Vierte Dimension

3+4/2007

Das Forth–Magazin

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

In dieser Ausgabe:

Das FOCUS–Zahlensystem

EuroForth 2007

Hexzahlen: $ vs. HX

Taxi–Nummer 1729

Der M16C von Renesas lernt Forth

Forth von der Pike auf — Teil 9

Dictionary im Filesystem

False Forth

Ist Forth Turing–vollständig?

Ulam–Spirale

Primzahlen und die Spirale von Ulam

Forth–Gruppe München

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 SwiftX86k und seit kurzem mit Holon11 und MPE IRTC für Amtel AVR.

LEGO RCX–Verleih

Seit unserem 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. 230,– € im Handel zu erwerben ist.

Inhalt:

1 RCX, 1 Sendeturm, 2 Motoren, 4 Sensoren und ca. 1.000 LEGO Steine.

Anfragen bitte an
Martin.Bitter@t-online.de

Letztlich enthält das Ganze auch nicht mehr als einen Mikrocontroller der Familie H8/300 von Hitachi, ein paar Treiber und etwas Peripherie. Zudem: dieses Teil ist „narrensicher“!

RetroForth
Linux · Windows · Native
Generic · L4Ka::Pistachio · Dex4u
Public Domain
`http://www.retroforth.org`
`http://retro.tunes.org`

Diese Anzeige wird gesponsort von:
EDV–Beratung Schmiedl, Am Bräuweiher 4, 93499 Zandt

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

Tel.: 02461/690–380
Fax: 02461/690–387 oder –100
Karl–Heinz–Beckurts–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 80 a D–18057 Rostock
Tel.: (0381) 46 13 99 10 Fax: (0381) 4 58 34 88

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.

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl–Schöpe

Tel.: 07044/908789
Buchenweg 11
D–71299 Wimsheim

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

Das FOCUS–Zahlensystem 6
Rafael Deliano

EuroForth 2007 10
Dagobert Michelsen

Hexzahlen: $ vs. HX 12
Albert Nijhof, HCC-Forth-gebruikersgroup, Niederlande

Taxi–Nummer 1729 15
Henry Vinerts, SVFIG USA

Der M16C von Renesas lernt Forth 17
Albert van der Horst

Forth von der Pike auf — Teil 9 21
Ron Minke

Dictionary im Filesystem 22
Matthias Trute

Lebenszeichen 24
Bericht aus der FIG Silicon Valley: *Henry Vinerts*

Gehaltvolles 25
zusammengestellt und übertragen von *Fred Behringer*

False Forth 27
Fred Behringer

Ist Forth Turing–vollständig? 31
Ulrich Hoffmann

Ulam–Spirale 34
Michael Kalus

Primzahlen und die Spirale von Ulam 36
Fred Behringer

Forth–Gruppe München 38
Bernd Paysan

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 19 02 25
80602 München
Tel: (0 89) 1 23 47 84
E–Mail: `Secretary@forth-ev.de`
`Direktorium@forth-ev.de`
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

Redaktion & Layout

Bernd Paysan, Ulrich Hoffmann
E–Mail: `4d@forth-ev.de`

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

Januar, April, Juli, Oktober jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Liebe Leser,

im Oktober diesen Jahres entstand in der Usenet–Newsgroup `de.comp.lang.forth` unter dem Titel `lokale Gruppen` eine angeregte Diskussion über unsere VD. Die Beteiligten ereiferten sich heftig daran wie unsere VD einen größeren Leserkreis ansprechen könnte. Dieses Ziel sollte durch einen Wandel der VD in eine *on Emmbeded-Control*–Fachzeitschrift erreicht werden, indem wir uns hauptsächlich auf dieses Thema konzentrieren und in größerem Umfang Artikel ohne forthigen Hintergrund aufnehmen. Da wir für eine ausreichend dicke Zeitschrift weit mehr Artikel als bisher bräuchten, dachten die Beteiligten über Autoren–Honorare und professionelle bezahlte Editoren nach. Weiter müsste das Erscheinungsbild unserer Vereinszeitschrift einem zeitgemäßen Outfit für eine marktgängige Fachzeitschrift weichen. Aber woher soll denn die Forth–Gesellschaft die Energie nehmen, wenn wir jetzt kaum in der Lage sind unsere Zeitschrift mit Artikel zufüllen? Wir müssten uns den kommerziellen und wirtschaftlichen Bedürfnissen dieses Geschäftes unterordnen. Damit würde die VD in der Flut heutiger Fachzeitschriften untertauchen und selbst bei größter Anstrengung wäre zu befürchten, dass wir uns mittelfristig nicht gegen die harte Konkurrenz am Markt behaupten könnten. Ein derartig einseitiges Vorgehen kann deshalb nicht unser Ziel sein und steht klar im Widerspruch mit der Satzung der Forth–Gesellschaft e.V., die jeder unter `www.forth-ev.de` einsehen kann.

Daher freue ich mich besonders, wieder eine neue VD mit interessanten Inhalten über Forth in seinen bunten Fassetten und Berichten über das Vereinsgeschehen in den Händen zuhalten. Dafür meinen herzlichen Dank an alle Autoren, besonders aber an unsere neuen Mitglieder die tatkräftig mitwirkten. Nicht zu vergessen die Arbeit unserer ehrenamtlich tätigen Editoren Ulrich Hoffmann und Bernd Paysan, die neben beruflichen Veränderungen (siehe Leserbrie f von Fred Behringer) und großen beruflichen Belastungen auch noch das Amt eines Direktors ausüben.

Ein Verein lebt eben von seinen engagierten Mitgliedern. Um so bedauerlicher ist deshalb die Nachricht, dass Rolf Schöne nach 5 Jahren das Forth-Büro abgibt. Damit ist klar, daß die Forth–Gesellschaft ein neues Forth–Büro braucht. Wir suchen daher dringend Interessenten für dieses Amt, die sich zutrauen die Verwaltung und Buchhaltung des Vereins gewissenhaft zu übernehmen. Ein weiterer Schwerpunkt liegt beim Druck und Versand der Vierten Dimension. Um die Belastung zu reduzieren, ist eine Aufteilung auf mehrere Schultern denkbar.

Deshalb mein Appell: Lieber Leser, denke mal darüber nach, wie deine aktive Beteiligung in der Forth–Gesellschaft aussieht. Vielleicht gibt es noch die eine oder andere freie Stunde, um eine der oben genannten Tätigkeiten zu übernehmen oder sich am Vereinsgeschehen mit neuen Projekten aktiv zu beteiligen und darüber in der VD zu berichten. Auch kleine vielleicht zunächst unscheinbare Dinge aus der täglichen Praxis sind gefragt und ein paar Zeilen wert. Sie sind für den Leser oft verständlicher und hilfreicher als Artikel über komplexe Themen. Nutzt die besinnlichen Stunden am Jahresende neue Ideen in Forth zu sammeln, um sie vielleicht zusammen mit einem Partner aus der Forth–Gesellschaft im kommenden Jahr 2008 anzugehen.

Ewald Rieger

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://www.forth-ev.de/filemgmt/index.php`

Die Forth–Gesellschaft e. V. wird durch ihr Direktorium vertreten:

Ulrich Hoffmann Kontakt: `Direktorium@Forth-ev.de`
Bernd Paysan
Ewald Rieger

Neues vom Forth–Büro

Wir heißen zwei neue Mitglieder willkommen:

Frank Frohriep, so an die 50, ist förderndes Mitglied. Er beschäftigt sich viel mit embedded Controllern.

Thomas Blender, mitten in den Zwanzigern (erfreulich), studiert Technomathematik in Karlsruhe.

Jedes Jahr um diese Zeit erhalten Sie Ihre Beitragsbriefe, so auch heute. Überweisen Sie bitte Ihren Mitgliedsbeitrag nach dem 01.01.2008 und vor dem 21.03.2008. Wenn Sie uns eine Einzugsermächtigung gaben, so werden wir den Beitrag am 21.03.2008 einziehen.

Daneben finden Sie Einladungen zur Tagung und Versammlung in Roggenburg. Nutzen Sie den Frühbucher–Rabatt.

Gegenwärtig haben wir 133 Mitglieder. Pütz, Allinger und Kripahle werden uns zum Jahresende verlassen. Sorgen Sie bitte für Nachwuchs für unsere gute Sache, wo immer Sie ihn finden können — Freunde, Bekannte, Kommilitonen ...

Rolf Schöne

BigForth, nanoForth, genuineForths wie amforth und gforth.

Neulich waren auf dclf interessante postings zu lesen, in welchen über die Themen nachgedacht wurde, die sich für unser Forth–Magazin „Vierte Dimension“ (VD) eignen könnten. Entzündet hatte sich dieser Thread an der Frage „lokale forth gruppen“ — wer trifft sich noch wo — nachden deutlich geworden war, dass unsere lokalen Forthgruppen bis auf eine verschwunden sind: Die Münchner Gruppe gibt es weiterhin.

Wenn ich das richtig verstanden habe, wurde die nanoForth–Seite, also die der kleinen Systeme, in unserer VD Rafaels Meinung nach unglücklich dargestellt und war zudem unterrepräsentiert. Und er sähe es gern, wenn Forth in unserem Blatt auch ‘eingebettet’ zum Thema gemacht würde statt explizit.

Mir wäre es auch lieber, wenn in der Abfolge ausgehend von einem speziellen Prozessor übers Forth zur Applikation die Betonung auf die Applikation gelegt würde. Gegebenenfalls kann das Problem auch ohne Verwendung eines bestimmten Prozessors dargestellt werden. Diese Untersuchung eines Problems mit **Hilfe** von Forth zu bewerkstelligen, stelle ich mir spannend vor. Dabei ist dann das Problem im Vordergrund, Forth wird zum Werkzeug für die elegante Art solcher Untersuchungen. Sei es, dass Hardware–, sei es dass Software–Probleme untersucht werden aus der Sicht des Embedded Control. Die andere Seite, ich nenne sie mal die bigForth–Seite, wurde von Bernd Paysan vertreten. Er war, wenn ich das richtig verstanden habe, auch der Auffassung, dass ausgehend von konkreten Problemen interessantere Beiträge für die VD gefunden werden könnten. Auch auf der Ebene der größeren Maschinen sind Probleme gut beschreibbar in Forth. Als Beispiel wurde an den Beitrag von Deliano und Storjohan über die Galois–Felder erinnert.

Wir hätten somit auf diesen beiden Seiten eine problembezogene Sicht, ergänzt um den Blick auf ein geeignetes Werkzeug. Ich finde es prima, das so zu sehen und anzugehen. Und denke, dass für beide Richtungen in der VD Platz genug ist. Und es sollte auch möglich sein, in der VD Fälle aufzuzeigen, wo ein Problem mit einem andern Werkzeug als Forth besser angegangen werden konnte, und auch zu sagen, warum das so gewesen ist.

Und dann hätten wir da noch die dritte Art der VD–Beiträge: Forth an–und–für–sich. Das nenne ich mal die genuine Form. Da ging es früher schon mal um einzelne Forthworte oder Sequenzen, die besprochen wurden, vom NEXT für den Prozessor xy, Standards, floored division, floating point, lokale Variablen und so etwas, um einige zu nennen. Auch das hatte und hat bestimmt weiterhin seinen Platz in der VD. Das kürzlich erschienene Sonderheft zum amforth für die atmel avr atmega micro controller family war von dieser Sorte. Dort wurde ein besonderer Entwurf eines Forth–Systems vorgestellt von Matthias Trute, in dem die Forth–Worte als jeweils eigenes File betrachtet wurden.

Und ich fände es tatsächlich schön, wenn es gelingen würde, zu diesen drei Bereichen gute Beiträge für die VD hereinzuholen.

Ich jedenfalls wünsche mir aus all den genannten Bereichen Beiträge, und es dürfen für mich auch durchaus Grundlagen dabei sein, wie DCF77, RFID oder Halbleiter—Gyros von Analog Devices, mit Beschaltungen dazu, auch ohne eine Zeile Forth. Das ist es ja, was mich persönlich an der Forth–Gesellschaft immer so angezogen hat, die Fülle des Know–hows in so vielen Bereichen. Also nur zu!

Viele Grüße, Michael

Glückwunsch an Professor Dr. Ulrich Hoffmann

Lieber Ulli,

als ehemaliger FG–Mitdirektor erkühne ich mich (in Absprache mit vielen, die dich kennen und mit dir zusammenarbeiten), dir zu deiner Berufung zum Professor für Software–Technik (Verteilte Anwendungen, Middleware–Technologie, Persistenztechniken, Spezifikation und Verifikation, Rechnernetze, Scientific Computing) an der Fachhochschule Wedel (ab 1. Oktober, zunächst noch mit Probe–Anlaufzeit bis zur eigentlichen Ernennung im Frühjahr 2008) ganz herzlich zu gratulieren. Jeder, der einen Beruf erlernt hat, ist stolz auf das Errungene. Hochschullehrer ist nicht nur Beruf, Hochschullehrer ist im wahrsten Sinne des Wortes zugleich Berufung. Als Hochschullehrer kann man ganz besonders stolz auf das bereits Geleistete sein. Nicht zuletzt schon deswegen, weil man erfolgreich durch die unerbittliche Mühle der Auslese gegangen ist, die viele für immer zermahlt und nur wenige übrig lässt. Und man sieht sich in der angenehmen Position, angehäufte Erkenntnisse mit voller Unterstützung aller zuständigen Stellen an die nachfolgende Generation weiterzugeben. Das macht Spaß. Ganz bestimmt kommt dabei auch weiterhin Forth nicht zu kurz — und die Forth–Gesellschaft kann ihrerseits stolz darauf sein, dich in ihrer aktiven Mitte zu sehen. Toi, toi, toi für das anstehende halbe Jahr der Bewährung und für den nun beginnenden Berufsabschnitt!

Fred

weitere Leserbriefe und Meldungen auf Seite 11

Das FOCUS–Zahlensystem

Rafael Deliano

Es hat sich später zwar herausgestellt, dass einige technische Details bereits in [6] vorweggenommen waren. Das tut seiner Bedeutung für die Popularisierung von LNS auf Mikroprozessoren aber keinen Abbruch.

Von den verschiedenen Publikationen ist [2] besonders ergiebig, da auf 40 Seiten auch Listings für den 8080 enthalten sind. Dort ist alternativ zum 8–Bit– auch ein 16–Bit–Format beschrieben, das aber auf dekadischem Logarithmus beruht, was für heutige LNS untypisch ist.

8–Bit

Von der Mantisse existiert nur das Vorzeichen (Bild 1). Der 7–Bit–Exponent besteht aus Integer (Tabelle 1) und Fraction (Tabelle 2). In der logarithmischen Darstellung entspricht der Offset 1000,000b dem Exponenten 0,0 und dem realen Wert 1,0 (Bild 2). Der Offset kompensiert den Verlauf des Logarithmus (Bild 3), der ja unterhalb 1,0 problematisch ist.

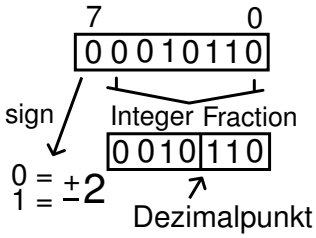


Bild 1: Zahlenformat

1111	=	7
1110	=	6
1101	=	5
1100	=	4
1011	=	3
1010	=	2
1001	=	1
1000	=	0
0111	=	-1
0110	=	-2
0101	=	-3
0100	=	-4
0011	=	-5
0010	=	-6
0001	=	-7
0000	=	-8

Tabelle 1: Integer

111	=	7/8
110	=	6/8
101	=	5/8
100	=	4/8
011	=	3/8
010	=	2/8
001	=	1/8
000	=	0/8

Tabelle 2: Fraction

I/O

Im Originaltext wurde angenommen, dass die Zahlen manuell gewandelt passend eingegeben werden (Tabelle 3). Negative Zahlen erhält man dann mit FNEGATE simpel durch Invertierung des obersten Bits. Rudimentäre Ausgabe ist durch einen Befehl F. möglich (Bild 4). Zumindest für Tests ist das gangbar.

ROMs

Für die Einbindung in Applikationen ist aber typisch Umwandlung in lineares Format nötig. Die Dynamik entspricht einem linearen 16–bis–17–Bit–Datenwort. Damit sind für die Ein–/Ausgabe 32–Bit–Datenworte nötig. Der Wert 234,75 wurde hier mit 10'000d skaliert und passt dann noch in 24 Bit. Berechnung der Tabelle erfolgt mit Mathcad. Wegen des Vorzeichenbits verkürzen sich der Adressbereich der Tabellen um je 1 Bit, da sie für positive Zahlen genügen. Die invlog–Tabelle ist mit 0,7k Byte vom Speicherverbrauch unkritisch (Bild 5).

$$1111\ 111 = 2^{7+7/8} = 234{,}75$$

$$1001\ 000 = 2^{1+0/8} = 2$$

$$1000\ 101 = 2^{0+5/8} = 1{,}5422$$

$$1000\ 000 = 2^{0+0/8} = 1$$

$$0111\ 000 = 2^{-1+0/8} = 0{,}5$$

$$0000\ 000 = 2^{-8+0/8} = 0{,}003906$$

Bild 2: Beispielwerte

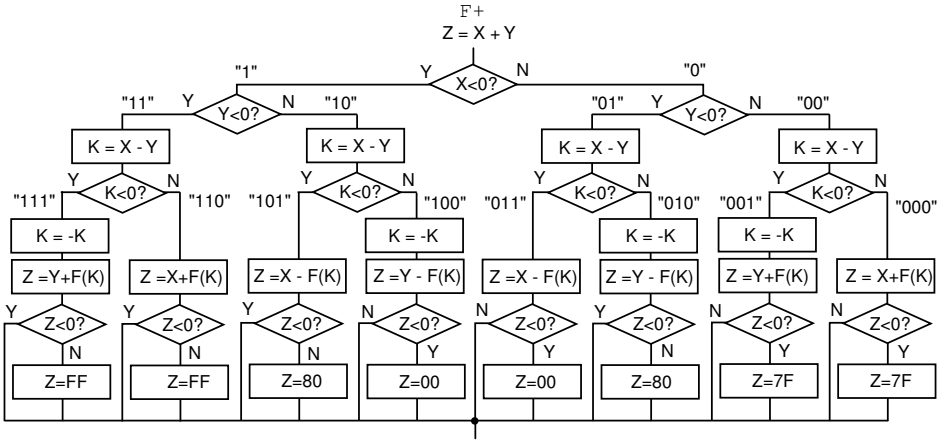


Bild 7: Addition

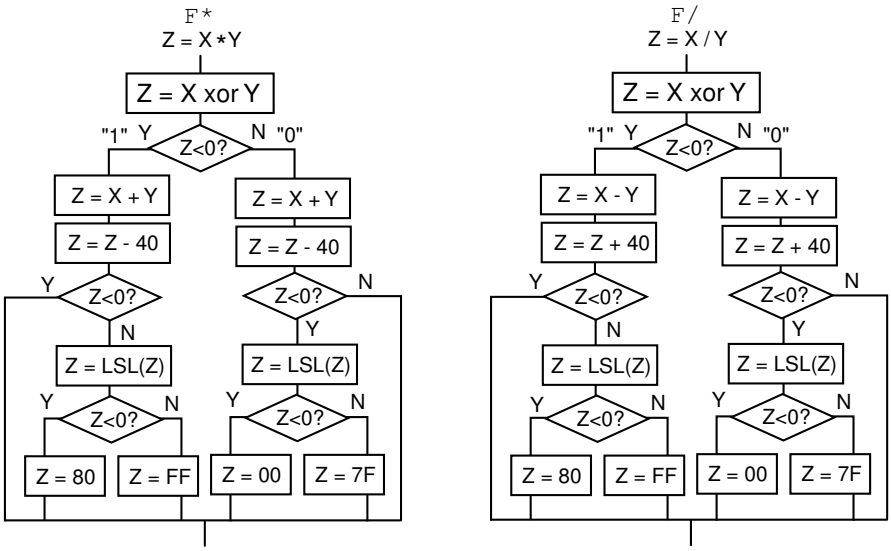


Bild 9: Multiplikation und Division

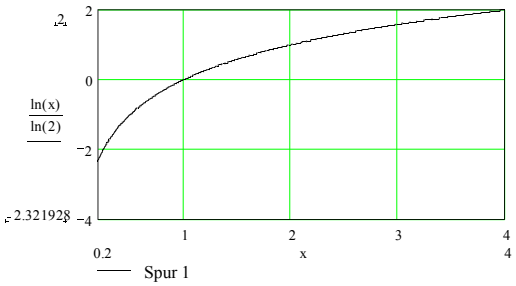


Bild 3: ld(x)

Eine direkte log–Tabelle wäre deswegen jedoch unrealistisch (Bild 6). Hier wird deshalb ersteinmal mit der Funktion SCANBIT die Position des obersten gesetzten Bits gesucht [7]. Es ergeben sich Werte von 5 ... 21d im Dynamikbereich, außerhalb greift eine Sättigungslogik. SCANBIT zerlegt die krumme Kurve also bereits in 17 Teilabschnitte, die selbst fast linear sind. Mit den folgenden 5 Bit wird eine Tabelle für die Feinauflösung angesteuert. Diese ist dreifach vorhanden, weil im Eingangsbereich Block 5 und 6 wegen der Kappung am Dezimalpunkt sich von Block 7 – 21 leicht unterscheiden. LOG ist prinzipiell nicht verlustfrei. Es ist unklar, ob das Rundungsverhalten der gewählten Routine optimal ist, hier stand Geschwindigkeit im Vordergrund.

```
— — —     |   50 \ 4,0
— — 0050  |   F.
+2^0+0/8
— — —     |   50 FNEGATE F.
-2^0+0/8
— — —     |
```

Bild 4: Simple I/O

Befehle

Subtraktion lässt sich hier durch Invertierung des Vorzeichens eines Operanden auf Addition zurückführen (Bild 8). Diese benötigt wegen der 4 Vorzeichen–Varianten, und da sie Addition und Subtraktion gleichzeitig behandelt, eine umfangreiche Fallunterscheidung (Bild 7). Normalerweise sind in LNS Additions– und Subtraktionstabelle getrennt, hier wurden sie in eine 256–Byte–Additions–Tabelle zusammengefasst. Die letzten Operationen dienen jeweils der Implementierung einer Sättigungslogik.

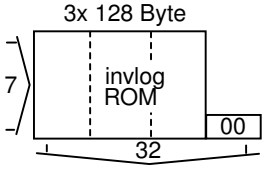


Bild 5: invlog–Tabelle

Multiplikation und Division (Bild 9) beruhen auf 2er–Komplement–Addition und –Subtraktion. Die magische Zahl 40 entspricht dem Offsetwert „1,0“. Anders als bei Addition gibt es hier keine Rundungsfehler.

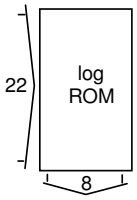


Bild 6: unrealistische log–Tabelle

Implementierung

In Applikationen wird man immer die Assemblerroutinen verwenden. Für Portierung ist aber eine FORTH–Variante nützlich, die der Assemblerversion möglichst ähnlich sein sollte. Timing und Speicherbedarf für einen 2,45MHz–68HC908GP32 in Tabelle 4, 5.

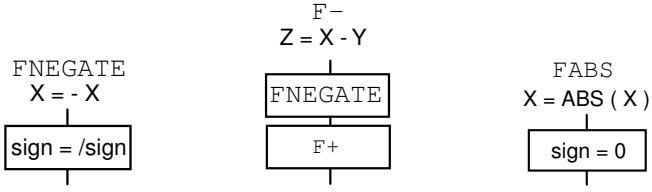


Bild 8: einfache Operationen

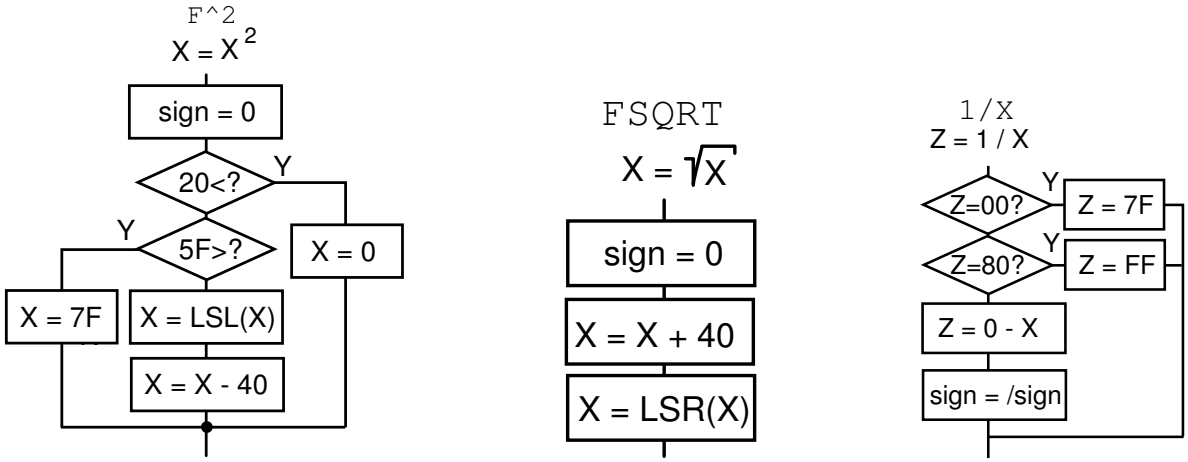


Bild 10: Quadrat, Wurzel, 1/X

	FORTH	Assembler
INVLOG	150–250	25–40
LOG	200–1000	30–120
F+	300	27
F* F/	100–150	15–20

Tabelle 4: Geschwindigkeit in usec

	FORTH	Assembler
INVLOG	60 + 4*256	76 + 3*256
LOG	325 + 113	141 + 113
F+	633 + 256	146 + 256
F* F/	154	58

Tabelle 5: Speicherverbrauch Code + Tabellen

Tests

Wegen des kompakten Datenformats kann man die FORTH– und Assemblerroutinen über den gesamten Wertebereich (d.h. 256 x 256 Berechnungen) gegeneinander testen, ob sie bitgenau identische Resultate liefern.

Literatur

[1] Lee, Edgar „The FOCUS number system“ IEEE Trans. Comp. Nov 1977
[2] Lee, Edgar „Focus Microcomputer Number System“ in: Lee „microcomputer design and applications“ Academic Press 1977
[3] Edgar „FOCUS microcomputer number system“ Com. of ACM March 1979
[4] Lee, Edgar Addendum to „The Focus Number System“ IEEE Trans. on Computers Sept. 1979
[5] Swartzlander Comment on „The Focus Number System“ IEEE Trans. on Computers Sept. 1979
[6] Swartzlander, Alexopolous „The Sign/Logarithm Number System“ IEEE Trans. on Computers Dez. 1975
[7] emb (9) Einfacher Logarithmus

Listing: FOCUS

```
HEX

TABLE ADD-TAB
8 C, 8 C, 7 C, ... 0 C, 0 C, 0 C,

: F+   \ ( X Y - Z )
OVER 80 AND
IF.         \ L1
DUP 80 AND
  IF. 2DUP - DUP 80 AND \ L11
    IF     \ L111
      NEGATE   \ - X Y K )
      FF AND ADD-TAB + C@ + SWAP DROP
      DUP 80 AND LNOT IF DROP FF THEN
    ELSE   \ L110      - X Y K )
      SWAP DROP FF AND ADD-TAB + C@ +
      DUP 80 AND LNOT IF DROP FF THEN
    THEN
  ELSE.   2DUP - DUP 80 AND   \ L10
    IF     \ L101   - X Y K )
      SWAP DROP FF AND ADD-TAB + C@ -
      DUP 80 AND LNOT IF DROP 80 THEN
    ELSE    \ L100
      NEGATE  \ - X Y K )
      FF AND ADD-TAB + C@ - SWAP DROP
      DUP 80 AND      IF DROP 00 THEN
    THEN
  THEN
ELSE. DUP 80 AND    \ L0
  IF. 2DUP - DUP 80 AND    \ L01
    IF     \ L011  - X Y K )
      SWAP DROP FF AND ADD-TAB + C@ -
      DUP 80 AND      IF DROP 00 THEN
    ELSE NEGATE  \ L010  - X Y K )
      FF AND ADD-TAB + C@ - SWAP DROP
      DUP 80 AND LNOT IF DROP 80 THEN
    THEN
  ELSE.    \ L00
  2DUP - DUP 80 AND
    IF NEGATE   \ L001  - X Y K )
```

```
41          FF AND ADD-TAB + C@ + SWAP DROP
42          DUP 80 AND      IF DROP 7F THEN
43        ELSE   \ L000  - X Y K )
44          SWAP DROP FF AND ADD-TAB + C@ +
45          DUP 80 AND      IF DROP 7F THEN
46        THEN
47      THEN
48    THEN FF AND ;
49
50    : FNEGATE   80 XOR ;      \ ( X - X' )
51    : FABS      7F AND ;      \ ( X - X' )
52    : F-        FNEGATE F+ ; \ ( X Y - Z )
53
54    : F*    \ ( X Y - Z )
55    2DUP XOR 80 AND
56    IF + 40 - DUP 80 AND LNOT \ ,,1''
57      IF 40 AND IF 80 ELSE FF THEN THEN
58    ELSE  \ ,,0''
59    + 40 - DUP 80 AND
60      IF 40 AND IF 00 ELSE 7F THEN
61      THEN
62    THEN FF AND ;
63
64    : F/    \ ( X Y - Z )
65    2DUP XOR 80 AND
66    IF - 40 + DUP 80 AND LNOT \ ,,1''
67      IF 40 AND IF 80 ELSE FF THEN
68      THEN
69    ELSE - 40 + DUP 80 AND  \ ,,0''
70      IF 40 AND IF 00 ELSE 7F THEN
71      THEN
72    THEN FF AND ;
73
74    : 1/X    \ ( X  --- X' )
75    DUP IF          \ +0 -> 7F
76      DUP 80 =      \ -0 -> FF
77      IF DROP FF
78      ELSE 0 SWAP - \ 2er-complement
79           80 XOR   \ invert sign
80           FF AND
81    ELSE DROP 7F
82    THEN THEN ;
83
84    : ^2    \ ( X  --- X' )
85    7F AND
86    DUP 20 U<
87    IF DROP 0
88    ELSE DUP 5F U>
89      IF DROP 7F
90      ELSE 1<SHIFT FF AND 40 -
91      THEN
92    THEN ;
93
94    : SQRT  \ ( X   --- X' )
95    7F AND
96    40 +
97    1SHIFT> ;
```

Byte	Dezimalwert	Byte	Dezimalwert	Byte	Dezimalwert	Byte	Dezimalwert	Byte	Dezimalwert
00	0,0039	1A	0,0372	34	0,3936	4E	3,364	68	32,000
01	0,0043	1B	0,0405	35	0,3856	4F	3,668	69	34,896
02	0,0046	1C	0,0442	36	0,4204	50	4,000	6A	38,055
03	0,0051	1D	0,0482	37	0,4585	51	4,362	6B	41,499
04	0,0055	1E	0,0526	38	0,5000	52	4,757	6C	45,255
05	0,0066	1F	0,0573	39	0,5453	53	5,187	6D	49,351
06	0,0066	20	0,0625	3A	0,5946	54	5,657	6E	53,817
07	0,0072	21	0,0682	3B	0,6484	55	6,169	6F	58,688
08	0,0078	22	0,0743	3C	0,7071	56	6,727	70	64,000
09	0,0085	23	0,0811	3D	0,7711	57	7,336	71	69,792
0A	0,0093	24	0,0884	3E	0,8409	58	8,000	72	76,109
0B	0,0101	25	0,0964	3F	0,9170	59	8,724	73	82,998
0C	0,0110	26	0,1051	40	1,000	5A	9,514	74	90,510
0D	0,0120	27	0,1146	41	1,091	5B	10,375	75	98,701
0E	0,0131	28	0,1250	42	1,180	5C	11,314	76	107,635
0F	0,0143	29	0,1363	43	1,297	5D	12,338	77	117,377
10	0,0156	2A	0,1487	44	1,414	5E	13,454	78	128,000
11	0,0170	2B	0,1621	45	1,542	5F	14,672	79	139,585
12	0,0186	2C	0,1768	46	1,682	60	16,000	7A	152,219
13	0,0203	2D	0,1928	47	1,834	61	17,448	7B	165,995
14	0,0221	2E	0,2102	48	2,000	62	19,027	7C	181,019
15	0,0241	2F	0,2293	49	2,181	63	20,749	7D	197,403
16	0,0263	30	0,2500	4A	2,378	64	22,627	7E	215,269
17	0,0287	31	0,2726	4B	2,594	65	24,675	7F	234,753
18	0,0313	32	0,2973	4C	2,828	66	26,909		
19	0,0341	33	0,3242	4D	3,084	67	29,344		

Tabelle 3: manuelle Wandlungstabelle

EuroForth–Konferenz 2007 auf Schloss Dagstuhl

Dagobert Michelsen

Teilnehmer der EuroForth-2007 Konferenz
hinten v.l.n.r. Janet Nelson, Sergey Baranov, Stephen Pelc, Nick Nelson, Carsten Strotmann, Urich Hoffmann
mitte v.l.n.r. Bernd Paysan, Federico Ceballos, Dagobert Michelsen,
Aton Ertl, Elke, Klaus Schleisiek, Peter Knaggs
vorne mitte v.l.n.r. Willem Botha, Bill Stoddart

Die diesjährige EuroForth fand vom 13. bis 16. September auf Schloss Dagstuhl statt, wie immer, wenn die Konferenz in Deutschland ist. Das idyllisch in der Nähe von Idar-Oberstein gelegene Tagungszentrum bot auch diesmal neben der ausgesprochen angenehmen Atmosphäre einen angemessenen Rahmen für Vorträge und Workshops rund um Forth.

Der nullte Tag vor der eigentlichen Konferenz war dem Thema Forth 200X, dem kommenden Forth-Standard, gewidmet, zu dem ich als *Observer* anwesend war. Die technischen Diskussionen drehten sich diesmal insbesondere um die standardisierte Einbindung von Multibyte-Characters (XCHARs). Ich hatte bereits erwartet, dass eine Standardisierung nicht ohne Arbeit zu bekommen ist. Dennoch war ich beeindruckt, wie um jedes einzelne Wort in dem zukünftigen Standard gerungen und wie fein die Begründungen gegeneinander abgewogen wurden. Nach der offiziellen Eröffnung durch den Veranstalter Klaus Schleisiek wurde in einem Vortrag von Anton Ertl über das Foreign–Function–Interface von gForth die Einbindung von C–Bibliotheken in Forth–Code vorgestellt. In der beschriebenen Erweiterung werden Stubs für die C–Funktionen manuell mit den gewünschten Signaturen definiert und dann zur Laufzeit on–demand übersetzt. Die Implementierung arbeitet zurzeit aufgrund des spezifischen Linker–Aufrufs nur unter Linux. An einer plattformübergreifenden Lösung wird gearbeitet.

Anschließend hat Bill Stoddard eine Methode zur Umkehrung von Zustandsänderungen in laufenden Forth–Programmen vorgestellt. Dazu werden die Änderungen im Speicher mitprotokolliert und dann auf Anforderung zurückgerollt.

Der zweite Tag begann mit einer Präsentation von Stephen Pelc über den SEAforth–Prozessor von Chuck Moores neuer Firma Intellasys. Der Prozessor verfügt über 24 unabhängige Kerne, welche in einer 6–mal–4–Matrix angeordnet sind und über 4 Kanäle mit den jeweiligen Nachbarn bzw. externem I/O kommunizieren können. Da keine Interrupts vorgesehen sind, muss dieses in Software realisiert werden. Die interne Struktur ist stackorientiert und orientiert sich wie von Chuck zu erwarten an dem Programmierkonzept von Forth. Durch die hohe Rechenleistung der 24 Kerne soll der Chip als General–Purpose–Chip bisher eingesetzte Spezialchips ersetzen. Die Firma ist in den letzten Jahren rasant von 4 auf 250

Mitarbeiter gewachsen, hat viele von den *alten Hasen* angeworben und sucht nach den Worten von Stephen händeringend weitere erfahrene Forth–Programmierer. Ob das der Anfang eines Revivals von Forth ist?

Danach folgte ein Vortrag mit anschließender Diskussion von Klaus Schleisiek über das Design des Microcore–Nachfolgers *Microcore 2.0*. Microcore ist eine freie Prozessorarchitektur, welche als VHDL–Beschreibung vorliegt und als Grundlagen eigener Entwicklungen etwa in FPGAs gebrannt werden kann. Herausragend ist die Möglichkeit, auf einfache Weise anwendungsspezifische Instruktionen zu ergänzen und so komplexe Aufgaben quasi in Hardware zu erledigen. Der ursprüngliche Microcore ist beliebig in der Datenbreite skalierbar, aber die Instruktionen haben 8 Bit. Microcore 2.0 wird 16 Bit breite Instruktionen haben, um die Dekodierung zu vereinfachen.

Bernd Paysan hat eine weitere Anwendung für BigForth und die Bibliothek *MINOS* für grafische Oberflächen vorgestellt. Die vorgestellte Anwendung dient zur interaktiven Ansteuerung aller Features und Register eines neuen Signalprozessors für den Audiobereich. Dabei wird die Hardware durch das Programm direkt angesteuert und ist sowohl unter Linux als auch unter Windows lauffähig.

Am Abend gab es ein kleines Konzert im Musikzimmer von der Formation *Purple Pool* aus Hamburg. Auf die Frage „Spielen die Free Jazz?“ antwortete Klaus Schleisiek, welcher neben der Ausrichtung der Konferenz auch das Rahmenprogramm gestaltet hat: „Nicht nur“. Neben quartorientierten Improvisationen wurden auch einige den barocken Räumlichkeiten angemessene klassische Stücke gespielt. Die Darbietungen auf insgesamt 22 Instrumenten wurden von den Teilnehmern gut aufgenommen.

Am letzten Tag war Raum für Spontanvorträge. Federico Ceballos hat Ideen für typsichere Record–Strukturen vorgestellt, Anton Ertl hat über eine Methode zum ausnahmesicheren Umsetzen von globalen Variablen innerhalb einer Funktion nachgedacht. Der Anfang vom Ende für Forth wurde von Klaus Schleisiek eingeläutet, welchem aber von allen Seiten energisch widersprochen worden ist. Insbesondere Carsten Strotmann konnte berichten, durch seine Aktivitäten in der Retrocomputing–Szene viele neue Interessenten gewonnen zu haben. Mit einer ganzen Reihe von Vorschlägen und nicht zuletzt durch das neue Produkt von Chuck Moore sollte die nächste EuroForth wieder mehr als die diesmal anwesenden 15 Teilnehmer zählen können.

Bernd Paysan hat Fotos von der Veranstaltung gemacht.

Die nächste EuroForth 2008 wird von Anton Ertl in Wien ausgerichtet und voraussichtlich Ende September 2008 stattfinden. Ich auf jeden Fall freue mich schon!

Leserbriefe (Fortsetzung von Seite 5)

zum AVR–Sonderheft

Hallo Fred,

hab ich doch glatt vergessen, Dich in die Empfängerliste zu schreiben :-(mea culpa.

Dafür kann ich hier gleich meinen Respekt äußern: Das (r8c–)Forth–Glossar auf der CD vom Mikrokontrollerheft ist ja echt mächtig gewaltig. Wow! Ich denke, das wird auch mir helfen, weil alles schön an einem Ort zusammengesammelt ist.

Mach's gut!
Erich

——— Original Message ———
From: Erich Wälde <ew.ng116837@online.de>
Subject: Rückmeldung AVR–Sonderheft
Date: Mon, 26 Nov 2007 21:27:14 +0100

Tag,

neulich hat jemand erwähnt, dass es zu dem *AVR–Sonderheft* der 4.Dimension immer noch so gar keine Äußerungen gibt. Na dann:

Ich fand das Heft *ganz gut*. Der Artikel von Ron Minke war mir als 4D-Leser natürlich schon bekannt, und daher nicht neu. ByteForth kannte ich auch schon, weil ich Besitzer eines AVRex–Boards von Willem Ouwerkerk bin.

Den Artikel zu AMForth fand ich sehr spannend. Meine Lesart: AMForth ist ein sehr ordentlich gemachtes Forth. Schön auch die Befehlsreferenz auf einer Seite. Soweit so gut. Allerdings ist jedes Forth für mich nur Theorie, solange es nicht auf einem Kontroller/Rechner läuft. Ja auch ich gehöre eher zur praktischen Fraktion mit Lötkolben. Und daher war mir der Artikel letzlich zu dünn.

Ich versuche gerade, einen atmega32 auf dem AVRex–Board mit AMForth zu verheiraten. Ich kann den atmega programmieren (mit avrdude und einem sehr simplen parport programmer). Ich kann die Beispielapplikation (template.asm) mit amforth 2.3 erfolgreich assemblieren. Das Programm übertragen, geht wohl auch, sagt jedenfalls avrdude. Aber auf der seriellen Schnittstelle rührt sich leider so gar nixx. Aber ich werd's schon noch rauskriegen.

Ja, ich kann natürlich auch Byteforth und den dazugehörigen dongle nehmen, aber dann muss ich dosemu anwerfen. Und das gefällt mir als Hardcore–Un*xer eben nicht.

Und ja, ich kann selbstverständlich auch ein Pollin–Board käuflich erwerben. Aber: das _muss_ doch auch mit dem schon vorhandenen Material möglich sein.

Also wenn's nach mir geht, darf das in der 4.Dimension ruhig konkreter sein, mit Schaltplan und so. Aber ich bin eben einer mit Lötkolben.

Grüße, Erich

weitere Leserbriefe und Meldungen auf Seite 14

Hexzahlen vorübergehend als $1AFF eingeben oder als HX 1AFF ?

Albert Nijhof, HCC-Forth-gebruikersgroup, Niederlande

Die mit freundlicher Genehmigung der HCC–Forth–gebruikersgroep hier wiedergegebene Übersetzung aus dem Holländischen wurde von Fred Behringer angefertigt. Das Original erschien in der Vijgeblaadje–Ausgabe 64 der niederländischen Forth–Gruppe im Oktober 2007 als Reaktion auf eine `comp.lang.forth`–Diskussion.

Vorwort des Übersetzers: Übersetzen ist nicht immer leicht. Manchmal muss man den Mut haben zu erfinden. Im Englischen scheint es üblich zu sein, beispielsweise das `66`, das im Intel–Real–Mode aus dem 16–Bit–Assembler–Befehl `AX` per `66 AX` den 32–Bit–Befehl `EAX` macht, als *prefix* zu bezeichnen. Aber auch das `$`, das in manchen Systemen die Forth–Eingabe `$4711` als Hexadezimalzahl kennzeichnet, wird als *prefix* bezeichnet. Im Deutschen ist das *in* in *in Berlin* eine Präposition, aber kein *Präfix*. Das *in* in *informieren* ist ein Präfix (DUDEN: vorn an den Wortstamm angefügtes Bildungselement), aber keine Präposition. Albert, der Autor des vorliegenden Artikels, verwendet für `$` im obigen Beispiel im niederländischen Original die Bezeichnung *voorteken*, zu Deutsch *Vorzeichen*. *Vorzeichen* ist mir im Deutschen zu sehr mit Plus und Minus verknüpft. In Beispielen wie dem obigen `66` (in `66 AX`) verwendet Albert die Bezeichnung *voorzetsel*, zu Deutsch *Präposition* oder *Verhältniswort*. Das kann ich genau so wenig vertreten. Zur ganz genauen Verdeutlichung spricht Albert auch von *losse voorzetsel* (dort, wo sie (als Forthwort) wirklich *lose* vorangesetzt werden). Ich überlasse es den Sprachgewaltigen in der Forth–Gesellschaft, eine geignete Bezeichnung zu finden, und verwende einfach das an die deutsche Rechtschreibung angepasste und im Übrigen ins Auge springende Wort *Vorsetzel* oder *loses Vorsetzel* (je nach Situation) als Bezeichnung. Und nun überlässt der Übersetzer das Wort dem Autor:

In der comp.lang.forth bemühen sich die Forthler um einen Vorschlag, *Vorsetzel* in das offizielle Forth–Standardisierungsdokument aufgenommen zu bekommen. Vorsetzel sind zum Beispiel das `$` und das `#` in `$100` und `#100`.

Ein vorn fest an eine Zahl angefügtes Vorsetzel bestimmt, welchen Wert das basisbestimmende Wort `BASE` beim Verarbeiten dieser Zahl verwenden soll. `$` bedeutet hexadezimal und `#` könnte dezimal bedeuten. Man ist sich über die endgültige Wahl der Zeichen noch nicht ganz einig. Die Basis wird durch `$` oder `#` nur zeitweilig verändert. Nach `$100` ist sie wieder dieselbe wie vorher. Im Kielwasser dieses Themas finden sich auch jene Vorsetzel, die den ASCII–Code oder den Kontrollcode liefern, wenn sie (als Vorsetzel) vor ein (beliebiges) Zeichen gesetzt werden.

ASCII

Ich fange mit dem Letzteren an, mit dem Einlesen von ASCII–Code. Zu diesem Behufe stehen (in ANS–Forth) die Worte CHAR und [CHAR] bereit, die aber zwei Nachteile haben:

1. Sie sind für ihre Funktion zu lang. Vor allem [CHAR] vernebelt die Sicht auf das Zeichen, um das es geht, und ermüdet beim Eintippen.
2. Der Programmierer muss ständig überlegen, ob er CHAR oder [CHAR] einzusetzen hat.

Der zweite Nachteil zeigt große Ähnlichkeit mit dem d–oder–dt–Problem bei den niederländischen Zeitwörtern. Das ist eine künstliche Rechtschreibhürde, die einzig und allein dazu dient festzustellen, ob der Schreiber seine Sprache beherrscht oder nicht. Wollte man das –dt abschaffen und einführen, dass *hij word* (er wird) und *jij vind* (du findest) ohne t dahinter korrektes Holländisch ist, dannn würde das bedeuten, dass eine Tradition verloren ginge, die auf viele Zeitgenossen nur irritierend wirkt. (Übers.: Vielleicht könnte man im Deutschen zur Erläuterung den Ausdruck *er verwandt* (zum Schreiben einen Kugelschreiber) und den Ausdruck *er verwand* (seinen Schmerz über den erlittenen Verlust) einander gegenüberstellen.)

ASCII–Code einlesen können Sie mit dem losen Vorsetzel `CH`, welches leicht definiert werden kann und die oben genannten Nachteile nicht hat:

Das Vorsetzel `CH`

Wenn Sie gegen State–Smart–Worte allergisch sind, lesen Sie lieber nicht weiter `;-)`

```
: STATE-SMART ( x -- ? )
   STATE @ IF POSTPONE LITERAL THEN ;
: CH
   CHAR STATE-SMART ; IMMEDIATE
```

`CH A` bewirkt auf jeden Fall dasselbe wie `CHAR A` außerhalb von Definitionen, und wie `[CHAR] A` innerhalb von Definitionen. Der Name `CH` ist kurz genug. `C` alleine wäre dagegen meiner Meinung nach zu kurz und daher verwirrend.

Das Vorsetzel CH (plus Leerzeichen) steht *immer* vor dem Zeichen, für das man den ASCII–Code haben will. In dem seltenen Fall, dass das Zeichen nicht unmittelbar folgt, sondern erst später eingelesen wird, muss CHAR verwendet werden (ganz so wie in der Definition von `CH` selbst). In den Augen eines Programmierers ist das natürlich logisch, da es dann ja nicht mehr um ein Vorsetzel geht, sondern um eine ganz andere Funktion.

[CHAR] wird nun nicht mehr benötigt und für denjenigen, der das gern haben möchte, gibt es auch

```
: CTRL
   CHAR 31 AND   STATE-SMART ;  IMMEDIATE
```

```
: FFBASE ( base <name> -- )
  CREATE   ,   IMMEDIATE
  DOES>   BASE @ >R   @ BASE !
  BL WORD COUNT   ['] EVALUATE CATCH
  R> BASE !   THROW ;

DECIMAL 16 FFBASE HX
        10 FFBASE DM
         2 FFBASE BN \ usw.

: STATE-SMART ( x -- ? )
   STATE @ IF POSTPONE LITERAL THEN ;

: CH
   CHAR STATE-SMART ; IMMEDIATE

: CTRL
   CHAR HX 1F AND STATE-SMART ; IMMEDIATE
```

Listing 1: Zusammenfassung aller vorgestellten Worte

Standard

CH hat nichts im Standard zu suchen, da man es in jedem Forth ganz schnell fabrizieren kann. Wenn Sie dafür einen anderen Namen haben wollen, geben Sie ihm einen anderen Namen. Die Arbeitsweise ist flexibel und entspricht der üblichen Forthmanier.

Bei den an das nachfolgende Forth–Wort festangefügten Vorsetzeln liegt die Sache anders. Die können nämlich nicht so einfach je nach Bedarf definiert werden. Stehen sie einem nicht zur Verfügung und will man sie sich machen, dann muss man am Forth–Kern herumbasteln. Stehen sie im verwendeten Forth zur Verfügung, aber will man sie erweitern oder andere Zeichen verwenden, dann gilt dasselbe: Das heißt dann Patchen, soweit man überhaupt herausbekommen kann, wo. Daher also der Versuch, sie in den Standard aufgenommen zu bekommen. Das einfache Hinzudefinieren geht ja nicht.

Sie haben es natürlich bereits erwartet (ich habe darüber schon gesprochen): Auch für die BASE–Vorsetzel gibt es eine einfache Alternative mit lose vorangesetzten Vorsetzeln. Das funktioniert in jedem normalen Forth und ist in der Namensgebung flexibel.

Mal schnell ein anderes BASE

```
: FFBASE ( base <name> -- )
  CREATE   ,   IMMEDIATE
  DOES>   BASE @ >R   @ BASE !
  BL WORD
  DUP FIND STATE @ ( 0<> ) OVER = SWAP 0= OR
  IF   DROP  COUNT   ['] EVALUATE     \ compile, or number?
  ELSE NIP                            \ execute
  THEN CATCH   R> BASE !   THROW ;

DECIMAL 16 FFBASE HX
        10 FFBASE DM
         2 FFBASE BN \ usw.
```

Listing 2: Eine Fassung von FFBASE, die auch bei Worten verwendet werden kann, die den Eingabestrom lesen.

```
: FFBASE ( base <name> -- )
  CREATE   ,   IMMEDIATE
  DOES>   BASE @ >R   @ BASE !
  BL WORD COUNT   ['] EVALUATE CATCH
  R> BASE !   THROW ;
```

FF ist in Holland ein SMS–Kürzel für *mal eben*: W8 FF = *Wacht even* (ausgesprochen: *wacht effe*) = *warte mal eben*, *wart mal 'nen Moment*. Mit FFBASE kann man ganz leicht für jeden gewünschten Wert von BASE Vorsetzel definieren. Das EVALUATE steht in einem CATCH, um nach einem Fehler während EVALUATE das globale BASE wiederherstellen zu können. Falls das für nicht so wichtig gehalten wird, geht es einfacher auch so:

```
  DOES>   BASE @ >R   @ BASE !
  BL WORD COUNT EVALUATE R> BASE ! ;
```

Die Vorsetzel HX DM ...

```
DECIMAL
16 FFBASE HX
10 FFBASE DM
...
```

HX 100 bedeutet 256, unabhängig davon, welchen Wert BASE (gerade) hat. Mit DM wird eine Dezimalzahl eingeführt.

HEX DM 27 . [rtn] 1B ok

Genau wie bei CH funktioniert das auch innerhalb von Definitionen.

```
DECIMAL
: .ASC ( -- )   HX 80 BL
  DO   I HX F AND   0= IF CR THEN
       I EMIT   I .
  LOOP ;
```

HX 80 und HX F werden als 128 und 15 compiliert, HX selbst findet sich natürlich in der compilierten Definition nicht mehr wieder.

Wie weit gehen wir (sie in comp.lang.forth) mit den fest angefügten Vorsetzeln?

Die allererste Reaktion auf den diesbezüglichen Vorschlag las sich im Stile von: Da muss dann aber auch bestimmt ein Vorsetzel für Oktalzahlen her, da das beim Schreiben eines Assemblers mitunter recht bequem ist.

Tja, über binär haben wir auch noch nicht gesprochen und jüngst war ich mit einem Programm beschäftigt, in welchem Siebenerzahlen gut gebraucht werden konnten, und Sechsunddreißigerzahlen... Müssen wir jedes Forthsystem mit Standardabsprachen über fest angefügte Vorsetzel für alle möglichen Basiszahlen überfrachten?

Mit FFBASE ist das kein Problem. Da machen Sie es einfach so, wie Sie es gerade benötigen.

Gratisdreingabe bei FFBASE–Vorsetzeln

Ganz nebenbei lassen sich die FFBASE–Vorsetzeln auch für Forth–Worte verwenden, die Zahlen ausgeben (nur interaktiv):

```
DECIMAL
10 HX .  [rtn] A  ok
-1 HX U. [rtn] FFFF  ok
```

Folglich also auch für das oben definierte .ASC

```
HX .ASC  [rtn]
    20 !21 "22 #23 $24 %25 &26 '27 (28 ...
DM .ASC  [rtn]
    32 !33 "34 #35 $36 %37 &38 '39 (40 ...
```

Willkommene Extraerscheinung. Sagen wir uns also auch gleich los von den ach so bequemen Worten wie .HEX H. .BIN usw.

Bemerkung: HX und DM akzeptieren keine Worte, die den Eingabestrom lesen.

Die Lesbarkeit

HX 100 liest sich besser als $100, davon bin ich überzeugt. Aber ich sehe auch ein, dass jemand, der jahrelang $100 verwendet hat, mit HX 100 niemals einverstanden sein wird. Gerade so, wie die Tradition das oben genannte Problem mit dem –dt (in der holländischen Sprache) am Leben erhält. Es wird immer Menschen geben, die sagen, dass das Einführen der Schreibweise *hij word* nicht möglich ist, "weil es *hij wordt* heißen muss". Es kommt halt, so denke ich, darauf an, was man gewohnt ist. Wie hat Kemal Atatürk es 1928 nur geschafft, dass ein ganzes Volk von der arabischen zur lateinischen Schrift überging?

Listing 1 auf der vorherigen Seite zeigt noch einmal alle hier eingeführten Forth–Worte zusammen.

Und zu guter Letzt zeigt Listing 2 auf der vorherigen Seite noch FFBASE in einer Fassung, die auch bei Worten verwendet werden kann, welche den Eingabestrom lesen.

Leserbriefe (Fortsetzung von Seite 11)

Aus comp.lang.forth gepickt (2007-10-26) und übersetzt:

Starting–Forth–Ausgaben

MatthewK:
Gibt es größere Unterschiede zwischen der ersten und zweiten Auflage von *Starting Forth*? Ich habe ein Buch, das nach der ersten Auflage aussieht, für ganz kleines Geld erstanden. :)

Elizabeth rather:
Mit der zweiten Auflage sollten die Änderungen dargestellt werden, die Forth so um die 1980iger Jahre hin zum Forth83 Standard durchgemacht hatte. Keine der beiden Auflagen repräsentiert den heutigen Gebrauch (der dem ANS Forth 94 folgt), trotzdem sind beide Bücher reizvoll, weil sie Forth–Konzepte aus einer sehr elementaren Sicht zeigen. Dass es eine erste Ausgabe ist, macht es aber nicht besonders wertvoll.

Wenn du Forth interessant findest, gibt es da modernere Bücher. Auf www.forth.com bieten wir die Fibel *Forth Application Techniques* als Tutorial zum Einstieg an, mit vielen Übungen und Beispielen darin. Und das *Forth Programmer's Handbook*, ein gründlicher Referenztext. Tschüß, Elizabeth

(Anmerkung: Die Bücher sind über Amazon USA oder UK zu beziehen. mka)

Ged Byrne fragte weiter nach:
Welche Forth–Implementierung würdest du empfehlen, um das Buch auf einem Apple Mac (Power PC) durchzuarbeiten?

Elizabeth:
Gforth http://www.gnu.org/software/gforth/ ist eine gute Allzweck–Forth– Implementierung, ANS–Forth–kompatibel, die auf dem Mac läuft.
Cheers, Elizabeth

(Anmerkung: Übrigens ebenso unter Windows; mka)

weitere Leserbriefe und Meldungen auf Seite 26

Ramanujans bemerkenswerte Taxi–Nummer 1729 und schnelle Forth–Lösungen

Henry Vinerts, SVFIG USA

Aus den E–Mail–Korrespondenzen des Jahres 2005 herausgesucht und übersetzt von Fred Behringer
(Es ging um die Übertragung nach Turbo–Forth, dem von Fred bevorzugten 16–Bit–Forth. Beispielsweise entspricht das von F83 her bekannte und überaus brauchbare schöne Wort `DARK` dem Wort `PAGE` aus ANS–Forth.)

27.9.2005
Hallo Fred, da bin ich wieder!
Die Zeit ist für mich wie im Flug vergangen, wahrscheinlich deswegen, weil inzwischen alles etwas langsamer abzulaufen scheint. Meiner neuen Hüfte geht es gut. Ich stütze mich noch auf einen Stock, aber nur, um sicher zu gehen, dass ich die Hüfte nicht zu stark beanspruche. Kürzere Strecken kann ich schon wieder ohne Stock gehen und das Autofahren gelingt auch immer besser. Nur zum Arbeiten mit Forth bin ich eine Weile lang nicht gekommen. Ich will dir aber heute über ein Problem berichten, das mich seit einiger Zeit beschäftigt.

Es geht um ein schnelles Forth–Programm zur Lösung eines Problems, das von Hardy und Ramanujan um die Taxinummer 1729 berichtet wird. Es handelt sich um Summen von Kubikzahlen. Die Geschichte soll sich wie folgt zugetragen haben:
Hardy war mit einem Taxi zu Ramanujans Haus gefahren und hatte sich darüber ausgelassen, dass das Taxi, mit dem er hergekommen sei, eine ziemlich uninteressante Nummer habe, nämlich 1729. Ganz im Gegenteil, soll Ramanujan geantwortet haben: Die Zahl ist sehr bedeutungsvoll. Sie ist die erste Zahl, die auf zwei verschiedene Arten als Summe zweier Kubikzahlen dargestellt werden kann.

```
1   \ Art Hampton's Ramanujan's cube program of 10/8/93
2   \ transcribed into TF83 by VHV, 8/29/05
3   2VARIABLE pair1
4   2VARIABLE sum1
5   : prcubed ( --) ." cubed " ;
6   : cubed   ( n -- D) DUP DUP * UM* ;
7   : testcubes ( --)  pair1 2@ 7 +     \ magic loop arguments
8      DO I cubed
9         pair1 2@ DROP I              \ yet again magic!
10          DO 2DUP I cubed D+  sum1 2@
11            D= IF  CR
12              pair1 2@ 2 U.R prcubed ." + " 2 U.R prcubed ." = "
13              I 2 U.R prcubed ." + " J 2 U.R prcubed ." = "
14              sum1 2@ 5 UD.R
15               LEAVE
16               THEN
17          LOOP 2DROP
18     LOOP ;
19
20  : sumcubes ( --)      DARK
21    18  1                       \ magic numbers found by testing
22    DO I  pair1 !                \ save the first integer
23       I cubed                   \ the first integer cubed
24         41 I 11 +               \ more magic loop arguments
25           DO I pair1 2+ !       \ save the second integer
26             2DUP I cubed D+     \ the sum of the cubes of the first pr
27             sum1 2!             \ to pass into variable is easier than on the stack
28             testcubes
29           LOOP 2DROP
30    LOOP  CR ;
```

Listing 1: Lösung des Taxi–1729–Problems für F–PC

```
1   decimal                                          \ CUBES.FTH, transcribed for TF83
2
3   : sums2                                          \ Jerry Mueller's second routine
4    array 32000 0 fill                              \ use as byte-sized array
5    41 1
6    do i 1+ 1
7       do i dup dup * *
8          j dup dup * * + 32000 mod
9          array + dup c@
10            if drop i j                           \ to print if necessary
11               i dup dup * um*                    \ calculate current sum and
12               j dup dup * um* d+                 \ leave it on stack
13               1 j 1 -                            \ set up
14                  do 1 i                          \ reverse loops
15                      do 2dup
16                         i dup dup * um*
17                         j dup dup * um* d+ d=    \ compare ALL previous sums
18                            if i j cr 5 .r 5 .r   \ with current sum, print all
19                               2dup 8 ud.r        \ if equal
20                               2over 5 .r 5 .r
21                            then -1
22                      +loop -1
23                +loop 2drop 2drop
24            else 1 swap c!
25            then
26      loop
27   loop ;
```

Listing 2: Eine andere Lösung des Taxi–1729–Problems für Turbo–Forth

Paul Snyder präsentierte den SVFIG–Mitgliedern im September 1993 das Problem, die ersten 10 Zahlen mit der eben genannten Eigenschaft herauszufinden. Sieger sollte derjenige sein, dessen Forth–Routine die Antwort in der kürzesten Zeit findet. Drei Wettbewerbsteilnehmer fanden die richtige Antwort. Absoluter Sieger war Art Hampton mit dem unten stehenden F–PC–Programm. Er machte dabei einige logische Annahmen und setzte mathematische Tricks ein, um übermäßig starke Berechnungen zu vermeiden.

Gib `SUMCUBES` ein und lass dich überraschen. Die Antwortzahlen kommen in ungeordneter Reihenfolge heraus. Aber das ist ja bei nur 10 Zahlen nicht gar so schlimm. Ich habe mir nun, getreu unserer seit einiger Zeit geführten Diskussionen, den Spaß erlaubt zu überprüfen, ob Arts Programm auch in Turbo–Forth läuft. Es läuft, es läuft — ohne die geringste Änderung. Und hier das Programm: [Listing 1 auf der vorherigen Seite]

Ein paar Tage später:

Hallo Fred, und noch einmal ich!

Du hast mir verschiedene Fragen gestellt. Ich will versuchen, sie zu beantworten — soweit ich Antworten dazu finde.

Welche Forths sind zugelassen? — Ich denke, dass ich dich (und Friederich) fragen muss, für welche Systeme das Problem bei den heutigen Forth–Programmierern auf Interesse stoßen würde. Damals, als Paul Snyder das Problem in der SVFIG bekannt machte, arbeiteten die meisten Forthler mit F–PC. Einer der drei jedoch, die Lösungen einreichten, Jerry Mueller, programmierte das Problem in einem Upper–Deck–Forth der Version 2.00 aus dem Jahre 1991. Ich habe den Eindruck, dass es umso interessanter sein könnte, die Programme miteinander zu vergleichen, je mehr verschiedene Forth–Versionen zugelassen werden. Was mich betrifft, ich würde mich dafür interessieren, wie elegant man das Problem mit einem Forth–System von minimaler Größe lösen kann.

Welche Maschinen sind zugelassen? — Paul Snyder beschränkte den Wettbewerb nicht auf irgendwelche Maschinen.

Wie sollen die Zeiten verglichen werden? — Ich meine nicht, dass die Zeitfrage bei den heutigen schnellen Maschinen eine wesentliche Rolle spielt. Zwei Programmierer verwendeten die F–PC–Worte `TIME-RESET` und `.ELAPSED` am Anfang und am Ende der Colon–Definition, die das Ergebnis lieferte. Ich kann die entsprechenden Worte in Turbo–Forth nicht finden. Also habe ich die Zeitmessfunktionen aus Art Hamptons Programm, das ich dir in einer Turbo–Forth–Fassung geschickt habe, weggelassen. Sein Programm umfasst 29679 Bytes. Als ich sein `SUMFINAL.EXE` auf meinem AMD–500–MHz–Desktop laufen ließ, wurde die Lösungszeit mit 0.05 Sekunden angegeben.

Dürfen die Routinen auch Forth–Assembler enthalten? — Ich weiß nicht so recht, was ich sagen soll. Wenn wir

uns dazu durchringen könnten, die Lösungszeit als sekundär zu betrachten und mehr Wert auf clevere Programmierung mit minimalem Speicheraufwand zu legen, wäre das heutzutage nicht wichtiger?

Ein Verbot von Assembler könnte ich dadurch umgehen, dass ich Teile von bestehenden *Primitives* kunstvoll in ein High–Level–Forth–Wort einbaue. Sind solche Tricks erlaubt? — Auch hierzu finde ich keine Antwort. Ich bin sicher, du findest geeignete Regeln selbst. Du kennst den State–of–the–Art und weißt, für was sich die VD–Leser am meisten interessieren.

Ich schicke dir hiermit also `CUBES.FTH`, das ich von Jerry Muellers Lösung der Kubikzahlen von Ramanujan entnommen habe. Nach Entfernung der Zeitmessfunktionen lief das Programm ohne irgendwelche Veränderungen auch in Turbo–Forth. Jerry schrieb es ursprünglich in Upper–Deck–Forth und passte es an F–PC an. Von dort übernahm ich den folgenden Teil für Turbo–Forth. [Listing 2 auf der vorherigen Seite]

Das, Fred, stellt meine momentane Beschäftigung mit Turbo–Forth dar. Durch die dritte Lösung (von Paul Snyder selbst) konnte ich mich bisher noch nicht durcharbeiten. Man hat nicht den Eindruck, dass sie kompakter als die beiden anderen ist. Meine eigene Lösung war seinerzeit in AutoLISP programmiert und verschlang viele Stunden Rechenzeit auf einer 80386/7–Maschine.

Ich danke dir, Fred, für die Erklärungen zum An– und Ausschalten des Cursors. Im Augenblick ist das alles, was ich wissen wollte. Ich habe nur selten Veranlassung gehabt, in die Tiefen des BIOSses hinabzusteigen. Das Beste, was ich in dieser Hinsicht je zustande brachte, war ein Prográmmchen, das automatisch CAPSLOCK beim Booten von PygmyForth in die Wege leitete. Das hat damals, soweit ich mich erinnere, auf Frank Sergeant Eindruck gemacht.

Für heute verbleibe ich mit den besten Wünschen,

Henry

Der M16C von Renesas lernt Forth

Albert van der Horst

Mit freundlicher Genehmigung der HCC–Forth–gebruikersgroep aus dem Niederländischen übersetzt von Fred Behringer. Das Original erscheint oder erschien etwa zur gleichen Zeit im Vijgeblaadje 65. Der Autor des vorliegenden Artikels war maßgeblich an der Entwicklung des elektronisch angesteuerten Metall–Xylofons Tingel–Tangel beteiligt. Wir, die aktiven Besucher der Jahrestagungen der Forth–Gesellschaft, haben 2004 auf Fehmarn die Bekanntschaft des Tingel–Tangels machen dürfen, als Willem Ouwerkerk und Albert Nijhof, die bei uns zu Gast waren, eine Vorführung veranstalteten (Bericht im VD–Heft 3/2004). Die Vorführung schlug ein wie eine Bombe. Alle waren begeistert und gleich zum Linux–Tag im anschließenden Jahr, wo auch die Forth–Gesellschaft einen Stand unterhielt, wurde auf Einladung des Direktoriums das Tingel–Tangel als *Kassenmagnet* mitgenommen, diesmal vorgeführt von Albert Nijhof und dem Autor des vorliegenden Artikels. Ganz beiläufig wurden die linux–interessierten Besucher so auch auf Forth aufmerksam und wir konnten einen Zuwachs an FG–Mitgliedern verzeichnen. Der Autor ist in Forth–Kreisen seit Längerem als Entwickler von ciForth bekannt.

Einleitung

Die meisten von uns führen ihre Entwicklungen auf demselben Computer durch, auf dem dann auch das Programm läuft. Bei eingebetteten Systemen ist das aber längst nicht selbstverständlich. Sie sind zu klein, um darauf auch noch ein Entwicklungssystem mit grafischen Möglichkeiten und einen optimierenden C–Compiler laufen zu lassen.

In den 70–er Jahren wurden winzige Systeme als Entwicklungssystem eingesetzt, z.B. mit sagen wir 8 kByte RAM und einer 100–kByte–Floppy. Eine solche Technik ist auch heute noch im Einsatz. Ein Beispiel dafür ist der BASIC–Interpreter in den Lego–Roboter–Bausteinen. Aber der beste Repräsentant dieser Technik heißt Forth.

Da hat man dann also ein ganz kleines Betriebssytem auf seinem eingebetteten System mit einem Befehls–Interpreter. Das ist ein großer Vorteil, wenn man unverlässliche, in Entwicklung befindliche Hardware verwendet, so z.B., wenn man sich selbst was zusammengelötet hat. Aber auch in der Industrie zählt dieser Vorteil. Schlechte Dokumentation hat übrigens dieselbe Wirkung wie unzuverlässige Hardware.

Demonstration einer Ansteuerung

Auf dem Treffen der Forth–gebruikersgroep am zweiten Samstag im vergangenen Oktober habe ich mit einem Einplatinen–Computer von Renesas ein Steuerungsproblem vorgeführt. Das Tingel–Tangel (ein Metall–Xylophon) spielte die Big–Ben–Melodie und schlug die Stunde. Auf dem Sig–Embedded–Treffen von Cimsolutions die Woche zuvor war diese Demonstration noch nicht gelungen. Immerhin wurden die Primzahlen bis zu 100.000 berechnet: Der Compiler arbeitete gut. Auch konnte ich hoffentlich vermitteln, welche Vorteile ein interaktiver Interpreter mit sich bringt: Mit einfachen Befehlen wurde nachgeprüft, dass die Leitungen zum Tingel–Tangel korrekt angelötet waren. Ich denke, dass man die Geschwindigkeit beim Entwickeln mit Forth in der Tat als einen Vorteil betrachten kann.

Die Ansteuerung mit den Timings und Ähnlichem ist nicht trivial, hat aber nicht mehr als nur einen Abend

Arbeit gekostet (wobei allerdings das Löten nicht mit eingerechnet ist).

Dies ging voraus.

Es ist nun schon wieder zwei Jahre her, dass die europäische (ursprünglich niederländische) Elektronik–Zeitschrift Elektor ihrem Dezemberheft 2005 eine Mikro–Computer–Platine als Geschenk beilegte, ein Reklame–Bravourstückchen des Renesas–Importeurs Glyn. (Der Übersetzer: Ich kann mich noch gut daran erinnern: Heinz Schnitter hatte uns darauf aufmerksam gemacht, als es schon fast zu spät war. Sämtliche Zeitschriften–Kioske waren elektor–ausverkauft. Nach vielem Herumlaufen wurde ich schließlich in Vaterstetten in der Umgebung von München (j.w.d.) fündig. Die wussten nicht, dass sie auf einem Schatz saßen.) Renesas, vormals Mitsubishu (wichtig für die Suche nach Informationen), ist der Marktführer auf dem Gebiet der mittelgroßen flash-basierten Mikro–Controller. Wir haben es hier also nicht mit Kleinstcomputern zu tun, die eine flackernde Leuchtdiode ansteuern (künstliche Wachslichter mit eingebautem Computer, 2 für 1 Euro 50, inklusive Lithium–Batterie[1])), sondern mit etwas größeren Systemen.

Für die anstehenden 10 Euro wollte ich da gern noch mehr davon wissen und ich machte mich daran, meinen Forth–Compiler (ciforth) auf diesen Chip zu portieren. Der Befehlssatz und die übrigen Eigenschaften stimmten mich enthusiastisch. Mit dem (gratis!) mitgelieferten Entwicklungspaket lief der Compiler schon unter dem Emulator recht ordentlich. Für eine serielle Verbindung zum Computer war noch etwas Hilfselektronik nötig. Als das nicht auf Anhieb hinhaute, beschloss ich, gleich mit einem größeren Evaluation–Board, das von Glyn für 50 Euro komplett geliefert wurde, ans Werk zu gehen. (Ich danke hiermit Cimsolutions für die Vermittlung bei der Anschaffung: Glyn scheute sich davor, an privat liefern. Da habe ich dann zehn davon kommen lassen, von denen noch einige weitergegeben werden können. Inklusive CD–ROM und seriellem Kabel.) Hiermit habe ich dann losgelegt, mit dem Ergebnis, dass ich jetzt ein ausgetestetes, dokumentiertes und vor allem brauchbares Entwicklungssystem habe.

Flash–basierte Mikro–Controller

Was enthält nun so ein mittelgroßer Controller heutzutage? Zunächst einmal kommen sie mit einer schwindelerregenden Vielfalt daher, was die eingebauten Steuerungsmöglichkeiten betrifft. Normale digitale E/A, um z.B. etwas anzuschalten, AD–Wandler, um z.B. einen Sensor auszulesen, programmierbare Timer, spezielle Hardware zum Ansteuern von Motoren, Schnittstellen zum CAN–Bus, wie er in Autos verwendet wird, ja man kann jederzeit einen Chip finden, der genau das tut, was man haben will, und nicht mehr. *Nicht mehr*, was natürlich vom Kostenstandpunkt aus gesehen sehr wichtig ist. Dieselbe Vielfalt findet man in Bezug auf den Speicher. Da geht es von 48 kByte Flash bis in die Megabytes. Diese kleinen Computer werden im Allgemeinen als Turnkey–Geräte verwendet. Man denke an Bordcomputer im Auto. Man schaltet sie ein und sie beginnen ihre Arbeit. Die Renesas–Prozessoren können Programme im Flash direkt ausführen.[2]) Das RAM ist dann nur für Daten da. Mein Evaluation–Board hat 32 kByte RAM und 512 kByte Flash an Bord.

Was wollen wir eigentlich tun?

Ein ausgewachsenes ciForth unter Linux belegt so an die 30 kByte[3]) und die Quellbibliothek beläuft sich auf einige Hundert kByte. Ein ciForth–System möchte gern im RAM laufen und die Bibliothek passt prima ins Flash. Mit anderen Worten, das Evaluation–Board erfüllt reichlich die Anforderungen, die man stellen muss, um ein Entwicklungssystem aufzubauen. Worauf wollen wir hinaus? Wir wollen die serielle Verbindung zur Kommunikation mit einem Forth verwenden, das auf der Platine läuft. Das ist das uralte Konsolen–Modell: Man tippt was ein, der Computer führt es aus, liefert seine Antwort ab und meldet sich mit einem Prompt. Sowas nennt man häufig eine DOS–Box[4]). Dann kann man beispielsweise mit ein paar interaktiven Befehlen austesten, wie man seinen Prozessor achtmal so schnell laufen lassen kann, ob man seine LEDs überhaupt richtig angeschlossen hat, was der Temperatursensor meldet usw. Zu alldem ist es nötig, dass das Forth–Programm beim Starten (aus dem Flash heraus, anders geht es nicht) erst ins RAM kopiert und dort dann ausgeführt wird.

Schwierigkeiten beim Entwickeln

Das Entwicklungssystem von Renesas (in Visual–Studio–Aufmachung) wurde unter Windows 98 SE installiert, da ich kein XP–System mit dem erforderlichen seriellen Anschluss habe. Aber damit funktionierte es dann auch auf Anhieb. Es wurde jedoch ganz schnell deutlich, wie stark C–orientiert das System ist. Es traten verschiedene Schwierigkeiten auf, die teilweise im Kontakt mit Glyn beseitigt werden konnten. Ärgerlich war auch das Extra–System, das nötig war, um über den zweiten seriellen Anschluss mit Forth kommunizieren zu können. Als es sich auch noch herausstellte, dass nach dem Laden einige RAM–Adressen systematisch mit 0FF belegt wurden, war das Maß voll. Ich erfuhr, dass es für diesen Chip einen Flasher für Linux gab und dass auch die Version 2.0, die freigegebene DOS–Version des Assemblers, unter dosemu, dem DOS–Emulator von Linux, ausgezeichnet lief. Das ciForth–Entwicklungssystem, das

[1] Ja, das ist auch ein eingebettetes System, und nein, es läuft nicht unter Windows CE.

[2] Populär ausgedrückt: Es sind PCs mit nur BIOS und keiner Festplatte.

[3] Beta 5.18 : 28.542 Bytes, um genau zu sein (und alles sitzt drin, nix mit dynamisch gelinkten Bibliotheken).

[4] Abfällig. Computer–Fachleute verwenden den Ausdruck *Console–Window*. Selbst Microsoft kommt übrigens zur Einsicht, dass für Computer–Fachleute ein Konsolenfenster unverzichtbar ist. Man google sich durch nach *Powershell*.

die Assembler–Datei erzeugt, im Verein mit Dokumentation und Testen, läuft natürlich unter Linux. Windows wurde definitiv als Entwicklungssystem verworfen. Es beunruhigte natürlich sehr, dass die eigentliche Struktur eines C–Programms unter niederen h–Dateien verborgen blieb, was um Himmels willen das Monitor–Programm im Flash[5]) alles tat, wie die ganze Interrupt–Struktur da eigentlich aussah, wie das mit dem id–Code eingerichtet war, den man fürs Flashen benötigt, und dann die Restart–Vektoren — neben den Interrupts — Watch–Dogs, nicht–maskierbare Interrupts, Single–Step–Gebahren ...

Bevor es dann richtig losgehen konnte, gab es noch ein Problem. Das Programm muss im RAM laufen, aber im Flash aufbewahrt werden. Das Renesas–Entwicklungssystem, aber auch der Linux–Flasher konnten damit nicht umgehen. Vom Linux–Flasher stand der Quelltext zur Verfügung, so dass ich nach einem Wochenende an harter Arbeit einen Flasher in Forth geschrieben hatte. Dieser Flasher hat noch einen anderen wesentlichen Vorteil. Er löscht nur die Blöcke, die er beschreiben will, so dass ich die Bibliothek und das Forth–System unabhängig voneinander flashen kann.

Es beginnt zu funktionieren.

Das erste Programm, zum Anschalten einer LED, war drei (Assembler–)Befehle groß. Des Weiteren musste nur noch der Reset–Vektor so gesetzt werden, dass er auf dieses Programm zeigt. Kurz und gut, alle Lösungen waren simpel. Interrupts? Aus! Id? Auf null! Monitor? Diesen Happen dumpen! Restart: Den Reset setzen. Das war's! Und ... es funktionierte tatsächlich. Jetzt mal schnell etwas einbauen, das dieses 3–Befehls–Programm sich selbst ins RAM kopieren lässt, so dass es dann da laufen kann. Da ging zunächst einmal nichts.[6]) Nach dem vierten Anlauf lief es dann schließlich. Nun denselben Spaß in Forth einbauen und das Debuggen konnte beginnen. Zwei Stunden später konnte ich die Start–up–Botschaft von ciForth bewundern: `R16C ciforth beta $RCSfile: cir30.gnr,v $ $Revision: 1.33 $`. Das bedeutete zugleich, dass das Ausführen von Tests möglich war. Jetzt ging es etwas besser voran. Der Arbeitszyklus bestand aus flashen, Jumper umstecken, erneut starten, Kermit aufrufen. Kermit ist ein *Terminal Emulator*, ein Programm, das sich über eine serielle Schnittstelle mit Forth unterhält. Jetzt kann ich also ein und dieselbe serielle Schnittstelle benutzen. Es ist kein zweites System und kein zweites Kabel mehr nötig.

Testen, testen.

Nun konnte damit begonnen werden, die 328 Blöcke, aus denen dieses Forth besteht, zu debuggen. Viele Blöcke sind zum Glück systemunabhängig und bereits ausgiebig getestet, aber 75 Blöcke sind in Assembler geschrieben und enthalten unvermeidlich Fehler. Forth ist interaktiv, höchst modular und hat viele eingebaute Debug–Möglichkeiten. Zehn Tage, nachdem das LED–Programm lief, konnte ich bereits den Hayes–Test — eine Art Validation der ISO–Aspekte eines Forth–Compilers — auf mein Forth loslassen, wodurch noch zwei Bugs aufgespürt wurden.

Sowohl Kermit wie auch ciForth sind flexibel. Nach Anpassungen des Prompts von ciForth, der dann wieder von Kermit verstanden wird, kann der Test völlig automatisch laufen, bis hin zur Überwachung der Ausgabe. Und für die Experten: Auch alle Ausnahmefälle (exceptions) werden getestet. Das geht, weil der Test über den Interpreter läuft und das System bei einer Ausnahmesituation im Interpreter landet. Automatisch testen ist vielleicht ein von allerlei Marketing–Gags ausgehöhlter Begriff. Unter automatisch verstehe ich Folgendes

```
albert@apple ~/renesas>make test
albert@apple ~/renesas>echo $?
0
albert@apple ~/renesas>
```

Der Befehl `make test` hat kein Ergebnis geliefert, sehr wohl jedoch einen Return–Code von 0, d.h. okay. Spaßeshalber frage ich den hier explizit ab. Die Struktur ist also so beschaffen, dass dieser Test selbst wieder in einem anderen Test als Baublock verwendet werden kann, und so gehört es sich auch. Das ciForth–Stammprogramm enthält für jeden Baublock neben dem Programm–Code und der Modulbeschreibung auch einen oder mehrere Tests. Jeder Test besteht aus einem Befehl und dem erwarteten Ergebnis. Das alles zusammen bildet den endgültigen Test und der wird neben Hayes auf den Befehl `make test` hin ausgeführt.

Die Verwendung des Flash–Speichers

Der Controller auf dem Renesas–Board verfügt über 7 Bereiche von 64 kByte, die getrennt geflasht werden können, 1 von 32 kByte, und dann noch 3 von 8 kByte und 2 von 4 kByte. Der oberste Bereich von 4 kByte erklärt der Platine, wo sie anzufangen hat, und muss immer belegt werden. Er zeigt auf das Startprogramm und an das kommen wir nicht mehr heran. 2 Bereiche von 8 kByte enthalten zusammen das Forth–System, 4 kByte bleiben übrig. Das erste Stück davon ist das Startprogramm, das das besprochene Forth–System ins RAM kopiert. Aber zuerst wird nachgeprüft, ob der Bereich von 32 kByte noch jungfräulich ist, was man nach dem Flash–Säubern an den 0FFs ablesen kann. Wenn nicht, dann ist dieser Bereich von einem Programm belegt. Das kann sowohl ein erweitertes Forth sein, wie auch ein Programm, das etwas Nützliches tut, wie z.B. das Abspielen von Big–Ben oder ein ferngesteuerter Mehrkanal–Lichtdimmer. Zwei Bereiche von 64 kByte enthalten die Forth–Bibliothek. Das lässt uns noch 5 Bereiche von 64 kByte und noch ein paar kleinere Bereiche.

[5] Wo das Windows–Entwicklungssystem mitspricht.

[6] Mehr erzähl ich nicht. Wer es mir nicht glaubt, der probiere sowas mal selbst aus.

Die Bibliothek

Zwei Bereiche von 64 kByte enthalten ein klassisches Forth–Block–System. Das bedeutet, dass es in Blöcken von 1 kByte mit Quelltext unterteilt ist, die allesamt getrennt voneinander eingeladen werden können. In herkömmlicher Weise enthält jede erste Zeile ein Kommentar mit dem Befehl, der hinzugefügt wird, sowie ein bisschen Buchhaltungsinformation. Das wird in ciForth über den Befehl WANT als eine *On–Demand–Bibliothek* verwendet. So sorgt `WANT class` dafür, dass der Befehl `class`, eine Erweiterung mit objekt–orientierten Möglichkeiten, geladen wird. Die Renesas–Bibliothek enthält das meiste von dem, was die ciForth–Systeme unter Linux und Windows auch haben:

Generische Algorithmen: Binärsuche, Quicksort, Mergesort Code–Untersuchung: Decompiler, Aufsuchen der Wortquellen Objekte: Definieren von Klassen und Methoden Beispiele und Benchmarks: Primzahlensieb, Verschachtelungsbenchmark, Byte–Benchmark, Automatisierung des Entwicklungskreislaufs Nützliche Tools: Speicherdump, Umschalten auf case–insensitive, `SAVE-SYSTEM` , `TURNKEY`

Was fehlt, ist der Pentium–Assembler, der hier keinen großen Nutzen hat. Auf die Werkzeuge `SAVE-SYSTEM` und `TURNKEY` gehen wir jetzt noch etwas näher ein.

Das Tingel–Programm als Turnkey

Die Quelle des Tingel–Programms ist in einem der freien Flash–Bereiche untergebracht, im Block–Stil. Mein Flash–Programm kann auch diese Bereiche beschreiben, so dass der Quelltext auf Wunsch unter Linux gehalten bleiben kann. (Hatte ich schon erzählt, dass mein Flasher mit Quelltexten genauso leicht umgehen kann wie mit Code?). Auf diese Weise kann es mit den gebräuchlichen Befehlen (`THRU`) eingeladen werden. Danach haben wir dann ein Forth, an das u.a. Befehle wie big ben boing big–ben angefügt wurden. Nach dem Eintippen von `12 big-ben` und bei angeschlossenem Tingel–Tangel wird dann also die Big–Ben–Melodie gespielt und anschließend erklingen 12 Glockenschläge. Der Befehl `SAVE-SYSTEM` bereitet den 32–kByte–Flash–Bereich sauber vor und legt dort unser neues System komplett mit Big–Ben hinein. Wenn wir am Morgen das Evaluation–Board wieder anschalten, können wir direkt `6 big-ben` eintippen, da anstelle des gewohnten Forth–Systems das erweiterte System eingeladen wird.

Der Befehl TURN–KEY geht noch einen Schritt weiter. Hierbei startet Forth nicht mehr mit dem Befehls–Interpreter, sondern mit einem mitzugebenden Befehl wie beispielsweise bei `SPOKENUUR TURN-KEY`. Der Befehl `SPOKENUUR` (zu Deutsch: Geisterstunde) ist dabei als `12 big-ben` definiert. Jedesmal wenn das System angeschaltet wird, ertönt nun die *Geisterstunde*.

Zusammenfassung

Die heutigen Mikro–Controller sind kräftig genug, um einfache Entwicklungssysteme mit aufzunehmen. Entwickeln auf dem Zielsystem hat gewaltige Vorteile, vor allem beim Hardware–Debuggen. Der Einsatz von Flash–Speicher erlaubt es, ohne spezielle Apparatur, wie EPROM–Programmierer, Turnkey–Systeme zu entwickeln und diese auch schnell weiter anzupassen. Evaluation–Boards sind dermaßen preiswert, dass sie als Turnkey eingesetzt werden können. Für 53 Euro zuzüglich Versandkosten können bei mir noch einige ECBM16C/62P–Evaluation–Boards bezogen werden.

Die SWAP–Bronze von Rolf Kretzschmar

Forth von der Pike auf — Teil 9

Ron Minke

Die mit freundlicher Genehmigung der HCC–Forth–gebruikersgroep in der Vierten Dimension in einer Übersetzung von Fred Behringer wiedergegebene achtteilige Artikelserie erschien ursprünglich in den Jahren 2004 und 2005 in der Zeitschrift *Vijgeblaadje* unserer niederländischen Forth–Freunde. Die Firma Atmel hat inzwischen größere Bausteine ihrer Serie entwickelt. Das reizte den Autor dazu, sein Projekt auf einen größeren Mikro–Controller zu übertragen. Auch die vorliegende Übersetzung aus dem Niederländischen stammt von Fred Behringer.

Hier kommt nun Teil neun der Wiedergabe des Versuchs, ein AVR–Forth–System mit der Voraussetzung *from scratch* zu erstellen.

Die Hardware–Entwicklung steht nicht still. Also wurde auch für die Hardware–Basis, auf welcher *Forth von der Pike auf* läuft, ein Update fällig. Und speziell das Modell, Forth ganz im inneren Speicher des Prozessors ablaufen zu lassen, rief nach einer Neufassung. In der Schaltung von Teil 7 dieser Serie war der Prozessor ein ATmega162 mit 1 kB RAM *an Bord*. Wenn wir den Platz, den die Forth–Verwaltung selbst benötigt (Stacks, Terminal–Puffer etc) abziehen, bleibt da für eigene Definitionen ungefähr 0,5 kB übrig. Das ist nicht viel, aber es reicht für ein paar Experimente.

Forth goes MEGA

Inzwischen hat die Firma Atmel eine Anzahl von *großen* Chips auf den Markt gebracht. Für eine bestimmte Anwendung am Arbeitsplatz (in der Sprache C) wurde ein solcher *großer Chip* benötigt. Die Wahl fiel auf den größten, den Atmel zur Zeit (Sommer 2007) liefern kann, den ATmega2560/ATmega2561. Intern sind beide Versionen gleich, am 2560 sitzen jedoch 100 Beine und am 2561 *nur* 64. Jedenfalls groß genug... Was haltet ihr von 256 kB Flash und 8 kB internem Speicher? Die 256 kB Flash sind nie vollzukriegen (obwohl... eine Webseite in den Chip?) und die 8 kB an internem Speicher können wir für Forth gut gebrauchen. Auf einmal stehen nun 7,5 kB für eigene Definitionen zur Verfügung! Um auf diesem Prozessor ein Forth unterzubringen, müssen am ursprünglichen Quelltext ein paar Anpassungen vorgenommen werden.

Die Version mit innerem RAM

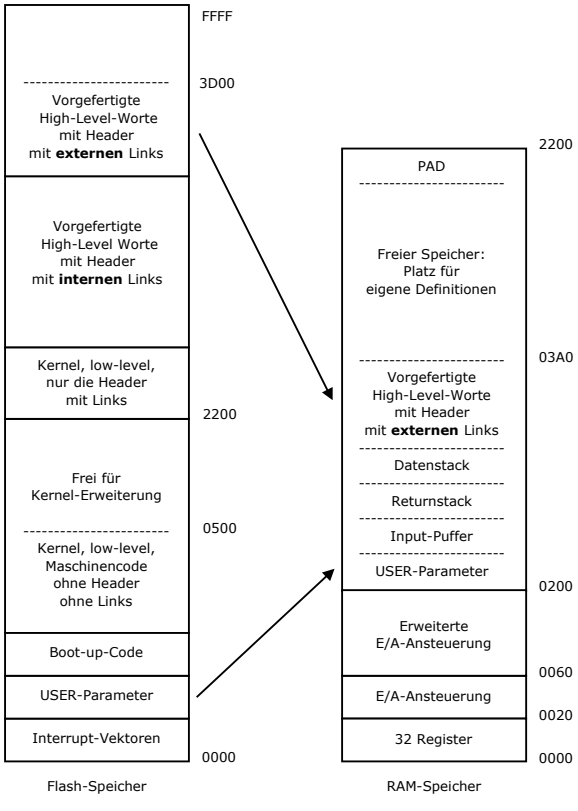
Woraus bestand noch gleich die Version mit innerem RAM? Wenn wir uns den Forth–Quelltext im Speicherplan des ursprünglich eingesetzten ATmega162 genauer anschauen (vergleiche Zeichnung in Teil 8), bemerken wir zwei Dinge:

- Internes RAM beim MEGA162: Von 0100 bis 0500
- Flash: Die Forth–Header im Kernel beginnen bei (ungefähr) 0500.

Wir unterschieden zwischen Gebieten unterhalb der Adresse 0500 und solchen oberhalb 0500. Gebiete unterhalb der Adresse 0500 sahen wir für den Zugang zum internen RAM vor, und oberhalb der Adresse 0500 machten wir von einem speziellen Befehl Gebrauch, um uns den Zugang zum Flash zu sichern. In einer Anzahl von Worten wurde ein Test eingebaut, um herauszufinden, ob die angebotene Adresse kleiner als 0500 ist. Wenn ja, dann handelt es sich um eine *innere* Adresse im internen RAM. Wenn nein, dann ist es eine Adresse im *vorgefertigten* Teil, der nur die Header und externen Links enthält. Das Ergebnis dieses Adressentests zeigt an, mit welchem Befehl die Daten hereingeholt werden müssen. In allen diesen Worten muss der Wert 0500 angepasst und durch einen entsprechenden Wert für den neuen ATmega2561 ersetzt werden. Der Wert beträgt 2200. Er setzt sich aus dem Wert 0200 für die erweiterte E/A–Ansteuerung und der Größe des internen RAMs (8 kB = 2000) zusammen.

Die neue Speicheraufteilung

Im neuen Speicherplan entsteht nun aber im Flashbereich eine Lücke von 0500 bis 2200 (der ursprüngliche Kernel ist ja nicht größer geworden). Aber bei einer Gesamtgröße von 256 kB fällt das überhaupt nicht ins Gewicht. Dieser freigewordene Platz kann für Kernelerweiterungen verwendet werden. Anzumerken bleibt zudem, dass von der Flash–Gesamtgröße nur die ersten 64 kB zum Einsatz kommen. Das um dem so genannten *extended addressing* zuvorzukommen. Der Speicherplan sieht jetzt also wie folgt aus:



Dictionary im Filesystem

Matthias Trute

Zusammenfassung

Es wird ein alternativer Ansatz für die Implementierung von Forth–Dictionarys beschrieben. Es gibt keinen Code oder eine lauffähige Implementierung. Dieser Artikel soll eine Diskussion über Implementierungsstategien von Forth–Systemen anregen.

Ausgangspunkt

Aktuelle Embedded–Systeme bieten die Möglichkeit, Standardbetriebssysteme zu betreiben, namentlich Linux. Anstelle von Festplatten dient hier Flashspeicher als Speichermedium, zusammen mit dafür optimierten Filesystemen. Auf diesen Systemen den alten Forth-Ansatz, alles selbst zu machen, umzusetzen, scheint angesichts der Komplexität der Aufgabe, eine Vielzahl an Treibern und Basisdiensten zu erstellen und in der doch kleinen Nutzergemeinde auch zu testen, nicht empfehlenswert. Es ist sinnvoller, die bereits getane Arbeit für das Betriebssystem auch zu nutzen und die Stärken von Forth auszuspielen.

Üblicherweise ist RAM bei diesen Systemen knapp bemessen. Da es nicht so einfach wie bei einem PC ist, RAM nachzurüsten, sollte RAM als Ressource für die Applikation und nicht für die Hausaufgaben des Systems betrachtet werden. Das geht am einfachsten, wenn man das, was im RAM liegt, und sich nicht wesentlich ändert, anderweitig unterbringt.

Dieses „nicht wesentlich“ ist ein Schlüsselwort. Flashspeicher kann einige Male (i.allg. sind das einige tausend Male) neu beschrieben werden. Damit wird Flashspeicher für alles interessant, was nur einige Male geändert wird. Bei Forth sind dies wesentliche Teile des Dictionaries: Die (Colon–) Worte.

Ein weiterer Aspekt kommt hinzu. Flash ist deutlich schneller als andere Speichermedien namentlich Festplatten. Besonders deutlich ist dies beim Faktor Zugriffszeit. Bis eine Festplatte den Lesekopf positioniert hat, dauert es eine Weile. Flash, wie auch RAM, kennen diesen Parameter (fast) nicht. Beim Datentransfer vom Flash zur CPU ist das geringere Tempo des Flash gegenüber dem RAM zu beachten, spielt aber bei embedded Systemen mit bestenfalls einigen Hundert MHz keine herausragende Rolle.

Das Ziel ist damit umschrieben: Das Dictionary soll im Flashspeicher liegen. Dieser Flashspeicher wird vom Betriebssystem in einem Filesystem verwaltet. Dieses berücksichtigt nebenbei auch den bereits erwähnten Effekt, das Flash nur eine begrenzte Zahl von Löschzyklen machen kann und verteilt die Zugriffe entsprechend.

Direkter Zugriff auf die Blöcke mit dem BLOCKS word set aus dem Standard verbietet sich weitgehend, da ja schon ein Filesystem vorhanden ist. Eine Variante ist ein großes Einzelfile, das mehr oder weniger ein Hauptspeicherabbild des kompletten Dictionarys ist. Dies ist sicherlich einfach umzusetzen.

Eine andere Idee wird nachfolgend skizziert. Dabei wird ausgenutzt, dass ein Dictionary nicht nur ein binärer Klumpen von Bits und Bytes ist, sondern eine wohldefinierte Struktur aufweist, die viele Parallelen zu einem Filesystem hat. Ebenso wird die Funktionsweise des Forth–Interpreters mit einer Shell in Bezug gesetzt.

Strukturen

Das Forth–Dictionary besteht bekanntlich aus einzelnen Vocabularys. In jedem einzelnen ist eine Menge von Worten enthalten, die man in erster Näherung als ausführbare Programme ansehen kann. Um die Vocabularys zu nutzen, wird eine Suchreihenfolge definiert. Zusätzlich gibt es eine Art aktuelles Vocabulary, in das alle neu definierten Worte gelangen.

Wird ein Wort gesucht, werden die Vocabularys in der festgelegten Reihenfolge duchsucht und das erste Wort, das passt wird aufgerufen. Das ist nicht anders bei der Shell. Sie mustert Verzeichnisse gemäß der in der Umgebungsvariablen `PATH` festgelegten Reihenfolge und das erste (ausführbare) Programm wird gestartet.

Legt man gedanklich diese beiden Strukturen übereinander, wird aus einem Vocabulary ein Verzeichnis und aus einem Forth–Wort eine ausführbare Datei.

Der Forthheader wird der Directoryeintrag mit dem Dateinamen und den Attributen, das Datafield wird der Dateiinhalt.

Was Forthworte von Shellbefehlen unterscheidet, ist die gemeinsame Nutzung von zwei (oder mehr) Stacks für die Parameterübergabe.

Auswirkungen

Flags

Das Forth–Dictionary hat einige Flags, die problemlos auf die Filesystempermissionflags abgebildet werden können: Das x–Bit könnte das Smudge–Flag werden (nur mit der umgekehrten Wertbedeutung), Das s–Bit kann für IMMEDIATE genutzt werden, R und W können ihre Bedeutung behalten.

Suchreihenfolge

Mit dem Ansatz, die Worte vom Filesystem verwalten zu lassen, gibt man die Herrschaft über die Reihenfolge, in der die Worte innerhalb eines Vocabularys gesucht werden, auf. Wo ist das ein Problem? Nur bei Worten, die redefiniert wurden. Alle anderen sind nur einmal vorhanden. Und das Redefinieren von Worten ist spätestens seit DEFER/IS als eine veraltete Programmiertechnik

einzustufen. Wer es aber haben will, kann über den Einsatz eines versionierenden Filesystems nachdenken, auch hierfür gibt es lange Erfahrungen (VAX).

Dateinamen und Wortnamen

Forthworte als Dateinamen gehen „fast immer“. Auch wenn es in der Unixshell einiger Vorkehrungen bedarf, um Dateien wie `<#` oder `?` anzulegen:

```
mt@forthy:~/dforth/FORTH$ ls
<#  !  ?  @  #>  abort
mt@forthy:~/dforth/FORTH$
```

Einzig zwei Zeichen sind in Dateinamen nicht zulässig: Das Null-Byte und der Directory–Separator `/`. Ersteres ist keine wirkliche Einschränkung, der `/` schon. Denn damit sind Worte wie `/` oder `UM/MOD` nicht erlaubt.

Dies ließe sich durch einige Tricks umgehen: Das in Forth–Worten nicht zulässige Leerzeichen kann als Ersatz herhalten, oder der Dateiinhalt wird komplexer, was den Dateinamen nicht mehr zwingend zum Wortnamen macht.

Ein dritter Aspekt ist die Groß/Kleinschreibung. Hier wird das Forth vollständig abhängig vom zugrundeliegenden System: Linux–Filesysteme sind in der Regel case sensitive, Windows speichert zwar die Schreibweise, beachtet aber Groß-Kleinschreibung nicht.

Neue Möglichkeiten

Wenn die Forth-Worte nicht mehr dicht an dicht im Dictionary liegen, sondern praktisch unabhängig voneinander sind, ergeben sich neue Programmieransätze.

Forth Worte können zur Laufzeit redefiniert werden und die neuen Worte haben auch Wirkung auf die bereits bestehenden (so als ob jedes Wort mit DEFER angelegt würde).

Ist der Directoryeintrag eine komplexere Datenstruktur (etwa die Geburtstage aus einer der letzten VD), kann er problemlos erweitert werden. Es gibt kein Wort, das durch das Anhängen von Daten an ein bestehendes Wort überschrieben werden könnte.

Ein weiterer Aspekt sind Usernamen und Gruppenrechte. In komplexeren Konstellationen könnten sie zur Absicherung von Bibliotheken genutzt werden.

Implementierung

Wie eingangs erwähnt, gibt es derzeit keine Implementierung. Einige Überlegungen sollen aber dennoch angestellt werden. Hier werden auch mögliche Showstopper diskutiert.

Dateiinhalt

Der Dateiinhalt der Einzelworte ist im Wesentlichen der Inhalt des klassischen Datafields.

Eine weitere Möglichkeit bietet sich für (Just–In–Time–)Compiler. Der Dateiinhalt kann komplexer strukturiert werden, um generierten Maschinencode anstelle oder zusätzlich zu Execution–Tokens vorzuhalten.

Bei den Gerätedateien im Verzeichnis `/dev` sind natürlich die betreffenden Geräte angesprochen.

Execution–Token

Das Execution–Token ist ein offener Punkt. Es wird bei Worten wie `'` (TICK), `EXECUTE`, `FIND` benutzt und ist zugleich Bestandteil des Datafields innerhalb der Colon–Definitionen. Da das Execution–Token sowohl zur Identifikation eines Wortes dient, als auch als „Zwischencode“ vom Compiler verarbeitet wird, ist hier Kreativität gefordert.

Eine naheliegende Idee ist der Einsatz eines Caches, der die Worte in den Dictionarys beim Start analysiert und so eine Art temporäres Execution–Token bereitstellen kann.

Handhabung

Wie kann man mit den Worten umgehen? Ein einfacher Ansatz ist das FILE Extension Wordset. Es bietet alle Möglichkeiten, Dateien zu manipulieren.

Die Umsetzung von Worten wie `:` oder `CURRENT` ist nicht weiter kompliziert, spannender wird `:NONAME`, da es ja keine namenlosen Files gibt.

Interessant werden `,` und `@`/`!`, die (auch) mit den Dictionaryeinträgen operieren. Da sie gleichzeitig benötigt werden, um auf RAM zuzugreifen, sind sie nicht eben trivial. Ein Ansatz könnte der von einigen Forth–lern propagierte Ansatz mit XDATA/UDATA sein, mit dem der Speicherbereich, auf den die erwähnten Worte operieren, umgeschaltet wird.

Housekeeping

Die gerne während der Entwicklung genutzten Worte `FORGET` und `MARKER` müssen auf Dateiattribute zugreifen, um alle Worte zu finden, die es zu entfernen gilt. Alternativ kann `FORGET` zu neuen Ehren kommen: Es kann jetzt gezielt *ein* Wort löschen. Hier ist natürlich zu beachten, das es kein Wort trifft, das von einem anderen Wort genutzt wird ...

Fazit

Dieser Artikel ist als Diskussionspapier gedacht. Die Ideen erheben nicht den Anspruch, zu einem ANS94–konformen System zu führen. Es ist auch noch offen, ob der Ansatz Filesystem als Dictionary wirklich benutzbar ist, Literaturquellen scheint es jedenfalls keine zu geben.

Lebenszeichen

Bericht aus der FIG Silicon Valley: *Henry Vinerts*

Dear Fred,

Der Sommer ist schon wieder vorbei und ich sehe, dass wir seit fast fünf Monaten keine E–Mails mehr ausgetauscht haben. Das tut mir Leid. Aus dem einen oder anderen Grund war ich seit meinem letzten Bericht an dich auf nicht mehr als anderthalb SVFIG–Treffen. Ich hoffe, dass es dir und euch gut geht, und statt den Herausgebern der Vierten Dimension, die ich nicht so gut kenne, regelmäßig Berichte aus dem SVFIG–Geschehen zu schicken, würde ich gern weiterhin mit dir direkt korrespondieren. Das haben wir ja schon seit langer Zeit so gehalten. Und ich würde dich bitten, diejenigen Stellen aus meinen Berichten, die deiner Meinung nach für die VD–Leser von Interesse sein könnten, an die VD–Redaktion weiterzuleiten.

Ich möchte dich aber keinesfalls noch stärker mit der Übersetzung meiner Berichte belasten. Das hast du ja bisher immer so geduldig und gewissenhaft getan. Ich werde also weiterhin, wie wir es vereinbart haben, Berichte von den SVFIG–Treffen schicken, aber mir die Freiheit nehmen, das eine oder andere Treffen auszulassen.

Ich habe gerade wieder damit begonnen, Schulkindern Schach beizubringen, zwei bis drei Mal die Woche, und ich spiele zweimal die Woche Schach und Tischtennis hier bei uns im Zentrum für Senioren. Ich komme kaum noch dazu, das Fernsehen einzuschalten, aber ich finde immer mehr Bücher in unserer heimischen Büchersammlung, die ich noch lesen möchte, bevor ich sie einpacke oder weggebe, falls wir doch noch aus unserem Haus wegziehen sollten. Ich versuche, meine Bücher, Zeitschriften, wichtige Papiere usw. einzuordnen, sozusagen um *Klarschiff* zu machen, aber es will mir nicht ganz gelingen. Ich bin viel zu sehr der ewige Sammler — ein Mensch, der alles aufhebt, was für ihn oder andere nützlich sein könnte. Ich kann mich nicht über einen Mangel an Dingen beklagen, die ich tun sollte, aber wie es aussieht, verbringe ich die meiste Zeit mit Dingen, die mir einfach nur Spaß machen.

Am 25. August ließ ich die Vormittagssitzung der SVFIG sausen und schloss mich stattdessen einem VCOA–(Volvo Club of America–)Treffen in Palo Alto an, mit meinem Kombiwagen aus dem Jahre 1966 bescheiden zwischen etwa 70 anderen Volvo–Fahrern. Von denen kamen die meisten in blitzblank herausgeputzten alten Kisten oder in aufgemotzten neueren Modellen an und waren genauso stolz auf ihre *Babys* wie Forth–Programmierer es auf ihre eigene Variante von Chuck Moores Original sind. Ich sprach mal mit diesem, mal mit jenem, hauptsächlich jedoch mit den Besitzern der älteren Wagen, darunter auch mit dem *berühmte*n oder *berüchtigten 4–Millionen–Kilometer–Menschen*, Irv Gordon, der seinem 1966–P1800 gerade weitere 3000 Meilen einverleibt hatte, indem er ihn tags zuvor von der Ostküste herfuhr. Gut, das war ein Blick auf eine andere Kultur, einen anderen *Kult*, gänzlich anders als der, der von jener Gruppe von etwa 15 Forthlern vertreten wird, zu denen ich mich dann am Nachmittag im Cogswell College gesellte. Im Rückblick muss ich sagen, dass ich mich zwar bei der letzteren Gruppe eher zu Hause fühlte, dass ich aber den Beschreibungen der mechanischen Projekte der ersteren Gruppe viel besser folgen konnte als das bei John Ribles Vorstellung seines neuesten Projektes, der SP2–CPU, der Fall war.

Mein VD–Heft 2/2007 traf am 17. September ein, gerade noch rechtzeitig, um es fünf Tage später zum Treffen mitzunehmen. Ich danke euch allen, allen hingebungsvollen Forth–Freunden aus Deutschland und Österreich! Diesmal verbrachte ich den ganzen Tag auf der SVFIG–Zusammenkunft im Cogswell–College. Morgens hörte ich mir Tings jüngste Lern–Experimente (mit HTML) an. Und am Nachmittag genoss ich LaFarr Stuarts Ausführungen über seine DOS– und Assembler–Routinen für extrem schnelle Berechnungen in der Mathematischen Zahlentheorie. Für den Rest des Tages hörte ich mir mit Interesse die Ausführungen von Masa Kasahara über die Suche nach nützlichen Tools oder nützlicher Hardware für sein *Audible Computing*–Projekt an. Er arbeitet an einer Methode, beim Umgang mit dem Computer nur den Hörsinn als Human–Interface einzusetzen. Speziell denkt er an Morse–Zeichen, mit Schaltern oder Maus–Tasten als E/A–Medium über die am Computer vorhandenen Ports und Lautsprecher.

Unter den Besuchern habe ich mit Vergnügen auch Dr. Glen Haydon getroffen. Er bedankt sich für das VD–Heft und er bat mich, dir, Fred, seine besten Grüße auszurichten.

Beeindruckt war ich vom Forth Family Tree in der Mitte des VD–Heftes. Zeitlich war es mir nicht mehr möglich, die Dinge der ganzen Gruppe zu erklären, aber ich reichte die Zeitschrift herum und schrieb die Kontakt–URL an die Tafel. Ich hoffe, dass Dave Jaffe sie in der SVFIG–Website mit aufnimmt. Nach dem, was ich gesehen habe, gibt es aus der Sicht der hiesigen Seite des Großen Teichs sicher noch viele *Forth–System Xs*, die man dem Forth–Tree einverleiben könnte.

Wie ich es bereits bedauerte, ist die Welt im Zeitalter der Höchstgeschwindigkeitskommunikation zwar klein geworden, aber die Unterschiede in Kultur und Sprache scheinen leider nicht aufzuhören, die Wege der Zusammenarbeit unter den Völkern in die Länge zu ziehen und zu verengen. War es Churchill, der sagte, dass Engländer und Amerikaner durch eine gemeinsame Sprache getrennt seien? Könnte man in unseren Tagen dasselbe nicht auch in Bezug auf einen Großteil der Mitglieder des Forth Family Trees sagen?

Wie ich sehe, fand die EuroForth–Konferenz zur selben Zeit statt, da wir uns in Kalifornien trafen. Im nächsten VD–Heft wird ja wohl ein Bericht darüber zu lesen sein? War jemand von unserem Kontinent da? Was hat sich eigentlich in Sachen ANS–Forth Neues ergeben? John Rible war gestern nicht auf unserem Treffen und ich fand auch sonst niemanden, der meine Fragen beantworten konnte.

Du siehst, ich habe eigentlich nichts wirklich Wichtiges zu berichten, das Forth allgemein betrifft und für die VD–Leser interessant sein könnte.

Jetzt muss ich aber Schluss machen. Die Zeit läuft mir davon. Ein Freund aus Michigan war zwischendurch am Telefon und wir haben uns zu lange unterhalten.

Henry Übersetzt von Fred Behringer

Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 63, August 2007

RetroForth — Paul Wiegmans

Retro ist, wie der Autor sagt, ein simplistisches Forth für Linux, Windows und BSD. Retro stammte ursprünglich von Tom Novelli und wurde dann von Charles Childers weiterentwickelt: `http://www.retroforth.com`. Quelltexte und ausführbare Programme stehen zum Download bereit. Es existiert eine aktive Benutzer–Gruppe, die eine ganze Reihe von Anwendungsprogrammen in RetroForth entwickelt hat. Retro ist in Assembler geschrieben (Rx core). Retro ist kein ANS–Forth, kann aber leicht ANS–kompatibel gemacht werden. Der Autor berichtet über seine ersten Erfahrungen mit RetroForth.

Programmeren in Forth (PiF) — Albert Nijhof

Im Rahmen des neu eingerichteten Workshops PiF bespricht Albert diesmal die weitreichenden Anwendungsmöglichkeiten des Wortes `SCAN ( ad1 len1 char - ad2 len2 )`. `SCAN` sucht im String `ad1,len1` diejenige Stelle, an welcher das Zeichen `char` das erste Mal vorkommt. Das davorliegende Stück String wird weggeschnitten. `ad2,len2` ist der (mit `char` beginnende) Reststring. `SCAN` gehört nicht zum Sprachumfang von ANS–Forth, ist aber, so Albert, in so ziemlich jedem Forth–System enthalten oder kann leicht ergänzt werden. Mit `SCAN` lassen sich elegant Menüs aufbauen. Albert gibt zwei Colon–Definitionen für `SCAN` an, insbesondere auch, um die (kunstvolle) Verwendung von `WHILE` zu demonstrieren. (Der Rezensent: Im Turbo–Forth–System ist `SCAN` explizit enthalten. Dort gibt es, in Form von `SKIP`, sogar noch eine Art von Komplementärwort dazu. `SCAN` lässt den Reststring mit dem ersten Zeichen `= char` beginnen, `SKIP` mit dem ersten Zeichen `<> char`.)

Mededelingen — Redactie

Die Redaktion teilt mit, dass auf dem Forth–Webserver ein neues Forum eingerichtet wurde: `http:///www.forth.hccnet.nl/apps/bbs`. Der Zugang ist frei, aber Registrierung wird verlangt. Hier können Fragen gestellt und Anmerkungen und kleine Berichte für andere HCC!Forth–Mitglieder oder für den Vorstand hinterlassen werden.

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 64, Oktober 2007

Forth vanaf de grond 9 — Ron Minke

Ron fügt seiner Artikelserie (Übersetzungen im AVR–Sonderheft 2007 der VD) einen neunten Teil hinzu. Statt des ATmega162 (mit nur 1 kB internem RAM) bereitet er jetzt sein *Forth von der Pike auf* für den kürzlich herausgekommenen ATmega2560/ATmega2561 vor. Dieser hat 256 kB Flash und 8 kB internes RAM. In einer Reihe von Forth–Worten lief beim ATmega162 das Umschalten von ROM nach RAM über die Adressgröße der Operanden. Bei Adressen über 0500 ging die Operation ins ROM (sprich Flash), bei Adressen darunter ins RAM. Der Übergangswert von bisher 0500 muss angepasst werden. Er beträgt jetzt 2200. Es bleiben jetzt 7,5 kB für eigene Forth–Definitionen. Ron gibt ein neues Speicherabbild an.

Data invoer via de inputstroom — Albert Nijhof

„In der Newsgroup comp.lang.forth ist man“, so der Autor, „damit beschäftigt, einen Vorschlag mit dem Ziel zu formulieren, basisgekennzeichnete Zahlen (wie $100 oder #100) in den ANS–Standard aufgenommen zu bekommen.“Albert kommt in diesem Zusammenhang auf eines seiner Lieblingsthemen zu sprechen, auf *Vorsetzworte*, mit denen eine bestimmte Eigenschaft des Nachfolgewortes gekennzeichnet wird. Er weist auf das Beispiel mit CHAR und [CHAR] hin. Dafür möchte er (in state–smartem Sinn) das vereinheitlichende Vorsetzwort CH verwenden. (Der Rezensent: Das ist natürlich genau

das altbekannte ASCII aus Forth 83 (F83, ZF, Turbo–Forth).) Albert möchte auch die oben genannten Basiskennzeichnungen durch Vorsetzworte ersetzen. Er gibt ein definierendes Wort (er nennt es FFBASE) an, mit dessen Hilfe beliebige Vorsetzworte `create`t werden können. FFBASE ist ein einfaches Utility–Wort, das jeder in jedem System nach Belieben verwenden kann und das in keinem Standard extra verankert zu werden braucht.

CIForth op Renesas bordje — Redactie

Die Renesas–Platine mit dem 16–bittigen M16C läuft jetzt auch unter CIForth. Albert van der Horst wird auf dem nächsten Treffen eine Vorführung veranstalten und das (uns Forth–Tagungs–Besuchern wohlbekannte) Metallofon *Tingel–Tangel* ansteuern. Es sind noch einige Platinen bei Albert vorrätig.

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 65, Dezember 2007

Renesas' M16C leert Forth — Albert van der Horst

Der ganze Artikel wurde vom Rezensenten übersetzt und kann im vorliegenden VD–Heft in deutscher Fassung gelesen werden (Seite 17).

Mededeling van het bestuur — Redactie

Die Jahresversammlung der Forth–gebruikersgroep wird fortan nicht mehr im April, sondern im Februar stattfinden.

One Laptop per Child — Redactie

OLPC (One Laptop Per Child) läuft seit einigen Jahren. Interessant: 1 Gigabyte Flash als Diskette. 200 Dollar für Entwicklungsländer, 400 Dollar für *reiche* Länder. Mitch Bradley hat das BIOS gemacht. Dieses Open–Firmware–BIOS ist ein erweitertes Forth (das also gleich nach dem Einschalten des Rechners sofort zur Verfügung steht. Mehr unter `http://wiki.laptop.org/go/Open_Firmware` und `http://wiki.laptop.org/go/FORTH`.

Frage des Rezensenten (an die Open–BIOS–Spezialisten unter uns): Gibt es eine Möglichkeit, dieses BIOS zeitweilig (so wie bei Zeichensätzen), vielleicht anstelle eines Schatten–BIOS, in meinen Rechner zu schalten? Am liebsten über einen Bootmanager (bei mir XFDISK).

Leserbriefe (Fortsetzung von Seite 14)

amforth User's Manual ist erschienen

Karl Lunt, Bothell, WA USA, hat Ende September 2007 ein User Manual zum amforth veröffentlich auf der Basis der Version 2.3. Darin beschreibt er im Detail, wie die amforth–Musterfassung für eigene Zielplattformen angepasst werden kann. Er benutzte dazu in seiner Windowsumgebung die freie Software von Atmel, das AVRStudio4.

Er hat das Dokument verfasst, um Mattias' Arbeit am amforth zu unterstützen. Lunt fand, es sei ein *exzellentes Forth* geworden und es bereite Vergnügen, damit zu arbeiten. Und er hat nun für die, die neu im amforth sind und etwas Unterstützung darin gebrauchen könnten, es den eigenen Bedürfnissen anzupassen, seine Erkenntnisse in dem Manual festgehalten. Lunt schrieb das Dokument mit OpenOffice 2.1 (`www.openoffice.org`), damit es dem GNU-Projekt hinzufügt werden kann, um amforth zu noch weiterer Verbreitung zu verhelfen. Dabei hat Karl Lunte es ausdrücklich erlaubt, dass sein User's Manual von anderen Autoren erweitert wird.

`http://amforth.sourceforge.net/amforth-userguide.pdf`
englisch, 14 Seiten PDF

Viele Grüße, Michael

Forth antik: Stackprozessoren

Hallo.

Schaut mal hier:

`http://www.forth-ev.de/staticpages/index.php/forth`

Habe folgendes ergänzt im Abschnit *Das Programmiersystem Forth*, weil so ganz ohne historischen Hinweis schien mir das nicht so gut:

„. . . Eine moderne Anwendung für Forth ist das Konzept der Open Firmware (IEEE-1275).

Forth wurde 1969 von Charles Moore entworfen. Es verbreitete sich rasch unter Mikroprozessor-Programmierern. Diese zunächst noch informelle Forth–Gesellschaft brachte ihre Anregungen ein in die entstehende Forth Interest Group (fig). So wurde 1977 ein erster Standard formuliert, das Forth–77, und die FIG (Forth Interest Group) gab modellhaft Implementationen für verschiedene Prozessortypen heraus — das legendäre fig-Forth. Viele Forth–Systeme für neuere Prozessoren sind aus dem fig–Forth hervorgegangen.

Inzwischen wurde Forth als American National Standard formuliert, dem die Forth–Systeme nun oft entsprechen — DPANS. Eine Übersicht über die Entwicklung ist im Forth Familiy–Tree von Anton Ertl, TU Wien, zu sehen.

“. . .

Ok, so?

Michael

False Forth in der c't — eine Artikelbesprechung

Fred Behringer

Exoterisch	=	allgemein verständlich.
Esoterisch	=	geheim gehalten.
Exotisch	=	anziehend, weil fremdländisch.

Martin Luther verfasste seine Schriften in einer Mischung aus Deutsch und Latein. Ich schließe mich an, wobei ich Latein durch Englisch ersetze.

FALSE ist kein Akronym, also schreibe ich False. Die falseschen Zitate lasse ich in der Originalsprache stehen. Zwischen Zitaten und Erklärungen unterscheide ich nur lasch.

Eigentlich soll die Anordnung und Auswahl der Zitate Erklärungen erübrigen. Um aber ein ganz klein bisschen Ordnung ins Geschehen zu bringen, kennzeichne ich Zitate da, wo es leicht geht, mit Anführungszeichen (LaTeX–Adepten würden hier wahrscheinlich Kursivschrift bevorzugen).

In der Nummer 22 der Zeitschrift c't vom 15.10.2007 auf den Seiten 192 bis 199 steht ein amüsanter Artikel über esoterische und exoterische Sprachen in einer für den Außenstehenden sehr verständlichen Ausführung mit höchst interessantem Hintergrund. Auch eine halbe Seite über den Begriff der Turing–Maschine fehlt nicht. (Ist Forth Turing–vollständig? Welches „Forth" : ANS, ANS–Core, Forth als System in einer gegebenen Implementation?) Weiteres siehe weiter unten.

Zuvor noch eine Mail vom Chaos Computer Club an das Forthbüro:

„`Betreff: Forth auf dem 21c3?`

```
Datum: Sun, 7 Nov 2004 20:53:08 +0100
Von: Corinna <corinnaATgeekinDOTde>
An: secretaryATforth-evDOTde

Hallo Forth-Team!

Wir suchen noch ReferentInnen fuer den 21.
Chaos Communication Congress zwischen
Weihnachten und Neujahr in Berlin.
"Wir" sind dabei die Haecksen, also Frauen &
Maedchen mit CCC-Bezug.

Geplant ist ein Schwerpunkt auf exotischeren
Programmiersprachen und Forth zaehl ich mal
dazu :-)"
```

Soviel zur Meinung der Außenwelt über Forth. Nun zum c't–Artikel von Oliver Lau:

„So richtig verständlich ist allerdings auch manch exoterische Sprache nicht — weshalb sie dem ein oder anderen vielleicht esoterisch vorkommen mag, etwa das fakten– und regelbasierte Prolog, Lisp mit dem Faible zu verketteten Listen oder Forth mit seiner gewöhnungsbedürftigen umgekehrten polnischen Notation (UPN)..."

Demnach ist Forth ganz normal exoterisch, es erscheint nur manch einem esoterisch — wegen der Umgekehrt–Polnischen Notation?

Na, Gott sei Dank! Das hebt Forth natürlich gleich eine halbe Stufe höher. Und wenn man die UPN wegnähme? Wird dann aus Forth eine hoffähige Sprache? Oder umgekehrt: Wenn man die UPN wegnimmt, bleibt Forth dann noch Forth?

Nicht umsonst hat unsereiner das berühmt–berüchtigte Interview mit Chuck Moore beim Versuch einer gerechten Wiedergabe im Deutschen in sich hineingesogen. Here comes what the Forth–Erfinder dazu meint (siehe VD–Heft 4 vom Jahre 2002, Seite 7 bis 14):

„Forth liefert kaum etwas Neues, aber die Zusammensetzung seiner Bestandteile ist einzigartig. Ich bin den Menschen und Einrichtungen, die es mir — zumeist ohne ihr Wissen — erlaubt haben, Forth zu entwickeln, dankbar. Und dankbar bin ich Ihnen dafür, dass Sie genug Interesse aufbringen, etwas darüber zu lesen."

Also doch: Jeder einzelne Bestandteil von Forth war und ist bekannt — und damit höchst exoterisch. Nimmt man aber einen einzigen davon weg, ist es nicht mehr Forth. Auf die geniale Zusammensetzung kam und kommt es an. Und die verdanken wir Charles (Chuck) Moore.

Als ob UPN esoterisch wäre! Kein Mensch hätte einen HP–Taschenrechner esoterisch genannt.

Und weiter beim c't–Autor Lau:

„Esoterische Sprachen entstehen typischerweise nicht mit Hinblick auf praktische Nutzbarkeit, sondern vielfach, weil jemand extremen Ehrgeiz und Aufwand in einen Teilaspekt investiert — oder einfach mal aus Jux."

„Zu den Turing–vollständigen Sprachen gehört zum Beispiel False, die der Niederländer Wouter van Oortmerssen im Jahr 1993 entworfen hat. Der Name: Eine Remineszenz an seinen Lieblings–Boolean–Wert. Der Anspruch: Eine mächtige Sprache mit einer möglichst verwirrenden Syntax zu entwerfen, die ein möglichst winziger Compiler in Maschinencode übersetzt. Van Oortmerssen lieferte auch gleich den Machbarkeitsbeweis in Gestalt einer nur 1024 Byte großen 68k–Assembler–Implementierung für den Amiga."

Der c't–Autor, Oliver Lau, spricht von Turing–vollständigen Sprachen. Wo finde ich eine saubere Definition des Begriffs „Sprache", einer Definition, mit welcher man bei der Untersuchung der T(uring)–Vollständigkeit arbeiten kann? Ist Forth T–vollständig? Ist nicht jede offene Computersprache T–vollständig? Gibt es außer Forth noch andere offene Computersprachen? False ist wohl nicht offen? Denn dann müsste es ja (als Sprache) in seinen Compiler (als System) eingreifen können (?) Fragen über Fragen (an die Informatiker und Forth–Fachleute).

Ein Charakteristikum von False scheint darin zu bestehen, dass bis zu 256 unterscheidbare Einzelzeichen, aber auch nur solche, als Befehle Verwendung finden. (Befehlszusammensetzungen sind möglich.) Es war für die vorliegende Besprechung nicht ganz sicher, welchen Zeichensatz die Vorlage des Handbuches von Wouter van Oortmerssen ursprünglich verwendete. Ich habe mich an Courier New (in NotePad und Word) gehalten und es dem Redakteur der VD überlassen, daraus ein brauchbares Satzbild zu machen. Wenn Zeichen in meiner Besprechung von False false wiedergegeben werden (man kommt sich vor wie Evelyn Hamann beim Witherspoon–Sketch), ist das auf den eben genannten Umstand zurückzuführen.

Übrigens sind 256 mögliche einstellige Befehle eine ganze Menge! Gforth–R8C hat 528 „Befehle“ — und die haben mir beim Elektor–Glossar schon ganz schön zu schaffen gemacht.

Sollten es aber wirklich nur die Einsparungen durch Übergang zu einbuchstabigen Forth–Worten sein? Dann könnte man ja vielleicht eine umkehrbar eindeutige (eine bijektive) Beziehung zwischen Forth–Wortschatz und False–Wortschatz herstellen? Mal versuchen! Erleichtert wird ein solches Vorhaben natürlich durch den Umstand, dass ja der Compiler bei False „außen vor“ bleibt. Man nehme in Forth den gesamten Colon–Compiler (das sind wohl hauptsächlich die Compile–Only–Worte aus ANS–Forth — die Kontrollstruktur–Worte wiederum aber auch wieder nicht?) heraus! Was bleibt für die eigentliche Sprache „Forth“ dann noch übrig? Eine wirklich minimalistische Sprache Forth?

Was mich betrifft, mich hat der in Forth geschriebene Forth–Compiler zur Freizeit–Beschäftigung mit Forth verleitet. Nicht unbedingt der dahinterstehende Meta–Compiler, der das gesamte Forth–System als Anwendungsprogramm zu schreiben gestattet. Der Colon–Compiler aber auf jeden Fall.

Und nun weiter beim c’t–Autor Oliver Lau:

„Wer Forth kennt oder etwas mit umgekehrter polnischer Notation (UPN) anzufangen weiß, kommt mit Forth recht flott voran. Folgender Code berechnet zum Beispiel (1+2)*4: 1 2 + 4 * “

und so weiter und so fort.

Aus dem Artikel ist nicht zu erkennen, ob beispielsweise zwischen 2 und + ein Leerzeichen als Trennzeichen liegt. Eine gewisse Antwort bekomme ich aus dem Beispiel der Fakultätsfunktion im Artikel von Lau und die anschließenden weiteren Erklärungen:

„Das Beispiel entbehrt nicht einer gewissen Ästhetik, die gewiss verlöre, wenn man den Code mit Leerzeichen, Einrückungen oder Zeilenumbrüchen entzerren würde. Wohl auch deshalb resümiert van Oortmerssen: Einrückungen, Leerzeichen und Kommentare sind für Nulpen; der wahre False–Programmierer schreibt kompakten (dichten) Code und greift auf Variablen höchstens für Funktionsdefinitionen zurück, aber nicht als Alternative zur Ablage von Werten auf dem Stack. Sicherlich war van Oortmerssen bewusst, dass der englische Begriff „dense“, den er im False–Handbüchlein verwendet, nicht nur „dicht“, sondern auch „blöd“ bedeutet.“

Der c’t–Artikel „Hexenwerk — ein Plädoyer für esoterische Programmiersprachen“ von Oliver Lau hat mich neugierig gemacht, insbesondere False. Das wollte ich genauer wissen und ich fing an, im „False–Handbüchlein“ zu „blättern“ (siehe unten). Einen Forth–Compiler von 1024 Byte Länge hätte ich liebend gern. (In den Home–Computer–Zeitschriften war früher viel von 1k–Programmen die Rede. Kurze Programme, die nur das tun, wofür sie entworfen wurden, das aber richtig.) Der c’t–Autor Lau geht in seinem Plädoyer für esoterische Sprachen in weiteren Abschnitten auf Brainfuck, Befunge, Whitespace und INTERCAL ein. Die lasse ich aus. Für das Erstgenannte besorge ich mir erst einen DUDEN der Gossensprache. Und einen Webster und ein „Oxford Concise Dictionary der Wörter der übleren Sorte“ muss ich mir auch noch heraussuchen. Bis zum nächsten Mal.

Ich darf also jetzt zu False übergehen und einige Zitate aus dem „Handbüchlein“ (immerhin 11 Seiten) bringen, um Wouter van Oortmerssens Bemühungen zu skizzieren.

```
„The language FALSE and its compiler were designed for only two reasons:
- building a working compiler in just 1k (!)
- designing a language that looks cryptic and fuzzy (in the APL tradition).

The result is a language that is quite powerful (for its size).
It’s a Forth type language with lambda abstraction and lots of other goodies.
I named the language after my favourite truthvalue.

I suppose you really have to be a Forth hacker or a hardcore programmer to get the most of it.

FALSE inherits its way of evaluating expressions from Forth, so it
really helps if you know that language, but for those who don’t:

All elements in the language are defined by what they push on and/or
pop from the stack. For example, a number like "1"  or "100"  simply
pushes its own value on the stack. An operator like "+"  takes the
two top elements of the stack, adds them, and pushes back the result:
```

```
1 2 + 4 * { (1+2)*4 }

The result of this expression is "12" . And this is the fac() function definition in FALSE:

[$1=$[\%1\]?~[$1-f;!*]?]f:

A FALSE lambda function is a piece of code between []. For example: [1+] is a function that adds
1 to its argument. 2[1+]! would result in "3" .

Control structures: FALSE only has an IF and a WHILE.

FALSE language overview:

syntax:     pops:      pushes:      example:       comment:
            -->top     -->top
----------- ---------- ------------ -------------  ------------------------
{comment}   -          -                           { this is a comment }
[code]      -          function     [1+]           { (lambda (x) (+ x 1)) }
a .. z      -          varadr       a              { use a: or a; }
integer     -          value        1
'char       -          value        'A             { 65 }
num`        -          -            0`             { emitword(0) }

:           n,varadr   -            1a:            { a:=1 }
;           varadr     varvalue     a;             { a }
!           function   -            f;!            { f() }

+           n1,n1      n1+n2        1 2+           { 1+2 }
-           n1,n2      n1-n2        1 2-
*           n1,n2      n1*n2        1 2*
/           n1,n2      n1/n2        1 2/
_           n          -n           1_             { -1 }

=           n1,n1      n1=n2        1 2=~          { 1<>2 }
>           n1,n2      n1>n2        1 2>

&           n1,n2      n1 and n2    1 2&           { 1 and 2 }
|           n1,n2      n1 or n2     1 2|
~           n          not n        0~             { -1,TRUE }

$           n          n,n          1$             { dupl. top stack }
%           n          -            1%             { del. top stack }
\           n1,n2      n2,n1        1 2\           { swap }
@           n,n1,n2    n1,n2,n      1 2 3@         { rot }
ø (alt-o)   n          v            1 2 1ø         { pick }

?           bool,fun   -            a;2=[1f;!]?    { if a=2 then f(1) }
#           boolf,fun  -            1[$100<][1+]#  { while a<100 do a:=a+1 }

.           n          -            1.             { printnum(1) }
"string"    -          -            "hi!"          { printstr("hi!") }
,           ch         -            10,            { putc(10) }
^           -          ch           ^              { getc() }
ß (alt-s)   -          -            ß              { flush() }
“
```

Soweit der Überblick von W. van Oortmerssen über seine Sprache False. (Und schon fängt es mit den systemabhängigen typografischen Inkompatibilitäten an: alt–s liefert bei mir unter Word 2000 etwas „Furchtbares“, alt–o liefert gar nichts. „Wir“ wissen, wo wir unser ß herbekommen! Das liegt auf der deutschen Tastatur oben rechts. Aber bei ø müssen wir uns schon was einfallen lassen.)

Das möge genügen. Und hier noch die Beweggründe des False–Schöpfers zur Erschaffung seiner Sprache:

„FALSE was created for fun, just to see how small a compiler I could write while still compiling a relatively powerful language.“

W. v. Oortmerssen gibt Hinweise auf weitere Quellen zum Thema:

„Steinar Knutsen has been so friendly to set up an FTP site so all you people can put your False related products there!

`ftp://ftp.nvg.unit.no/pub/lang/false/` for the distribution.

`ftp://ftp.nvg.unit.no/pub/lang/false/src/` for sources not in the distribution.

Chris Pressey has a False homepage at:

`http://www.cats-eye.com/cet/soft/lang/false/`

Ben Hoyt (benhoytATclearDOTnetDOTnz) has made a full implementation of False as both interpreter and compiler (for 386 DOS) written in ANSI–Forth! The source code is something to be seen. Get the archive at: `http://wouter.fov120.com/files/lang /false/forth_false.zip` .“

Diesen `forth_false.zip`, wenn er schon mal something to be seen ist, habe ich mir mal angeschaut. Ausprobieren wollte ich den in Forth geschriebenen False–Interpreter und den ebenfalls in Forth geschriebenen False–Compiler von Ben Hoyt (2000) auf den Systemen 32/FORTH 3.07 (DPMI–getrieben) von Rick VanNorman (1993) und MinForth 1.5 von Andreas Kochenburger. Beide sind, wie von Ben Hoyt gefordert, 32–Bit–Systeme. „Fully ANS compliant“. Aber: Ein ANS–Forth–System, das ist hinreichend bekannt und man wagt es kaum, es zu wiederholen, ist genau ein ANS–Forth–System. Nicht die Sprache. Die ist fully standardisiert. Aber das System, mit dem man es zu tun bekommt. Nun, bis zur Überprüfung der Definierbarkeit der (generischen) Lambda–Funktion [1+]i: bin ich nicht vorgedrungen. (Die darin enthaltene eigentliche False–Lambda–Funktion heißt [1+].) In beiden Systemen nicht. Im MinForth–System zeigte der Bildschirm wenigstens die Erkennungsmeldung

Portable False interpreter in Forth — by Ben Hoyt — 5 February 2000

an. Aber der Forth–Quelltext von Ben Hoyt ist prima verständlich und die genauere Untersuchung halte ich mir für später auf.

W. v. Oortmerssen schließt mit folgenden Danksagungen ab:

„I want to thank the people who contributed sourcecode, among others:

Ben Schaeffer, Ed Mackey, Eelko de Vos (and the rest of The TU–Delft False Fanclub, Maarten and Rene), Herb Wollman, Lionel Vintenat, Marcel van Kervinck, Peter Bengtsson, Steinar Knutsen, Thomas Fischbacher, and Tomas Partl

and the many more people that have shown their interest in the language. One of them put it like this:

> #define FLAME ON Dear Mr. Wouter van Oortmerssen, Sir, Bwana!
>
> We (FORTH enthusiasts of Southern German Banana Republic of Bavaria) are not amused by your FALSE language.
>
> No Sir, not at all.
>
> Some of us are still jumping up and down in the coconut tree muttering obscenities like „DUP RECURSE that *@! Oortmerssen!“. (Some are say even things like „'Oortmerssen EXECUTE“)
>
> You have had the intention of writing an absolutely cryptic while extreme terse & powerful language. In that you've succeeded marvelously. Yet why, Great Moore! have you taken FORTH for the template? Forth is quite cryptic already, yet you must you render it completely unreadable. In Bavaria this is regarded as a grave public offense, sentenceable to not below of 3 years of pure K&R C programming. You could plead for clemency by pointing out your having introduced the lambda computational concept into FORTH, though. I will grant you that much.“

Die Einrückung kennzeichnet das Zitat im Zitat.

W. van Oortmerssen ist (oder war zum Zeitpunkt der Erstellung seiner momentanen Homepage) Lecturer at the Guildhall. Ob er wohl eine Möglichkeit gesucht hat, seinen Studenten ohne viel Federlesens Erfolgserlebnisse beim Computersprach– oder/und Compiler–Studium zu verschaffen?

Hier die Anschrift des False–Schöpfers: OortmersATfwiDOTuvaDOTnl

Zurück zur Besprechung des c't–Artikels von Oliver Lau: Ich kann (als Rezensent) nicht genau erkennen, in welche Reihe der c't–Autor Forth tatsächlich einstuft, exoterisch, exotisch oder esoterisch.

Albert Nijhof in seinem Arbeitsbuch „De Programmeertaal FORTH“ (deutsche Übersetzung beim mv–Verlag): „Forth is de computertaal die het best geheim gehouden is.“ (Dick Pountain)

Der Rezensent: Dus toch esoterisch?

Ist Forth Turing–vollständig?

Ulrich Hoffmann

Ist Forth Turing–vollständig?
Diese Frage wirft Fred Behringer in seinem Artikel False Forth (Seite 27) auf.

Nun — um der Antwort näher zu kommen, müssen wir erstmal ein wenig darüber erfahren, was Turing–Maschinen sind und was eigentlich Turing–Vollständigkeit bedeuten soll.

Anfang des 20. Jahrhunderts — als es noch gar keine Computer gab — haben sich viele Mathematiker Gedanken darüber gemacht, welche Aufgaben Maschinen wohl verrichten können würden bzw. an welchen sie prinzipiell scheitern müssten. Mathematiker fassten das exakt in die Fragestellung, welche mathematischen Funktionen von einem Automaten berechnet werden können und welche eben nicht. Heute sind diese Fragestellungen Teil der theoretischen Informatik (in der Komplexitätstheorie fragt man zusätzlich, wie aufwändig eine Funktion berechenbar ist).

In diesem Zusammenhang hat der britische Mathematiker Alan Turing 1936 ein einfaches Modell — die Turing–Maschine — für einen universellen Automaten vorgeschlagen, der in der Lage ist, alle berechenbaren Funktionen auch tatsächlich auszurechnen. Damit war auch eine Möglichkeit gefunden, von einer gegebenen Funktion zu entscheiden, ob sie berechenbar ist oder nicht: Kann man ein Programm für die Turing–Machine schreiben, das die Funktion ausrechnet, so ist sie berechenbar. Kann man beweisen, dass kein Programm der Turing–Maschine die Funktion ausrechnet, so ist sie nicht–berechenbar.

Eine Turing–Maschine stellt man sich am besten folgendermaßen vor: Sie ist ähnlich aufgebaut wie ein Tonbandgerät, dessen Magnetband in Felder eingeteilt ist. Auf jedem dieser Felder kann höchstens ein Zeichen stehen (es kann auch leer sein).

Wie in Abbildung 1 zu sehen ist, besitzt die Turing–Maschine M einen Schreib–/Lesekopf und sie kann nur das Feld beschreiben oder auslesen, das sich unter dem Magnetkopf befindet. Das Band kann in einem Schritt um ein Feld nach links oder rechts bewegt werden. Für die Turing–Maschine wird zusätzlich eine endliche Menge möglicher Zustände definiert. Zu jedem Zeitpunkt befindet sie sich in einem dieser Zustände. In einer Zustandsübergangstabelle T (dem Programm der Maschine) wird mit jeder Zeile festgelegt, in welchem Zustand und bei welchem Zeichen unter dem Kopf sich die Maschine wie verhalten soll: welches Zeichen soll sie auf das Band schreiben, wie soll sie den Kopf bewegen und welches ist der Nachfolgezustand.

Zu Anfang erwartet die Turing–Maschine die Eingabe der auszurechnenden Funktion (kodiert) auf dem Band und startet in einem vereinbarten Anfangszustand s_0. Sie arbeitet nun Schritt für Schritt, bis die Tabelle für den schließlich erreichten Zustand (und Wert unterm Kopf) keine Aktion festlegt. Dann hält die Maschine an und das Band enthält das (kodierte) Ergebnis der Berechnung. Gerät die Turing–Maschine in eine Endlosschleife, hat sie ihr Ziel verfehlt und liefert kein Ergebnis.

Um einen Schritt durchzuführen, geht die Maschine folgendermaßen vor: Sie befindet sich in einem Zustand s und kann unter dem Kopf das Zeichen c lesen. Die Tabelle bestimmt für s und c die Aktionen der Maschine: Sie schreibt das angegebene Zeichen, bewegt den Kopf in die angegebene Richtung und wechselt in einen neuen Zustand. Damit ist dieser Schritt beendet und die Maschine ist bereit für den nächsten Schritt.

Die Turing–Maschine ist ein theoretisches Werkzeug, um die Berechenbarkeit von Funktionen zu untersuchen. Sie ist aus zwei Gründen unpraktikabel: erstens ist ihr Befehlsvorrat sehr primitiv: sie kann nicht einmal elementar Zahlen addieren und benötigt schon dafür ein Programm, wie etwa das in Abbildung 1 auf der nächsten Seite. Zweitens fordert Turing, dass das Band seiner Maschinen unendlich groß ist, d.h. rechts wie links beliebig viele Felder zur Verfügung stehen. Eine solche Maschine kann man nicht bauen.[1] Aber — das ist ja auch gar nicht Sinn und Zweck einer Turing–Maschine.

[1] Das Band ist übrigens das einzige Unendliche an der Turing–Maschine, der Zeichenvorrat für die Band–Symbole, die Menge der Zustände und auch die Zustandsübergangstabelle sind endlich. Alles, was die Turing–Maschine von einem endlichen Automaten unterscheidet, ist also ihr beliebig großes Band.

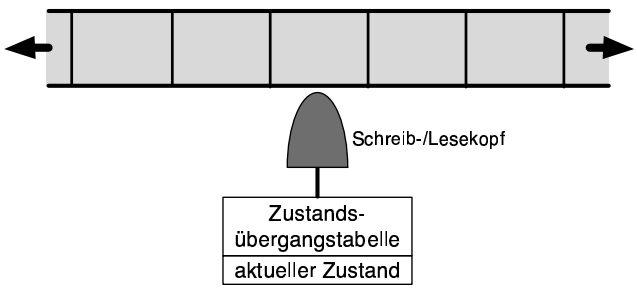


Abbildung 1: Der Aufbau einer Turing–Maschine

Zustand s	Eingabe c	Ausgabe	Kopfbewegung	Folgezustand	Kommentar
0	|	|	nach rechts	0	überspringe alle Striche
0	_	|	nach rechts	1	schreibe einen Strich in die Lücke
1	|	|	nach rechts	1	überspringe alle Striche
1	_	_	nach links	2	auf den letzten Strich
2	|	_	nach links	3	letzten Strich löschen, auf vorletzten
3	|	_	nach links	4	vorletzten Strich löschen
4	|	|	nach links	4	überspringe alle Striche rückwärts
4	_	_	nach rechts	5	auf den ersten Strich positionieren

Abbildung 1: Die Zustandsübergangstabelle einer Turing–Machine, die zwei Strichzahlen addiert

Gibt es eigentlich Funktionen, die Turing–Maschinen ausrechnen können, die aber von einfacheren Mechanismen (etwa einfachen Keller–Automaten) nicht berechnet werden können? Ja — das sind die nicht–primitiv–rekursiven (aber μ–rekursiven) Funktionen, etwa die Ackermannfunktion.

Gibt es eigentlich Funktionen, die Turing–Machinen nicht ausrechnen können? Ja — das sind u. a. spezielle Fragestellungen über das Verhalten von Turing–Maschinen selbst, insbesondere das so genannte *Halteproblem*: Hält eine gegebene Turing–Maschine, gestartet auf einer Eingabe, nach endlich vielen Schritten an oder läuft sie unendlich lange weiter? Tatsächlich kann eine Turing–Maschine die Antwort auf diese Frage nicht ausrechnen (selbst nach endlich vielen Schritten immer das richtige Ergebnis liefern). Auch kein anderer bekannter Mechanismus kann diese Frage beantworten. In diesem Sinne ist die Turing–Maschine universell: sie berechnet alle berechenbaren Funktionen.

Betrachtet man nun andere Mechanismen, etwa Programmiersprachen, dann stellt sich die Frage, ob diese die gleichen Funktionen berechnen, wie Turing–Maschinen. Ist das der Fall, spricht man davon, sie seien *Turing–vollständig*. Eine Technik, diese Frage zu beantworten, versucht Turing–Maschinen mit Hilfe des betrachteten Algorithmus zu simulieren. Dabei geht man davon aus, dass beliebig viel Speicher zur Verfügung steht. Gelingt die Simulation allgemein, so kann man jede von einer Turing–Maschine berechenbare Funktion selbst berechnen, indem man die zugehörige Turing–Maschine simuliert.

Ist Forth also Turing–vollständig? Um das zu beantworten, müssen wir also einen Turing–Maschinen–Simulator in Forth programmieren. Das ist nicht schwer und im Listing unten zu sehen. Prima — Forth ist also Turing–vollständig. Und jetzt?

Der angegebene Simulator verwendet das Dictionary, um das Band zu realisieren. Wie wäre es aber, wenn man kein Dictionary hätte und nur die beiden Forth–Stacks (beide beliebig groß)? Wäre dann immer noch ein Simulator möglich?

```
1   \ Turing.fs  Turingmaschinen-Simulator in Forth    uho 2007-12-04
2
3   : 3dup ( a b c -- a b c a b c )  >r 2dup r@ -rot r> ;
4   : 3drop ( a b c -- )  2drop drop ;
5
6   Variable 'head          \ Kopfposition: Adresse der aktuellen Zelle
7   Variable 'state-table   \ Anfangsadresse der Zustandstabelle
8   Variable status         \ aktueller Zustand
9
10  \ Kopfpositionierung
11  : sym ( -- addr )  'head @ ;
12  : moveright  ( -- )  1 cells 'head +! ;
13  : moveleft  ( -- )  -1 cells 'head +! ;
14  : headmove ( m -- )  IF moveleft ELSE moveright THEN ;
15
16  \ Zeile der Zustandsübergangstabelle:
17  \ { Zustand | Eingabesymbol | Ausgabesymbol | Kopfbewegung | Folgezustand }
18  : >state ( 'row -- 'state ) ;
19  : >insym ( 'row -- 'state )  1 cells + ;
20  : >outsym ( 'row -- 'outsym )  2 cells + ;
21  : >movement ( 'row -- 'movement )  3 cells + ;
22  : >newstate ( 'row -- 'newstate )  4 cells + ;
23  : +row ( 'row0 -- 'row1 )  5 cells + ;
24
```

```
25  \ Durchsuchen der Zustandsübergangstabelle:
26  : found? ( s c 'row -- f )  swap over  >insym @ = >r  >state @ = r> and ;
27  : another-row? ( 'row -- f )  >r -1 -1 r> found? 0= ;
28  : @answer ( 'row -- s m c )  dup >r >newstate @ r@ >movement @ r> >outsym @ ;
29
30  : ?state-table ( s0 c0 -- s1 m c1 | -1 -1 -1 )
31      'state-table @
32      BEGIN ( s0 c0 'row )
33        dup another-row?
34      WHILE ( s0 c0 'row )
35        3dup found? IF >r 2drop r> @answer EXIT THEN
36        +row
37      REPEAT
38      drop 2drop
39      -1 -1 -1 ;
40
41  \ Ein Schritt
42  : valid-step? ( s m c -- f )  -1 <>   swap -1 <> and   swap -1 <> and ;
43  : next-action ( -- s m c )  status @  sym @  ?state-table ;
44  : do-step ( s m c -- )  sym !  headmove  status ! ;
45
46  \ Laufen bis kein Zustandsübergang definiert
47  : run ( -- )  0 status !
48     BEGIN
49       next-action  3dup valid-step?
50     WHILE ( s m c )
51       do-step
52     REPEAT
53     3drop ;
54
55  \ Notation für das Band und die Zustandsübergangstabelle
56  : <*>  ( -- )  here 'head ! ;
57  0 Constant ->
58  1 Constant <-
59  char _ Constant _
60  char | Constant |
61  : end-table ( -- )  -1 , -1 , -1 , -1 , -1 , ;
62
63
64  \ Turing-Maschine, die Strichzahlen addiert
65  \ Die Zahl n>=0 wird durch n+1 Striche auf dem Band
66  \ kodiert. Die beiden Parameter sind durch eine Lücke _
67  \ voneinander getrennt, also für 2+3 steht
68  \ ... _ _ _ _ | | | _ | | | | _ _ _ ... auf dem Band und
69  \            <*>
70  \ der Kopf steht auf dem ersten Strich der Zahl 2.
71
72  Create tape
73    _ ,  _ ,  _ ,   <*> | ,  | ,  | , _  ,  | ,  | ,  | ,  | ,  _ ,  _ ,  _ ,
74
75  Create state-table   here 'state-table !
76  \ state input   output movement newstate
77      0 ,   | ,     | ,     -> ,    0 ,  \ überspringe alle Striche
78      0 ,   _ ,     | ,     -> ,    1 ,  \ schreibe einen Strich in die Lücke
79
80      1 ,   | ,     | ,     -> ,    1 ,  \ überspringe alle Striche
81      1 ,   _ ,     _ ,     <- ,    2 ,  \ auf den letzten Strich
82
83      2 ,   | ,     _ ,     <- ,    3 ,  \ letzten Strich löschen, auf vorletzten
84      3 ,   | ,     _ ,     <- ,    4 ,  \ vorletzten Strich löschen
85
86      4 ,   | ,     | ,     <- ,    4 ,  \ überspringe alle Striche rückwärts
87      4 ,   _ ,     _ ,     -> ,    5 ,  \ auf den ersten Strich positionieren
88  end-table
```

Ulam–Spirale

Michael Kalus

Ein bescheidener Versuch und die Aufforderung, weiter zu machen.

Neulich stöberte ich im Internet und stieß dabei auf die Ulam–Spirale. Eine interessante Sache. Nun, erklären kann ich das Phänomen natürlich auch nicht. Aber natürlich kam mir in den Sinn, wie so eine Spirale denn wohl per Forth–Programm ausgedruckt werden könnte.

Nun, was ist eine Ulam–Spirale? Es ist eine einfache Methode, Primzahlen grafisch darzustellen. Sie wurde 1963 von dem polnischen Mathematiker Stanisław Marcin Ulam während eines wissenschaftlichen Vortrags entdeckt, als er, aus Langeweile, Zahlenreihen auf ein Papier kritzelte. Er begann mit einer *1* in der Mitte und fuhr dann in Spiralform fort:

```
43   44   45   46   47   48   49 ...
42   21   22   23   24   25   26
41   20   07   08   09   10   27
40   19   06   01   02   11   28
39   18   05   04   03   12   29
38   17   16   15   14   13   30
37   36   35   34   33   32   31
```

Werden nur alle Primzahlen dargestellt, erhält man folgendes Muster:

```
43    -    -    -   47    -    - ...
 -    -    -   23    -    -    -
41    -   07    -    -    -    -
 -   19    -    -   02   11    -
 -    -   05    -   03    -   29
 -   17    -    -    -   13    -
37    -    -    -    -    -   31
```

Überraschenderweise liegen ja sehr viele Primzahlen auf diagonalen Geraden. Mehr als eine nette Spielerei? Nun, falls jemand mehr dazu sagen kann, nur zu!

Hier also nun mein Versuch, diese Spirale mittels Forth zu drucken. Die Begrenzung ergibt sich aus der Papiergröße bzw Bildschirmgröße. Die Zahlen ergeben sich zwar einfach nur aus n=n+1 aber sollen ja spiralförmig um ihr Zentrum herum niedergeschrieben werden. Diesen Vorgang bilde ich ab, indem die Zahlen zunächst spiralförmig in ein Array geschrieben werden. Hinterher kann das Array dann zeilenweise ausgelesen und gedruckt werden.

Nun bin ich noch auf der Suche nach Folgendem: Wie legt man in Forth das Array dynamisch an, damit so etwas wie

```
: ulam ( xmax ymax -- ) ... ;
```

möglich wird? Und wie lassen sich in Forth die Primzahlen erkennen und markieren?

Viel Spaß beim Knobeln, Michael

Quelle: `http://de.wikipedia.org/wiki/Ulam-Spirale`

```
\ Ulam-Spirale drucken.
decimal
only forth
vocabulary ulam
also forth ulam definitions

\ use odd number to get a center cell
15 constant xmax
15 constant ymax

here xmax ymax * cells allot here swap constant feld constant feldende

: feldadr  ( x y - adr )
  xmax * cells swap cells + feld + ;  \ ok mka

: .feld ( -- ) \ test adr of array
  ymax 0 DO cr
   xmax 0 DO
    i j feldadr .
   loop
  loop cr ;

: .feld@ ( -- ) \ test content of array
  ymax 0 DO cr
   xmax 0 DO
    i j feldadr @ 5 .r
   loop
```

```
28    loop cr ;
29
30  : feld!  ( n x y -- )
31    feldadr ! ;
32
33  : feld@  ( x y -- n )
34    feldadr @ ;
35
36  : center-adr ( -- adr ) \ center position of ulam spirale
37    xmax 2/ ymax 2/ feldadr ; \ ok mka
38
39  : center ( -- x y )
40    xmax 2/ ymax 2/ ;
41
42
43  variable x
44  variable y
45
46  variable xR
47  variable xL
48
49  variable yO
50  variable yU
51
52  variable richtung  \ 0=right, 1=up, 2=left, 3=down
53
54  : init-xy ( -- )
55    0 richtung !
56    center y ! x !
57    x @ dup xR ! xL !
58    y @ dup yO ! yU !  ;
59
60  : ..xy ( -- )
61    cr ." x  " x @ .
62    cr ." y  " y @ .
63    cr ." richtung=" richtung @ .
64    cr ." xR " xR @ .
65    cr ." xL " xL @ .
66    cr ." yO " yO @ .
67    cr ." yU " yU @ . ;
68
69  : x+ ( -- )   1 x +! x @ xR @ > IF x @ xR ! 1 richtung ! THEN ;
70  : x- ( -- )  -1 x +! x @ xL @ < IF x @ xL ! 3 richtung ! THEN ;
71  : y+ ( -- )   1 y +! y @ yO @ > IF y @ yO ! 2 richtung ! THEN ;
72  : y- ( -- )  -1 y +! y @ yU @ < IF y @ yU ! 0 richtung ! THEN ;
73
74  : ulam+ ( -- )
75    richtung @  0 = IF x+ exit then
76    richtung @  1 = IF y+ exit then
77    richtung @  2 = IF x- exit then
78    richtung @  3 = IF y- exit then
79    abort" richtung unmoeglich" ;
80
81  init-xy
82
83  : ..  ulam+ ..xy ; \ test
84
85  : ulam ( -- )
86    xmax ymax * 0 DO
87     i 1+ x @ y @ feldadr ! ulam+  \ array fuellen
88    loop
89    .feld@ ; \ array drucken
90
91  words
92
93  \ finis
```

Primzahlen und die Spirale von Ulam

Fred Behringer

Eines sieht man sofort: Die von Michael Kalus (im vorliegenden Heft) angegebene Spirale ist rechtsdrehend, die angelsächsischen Spiralen (beispielsweise bei Weisstein, Eric W.: Prime Spiral) sind linksdrehend. Wir, die wir in einer dreidimensionalen Welt leben, können natürlich das Blatt, auf welchem die Spirale gezeichnet ist, einfach wenden. Und dann stellt sich der Unterschied als gegenstandslos heraus. (Was aber macht man aus dem vielzitierten Ausspruch *die Natur (Teilchenspins etc) bevorzugt Linksdrehung*?) Soweit ein Einleitungsersatz, der zum Nachdenken anregen soll.

Wenn man mit redaktionellen Aufgaben beschäftigt ist, und VD–Korrekturarbeiten gehören dazu, dann hat man den Vorteil, dass man die Beiträge der Autoren schon zu einem sehr frühen Zeitpunkt zu Gesicht bekommt. Ich freue mich, dass Michael die Sache mit der Ulam–Spirale ans VD–Licht gezogen hat, und ich erlaube mir, ein paar Gedanken dazu schon jetzt zu Papier zu bringen. Ulams Spirale war bisher nicht Gegenstand meiner Überlegungen. Es ist aber sicher recht reizvoll, da einzusteigen. Gespannt bin ich auf alle Fälle darauf, was die anderen mathematisch Interessierten der FG dazu beitragen werden.

Michaels eigentliches Problem hinter seinem Artikel scheint das Problem der dynamischen Speicherzuteilung zu sein? Reservierter Platz für Arrays, die im Verlauf des Rechengangs anderen Arrays mit anderen Dimensionen Platz machen sollen. Bei mir waren es Ende der Sechziger–Jahre Matrizen im Umfeld der Linearen Optimierung (programmieren in ALGOL oder in FORTRAN, war da die Frage). Heute bringt mir diese Frage nichts mehr, aber interessieren tut sie mich nach wie vor.

Primzahlen sind ungeheuer wichtig! Das erhellt schon aus der Tatsache, dass mindestens fünf Mitglieder der Forth–Gesellschaft ihre E–Mails untereinander GPG–verschlüsselt (OpenPGP) austauschen — und diese Art der Verschlüsselung macht stark von Primzahlzerlegungen Gebrauch.

Klar ist zunächst: Es gibt unendlich viele Primzahlen. Bevor ich zur Ulam–Spirale komme, hier der Versuch einer

Definition: 1 wollen wir nicht zu den Primzahlen zählen, 2 sei (die erste und einzige gerade) Primzahl. Eine natürliche Zahl (a positive integer) größer als 2 sei per definitionem Primzahl, wenn sie sich nicht durch (mindestens) eine andere Primzahl (ohne Rest) teilen lässt.

Satz (Euklid, etwa um 300 v.Chr.): Sei n irgendeine natürliche Zahl. Sei $M(n)$ die Menge der ersten n Primzahlen. Dann gibt es eine Primzahl, die nicht Element von $M(n)$ ist.

(In dieser Behauptung steckt natürlich der (modernere und aber auch dehnbarere) Begriff *unendlich viele* drin.)

Beweis: Man bilde das Produkt aller Elemente von $M(n)$ und addiere 1. Die so entstandene Zahl, sie möge $m(n)$ heißen, lässt sich durch keine der in $M(n)$ enthaltenen Primzahlen teilen: Immer bleibt der Rest 1. Es gibt zwei Möglichkeiten: Entweder $m(n)$ ist selbst Primzahl oder $m(n)$ lässt sich durch mindestens eine Primzahl teilen.

Angenommen, $m(n)$ ist Primzahl. Dann haben wir eine Primzahl konstruiert, die nicht Element von $M(n)$ ist, und der Beweis ist erbracht.

Beispiel:
$n = 1$,
$M(n) = \{2\}$,
$m(n) = 3$.

Oder aber $m(n)$ ist keine Primzahl. Dann muss es nach Definition eine Primzahl geben, wir wollen sie $p(n)$ nennen, durch die $m(n)$ (ohne Rest) geteilt werden kann. $p(n)$ kann nicht in $M(n)$ enthalten sein (da ja der Rest sonst 1, und nicht 0, wäre) und der Beweis (der Satzaussage) ist ebenfalls erbracht.

Beispiel:
$n = 8$,
$M(n) = \{2, 3, 5, 7, 11, 13, 17, 19\}$,
$m(n) = 9699691 = 347 * 27953$,
$p(n) = 347$.

(Man konzentriere sich auf das, was im Satz ausgesagt werden soll. Beispielsweise hat dieser Satz nicht die Aufgabe nachzuweisen, dass 347 im eben genannten Beispiel eine Primzahl ist. Er behauptet nur, dass es (mindestens) eine Primzahl mit der genannten Eigenschaft gibt. Auch erhebt dieser Satz nicht den Anspruch, eine Konstruktionsmethode für die ersten n Primzahlen zu liefern.)

Schlussfolgerung: Egal, wie groß wir n wählen, immer gibt es eine Primzahl $p(n)$, die noch größer ist als alle in $M(n)$ enthaltenen Primzahlen.

Und jetzt zur Spirale von Ulam: Für das Weitere behandeln wir 2 als Sonderfall.

Klar ist, dass jede Primzahl außer 2 ungerade ist.

Aus der Spirale von Michael mit den verhältnismäßig wenigen Elementen lässt sich immerhin Folgendes ablesen (Beweis aus der Spiralenkonstruktion heraus):

(1) In jeder Zeile wechseln sich die geraden und die ungeraden (natürlichen) Zahlen ab, in aufeinanderfolgenden Zeilen gegeneinander versetzt.

(2) In jeder Spalte wechseln sich die geraden und die ungeraden (natürlichen) Zahlen ab, in aufeinanderfolgenden Zeilen gegeneinander versetzt.

(3) Jede Diagonale, ob von unten links nach oben rechts oder von unten rechts nach oben links, besteht entweder aus lauter ungeraden Zahlen (wir wollen die betreffende Diagonale *ungerade Diagonale* nennen) oder aus lauter geraden Zahlen (die betreffende Diagonale sei dann *gerade Diagonale* genannt).

Für Diagonalen von unten links nach oben rechts gilt: Gerade Diagonalen wechseln sich mit ungeraden ab.

Für Diagonalen von unten rechts nach oben links gilt: Gerade Diagonalen wechseln sich mit ungeraden ab.

Gerade Diagonalen können offensichtlich keine Primzahlen ungleich 2 enthalten.

Es wechselt sich also immer eine Diagonale, in der Primzahlen vorkommen (können), mit einer Leerdiagonalen ab, in welcher grundsätzlich keine Primzahl vorkommt. Von 2 sehen wir dabei als Ausnahmefall ab.

Schon die (verhältnismäßig kleine) Spirale von Michael legt die Vermutung nahe, dass es keine ungerade Diagonale ohne Primzahlen gibt. Bewiesen ist diese Vermutung mit solch einem Anfangsstück einer Spirale, so groß es auch gewählt sein mag, natürlich nicht.

Aus (1) bis (3) ersieht man, dass es keine Zeile und auch keine Spalte gibt, bei der von vornherein ausgeschlossen werden kann, dass sie Primzahlen enthält.

Schon die (verhältnismäßig kleine) Spirale von Michael legt die Vermutung nahe, dass es keine Zeile und auch keine Spalte ohne Primzahlen gibt. (Eine Vermutung muss natürlich bewiesen oder widerlegt werden. Eine noch so gute Zeichnung kann keinen Beweis ersetzen.)

Bemerkenswert ist an den Primzahlen nicht, dass sie *alle auf Diagonalen liegen*. Wo sollen sie sonst liegen? Sie liegen ja auch *alle auf Zeilen* und sie liegen auch *alle auf Spalten*.

Bemerkenswert ist der Eindruck, den man gewinnt, wenn man sich die 399x399–Spirale bei `http://mathworld.wolfram.com/PrimeSpiral.html` ansieht. Es sieht so aus, als ob man (in einem vorweihnachtlichen Bastelkurs) eine Tapete mit Kleister beschmiert und wahllos mit Druckerschwärze (aus einer Farbdruckerdüse) besprenkelt hat und dann einen feinmaschigen Kamm diagonal einmal so und einmal so hindurchzieht. Man wird bei der genannten größeren Spirale aus dem Internet das Gefühl nicht los, dass sich da Diagonallinien durch das Muster ziehen.

Das ist kein großes Wunder: Jede zweite Zahl in einer Zeile und jede zweite Zahl in einer Spalte ist keine Primzahl. In den Zeilen und in den Spalten haben die (als Tintenpunkte dargestellten) Primzahlen je zu je mindestens den Abstand 2 (mindestens ein Lücke dazwischen), während es überhaupt nicht einzusehen ist, warum in den ungeraden Diagonalen nicht ganze Strecken hinweg ein Punkt am anderen *kleben* sollte. Was man bei den Diagonalen sieht, sind dann Bruchstücke einer Geraden, und das Auge ist leicht geneigt zu extrapolieren. Und noch viel *schlimmer*: Durchs ganze Tableau hindurch wechselt sich immer eine aus Geradenbruchstücken zusammengesetzte Diagonale mit einer gänzlich leeren Diagonalen ab. (Richtig *sehen* tut man das erst dann, wenn man darauf aufmerksam gemacht wird.)

Bei den Zeilen und bei den Spalten können keine ähnlichen Linienbilder entstehen: Jede zweite Zahl in einer Spalte ist gerade und damit (von 2 mal abgesehen) keine Primzahl und nebeneinanderliegende Spalten sind in dieser Hinsicht versetzt. Entsprechend bei Zeilen.

Das Einzige, was einem verwunderlich vorkommen kann und Anlass zu Spekulationen gibt, ist der bei Betrachtung des Spiralenbildes leicht entstehende Eindruck, dass einzelne unter den Diagonalen (von rechts nach links wie von links nach rechts) vom Auge stärker erfasst werden und dass die so hervortretenden Diagonalen in ziemlich gleichen Abständen voneinander auftreten.

Noch ein schneller Gedanke zum Konstruktionsprogramm: Die Ecken der (an sich ja *eckigen*) Spirale (bei Michael) sind: 1 2 3 5 7 10 13 17 21 26 31 37 ... und die Abstände betragen: 1 1 2 2 3 3 4 4 5 5 6 ... Die Diagonale durch 1 von unten rechts nach oben links geht durch die *Punkte* 1 3 7 13 21 31 43 ... mit den Abständen 2 4 6 8 10 12 ..., und diese Abstände selbst haben voneinander den konstanten Abstand 2.

Wie lassen sich in Forth die Primzahlen erkennen und markieren?,

lautet eine der Fragen, die Michael en passant in seinem VD–Artikel stellt. Hierzu fällt einem sofort das allgegenwärtige Sieb des Eratosthenes ein, zu dem es eine reichhaltige Literatur gibt. (Dieses Sieb wird auch gern als Messlatte (benchmark) für Rechengeschwindigkeiten verwendet.) Im VD–Heft 2/1998 steht ein Artikel von mir mit einem Forth–Programm, das Primzahlen auszusieben gestattet. (Martin Bitter entdeckte im VD–Heft 3/1998 ein paar Caveats bei der Übertragung nach ZF.) Ich erreiche das im Real–Mode unter DOS (1 Gigabyte über 32–Bit–Adressen linear anzusprechen, bereitet keine große Schwierigkeit und wird im Programm praktiziert). Das Programm arbeitet mit einem 16–Bit–Forth–System (Turbo–Forth). Man kann aus Geschwindigkeitsgründen natürlich nicht alles in High–Level–Forth programmieren. Aber der in Turbo–Forth eingebaute 16–Bit–Assembler genügt: `EAX` führe ich über `: EAX 66 C, AX ;` ein usw., und Ähnliches bei z.B. `[EBX]` und den 32–Bit–Adressen. Ein expliziter 32–Bit–Assembler wird nicht benötigt.

Im eben genannten VD–Heft 2/1998 findet man auch Hinweise auf zwei Artikel von mir aus den Jahren 1988 und 1989, in denen ich auf Forth und Primzahlen eingehe und beispielsweise das Sieb dadurch abkürze, dass ich die Vielfachen von 2 und die Vielfachen von 3 von vornherein weglasse. Ein anderer Autor (Daniel Flipo, 1989) hat in Antwort darauf vorgeschlagen, auch die Vielfachen von 5 wegzulassen. Da fängt es dann aber an, dass man Platzersparnis und Aufwand gegeneinander abwägen muss.

63963077 63963079 sind Primzahlzwillinge. Um das herauszufinden, habe ich damals mit dem erwähnten Forth–Programm 30 Sekunden gebraucht. $3-5, 5-7, 11-13$ sind offensichtlich Primzahlzwillinge. Kann jemand den Nachweis darüber erbringen, dass es unendlich viele Primzahlzwillinge gibt? Kann jemand zeigen, dass es vielleicht nur endlich viele gibt? Kann jemand die Frage klären, ob diese Frage überhaupt entscheidbar ist? Und dann innerhalb eines welchen Systems?

Bericht der Forth–Gruppe München

Bernd Paysan

Teilnehmer des Treffens
v.l.n.r. Hans Eckes, Daniel Ciesinger, Rafael Deliano, Bernd Paysan, Heinz Schnitter

Michael Kalus wollte wissen, wie es denn um die lokalen Gruppen bestellt ist — und bekam als erste Antwort, dass die Gruppe in Moers definitiv nicht mehr existiert. Die Münchner Gruppe dagegen ist immer noch am Leben, auch wenn die Standortfindung zeitweise problematisch war. Inzwischen sind wir in dem vietnamesischen Restaurant „Chilli Asia“ in der Dachauer Straße 151 gelandet. Das Restaurant hat keine so große Parkplatznot wie die anderen Versuche in München, und ist nicht so weit ab vom Schuss wie das inzwischen leider geschlossene Lakschmi in Oberschleißheim.

Vorteil der Suche war, dass vor jedem Treffen Traffic auf `de.comp.lang.forth` erzeugt wurde, was die Zahl der Teilnehmer proportional zum organisatorischen Chaos in die Höhe trieb. Auch dieses Treffen war gut besucht, weil kurz davor Rafael Deliano mit Vorschlägen zur VD die Diskussion angeheizt hat. Das Thema war dann auch schon weitgehend ausdiskutiert, und spielte auf dem Treffen nur eine unbedeutende Rolle.

Der Hauptinhalt des Treffens war dann das 1–Wire–Interface von Dallas Semiconductors, und wie man ein darüber verbundenes iButton–Thermometer mit Forth am PC ausliest. Die iButtons sind relativ große und robuste Edelstahlgehäuse für Chips, die neben Masse nur noch einen Signal–Eingang haben, eben das 1–Wire–Interface; dieses liefert auch die Stromversorgung für den Chip im iButton. Die meiste Zeit ist das 1–Wire–Interface also mit einem Pull–Up auf 5V gezogen; für die eigentliche Übertragung wird dann kurz (0) oder ganz kurz (1) auf 0 gezogen. Es können „beliebig“ viele Geräte am Bus hängen, wobei natürlich die kapazitive Belastung reale und relativ enge Grenzen setzt.

An den PC schließt man dann einen iButton–Reader an; der steckt am USB–Port. In diesem Reader ist ein Wandler von USB nach RS–232, und ein Chip von Dallas, der die serielle Schnittstelle dann nach 1–Wire wandelt. Das Forth–Programm spricht also über eine (etwas kompliziert aufgesetzte) serielle Schnittstelle mit dem iButton.

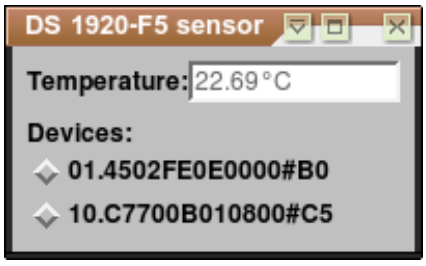


Bild 1: Sensor

Man sieht in Bild 1 deutlich, was das 1–Wire–Interface kann: Neben der eigentlichen Sensor–Information (dem Temperaturwert) werden auch noch die Adressen der am Bus hängenden Geräte ermittelt. Jedes Gerät hat eine eindeutige 64–Bit–Adresse, mit der es am Bus individuell angesprochen werden kann; hängt nur ein Gerät am Bus, kann man auf die Adresse auch verzichten.

Hier hängen zwei Geräte am Bus, wovon eines ein Serien–Nummern–Chip ist, und das andere der Temperatursensor. Da uns der Serien–Nummern–Chip nicht in die Quere kommen kann, lesen wir die Temperatur ohne Adresse:

```
: readout
   noaddress
   $44 cmdr &750 ms \ convert temperature
   noaddress
   $BE cmdr \ read scratchpad
   dummies9 ;
```

Forth–Gruppen regional

Mannheim
Thomas Prinz
Tel.: (0 62 71) – 28 30 (p)
Ewald Rieger
Tel.: (0 62 39) – 92 01 85 (p)
Treffen: jeden 1. Dienstag im Monat
Vereinslokal Segelverein Mannheim e.V. Flugplatz Mannheim–Neuostheim

München
Bernd Paysan
Tel.: (0 89) – 79 85 57
`bernd.paysan@gmx.de`
Treffen: Jeden 4. Mittwoch im Monat um 19:00, im Chilli Asia Dachauer Str. 151, 80335 München.

Hamburg
Küstenforth
Klaus Schleisiek
Tel.: (0 40) – 37 50 08 03 (g)
`kschleisiek@send.de`
Treffen 1 Mal im Quartal
Ort und Zeit nach Vereinbarung (bitte erfragen)

Mainz
Rolf Lauer möchte im Raum Frankfurt, Mainz, Bad Kreuznach eine lokale Gruppe einrichten.
Mail an rowila@t-online.de

Gruppengründungen, Kontakte

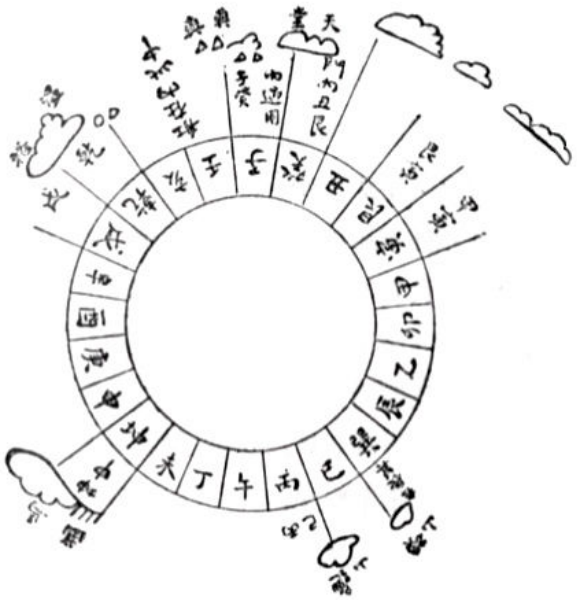
Hier könnte Ihre Adresse oder Ihre Rufnummer stehen — wenn Sie eine Forthgruppe gründen wollen.

µP–Controller Verleih

Carsten Strotmann
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix)	**Klaus Schleisiek–Kern** Tel.: **(0 40) – 37 50 08 03** (g)
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 43 51) – 71 22 17** (p) Fax: **– 71 22 16**
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 70 44)–90 87 89** (p)



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 Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.

Einladung zur

Forth–Tagung 2008

vom 25. bis 27. April 2008

im Kloster Roggenburg (`www.kloster-roggenburg.de`)

Klosterstraße 2, 89297 Roggenburg (bei Ulm)



Programm

Donnerstag, 24.04.2008
Tag *null* Anreise für die *frühen Vögel*

Freitag, 25.04.2008
15:00 Uhr Beginn der Tagung

Samstag, 26.04.2008
14:00 Uhr Exkursion

Sonntag, 27.04.2008
09:00 Uhr Mitgliederversammlung
14:00 Uhr Ende der Tagung

Anreise siehe: `http://www.kloster-roggenburg.de/klro.04/anfahrt/anfahrt.html`



Zur Geschichte des Klosters Roggenburg

Das Kloster Roggenburg wurde 1126 gegründet. 1444 wurde das Stift zur Abtei erhoben und erhielt 1544 die Reichsunmittelbarkeit. Am 4. September 1802 wurde das Reichsstift Roggenburg von Bayerischem Militär besetzt und der Konvent aufgelöst. Seit der Wiederbesiedelung des Klosters bemüht sich die Klostergemeinschaft um die Sanierung des Gebäudes und um ein Nutzungskonzept für das Klosterareal. 2001 wurde der „Prälatengarten“ als Haus für Kunst und Kultur eingeweiht, ein Jahr später erfolgte die Inbetriebnahme eines Neubaus des Bildungszentrums für Familie, Umwelt und Kultur, sowie des Klostergasthofes und des Klosterladens. Weitere Informationen zum Kloster: `http://www.kloster-roggenburg.de/klro.04/kloster/index.html`



Quellen: `http://www.kloster-roggenburg.de`