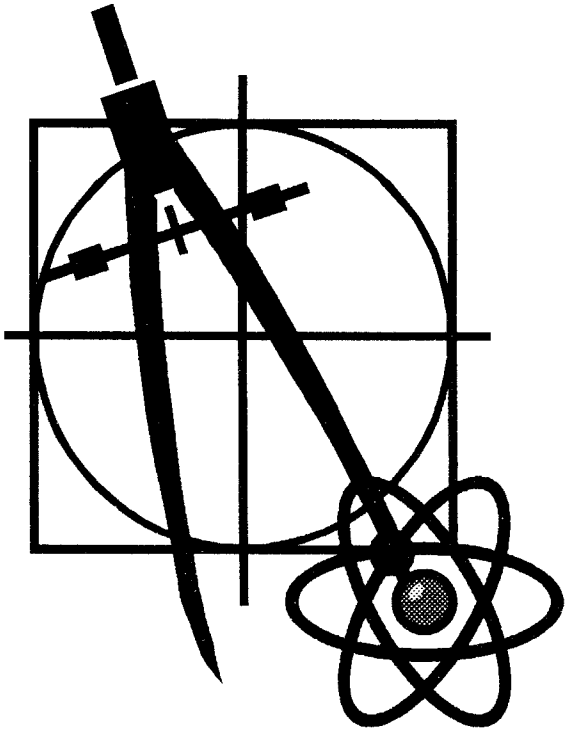
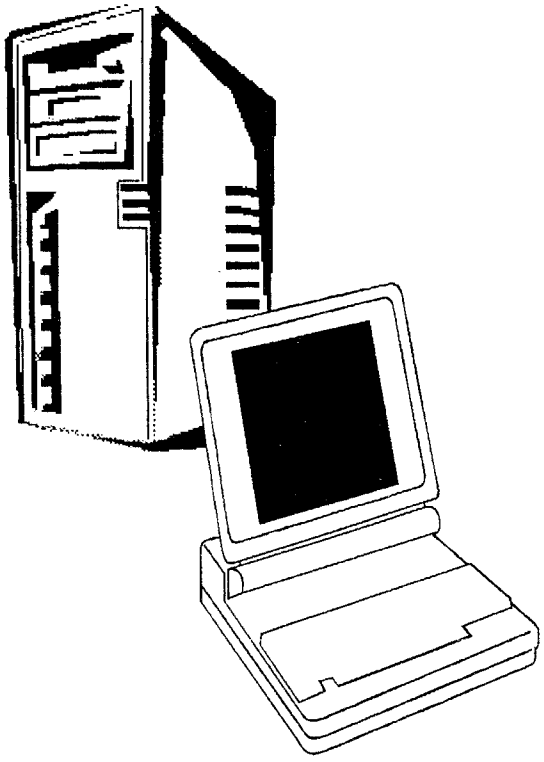
VIERTE DIMENSION

1 / 2002

Das FORTH-Magazin

für Wissenschaft und Technik, für kommerzielle EDV, für MSR-Technik, für den interessierten Hobbyisten.





In dieser Ausgabe

18. Jahrgang - **Ausg. 1/2002** – 10,-DM

Organ der FORTH-Gesellschaft e.V.

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

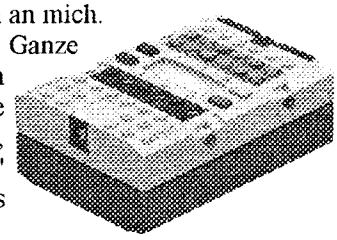
Feldstrasse 143
D-22880 Wedel
Fon 04103 - 808989 - 0
Fax 04103 - 808989 - 9
mail@tematik.de
www.tematik.de

Gegründet 1985 als Partnerinstitut der FH-Wedel beschäftigten wir uns in den letzten Jahren vorwiegend mit Industrieelektronik und Präzisionsmesstechnik und bauen z.Z. eine eigene Produktpalette auf.
Know-How Schwerpunkte liegen in den Bereichen Industriewaagen SWA & SWW, Differential-Dosierwaagen, DMS-Messverstärker, 68000 und 68HC11 Prozessoren, Sigma-Delta A/D. Wir programmieren in Pascal, C und Forth auf SwiftX68k und seit kurzem mit Holon11 und MPE IRTC für Atmel AVR.

LEGO RCX-Verleih

Seit unserm Gewinn (VD 1/2001 S.30) verfügt unsere Schule über so ausreichend viele RCX Komponenten, dass ich meine privat eingebrachten Dinge nun Anderen, vorzugsweise Mitgliedern der Forth Gesellschaft e.V. zur Verfügung stellen kann!
Angeboten wird: Ein komplettes LEGO-RCX-Set, so wie es für ca. 450 DM im Handel zu erwerben ist. Inhalt:

1 RCX, 1 Sendeturm, 2 Motoren, 4 Sensoren und ca. 1000 LEGO© Steine.
Anfragen über die Redaktion an mich.
Letztendlich enthält das Ganze auch nicht mehr als einen Mikrocontroller der Familie H8/300 Familie von Hitachi, ein 'paar' Treiber und 'etwas' Peripherie. Zudem: dieses Teil ist 'narrensicher'!



Martin Bitter

Forth Engineering
Dr. Wolf Wejgaard

Tel.: +41 41 377 3774 - Fax: +41 41 377 4774
Neuhöflirain 10
CH-6045 Meggen http://holonforth.com

Wir konzentrieren uns auf Forschung und Weiterentwicklung des Forth-Prinzips und offerieren HolonForth, ein interaktives Forth Cross-Entwicklungssystem mit ungewöhnlichen Eigenschaften. HolonForth ist erhältlich für 80x86, 68HC11 und 68300 Zielprozessoren.

Dipl.-Ing. Arndt Klingelnberg

Tel.: ++32 +87 -63 09 89 (Fax: -63 09 88)
Waldring 23, B-4730 Hauset, Belgien
akg@aachen.kbbs.org

Computergestützte Meßtechnik und Qualitätskontrolle, Fuzzy, Datalogger, Elektroakustik (HiFi), MusiCassette HighSpeedDuplicating, Tonband, (engl.) Dokumentationen und Bedienungsanleitungen

KIMA Echtzeitsysteme GmbH

Tel.: 02461/690-380
Fax: 02461/690-387 oder -100
Karl-Heinz-Beckurtz-Str. 13
52428 Jülich

Automatisierungstechnik: Fortgeschrittene Steuerungen für die Verfahrenstechnik, Schaltanlagenbau, Projektierung, Sensorik, Maschinenüberwachungen. Echtzeitrechnersysteme: für Werkzeug- und Sondermaschinen, Fuzzy Logic

FORTecH Software
Entwicklungsbüro Dr.-Ing. Egmont Woitzel

Budapester Straße 80A 18057 Rostock
Tel.: +49 (0381) 46139910 Fax: +49 (0381) 4583488

PC-basierte Forth-Entwicklungswerkzeuge, comFORTH für Windows und eingebettete und verteilte Systeme. Softwareentwicklung für Windows und Mikrocontroller mit Forth, C/C++, Delphi und Basic. Entwicklung von Gerätetreibern und Kommunikationssoftware für Windows 3.1, Windows95 und WindowsNT. Beratung zu Software-/Systementwurf. Mehr als 15 Jahre Erfahrung.

Ingenieurbüro
Dipl.-Ing. Wolfgang Allinger

Tel.: (+Fax) 0+212-66811
Brander Weg 6
D-42699 Solingen

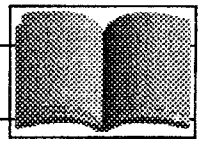
Entwicklung von µC, HW+SW, Embedded Controller, Echtzeitsysteme 1-60 Computer, Forth+Assembler PC / 8031 / 80C166 / RTX 2000 / Z80 ... für extreme Einsatzbedingungen in Walzwerken, KKW, Medizin, Verkehr / >20 Jahre Erfahrung.

Ingenieurbüro
Klaus Kohl

Tel.: 08233-30 524 Fax: —9971
Postfach 1173
D-86404 Mering

FORTH-Software (volksFORTH, KKFORTH und viele PD-Versionen). FORTH-Hardware (z.B. Super8) und -Literaturservice. Professionelle Entwicklung für Steuerungs- und Meßtechnik.

Inhaltsverzeichnis



Dieser Ausgabe liegt als Losblatt ein Anmeldungsformular zur Forth-Tagung 2002 in Garmisch Partenkirchen bei.

Editorial

IMPRESSUM

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth-Gesellschaft e.V.
Postfach 16 12 04
D-18025 Rostock
Tel.(Anrufbeantw.): 0381-400 78 28
Fax: 0381-400 78 28
E-Mail:
SECRETARY@FORTH-EV.DE
DIREKTORIUM@FORTH-EV.DE

Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto. 563 211 208

Redaktion & Layout (vorübergehend)

Martin Bitter
Möllenkampweg 1a
46499 Hamminkeln
Tel.: 02857-1419
E-Mail: VD@FORTH-EV.DE
martin.bitter@forth-ev.de

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluß 2001

März, Juni, September, Dezember
jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

DM 10,- zzgl. Porto u. Verp.

Fünf!

Fünf Ausgaben der Vierten Dimension – editiert von mir – liegen nun hinter Ihnen, liebe Leser. Fünf Ausgaben der Vierten Dimension liegen nun auch hinter mir!

Was hat es 'gebracht'?

- mir zu einem Teil den Drachenpreis – eine Überraschung, über die ich mich noch genauso freue wie am ersten Tag in Dingden.

- Ihnen und der Forthgesellschaft - 1¼ Jahr Weiterbestehen der Vierten Dimension, die ja letztendlich von uns allen für alle gestaltet wird.

Ich habe, wenn auch nicht von Angesicht zu Angesicht, einige aufregende und nette Menschen kennen gelernt, das reichte von Anfragen: „ Meinst Du das soll (so) in die Vierte Dimension?“ bis hin zu lebhaften E-Mail Diskussionen über persönliche Meinungen und Ansichten zu den vielfältigsten Themen rund um Forth und das weitere Universum.

Kontakte in die weite Welt wurden geknüpft und unterhalten – kurzum mein Horizont hat sich erweitert.

Nicht zuletzt: als Editor wusste ich immer als erster, was in der Vierten Dimension steht, konnte schon vor Erscheinen des jeweiligen Bandes zurückfragen und weiterdenken :-) (um Missverständnisse auszuräumen: weiter als mein bisheriger Horizont reichte – nicht weiter als der Autor.)

Natürlich ist die Herausgabe einer Zeitschrift, wie sie die Vierte Dimension darstellt, mit Arbeit und Zeitaufwand verbunden. Davon liegt im konkreten Fall ein gut Teil in der Sache selbst, ein weiter Teil in meiner chaotischen Person. 'Ordentlichere' Menschen als ich, kämen mit weniger Arbeit und Zeit zu ähnlichen Ergebnissen. Das soll heißen: In vielen Dingen hat mir diese Arbeit in den letzten 14 Monaten sehr viel Spaß gemacht- der Stress war (gerade noch?) erträglich.

Ich bedanke mich bei allen Autoren für die gute und gedeihliche Zusammenarbeit, bei Egmont und Ute Woitzel, sowie bei Thomas Beierlein, die die Vervielfältigung und den Vertrieb in mühevoller Arbeit besorgen. Und noch einmal bei allen Autoren.

Bei meiner Frau muss ich mich ganz besonders bedanken, hat sie doch oft auf meine Gegenwart verzichten müssen (Wer mich kennt fragt jetzt: Ist das wirklich schlimm?). Und meinen Ärger (Druckertreiber! etc.) geduldig ertragen.

Ich entschuldige mich für die vielen Tippfehler (manche tun bei wiederholtem Lesen umso mehr weh!) und dafür, dass ich nicht mit allen Autoren so intensiv Kontakt halten konnte, wie ich es gerne gewollt hätte und wie sie es gewiss verdient haben.

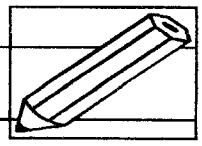
Mit einem lachenden und mit einem weinenden Auge reiche ich die Vierte Dimension wider zurück an Friederich Prinz.

Fritz: Viel Spaß!

Quelltext Service

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können diese Texte auch direkt bei uns anfordern und sich zum Beispiel per E-Mail schicken lassen. Schreiben Sie dazu einfach eine E-Mail an die Redaktionsadresse.

MB



Neue Mitglieder:

Fa. Söring GmbH Medizintechnik

Vor über 10 Jahren unternahm der Firmengründer, Holger Söring, mit seinem Bruder Jörg einen gewagten Schritt: heraus aus der sicheren Angestelltenposition in die Selbständigkeit. Das hochgesteckte Ziel war die Entwicklung eines Ultraschall-disektors, der speziell in der Leber- und Neurochirurgie Anwendung findet zur selektiven Gewebetrennung. Dies gelang mit dem SONOCA I der noch ohne Software einwand-frei arbeitete – wahrscheinlich auch deswegen ☺. „Tissue Managment" lautete von nun an die Devise und damit war der nächste Schritt vorprogrammiert, nämlich die Entwicklung von Hochfrequenzchirurgiegeräten. Das hochgesteckte Ziel: Argon-Plasma-Koagulation. Dies ist eine Technik zur kontaktlosen Blutstillung bei operativen Eingriffen. Nun hielt auch Software in Form von C / Assembler auf einem 8051-Derivat Einzug. Zuerst als Singlechiplösung, dann mit externem Speicher. Im Laufe der Zeit kamen viele unterschiedliche Gerätevarianten sowohl im Ultraschall- als auch im Hochfrequenz-Bereich hinzu.
Parallel zu der bisherigen Evolution ist eine Neuentwicklung in Arbeit. Ziel dabei ist es, einzelne Hardwaremodule mit eigener Intelligenz, in Form eines Hitachi HS8-Derivats, auszurüsten, deren Programmierung in FORTH, genauer gesagt FieldFORTH der Fa. FORTecH, erfolgt. Bestand die bisherige Aufgabe der Gerätesoftware ausschließlich darin die einzelnen Funktionen zu realisieren kommen heute und in Zukunft weitere hinzu, die den Umfang der reinen Funktionalität deutlich übersteigen. Stichwort: Funktionale Sicherheit – ein Konzept, das in Zusammenarbeit mit dem TÜV erstellt wurde. Im Klartext: Überwachung der Gerätefunktionen, die nach bestimmten Kriterien fortlaufend oder beim Selbsttest geprüft werden müssen. Mit den intelligenten Boards wird der Selbsttest weiterhin dazu benutz, beim externen Bestücker Produktionstests durchzuführen. Ein weiterer Schwerpunkt liegt in der Unterstützung des Geräteabgleichs und der Fertigungsdokumentation bis hin zu Selbsttest und Debugging sowie Servicesupport. Das erste Modul ist bereits im Einsatz, weitere werden folgen. Der Einsatz von FieldFORTH hat sich in allen bisherigen Anforderungen bestens bewehrt. Im Rahmen der Entwicklung des ersten Moduls entstanden eine Reihe von Tools, die die weitere Entwicklung vereinfachen werden. Dazu zählt die Compilererweiterung um eine Syntax für Entscheidungstabellen (siehe VD 4 / 2001 „Entscheidungstabellen").
Mit einer ebenfalls auf graphischen Symbolen basierenden Erweiterung lassen sich Zustandsmaschinen in tabellarischer Form beschreiben.
Soweit erst mal zu dem Neumitglied Fa. Söring. Wer uns mal besuchen möchte ist jederzeit herzlich Willkommen entweder virtuell unter www.soering.de oder reell in Quickborn bei Hamburg.
Viele Grüße Klaus Zobawa

Rolf Schöne

Totgesagte leben länger. Forth macht da keine Ausnahme. Die Situation in Amerika sollte uns nicht irritieren. Der Niedergang einer angesehenen Zeitschrift bedeutet noch nicht, daß die Liebe zu Forth erloschen ist. Heutzutage gibt es eben andere Möglichkeiten, Erfahrungen auszutauschen, als noch vor zwanzig Jahren. Forth sucht sich seinen Platz in der Gemeinschaft der Computer-Sprachen und -Systeme. Es ist sicher kein Zufall, daß die englischen Forth-Freunde stark mit ihrem Gemeinschaftsprojekt "Controller Board (HC11 mit PygmyForth)" beschäftigt sind, daß die holländischen Forth-Freunde seit geraumer Zeit fast ausschließlich an ihrem Roboterselbstbauprojekt "autonomer Roboter Ushi (AT89C2051 mit BytcForth)" arbeiten und daß wir in der Forth-Gesellschaft zunehmend Artikel über die von Martin Bitter initiierte Beschäftigung mit dem Lego-Roboter (pbForth) veröffentlichen.
Und es stimmt nicht, daß wir keinen Zulauf von Jüngeren haben. Vielleicht nicht so sehr die Allerjüngsten, die Gameboy-Kids, was durchaus schade ist. Dafür aber die angehenden Entwicklungs-Ingenieure, die Forth ausloten und sich mit anderen Interessenten darüber austauschen möchten. Erstaunlich ist, daß die Arrivierten, die auf Höchstniveau ihrer Schaffenskraft Stehenden, der FG-Gemeinschaft aktiv die Treue halten. Zusehends kommt aber auch eine Gruppe von Mitwirkenden hinzu, die in Forth eine Möglichkeit der Selbstverwirklichung nach erfolgreich abgeschlossener beruflicher Laufbahn suchen und finden. Die Menschheit wird immer langlebiger, und es ist wirklich nicht einzusehen, warum man Forth nicht auch unter dem Blickwinkel des sich langsam und zwanglos aus dem Berufsstreß Verabschiedenden betrachten können soll.
Vom Eintrittsdatum her das jüngste Mitglied der Forth-Gesellschaft ist Herr Rolf Schöne, den ich hiermit und im obigen Sinne im Namen des Direktoriums aufs herzlichste begrüße.
Ich kenne Rolf Schöne seit 30 Jahren. Als wir damals am Institut für Angewandte Mathematik der Technischen Universität München von der Deutschen Forschungsgemeinschaft nach und nach mehrere Analog-, Digital- und Hybrid-Rechenanlagen bewilligt bekamen, dachte zunächst niemand an die immensen Folgekosten, die entstehen, wenn man mit unausgereifter Technik arbeitet. Rolf Schöne hat sich damals als Wartungs- und Betriebsingenieur mit Eigen- und Zusatzentwicklungen bei uns allen einen Namen gemacht. Sein späteres Wirken in der Rechner-Betriebsgruppe des Zentrums für

Mathematik brachte ihm weitere Achtung ein. Ein absoluter Neuling ist Rolf Schöne nicht. In den Jahren um 1990 herum war er bereits Mitglied der FG. Damals ließ ihm aber die berufliche Anspannung keine Zeit für aktivere Beschäftigung mit Forth. Rolf Schöne interessiert sich insbesondere für die Lego-Roboter-Linie in unserer FG und hat bereits Ideen dazu in petto.

Fred Behringer

Turbo für die ganz schnellen?

Hallo Martin,
Übrigens, der IR-Empfänger im RCX oder Turm ist baugleich auch in etwas höherer Frequenz zu haben, der Umbau wäre eine leichte Operation.
Photo Modules for PCM Remote Control Systems
Available types for different carrier frequencies

- TSOP1130 30 kHz TSOP1133 33 kHz
- TSOP1136 36 kHz TSOP1137 36.7 kHz
- TSOP1138 38 kHz TSOP1140 40 kHz
- TSOP1156 56 kHz

Rettet uns das? Wir werden sehen.

Grüße, Michael

2001 – Vorbei?

Alle, die dies nicht so ganz glauben können wollen – können ja einmal bei:
http://www.ifilm.com/ifilm/product/film_info/0,3699,1338602,00.html vorbeischauen. Wer's tut, kennt dann auch den Bezug zu einigen der vorherigen Meldungen.

MB

pbForth Version 2.0.0 verfügbar

Bei Ralph Hempel steht seit kurzer Zeit pbForth in der Version 2.0.0 zum Download bereit.
pbForth ist in der neuen Version wesentlich kleiner und kompakter. Dies wurde im Wesentlichen durch Entfernen einiger Worte erreicht, die nicht zum CORE ANSI Standard gehören. Ralph Hempel hat sich dazu entschlossen, weil ihn viele Rückmeldungen verunsicherter Anwender erreichten, die mit dem Multitasker und den Wordlists nicht zurechtkamen und sich dadurch verwirrt fühlten.
Ich persönlich finde das ausgesprochen Schade. Denn gerade die Möglichkeit mit Wortlisten zu arbeiten, macht im Unterricht sehr viel Sinn. Aber wer weiß, vielleicht gibt es das Vermisste bald zum Eincompilieren in Hochsprache?
Über ein X-MODEM Protokoll läßt sich der gesamte RCX-Systeminhalt als *.SREC Datei auf den PC zurückladen. So kann jeder Anwender sich sein Wunschsystem gestalten und es dauerhaft sichern.
Das Übertragungsprotokoll zwischen PC und dem Lego-RCX ist verbessert worden. Es soll nun zuverlässiger sein. Allerdings muss Software, die die bisherigen Besonderheiten des Protokolls nutze (ein wenig) umgeschrieben werden.
http://www.hempeldesigngroup.com/lego/pbFORTH/index.html

Errata

Lieber Martin,
zu meinem Schreck habe ich Ungereimtheiten in meinem Beiträgen gefunden und bitte um Vergebung - der Sinn war hoffentlich nicht zu sehr entstellt dadurch.
In Schaltplan ist R der Widerstand mit dem Wert 330K. Im Text steht leider nur R statt R=330K. Im Text steht auch noch "...sehr hochohmig an Punkt A angekoppelt hält DTR die Schaltung aktiv." Im Bild ist dieser Punkt dann aber als R markiert, also als der Widerstand R=330K wie auch sonst im Text. Am 9-poligen Stecker des Turms ist Pin2/3 RxD/TxD, Pin4=DTR, Pin5=Gnd und Pin7/8 RTS/CTS - aber das hatte sich ja schon rumgesprochen, oder?

Grüße, Michael

In der Vierten Dimesion Nr. 4/2001 auf der Seite 28 steht im dritten Absatz der linken Spalte: „ ... Aus dem Vorrat von 12 Zeichen (0000 bis 1111) ...." In dem gemeinten Zahlensystem lassen sich mit vier Bits 16 Zeichen darstellen, aber vielleicht kennt der Fred ja ein Zahlensystem in dem 1111 für 12 steht?

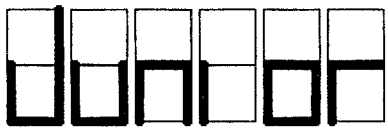
MB

Stimmt's?

Ich weiß nicht mehr, wo ich diesen Text gefunden habe, glaube mich aber zu erinnern, dass es ein Muster für ein OCR-Programm war. Es geht um Programmierer.
"Es ist für den Unverständigen schwer zu begreifen woran sie eigentlich arbeiten. Befragt man sie, so erhält man übrigens detaillierte und geduldige Auskunft darüber, dass sie an etwas arbeiten, was die unabdingbare Voraussetzung für ein weiteres Vorhaben ist, das vielleicht seinerseits nur Mittel zum Zweck ist. Nie findet man sie mit etwas Endgültigem, es scheint die Essenz ihres Strebens zu sein, dass sich alles im Fluss befindet. Vielleicht hat ihr Hobby eigentlich keinen Zweck und ist somit das edelste überhaupt; sie arbeiten unermüdlich für etwas, das sie nie erreichen, dem sie nicht einmal nahekommen, ein Zustand endloser Glückseligkeit!" *MB*

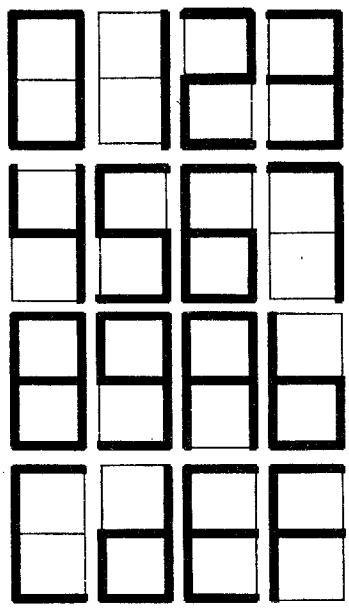
Alphabetisierungskampagne

Guten Morgen Martin,
warst du es nicht der in der letzten VD einen Zeichensatz für den Legoklotz suchte?



Hier einige Anregungen für deine Schüler aus einem Buch zu einer Experimentierplatine um den 6502 herum aus jenen Zeiten - "Junior Computer", 1980, Elektorverlag - das Teil hatte eine 7-Segement HEX Anzeige.

Viel Spaß, Michael



Danke, Michael!
Aber die Junior-Leute (oder Du?) drücken sich vor den wirklich schwierigen Buchstaben 'm', 'w' und 'x'. Vorschlag meiner Schüler: im Falle 'w' durch 'v' oder auch durch 'vv' ersetzen. Bei 'm' und 'x' suchen wir noch.
MB

Tower forever – unter DOS

Rolf Schöne

Der Artikel „Tower forever – die zweite" von Michael Kalus und Adolf Krüger in VD 4/2001 S.10f ist gelungen. Ein dickes Lob an die beiden. Ich habe sofort die Diode eingelötet (das ist die sauberere der beiden vorgeschlagenen Lösungen) und bin unter DOS mit dem Terminalprogramm RCX_ZF von Martin Bitter sofort auf dem Schlauch gestanden.

Warum? Das Signal DTR wird zwar von Hyperterminal unter Windows, nicht aber von DOS bedient. Dem kann aber mit einem kurzen DEBUG-Skript abgeholfen werden, das DTR auf die geforderten +12V anhebt. DEBUG hat jeder, der wie ich immer noch an DOS hängt, weil er wissen will, was da so abläuft.
Das Skript (einzugeben ist nur alles links von „ ; ") können Sie entweder in DEBUG direkt ausführen oder vorher mit einem Editor zur Dokumentation in eine Datei „dtr-ein.src" fassen, die Sie dann mit „DEBUG < dtr-ein.src" abarbeiten lassen.

```
a 100                   ; Assembler-Modus Ein
mov dx,2fc              ; 3fc=COM1, 2fc=COM2
mov al,1                ; 1=DTR on, 0=DTR off
out dx,al               ; Set Modem Register
xor ax,ax               ; ax=0 für „Kein Fehler"
int 20                  ; zurück zum DOS
                        ; Leerzeile (Ass-Mod Aus)
r cx                          ; Länge angeben
0a                      ; 10 Bytes, nicht mehr!
n dtr-ein.com           ; Datei-Namen definieren
w                       ; Datei speichern
q                       ; Debug verlassen
```

Wie Sie im Kommentar sehen, kann man auf diese Weise auch COM1 bedienen, und in den Stromspar-Modus zurückschalten kann man den Tower auch (dazu den Datei-Namen auf z.B. „dtr-aus.com" ändern).
Fragen bitte an rolf@rolf-schoene.de

Die Datei dtr-ein.com

```
a 100
mov dx,2fc
mov al,1
out dx,al
xor ax,ax
int 20

r cx
0a
n dtr-ein.com
w
q
```

Das nenne ich löblich!
Kaum als neues Mitglied begrüßt, (vgl. S. 5) schon einen Betrag zu der Vierten Dimension geleistet!

Verba docent
exempla trahunt!

MB

Wirklich Schluß?

Hallo Michael und Adolf,
ich habe mir für die Besprechung in der Forthwrite nun auch Euren Artikel "Zur Lego-Mindstorms-Infrarot-Datenübertragung" intensiv durchgelesen. Eine sehr gute Analyse! Euer Ergebnis, Mehrfachbetrieb geht mit der gelieferten Hard-und Firmware nicht, fordert natürlich jeden anständigen Widerspruchsgeist - und Forthler sind das von Natur aus - heraus. Ich bin mal gespannt, was die Protokoll-Spezialisten-Gruppe dazu sagt.
Mit herzlichem Gruß Fred

Preisausschreiben

Martin Bitter

Während einer angeregten Diskussion über Dies-und-Das mit Ulrich Paul erwähnte ich als Beispiel für kompakten Code und den Entwicklungsvorsprung der Natur gegenüber der Informatik einen (mir leider nicht mehr namentlich bekannten) Virus und meinte dabei, dass es wohl sehr schwer sei, dessen 'Trick' in einer Programmiersprache nachzuahmen. Ulrich meinte: „Setz' doch einen Preis aus!" Das geschieht hiermit!

Zu gewinnen sind: Nicht nur Ruhm und Ehre sondern auch zwei der berühmten Kaffeebecher mit Swappiemotiv aus der 'Dingdener Tagungsserie 2001'.

Viren und soweit bekannt alle Lebewesen, speichern ihre Erbinformation in DNS-Ketten. Dabei sind gerade Viren Meister der Komprimierung, indem sie eine große Menge (redundanter) Erbinformation ausgelagert haben und Teile der DNS ihrer Wirte benutzen. Viren alleine sind nicht fortpflanzungsfähig, sie benötigen dazu den Vererbungsapparat ihrer Wirte. Manche zählen sie deshalb nicht zu den Lebewesen.

In einem DNA Strang können ca. 20 verschiedene Eiweiße, als 'Bausteine' des Lebens, kodiert werden. Dies geschieht jeweils durch ein Triplett der vier Basen **Cytosin** (C), **Tymin** (T), **Guanin** (G), **Adenin** (A). Mit vier Basen wären mehr als 20 Kombnationen möglich, es werden aber 'nur' die erwähnten 20 genutzt. Einige Viren haben die Enden ihres DNA-Strangs zu einem Ring verbunden. Bei Ein- und Mehrzellern und Viren kommt es häufig vor, dass Teile der DNA mehrfach abgelesen und in Proteine übersetzt werden. Das ist vergleichbar mit DO ... LOOP Konstrukten. Mindestens ein Virus (s.o.) hat dieses Verfahren noch weiter optimiert: Seine DNA wird beim zweiten Lesedurchlauf versetzt gelesen!

Bei einem Computerprogramm entspräche dies einem Zurücksetzen des PC (Programmcounters) mit Missalignment!!

Und das ist auch schon die Aufgabe:

Schreiben Sie ein Forthprogramm, in dem ein Teil des Codes zweimal durchlaufen wird, derart, dass beim zweiten Durchlauf der Code um die Größe einer halben Zelle versetzt gelesen wird. Der zweite Durchlauf kann auch durch Anhängen einer Kopie des (Teil-) Programmkodes mit einer fehlenden halben Zelle simuliert werden.

Bsp. Es sei dies ein Hex-Dump eines Stückes funktionierenden Kodes einer 16-bit Forthmaschine:

9A 0B 87 79 06 EE 5E 5E 00 06 88 7F D8 72 50 FE 67

dieses Kodestück beraube man seines ersten und letzten Bytes (=1/2 Zelle),:

A0 B8 77 90 6E E5 E5 E0 00 68 87 FD 87 25 0F E6

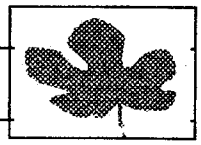
und hänge es an das erste Codestück an:

9A 0B 87 79 06 EE 5E 5E 00 06 88 7F D8 72 50 FE 67 A0 B8 77 90 6E E5 E5 E0 00 68 87 FD 87 25 0F E6

Nun muss dieser Kodestrang interpretierbar sein!

Bewertungskriterien: (in aufsteigender Reihenfolge, und kombinierbar)

- Maschinencode vs. Hochsprache. Bei ähnlicher Problemlösung gewinnt der Vorschlag in Hochsprache!
- Sinnhaftigkeit: das Programm stürzt nicht ab, eine der beiden Teilsequenzen macht etwas 'sinnvolles', beide Teilsequenzen machen etwas 'sinnvolles'. Was wie sinnvoll ist entscheide ich .-)
- Länge der Sequenzen: je länger je lieber.
- Dirty tricks: Aneinanderhängen von Bild- oder Sounddaten zu 'non-Sense'-Bildern (Sounds) werden ebenso wenig berücksichtigt, wie short oder far Jumps hinter die betreffenden Kodestücke. Beide Kodeteile müssen ganz durchlaufen werden!
- Darstellbarkeit: Der Teilnehmer muss durch seine Art der Darstellung (für mich!) nachvollziehbar machen, dass sein



Programm funktioniert. Ich bin da auf Treu und Glauben angewiesen. (Immerhin Win- und Linux-PC Programme kann ich verifizieren).

- Nebenwettbewerb: Nach dem gleichen Prinzip kann auch ein Textstück eingereicht werden. Das ist ein ASCII-kodierter Text, der grammatikalisch und semantisch Sinn macht und in seiner Hex-Dumpdarstellung dem obigen Beispiel entspricht.

Der Gewinner oder die Gewinnerin werden in der Vierten Dimension 2/2002 bekannt gegeben. Die Preise werden auf dem Postweg kostenfrei zugesandt. Sollte der Gewinner oder die Gewinnerin an der Forthtagung in Garmisch (Einladung in diesem Heft) teilnehmen, erfolgt die Preisübergabe in feierlichem Rahmen dort .

Die eingesandten Beiträge dürfen von der Redaktion veröffentlicht werden. Allerdings besteht kein Veröffentlichungsanspruch (stellt Euch einmal vor alle machen mit!).

Lösungen bitte per E-Mail an:
VD@FORTH-EV.de oder
martin.bitter@forth-ev.de

Mit der 'gelben Post' an:
Martin Bitter
Möllenkampweg 1a
46499 Hamminkeln

Allen Teilnehmern und Teilnehmerinnen wünsche ich viel Spaß!

Martin Bitter

Gehaltvolles

zusammengestellt und übertragen von Fred Behringer

VIJGEBLAADJE der HCC Forth-gebruikersgroep, Niederlande Nr. 28, Oktober 2001

Nieuws op 8051 gebied II
Willem Ouwerkerk <w.ouwerkerk@kader.hobby.nl>

Willem beschreibt kurz einige Abkömmlinge des 8051: Siemens SAB80C515A und SAB80C535C, ATMEL (TEMIC) T89C51RD2, Cygnal C8051F012, Dallas DS89C420 und DS87C550, Philips P89C668, Analog Devices AduC812S.

De belofte voor 2001
Willem Ouwerkerk

In diesem vierten Teil der Beschreibung des holländischen Roboter-Bauprojekts (Ushi, die an der Tischkante entlang fährt, ohne runterzufallen) wird das zweite fertige Versuchsmodell und die grundlegende Byte-Forth-Software besprochen. Es werden sechs Miglieder der holländischen Forth-Gruppe, die zum Gelingen beigetragen haben, namentlich genannt. Ushi kann mit einer RC5-Fernbedienung gesteuert werden. Über diese Fernbedienung ist es auch möglich, den Roboter in den "autonomen Zustand" zu versetzen - und ihn bei Bedarf auch anhalten zu lassen. Im autonomen Zustand fährt er drauflos, weicht mit Hilfe des Abstandsensors GP2D02 Hindernissen aus und versucht, nicht vom Tisch zu fallen.

Niederländisch ist gar nicht so schwer. Es ähnelt sehr den norddeutschen Sprachgepflogenheiten. Und außerdem ist Forth sowieso international. Neugierig ? Werden Sie Förderer der

HCC-Forth-gebruikersgroep.

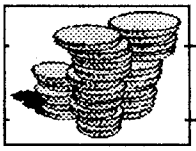
Für 20 Gulden pro Jahr schicken wir Ihnen 5 oder 6 Hefte unserer Vereinszeitschrift 'Het Vijgeblaadje' zu. Dort können Sie sich über die Aktivitäten unserer Mitglieder, über neue Hard- und Softwareprojekte, über Produkte zu günstigen Bezugspreisen, über Literatur aus unserer Forth-Bibliothek und vieles mehr aus erster Hand unterrichten. Auskünfte erteilt:

Willem Ouwerkerk
Boulevard Heuvelink 126
NL-6828 KW Arnhem
E-Mail: *w.ouwerkerk@kader.hobby.nl*

Oder überweisen Sie einfach 20 Gulden auf das Konto 525 35 72 der HCC-Forth-gebruikersgroep bei der Postbank Amsterdam. Noch einfacher ist es wahrscheinlich, sich deshalb direkt an unseren Vorsitzenden, Willem Ouwerkerk zu wenden.

Oproep

Der Redakteur ruft dazu auf, mehr Forth-Artikel aus anderen Gebieten einzureichen, damit das Vijgeblaadje nicht weiterhin so hardwarelastig bleibt, wie es das in den vergangenen Monaten war. Die Homepage der holländischen Forth-Freunde: http://www.forth.hccnet.nl/



Stack-FORTH

Einige Gedanken zur virtuellen Maschine

Sören Tidemann

FORTH wurde konzipiert um unnötige Probleme zu vermeiden.

FORTH ist ein Werkzeug, um einfache Lösungen in den Vordergrund zu stellen.

FORTH basiert auf einer (virtuellen) Stack-Maschine.

Kategorie 1:
2*, 2/, not, and, xor, +, +* **(*, /, ...)**

Kategorie 2a:	***Kategorie 2b:***	
a, dup, over	**a!, r>, >r, drop**	**(r@, swap, ...)**

Kategorie 3a:	***Kategorie 3b:***	
#	**@, @+, !, !+**	**(c@, c!, ...)**

Kategorie 4:
(-)if ... then, -;, : ... ; (begin . until,
begin ... while... repeat, ...)

Kategorie 5:
nop **(nip, tuck, pick, bung, rot, roll,**
case ... endcase, ...)

Kategorie 1 zeigt die zum Rechnen und zur Datenbearbeitung notwendigen Operationen. Der effektivste Weg ist, die Operanden auf dem Stack zu verwalten. Für mehrere Operationen werden möglicherweise Kopien benötigt. Also sind in Kategorie 2a der Stackmaschine Anweisungen aufgeführt, die es erlauben, den Top of Stack (T), den Next of Stack (N) und den Inhalt des Adressregisters (A) zu kopieren. Aus diesem Design erkennt man schon, dass diese Stackmaschine eigentlich nur zu 3 Argumenten für eine Funktion anhält! Nur die 3 erwähnten Positionen können mit der Maschine direkt erreicht werden! Zwischenergebnisse können mit den Anweisungen in Kategorie 2b zwar verwaltet werden (sie können auf die verschiedenen Teile des Stacks gerettet und später wieder zusammengeführt werden), man sollte sich aber hüten, die Anweisungen in 2b dazu zu missbrauchen, 4 oder mehr Werte auf dem Stack zu jonglieren. Dies wird höchstwahrscheinlich zu ineffektiverem Code führen, als wenn Speicherzugriffsbefehle aus der nächsten Kategorie verwendet würden (die Idee der schnellen Verwaltung von Operanden auf dem Stack wäre damit zunichte gemacht). Es wird hier schon deutlich, dass ein klares, zielstrebiges Design und eine intelligente Verwaltung von Daten notwendig sind, um wirklich Vorteile aus dem Stack zu ziehen! Aber irgendwie müssen ja die Operanden überhaupt auf den Stack kommen. Natürlich, dazu stellt Kategorie 3a den Literalbefehl und 3b nach vorheriger Initialisierung des Adressregisters die Speicherzugriffsbefehle zur Verfügung. Ein eigenständiges Adressregister ist sinnvoll, da es den Stack von unnötigen Rangierereien entlastet und ihn damit eben nicht verschmutzt (analog wird ja der Datenteil des Stacks auch nicht mit Rückkehradressen belastet und durchsetzt. Diese landen ja bekanntlich im Returnteil). Vielleicht ist es aufgefallen, dass ich die Begriffe Datenstack und Returnstack vermeide. Ich sehe die Logik in FORTH eher durch EINEN STACK mit VERSCHIEDENEN EINGÄNGEN und Richtungen beschrieben. Das STACKFENSTER (R -T -(N)) mit den Eingängen R und T wird über den Stack geschoben ...

Mit dem Autoinkrement lassen sich z.B Arrays bequem durchwandern. Selbstverständlich kommt ein Programm nicht ohne Kontrollstrukturen aus. Doch der Grund, warum diese erst in Kategorie 4 aufgeführt werden ist, dass Kontrollstrukturen allein nicht effizient sind. Es bedarf einer optimalen Lösung, um unnötige Kontrollstrukturen zu vermeiden. Kategorie 5 zeigt die am wenigsten effektivsten Anweisungen. Sie gehören nicht in die zugrunde liegende (virtuelle) Maschine.

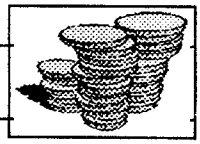
Die Kategorien enthalten den von Chuck Moore als 'kleinsten optimalen' angesehen Satz von FORTH-Primaries. In Klammern dahinter habe ich einige gängige weitere herkömmliche FORTH-Primaries angedeutet.

In Kategorie 1 leistet das 'xor' etwas mehr als ein 'or'. Beide sind nicht notwendig. (Das 'or' kann leicht dargestellt werden: 'over com and xor')

Es scheint ein '-' zu fehlen, doch das Einerkomplement reicht in Verbindung mit dem '+' aus. Es ist eine Frage des Settings der Daten. Es ist möglich, sich an die Einerkomplement-Subtraktion zu gewöhnen. Oftmals stellt man fest, dass man die Daten so verwalten kann, dass diese Subtraktion gerade richtig ist. (Trotzdem kann das echte '-' natürlich leicht dargestellt werden)

Das '+*' dient als Multiplikationsschritt, da eine vollständige Multiplikation in diesem Set nicht enthalten ist.

Kategorie 2 enthält die Anweisungen 'a' und 'a!' um das Adressregister zu bedienen. Auf den besonderen Wert wurde schon hingewiesen. FORTH-Code wird einfach sauberer.



Beispiel:

```
Addr # a!
@+ @+ @+ a
...
```

ist sauberer als:

```
Addr
DUP @ SWAP CELL+
DUP @ SWAP CELL+
DUP @ SWAP CELL+
...
```

'over' oder 'swap'? Das ist eine gute Frage. Beide sind in der zugrundeliegenden Maschine nicht notwendig. Sie dienen dem Erreichen des Next of Stack. Wer von beiden kann mehr? Das 'over'! Es ist ja ein gewisses 'swap' UND kopiert gleichzeitig UND ist einfacher zu realisieren!

Ein 'r@' wäre echt nett ... aber es kann leicht nachgestellt werden. (r> dup>r); daher ist auch kein Platz dafür in der minimalen Maschine.

In Kategorie 3 tauchen dementsprechend konsequent zusätzlich der Autoinkrement-@, und -'!' auf. Ein Autodekrement ist nicht notwendig, eine Richtung reicht völlig aus. Byteweiser Zugriff kann programmiert werden, es ist kein echter Nachteil, dieses nicht im Grundwortschatz zu haben.

Kategorie 4 enthält das notwendige 'if (Sprung bei T=0, sonst weiter) ... then', wie auch eine Variante, die das Carry-Flag testet (-if, Sprung bei C=0), die besonders für Rechnungen, wie auch für Vergleiche sehr geeignet ist.

Das '-;' ist die sogenannte Tail-recursion (ein JUMP). Es wandelt einen CALL in einen JUMP:

```
: los_geht's   Wort_1 Wort_2 ... Wort_x ;
```

kann etwas optimiert werden. Wort_x wird nach seinem eigenen 'return' dann den 'return' von 'los_geht's' ausführen. Besser ist, statt des 'CALLS' zu Wort_x, einen 'JUMP' zu Wort_x zu compilieren:

```
: los_geht's   Wort_1 Wort_2 ... Wort_x -;
```

Jetzt beendet Wort_x auch gleichzeitig 'los_geht's'!

Mithilfe der Tail-recursion können auch einfach sämtliche nur denkbare Kontrollstrukturen erzwungen werden, ohne diese explizit definieren zu müssen:

```
: los_geht's   Bedingung if  Wort_1 Wort_2 ...
Wort_x  los_geht's -; then ;
```

entspricht:

```
: los_geht's   BEGIN Bedingung WHILE Wort_1 Wort_2
... Wort_x REPEAT ;
```

und drückt mindestens genauso gut aus, was gemeint ist.

Zu Kategorie 5 braucht nicht viel gesagt werden. Das 'nop' ist hardwareseitig zum Wort-Align bei Chuck's 'echten' FORTH-Maschinen erforderlich.

Alle in Klammern aufgeführten Jonglier-Künste haben viel mit dem Lösen von überflüssigen Problemen und wenig mit FORTH zu tun. Schlimmer noch, sie sind oftmals nicht nur langsam, sondern blockieren auch ein saubereres Design! Selbst Variablenzugriffe sind ja schneller als so manches dieser Worte!

In dieser Form sind 25 FORTH-Primaries beschrieben, die eine ganz saubere Stackmaschine beschreiben. Dies ist meiner Meinung nach der Ansatzpunkt, um FORTH zu vermitteln. Warum sollten wir x86'er Maschinencode, oder ARM-Opcodes oder 68000'er oder was sonst noch immer zu Rate ziehen, um FORTH zu erklären? Wir können es doch besser! Wie in den Anfangstagen der uralten FIG-Listings, besteht doch der Trick darin, FORTH in FORTH zu erklären. Und wir haben jetzt ein ideales Werkzeug! Chuck's Maschinenforth. Einfacher geht's wirklich nur noch unter Inkaufnahme von Ineffizienz. Chuck's Primary-Satz steht genau auf dieser Grenze. Er ist für sich allein noch gerade effektiv genug, um alles zu erledigen, ohne dass es schmerzt. Ganz im Gegenteil, die Vorteile liegen im Detail! Alle bequemen Dinge, die er nicht aufweist, können ganz einfach mit den vorhandenen Worten aufgebaut werden. Es sind 25 echte Primaries! Diese Maschine kann sehr effektiv programmiert werden! Und, was viel wichtiger ist, sie ist SAUBER! Ich meine damit, dass so furchtbar schmutzige Dinge wie 'DEPTH', 'SP@', 'RP@', 'PICK' oder 'ROLL' dann einfach NICHT funktionieren! Dies sind Worte, die die Idee der Stackmaschine katapultieren! Warum? Weil diese Worte eine Indizierung jedes Eintrages verlangen! Dies hat eher mit Arrays denn mit Stack zu tun. Diese Worte zeugen von dem Versuch, die reine Stackmaschine zu optimieren, indem man sie konventioneller macht. Dies hat mit Implementierungen auf Maschinen zu tun, die eben als Stackmaschinen mehr schlecht als recht geeignet sind. Wenn die Stacks im externen Hauptspeicher simuliert werden müssen, ja, dann ist die Stackmaschine FORTH auf diesen Rechnern langsamer als 'C'. Es sei denn man 'borgt' sich mal hier oder dort etwas! Genau dies tun die oben genannten Worte in der Regel! Aber nochmal, dies führt von FORTH weg. Ein Einsteiger müsste, um FORTH dann zu verstehen, neben der entsprechenden Maschinencharakteristik auch noch andere Philosophien der Programmierung verstehen! Und ein Umsteiger wird sich schnell fragen, warum er sich das antun soll, er kann es ja bereits schon schneller!!! Mit dem Maschinenforth-Primaries aber ist FORTH nicht nur auf der Hardware-Ebene realisiert. Diese einfachen Dinge können in wenigen Stunden in nahezu

1:1 Entsprechung auf sämtliche gängigen Prozessoren übertragen werden. Wir können uns hinstellen und sagen: Schau, 'dup' ist auf dieser Maschine der Name für ... Damit haben wir es geschafft! Der Einsteiger wird sich mit Freuden auf diese einfachen und sprechenden Namen stürzen und seine Maschine damit erobern! Der Umsteiger wird erstaunt feststellen, wie einfach er der Maschine näher rücken kann. Und für unsere Genugtuung wissen wir, dass es für diesen Satz von Primaries echte FORTH-Hardware gibt, die rasend schnell, wahnsinnig speichereffizient ist und auf denen 'C' nicht gut zum Laufen zu bringen ist ;-)

Ich stimme Ulrich Paul vollkommen zu, dass so etwas wie:

```
1) ...                  ( a b c )
   >R OVER OVER         ( a b a b )
   R@ -ROT              ( a b c a b )
   R>                   ( a b c a b c )
   ...
```

effizienter durch

```
2) REORDERER 3DUP ( a b c -- a b c a b c )
```

erledigt wird.

REORDER(ER) ist eine fantastische Lösung für ein Problem, dass aus einem falschen FORTH-Verständnis heraus geboren wurde.

Wie im übrigen 'CASE ... ENDCASE' auch. Nun ja, es gibt viel davon ...

Das Problem ist eigentlich nicht, dass es solche Sachen gibt, sondern dass man sie benutzt, wenn sie da sind! Wenn ich ANS-Forth kodiere, dann kann ich mir nicht helfen, in meinem Code tauchen auch all diese Dinge auf. Komischerweise vermisse ich nichts davon, wenn ich beim Maschinenforth bin. Ich vermisse 'swap' dort wirklich nicht. Und weiter, nicht nur, dass es mich nicht stört, nein, ich habe gar kein echtes '-', ob nun als Makro oder als Wort, definiert. Ich brauche es komischerweise dort nicht. Das Denken ist ein anderes.

Bei 1) könnte sich einem immerhin noch der Gedanke aufdrängen, dass da irgendetwas nicht stimmt. Dieses mulmige, zu Übelkeit und Brechreizen führende Gefühl im Magen ..., bei 2) taucht das nicht auf. Man fühlt sich definitiv besser, an der eigentlichen Ursache hat dies aber ebenso definitiv nichts geändert.

Also, vergessen wir diese Dinge für ein paar Momente.

Es ist möglich, sehr komplexe und doch effiziente Programme mit nur 4 Returnstack-und 4 Datenstackpositionen zu schreiben.

(Mehr zu meinem MEDRA-OS und dem Window-Manager ALEGRA in den nächsten VD's)

a) Ein wenig des Nachdenkens über Strukturen und Algorithmen ist halt gefragt ...

b) Und Geduld ...

c) Und die Bereitschaft sein Denken immer wieder in Frage zu stellen und zu a) zurückzugehen ...

d) Und die Kraft, der Versuchung eines 3DUP nicht zu erliegen sondern zu c) zu verzweigen ...

Dann wird man mit dem Gefühl belohnt, es richtig verstanden zu haben ...

Und nebenbei entdeckt man den FORTH-Gedanken wieder neu ...

Grüße aus Freiburg,

Sören

Gehaltvolles

zusammengestellt und übertragen von Fred Behringer

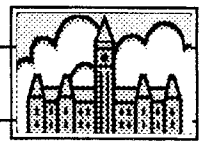
FORTHWRITE der FIG UK, Großbritannien Nr. 114, November 2001

Die Anzeige der Forth-Gesellschaft zur Mitgliederwerbung steht diesmal in Großaufmachung auf der Innenseite des vorderen Deckblattes. Wir können sehr zufrieden sein.

Chris Jakeman begrüßt in seinem Editorial ein neues Mitglied und einen Wiedereinsteiger. Er weist darauf hin, daß in der vorliegenden Ausgabe auch drei Nichtmitglieder Beiträge veröffentlicht haben. Die Forthwrite steht auch für Nichtmitglieder weit offen. Die Website (Webmaster Jenny Brien) wird von über 1000 Interessierten pro Monat besucht.

2 Forth News

Tiny Open Firmware; Forth für .NET; FICL 3.01; FIJI, der Forthige Java-Interpreter; FTRAN; http://www.forth.org.ru, die russische Website, hat jetzt auch eine englische, französische und deutsche Übersetzung; "Ein Forth-Leitfaden für



Anfänger" von Julian Noble jetzt in HTML-Format; Online-Fan-Club für Win32Forth; die F11-UK-Mailing-Liste ist umgezogen .

5 Tiny Open Firmware - Extensibility for Small Embedded Systems
Brad Eckert <brad@tinyboot.com>

Der Autor, ein Physiker und Hardware-Ingenieur, ist seit 15 Jahren damit beschäftigt, eingebettete Systeme zu entwickeln. Open Firmware trägt bekanntlich zur Plug-and-Play-Möglichkeit für Apple-, Sun- und Power-PCs bei. Brads "Tiny-Open-Firmware" macht dies auch für kleinere eingebettete Systeme möglich. Ein kostenloses Windows-Programm kann unter dem Namen "Firmware Studio" heruntergeladen werden von: http://www.tinyboot.com . Der Quelltext ist dort ebenfalls beigelegt. Firmware Studio verwendet einen normalen Forth-Interpreter, um in den Wirts-PC Forth-Code zu laden, und in den Zielprozessor Maschinen-Code. Mit einem Tokenizer kann man anstelle des Maschinen-Codes im Zielprozessor auch tokenized Forth erzeugen.

11 From the 'Net

"I Hate Forth" steht provokativ auf http://www.embedded.com von Jack Ganssle. Es werden 14 Namen aufgezählt, die in verteidigender Weise positive Stellung für Forth bezogen haben. Steven R. Commer: "Wenn sich Leute einen Strick kaufen und sich daran aufhängen, kann man den Strick nicht dafür verantwortlich machen. Wenn man bei manch einem Forth-Programmierer den Code nicht lesen kann, liegt das nicht an Forth."

12 An Interview with Tom Zimmer - Forth System Developer
Jim Lawless: http://www.radiks.net/jimbo

Mit Jims Genehmigung von Jims Homepage kopiert. Tom Zimmer hat 1968 die Highschool erfolgreich abgeschlossen. Er arbeitete dann bei Pacific Telephone als "Central Office Equipment Man", was auch immer das heißen mag. Ein Freund machte ihn etwas mit Elektronik und Computer-Hardware bekannt. Aber lesen Sie selbst weiter! Sieben Seiten aus dem Munde eines Menschen, den wir alle irgendwie bewundern. Hier einige Abschnittsüberschriften: Wie machten Sie das erste Mal mit Forth Bekanntschaft? Was hat Sie veranlaßt, für so viele Systeme Forth-Software zu entwickeln? Warum haben Sie das anfangs immer in Form von Steckmodulen getan? (Was sagte Ihre Frau dazu, daß Sie stets bis zur Erschöpfung arbeiteten?) Hatten Sie Industriekontakte zur Vermarktung Ihrer Entwicklungen? Hatten Sie die Assemblersprache für alle Prozessoren, mit denen Sie sich beschäftigten, wirklich gemeistert? ... Es folgt ein für mich, den Rezensenten, sehr interessanter und wichtiger ganzer Abschitt darüber, warum sich nach Toms Meinung jeder junge Programmierer zu allererst auch einmal mit Forth beschäftigen sollte - und welche andere Sorte von Menschen besser davon Abstand nehmen. Aber lesen Sie selbst. Ich kann das wärmstens empfehlen! -- Noch ein letztes Wort: Es werden mindestens sieben Firmen aufgezählt, in denen Tom Zimmer nach dem Highschool-Abschluß mit Entwicklungsarbeiten beschäftigt war. (Theoretische) Ausbildung wird nicht erwähnt. Zu einem großen Teil hat er auch "mit Forth Geld verdient". Als reinen Hobbyisten kann man ihn ganz bestimmt nicht bezeichnen. Aber die Begeisterung für Forth spürt man aus seinen Worten heraus. Ein Praktiker, der in Forth seine Berufung gefunden hat.

20 F11-UK
jeremy.fowell@btinternet.com

Alle nötigen Informationen über die Forth-Controller-Karte der Forth Interest Group UK . Forth läuft auf der Karte interaktiv, bis zu 8 Analogleitungen, 20 E/A-Leitungen, Erweiterungen über den HC11-SPI-Seriell-Bus mit 2 oder mehr von 20 Leitungen, RS232, UART onboard.

21 euroForth 2001

Inzwischen gelaufen. Mehr als 16 Vorträge. Ehrengast Charles Moore.

22 FIG UK - AGM Report
Chris Jakeman <cjakeman@bigfoot.com>

Versammlung im Hause von Dr. und Mrs. Neale. Jeremy Fowell wurde als Vorsitzender (Chairman) bestätigt, Graeme Dunbar als Bibliotheksverwalter. Website auf 60 Seiten angewachsen, 1000 Besucher und 600 Forthwrite-Downloads pro Monat, Webmaster Jenny Brien sehr aktiv, allmonatlicher IRC-Chat mit regem Zuspruch, Forthwrite liegt in verschiedenen Universitätsbibliotheken aus, "we should try to learn from our German colleagues in Forth Gesellschaft who run a friendly and effective annual FIG conference", es soll eine FIG-UK-CD herausgegeben werden.

23 A Call to Assembly 1/3Julian Noble <jvn@virginia.edu>

Der zweite Teil der dreiteiligen Artikelserie über die Verwendung von Assembler in Forth. Diese Artikelserie war ursprünglich für die amerikanische Vereinszeitschrift Forth Dimensions gedacht, die ja nun leider sanft entschlafen ist. Julian richtet sein Augenmerk auf F-PC mit Hinweisen auf ANS-Forth. Es geht diesmal um die Überführung von Klein- in Großbuchstaben (`LCASE?` und `UCASE`) und deren Assemblerfassungen. ANS-Forth läßt diese Dinge offen und dem

System-Implementierer anheimgestellt. Der Autor diskutiert auch die bei C und QBASIC auftretenden abweichenden Stringformate und gliedert in plattformabhängige und plattformunabhängige Teile seiner Programmfassungen. Interessant sein Vorschlag eines "Mikromini-Assemblers" (<% 5B FF 37 %> statt 5B C, FF C, 37 C,), um ganz auf die Schnelle "Inline-Code" zu erzeugen.

30 Charles Moore Interview on Slashdot
George Morrison <gdm@gedamo.demon.co.uk>

Auf http://slashdot.org, der Slashdot-Webseite, findet man laufend Berichte über neuere technische Entwicklungen. Kürzlich stand hier ein Interview mit Charles Moore über dessen 25x-Prozessor und Forth. C.M. ist von anderen Sprachen als Forth wenig beeindruckt und sieht keinen Fortschritt in deren Entwicklung. Er bleibt dem Forth-Paradigma verhaftet. Wenn er in Forth eine Schwachstelle findet, geht er einfach hin und korrigiert sie.

31 Vierte Dimension 3/01
Alan Wenham <101745.3615@compuserve.com>

Alan bespricht wieder den Inhalt. Überrascht bin ich, der Rezensent, über den Vorspann von Chris Jakeman, dem Redakteur. Der Vorspann lautet: "Die Vierte Dimension enthält sehr wertvolles Material und mehr als ein Mitglied (der FIG UK) hat vorgeschlagen, diesen oder jenen Artikel in voller Länge zu übersetzen. Wir haben Freiwillige unter uns, die das können. Machen Sie uns also Ihre Artikelvorschläge. Setzen Sie sich über Ihre Auswahl mit Alan in Verbindung und wir werden in der darauffolgenden Forthwrite eine Übersetzung veröffentlichen." Da bin ich, der Rezensent, mal gespannt. Irgendwie sehe ich da auch für uns in der FG wieder Arbeit auf uns zukommen. Aber freuen würden mich solche Aktivitäten sehr.

34 Did you Know? - large Forth projects (1)
Chris Jakeman <cjakeman@bigfoot.com>

In comp.lang.forth wurde gefragt: Wieso wird das übergroße Forth-Projekt bei Riyadh Airport als Erfolg gewertet, wo es doch mit restlos veralteter Hardware arbeitet? Elizabeth Rather verteidigt sich: Uralt-Hardware schon 1980, ursprünglich teilweise in PLM und FORTRAN programmiert, Hardware inzwischen mehr als veraltet,, alle Jahre wieder taucht ein Projektverantwortlicher auf (ihre Kontrakte werden auf nur zwölf Monate ausgestellt und der Nachfolger weiß nichts vom Vorgänger und von den Vorgängen), der der Meinung ist, Modernisierungen seien nicht möglich, da das Ganze in Forth programmiert ist. Und so bleibt alles beim alten, obwohl Forth sich einer modernisierten Anpassung alles andere als widersetzen würde.

35 Letters

Zwei Briefe an die Redaktion, einer von Federico de Ceballos, einer von Fred Behringer. Federico berichtet über den baldigen Abschluß seiner Doktorarbeit und darüber, daß er in diesem Zusammenhang einen Compiler für ein Forth-Dialekt entwickelt hat, mit dem er vier verschiedene Cross-Compiler erzeugen kann. Fred preist Julian Nobles Artikel aus der Forthwrite 113 über "Assembler in Forth" und macht darauf aufmerksam, daß man die Bit-Ordnungs-Reversion viermal so schnell und gleich 32 Bit breit fassen kann. Fred testet seine Programme unter ZF und Turbo-Forth.

Bericht von der 17. euroFORTHKonferenz (Schloß Dagstuhl, 23. - 25. November 2001)

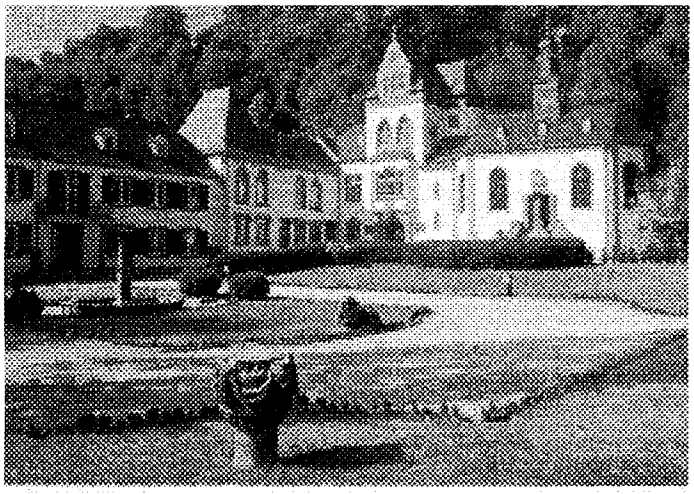
Ulrich Hoffman

Nach Schloß Dagstuhl zu fahren ist für mich immer ein Erlebnis. Nicht nur, daß aus Kiel die Anreise recht beschwerlich und langwierig ist, auch der Aufenthalt im Schloß ist ein Abenteuer.

Es ist groß und kann unübersichtlich sein, sozusagen ein Hogwarts für Forth-Begeisterte. Doch es ist ein herrlicher Platz, bei dem Eigenverantwortung groß geschrieben wird: Keine Türschlösser, selbst notierte Nebenkosten, eine 24 Stunden täglich zugängliche Bibliothek (allerdings ohne verbotene Sektion), ein ungemein freundliches Personal, exzellente Verpflegung; kurz ein idealer Ort in klösterlicher Abgeschiedenheit, den Rest der Welt zu vergessen und sich auf wesentliche Dinge zu konzentrieren.

26 Forth-Enthusiasten aus 9 Ländern (Frankreich, Großbritanien, Irland, Spanien, Österreich, Deutschland, USA, Südafrika, Schweiz) haben denn auch wieder zur internationalen euroFORTH-Konferenz gefunden, die in diesem Jahr zum 17. mal stattfand. Als besonderer Gast konnte Charles Moore begrüßt werden, der über neueste Entwicklungen rund um colorFORTH und seinen neuen Prozessor c18 berichtete.

colorFORTH ist Chuck's persönliche, vollkommen eigenständige PC-Entwicklungsumgebung mit deren Hilfe er seine Ideen realisiert. In diesem Forth-System markieren unterschiedliche



Farben von Worten unterschiedliche Funktionen, seien es zu definierende Worte, Kommentare, Variablen, oder immediate Worte. Mehr Informationen über colorFORTH kann man auf der Webseite http://www.colorforth.com/ finden.

Die Konferenz, die vom 23. bis zum 25. November 2001 zum 3. mal auf Schloß Dagstuhl stattfand, hatte ein reichhaltiges Vortragsprogramm (siehe Kasten "Die euroFORTH-Vorträge"). Natürlich sind die Vorträge nur ein Teil dessen, was auf einer Forth-Konferenz geschieht, die Gespräche mit den andere Teilnehmern bis in den Morgen hinein und das Knüpfen neue Kontakte sind essentiell. Auch die Workshops (diesesmal: Processor Cores, Objects, Scripting, Open Boot, TCP/IP, und colorFORTH), auf denen an Themen orientiert aber dennoch sehr frei diskutiert wird, nehmen einen wichtigen Platz ein.

Ein Höhepunkt der besonderen Art ist der traditionell stattfindende euroFORTH-Wettbewerb, der in diesem Jahr unter dem Motto stand: "Mein dümmster aber elegantester Software Fehler". Der Preis einer guten Flasche Rotwein wurde an Bernd Paysan für einen Fehler verliehen, bei dem ihm die Testabteilung über Wochen Fehlerlisten geschickt hat, auf denen, wie sich schließlich herausstellte, immer nur Ausgaben von Folgefehlern aber nie die der Ursprungsfehler aufgenommen waren.
Wie immer war die Zeit viel zu kurz, um alle neuen Ideen in der nötigen Ausführlichkeit zu diskutieren. Die Aufarbeitung wird sicherlich noch eine lange Zeit in Anspruch nehmen...

Peter Knaggs hat dankenswerter Weise den Tagungsband produziert, der in Kürze auf der euroFORTH-Webseite (http://dec.bournemouth.ac.uk/forth/euro/) in elektronischer Form erhältlich sein wird; Bill Stoddard übernahm die Konferenzorganisation.

Alles in Allem war die euroFORTH 2001 ein voller Erfolg. Ich freue mich schon auf die nächste Konferenz 2002, die vermutlich in Wien stattfinden wird. Wer auf dem Laufenden bleiben will oder mit dem Gedanken spielt auch mal auf einer euroFORTH-Konferenz dabei zu sein, sollte sich auf den euroFORTH-email-Verteiler setzen lassen, auf dem die Konferenz-Ankündigungen verschickt werden
(http://dec.bournemouth.ac.uk/forth/euro/).

Forthige Grüße,
Ulrich Hoffmann

Vortragstitel	Autor
„An HTTP Server in Forth“	Bernd Paysan
„Forth vs the Beast: Taming an Agressive Watchdog“	Malcom Bugler
„A Windows Driver Program Written in Forth“	Nick Nelson
„CANed Objects“	Alan Robertson
„The C18 ColorForth Compiler“	Chuck Moore
„Threaded Code Variation and Optimziation“	Anton Ertl
„Top Heavy Trees“	Hugh Aguilar
„The Common Case in Forth Programs“	Dave Greg, Anton Ertl, John Waldron
„Using Communicating State Machines to Design an Interrupt Driver Task Scheduler“	Bill Stoddart
„Treating Data as Source“	Jenny Brian
„OO Package for Embedded Control“	Daniel Ciesinger
„Joy: Forths Functional Cusin“	Reuben Thomas, Manfred von Thun
„colorForth and the Art of the Impossible“	Howerd Oakford
„25x Emulator“,	Chuck Moore
„The Mite VM: bridging the complexity gulf“	Reuben Thomas
„A Minimal Development Environment for the AVR processor“	Federico de Ceballos
„Micro Core, and open source, scalable VHDL synthesisable dual stack Harvard processor for FPGAs“	Klaus Schleisiek

Ein Interview mit Tom Zimmer : Forth System Entwickler

Forthwrite; November 2001, Ausgabe 114, Seite 12

Jim Lawless, Copyright 2000

Wenn Sie jemals mit einem Forth Compiler gearbeitet haben, bestehen gute Chancen, daß Ihnen der Name „Tom Zimmer" zu Ohren gekommen ist. Tom ist seit mehreren Jahrzehnten einer der Dreh- und Angelpunkte in der Forth-Gemeinde. Tom hat mehrere Forth-Systeme für 8-Bit Mikrocomputer entwickelt, die in den 80ern den Markt der Heim-Computer dominiert haben. Tom ist der Schöpfer des Freeware Systems Win32Forth.

Was ist ihr ausbildungsmäßiger Hintergrund?
Ich habe keine formale Programmierausbildung erhalten. Ich habe die Highschool 1968 abgeschlossen, vor langer Zeit und weit entfernt von hier. Damals war ich einfach an Elektronik interessiert, und ich hatte einen Freund mit Namen Dick Cappels, der mir die Einzelteile eines Computers verkaufte und mir riet, hinunter nach Wiley Elmar in Sunnyvale in Kalifornien zu gehen – und meinen neuen Computer mitzunehmen.

Der (Compter) hatte eine RCA CDP-1802 CPU, einen statischen Prozessor. Das war für die damalige Zeit ein ziemlich seltsames Ding. Die meisten Prozessoren dieser Zeit waren dynamisch und ließen sich nicht unterhalb von 500 kHz betreiben. Der 1802 ließ sich mit nahezu jedem gewünschten Takt betreiben, bis 0 Hz hinunter. Ich hatte den Prozessor mit 1 k statischem Memory zu einem einfachen Computer verdrahtet und programmierte diesen in Maschinensprache. Mir standen drei Taktraten zur Verfügung: Einzelschritt, 10 Hz und ungefähr 500 kHz. Das war mein erster Kontakt mit Computern.

Nach der Highschool arbeitete ich für Pacific Telephone als COEM (central office equipment man – *Hauptstellen-Ausrüstungs-Mann*). Das war noch in den Tagen, als die Namen von Jobs auf das Geschlecht des Mitarbeiters rückschließen ließen.

Nach einem kurzen Aufenthalt beim Militär, als Kommunikations-Kontroller, heuerte mich der schon angesprochene Freund als Elektronik-Techniker für eine kleine Firma an, die die ersten Video Disk Rekorder herstellten. Diese Geräte entsprachen nichts von dem, was Sie sich heute unter dieser Bezeichnung vorstellen könnten. Sie waren viel größer und hatten viele mechanische Teile, die je nach Kundenwunsch variieren konnten.

Die Video Rekorder enthielten einen Mikrokontroller, der von Mike O'Malley in Berkeley mit Forth programmiert wurde. Er hat diese Arbeit auf eine Art beratender Basis gemacht. Er brachte uns ein EPROM, wir bauten das in den Rekorder, und der lief. Wir waren jedes Mal verblüfft wenn sein Code funktionierte, weil er nicht die geringste Hardware besaß, auf der er den Code hätte entwickeln können. Er erklärte uns, daß er irgendeine Art von Simulator besäße, den er zum Testen nutzen würde.

Später wollte Dick von mir, daß ich einen Hardware-Kontroller für den Video Disk Rekorder entwerfe, der nicht auf (eigenen) Prozessoren basierte. Mike trieb uns nämlich mit ein oder zwei Dollars vor sich her, die er für ein Byte fertigen Codes haben wollte, und wir hielten das für teuer. Darum entwarf ich diesen Kontroller. Das war mein erstes großes Hardwareprojekt.

Ich besaß keine formelle Ausbildung in Hardware und deren Design, es sei denn, sie zählen einen Elektronikkurs in der Highschool dazu. Aber der Kontroller lief und wurde Bestandteil unseres Produktes. Allerdings war er nicht annähernd so fehlerfrei wie Mike's forth-kodierte Kontrollerversion. Darum gaben wir die Idee, den Rekorder ausschließlich durch Hardware steuern zu wollen, gleich wieder auf.

Jedenfalls standen meine Ansichten zur Elektronik und zu Computern seit diesem Erlebnis ein für alle Male fest. Ich habe sie seither nie revidiert.

Wie kamen Sie zum ersten Mal mit Forth in Berührung?
Meinen ersten Kontakt mit Forth habe ich gerade beschrieben. Aber das erste Mal, das ich selbst versuchte, Forth zu nutzen, war viel später, als ich für Calma arbeitete. Die haben CAD Workstations gebaut, und ich wurde eingestellt, um in der Hardwarediagnose zu arbeiten. Ich bekam eine kaum lesbare Photokopie eines Listings von Forth für den 8080 Prozessor. Ich habe das Listing in ein Intel MDS (Micro Controller Development System) getippt, assembliert und ans Laufen bekommen.

Ich hatte keine Ahnung, was Forth sein sollte, aber ich hatte gehört, daß es gut im interaktiven Debugging war. Und ich war interessiert.

Das 8080 Forth hatte mir eigentlich schon das Konzept des virtuellen Memories in die Hände gegeben, aber ich selbst war damals so weit von solchen Dingen entfernt, daß ich dies alles mit Stumpf und Stiel herausriß. Zu dieser Zeit, in den späten 1970ern, hatte ich von der Forth Interest Group (FIG) noch

nichts gehört, so daß ich weder zu dieser Gruppe noch zu irgend Jemandem anderes in der Forthgemeinde Kontakt hatte. Ich habe damals einfach das interessante Konzept einer interaktiven Programmiersprache erforscht.

Gegen Ende meiner Zeit bei Calma bekam ich ein FIG Listing von Forth für die VAX, und brachte auch dieses Forth ans Laufen. Wir nutzen Forth, um Hardwarediagnosen zu schreiben. VAX-Forth war aber eine ziemliche Herausforderung, weil ich es zwar assemblieren konnte, aber nicht in das VMS, das Betriebssystem der VAX zu integrieren verstand. Ich mußte in einigen Systemdateien die Orte von Systemaufrufen ausgraben, mit deren Hilfe ich eine Schnittstelle zum VMS schaffen konnte.

Wie ich in comp.lang.forth erfahren konnte, haben Sie Forth-Software für eine Reihe von Mikrocomputern in den 80ern entwickelt oder zumindest mit entwickelt. Durch welche Ereignisse wurden Sie jeweils in die Entwicklung dieser Produkte einbezogen ?
Forth hatte mich sehr aufgeregt, als ich bei Calma Erfahrungen damit sammeln konnte. Ich besorgte mir einen Ohio Scientific Computer, der auf einem 6502 basierte. Ich nahm das 8085 Forth, das ich bei Calma entwickelt hatte und übersetzte es in Handarbeit in 6502er Maschinencode um Forth auf meinem Ohio Scientific laufen lassen zu können. Ich war damals noch ziemlich jung und ich frage mich noch heute, warum meine Frau mir all die Zeit zugestand, die ich in meinem Arbeitszimmer verbracht habe. Aber ich denke, daß ich damals so aufgeregt über Forth war, daß ihr gar keine Chance blieb, mich wirklich einzufangen.

Um 1979 herum hörte ich von der FIG, und Robert Reiling überließ mir ein FIG Listing für den 6502. Das sah sehr interessant aus und schien mir von sehr viel mehr Menschen akzeptiert zu sein, als mein eigenes Forth je schaffen könnte. Darum arbeiteten Bob und ich daran, das FIG Forth auf den Ohio Scientific zu portieren. Ich meine, Bob hat es einfach eingetipt und mich dann losgelassen, damit ich es auf der Hardware ans Laufen brachte.

So wechselte ich 1979 von meinem eigenen Forth zu FIG – und kam weiter voran. Weil viele Hersteller in diesen Tagen Personal Computer anboten, kaufte ich einen, vergrub mich darin und entwickelte ein Forth dafür. Das war einfach eine Möglichkeit Spaß zu haben und gleichzeitig ein wenig Geld zu verdienen.

Das nächste PC Forth an dem ich arbeitete, war VIC-Forth für den Commodore VIC-20. Ich erinnere mich nicht mehr, was dann als nächstes kam; 64Forth für den Commodore 64 oder ColorForth für den Radio Shack Color Computer. VIC-Forth war eine 8k-Speicherkartridge, Color Forth gab es auf einer 12k Kartridge und 64Forth auf einer 16k Kartridge. Jedes erfolgreiche System hatte eine größere Speicherkapazität.

Warum haben Sie die Implementierung jeweils als Kartridge ausgeführt ?
Diese Computer hatten keine Diskettenlaufwerke. Die einzige wirkliche Alternative hierzu waren Kassetten. Ich mußte Kassetten während der Entwicklung nutzen, aber ich wollte ein Forth bauen das einfach zu lernen und zu nutzen war. Der Nutzer sollte, außer zum Abrufen und Speichern von Daten, nicht mit Kassetten umgehen müssen. Später, beim 64Forth, kamen sicherheitliche Überlegungen hinzu. Es gab Anbieter, die Kartridge-Ripper vertrieben. Darum bekam 64Forth einen begrenzten Kopierschutz, der das Auslagern ins RAM ausschloß. Das Forth mußte im ROM verbleiben - anderenfalls überschrieb es sich selbst. Grausam, aber das war in den Tagen bevor ich dazu überging, ausschließlich Public Domain Systeme zu entwickeln.

Color Forth lief auf einem 8609 Prozessor und basierte auf einem Forth das ich aus dem einzigen geschützten FIG Listing entwickelt hatte. Es kam von einem Anbieter in Süd-Kalifornien. Ich erinnere mich aber nicht an seinen Namen. Wie auch immer; ich schloß einen Vertrag mit ihm, um die Honorare für die unterschiedlichen Verfasserrechte aufzuteilen – und das Forth wurde vertrieben.

64Forth war damals das profitabelste Forth. Es wurde von HES (Human Engineered Software) in Burlingame, Kalifornien vertrieben. Ich selbst habe rund 25.000,- Dollar an den Rechten für 64Forth verdient, bevor HES finanziell kollabierte. HES schuldet mir bis heute 9.000,- Dollar. Das hat mich aber niemals wirklich berührt. Ich war eigentlich hoch erfreut, daß 64Forth sich so gut verkaufen ließ.

Ich glaube, daß die damals eine Menge Zeug zusammen hatten, das während einiger Jahre zwischen verschiedenen Forth Anbietern herumgereicht wurde, bis endlich das Interesse erlahmte.

Jedenfalls hatte jedes meiner Produkte ein ziemlich brauchbares Manual, das ich geschrieben hatte, und HES investierte eine nicht unbedeutende Summe in die Verpackung von 64Forth und VicForth, damit diese auch attraktiv waren. Ich bin sicher, daß dies deutlich zur Popularität dieser Produkte beigetragen hatte.

Wie sind Sie bei der Publikation und der Vermarktung vorgegangen ?
Hatten Sie damals Kontakte zur Industrie ?
Ich hatte keinerlei Kontakte, aber in diesen Tagen war so viel weniger Software verfügbar als heute, daß ich einfach einen Software Verleger kontaktieren und fragen konnte, ob er meine Produkte gemeinsam mit seiner Produktlinie anbieten wollte. Der Hunger nach Software war riesig. HES war ein echtes Entwicklungsunternehmen das damals auch in Verpackung und Werbung investierte. Sie hatten auch Kontakt zu Kartridge Herstellern die „Chip on Board" realisieren konnten, wodurch die Notwendigkeit entfiel, ROM mitzuliefern. Das hat natürlich die Produktionskosten gesenkt.

HES hat ein sehr schönes Gesamtpaket für beide, VicForth und 64Forth, hergestellt. Ich bin sicher, daß die Verpackung alleine für einige Umsätze gesorgt hat.

Die Assemblersprachen der verschiedenen Prozessoren dieser Zeit (6502,6809 usw.) haben Sie allesamt gemeistert ?
Assemblersprache ist Assemblersprache ist Assemblersprache. Wenn Sie eine gesehen haben, haben Sie alle gesehen, mit der einen möglichen Ausnahme 1802, die sich sehr von allen anderen Assemblersprachen unterscheidet. Ich habe Assemblerprogrammierung im Vorbeigehen gelernt. Ich habe einfach ein neues Buch gekauft und die darin enthaltenen Instruktionen im Geiste in diejenigen übersetzt, die ich bereits kannte.

Später, bei Maxtor, war ich als Diagnostik-Programmierer angestellt, um deren Diskettenlaufwerke zu testen.

Wir haben dort 8086er benutzt und mit Laxen und Perry Forth angefangen. Wir haben eigene Forthe entwickelt um damit Diskettenlaufwerke mit hoher Datendichte, die Maxtor damals produzierte, zu testen.

Forth-basierte Software wurde dort in einem eigenen Netzwerk genutzt, um Diskettenlaufwerke einem 48-stündigen „Einbrennen" zu unterziehen und die Ergebnisse auszudrucken.

Ich habe dort rund drei Jahre gearbeitet und verschiedene Public-Domain Forthe entwickelt, wie Zforth, Tforth, Hforth, HF, ZF und F-PC.

Haben Sie jemals andere kommerzielle Systeme entwickelt als Forth-Compiler?
Gute Frage. Eine zeitlang schien es so, als sei alles worin ich gut war, Forth Systeme zu bauen, und nicht Applikationen zu schreiben. Ich denke mir, Forth war eine Applikation. Über die Jahre hinweg habe ich an verschiedenen Applikationen gearbeitet, aber alle scheinen darauf basiert zu haben, daß zunächst ein Forth System geschrieben werden mußte. Ich weiß, daß viele Menschen mit der dahinter verborgenen Philosophie nicht einverstanden sind. Aber zu dieser Zeit hatte ich das Gefühl, daß es absolut notwendig sei, die Kontrolle über das jeweilige Entwicklungssystem zu behalten.

Heute, wo Visual C++ dermaßen überlegen ist, können wir allerdings auf Microsoft vertrauen, wenn sie uns ein Entwicklungssystem anbieten (Ich albere gerade ein wenig herum).

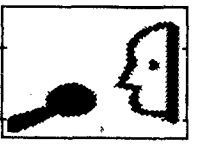
Das ist eine interessante Aussage. Denken Sie, daß jüngeren Programierern etwas fehlen wird, wenn Sie in ihrer Ausbildung nicht mit Forth konfrontiert werden ?
Absolut. Die meisten Leute sind nicht vertraut mit Forth, halten es bestenfalls für eine fast vergessene Sprache der Vergangenheit. Das Gleiche ließe sich aber auch von unserem (kulturellen) Erbe sagen, unabhängig davon, in welchem Land wir geboren wurden.

Geschichte ist aus verschiedenen Gründen wichtig, nicht zuletzt deshalb, weil sie uns zeigt, wie wir mit der Zukunft umgehen sollten.

Forth's wichtigstes ‚Feature' hat wenig damit zu tun, das es eine stapelorientierte Sprache ist, vielmehr ist dies die Art und Weise wie es mit dem Nutzer als Ganzes interagiert. Forth's Erweiterbarkeit, Struktur, Modularität und seine sehr einfache Syntax sind die Schlüsselattribute, die dem Prorammierer die Freiheit geben, Problemlösungen auf eine Art und Weise zu strukturieren, die Programmierern anderer Sprachen vorenthalten bleibt und oft gar nicht verstanden wird.

Zugriff auf alle Quellen des Entwicklungssystems zu haben gibt Ihnen die Freiheit, das System zu erweitern, oder Probleme zu lösen, die der Kunde gar nicht gesehen hat. Freiheit ist mir wichtig, so wie sie es Jedermann sein sollte. Sie müssen bedenken, daß Freiheit immer auch Verantwortung mit sich bringt. Forth gibt Ihnen Freiheit und die Macht, jedes Problem einer passenden Lösung zuzuführen. Es gibt Ihnen auch die Macht, sich selbst in den Fuß zu schießen, oder in einen wesentlich sensibleren Bereich. Wenn Sie also mit der Freiheit und der Macht nicht umgehen können, dann sollten Sie Forth besser fern bleiben.

Arbeiten Sie aktuell an einer Software ?
Wenn, an welcher
Ja, ich arbeite für ThermoQuest, als Programmierer. Andrew McKewan hat mich angestellt, damit ich dabei helfe, eine sehr



große DOS basierte Forth Applikation nach Windows NT zu portieren. Wir haben uns das Problem angeschen und einfach das einzige kommerzielle Forth gekauft, daß zu dieser Zeit für Windows NT zur Verfügung stand. Unglücklicherweise war das zu dieser Zeit nicht besonders ausgereift. Wir hatten keinen Zugriff auf den Quellcode des Kernels, so daß wir immer wieder in Bugs hineinliefen oder auf philosophische Ungereimtheiten stießen.

Andrew hat dann seinen eigenen 32 Bit Forth Kernel an einem einzigen Wochenende implementiert. Wir brachten den Kernel ans Laufen und nutzen dabei den kommerziellen Assembler unseres Auftraggebers, für den wir eine Lizenz hatten. Wir haben allerdings nie dessen Quellen genutzt. Ich denke aber, daß wir einige Ideen unseres Auftraggebers verwendet haben.

Wie auch immer, Andrew brachte den Kernel ans Laufen und übergab ihn an mich; zur „Erweiterung".

Ich gab den Kernel in die „Public-Domain", wo ich ihn während der ganzen Entwicklungszeit nie heraus nahm. Ich war immer sehr sorgfältig dabei, aus den generellen Quellcodes des PD Forth-Systems alle Codes herauszuziehen, die ausschließlich meinem Arbeitgeber gehören.

Ein Beispiel hierfür ist, das Win32Forth ein 32-Bit Forth ist und uns mit der Frage konfrontierte, ob wir die 16-Bit Quellen der Applikation unseres Auftraggebers nach 32-Bit portieren sollten oder nicht. Weil aber die Applikation mehrere Megabyte groß war und wir mit unserer eigenen Arbeit zuverlässig bleiben wollten, entschlossen wir uns, die Applikation 16-bittig zu lassen und ein 16 zu 32-Bit Übersetzungsmodul zwischen unser Forth und der Applikation zu setzen. Das hielt die Probleme, denen wir uns gegenüber sahen, im Bereich von Kompatibilitätsfragen und ermöglichte uns eine einfache Portierung. Wir haben dann der Applikation noch ein Windows GUI spendiert, damit sie im Windows-Markt Akzeptanz finden konnte.

Die gesamte Portierung hat vom Start bis zur aktuellen Produktversion rund 9 Monate gedauert, wobei im Mittel vier Programmierer gleichzeitig beschäftigt waren. Das ist sicher eine große Aufgabe, aber die Applikation hat beim Kunden längst ihre Zuverlässigkeit bewiesen.

Eine interessante Geschichte am Rande ist folgendes: Das Übersetzungsmodul zwischen 16 und 32-Bit enthielt eine große Menge an Debugging-Codes für Bereichsprüfungen bei Speicheroperationen. Als wir das Produkt herausgaben, ließen wir den Debugging-Code aktiv. Wir waren uns nicht ausreichend sicher, alle Fehler gefunden und ausgemerzt zu haben. Dann, ein Jahr später, als wir die nächste Version der Applikation auslieferten, entfernten wir die Bereichsprüfungen. Plötzlich war die Applikation aufsehenerregend schneller und trotzdem genauso zuverlässig wie vorher, weil wir während des vergangenen Jahres die allermeisten Probleme aufarbeiten konnten. Die Marketingabteilung unseres Kunden hat dies aber als neues ‚Feature' verkauft: „viel schneller".

Andrew McKewan, Robert Smith und ich waren die ersten, die etwas zum Win32Forth beigetragen haben, gefolgt von Y.T.Lin und Andy Corsack. Später sprach ich mit Jim Schneider darüber, einen vollständigen 486 Assembler zu schreiben. Was er dann auch tat, und womit er das System vervollständigte. Andrew steuerte noch objektorientertes Programmieren bei, ziemlich früh sogar, modeliert nach dem MOPS OOP Forth System für den Macintosh. OOP war sehr wertvoll wenn es darum ging, die Komplexität der Windows API zuverlässig zu handhaben. Im Laufe der Jahre haben verschiedene Leute Fehlermeldungen und Lösungsvorschläge beigesteuert und dem Win32Forth viele Erweiterungen geschenkt.

Das Forth war auch für ein Jahr an einen kommerziellen Anbieter verkauft, erwies sich aber als zu komplex für dessen Zwecke.

Heute programmiere ich überwiegend in Visual C++. Eigentlich habe ich ‚C' immer gehaßt, aber nach fünf Jahren ist es erträglich. Wenn ich in ‚C' programmiere, vermisse ich die besondere Macht von Forth, für schwierige Probleme Kompilezeit-Lösungen erzeugen zu können.

Ich denke, ich bin für die Entwicklung von Forth Systemen ausgebrannt. Aber wer weiß schon, was die Zukunft bringt. Wenn ein neuer interessanter Computer oder ein neues Betriebssystem auftauchen, springe ich vielleicht auf diesen Zug und tauche wieder in ein Forth-System-Entwicklungs-Projekt ein.

Was ist mit BeOS ?

Ich habe erst kürzlich eine Anfrage nach Forth für BeOS in comp.lang.forth gelesen.

Ich bin ein Anwalt von Macintosh, und BeOS hat mich interessiert als es schien, daß es auf dem Mac laufen würde. Aber jetzt passiert genau das nicht. Deshalb habe ich mir das System schon lange nicht mehr angesehen.

Interview: Tom Zimmer

Hatten Sie jemals Spaß an der Idee, einen Forth-Compiler für eine Spielekonsole zu schreiben ?
Nein. Aber ich könnte mich dafür interessieren, ein Forth für eines der Geräte in der Art der PDA zu schreiben. Ich meine, daß es bereits Forthe für den Palm gibt. Ich denke, daß dieser Markt gerade erst öffnet und daß noch interessante Geräte kommen werden; vielleicht dann.

Haben Sie darüber nachgedacht, Win32Forth jetzt zu verkaufen ?
Ich habe darüber nachgedacht. Aber meine Erfahrungen sagen mir, daß es sehr schwer ist, Entwicklungssysteme zu verkaufen. Win32Forth ist Public Domain, so daß andere Menschen Vorteile daraus ziehen können. Aber ich habe ebenso Vorteile an dem, was andere Menschen beisteuern.

Ich bevorzuge Public Domain, über GPL, weil es die wenigsten Restriktionen an die Nutzung der Software knüpft.

Natürlich könnte Irgendjemand Win32Forth nehmen und ein kommerzielles System daraus machen, oder ein kommerzielles Programm damit schreiben ohne mir und den anderen Schöpfern des Systems den entsprechenden Obulus zu geben. Aber ich bin ebenso frei darin, den Code den andere zu unserem System dazu gegeben haben, selbst in einem kommerziellen Programm zu nutzen. Ich versuche selbst stets dort zu bezahlen, wo Bezahlung angemessen ist. Was micht antreibt ist aber der Wunsch, die Probleme in Applikationen lösen zu können, nicht Geld zu bekommen für irgendwelche Codesegmente die ich schon vor Jahren geschrieben habe.

Interessanter Weise wurde Win32Forth vor einigen Jahren von einem kommerziellen Anbieter gegen Umsatzbeteiligung gekauft. Der wollte das System dokumentieren und als kommerzielles Produkt vermarkten. Das Problem war, daß Win32Forth so groß ist, daß es nicht wirklich in die Philosophie der Entwicklungstools dieses Herstellers paßt. Das System kam wieder zu mir zurück.

Wie viele Kopien jeder Ihrer kommerziellen Compiler wurden verkauft ?
Ich kann das gar nicht genau sagen, aber meine Nachforschungen sagen mir, daß mehr als 10.000 Kopien von 64Forth produziert wurden. Ich bekam meinen Anteil für rund 7.000 davon, bevor HES in den Konkurs ging. So ungefähr 3.000 bis 4.000 Kopien von VIC-Forth wurden verkauft und sehr viel weniger von Color-Forth und OSI- Forth.

Was brachte Sie dazu, ein DOS Forth zu entwickeln, mit einer IDE die anderen Compilern dieser Zeit ähnlicher war als einem traditionellen Forth ?
Ich nehme an, Sie sprechen jetzt über TCOM, weil dies das einzige Forth ist, das eine wirkliche IDE hat. TCOM wurde entwickelt um das Schreiben von DOS Applikationen zu erleichtern. Eines der Probleme aller meiner Forth-Systeme war deren Größe. Sie waren immer groß und fett, mit einer Menge von Tools und Bibliotheken von Werkzeugen. All dies Zeug führte zu großen Executables.

TCOM wurde von Anfang an so konzipiert, daß er nur diejenigen Teile der Sprache in die Applikation integriert, die notwendig sind, um die Applikation aufzubauen. Das Ergebnis waren sehr kleine Executables.

Natürlich wollen Sie trotzdem Ihre Programme debuggen. Also mußte ein Debugger her. Weil TCOM *.COM Executables produzierte, die keinerlei Debug Informationen enthielten, und weil ich den Zielcode nicht mit dem geringsten Ballast belasten wollte, nahm ich den Weg, zusätzliche Data Files erzeugen zu lassen, denen der Debugger entnehmen konnte, welche Zeile des Quellcodes zu welcher Stelle des Zielcodes gehörte. Das ermöglichte es mir, Listings der gleichen Art wie Standard Assembler aus TCOM Executables zu erzeugen und diese symbolisch zu debuggen. Das hat sehr gut funktioniert.

TCOM enthielt meist ein ganzes Bündel von Zielprozessoren, mindestens; 8086, 8096, 8080, 68hc11, 6805 und den Samsung Super8, 56000, und 57000 Prozessoren. Das Paket enthielt auch ein Bündel von Beispielapplikationen für den 8086, mehr als 70 Stück, denke ich.

Ich habe auch einen einfachen BASIC Compiler für den 8086 als Target für TCOM geschrieben.

TCOM enthielt alle Quellen für den gesamten Compiler, alle Beispiele, den Debugger und alle Listings für alle Generatoren der verfügbaren Targets.

TCOM habe ich mit F-PC entwickelt.

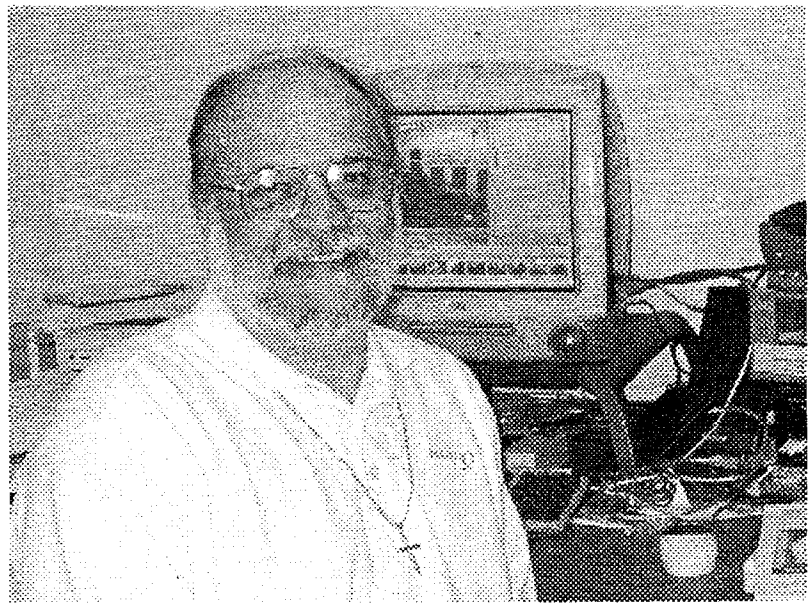
Haben Sie in den 80ern an industriellen Messen teilgenommen ?
Oh ja. Aber nur an solchen, die mit Forth in direkter Beziehung standen.

In den 80ern gab es eine Menge unterschiedlicher Aktivitäten in Forth. Da gab es verschiedene Hardware Anbieter und mehrere Software Häuser, die sich mit Forth befaßten. Jetzt ist

alles etwas ruhiger geworden. Und ich denke, daß Forth in den Untergrund gegangen ist. Es könnte natürlich niemals ein genereller Ersatz für Visual C sein, aber es ist natürlich wunderbar geeignet in Umgebungen mit begrenzten Resourcen.

Wenn wir heute immer schnellere und schnellere Computer sehen, bald mit Gigabyte an RAM und Terabyte an Festplattenspeicher, drängt sich der Gedanke auf, daß es Umgebungen mit begrenzten Resourcen nicht mehr lange geben wird. Aber im Bereich der Konsumentenprodukte und ganz besonders in beliebigen Bereichen des [elektronischen] Massenmarktes ist Forth eine praktische Alternative. Es erlaubt schnelle Entwicklungen und Tests bei geringen Kosten. Ich denke, es wird immer die Geheimwaffe kleiner Entwickler sein, die in die Märkte der großen Entwickler mit hunderten von Programmierern einbrechen.

Wie paßt Forth in Ihre Zukunft?
Nun, ich beschreibe mich selbst als C-Programmierer, der eigentlich ein Forth-Programmierer ist. C hat mir Beschäftigung gegeben, und Forth gibt mir Tools für Hard- und Software Debugging.



Wenn ich mit anderen C-Programmierern an einem großen Projekt zusammenarbeite, dann baue ich immer einen Forth Interpreter in die Applikation, zu Debuggingzwecken. Die Hardwarejungs lieben das, weil es ihnen die Macht gibt, wirklich herauszufinden, was mit der Hardware passiert. Für das Austesten der Software ist dieser Interpreter großartig, weil er eine interaktive Möglichkeit zur Kommunikation mit der Hardware bietet. Sie können herausfinden, wie man am besten mit der Hardware kommuniziert; bevor Sie einen Treiber in C schreiben.

Ich denke, die meisten C-Programmierer schauen auf Forth und verstehen einfach nicht, warum sie daran interessiert sein sollten. Und es fällt ihnen gar nicht ein, ein wenig Zeit darin zu investieren, das herauszufinden.

Ich denke an Forth immer als an einen Satz feiner Handwerkzeuge. Microsoft, als Gegenpart, bietet das ultimative Powerwerkzeug an, Visual C++ mit der MFC. Das ist die computergesteuerte (Kaffee)Mühle, die zu betreiben drei Doktorgrade erfordert. Dann können Sie ihren Job ziemlich schnell erledigen, aber sie werden es hassen ihn zu tun, weil das Werkzeug so ein Monster ist und keinerlei Fehler verzeiht.

MFC versteckt Informationen auf wunderbare Weise, damit Sie sich auf die Lösung allgemeiner Probleme konzentrieren können, aber unglücklicherweise benötigen Sie gerade eine Menge der Informationen, die vor Ihnen versteckt werden, weil MFC sonst in vielen Situationen nicht richtig arbeitet. Es ist wie ein auf Sand gebautes Haus.

Forth ist auf der anderen Seite wie die Gründung auf Fels, auf der Sie nun ihr Haus bauen können. Es ist einfach zu verstehen und völlig fehlerfrei. Natürlich ist Win32Forth in die Microsoftfalle getrampelt. Bei dem Versuch die gesamte Komplexität von Windows abzubilden, ist das System selbst ungeheuer komplex geworden, obwohl es eigentlich ein recht simples Forth sein könnte. Das ganze OOP wurde dem System hinzugegeben, um der Komplexität zu begegnen. Das half auch, aber diese Hilfe gab es nicht umsonst. Manchmal denke ich, daß der Preis für die wachsende Komplexität einfach zu hoch ist.

Nun, ich nehme an, es wäre jetzt besser, wenn ich die Schmollecke verlassen und zur Programmierung in Visual C++, MFC und zu meinem jüngsten Projekt in Java zurückkehren würde, zu einem völlig neuen Abenteuer.

Danke für das Interview, Tom!
Das Interview hat Jim Lawless geführt und auf seiner Homepage unter http://www.radiks.net/~jimbo veröffentlicht.

Die Übersetzung stammt von Friederich Prinz.

Die Veröffentlichung der Übersetzung geschieht mit freundlicher Genehmigung von Jim Lawless.

From the Big Teich ...

Henry Vinerts

Diesen Bericht von Henry Vinerts aus dem Silicon Valley hat – wie stets – Thomas Beierlein für uns übersetzt. Danke!

Lieber Fred und Martin,
es ist nun wieder fast eine Woche seit unserem letzten SVFIG Treffen und ich muß feststellen: Je länger ich warte, um so fauler werde ich den nächsten Bericht zu schreiben. Daher laßt mich einen schnellen zu Euch "rüberwerfen".

Aufgrund einer Änderung hatten wir drei Redner, aber wie üblich füllte Ting den größten Teil der Zeit. Die Gruppe wuchs von circa 14 am Morgen bis über 20 am Nachmittag. Außer einigen Meinungen über Windows XP und Microsoft im allgemeinen, verweilten wir nicht beim gegenwärtigen Terrorismus-Thema.

Dr. Ting begann mit einem Aufruf eine Win32Forth Arbeitsgruppe zu bilden, um die nächste Release abzudecken, bessere Dokumentation hinzuzufügen, Zugriff auf Geräte usw. Es scheint daß Tom Zimmer sich von der Verantwortung für Win32Forth zurückziehen möchte. Wie ich zuvor schon anmerkte hat Tom Kalifornien vor einigen Jahren schon verlassen und ist nach Texas gegangen. John Peters war per Email in Kontakt mit Tom und er zählte einige Ideen auf, die eine Arbeitsgruppe implementieren könnte um Win32Forth auf dem Stand der Zeit zu halten.

Ting gestand, daß die Welt ihn schließlich und endlich von FPC und eForth zur Windows Platform treibt, speziell in seinen letzten Arbeiten in Taiwan, bei denen er die verschiedenen Wege untersuchte Forth-Eingaben mit chinesischen Zeichen zu realisieren. Er entwickelt auch Programme um taiwanesischen Schulkindern Forth zu lehren. Er brauchte dazu Hilfe um Sound zu diesen Programmen hinzuzufügen und fand sie bei Doug Dillon. Dieser hatte sich vorbereitet, um uns eine Vorlesung über den Zugriff auf DLL's zur Soundkarten-Ansteuerung sowohl mit Win32Forth, als auch mit Forth Inc.'s Swift Forth zu halten.
Es war genug Zeit vor der langen Mittagspause übrig, so daß Ting uns eine sehr interessante Beschreibung des chinesischen Mondkalenders geben konnte, der ununterbrochen und fehlerfrei seit über 4000 Jahren läuft. Dagegen ist unser Gregorianischer Kalender noch ein Baby. Natürlich hat Ting einen Weg (mit Win32Forth) ausgearbeitet, um letzteren in den chinesischen Kalender umzurechnen, wobei er besondere Aufmerksamkeit der Bestimmung des korrekten Datums für das chinesische Neujahr widmete. Nebenbei erwähnte er, daß circa 30 Leute beim letzten Treffen der taiwanesischen Forth-Gruppe anwesend waren.
Ich habe das deutliche Gefühl, daß Dr. Ting niemals schläft. Es ist erstaunlich, wie viel er für Forth produziert hat und wie verschieden seine Interessen sind. Er beschloß den Tag mit einem anderen Beispiel über eine Sache, die kürzlich sein Interesse erweckt hat und die er für Wert hielt zu studieren und uns darüber zu berichten. Es war ein anderes Forth-System, welches von einem Studenten in Australien kreiert wurde. Es kann von http://pringle.sphosting.com heruntergeladen werden und das ist alles, was ich darüber sagen möchte. Es scheint mir immer noch, außer für Chuck Moore, daß jeder Programmierer seiner eigenen cleveren und einzigartigen Forth-Version in relativer Unbekanntheit bleibt und damit zufrieden ist, sich selbst zu loben. Aber ist es nicht wundervoll mit Enthusiasmus zu arbeiten, solange die, für die wir verantwortlich sind, nicht hungernd und frierend herumlaufen?

Es ist ziemlich wahrscheinlich, daß ich das November-Treffen verpassen werde, aber, so Gott will, sollte ich zurück sein, um euch allen ein Frohes Weihnachten im Dezember zu wünschen.
Mit besten Wünschen, Henry

Warum wird die Bedeutung von OO überschätzt ?

Andreas Klimas

Wie schon im letzten Heft versprochen, schreibt Andreas Klimas hier über seine Gadanken zur OOP. In lockerer Reihenfolge werden diesem Artikel einige weiter folgen.

MB

Zunächst sollten wir uns über die drei wichtigsten Säulen der Objektorientierung bewusst werden.

1. Kapselung

2. Polymorphismus

3. Vererbung

Kapselung ist in Forth ein Kinderspiel. Nachdem dies sowieso eines der Grundkonzepte ist möchte ich gar nicht weiter darauf eingehen.

Polymorphismus ist da schon interessanter. Wir verstehen darunter, dass ein Client eine Funktion aufruft, ohne zu wissen, welcher Datentyp sich konkret dahinter verbirgt. Auch das ist mittels Vektoren in Forth bereits sehr gut gelöst.

Vererbung ist die einzige wirkliche Errungenschaft der Objektorientierung. Hier geht es im Wesentlichen darum, eine bestehende Funktionalität punktuell zu erweitern oder zu verändern, ohne die bestehende Funktionalität angreifen zu müssen. Aber was heißt das nun ? Definieren wir als Beispiel eine Klasse 'Button' und bringen wir ihr bei, dass dieser, wenn er geklickt wird, eingedrückt am Bildschirm zu erscheinen hat. Was aber, wenn ich einen speziellen Button brauche, der sich nicht nur beim Klicken eingedrückt präsentiert, sondern auch noch eine Aktion auslösen soll. In der OO Welt würde man einfach eine neue Klasse TuWas erfinden und diese von Button ableiten (die Funktionalität vom Button erben). Die neue Klasse TuWas würde nur den Spezialfall 'klicken' implementieren, nämlich, dass sie das Klicken vom Button (also der Superklasse) und danach meine spezielle Aktion ausführt.

Das hört sich ganz toll und verlockend an, und das ist es auch. Was aber hat das mit dem Titel 'Warum wird die Bedeutung von OO überschätzt' zu tun ? Ganz einfach, das Problem ist die Evolutionsdauer einer Klassenbibliothek. Wenn wir eine Anwendung entwickeln, habe wir in der Regel nicht fünfzehn oder zwanzig Jahre Zeit. Solange hat es nämlich gedauert, bis wir GUI verstanden und mit passenden Klassenbibliotheken versehen haben. Es liegt einfach in unserer Natur, dass wir komplexe Zusammenhänge nur von Iteration zu Iteration besser verstehen lernen. Wie oft haben wir uns gesagt: „Beim nächsten Mal programmiere ich die Anwendung ganz anders.“? Sieht man sich die Vererbungstiefe von normalen Anwendungen an, so haben 90% aller Klassen genau eine Superklasse und nur 1% hat drei oder mehr Superklassen in der Vererbungskette. In GUI Bibliotheken kann schon mal ein Klasse zehn Superklassen besitzen. Unser Problem liegt ganz einfach darin, dass wir erst im Laufe der Zeit erkennen, wie man die Klassenhierarchien aufteilt und welche Klassen welche Aufgaben übernehmen sollen. Das heißt, wir profitieren leider weniger von Vererbung als man uns gerne weiß machen will (daher werden auch heute noch Projekte überzogen - die Werkzeuge mögen besser geworden sein, unser Denkfähigkeit leider nicht).

Der größte Vorteil der Objektorientierung für normale Anwendungen liegt in der Bildung von Verantwortlichkeiten, also in der Gruppierung von wohl definierten Funktionalitäten zu (gekapselten) Datenstrukturen, ohne großartig auf die Vererbung zu achten. Wiederverwendung kann an dieser Stelle auch durch Delegation erreicht werden. Unserem Button würde ein Vektor übergeben, und bei einem Klick dieser ausgeführt werden.

Außerdem sollte man nicht vergessen, dass bei einer Objektorientierten Programmierung unbedingt ein Hierarchiebrowser benötigt wird. Es wäre so, als würde man sich die Dokumente in einem HTML Browser zwar ansehen können, nur anstelle einfach auf einen Link zu klicken, müsst man diesen mit der Hand eintippen. Man würde wohl den Faden rasch verlieren.

Zusammenfassung

Der Kern der Objektorientierung ist Vererbung. Vererbung ist jedoch eine Form der Organisation, die ein tiefes Verständnis des Problems und der Lösung bedarf, und das benötigt leider sehr viel Zeit (für GUI zwanzig Jahre). Ebenso sind geeignete Werkzeuge unbedingt nötig (ein normaler Editor reicht nicht). Ich persönlich würde daher Vererbung nur in Verbindung mit _wirklich_ objektorientierten Sprachen (z.B. Smalltalk) und nur dort wo Rechenleistung und Speicher keine Rolle spielt (Windows PC's) einsetzen.

Die Seite für den Umsteiger FINDRAMD.COM - Assemblerprogrammierung in Forth

Fred Behringer
<behringe@mathematik.tu-muenchen.de>

Auf Windows-98- und Windows-ME-Notdisketten findet man die "Werkzeuge für den Notfall" in einer CAB-Datei verpackt. Beim Booten wird (über die CONFIG.SYS) eine RAM-Disk eingerichtet und die CAB-Datei wird per EXTRACT in diese RAM-Disk entpackt. Damit gelingt es, auf eine 1.44-MB-Diskette mehr als 2MB an Datei-Material unterzubringen.

Schwierig ist für das Notfall-System die Frage, wo sich denn eigentlich die RAM-Disk befindet. Dem COPY-Befehl für die CAB-Datei muß man ja den Laufwerk-Buchstaben vorgeben. Die Notdiskette sollte für alle (im Hause des Benutzers befindlichen) Maschinen verwendbar sein und auf jeder dieser Maschinen kann die RAM-Disk einen anderen Laufwerk-Buchstaben zugeordnet bekommen haben. Das hängt ganz davon ab, wieviele Laufwerke auf der Festplatte der jeweiligen Maschine eingerichtet wurden. DOS nimmt den ersten freien Buchstaben hinter den Festplatten-Laufwerken für die RAM-Disk.

Auf den erwähnten Notdisketten ist für diese Suche eine Datei namens FINDRAMD.EXE verantwortlich. Sie ist in eine Such-, Entpack- und Kopier-Batch-Datei eingebettet.

Man könnte es so einrichten, daß der Laufwerk-Buchstabe für die RAM-Disk von Fall zu Fall per Hand eingegeben wird. Schöner ist eine vollautomatische Lösung.

Wer sich mit der Assembler-Programmierung beschäftigt hat, weiß, daß es keinen Intel-Maschinenbefehl gibt, der den ersten noch freien Laufwerk-Buchstaben zu finden gestattet. Ein Trick hilft weiter. Es gibt einen Interrupt, der ein vorgegebenes Laufwerk einzuschalten gestattet, und einen, der abzulesen gestattet, welches Laufwerk gerade eingeschaltet ist. Wird bei letzterem ein Laufwerk eingeschaltet, das es gar nicht gibt, dann bleibt die Maschine einfach auf dem zuletzt eingeschalteten Laufwerk stehen und es geschieht nichts weiter. Man verwendet bei dem besagten Trick diese beiden Interrupts in einer Schleife und vergleicht in jedem Schritt die Nummer des vermeintlich eingeschalteten Laufwerks mit der Nummer des tatsächlich eingeschalteten Laufwerks. Sobald diese beiden Nummern nicht mehr übereinstimmen, hört man mit der Schleife auf. Das dann eingeschaltete Laufwerk ist das letzte vorhandene, also die RAM-Disk.

Vorsicht muß man walten lassen, wenn man dieses Verfahren für einen anderen Zweck als für die Not- oder Diagnosediskette verwenden möchte. Im "normalen" Betrieb hat bei mir die RAM-Disk den Laufwerk-Buchstaben T: . Danach kommen dann aber noch das CD-Laufwerk mit U: und das ZIP-Laufwerk mit V: . Aber auch hier käme ich mit einem solchen Verfahren an den Laufwerk-Buchstaben der RAM-Disk, wenn ich nämlich weiß, daß bei jeder meiner zu untersuchenden Maschinen genau gleichviel Laufwerke auf die RAM-Disk folgen.

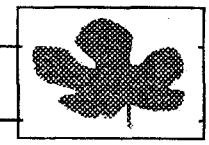
Für den Assembler-Programmierer, und das ist der hier angesprochene Umsteiger, ist das Vorhaben eine Kleinigkeit. Er schaut in einem Buch nach, welches die BIOS-Interrupts (hier Laufwerk-Abfrage) auflistet, beispielsweise in dem überall grassierenden Buch von Alle Metzlar (ich war in der ersten Auflage erwähnterweise für weite Teile der Übersetzung verantwortlich) und "schreibt" das Programm.

Im vorliegenden Artikel soll gezeigt werden, daß man für solche Kleinstvorhaben keinen Assembler mit Linker und Sonstigem braucht. Die in Forth einzugebenden Assemblerelemente, die im übrigen den eigenen Bedürfnissen leicht angepaßt werden können, reichen völlig.

Als Nebeneffekt stellt sich heraus, daß die Länge der hier erzeugten Datei FINDRAMD.COM **nur 20 Bytes** beträgt, während das eingangs erwähnte FINDRAMD.EXE aus der Windows-98-Notdiskette 6855 Bytes benötigt.

```
HEX

LABEL FINDRAMD   \ RAM-Disk aufspüren, einschalten
   DL DL XOR             \ DL = 0
   BEGIN
      0E # AH MOV 21 INT  \ Laufwerk DL einschalten
                          \ (0 = A: usw.)
       19 # AH MOV 21 INT \ Welches Laufwerk ist
                          \ eingeschaltet?
       AL DL CMP 0=       \ Stimmen Laufwerk-Nummern
                          \ noch überein?
     WHILE DL INC         \ Ja, dann nächstes
                          \ Laufwerk testen
     REPEAT
     20 INT               \ Programm beenden
FINDRAMD HERE             \ Ab Labeladresse FINDRAMD
SAVE FINDRAMD.COM         \ als .COM-Datei speichern
```

In ZF muß die letzte Zeile durch den Ausdruck `OVER - DOS FINDRAMD.COM` ersetzt werden. Dabei beachte man zusätzlich, daß diejenige Version von ZF, die ich hier verwende, höchstwahrscheinlich eine von Friederich Prinz und seiner Moerser Gruppe abgeänderte und verbesserte Fassung ist. Zwischen Tom Zimmer und Marc Petremann scheint es von Anbeginn (etwa 1987) an keine Zurkenntnisnahme und



keine Absprache gegeben zu haben. Solche Vorkommnisse machen Standardisierungsbestrebungen, wie die bei ANS-Forth, doppelt bedeutungsvoll.

Sobald man das obenstehende Programm unter Forth einmal aufgerufen hat (was bei Turbo-Forth schon beim Einladen per `INCLUDE FINDRAMD.FTH` geschieht, bei ZF per `FLOAD FINDRAMD.FTH`), steht unter der DOS-Eingabeaufforderung (zum beliebig oftmaligen direkten Aufruf oder zur Einbettung in eine Batch-Datei) die Datei `FINDRAMD.COM` zur Verfügung. Mit dieser wird die RAM-Disk (genauer gesagt, das letzte Laufwerk des Systems) gesucht und eingeschaltet.

Selbstverständlich kann man das obige Programm auch als `CODE`-Definition fassen und von Forth aus aufrufen. Statt `LABEL FINDRAMD` braucht man dazu nur `CODE FINDRAMD` zu setzen, und hinter `20 INT` statt der dort folgenden zwei Zeilen ein `NEXT END-CODE` einzuschieben.

Will man das Ganze in High-Level-Forth kleiden, dann verwende man:

```
HEX
: FINDRAMD ( -- )
    -1 BEGIN 1+ DUP DUP SELECT DRV <> UNTIL DROP ;
```

In einem solchen Fall müßte man dann aber das gesamte Forth-System (auf der Notdiskette) mit sich herumschleppen.

Die obigen beiden Zeilen gelten für Turbo-Forth. In ZF setze man für `DRV` den Ausdruck `0 19 BDOS`. Auch hier gilt das, was ich soeben über Zusammenarbeit, Absprache und Standardisierung gesagt habe. Allerdings ist das in der Sprache Forth, und das ist deren Stärke, weitaus weniger schlimm als in anderen Sprachen. Niemand hindert mich daran, mir in ZF ganz schnell und ein für alle Male das Wort `: DRV 0 19 BDOS ;` einzubauen. Nichts ist leichter als das!

Gehaltvolles

zusammengestellt und übertragen von Fred Behringer

VIJGEBLAADJE der HCC Forth-gebruikersgroep, Niederlande

Nr. 29, Dezember 2001

Diese Ausgabe dreht sich um das holländische Roboter-Selbstbauprojekt und wird von dessen Wortführer, Willem Ouwerkerk, den Vorsitzenden der HCC-Forth-gebruikersgroep, allein bestritten. Schade, daß man so selten einem anderen Autor begegnet. Die Aktivitäten der Fgg scheinen enorm groß zu sein und genügend andere Mitwirkende gibt es, wahre Teamarbeit, aber schreiben tut keiner.

Forth op de HCC-dagen
Willem Ouwerkerk <w.ouwerkerk@kader.hobby.nl>

Willem weist auf die kommenden HCC-Tage hin (Hobby Computer Club). (Die Fgg ist eine Untergruppierung des HCCs. Mitgliedsbeiträge für die Fgg werden an den HCC abgeführt, wofür sich gewisse Anrechte auch in Hinsicht auf die anderen Untergruppierungen des HCCs ergeben.) Die Fgg wird auf den HCC-Tagen einen eigenen Stand unterhalten ("die Lötkolben werden gerade vorgewärmt"). Betaversionen von AVR-ByteForth und 8051-ByteForth können ausprobiert werden, Ushi, der Roboter, wird vorgeführt, Albert van der Horst wird sein 32-Bit-Ciforth (für DOS im Protected Mode, Linux, Windows und Stand-alone) erläutern.

De belofte voor 2001
Willem Ouwerkerk

In diesem fünften Teil der Beschreibung des holländischen Roboter-Bauprojekts (Ushi, die an der Tischkante entlang fährt, ohne runterzufallen) wird in der Hauptsache ein Ultraschall-Abstandsmesser (3-200 cm) besprochen. Es wird ein gut beschriftetes Schaltbild angegeben, nach welchem ich (der Rezensent) mich ohne weiteres an einen Nachbau herantrauen würde. Außerdem gibt der Autor ein vollständiges Programm in AVR-ByteForth 1.00 an. Ein ähnlicher abstandmessender Sensor wurde schon in Vijgeblaadje 7 beschrieben, ein Infrarot-Sensor mit dem GP2D02 in Heft 23. Der hier beschriebene Sensor auf Ultraschallbasis (40 kHz) kommt ohne teure Einzelteile von Conrad aus.

FORTH-Tagung 2002

vom 19. bis 21.April 2002 in Garmisch-Partenkirchen

Liebe FORTH-Freunde,

die FORTH-Tagung'02 findet in der Zeit vom 19. bis 21. April in Garmisch-Partenkirchen statt.
Ulrike und ich werden die Tagung organisieren, Fred sammelt die Vorträge und macht den Tagungsband.

Das Tagungsprogramm wird wieder sehr interessant werden: Als Gastredner erwarten wir Willem Ouwerkerk und Albert Nijhof von der holländischen Forth-gebruikersgroep. Über das Tagungsprogramm informieren wir auch in der Newsgroup de.comp.lang.forth und auf unserer Homepage http://www.forth-ev.de/

Als Tagungshotel haben wir das Wellness- und Sporthotel Forsthaus Graseck http://www.forsthaus-graseck.de/ in Garmisch-Partenkirchen ausgewählt. Das Hotel hat eine hauseigene Seilbahn. Alle Zimmer sind mit Bad/Dusche, WC, Telefon, TV und Balkon ausgestattet.

Die Anreise mit dem Auto oder der Bahn über München nach Garmisch-Partenkirchen ist bequem möglich. Für Bahnreisende bietet das Hotel einen Abholservice vom Bahnhof an (bitte mit dem Hotel direkt vereinbaren). Mit dem Auto fährt man ab München auf die Autobahn A95 Richtung Garmisch-Partenkirchen, dort folgt man den Hinweisschildern "Olympiastadion", "Skistadion", "Eckbauer" oder "Partnachklamm". Ab dem Skistadion dürfen Tagungsgäste mit dem Auto bis zur Talstation der hoteleigenen Bergbahn fahren.

Als Zusatztag bieten wir den 18. April 2002 an. Je nach Witterung planen wir eine Bergwanderung oder den Besuch eines Museums. Zu einem Besuch laden ein: das Franz-Marc-Museum, http://www.franz-marc-museum.de/ das Walchenseekraftwerk http://www.loisachtal.net/html/walchensee-kraftwerk.html und für eine Bergwanderung der Herzogstand www.herzogstandbahn.de/. Diese Aktivitäten bieten wir, je nach Witterung und den Wünschen der Teilnehmer, für den Donnerstag nachmittag und für den Freitag vormittag an. Auch die schönsten Schlösser von König Ludwig II., Linderhof und Neuschwanstein, sind von Garmisch-Partenkirchen aus gut zu erreichen. Hier noch ein paar interessante Web-Seiten: http://www.trimini.de/ http://www.walchensee.net/ http://www.kochel.de/

Das Tagungsprogramm wollen wir locker gestalten, um Freiräume zu schaffen für Workshops und Diskussionen. Vortragende sollten bis spätestens 15. März 2002 ihre Anmeldung inklusive Abstract einsenden, damit wir den Tagungsablauf planen können. Falls die Aufnahme eines Artikels zum Vortrag in die Proceedings gewünscht wird, muss dieser bis zum 01.04.2002 druckfertig vorliegen (DIN A4, 2 cm Rand an allen Seiten, Fußzeile mit Namen und Thema, Seitennummerierung).

Sorgfältige Durchsicht auf Schreibfehler und gutes Layout liegen in der Verantwortung der Autoren. Schickt also der Einfachheit halber Eure Manuskripte bitte druckfertig in Form von (ungeknickten) DIN-A4-Seiten direkt an Fred.

Drei Vortragsankündigungen liegen bereits vor: Cross-Compilation, Lego-Schreibautomat, Forth im OR-Bereich.

Weitere Fragen beantworten wir gerne unter:
Heinz.Schnitter@physik.uni-muenchen.de oder behringe@mathematik.tu-muenchen.de

Ihre Anmeldung sollten Sie so bald wie möglich an das Tagungsbüro schicken, da wir das Zimmerkontingent im Hotel bereits in der ersten Märzwoche festlegen müssen. Noch sind Zimmer frei, so dass wir Teilnehmer nachmelden können. Alle Anmeldungen werden nach dem Prinzip "first come, first served" bearbeiten. **Wer zu spät kommt, schläft im Tal!**

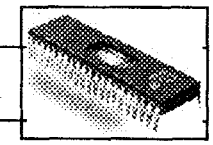
Hier noch einmal die Adressen:

Für die Anmeldung:
Heinz Schnitter
Tagungsbüro FORTH'02
Postfach 1110
85701 Unterschleissheim

für die Vorträge:
Fred Behringer
bei Rohrmayer
Johann-Strauß-Str. 16
85591 Vaterstetten

Wir freuen uns auf Ihr Kommen

Heinz und Ulrike Schnitter
Fred Behringer



MuP21/F21 Bootprozess

Von Sören Tiedemann

Mit diesem Artikel startet Sören Tiedemanns mehrteilige Reihe über die Programmierung der Mikroprozessoren der MuP21/F21 Familie. In den nächsten Vierte Dimensionen wird es weitergehen.

MB

1. Einführung
2. Memory-Map und Buslogik
3. 8-bit Bootmodus
4. Bootroutinen
5. Software

Einführung

In diesem Abschnitt soll der Bootvorgang der Mikroprozessoren MuP21, kurz P21, und F21 verdeutlicht werden. Beide Prozessoren wurden von **Chuck Moore** entworfen. **P21** war der erste Vertreter der sogenannten **MISC** (Minimal Instruction Set Computer) -Chips. Neuerdings etabliert sich aber eine klarere Bezeichnung, **NOSC** (No Operand Stack Computer). **P21** wird bekanntlich von **Dr. C.H. Ting (Offete Enterprises)** unterstützt, **F21** von **Jeff Fox (Ultra Technology**).

Beide Prozessoren setzen die lange Tradition von erfolgreicher FORTH-Hardware fort und öffneten zudem neue Sichtweisen auf die überholte, 30-Jahre alte **FORTH-VIRTUAL-MASCHINE**. In den letzten Jahren ist viel über Chuck's Design-Philosophie gestritten worden. Es ist ebenso sehr viel über die Prozessoren geschrieben worden, leider aber scheinen Chuck's radikale Ideen auch immer wieder einen offensichtlich nicht zu vermeidenden FORTH-politischen Streit mit sich zu bringen, in dessen Zuge gerade seine letzte Generation von Hardware außerordentlich zu leiden hatte.

Seit fast vier Jahren beschäftige ich mich nun mit seinen Chips, beobachte die Reaktionen auf seine Ideen und stelle immer wieder mit Erstaunen fest, wie sehr diese Chips die friedliche FORTH-Gemeinde radikalisieren. Klar, das zugrundeliegende Modell spaltet, entweder man mutiert in einen hoffnungslos Gläubigen (wie ich) oder man wird zu einer giftspritzenden Bestie, zwischen diesen Polen scheint es wenig zu geben bis auf diejenigen, denen das Ganze einfach zu dumm ist und die es links liegen lassen.

Nun, viele FORTH-Systeme kommen mit sämtlichen erdenklichen Tricks daher, ihren 30-Jahre alten Unterbau zu modernisieren. Und dies gelingt dank FORTH auch ganz gut. Aber dies ist NICHT FORTH, wie ich es verstehe. Nicht umsonst entstanden die leider oft missverstandenen Begriffe **Maschinenforth** oder später, schon viel besser, das daraus entstehende **COLORFORTH**.

Nun, wie dem auch sei:

- P21 existiert tatsächlich
- F21 existiert tatsächlich
- P21 funktioniert wie erwartet
- F21 funktioniert wie erwartet

Davon handelt dieser Artikel.
Und davon, dass eine 1KWord-page nahezu unendliche Weiten bedeutet.

Also handelt dieser Artikel von **FORTH**, woran wir alle Spaß haben!

Freiburg, Soeren Tiedemann

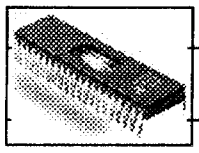
Memory-Map und Buslogik

P21 und F21 haben einen Adressraum von 21-bit.
Der untere 20-bit Adressraum adressiert FPM-DRAM, die oberen 20-bit adressieren slow- und fast-ROM, slow- und fast-SRAM/IO, sowie vorhandene Konfigurationsregister.
Für die ROM-Bereiche werden jeweils eine 256KB große Page im Adressraum bereitgehalten, über ein Konfigurationsregister können statisch 4 verschiedene Pages mit einem Gesamtraum von 1MB ROM nach außen geführt werden.

Auszug aus der F21-Memory-Map

Adressraum	Verwendung
000000 - 0FFFFF	20-bit FPM DRAM, 1MWort, 2.5 MB (60ns)
100000 - 103FFF	Slow 20-bit SRAM, 16KWorte, 40KB
140000 - 143FFF	Fast 20-bit SRAM, 16KWorte, 40KB (12ns)
180000 - 1BFFFF	Slow 8-bit ROM, 4x256KB, 1MB (250ns)
1C0000 - 1FFFFF	Fast 8-bit ROM, 4x256KB, 1MB

Im hohen Adressbereich stehen weitere Konfiguartionsregister zur Verfügung, die bei F21 unter anderem den on-Chip Parallelport steuern, sowie die vorhandenen Koprozessoren initialisieren.



Bootprozess MuP21/F21

Neben den Chipselect- besitzen P21 und F21 die Adressleitungen A0-A9, die als RAS/CAS-Signale gemultiplext an das FPM-DRAM gebracht werden. Beide CPU's besitzen 20 Datenleitungen, D0-D19.

Eigentlich dürfte für F21 damit nur 1K20-bit SRAM zur Verfügung stehen und dies ist auch korrekt, aber es können einige zusätzliche Adressleitungen aus den 12 Parallelportleitungen gewonnen werden, um 16KWorte 20-bit SRAM zu adressieren.

Intern arbeiten P21 und F21 mit 21-bit Registern, das höchste Bit dient dabei als Carry oder für die Adressgenerierung (z.B Chipselect). Die Chips sind wortadressierende CPU's.

1MWort sind daher 1M20-bit, daher 2.5MB, 16KWorte stellen dementsprechend 40KB zur Verfügung.

Im Falle der ROM-Adressierung sind allerdings nur 8 Datenleitungen verbunden, um zu Standardbausteinen kompatibel zu sein. Die hohen 10 Datenleitungen dienen hierbei als Adressleitungen A10-A19. Daher können tatsächlich 1M8-bit in diesem Fall direkt adressiert werden. (2 Bits, A18 und A19, werden statisch über ein Konfigurationsregister gesetzt, der dynamisch adressierbare Raum reduziert sich auf A0-A17, auf die bereits erwähnten 256KB,von denen also 4 Pages gewählt werden können).

Die Folge dieser Adressierungsarten ist, dass die Prozessoren über zwei 'Modi' verfügen, den 20-bit Standard- sowie den 8-bit Bootmodus. Dies wird Inhalt des nächsten Kapitels sein.

Eine Besonderheit der Prozessoren muss noch erläutert werden. Chuck verwendet eine Buslogik, bei der alternierende Bits invertiert sind. Eine 8-bit 'Null' sieht folgendermaßen aus:

10101010

Anders formuliert: P21 und F21 besitzen eine Hex AAAAA-Logik. Im obigen Beispiel erhalten wir also durch XOR AA unsere 'Null'. In der Regel brauchen wir Programmierer uns nicht darum zu kümmern. Eine Null ist halt eine Null, nur die Repräsentation auf den tatsächlichen Leitungen ist eben ... wie erwähnt.

Der 8-bit Bootmodus

Um den 8-bit Bootmodus zu verstehen muss ein Blick auf den 20-bit Standardmodus geworfen werden. Bei einem Instructionfetch werden 20-bits über die Datenleitungen aus dem externen Speicher geholt. Chuck verwendet in seinem Maschinenforth-Modell eine Repräsentation von vier Instruktionsklassen für die FORTH-Primaries:

- 0 Transfer-Instruktionen
- 1 Speicherzugriffs-Instruktionen
- 2 ALU-Instruktionen
- 3 Register-Instruktionen

Eine Anzahl von unter 16 Primaries war ihm ein wenig zu ineffektiv, eine Anzahl irgendwo zwischen 16 und 32 allerdings ausreichend. Dies führte zur Entwicklung von 5-bit Opcodes.

Jedes dieser Opcode-Bits steuert fast direkt eine Mikrofunktion, unter Vermeidung von aufwendiger Dekodierlogik. Die 2 höchsten Bits codieren die Instruktionsklasse, das mittlere der 5-bit ist meist für read/write-Unterscheidung zuständig, die beiden unteren Bits dienen der Enumeration.

Von den 32 möglichen Opcodes werden 25 (P21) bzw. 27 (F21) verwendet.

In ein 20-bit Speicherwort können also 4 Instruktionen gepackt werden, in die sogenannten Slots.

Im späteren F21 konnte Chuck tatsächlich folgendes verwirklichen:

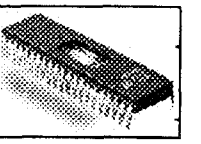
00000111112222233333

Das 20-bit Speicherwort beginnt mit den 5-bits der ersten Instruktion (Slot 0) usw. bis zu den letzten 5-bits (Slot 3).

Bei P21 sah die Situation verwickelter aus. Chuck verwendete auch hier eine alternierende Abfolge, diesmal der Slot-bits (Dies bringt mit sich, dass durch die AAAAA-Logik nur die Slots 0 und 2 betroffen und damit invertiert sind, die Slots 1 und 3 sind nicht betroffen), aber diese Anordnung diente der Reduzierung von Gates und der Beschleunigung des Ripple-Carry-Addierers:

01010101012323232323

Slot-bits 0 und 1 sind in den oberen 10-bits ge'scrambled', Slotbits 2 und 3 in den unteren. Nun, ganz der Wahrheit entspricht dies noch nicht. Schauen wir uns im Hinblick auf einen 8-bit Modus einmal die Datenleitungen D0-D7 im obigen Beispiel an. In den unteren 8 Bits liegt weder eine komplette Instruktion aus Slot 2 noch aus Slot 3. Wir bekämen also keinen vollständigen, gültigen Opcode! Also 'verdrehte' Chuck die Bit-



Nummern 7 und 8, dies ergibt jetzt folgendes Bild:

0101010101223323232 3

Schauen wir uns jetzt einmal die unteren 8-bit an: 33x3x3x3! Nun sind alle 5 Bits des Slot 3 vorhanden. Damit kann auch in den 8 unteren Bits immerhin ein gültiger Opcode liegen! Dies ist wesentlich für den Bootmodus! Wie gesagt ist diese Rangelei bei F21 nicht notwendig, dort sieht die Situation in den unteren 8 Bit so aus: xxx33333. Werden die 3 übriggebliebenen Bits (die 'x') irgendwie verwendet? Die Antwort ist: ja. Sie dienen als Adressbits für die Sprunganweisungen in diesem dadurch sehr eingeschränkten 8-bit-Bootmodus.

Bei drei zur Verfügung stehenden Bits ergeben sich also 8 verschiedene Zieladressen! Wie wird jetzt aus den 3-bit-Adressen eine 8-bit Zieladresse gewonnen? Indem die Instruktionsbits mitzählen!!! Es werden also die gesamten 8 Bits als Adresse interpretiert! Damit erklärt sich eine weitere Besonderheit, nämlich die, dass die möglichen Zieladressen für die verschiedenen Jump-Opcodes nicht identisch sind!

An dieser Stelle möchte ich noch einmal klärend auf die Buslogik zurückkommen. Auf Präsentationen breitete sich bezüglich dieses Themas immer wieder Konfusion aus. Chuck wechselt als Entwickler sehr oft die Perspektive, aus der er seine Schöpfungen beleuchtet. Einmal argumentiert er aus Sicht der CPU, dann wieder aus Sicht einer externen Logik. Ich habe nichts zu entwickeln und argumentiere immer, wenn nicht explizit anders gewählt, aus der Sicht von P21/F21. Eine Null wird von P21/F21 als Null geschrieben und als Null selbstverständlich gelesen. P21/F21 arbeiten mit NUMMERN. Und nichts unterscheidet sie darin von anderen Rechnern. Schaut man sich allerdings mit einer Maschine anderer Buslogik (u.a. herkömmlicher PC) die z.B. auf ein externes Medium (FLASH, battery-packed SRAM ...) nach außen gebrachten Werte an, dann wird man vergeblich nach Übereinstimmung suchen.

Beispiel 1: Man schreibt mit P21 an Adresse Hex 145 einer 64KB großen SRAM-Karte den Wert Hex FF. Wenn diese SRAM-Karte über einen handelsüblichen Kartenleser mithilfe eines PC's ausgelesen wird, wird man auf keinen Fall an Adresse 145 fündig und auch nirgendwo den Wert FF vorfinden. Adresse Hex 145 AAAA XOR entspricht Hex ABEF und Hex FF AA XOR entspricht Hex 55! Der PC muss an Adresse ABEF auf der Karte den Wert 55 auslesen!

Diese Werte bezeichnet Chuck als PATTERN im Gegensatz zu NUMMERN. Der P21/F21- Nummer 145 entspricht das (PC)-Pattern ABEF. Mit diesen Umwandlungen hat man beim Crosscompilieren von und zu anderen Plattformen zu tun, sowie bei I/O zu externen Logiken.

Beispiel 2: Während der seriellen Kommunikation möchte man mit P21/F21 ein CR an den (PC)-Server schicken. Nun, die P21/F21-Nummer Hex 0D würde dort aber als A7 ankommen. Also muss sie vorher mit AA XOR in eben dieses A7-Pattern umgewandelt werden. Dies wird dann korrekt als CR durch den PC empfangen. Aus diesem Grund hat Chuck der einfacheren Verwendung wegen zwei Literalbefehle eingeführt: den '#-Opcode (für Nummern) und den 'p-Opcode (für Pattern). Aus Sicht von P21/F21 werden Pattern zuerst mit AAAAA XOR behandelt, aus PC-Sicht ist es genau umgekehrt, dort müssen die P21/F21-Nummern mit AAAAA XOR behandelt werden.

Chuck schaut häufig von außen auf seine Chips, auch aus diesem Grunde sind die veröffentlichten, vereinfachten, kompakten Opcode-Werte PATTERN und entsprechen NICHT den internen Darstellungen bei P21/F21. Dies macht Sinn, da wohl in der Mehrzahl der Fälle crosscompiliert wird, und diese Pattern halt mit dem PC ohne Umwandlung auf das Bootmedium geschrieben werden können.

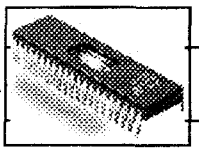
Aufgrund der etwas verwickelteren Situation durch das 'Scrambling' der Bits bei P21 beziehen sich alle folgenden Beispiele auf P21. Das Vorgehen ist jedoch auf F21 übertragbar. Hier noch einmal die unteren 8 Bits des 8-bit Bootmodus' von P21:

i i a i a i a i

Die eingestreuten Adressbits werden nur bei den Sprunganweisungen verwendet, sonst sind sie 0. Als Beispiel hier einmal der DUP-Opcode (Hex 1A) in 8-bit Bootmodus-Schreibweise.

Vereinfachtes, kompaktes 5-bit Opcode-Pattern	Richtige, ge'scramble'te 8-bit -Pattern-Darstellung	Interne P21-Nummer (Pattern AA XOR)
DUP: hex 1A, 5-bit bin 11010	8-bit bin 11a0a1a0, hex C4	8-bit bin 01101110, hex 6E

Wird dieses 'dup' mithilfe eines PC's auf das Bootmedium geschrieben, dann muss Hex C4 verwendet werden. Sollte dieses 'dup' mit P21 selbst geschrieben werden, dann muss man 'p C4, also 6E nehmen. Nun sollte der Eroberung der



Bootprozess MuP21/F21

Sprungbefehle mit eingebetteten Adressbits nichts mehr im Wege stehen.

Als Beispiel soll der CALL-Opcode dienen. Welche Adressen können während des Bootvorgangs mit dem CALL erreicht werden? Der CALL hat das Opcode-Pattern 04 (00100 in 5-bit-Darstellung):

Adressbits	i i a i a i a i - Kombination	P21-Nummer
aaa: 000	00010000, hex 10	hex 10 AA XOR=BA
aaa: 001	00010010, hex 12	hex 12 AA XOR=B8
aaa: 010	00011000, hex 18	hex 18 AA XOR=B2
aaa: 011	00011010, hex 1A	hex 1A AA XOR=B0
aaa: 100	00110000, hex 30	hex 30 AA XOR=9A
aaa: 101	00110010, hex 32	hex 32 AA XOR=98
aaa: 110	00111000, hex 38	hex 38 AA XOR=92
aaa: 111	00111010, hex 3A	hex 3A AA XOR=90

Als Letztes muss noch erwähnt werden, dass in Sprungbefehle eingebettete Adressen, dies gilt ebenso für den 20-bit Modus, durch P21/F21 invertiert interpretiert werden. Die internen Nummern aus der letzten Spalte der obigen Tabelle werden komplett als Adresse interpretiert, die tatsächlichen Zieladressen sind die invertierten Werte. Damit haben wir es dann geschafft, es ergeben sich folgende CALLs mit deren Adressen:

BA FF XOR = 45
B8 FF XOR = 47
B2 FF XOR = 4D
B0 FF XOR = 4F
9A FF XOR = 65
98 FF XOR = 67
92 FF XOR = 6D
90 FF XOR = 6F

P21 und F21 booten mit der internen Adressnummer 1AAAAA. Dies liegt in Übereinstimmung mit der Memory-Map im slow-ROM-Bereich. Die Adressleitungen zeigen dabei das Pattern 00000.

Durch die Sprungbefehle werden nur die niedrigen Adressbits geändert, die hohen Adressanteile bleiben unbeeinflusst.

Nehmen wir jetzt einmal eine Boot(karten)-Epromgröße von 64KB an, dann ergibt sich folgende Boot-Map:

Aus P21-Sicht		Aus PC-Sicht	
AA45	: mögliche CALL-Adresse	00EF	: möglicher CALL
AA47	: -"-	00ED	: -"-
AA4D	: -"-	00E7	: -"-
AA4F	: -"-	00E5	: -"-
AA65	: -"-	00CF	: -"-
AA67	: -"-	00CD	: -"-
AA6D	: -"-	00C7	: -"-
AA6F	: -"-	00C5	: -"-
AAAA	: BOOT	0	: BOOT

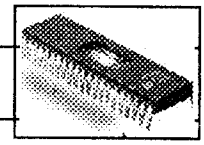
Nun, diese Einsprünge (wie gesagt, für jeden Sprungbefehl andere) liegen in jeder 256Byte-Page! Ein freier Wechsel kann natürlich jederzeit über den gesamten Adressbereich mithilfe der Kombination

```
Adressliteral generieren,
zum Returnstack 'pushen'
und 'ret' ausführen,
```

erzwungen werden. Jeff Fox konnte zeigen, dass komplexe F21 Test-Programme tatsächlich im 8-bit Modus aus dem ROM laufen können.

In der Regel jedoch dient der 8-bit Modus nur dem Erzeugen einer kleinen Kopierschleife im 20-bit Speicher, die dann das weitere OS aus dem ROM ausliest und im 20-bit-Speicher etabliert.

Bevor ich zu den konkreten Routinen komme, möchte ich noch die Speicherzugriffsbefehle erwähnen, die im 8-bit Modus auch ein lustiges Eigenleben entwickeln. Ob im 8-bit oder im 20-bit Modus, die Fetch-Befehle holen ein entsprechendes Literal aus der folgenden Adresse und legen es auf den Stack. Nun, der Stack ist 21-bit breit. Das Carry-bit wird grundsätzlich gelöscht. Werden im 8-bit Modus nun die 12 verbliebenen hohen Bits ebenfalls gelöscht? Mitnichten! Wenn wir uns noch einmal daran erinnern, dass die Datenleitungen genau dieser Bits im 8-Bit Modus als Adressleitungen zum ROM dienen,



dann wird klar, was in diesen Bits landet! Der hohe Adressanteil der gerade aktuellen Speicherzelle! Also, Vorsicht ist hier geboten, die unteren 8 Bits enthalten beim Fetch im Bootmodus das gewünschte Literal, die hohen Bits müssen allerdings manuell gelöscht werden.

```
Chuck's 8-bit BOOT-Assembler für P21:

VARIABLE H                \ rom- Compilationszeiger
CREATE rom 65536 ALLOT    \ rom-Image für das
                          \ Bootmedium,
                          \ Beim Schreiben müssen
                          \ nur noch die Adressen
                          \ ge'scramble't werden.
                          \ Die Inhalte sind
                          \ bereits Pattern.
: LOC
  CONSTANT DOES> @ H ! ;        \ rom-Zeiger setzen

VOCABULARY 8-bit
8-bit DEFINITIONS

: ,
  H @ rom + C!1 H +!;     \ Byte in den rom-
                          \ Bereich schreiben
: INST
  CONSTANT DOES> @ ,;     \ Opcode-Byte
                          \ compilieren
: p
  44 , , ;                \ Crosscompiling
                          \ bedeutet hier, dass
: #                       \ Nummern mit AA
  AA XOR p ;              \ gewandelt werden


\ Die Opcodes in P21's  i i a i a i a i -
\ Darstellung.

41 INST @+       45 INST @       51 INST !
55 INST !        80 INST com     81 INST 2*
84 INST 2/       85 INST +*      90 INST -or
91 INST and      95 INST +       C4 INST dup
C5 INST over     D4 INST nop     D5 INST drop
C0 INST pop      C1 INST a       D0 INST push
D1 INST a!       01 INST ;'
```

Das ist bereits alles! An diesem Beispiel sieht man schon die Einfachheit des Systems!

Sprungbefehle und (gleichzeitig) deren Adressen werden mit den konkreten Bootroutinen definiert.

Bootroutinen

Die Entwicklung von P21 hat verschiedene Generationen dieses Chips in verschiedenen Packages hervorgebracht. Ich beschreibe hier eine generelle Bootroutine, die den kleinen 20-bit Loader im DRAM aufsetzt. Dieser Bootcode kommt bei allen 40 Pin DIP Plastik MuP21-Chips, sowie den MuP21h-Chips im 44-pin PLCC-Package zum Einsatz. In diesen Packages gibt es ein Problem mit der 'a!'-Instruktion im 8-bit Modus, sodass dieser Bootcode ein 'workaround' enthält, dass z.B. auch mit den keramischen 40-pin DIP Prototypen funktioniert, die dieses 'a!'-Problem nicht aufwiesen, dafür aber statt 6 nur 4 funktionsfähige Datenstack-Plätze bereitstellten.

Als erstes folgen die CALL-Opcodes zu den benötigten Unterprogrammen des Bootcodes:

```
18 INST byte      AA4D LOC :byte \ Opcode 18 ist ein
                                 \ CALL zu Adresse
                                 \ AA4D s.o.
30 INST word      AA65 LOC :word \ Opcode 30 ist ein
                                 \ CALL zu Adresse
                                 \ AA65 s.o.
3A INST 0a!       AA6F LOC :0a!  \ Opcode 3A ist ein
                                 \ CALL zu Adresse
                                 \ AA6F s.o.
```

Nun noch einige bedingte/unbedingte JUMP-Opcodes für Schleifen des Bootcodes:

```
22 INST jump_test       \ Opcode 22 ist ein
                        \ unbedingter Sprung zu
                        \ Adresse AA77
24 INST T=0_ende        \ Opcode 24 ist ein
                        \ bedingter Sprung zu
                        \ Adresse AA71
20 INST jump_loop       \ Opcode 20 ist ein
                        \ unbedingter Sprung zu
                        \ Adresse AA75
```

Und natürlich der Booteinsprung:

```
AAAA LOC :BOOT
```

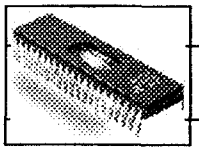
Jetzt kann's losgehen:

```
:0a!                    \ AA6F:     Starte Compilation
                        \ des Wortes '0a!'
          a jump_test   \ Hole A auf den Stack, springe
                        \ zu Label _test.
( _ende)  ;'            \ AA71:  Wenn Adressregister
                        \ A=0, dann sind wir fertig.
  nop nop nop           \ AA72: 3 Füllbytes
( _loop)  @+ a          \ AA75: Hole A+1 auf den Stack
( _test)  T=0_ende      \ AA77: Wenn TOS=0 dann springe
                        \ zu Label _ende
          jump_loop     \ AA78:     Springe unbedingt an
                        \ das Label _loop
```

0a! wird aufgrund des a!-Bugs im 8-bit-Bootmodus einmal gerufen. Um das OS aus dem Bootmedium ins DRAM ab Adresse 0 zu kopieren muss das Adressregister einmal initialisiert werden. :BOOT schreibt eine Null ins Adressregister, dies kann funktionieren, kann aber auch fehlerhaft sein. Also ruft :BOOT kurz darauf diese kleine Schleife, die im Falle das A nicht 0 enthält, das Autoinkrement des Adressregisters nutzt, um A solange zu inkrementieren, bis durch Überlauf wirklich eine Null erzeugt wurde!

```
:byte ( ca cb --word )   \ AA4D: Definiere 'byte'
  2* 2* 2* 2*
  2* 2* 2* 2*     \ cb 8 Bits nach links schieben
  push            \ retten
  00 # -or        \ hoehere 12 Bits in ca (mit
                  \ kleinem Trick) löschen *!*
  pop -or ;'      \ ca in die unteren 8 freien Bits
                  \ von cb hineinodern und zurück.
```

Byte verschmilzt 2 Bytes. Da noch nicht sichergestellt ist, dass das untere Byte wirklich 'sauber' ist, dass heißt in den hohen Registerbits auch wirklich 0 steht, verwendet diese Routine einen kleinen Trick um dies zu erreichen.



Bootprozess MuP21/F21

Es wurde bereits erwähnt, dass die höheren 12 Bits den hohen Adressanteil enthalten. Nun, dieser hohe Adressanteil ist auch bei erneutem Holen eines beliebigen Literals wieder vorhanden. Die scheinbar sinnlose Folge 00 # -or löst dies auf elegante Weise. Das -or löscht die störenden hohen 12 Bits mittels des 0-Literals aus dem unteren Byte! Folglich kann dann das untere Byte gefahrlos hineinge-oder-t werden.

```
:word  ( c1 c2 c3 -- word ) \ AA65: Definiere 'word'
  byte byte                 \ 3 Bytes zu einem 20-
                            \ bit-Wort verschmelzen
  !+ ;'                     \ 20-bit Wort ins 20-bit
                            \ DRAM schreiben,
                            \ A inkrementieren.
```

Word bekommt 3 Bytes und verschmilzt diese zu einem 20-bit Wort. Vom höchsten Byte werden nur 4 Bits genutzt, daher ist es nicht notwendig, das höchste Byte ebenfalls zu 'säubern'.

Das gewonnene Wort wird über das Adressregister ins 20-bit DRAM geschrieben, wobei das Adressregister inkrementiert wird und somit auf die nächste DRAM-Adresse zeigt.

Die DRAM-Kopierroutine vervollständigt sich am Ende selbst. Beim ersten Durchlauf holt sie ihr eigenes '+ until' aus dem ROM und kopiert in diesem Beispiel 8K-Worte.

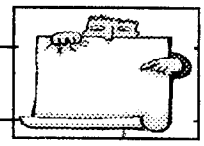
Ja, Moment, wenn sie sich selbst vervollständigt, dann muss sie ja schon irgendwo im Speicher stehen! Richtig, das gesamte OS liegt in dieser 3-Byte Aufschlüsselung, durch einen 20-Bit Assembler generiert, ab einer beliebigen Adresse im ROM (hier ab 0000), beginnend mit der DRAM-Bootroutine. Ja, warum ist diese dann zusätzlich noch 'hard'-codiert? Der Bootcode könnte die 3-Byte-Worte dann doch dort auslesen und sie ins DRAM kopieren! Ja, wenn dieser ärgerliche 'a!'-Bug nicht wäre! Mit meinem Keramik-MuP21 mache ich das auch so, aber es erfordert im 8-bit Bootcode ein Umladen des A-Registers. Und dies kann schiefgehen! In der hier vorliegenden Variante muss Chuck das A-Register während des gesamten 8-bit Modus nur einmal mit Null initialisieren. Danach wird an diesem Register nichts mehr geschraubt, bis der 20-bit Modus erreicht wird. Ab da: grünes Licht!

Die Bootroutine

```
:BOOT                   \ AAAA:       Power-up/Reset Booteinsprung
  pop pop               \ ein wenig an den Stacks ziehen, zur Initialisierung der Hardware
  dup -or a!            \ eine 0 erzeugen und nach A schreiben.
          0a!           \ sichergehen, dass A=0 ( dies ist der DRAM-Beginn )

83 p 0E p 0C p word     \ ': byte     2* 2* 2* 2*           \ ROM: 0000, DRAM: 00
46 p 0E p 0C p word     \             2* 2* 2* '#
55 p AA p 0A p word     \             FF
21 p 4E p 06 p word     \             @+ and -or ;'

F9 p 4B p 0F p word     \ ': word     a push nop a!         \ ROM: 000C, DRAM: 04
FC p 0F p 06 p word     \             @+ 2* nop nop
55 p 81 p 00 p word     \                 byte
55 p 49 p 08 p word     \             2* byte
F9 p 0B p 0F p word     \             a pop nop a!
01 p C7 p 0B p word     \             push !+ pop ;'

a push                                                  \ ' BOOT ( 0A ) >R

DC p 21 p 06 p word     \ ': BOOT     '# com '# nop         \ ROM: 001E, DRAM: 0A
65 p 55 p 0F p word     \             '-5FFCF               \ 1A0030, ROM-Position 0030
AA p A2 p 0A p word     \             800
51 p C1 p 0A p word     \             begin   push word
FC p 13 p 0B p word     \                     pop '# nop nop
2A p AA p 0A p word     \                     80

                        \             + until               \ ROM: 0030, DRAM: 10
                                                            \ wird von der DRAM-
                                                            \ Routine beim ersten Mal geholt
                                                            \ Sprung zu DRAM-BOOT
```

Es ist deutlich, wie :BOOT das A-Register initialisiert und dann jeweils drei Pattern-Literale lädt und word ruft, um das gewonnene 20-Bit-Wort ins DRAM zu schreiben. Die 'hard'-codierte 20-bit DRAM-Kopierroutine ist im Kommentar als Klartext aufgeschlüsselt. Das 'a push' von :BOOT sichert den DRAM-Pointer, der in diesem Moment auf (DRAM)BOOT zeigt, auf den Returnstack, so kann am Ende ganz einfach ins DRAM und damit in den 20-Bit Modus gesprungen werden.

Die 20-bit-Routine findet das A-Register auf ihr eigenes Ende zeigend vor (DRAM-Position hex 10). Sie holt sich zu Beginn ein Literal (5FFCF) und invertiert dieses (1A0030). Nun, dies ist der ROM-Pointer auf ihr eigenes Ende im ROM (Der ROM-Offset ist Faktor 3 größer, 3 Bytes ergeben ein 20-bit Wort). Nach dem ersten Durchlauf hat sie sich selbst vervollständigt und kopiert das gesamte OS. Letztlich liegt das OS aus dem ROM ab 0000 im 20-bit DRAM ab 00000 vor und



fängt ja automatisch an, sich auszuführen sobald die Kopierschleife verlassen wird!

In der Regel folgt dieser DRAM-Kopierroutine ein weiterer Initialisierungsschritt. Der Video-Koprozessor-Code wird im DRAM erzeugt, dieser stellt einen Video-Frame zur Verfügung und regelt auch den wichtigen DRAM-REFRESH. Erst wenn der Video-Koprozessor eingeschaltet werden kann und er seinen Frame abarbeitet ist man mit dem Hauptspeicher auf der sicheren Seite.

Diese Schritte, beginnend mit dem 20-bit Assembler, dem Video-Koprozessorcode-Assembler und dem Video-Frame-Generator werden Inhalte der nächsten VD's sein.

Software

Natürlich in starker Anlehnung an Chuck's, Jeff's und Dr. Ting's Vorgaben, habe ich ein eigenes Package unter BigForth entwickelt, dass recht bequem die Erzeugung von Boot-Images für P21 und F21 erlaubt. Die hier gegebenen Beispiele beziehen sich ja auf P21, doch kann das Vorgehen auf F21 übertragen werden.

Diese Images können dann auf EPROMS oder SRAM-Karten verwendet werden.

Das gesamte Package kann über die FORTH-Gesellschaft oder direkt über meine Homepage geholt werden: www.planet-interkom.de/soeren.tiedemann

Hier finden sich auch eine Einführung in die Opcodes und ein paar grundlegende Gedanken zur Programmierung dieser CPU's im 20-bit Modus.

Literatur:

OK-OS	Chuck Moore
MuP21 Programming Manual,	Dr. C.H. Ting
More on Forth Engines, Vol. 22	Dr. C.H. Ting
More on Forth Engines, Vol. 25	Dr. C.H. Ting
P21Forth User Manual,	Jeff Fox, *
F21 Microprocessor Specifications	Jeff Fox, *
F21d Datasheet	Jeff Fox, *
MISC-Minimal Instruction Set Computer	S.Tiedemann

(* = Ultra Technology d. Red.)

Zu allen diesen Quellen findet man Verweise oder Informationen unter Jeff's Homepage:

www.ultratechnology.com

Call for Papers

Wir wollen auch diesmal wieder unsere Tagung mit einer möglichst großen Zahl von Kurzvorträgen (je 15 Minuten) würzen. Schickt mir bitte möglichst früh Eure Absichtserklärungen:

Fred Behringer <behringe@mathematik.tu-muenchen.de> .

Drei Anmeldungen liegen bereits vor: (1) Cross-Compilation, (2) Lego-Schreibautomat, (3) Forth im OR-Bereich .

Sorgfältige Durchsicht auf Schreibfehler und gutes Layout liegen in der Verantwortung der Autoren. Schickt mir also der Einfachheit halber Eure Manuskripte bitte druckfertig in Form von (ungeknickten) DIN-A4-Seiten an:

Fred Behringer
bei Rohrmayer
Johann-Strauß-Str. 16
85591 Vaterstetten

Ich werde die eingereichten Blätter in einem Copy-Shop (schwarz/weiß) vervielfältigen (lassen). Die "Tagungsbände" werden dann schlichte Einheftungen loser Blätter in nicht zu strenger Reihenfolge ohne Vorschriften in Bezug auf Paginierung, Schriftart, Schriftgröße etc. werden.

Es ist wieder vorgesehen, die einzelnen Vorträge in loser Reihenfolge in den anschließenden Heften der Vierten Dimension zu veröffentlichen.

Sollten Übersetzungen nötig werden (Gäste aus England oder/und Holland), so bitte ich um ganz besonders frühe Kontaktaufnahme.

Selbstverständlich stehe ich für Sonderabsprachen gern zur Verfügung.

Fred Behringer

Pontifex

Friederich Prinz

...meint hier nicht den Brückenbauer zwischen Himmel und Erde, sondern, auch nicht immer so trivial, den Brückenschlag von einem Flußufer zum nächsten. Pontifex ist die Nachfolgeversion von „Bridge Builder", einem Programm zum virtuellen Bau von Brücken. Pontifex's Hersteller stellt eine Demoversion zum Download unter WWW.CHRONICLOGIC.COM bereit. Die Demoversion bietet 10 verschiedene Spieleebenen an, bzw. 10 vorgegebene Brückenbauaufgaben, mit wachsenden Anforderungen an die Phantasie des Spielers. Gegenüber der ersten Version, auf die uns Martin Bitter vor ungefähr einem Jahr aufmerksam gemacht hatte, stehen heute zwei unterschiedliche „Stahlsorten" und auch Kabel als lastabtragende Elemente bereit. Wie schon vor einem Jahr kann der ungeübte Spieler sich sehr schnell derbe „verbauen". Die Auswahl einer ungünstigen Konstruktion und teurer Materialien führt relativ zügig dazu, daß dem Freizeitstatiker mitten im Bau das Geld ausgeht. Und wenn wir schon bei „zügig" sind, dann sollte auch nicht unerwähnt bleiben, daß in der neuen Version der erste Probezug die neue Brücke mindestens vier Mal überqueren muß, wenn das Bauwerk als abgenommen akzeptiert werden soll.

Das Ganze ist heute „dreidimensional" und mit filigranen Stabwerken nett anzuschauen. Die Option, sich nicht die Brücke selbst anzusehen, sondern den „Streß" der Stabwerke sichtbar zu machen, bringt richtig „Leben in die Brücke" – häufig deutlich mehr als man möchte. Dazu muß unbedingt auf die „Sounds" aufmerksam gemacht werden, die vor allem bei einstürzenden Bauwerken wirklich höhrenswert sind.

Die Demo ist, wie schon die erste Version des Bridge Builders, kostenfrei beim Hersteller herunterladbar.

Eine „professionelle" Version läßt sich dort nach Überweisung von 20,-$ US ebenfalls beziehen.

Insgesamt ist Pontifex eine nette Spielerei an regnerischen Winterabenden. Wenn Spaziergänge ins Wasser fallen müssen, dann dürfen Brücken das auch...

file:///E|/4D/1_2002/koyocera/Material/Brücke2.gif

Forth-Gruppen regional

Hamburg **Küstenforth**
Klaus Schleisiek
Tel. 040 / 375008-13 g
kschleisiek@send.de
Treffen jeden 4. Freitag im Monat
16:30 Uhr, Fa. SEND, Stubbenhuk 10
20459 Hamburg

Moers **Friederich Prinz**
Tel.: 02841-58398 (p) (Q)
(Bitte den Anrufbeantworter nutzen !)
(Besucher: Bitte anmelden !)
Treffen: (fast) jeden Samstag,
14:00 Uhr, **MALZ, Donaustraße 1**
47443 Moers

Mannheim **Thomas Prinz**
Tel.: 06271-2830 (p)
Ewald Rieger
Tel.: 06239-920 185 (p)
Treffen: jeden 1. Mittwoch im Monat
Vereinslokal Segelverein Mannheim e.V.
Flugplatz Mannheim-Neuostheim

München **Jens Wilke**
Tel.: 089-89 76 890
Treffen: jeden 4. Mittwoch im
Monat, **China Restaurant XIANG**
Morungerstraße 8
München-Pasing

µP-Controller Verleih

Thomas Prinz
Tel.: 06271-2830 (p)
micro@forth-ev.de

Gruppengründungen, Kontakte

Fachbezogen 8051 ... (Forth statt Basic, c-Forth)
Thomas Prinz
Tel.: 06271-2830 (p)

Forth-Hilfe für Ratsuchende

Forth allgemein
Jörg Plewe
Tel.: 0208-49 70 68 (p)

Jörg Staben
Tel.: 02173-75708 (p)

Karl Schroer
Tel.: 02845-2 89 51 (p)

Spezielle Fachgebiete

Arbeitsgruppe MARC4	Rafael Deliano Tel./Fax: 089 -841 83 17 (p)
FORTHchips (FRP 1600, RTX, Novix)	Klaus Schleisiek-Kern Tel.: 040 -375 008 03 (g)
F-PC & TCOM, Asyst (Meßtechnik) embedded Controller (H8/5xx//, TDS2020, TDS9092),Fuzzy	Arndt Klingelnberg, Consultants akg@aachen.kbbs.org Tel.: ++32 +87 -63 09 89 pgQ (Fax -63 09 88)
KI, Object Oriented Forth, Sicherheitskritische Systeme	Ulrich Hoffmann Tel.: 04351 -712 217 (p) (Fax: -712 216)
Forth-Vertrieb	volksFORTH / ultraFORTH RTX / FG / Super8 / KK-FORTH Ingenieurbüro Klaus Kohl Tel.: 08233-3 05 24 (p) Fax : 08233-99 71 mailorder@forth-ev.de
Forth-Mailbox (KBBS)	0431-533 98 98 (8 N 1) Sysop Holger Petersen hp@kbbs.org Tel.: 0431-533 98 96 (p) bis 22:00 Fax : 0431-533 98 97 Helsinkistraße 52 24109 Kiel

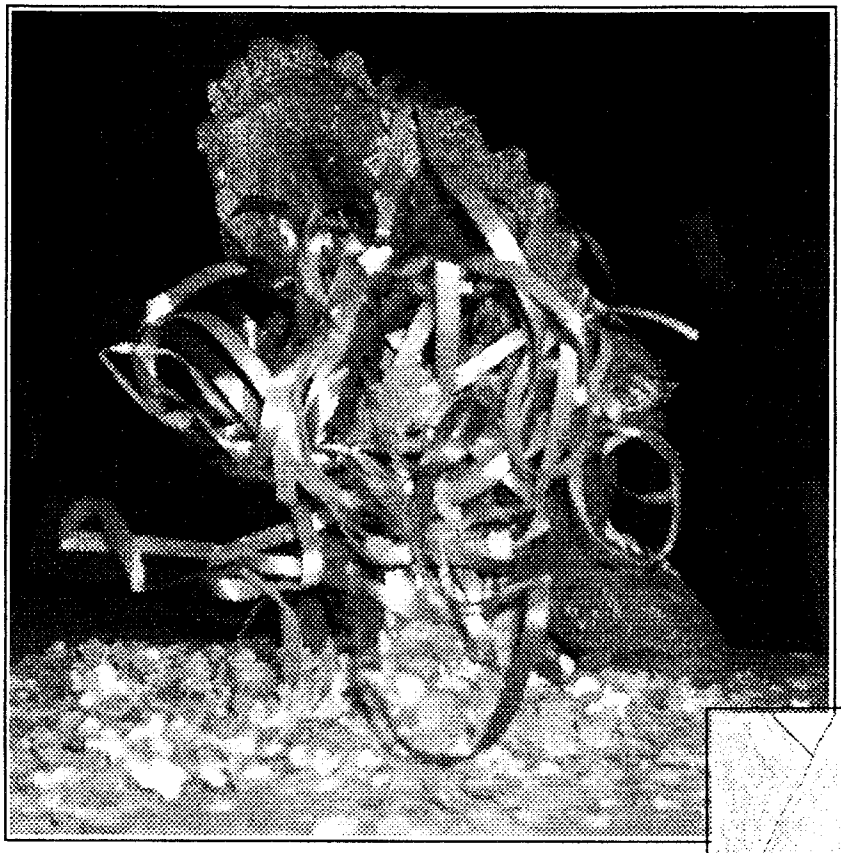
Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren ? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten ? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen ?

Schreiben Sie einfach der VD - oder rufen Sie an - oder schicken Sie uns eine E-Mail !

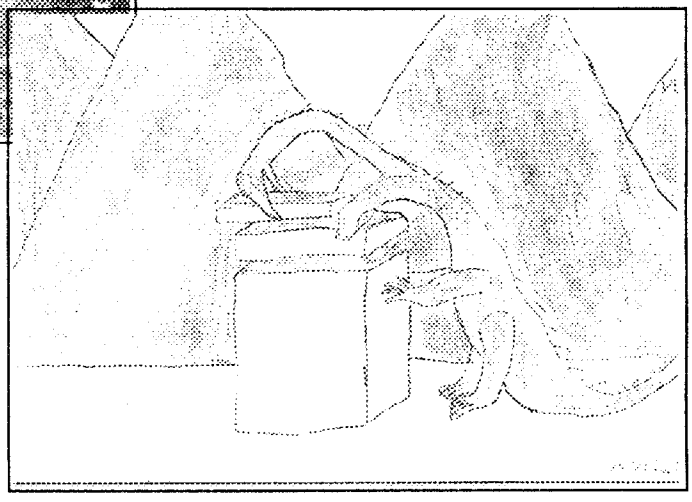
Hinweise zu den Angaben nach den Telefonnummern:

Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeistzeiten
g = geschäftlich

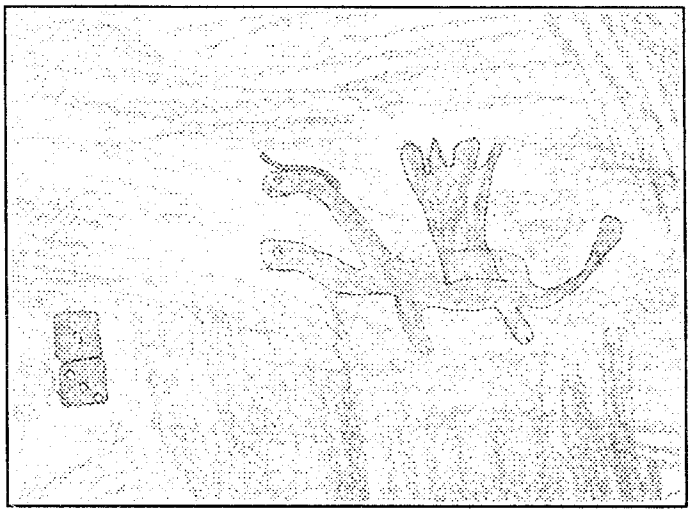
Die Adressen des Büros der Forthgesellschaft und der VD finden Sie im Impressum des Heftes.



Die seltsamen Abenteuerdes Swapp-Drachens im November 2001,



die Folgen
und ...



wie es dazu kam.

In der Vierten Dimension Nr. 2 2002