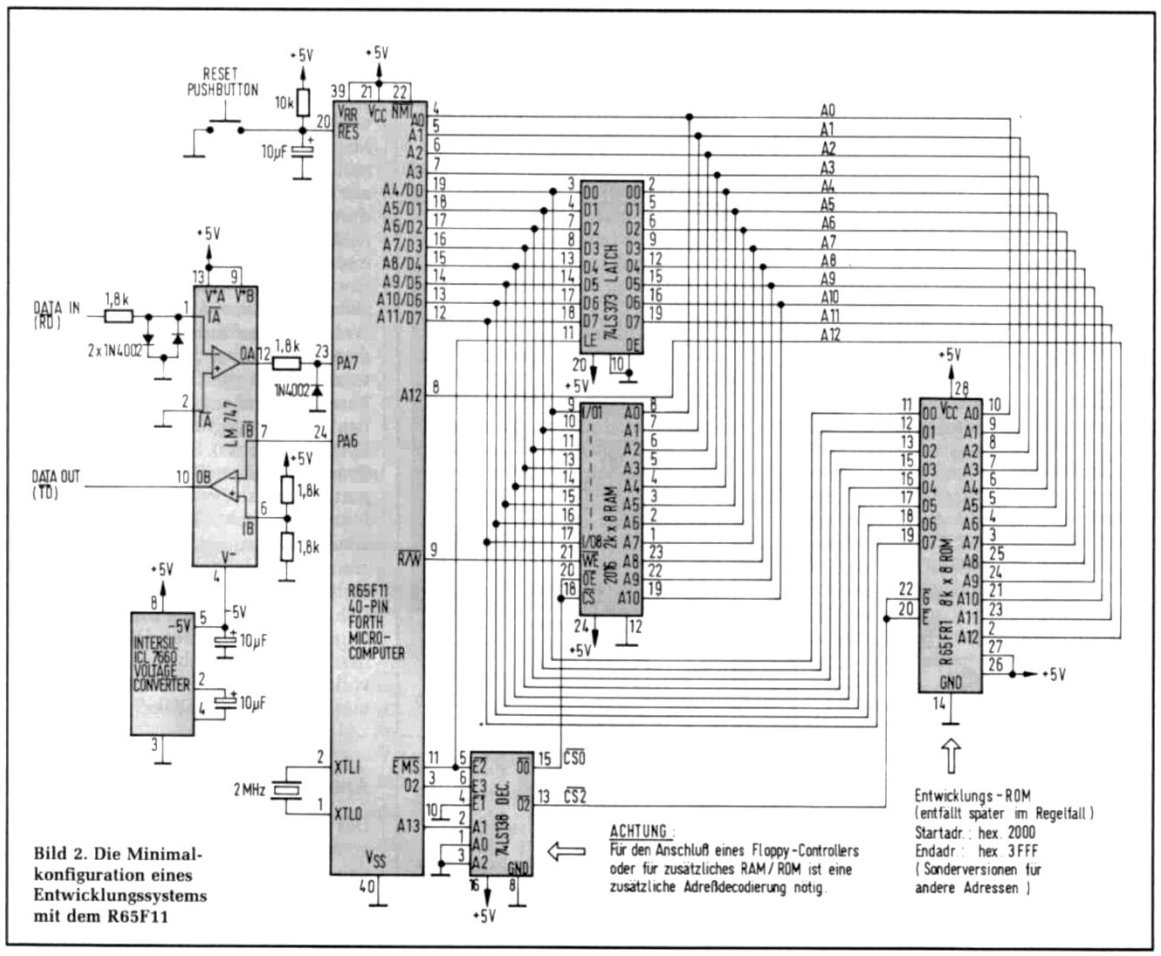


Bild 2. Die Minimalkonfiguration eines Entwicklungssystems mit dem R65F11

tung von Bild 2 um die Controller-Schaltung von *Bild 3*, so ist mit denkbar geringstem Aufwand ein vollständiger Rechner entstanden. Der Mikrocomputer R65F11 erwartet, falls man ihm nichts anderes sagt, einen Floppy-Disk-Controller vom Typ WD-1793 oder WD-2793 bei Adresse hex 100 und stellt dann die nötige Treibersoftware und Kommandoebene bereit. Es können bis zu vier 5¼-Zoll-Floppy-Laufwerke angeschlossen und somit bis zu 2,4 MBytes Speicher verwaltet werden, ohne daß der Benutzer eine einzige Zeile eigenen Code zu schreiben hätte. Wer mag, kann daher den gesamten Entwicklungszyklus mit seinem Applikationsaufbau durchführen.

Das Operations-System

Sinn und Zweck des On-Chip-Betriebssystemes ist es, die Funktion des Mikrocomputers nach dem Einschalten des Gerätes bzw. nach einem Power-On-Reset in geordneter Weise einzuleiten, Nutzprogramme zu laden und zu starten, alle System-Ressourcen verfügbar zu machen und Werkzeuge zu ihrer Handhabung bereitzustellen. Diese und viele andere Funktionen, insbesondere natürlich auch Entwicklungsunterstützung, werden vom Forth-System übernommen. Nach einem Reset wird über den Reset-Vektor (Bild 1) die Forth-Kaltstartprozedur COLD aufgerufen. Hier wird der Microcomputer und insbesondere die auf ihm softwaremäßig installierte virtuelle Forth-Maschine in Gang gesetzt. Zum Beispiel werden die Interruptquellen gesperrt und die serielle Schnittstelle wird auf asynchronen Betrieb gesetzt. Auch das Disk-System wird mit Dafault-Werten initialisiert. Die Kernfunktionen laufen nun. Dann interessiert sich das System für seinen Aufbau und die nähere Vergangenheit, und hier wird es nun richtig spannend.

Welcher Start?

Eine bestimmte Variable wird daraufhin abgefragt, ob ein Kaltstart (= totale Neuinitialisierung) oder ein Warmstart erfolgen soll: enthält sie nicht einen ganz bestimmten Wert (Autostart-Pattern hex A55A), so wird Kaltstart angenommen. Alle Interrupts im System sind (ebenso wie das gesamte I/O-System) vektorisiert, so daß sie vorsichtshalber erst einmal auf Reset zeigen können. Die I/O-System-Vektoren zeigen auf die Standard-I/O-Prozeduren, und alle System-Variablen werden dann ebenfalls auf Anfangswerte gesetzt.

Im Warmstart-Falle werden dagegen lediglich der Terminal-Input-Buffer bzw. die zugeordnete User-Variable TIB sowie zwei Variable geladen, in denen die Werte für die beiden System-Stacks gehalten werden (RO und SO).

In diesem Stadium beginnt für das Betriebssystem nun eine Phase des Suchens nach einem ausführbaren Programm. Hierzu wird zunächst einmal der Adreßraum von hex 0400...3F00 in 1-KByte-Schritten auf Vorhandensein der Autostart-Markierung abgesucht. Wird das Muster gefunden, so wird auf die direkt im Anschluß an dieses Speichermuster erwartete Ausführungsadresse zugegriffen und das zugehörige Programm damit zur Ausführung gebracht. *Bild 4* zeigt das Schema.

Die genannte Adresse muß im übrigen die Parameterfeld-Adresse PFA eines High-Level-Forth-Wortes (Programmes) sein. Diese etwas ungewöhnliche Rockwell-Konvention führt, falls man ein Maschinensprache-Programm aufrufen möchte, zu der an sich unnötig komplizierten Pointer-Struktur in Bild 4 b. Ein durch und durch gutes Produkt darf aber natürlich in einer Kleinigkeit auch einmal ein wenig „holprig" sein.

Hat das System ein Programm auf diese Weise nicht gefunden, so gibt es sich dennoch nicht geschlagen. Nach Aussenden des Textes „No ROM" an das Terminal, wird auf das Floppy-Disk-Laufwerk 0 zugegriffen. Die ersten 128 Bytes von Track 0, Sektor 1, werden in einen bestimmten RAM-Bereich transportiert und als Boot-Programm zur Ausführung gebracht. Dieses Programm kann ein beliebiges Maschinen-Programm sein, das den genannten Maximalumfang beachtet. Es ist somit möglich, ganze Programme von Diskette zu holen und auszuführen. Dies schließt einen Restart von Forth oder auch eine völlige Umkonfiguration des I/O-Systemes und sogar des Disk-Systemes ein.

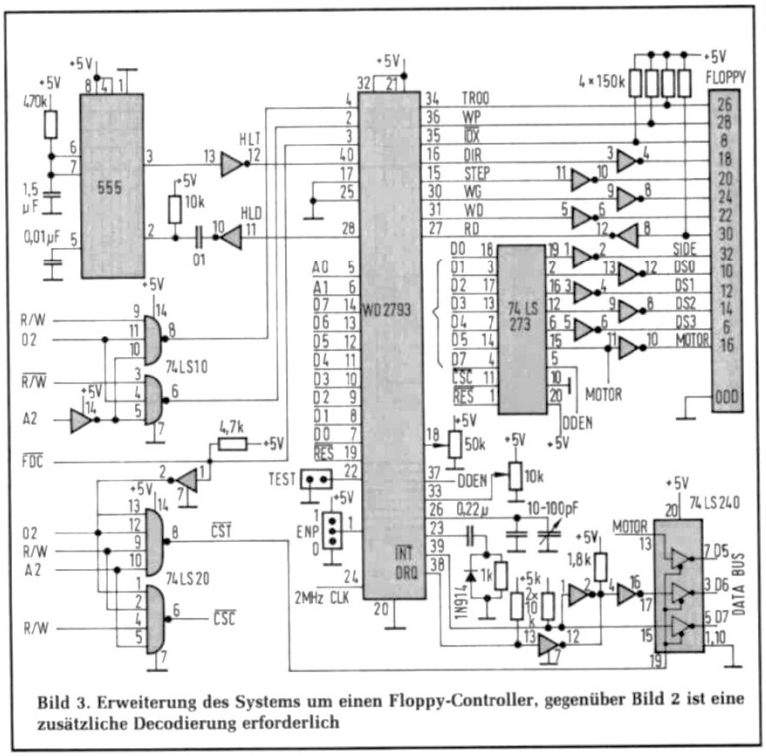


Bild 3. Erweiterung des Systems um einen Floppy-Controller, gegenüber Bild 2 ist eine zusätzliche Decodierung erforderlich

Autostart des Entwicklungs-ROMs

Das Entwicklungs-ROM muß natürlich, wenn und solange es im System vorhanden ist, über die Power-On-Sequenz auch wirklich erreichbar sein. Es ist daher ein gutes Beispiel für die geschilderte Autostart-Prozedur. Falls die Auto-

start-Markierung nicht bereits vorher (in einem niedrigeren Adreßblock) gefunden wurde, so wird sich das Forth-Entwicklungssystem über die serielle Schnittstelle beim Terminal zur Stelle melden mit der Message „RSC-FORTH V 1.5" und hiermit seine Dienste offerieren.

Der Mikro-Monitor

Bevor das System, wie oben schon geschildert, auf der Suche nach einem auszuführenden Programm evtl. auch der Diskette zu nahe treten darf, bietet es noch eine Alternative hierzu an. Nachdem die bewußte Meldung „No ROM" an das Terminal abgesetzt ist, wird der (serielle) Empfangsport auf Vorliegen des ASCII-Zeichens CTRL R (hex 12) abgefragt. Dieses Zeichen ist als Zufallsmuster in der Einschaltphase nicht zu erwarten. Es wird daher als Aufforderung interpretiert, den im On-Chip-Forth-Kern untergebrachten Mikromonitor aufzurufen, falls zusätzlich Reset gedrückt ist.

Vorbereiten eines Forth-Programmes für Autostart

Wie erreicht man nun, daß der R65F11-Computer nach dem Einschalten stets ein bestimmtes Programm selbständig zur Ausführung bringt? Vorausgesetzt, daß das Entwicklungs-ROM sich (noch) im System befindet, können wir uns für den Aufruf einmal ein simples Programm vorstellen, etwa das folgende, welches einfach Zeile für Zeile einen kleinen Text endlos auf den Terminal-Bildschirm bringt:

```
: PROGRAMM BEGIN CR. „Dies ist ein
Programm" AGAIN ;
```

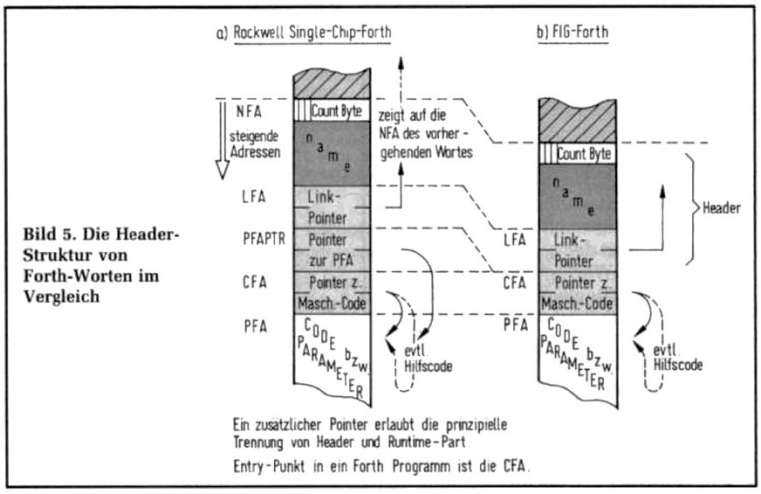


Bild 5. Die Header-Struktur von Forth-Worten im Vergleich

Um dieses Prgrämmchen mit dem sinnigen Namen „PROGRAMM" zum von nun an einzigen Lebenszweck des Computers zu deklarieren, sagen wir dem Rechner nun schlicht und einfach

```
HEX 400 AUTOSTART PROGRAMM
```

Und das war es dann auch schon. Nach jedem Reset wird der Rechner nun diese Endlos-Schleife zur Ausführung bringen und nicht aufhören zu behaupten, daß dies ein Programm sei. Hex 400 wurde natürlich willkürlich gewählt.

Der AUTOSTART-Befehl ist eine von vielen Forth-Erweiterungen, die das Entwicklungs-ROM als Service-Funktionen bereithält. Typisch hierfür sind auch Befehle wie H/C (für die Erzeugung von sog. „headerlosem Code", s. u.) oder EEC! (zum Programmieren von EPROMs).

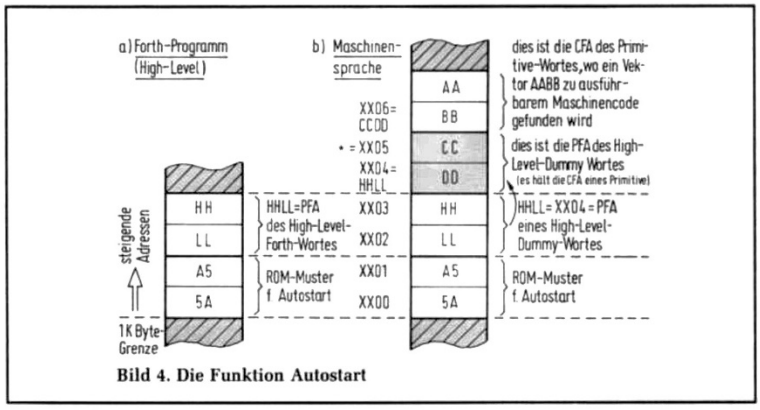


Bild 4. Die Funktion Autostart

EPROMs im Zielrechner programmieren

Für Industrie-Standards wie das EPROM 2764 bzw. EEPROMs Rockwell 5213/2816 bietet das Forth-Entwicklungs-ROM den Befehl „EEC!". Dieses Wort erwartet drei Parameter:
EEC! (Byte Adresse Count → %)
und programmiert dann ein Byte in die gewünschte (EPROM-)Adresse. Hierzu wird der Adreß-/Daten-Bus für eine Anzahl Taktzyklen konstant gehalten. Dabei ist die erforderliche Zeit natürlich eine Bauteile-Eigenschaft. So wird man für den 2816 (bei 1MHz Takt) Count = dez. 10 000 = hex 2710 wählen, um die erforderlichen 10 ms zu gewährleisten. Baut man diesen Befehl in eine simple Schleife ein, so kann man bequem ganze Speicherbereiche programmieren. Wir nehmen einmal an, daß das Prgrämmchen „PROGRAMM" in ein RAM mit Startadresse hex 4000 compiliert wurde, wo ja auch das Autostart-Muster hingesetzt wurde. Um dies z. B. in ein in Adresse hex 800 liegendes EEPROM 2816 zu burnen, würde man ganz einfach von Hand ein Hilfsprogramm eingeben und aufrufen:

```
: burn 800 400 DO I C@ I 400 +
2710 EEC! LOOP ;
```

Das EPROM ist danach fertig.

Die Wort-Struktur für headerlosen Forth-Code

Das soeben ins EPROM gebrannte Prgrämmchen trägt den Namen „PROGRAMM". Dieser Name ist nun mit Be-

standteil des EPROMs geworden. Er beansprucht dort Platz, obwohl er in Form eines sog. Headers normalerweise nur in der (interaktiven) Programmerstellungs-Phase wichtig war, nicht jedoch während des Programmlaufes. Zuweilen möchte man diesen unnötigen Speicherbedarf vermeiden. Das Singlechip-Forth von Rockwell unterstützt daher die Generierung von sogenanntem headerlosen Code. *Bild 5* zeigt den allgemeinen Aufbau eines Forth-Wortes, und zwar den eines Rockwell-Singlechip-Forth-Wortes und eines FIG-Forth-Wortes zum Vergleich. Der Unterschied besteht darin, daß gegenüber FIG-Forth der Header um einen Pointer (PFAPTR) erweitert wurde. Dieser Pointer markiert diejenige Stelle im Adreßraum, wo das eigentliche Programm beginnt. Das ist die Parameterfeld-Adresse PFA, die im Regelfalle direkt im Anschluß an die Codefeld-Adresse CFA zu finden sein wird. Während dies in FIG-Forth jedoch in vielen Fällen zwingend ist, erlaubt das Singlechip-Forth die Auflösung dieses Schemas.

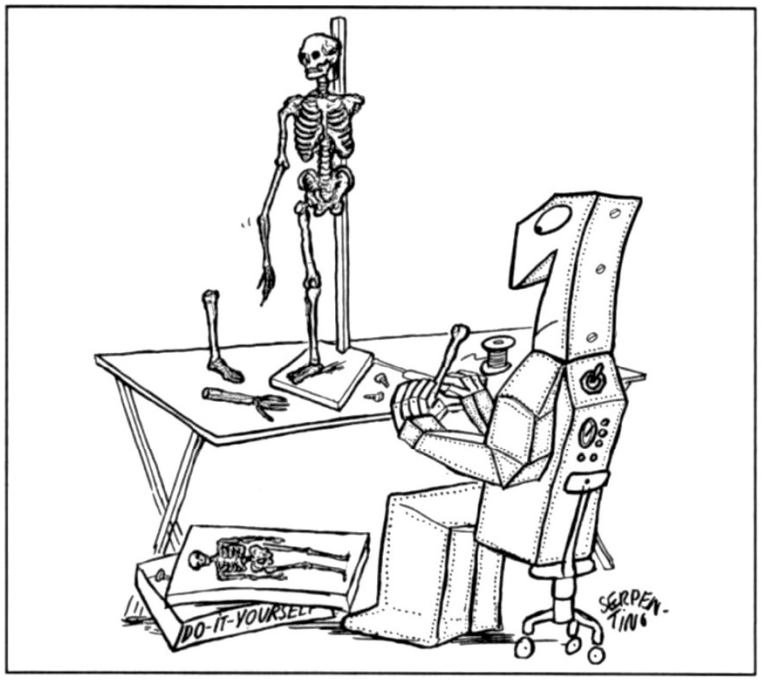
Zur Veranschaulichung kehren wir am besten noch einmal zu unserem Prögrämmchen „PROGRAMM" zurück. Wenn wir lediglich dessen Runtime-Code in unserem endgültigen EPROM haben wollen, so müssen wir dem armen Rechner dies ganz einfach sagen. Hierfür stellt er den Befehl „H/C" bereit. Zur Erinnerung: der Runtime-Code hatte ab hex 400 zu entstehen. Dies muß der Forth-Computer natürlich wissen. Zuvor aber verraten wir ihm noch, in welchem Hilfs-RAM wir ihm provisorisch seine Header aufzubauen gestatten. Wählen wir hierfür hex 1000, so können wir unser PROGRAMM nun neu und schöner machen mit den Anweisungen:

```
HEX 1000 H/C (Anweisung, headerlosen
      Code zu generieren und als Notiz-
      buch später entfallendes RAM in
      1000 zu verwenden)
: PROGRAMM ...; (den Rest kennen wir
ja schon von früher...)
400 AUTOSTART PROGRAMM
```

Das war es dann bereits und EPROMS programmieren können wir ja schon.

Die serielle Schnittstelle

Der R65F11 verfügt über eine vollduplex-fähige serielle Schnittstelle, für die in anderen Systemen ein eigener USART-Baustein nebst Timer-IC aufzuwenden wäre. Beim Kaltstart werden Sende- und Empfangskanal auf asynchrone Betriebsweise mit 16facher Taktrate des internen Timers A gesetzt, der auf resultierende 1200 Bd initialisiert ist. Die Übertragung erfolgt dann mit 1 Startbit, 7 Datenbits, 2 Stopbits und ohne Parity. Das Anwenderprogramm kann auch andere Werte setzen, wobei 12 verschiedene Baudraten von 50...9600 Bd, 5...8 Datenbits und Parity vereinbar sind. Sender und Empfänger wickeln natürlich ein sauberes Handshake (mit oder ohne Interruptverwaltung) ab und können getrennt ein- und ausgeschaltet werden.
Falls man als internen Takt den Systemtakt vereinbart, sind Übertragungsraten von 31,25 bzw. 62,5 kBd (bei 1 bzw. 2 MHz Systemtakt) erreichbar. Der Timer A wird dadurch frei. Unter Verwendung eines externen Taktes kann die Schnittstelle auch als synchrone Schnittstelle im sog. Schieberegister-Mode betrieben werden.



Wachen oder schlafen?

In Mikrocomputer-Netzwerken ist es üblich, eine Zieladresse an den Anfang einer jeden Nachricht zu setzen. Bereits mit sehr kleinem Overhead sind so recht ausgedehnte Netze möglich. Unser Chip ist natürlich auch hier schon präpariert. Das sog. „Wake-Up-Feature" erlaubt in einem solchen Netzwerk jedem der nicht angesprochenen Computer, das (u. U.) „dicke Ende" einer Nachricht zu ignorieren, bis irgendwann einmal eine neue Message beginnt. Kriterium hierfür ist der Empfang eines „leeren" Strings von zehn aufeinanderfolgenden Einsen. Genaugenommen kennzeichnet dies das Ende des zu ignorierenden Betriebes im Kommunikationsnetz: alle Rechner beachten nun das (irgendwann einmal sicher auftauchende) nächste ASCII-Zeichen und das Spiel beginnt von vorne. Eine vollständige Beschreibung dieses vielseitigen Bausteines würde den Rahmen eines Aufsatzes sprengen. Bank-Select- und Overlay-Techniken, das Interrupt-Handling oder Multitasking, seien lediglich erwähnt unter Hinweis auf die Original-Dokumentation des Herstellers.

Literatur

[1] *Waller, L.*: Forth interpreter on microcomputer eases programming. Electronics, Februar 1983.
[2] *Zech, R.*: Die Programmiersprache Forth. Franzis-Verlag, München.
[3] R65F11 and R65F12 forth based microcomputers. Rockwell Document No. 29651N49.
[4] A low cost development module for the R65F11 forth microcomputer. Rockwell Document No. 29651N65.
[5] R65FRX and R65FKX – RSC forth development and kernel ROMs. Rockwell Document No. 29651N80.