Vierte Dimension

1/2008

Das Forth–Magazin

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

In dieser Ausgabe:

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 GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Bergstraße 10	D–18057 Rostock
Tel.: +49 381 496800-0	Fax: +49 381 496800-29

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.

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,

ich begrüße Euch zu einer neuen Ausgabe des Forth–Magazins. Wir schreiben das Jahr 2008 und die Vierte Dimension erscheint nun bereits im 24. Jahr!

Ich freue mich, dass wir in diesem Heft Artikel von neuen und auch alten Autoren lesen können: Wir erfahren von Reinhold Straub, wie er Forth und die Script–Sprache Tcl mischt. Luca Masini berichtet uns über das Problem 146 des Euler–Projekts und seine Lösung in Forth. Bernd Paysan stellt vor, wie man den 1–Wire–Bus in Forth anspricht — Michael Kalus und ich führen vor, wie man Texte so verwürfelt, dass sie dennoch lesbar bleiben, oder nicht?

Wahrscheinlich ist es dem einen oder anderen Leser ja schon aufgefallen: Der Artikel von Rafael Deliano über das FOCUS–Zahlensystem, der im Doppelheft 2007/3+4 erschienen ist, sprach ganz selbstverständlich, ohne weiteren Kontext über LNS (Logarithmische Zahlensysteme). Nun — da ist mir ein bedauerlicher faux pas unterlaufen: Vor dem FOCUS–Artikel hätten nämlich zwei Grundlagenartikel zu Logarithmischen Zahlensystemen erscheinen sollen. Rafael Deliano macht dementsprechend seinem Ärger in de.comp.lang.forth Luft:

> Betreff: VD 3+4/07
> Datum: Fri, 21 Dec 2007 13:38:18 +0100
> Autor: Rafael Deliano
> ... *Als Autor ist man in solchen Doppelheften ohnehin Kollateralschaden*: der Artikel über das FOCUS-Zahlensystem war der 3. in einer Reihe, alle drei lagen in der Redaktionsschublade. Der erste war Einführung in LNS (d. h. Logarithmic Number Systems), der zweite ein kleine konventionelle Implementierung. Der letzte war dann eben etwas knapper geschrieben, weil ich beim Leser voraussetzte, daß er die Grundlagen schon hat.

Mir tut das sehr leid. Natürlich habe ich nicht beabsichtigt, die Artikel in der falschen Reihenfolge zu drucken, noch weniger, dass sich irgendjemand als *Kollateralschaden* fühlen muss. Die fehlenden Artikel findet Ihr nun in dieser Ausgabe. Bitte lest sie und wendet Euch dann auch noch einmal dem FOCUS–Artikel zu.

In diesem Zusammenhang möchte ich darauf hinweisen, dass Rafael diese und noch viele weitere lesenswerte Artikel in seiner Zeitschrift *embedded* (`http://www.embeddedforth.de/`) herausgibt. Dankenswerterweise erhalten FG–Mitglieder die *embedded*–Ausgaben kostenlos (sonst 5€/Ausgabe). Zumindest einen unserer neuen Mitglieder — Marcel Hendrix, Volker Vanoni und Jürgen Pfitzenmaier, die ich an dieser Stelle recht herzlich begrüße — hat auch das wohl bewegt, der Forth–Gesellschaft beizutreten.

Seit Februar 2008 gibt es den Forth–Chat im Channel `#forth-ev` im Internet–Relay–Chat (*.de-IRCnet), zu dem wir uns Mittwochs 20:00 Uhr treffen, um über Forth zu diskutieren. Jeder Interessierte ist herzlich eingeladen, auch daran teilzunehmen. Was man dafür machen muss, und auch die Log-Files der bisherigen Diskussionen, lässt sich über `www.forth-ev.de` abrufen.

Die Forth–Tagung steht kurz bevor. Kurzentschlossene können sich noch anmelden. Weitere Details findet Ihr auf der Rückseite dieses Hefts.

So — Nun viel Spass beim Lesen der durchweg interessanten Artikel dieser Ausgabe.

Ulrich Hoffmann

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

Projekt Euler

Danke an Luca Masini (vergleiche auch Lucas Artikel auf Seite 16), der uns auf das Projekt Euler `http://projecteuler.net/` aufmerksam gemacht hat. Aufgabe hier ist es, 186 verschiedene mathematische Probleme in einer Programmiersprache seiner Wahl (wir nehmen natürlich Forth :-) zu lösen. Aufgabe 1 lautet „Addiere alle natürlichen Zahlen unter 1000, die Vielfache von 3 oder 5 sind“. Nun das ist schnell programmiert und braucht nur ein paar Zeilen. Aufgabe 186 über den Vernetzungsgrad in einem Telefonnetz ist da schon anspruchsvoller.

Wer traut sich an die Aufgaben ran und veröffentlicht sie dann hier in der Vierten Dimension?

Ulrich Hoffmann

Forth Team Programming

Liebe Redaktion, hier die Übersetzung einer kurzen Passage aus c.l.f, die ich mit einer Frage an die Leser verknüpfen möchte.

Sind wir bereit für eine Open–Source–Forth–Library?

comp.lang.forth
Elizabeth D Rather, 7. März 2008, 1:06 pm

> m-coughlin wrote:
> ...Programmierer, die große Forth–Systeme schreiben, publizieren ihren Code nicht für andere Nutzer. Es ist VIEL schwieriger, Forth–Quellen für andere zu schreiben als für sich selbst, erinnert man sich doch selbst, was die Worte tun. Zu wissen und sich zu erinnern, was andere Programmierer getan haben, ist das Problem.
>
> Elisabeth:
> Programmierer, die große Forth–Systeme schreiben, arbeiten in Teams. Erfolgreiche Teams (die Forth oder irgendeine Sprache nehmen) entwickeln Strategien, um innerhalb des Teams das Codesharing, die Dokumentation und die Lesbarkeit zu maximieren. Akzeptiert man einmal das Konzept, den Code für das Team zu schreiben, fällt das in Forth keinesfalls schwerer als in jeder anderen Sprache. Forth ist seit über dreißig Jahren erfolgreich in Team–Programmier–Projekten in Gebrauch.
> Grüße, Elisabeth

Viele Grüße, Michael

forth_false und ANS Forth

Am 23. Dezember 2007 findet sich in de.comp.lang.forth ein Beitrag von Anton Ertl, den wir mal als Leserbrief werten :-)

> Betreff: forth_false und ANS Forth (was: VD 3+4/07)
> Datum: Sun, 23 Dec 2007 18:18:49 GMT
> Autor: (Anton Ertl)
>
> Rafael Deliano writes:
> > Die Diskussion in dclf ist einigermaßen unverdaut auch
> > in die VD übergeschwappt.
>
> Naja, vielleicht lesen dann ja auch einige der VD–Leser Folgendes:
>
> Fred Behringer behauptet in seinem FALSE–Artikel, dass ANS–Forth–Programme nicht auf ANS–Forth–Systemen laufen, und belegt das damit, dass Ben Hoyts FALSE–Interpreter <`http://wouter.fov120.com/files/lang/false/forthfalse.zip`> nicht auf zwei (relativ obskuren) Forth–Systemen läuft, die von sich behaupten, ANS–Forth–Systeme zu sein.
>
> Wenn das so wäre, wäre der Standard ein totaler Fehlschlag, denn die Kompatibilität zwischen Programmen und Systemen ist das wichtigste Ziel eines Standards (wenn auch einige behaupten, dass die Kompatibilität der Programmierer auch wichtig ist).
>
> Ich habe einmal die Probe aufs Exempel gemacht und das gleiche Programm auf einigen ANS–Forth–Systemen hier probiert, jeweils ohne irgendwelche magischen Phrasen zu verwenden, die optionale Teile nachladen:
>
> gforth 0.6.2: läuft.
> iforth 2.1.2541: läuft.
> bigFORTH 2.1.6: läuft.
> VFX–Forth for Linux IA32 4.05 Alpha
> [build 0118]: läuft nicht, es fehlt
> zumindest FLUSH-FILE und "0." (also Doubles).
>
> Bei VFX–Forth könnte man sich vielleicht mit magischen Phrasen behelfen, aber auf die Schnelle habe ich in der Doku nichts gefunden.
>
> Meine Schlussfolgerung ist, dass ANS–Forth durchaus erfolgreich ist, zumindest soweit es dieses Beispiel und einige andere betrifft. Wenn einige Forth–Systeme es nicht richtig unterstützen, dann bleiben einem folgende Optionen:
>
> – Das als Fehler an den Entwickler melden, so dass er es ausbessert (werde ich gleich einmal mit VFX–Forth machen).
> – Auf ein System wechseln, das ANS Forth richtig unterstützt.
>
> - anton
> --
> M. Anton Ertl
> Some things have to be seen to be believed
> Most things have to be believed to be seen
> anton@mips.complang.tuwien.ac.at
> http://www.complang.tuwien.ac.at/anton/home.html

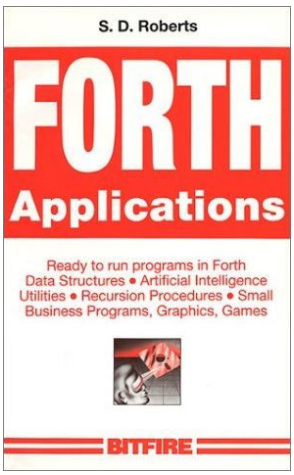
und als Reaktion darauf von Rafael Deliano

> Betreff: Re: forth_false und ANS Forth (was: VD 3+4/07)
> Datum: Mon, 24 Dec 2007 11:38:44 +0100
> Autor: Rafael Deliano
>
> >> Die Diskussion in dclf ist einigermaßen unverdaut
> >> auch in die VD übergeschwappt.
> > Naja, vielleicht lesen dann ja auch einige der VD–Leser
> Wird halt wieder unverdaut zurückschwappen.
>
> > Meine Schlussfolgerung ist, dass ANS–Forth durchaus
> > erfolgreich ist, zumindest soweit es dieses Beispiel
> > und einige andere betrifft.
> Totes Rennen.
>
> An Routinen, die ich nicht selbst codiert habe, verwende ich z.B. nur eine Funktion von Klaus Kohl für die Quadratwurzel, die der vermutlich mal aus einem FORTH–FFT–Programm in DrDobbs abgezweigt hat und später in VD veröffentlichte. Ich habe hier auch im Fundus noch ein DES in Forth aus clf das aber 32 Bit ist, so dass ich es noch nicht testen konnte, weil ich es auf 16 Bit ändern müsste.

> „Portabilität“ wäre relevant, wenn nennenswert nützliche und dokumentierte Source veröffentlicht würde. Von Google gibts bekanntlich experimentelle Suche nach source http://www.google.com/codesearch/advanced_code_search?hl=de, die unter „F“ aber eben nur „Fortran“ hat. Ist nicht die Schuld von google: es gibt keinen nennenswerten frei verfügbaren für Endanwender relevanten Sourcecode in Forth.
>
> Insofern halte ich es mit Charles Moore "give me your idea and I embed it in my code". Es ist auch für die VD besser, Routinen system– und programmiersprachenunabhängig z.B. als Flowcharts zu veröffentlichen und dem Leser die Codierung in sein System zu überlassen.
>
> MfG JRD

Interessant wäre, zu erfahren, wie es anderen Forth–Programmierern ergeht.
Zeit für einen Leserbrief? Ulrich Hoffmann

Besprechung von *FORTH Applications* von S. D. Roberts



Schon 1989 erschien das Buch *FORTH Applications* (ISBN 0-911827-00-5) von S. D. Roberts bei Elcomp Publishing. Noch heute kann man es bei Amazon.com für $9.95 plus Porto/Verpackung kaufen.

In dem 154 Seiten dünnen, englisch–sprachigen Büchlein erläutert Roberts zunächst die Implementierung von Zahlen– und Text–Ein– und –Ausgaben in Forth, spricht dann über Strings und geht auf Forth–Realisierungen der dynamische Datenstrukturen *gelinkte Liste*, *binäre Bäume* (inkl. *AVL–Bäume*) ein.

Das nächste Kapitel widmet sich Entscheidungsbäumen und gibt zwei kurze Beispiele aus dem Bereich der *künstlichen Intelligenz*: Klassifizierung von Mineralien und Fehlerdiagnose einer elektrischen Schaltung. Es endet mit einer knappen Beschreibung assoziativer Strukturen zwischen Worten.

Im Kapitel *Tools for Development* werden Abwandlungen von bereits in den Forth Dimensions erschienenen Definitionen für das Setzen von Breakpoints [1] und das Suchen von Texten in Screens [2] vorgestellt.

Tabellengestützte Berechnung von Festpunkt–Sinus beziehungsweise –Cosinus und die Programmierung einer *Turtle–Graphic* ist das Thema des folgenden Kapitels. Die grafische Darstellung basiert auf einer CGA–(=Color Graphics Adapter)–Graphik mit den Zeichenprimitiven `WHITE`, `BLACK`, `HGR`, `PLOT` und `LINE`. Leider wiederholt diese Implementierung die wiederholt anzutreffende Ungeschicktheit, die Werte der Sinus–Tabelle auf dezimal 10000 zu skalieren, anstatt auf hexadezimal 10000, und so $\frac{5}{6}$ der Genauigkeit verschenkt werden.

Anhand der *Türme von Hanoi* wird dann die Definition direkt–rekursiver Worte beschrieben. Es wird das Wort `SELF` definiert, das erlaubt, einen rekursiven Aufruf des aktuell definierten Wortes zu kompilieren. Außerdem wird die `SMUDGE`–Technik beschrieben, um den Namen der aktuellen Definition bereits während der Definition des aktuellen Wortes sichtbar zu machen. Heutige Standard–Systeme handhaben Rekursion mithilfe des `SELF` entsprechenden Wortes `RECURSE` und benötigen die hier beschrieben Technik nicht mehr.

Das nächste Kapitel widmet sich einem Programm zum formatierten Drucken von Rechnungen. Die Daten werden in einer Beldschirmmaske erfasst in Forth–Böcken abgelegt und dann selektiv zum Drucken aufbereitet.

Ein kleines Zahlenspiel, in dem die Ziffern `0` und `1` in einer 3x3–Matrix nach vorgegebenen Regeln in eine Zielkombination gebracht werden müssen, ist Gegenstand des nur 5 Seiten umfassenden Kapitels *Brain Teaser*.

Den Abschluss bildet ein minimalistisches `BLOCK`–basiertes Datenbanksystem und eine darauf aufsetzende Adressverwaltung.

Die Beispiele im Buch sind teilweise für MVP–Forth, teilweise für LMI–Forth (Forth–83) formuliert. Sie sind durchgängig in GROSSBUCHSTABEN geschrieben, was die Lesbarkeit nicht unbedingt fördert. Stack–Kommentare finden sich gelegentlich, ansonsten sind die Beispiele im wesentlichen unkommentiert und nur im Prosatext erläutert. Dadurch erschließen sich die Beispiele nicht unmittelbar und müssen intensiv durchgearbeitet werden, um sie zu genau zu verstehen.

Alles in Allem ist *FORTH Applications* eine Sammlung von diversen, durchaus auch interessanten, älteren Forth–83/FIG–Forth–Programmen. Zum Ende des Buches gewinnt man allerdings den Eindruck, als würden nicht näher zusammenhängende Programme einfach aneinandergereiht.

Eher akademische Beispielprogramme als *Applications* zu bezeichnen, ist sicher irreführend. Unter Applikationen würde man wohl eher praktisch einsetzbare Anwendungen verstehen. Dennoch ist das Buch eine interessante Lekture für den Forther, der sich von unterschiedlichen Forth–Dialekten nicht abschrecken lässt und dem es nichts ausmacht, sich durch schlecht kommentierten und aus heutiger Sicht schlecht formatierten Forth-Quellcode durchzubeißen. Ulrich Hoffmann

Literatur

[1] *Breakpoint*, Forth Dimensions, V5#1
[2] *An Easy Directory System*, Forth Dimensions, V5#3

Neues aus dem Forth-Büro

Rolf Schöne gibt auf der kommenden Forth–Tagung im April die Geschäfte des Forth–Büros weiter. Dazu lässt er uns wissen:

> Danke für die gute Zusammenarbeit. Ihre Partner werden in Zukunft Andrea und Ewald Rieger sein.
> Rolf Schöne

Danke auch Dir, Rolf. Die Zusammenarbeit mit Dir hatte sich sehr gut eingespielt. Schade, dass Du nicht weitermachst. Ich wünsche Dir im Namen aller Mitglieder alles Gute für die Zukunft. Ulrich Hoffmann

Logarithmische Zahlensysteme

Rafael Deliano

Multiplikation und Division sind bei Festkommazahlen aufwändig. Vom Rechenschieber ist bekannt, dass bei logarithmischer Zahlendarstellung diese Funktionen einfach sind, sie reduzieren sich auf Addition und Subtraktion.

Für Potenzen und Wurzeln sind nur noch Multiplikation und Division nötig, für Quadrat und Quadratwurzel genügen Shiftbefehle (Bild 1). Andererseits sind Addition und Subtraktion logarithmierter Zahlen schwierig.

Linear	LNS
$C = A * B$	$C = A + B$
$C = A / B$	$C = A - B$
$C = \sqrt[n]{A}$	$C = A / n$
$C = A^n$	$C = A * n$

Bild 1: Operationen

Die primitive Lösung für z.B. Multiplikation war somit über eine log–Tabelle das Zahlensystem zu wechseln, die Addition auszuführen und dann über eine invlog–Tabelle zurück auf Festkomma zu wandeln (Bild 2). Mit ersten Verfeinerungen wurde das bereits in den 60er Jahren propagiert [1].

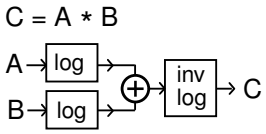


Bild 2: Multiplikation

Verwendet wird heute durchwegs der binäre Logarithmus (Bild 3). Man ist, wie man am Funktionsverlauf sieht, auf positive Zahlen beschränkt.

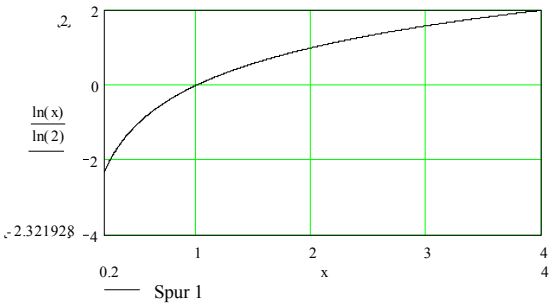


Bild 3: $y = ld(x)$

Für bipolare Signale wird also ein zusätzliches Vorzeichenbit nötig. Zudem muss das Signal größer 1,0 sein. Dagegen hilft Umformulierung der Gleichungen, um günstige Koeffizienten zu erhalten (Bild 4).

$$y = \frac{x}{0,1} = x * 10$$
$$y = x * 0,1 = \frac{x}{10}$$

Bild 4: Umwandlung für günstige Koeffizienten

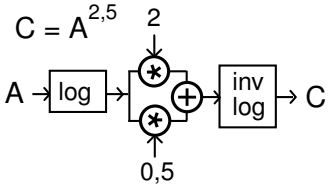


Bild 5: Exponentialfunktion

Besonders bei Exponentialfunktionen ergeben sich oft überraschend einfache Lösungen (Bild 5). Durch die weite Dynamik ist auch das Überlaufverhalten günstig. Jedoch sind in den meisten Gleichungen Additionen nötig (Bild 6), was Formatwandlung bedeutet. Abgesehen von der Rechenzeit besonders wegen der Quantisierungsfehler kritisch. Dem kann man natürlich durch erhöhte Auflösung begegnen. Aber wenn die Umwandlung durch Tabellen erfolgen soll, stößt man schnell an praktische Grenzen.

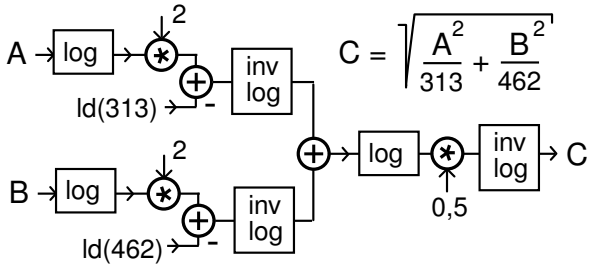


Bild 6: gemischte Rechnung

Signalverarbeitung

Der Speicherbedarf der ROMs war und ist einer der Pferdefüße, die den praktischen Einsatz von LNS („Logarithmic Number Systems“) bisher aufgehalten haben. Obwohl heute manchmal als Ersatz für Float propagiert, wurde das Format ursprünglich speziell für Signalverarbeitung als geeignet angesehen [2]. Denn A/D– und D/A–Wandler waren auf 8–12 Bit beschränkt und damit blieb der Speicherbedarf der Tabellen erträglich. Anders als Festkommazahlen, die konstante Schrittweite haben, werden nichtlinear kleine Werte mit größerer Auflösung dargestellt. Damit erhält man bei DC–freien Signalen über einen weiten Dynamikbereich einen günstigen Signal/Rausch–Abstand, selbst wenn man nur mit kurzen Wortlängen arbeitet. Im Telefonsystem macht man bei PCM–Codecs seit langem von dieser Eigenschaft gebrauch (Bild 7).

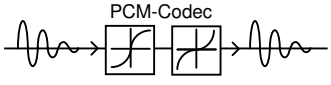


Bild 7: Telefonie

Ein PID–Regler kann als Filter angesehen werden. Abhängig von der Implementierungsform wird der Filterkern im eingeregelten Zustand zumindest am Eingang

fast DC–frei sein, das Fehlersignal e ist auf null ausgeglichen. LNS kann damit zweckmäßig sein (Bild 8).

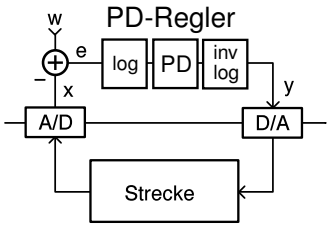


Bild 8: Regler

Außer weiter Dynamik kann vereinfachte Arithmetik durch günstige Stromaufnahme von Vorteil sein. Mit Ausrichtung auf Anwendung in Hörgerät wurde in [50] ein experimenteller Filterbankprozessor vorgestellt, der durch logarithmische A/D– und D/A–Wandler ergänzt werden sollte [51], so dass die Tabellen für die Formatwandlung prinzipiell entfallen. Manche Sensorsignale wie Licht oder Lautstärke benötigen viele Dekaden Dynamik und damit einen 14–16–Bit–A/D–Wandler. Wenn man solche Signale aber mit einem analogen Logarithmierer behandelt, genügt in manchen Anwendungen ein 8–Bit–Wandler (Bild 9). Die Kennlinie der Analogschaltung ersetzt hier auch die digitale Tabelle, allerdings muss das LNS–System auf sie abgestimmt werden.

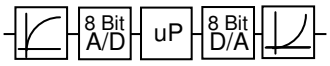


Bild 9: Analoge „Tabellen“

Additionstabellen

Die Umwandlung mit Tabellen führt zu Rundungsfehlern. Man kann verhindern, dass sich diese akkumulieren, indem man nur für Ein– und Ausgabe auf Festkomma wandelt. Und ansonsten im LNS–Format rechnet. Damit benötigt man aber je eine weitere Tabelle für Addition und Subtraktion. Die Vorteile überwiegen jedoch, im Verlauf der 70er Jahre hat sich das bald als die bevorzugte Variante eingebürgert.

Einen nicht zu unterschätzenden Beitrag zur Popularisierung hat dabei die Publizierung des FOCUS–Zahlensystems [4] von Lee und Edgar 1977 bzw. 1979 geleistet, da dieses die konkrete Implementierung auf einem 8080–Mikroprozessor beschrieb und dafür auch die Listings veröffentlicht wurden.

Obwohl die praktischen Anwendungen bisher spärlich geblieben sind, hat die Zeit seither für LNS gearbeitet. Einerseits weil immer mehr Speicher billig verfügbar wurde. Andererseits hat eine Handvoll Wissenschaftler dafür gesorgt, dass über Interpolation und Eliminierung vorhandener Redundanzen die Tabellen verkleinert wurden, ohne dass Zugriffszeit und Genauigkeit zu sehr litten.

In den 90er Jahren wurde versucht, LNS in Form von Softwaresystemen kommerziell und patentiert auf den Markt zu bringen. Marktführer dürfte die 1993 von Les Pickett gegründete Firma gewesen sein [10]–[12]. Anwendung z.B. die Luftdrucksteuerung im Passagierraum der Boeing 767 und ähnlicher Flugzeuge seit 1977. Wobei der Hersteller das System nicht als LNS, sondern als EFP („Exponential Floating Point“) bezeichnete. Etwa anno 2000 ist die Firma über Turbulenzen an der Börse gestolpert.

Nicht viel besser ging es dem Unternehmen von Mark Winkel [20], der basierend auf von Mark Arnold entwickelter Technik [21] das gleiche Geschäftsmodell verfolgte, aber auch noch Picketts Patent umgehen musste. Diese Gruppe scheint aber mit Zielrichtung auf FPGA–Cores und ICs weiterhin aktiv zu sein.

Auch große Halbleiterhersteller wie Motorola und VLSI Technology waren aktiv und haben zumindest Patente angemeldet. Motorola stellte 1999 einen experimentellen 32–Bit–Coprozessor vor [30] [31], der 64 Arithmetikeinheiten enthält und damit 10 (US) Billionen Flops pro Sekunde erreicht. Anwendung war Video.

Der Esprit–finanzierte „European Logarithmic Microprocessor“ hat es bisher zu Patent, einer FPGA–Implementierung [40] und einem teilweisen ASIC–Prototypen [41] gebracht. Die Ziele sind mit 32 Bit Wortbreite und damit Annäherung an IEEE–Float hoch gesteckt [42]. Finanziell unterstützt von der englischen Regierung entstand dann unter Coleman 2005 die Firma Northern Digital als spin–off der Newcastle University. Samples in 0,18–micron CMOS mit 125 MHz Takt sind wohl verfügbar. Da die Firma aber auf Vermarktung von IP ausgelegt ist, kann man das IC einstweilen noch nicht kaufen. Angepeilte Anwendung auch hier Video.

[1] Mitchell „Computer Multiplication and Division using Binary Logarithms“ IEEE Trans. Comput. August 1962
[2] Kingsbury, Rayner „Digital filtering using logarithmic arithmetic“ Electronic Letters vol. 7 1971
[4] Lee, Edgar „The FOCUS number system“ IEEE Trans. Comp. Nov 1977
[10] `www.logpoint.com`
[11] L. Pickett „Method and Apparatus for Exponential/Logarithmic Computation“ U.S. Patent 5.197.024.23 March 1993
[12] „Software Math Coprozessor Rivals Hardware Speeds“ Electronic Design August 1997
[20] `www.xlnresearch.com`
[21] Arnold „Method and Apparatus for Fast Logarithmic Addition and Subtraction“ U.S. Patent 5.337.266.9 August 1994
[22] „Fast Math: Software Faster Then a Coprozessor“ C User's Journal Juli 1991
[30] Pan „A 32b 64–matrix parallel CMOS processor“ 1999 IEEE Intl. Solid–State Circuits Conf.
[31] „10bn Flops using logs“ Electronics Weekly Feb. 1999
[40] Kadlec et Al. „32–bit Logarithmic ALU for Handel C2.1 and Celoxica DK1“
[41] Chester, Coleman „Development of a High–Speed 32b Real Arithmetic Core for DSP and Graphics Applications using Logarithmic Number System Techniques“
[42] Coleman, Chester „A 32–Bit Logarithmic Arithmetic Unit and Its Performance Compared to Floating Point“
[50] Morley, Sullivan, Engel „A VLSI FIR Digital Signal Prozessor Using Logarithmic Arithmetic“ in: Broderson „VLSI Signal Processing III“ IEEE 1988
[51] Engel, Morley, Kwa, Fretz „Integrated Circuit Logartihmic Digital Quantizers with Applications to low–power data interfaces for speech processing“ in: Moscovitz „VLSI Signal Processing IV“ IEEE 1991

Simples LNS

Rafael Deliano

Die Darstellung in [62] passt direkt für Mikroprozessoren. Die Z80–Source wurde auf FORTH und HC08 portiert.

Das Grundschema hat sich seit Mitchell [1] nicht verändert.

$$ld(x) = M + \frac{X - 2^N}{2^N}$$

Integer 0...15 — Fraction 0...0,99

Bild 1: Aufspaltung

Man sucht sich mit der Funktion SCANBIT [61] das oberste gesetzte Bit. Dessen Position ergibt den Integer-Teil des Logarithmus, die folgenden Bits die Fraction (Bild 1). Für den groben Logarithmus BITLOG in [61] wurden als Fraction nur 3 Bits verwendet. Bei so niedriger Auflösung kann man den Bereich von ld(1) bis ld(2) als Gerade annehmen, wie dort geschehen (Bild 2).

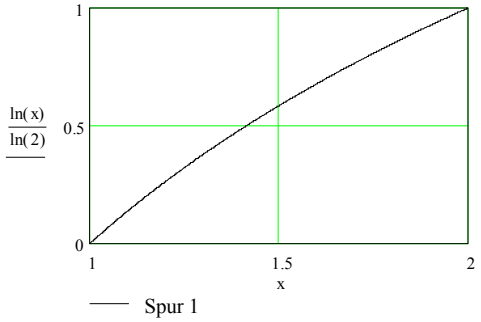


Bild 2: Verlauf ld(x)

Eine 8–Bit–Fraction ergibt deutlich bessere Auflösung, man muss nun aber die Kurve in Bild 2 annähern. In [62] wird dafür eine 256–Byte–Tabelle (Bild 3, 4) verwendet.

LOG

Die Routine liefert eine 16–Bit–Integerzahl die den Dezimalpunkt genau in der Mitte hat (Bild 6), d.h. $y = 256 * \mathrm{ld}(x)$. Dass oben 4 Bits unbelegt sind, ist sehr erwünscht, weniger Probleme mit Überlauf, wenn man mit fester 16–Bit–Wortlänge rechnet.

$x := 0..255$
$g_x := 256 \cdot \frac{\ln(1+\frac{x}{256})}{\ln(2)}$
$d_x := \mathrm{runden}(g_x)$

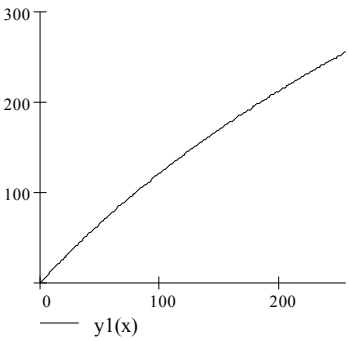


Bild 3: Berechnung L-TAB

ILOG

Im Rückwandlungsbefehl muss man ersteinmal prüfen, ob das obere Byte größer 0F ist, und dann wegen Überlauf direkt FFFF ausgeben. In Assembler wird man die Daten wie gezeigt in einem 24–Bit–Register anordnen und dann abhängig vom oberen Byte entsprechend oft hochschieben.

Speicherverbrauch und Geschwindigkeit sowohl für Assembler als auch nanoFORTH auf einem 2,45–MHz–68HC08 sind in Tabelle 1 angegeben. Zuzüglich 512 Byte für die Tabellen.

$x := 0..255$
$g_x := 256 \cdot \left[\exp\left[(\frac{x}{256}) \cdot \ln(2)\right] - 1\right]$
$d_x := \mathrm{runden}(g_x)$

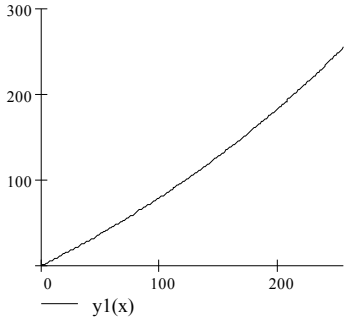


Bild 4: Berechnung IL-TAB

In Listing 2 sind die beiden Beispielprogramme aus [62] aufgeführt. Vgl. hierzu die Flussdiagramme in Bild 5, 6 des einführenden Artikels auf Seite 7.

	FORTH		Assembler	
	Byte	usec	Byte	usec
LOG	103	300-700	39	20-50
ILOG	171	200-400	50	40-100

Tabelle 1: Speicher & Rechenzeit

Für $x^{2,5}$ sind in Bild 7 die Fehlerplots. Der absolute Fehler steigt natürlich, wenn die Zahlen größer werden, bis dann endgültig Überlauf eintritt. Bei relativem Fehlerplot hat man aber nur bei sehr kleinen Zahlen und im Überlauf extremen Fehler.

Literatur

[61] emb (9) oder VD 1/2004 „Einfacher Logarithmus“
[62] Guagliano „Simple logarithms speed microprocessors math operations“ EDN 24. Jan 1985 in: Hickman „Electronic Circuits, Systems an Standards: the Best of EDN“ Butterworth-Heinemann 1991

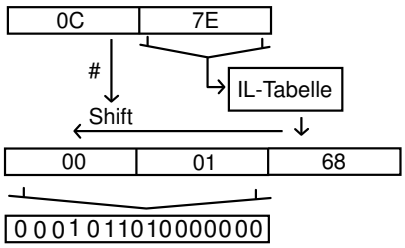


Bild 5: Berechnung ILOG

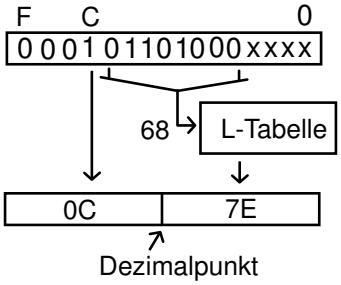
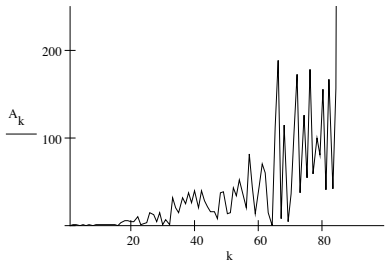


Bild 6: Berechnung LOG

$k := 1..100 \quad A_k := |a_k - k^{2,5}|$



$A_k := \frac{k^{2,5}}{a_k}$

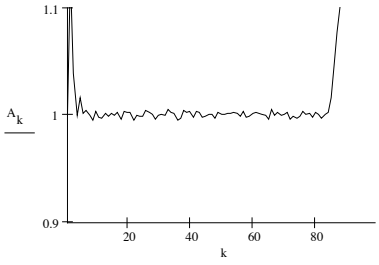


Bild 7: Fehlerplot $x^{2,5}$

Listing 1: Tabellen, LOG, ILOG

```
TABLE L-TAB
   00 C, 01 C, 03 C, 04 C, 06 C, 07 C, 09 C, 0A C,
   ...
   FA C, FB C, FC C, FC C, FD C, FE C, FF C, FF C,

TABLE IL-TAB
   00 C, 01 C, 01 C, 02 C, 03 C, 03 C, 04 C, 05 C,
   ...
   F5 C, F6 C, F8 C, F9 C, FA C, FC C, FD C, FF C,

: LOG           \ ( UN1 - UN2 )
   DUP 99 /?ERROR \ error if UN1 = 0
   0F 0 DO
      DUP 8000 AND
      IF I LEAVE ELSE 1<SHIFT THEN
   LOOP
   0F SWAP - 8<SHIFT
   SWAP 1<SHIFT 8SHIFT> L-TAB + C@
   OR ;

: ILOG          \ ( UN1 - UN2 )
   DUP F000 AND
   IF DROP FFFF    \ overflow UN2 = FFFF
   ELSE.
     DUP FF AND IL-TAB + C@ 100 OR
     SWAP 8SHIFT>
     DUP 08 =
     IF DROP         \ no shifts
     ELSE DUP 08 U<
     IF   7 SWAP - 0 DO 1SHIFT> LOOP
     ELSE 9      - 0 DO 1<SHIFT LOOP
     THEN
     THEN
   THEN ;
```

Listing 2: Beispiele

```
: X^2,5    \ ( UN1 - UN2 )
   LOG
   DUP 1<SHIFT SWAP 1SHIFT> +
   ILOG ;

: FUNCTION    \ ( B A - C )
   LOG
   1<SHIFT
   084A -         \ 313d
   ILOG
   SWAP LOG 1<SHIFT
   08DA -         \ 462d
   ILOG
   +
   LOG 1SHIFT> ILOG ;
```

Kompletter Sourcecode auf:

`http://www.embeddedFORTH.de`

Für Portierung auf andere Forth–Varianten vergleiche dort im GP32–Manual die Seiten 4.2 und 4.3.

Wie macht man einen Word-Scrambler in Forth?

Ulrich Hoffmann, Michael Kalus

Worte verwürfeln — scramble words

Ausgangspunkt war eine zufällige Begegnung im Jahr der Mathemathik. In einer Zeitschrift, deren Titel mir, Michael, inzwischen wieder entfallen ist, blieb ich bei dem Link auf Glynns Scrambler hängen. Und begab mich auf die Reise ins Land des Mischens und des Zufalls: `http://www.lerfjhax.com/scrambler`

So ein Word–Scrambler mischt die Buchstaben in den Worten. Das psychologisch Erstaunliche daran ist, dass wir danach den Text immer noch lesen können! Und besonders dann, wenn wir den Text schon kennen, lesen wir ihn ziemlich flüssig auch in gescrambelter Form.

Es scheint, unser Gehirn kann eigentlich gar nicht lesen, sondern erkennt dabei ganzheitliche Symbole, die wir bereits gelernt haben. Jedes Wort ist so ein Symbol. Wir sehen es als Ganzes und buchstabieren eigentlich nur Worte die uns noch unbekannt waren. Schreiben wir selbst etwas auf, sehen wir die Fehler in der Schreibweise zunächst auch gar nicht, weil wir zu gut wissen, was da stehen sollte.

Es sceinhct, uensr Ghrein knan eclgetiinh gar nchit lseen, senrdon ekrnnet deabi gheinzahlitce Soybmle, die wir beirets geelnrt heabn. Jdees Wrot ist so ein Smyobl. Wir sheen es als Gneazs und bithaberucsen eitciegnlh nur Wtore, die uns ncoh ubannknet waern. Sceebrhin wir sslbet ewtas auf, sheen wir die Fheler in der Swerhebiscie zhäcusnt acuh gar nchit, wiel wir zu gut wsiesn, was da seethn slltoe.

scramble <word1> <word2> ... <wordN>

In jenem Land der Algorithmen traf ich Ulli. Unsere simple Version des scramble erlaubt uns nun zu experimentieren - und besser zu verstehen, was Will Glynn wohl gemeint haben könnte, als er schrieb: „Ich habe keine Ahnung, wie oder warum jede Woche tausende Leute hier her kommen, um Worte zu scrambeln. Als Reaktion darauf habe ich mich aber entschlossen, den word scrambler immer noch mal zu machen. Dieses ist inzwischen die dritte oder vierte Überarbeitung...“

Unser fundamentales Scramble unterscheidet gar nicht, welche Zeichen getauscht werden sollen. Dem menschlichen Leser entgegen kommen wir lediglich durch die Vorgabe, dass die ersten und letzten Zeichen eines Wortes vom Tausch ausgenommen werden. Sonst macht es uns ja gar keinen Spaß mehr, unser Hirn müht sich vergebens, noch was zu erkennen. Aber mit dieser kleinen Hilfe geht es schon recht gut! Und nun kann man auch studieren, was alles so passiert. Versuche doch mal etwas wie:

```
scramble (.) (.o) ABCE 12345 *bert.bert*
```

also Mixturen der Art, dass auch Satzzeichen und Ziffern darin vorkommen.

Dann wird schnell klar, dass es nicht immer Sinn machen kann, alles auf die gleiche Weise zu scrambeln. Will man nämlich erreichen, dass das Ergebnis für einen Menschen verständlich bleiben soll, dürften Zahlen, Ziffern, und andere Zeichen als die Buchstaben nicht mitgemischt werden.

scramble-word2

Natürlich ist es möglich, dass eine randomisierte Permutation auch mal dazu führt, dass die inneren beiden Zeichen eines Wortes mit nur vier Buchstaben gar nicht getauscht werden. Wir wollen aber nicht, dass so ein Wort gleich bleibt, also tauschen wir in solch einem Fall die inneren beiden Zeichen immer aus.

Aber auch bei Worten mit fünf Zeichen gibt es noch optische Probleme. Scramble doch mal „lesen“— es bleibt oft gleich. Zwar wurden die „e“vertauscht, aber das ist im Ergebnis optisch nicht das, was wir wollten. So bleibt es uns, darüber nachzudenken, wie an dieser Stelle anders verfahren werden könnte, damit die resultierende Sequenz nicht zufällig gleich der Ausgangssequenz ist.

Bei Worten mit mehr als fünf Zeichen, also vier und mehr in der Mitte, wird es dann schon sehr unwahrscheinlich, dass beim randomisierten Tauschen noch was gleich bleibt. Ab da kann also einfach drauflosrandomisert werden, finden wir. Statistisch können wir das nicht belegen, es ist mehr so ein Gefühl aus den ersten Beobachtungen. Und auch das Problem der Ziffern und Satzzeichen besteht noch.

Beschäftigt man sich also mit algorithmisch hergestellter Unordnung, wie so einer Permutation eines einfachen Arrays, findet man bei Knuth immer was Passendes. Den Algorithmus für unseren Fall bezog er aus Arbeiten von Richard Durstfeld. Und damit sind wir schon im Bereich der Randomisierung, des Mischens und Chiffrierens mittendrin. Wir finden solche Spielereien mit einfachen Grundlagen immer sehr spannend. Warum? Nun, weil wir aus Erfahrung inzwischen weiß, das Jacques Hadamard, ein französischer Mathematiker (8. Dezember 1865 – 17. Oktober 1963) (!), mit seiner Beschreibung des kreativen Prozesses ganz richtig lag [3].

Im scharfen Gegensatz zu anderen Autoren seiner Zeit, die auf Sprache setzten und Kognition - und das zieht sich noch immer durch unsere Schulen — beschrieb er sein eigenes mathematisches Denken als weitgehend wortlos, oft begleitet von mentalen Bildern, die eine vollständige Lösung für ein Problem darstellten. Daraufhin begutachtete er rund 100 der führenden Wissenschaftler seiner Zeit (um 1900), ließ sich schildern, wie sie arbeiteten. Viele ihrer Antworten spiegelten seine Auffassung

wider, einige berichteten von mathemathischen Konzepten als Farbzusammensetzungen.

Um solche Vorstellungen zu entwicklen, ist es absolut notwendig, herumzuspielen, die Vorgänge sinnlich werden zu lassen, bildlich vor sich zu haben sozusagen. Und wir sind immer begeistert davon, wenn so etwas mit Forth in oft simpler Weise gelingt.

Viel Spaß beim Spielen.

Quellen

[1] Algorithm 235 *Random Permutation* Richard Durstfeld, 2.Jan.1964, San Diego, Calif. Algorithm for generating a random permutation of a finite set. Algorithmus um endliche Sets zufällig zu mischen. Artikel aus den Communications of the ACM volume 7, issue 7

[2] Synonyme: Fisher–Yates–Shuffle, Knuth–Shuffle `http://en.wikipedia.org/wiki/Knuth_shuffle` und: Das „mische ein Kartenspiel"-Problem. `http://de.wikipedia.org/wiki/Mischen`

[3] *The Mathematician's Mind: The Psychology of Invention in the Mathematical Field* Jacques Hadamard `http://press.princeton.edu/titles/5896.htm` `http://en.wikipedia.org/wiki/Jacques_Hadamard#On_creativity`

[4] Schicker Word–Scrambler im Browser: `http://www.lerfjhax.com/scrambler`

Listing

```
1  \ word scrambler based on random shuffle
2
3  vocabulary scrambler
4  scrambler definitions
5
6  variable seed
7
8  base @ hex 10450405 constant generator
9  base !
10
11 : rnd  ( -- n )  seed @ generator UM* DROP 1+ DUP seed ! ;
12
13 : random ( n -- 0..n-1 )  rnd UM* NIP ;
14
15 : init ( -- )
16     TIME&DATE  12 * + 31 * + 24 * + 60 * + 60 * +  seed !  ;  init
17
18 : c>< ( c-addr1 c-addr2 -- ) \ character exchange
19     2dup c@ ( c-addr1 c-addr2 c-addr-1 c2 )
20     swap c@ ( c-addr1 c-addr2 c2 c1 )
21     rot c!  ( c-addr1 c2 )
22     swap c! ;
23
24 : cshuffle ( c-addr n -- ) \ shuffle Durstenfeld/Knuth
25    BEGIN ?dup WHILE ( c-addr i )
26       2dup 1- chars + >r
27       2dup random chars + r>  c><
28       1-
29    REPEAT drop ;
30
31 : scramble-word2 ( c-addr len -- ) \ some case handling included.
32     dup 4 < IF 2drop exit THEN
33     dup 4 = IF over char+ dup char+ c><  2drop exit THEN
34     2 - swap char+ swap cshuffle ;
35
36 : scramble ( <word> -- )
37     cr
38     BEGIN
39        bl word count
40        dup
41     WHILE ( c-addr len )
```

```
42          2dup scramble-word2 type space
43        REPEAT
44        2drop ;
45
46    words
47    scramble sah ein knab ein röslein stehen
48    cr cr
49
50    \ study random behavior, execute serveral times.
51      \ : xxx  ( -- )  0 10 do cr  i .  i random .   -1 +loop ;
52      \ scramble (.) (.0) (.:0) (.:=0) (.:=#0) (.:=#*0)
```

Lebenszeichen

Bericht aus der FIG Silicon Valley: *Henry Vinerts*

Januar 2008

Dear Fred,

ich möchte dir gern berichten, dass ich gestern ausnahmsweise wieder mal einen ganzen Tag auf dem SVFIG–Treffen zugebracht habe. Und was mich gefreut hat: Mein VD–Exemplar 3+4/2007 ging durch die Hände praktisch aller 18 oder mehr Anwesenden. Und mehr als das: Zwei Teilnehmer baten mich, ihnen Kopien von Artikeln zu senden, auf die sie aufmerksam wurden. Anscheinend haben beide hinreichende Kenntnisse der deutschen Sprache. Ulrich Hoffmanns *Ist Forth Turing–vollständig?* geht an Masa Kasahara und Rafael Delianos *Das FOCUS–Zahlensystem* wurde von Dick Delp verlangt. Dick bat mich auch um deine Übersetzung meines Artikels über Ramanujans Taxi–Nummer.

Das Tagungsprogramm bestand hauptsächlich aus drei längeren Vorträgen, zwei davon von verhältnismäßig neuen Teilnehmern. Natürlich hielten wir uns an unsere Tradition und ließen Ting den Vormittag mit Einzelheiten zu einem seiner laufenden Projekte ausfüllen: Das Schreiben mehrspuriger MIDI–Dateien auf dem PC. Am Nachmittag präsentierte Brad Nelson seine Neuimplementation von Chuck Moores colorForth, die Windows– and Linux–System–Aufrufe bewältigen kann. Brad nennt es Rainbow Forth und hat seine Quellen nach Sourceforge.net gelegt. Der dritte kreative Vortragende war Steven Nichols. Steven hat einen Makro–Cross–Assembler entwickelt, ML1, der unter DOS läuft und es dem Anwender erlaubt, seine eigene benutzerdefinierte Sprache zu bauen, mit einem Schwierigkeitsgrad und einer Verarbeitungsgeschwindigkeit irgendwo zwischen BASIC und Assembler.

John Rible führte den neuen USB–betriebenen IntellaSys–S24chip vor, über den in den Vorträgen des nächsten SVFIG gesprochen werden soll. Der Chip ist so klein, dass er in ein Katzenohr passen würde. Dennoch wurde er schon mit externen Tastaturen und anderen Peripheriegeräten ausprobiert und, wie ich annehme, sogar mit einer Maus — zur Freude der Katze.

Das war's für heute, Fred, sicher genug, um dir zu zeigen, dass wir hier immer noch eine gehörige Anzahl von Leuten haben, die sich unter der Flagge von Forth versammeln, auch wenn es vielleicht nicht mehr ein und dieselbe Sprache ist, die sie verbindet.

With kind regards,
Henry

Übersetzt von Fred Behringer

Lieber Henry,

vielen Dank für deine Zeilen. Was Ramanujans Taxi betrifft, auf das du uns in deinem VD–Artikel aufmerksam machtest, so kann es vielleicht nicht schaden, wenn du den von dir oben erwähnten Dick Delp darauf aufmerksam machst, dass Marcel Hendrix die Gelegenheit nutzte, in de.comp.lang.forth und anderswo mit Anton Ertl eine sehr interessante und lange Diskussion darüber zu führen, mit welchen Programmänderungen es möglich wird, Geschwindigkeitshöchstleistungen herauszuholen. Irgendwie habe ich in Erinnerung, dass er in diesem Zusammenhang auch sein Transputer–Forth–System (für den und mit dem T800 von INMOS) ins Spiel bringt, ich kann die genaue Stelle aber nicht mehr finden. Für deine Mitarbeit an der Gestaltung der Vierten Dimension unseren besten Dank.

Herzlichen Gruß
Fred

Aus der Kiste der nützlichen Forth–Worte: Cursor toggeln

Fred Behringer

Der vorliegende Artikel kann als eine Übung im Sinne des leider nicht mehr auf unserer Welt weilenden Wissenschaftlers und Forth–Freundes Julian Noble betrachtet werden: Julian Noble rief in einer dreiteiligen Artikelfolge (Übersetzung in den VD–Heften 2/2002 bis 4/2002) dazu auf, mehr Assembler in Forth zu verwenden. Das Einbinden von (assemblierter) Maschinensprache geht in Forth wesentlich leichter als in anderen Hochsprachen. Noble: „Es wäre besser, Forth den ungläubig Außenstehenden über den Assembler schmackhaft zu machen."

Ich habe die im Vorliegenden aufgeführten Forth–Worte in Turbo–Forth unter DOS 6.2 auf einer AMD–K6–2/500–Anlage mit SIS–6326–Karte entwickelt und getestet.

Beim Programmieren von Spielen (Tetris, Life, Sudoku, Schiebespiel, Türme von Hanoi, allgemeine Menüführung) stört oft der *stehen gebliebene* blinkende Cursor. In den Forth–Sprachstandards gibt es keine Worte zur Manipulation des Cursors. In Turbo–Forth findet man das Wort (frame) zur Veränderung der Form des Cursors. Aber für das schlichte Aus–, Ein– oder Umschalten des Cursors (durchaus elementare Wünsche) ist auch in Turbo–Forth kein Wort vorgesehen. Dem soll (Forth macht's leicht möglich) mit den vier Forth–Worten dieses Artikels abgeholfen werden.

Natürlich enthält das Hingeschriebene nichts Neues. Es steht alles in den Unterlagen. Aber nicht jeder hat alle (nur gelegentlich) benötigten Unterlagen griffbereit auf seinem Schreibtisch, der Einsteiger schon gar nicht.

Ich gehe von einem PC–kompatiblen System aus, dessen Grafik–Karte sich (für Forth unter DOS) in den VGA–Modus schaltet oder schalten lässt.

Eine Besonderheit der VGA–Karte ist die so genannte *Cursor–Emulation*: Bei der CGA–Karte ist das Cursor–Rasterfeld kleiner als bei der EGA– oder der VGA–Karte. Der *normale* Unterstrich–Cursor der CGA–Karte würde bei der VGA–Karte etwa in der Mitte des dortigen Rasterfeldes zu liegen kommen. Aus Gründen der Programm–Kompatibilität hat man deshalb seinerzeit (schon bei der EGA–, aber dann eben auch bei der VGA–Karte) die *Cursor–Emulation* eingeführt, welche insbesondere dafür sorgt, dass der CGA–Unterstrich–Cursor auch bei VGA–Betrieb als Unterstrich wiedergegeben wird. Daneben noch einige weitere, mehr oder weniger logische Anpassungen, die aber für die Cursor–Formgebung genügend viele Variationsmöglichkeiten (Blockcursor etc.) lassen. Im Übrigen lässt sich die Cursor–Emulation im VGA–Betrieb über die Funktion ah=12h, bl=34h des Interrupts 10h (BIOS–Bildschirm–Interrupt) abschalten (al=0/1: ein/aus).

Mir geht es nicht um die Feinheiten der Cursor–Gestaltung, sondern um das generelle Ein– oder Aus– oder Umschalten. Und das lässt sich im VGA–Betrieb über das Kontroll–Register (des Video–Controllers 6845) für die Anfangs–Rasterzeile erreichen. Dazu müssen (geht über das CPU–Register ch für den BIOS–Interrupt 10h, ah=1) im Kontroll–Register die Bits 5–6 (Zählung beginnt bei 0) bearbeitet werden (0/1: Cursor ein/aus). Bit 7 hat keine Wirkung. Ich lasse vom Programm alle drei Bits (5–7) auf 1 (Cursor aus) oder 0 (Cursor ein) setzen. Es scheint aber mit Bit 5 allein oder mit Bit 6 allein auch schon zu gehen (?)

Die Cursor–Emulations–Einrichtung (Interrupt 10h/12h/34h), die *normalerweise* eingeschaltet ist, rühre ich nicht an. Sie interessiert mich im vorliegenden Zusammenhang weniger und ich wollte die Sicht für das hier eigentlich zu Sagende nicht versperren. An sich würde es aber unter DOS einem Forth–System unserer Zeit, da wir CGA–Karten gar nicht mehr kennen, gut anstehen, den VGA–BIOS–Vorgabewert auf *keine Cursor–Emulation* zu ändern. Dann würde es nämlich genügen, die untere Rasterzeile kleiner als die obere zu wählen, um den Cursor auszuschalten. Dann käme man in den Vorschlägen des vorliegenden Artikels mit curs–shape allein schon durch und könnte sich die drei anderen Forth–Worte sparen. Über den besagten Interrupt 10h/12h/34h (zum Ausschalten der Emulation) habe ich in den Unterlagen unterschiedliche Aussagen gelesen. Ich hatte keine Veranlassung, die Dinge genauer nachzuprüfen.

An sich kommt man mit curs–shape in der hier vorgeschlagenen Fassung auch jetzt schon allein durch: In denjenigen Fällen (in allen?), in welchen kein Grund zur Cursor–Emulations–Umwandlung besteht (es geht ja bei der Emulation hauptsächlich nur um die Bewahrung des Unterstrichs als Cursor), scheint die Grundregel zu gelten, die da besagt, dass bei einer unteren Rasterzeile, die kleiner als die obere ist, der Cursor ausgeschaltet wird. Nachgeprüft habe ich das bei oben/unten = 0f/00 bis 00/00. Schwierig ist es nur, aus den Unterlagen heraus genau zu erfassen, in welchen Fällen (und wie) die Emulation eigentlich angewandt wird. Interessant ist der Sonderfall cl/ch=00/00. Da wird der Cursor wiederum nicht ausgeschaltet. Da ist aber die Nummer der unteren Rasterzeile auch nicht (streng) kleiner als die der oberen! Mit einem generellen Auf–0–setzen der unteren Rasterzeile käme man also zur Ausschaltung des Cursors nicht durch!

Listing auf der folgenden Seite.

Listing

```
1   hex
2
3   code curs-on ( -- )            \ Cursor einschalten
4       ds di mov                  \ Datensegment aufbewahren,
5       bx bx xor  bx ds mov       \ dann auf 0 setzen.
6       460 [bx] cx mov            \ Rasterzeilen: ch oben, cl unten
7       di ds mov                  \ Datensegment wiederherstellen
8       1f # ch and                \ Bits 5-7 der obersten Rasterzeile auf 0
9       1 # ah mov                 \ Interrupt-Funktion nach ah
10      10 int                     \ Bildschirm-Interrupt aufrufen
11      next end-code
12
13  code curs-off ( -- )           \ Cursor ausschalten
14      ds di mov                  \ mov-Operationen beziehen sich
15      bx bx xor  bx ds mov       \ auf das Datensegment ds
16      460 [bx] cx mov
17      di ds mov
18      e0 # ch or                 \ Bits 5-7 der obersten Rasterzeile auf 1
19      1 # ah mov  10 int
20      next end-code
21
22  code curs-togg ( -- )          \ Cursor umschalten
23      ds di mov
24      bx bx xor  bx ds mov
25      460 [bx] cx mov
26      di ds mov
27      e0 # ch xor                \ Bits 5-7 der obers. Rasterz. 0/1 -> 1/0
28      1 # ah mov  10 int
29      next end-code
30
31  code curs-shape ( anf end -- ) \ Cursor-Formgebung
32      ds di mov
33      bx bx xor  bx ds mov
34      460 [bx] cx mov
35      di ds mov
36      ax pop                     \ Unterste Cursor-Rasterzeile auf
37      1f # al and  al cl mov     \ (0..1f) beschränken und nach cl legen.
38      ax pop                     \ Oberste  Cursor-Rasterzeile auf
39      1f # al and                \ (0..1f) beschränken, dann bisherigen
40      e0 # ch and                \ Einschaltzustand (Bits 5-7) retten
41      al ch or                   \ und beides nach ch zusammenführen.
42      1 # ah mov  10 int
43      next end-code
44      \ Der so implementierte Cursor kann auch im ausgeschalteten
45      \ Zustand (im Verborgenen) neu geformt werden, ohne dass er
46      \ sich dadurch schon einschaltet. Beim anschließenden
47      \ Einschalten zeigt er dann seine neueingestellte Form.
```

Projekt Euler — Problem p146

Luca Masini [1]

Einleitung zum Projekt Euler

Die Projekt–Euler[2]–Internet–Seite bietet eine Reihe von mathematischen Programmierproblemen an.

Neben der Mathematik selbst, die hilft, elegante und effiziente Lösungsmethoden zu finden, sind für die vorgestellten Probleme ein Rechner und Programmierfähigkeiten wesentliche Zutaten, um überhaupt zu einer Lösung zu kommen.

Wer ein Interesse an der Mathematik hat, kann hier sehr interessante Probleme finden.

Die Probleme haben verschiedene Schwierigkeitsgrade, aber mit steigender Erfahrung wird man lernen, auch die schwierigen Probleme zu lösen.

Am Ende ist es auch eine gute Gelegenheit, um ein bisschen Erfahrung mit Forth zu sammeln!

Ein Beispiel: Problem 146

Als Beispiel nehme ich das Problem 146, hier zu finden: `http://projecteuler.net/index.php?section=problems&id=146` und mit folgendem Satz beschrieben:

> The smallest positive integer n for which the numbers n^2+1, n^2+3, n^2+7, n^2+9, n^2+13, and n^2+27 are consecutive primes is 10. The sum of all such integers n below 1 million is 1242490. What is the sum of all such integers n below 150 million?

Hier gilt es also, sechs Primzahlen zu erkennen, von denen die ersten vier Primzahlen sog. *Primzahlvierlinge* sein sollen.

Primzahlvierlinge sind vier Primzahlen der Form $(p, p+2, p+6, p+8)$. Wenn also p die kleinste Primzahl von Primzahlvierlingen ist, dann soll man jeden einzelnen Primzahlvierling so durch die natürliche Zahl n beschreiben können:

$$\begin{aligned} n^2+1 &= p \\ n^2+3 &= p+2 \\ n^2+7 &= p+6 \\ n^2+9 &= p+8 \end{aligned}$$

Eine kurze Suche im Internet findet auf der Seite `http://mathworld.wolfram.com/PrimeQuadruplet.html` eine Erklärung mit weiteren Eigenschaften von Primzahlvierlingen.

Insbesondere gilt folgendes:

> With the exception of (5, 7, 11, 13), a prime quadruple must be of the form (30 m + 11, 30 m + 13, 30 m + 17, 30 m + 19)

Aus

$$p = n^2 + 1 = (30m + 10) + 1$$

erhalten wir

$$n^2 = (30m + 10) = 10(3m + 1)$$

Deshalb ist n immer ein Vielfaches von 10, aber nie ein Vielfaches von 3.

Die Anzahl der Tests wird damit von 150 Millionen auf 10 Millionen verringert, aber es sind immer noch für jedes n teure Primzahltests zu machen.

Mit einigen mathematischen Schritten, die relativ lange und langweilig sind, ist es auch möglich zu zeigen, dass

$$(n \bmod 7 = 3) \; or \; (n \bmod 7 = 4)$$

ist. Diese Bedingung erlaubt, die Zahl der Tests weiter zu vermindern und wird im Code verwendet.

Algorithmus

Die äußere Schleife des Algorithmus erhöht den Index um 10. Außerdem kann sie alle durch drei teilbaren Zahlen weglassen, und n `mod` 7 muss gleich 3 oder 4 sein.

Die Indizes, die diese Bedingungen erfüllen, werden an `(main-ba)` übergeben, das eine Reihe von Tests über die Restwerte ausführt. Dann macht es einen Primzahltest mit `miller_rabin` und gibt schließlich TOS zurück, wenn der Parameter die Bedingungen des Problems erfüllt, sonst 0.

Dieser Wert wird verwendet, um die lokale Variable `sum` zu vergrößern, die am Ende der Schleife die Lösung des Problems enthalten wird.

```
: main { W: limit -- d }
    0. { D: sum }
    limit 1+ 10 do
        \ i must be divisible by 10,
        \ loop only through multiples of 10
        i 3 mod 0<> if
            \ i can't be divisible by 3
            i 4 + 7 mod 1 <= if
                \ i % 7 must be either 3 or 4
                i (main-ba) s>d sum d+  to sum
            then
        then
    10 +loop
    ." sum=" sum ud. cr
;
```

Der Miller–Rabin–Primzahltest

Der Primzahltest ist durch das Wort `miller_rabin` (siehe `http://en.wikipedia.org/wiki/Miller-Rabin_primality_test`) realisiert.

Es handelt sich um einen probabilistischen Primzahltest. Erhält er eine natürliche Zahl n als Eingabe, so gibt er aus, ob diese Zahl eine Primzahl ist oder nicht. Gibt er

[1] luca.masini@member.fsf.org

[2] `http://projecteuler.net/index.php?section=about`

dabei *FALSE (n ist keine Primzahl)* aus, so ist das immer richtig. Gibt er allerdings *TRUE (n ist wahrscheinlich eine Primzahl)* aus, so ist dies mit geringer Wahrscheinlichkeit falsch.

Für die Implementierung dieses Tests braucht man einen Zufallszahlengenerator. Hier wurde einer der *Scientific Forth Library*[3] benutzt.

Logik– und Shift–Wörter für Doppelwortoperanden

Der Zufallszahlengenerator `ran4` und der Primzahltest `miller_rabin` brauchen logische Worte, die mit Doppelzellen arbeiten. Diese Worte wurden wie folgt implementiert:

```
: dand    ( d d -- d    ) rot and >r and r>       ;
: dinvert (   d -- d    ) swap invert swap invert ;
: dlshift ( d u -- d    ) 0 ?do d2* loop          ;
: dor     ( d d -- d    ) rot or  >r or  r>       ;
: dnot    (   d -- 0.|1.) d0= abs s>d             ;
: drshift ( d u -- d    ) 0 ?do d2/ loop dabs     ;
: dxor    ( d d -- d    ) rot xor >r xor r>       ;
```

Arithmetische und modular–arithmetische Wörter für Doppelwortoperanden

Der Teil, der den größten Aufwand verursachte, war die Implementierung der modulo–arithmetischen Worte für Doppelzellen. Folgende Worte sind im Quellcode zu finden:

```
d**mod  { D: base D: power D: m -- d  }
\ Modular exponentiation
d*mod   { D: a    D: b     D: m -- d  }
\ Modular multiplication
d+mod   { D: a    D: b     D: m -- d  }
\ Modular addition
d*      ( d1 d2                 -- d  )
\ Multiplication for double cells
d/      { D: u    D: v          -- d  }
\ Division for double cells
```

Einige Worte wurden auch aus *Long Divisors and Short Fractions (Nathaniel Grossman*[4]*)* übernommen.

Versuchsweise wurde das Wort `d/` auch mit *float* implementiert, und war dann auch deutlich schneller, aber das hatte eine nicht korrekte Lösung zur Folge:

```
Found n=10
Found n=315410
Found n=927070
Found n=2525870
Found n=8146100
Found n=16755190
Found n=39313460
Found n=97387280
Found n=119571820
Found n=121288430
Found n=130116970
Found n=139985660
Found n=144774340    <-- WRONG
sum=821107610
```

Zusammenfassung

Der präsentierte Code löste das Problem in 6 Minuten und einer Sekunde auf meinem Laptop (Intel Core 2 Duo Prozessor T7200, Merom, 2.00 GHz, 667 MHz FSB, 4). Ich bin immer noch nicht glücklich über die Performance, und denke, es sollte möglich sein, unter einer Minute bleiben zu können.

Source Code

```
1   #! /usr/bin/gforth
2
3   \ ---- logic and shift words for double cell operands -------------------------
4
5   : dand    ( d d -- d     ) rot and >r and r>       ;
6   : dinvert (   d -- d     ) swap invert swap invert ;
7   : dlshift ( d u -- d     ) 0 ?do d2* loop          ;
8   : dor     ( d d -- d     ) rot or  >r or  r>       ;
9   : dnot    (   d -- 0.|1. ) d0= abs s>d             ;
10  : drshift ( d u -- d     ) 0 ?do d2/ loop dabs     ;
11  : dxor    ( d d -- d     ) rot xor >r xor r>       ;
12
13
14  \ ---- ran4 : a random number generator --------------------------------------
15
16  : (FuncG) ( d dc1 dc2 -- d )
17      2>r dxor 2dup um* 2swap dup um* dinvert rot dup um* d+ swap 2r> dxor d+
18  ;
19
20  : (PseudoDes) ( d d -- d d )
21      2swap 2over  $BAA96887E34C383B. $4B0F3B583D02B5F8. (FuncG) dxor
22      2swap 2over  $1E17D32C39F74033. $E874F0C39226BF1A. (FuncG) dxor
23      2swap 2over  $03BCDC3C60B43DA7. $6955C5A61D38CD47. (FuncG) dxor
24      2swap 2over  $0F33D1B265E9215B. $55A7CA46F358B432. (FuncG) dxor
```

[3] `http://www.taygeta.com/fsl/sciforth.html`

[4] http://www.forth.org/fd/FD-V06N3.pdf

```
25  ;
26
27  2variable Counter
28  2variable Sequence#
29
30  : start-sequence  ( dcounter dseq# -- )  Sequence# 2!  Counter 2! ;
31
32  : ran4 ( -- d )
33     Sequence# 2@  Counter 2@  (PseudoDes)
34     2swap 2drop
35     Counter 2@ 1. d+  Counter 2!
36  ;
37
38
39  \ ---- arithmetic words for unsigned double cell operands ---------------------
40
41  : d*    ( d d -- d )    3 pick * >r  tuck * >r  um*  r> +  r> +    ;
42
43  : t*    ( ud un -- ut )    dup rot um* 2>r um* 0 2r> d+ ;
44
45  : t/    ( ut un -- ud )
46       >r   r@ um/mod swap
47     rot 0 r@ um/mod swap
48     rot   r> um/mod swap drop
49     0 2swap swap d+
50  ;
51
52  : u*/    ( ud un un -- ud )    >r t* r> t/ ;
53
54  \ initialize SUPERBASE (normally $100000000. on 32 bits machine, with fallback value of $10000.)
55
56  s" max-u" environment? 0= [if] $10000. [then] 0 1. d+ 2constant SUPERBASE
57
58  : d/ { D: u D: v -- ud_quot }
59     v 0. d= if -10 throw then                                    \ throw for division by zero
60     v u du> if 0. exit then                                      \ if v is bigger then 0.
61     v u d=  if 1. exit then                                      \ if v is equal then 1.
62     v 0= if >r u 1 r> u*/ exit then                              \ use mixed precision word
63     drop
64     v                                                 { v1 v0 }  \ v  = v0 * b + v1
65     v0 -1 = if 1 else SUPERBASE 1 v0 1+ u*/ drop then  { d     }  \ d  = b/(v0+1)
66     v d 1 u*/                                         { w1 w0 }  \ w  = d * v = w0 * b + w1
67     u over 0 w1 w0 u*/ d- d w0 u*/ nip 0
68  ;
69
70  : dmod    { D: d1 D: d2 -- d }    d1 2dup d2 d/ d2 d* d-    ;
71
72  : dumin    ( d1 d2 -- d )    2over 2over du> if 2swap then 2drop    ;    \ d is min(d1,d2)
73
74  \ initialize UMAX (normally $ffffffffffffffff. on 32 bits machine, with fallback value of $ffffffff.)
75
76  s" max-ud" environment? 0= [if] $ffffffff. [then] 2constant UMAX
77
78  : d+mod { D: a D: b D: m -- d }     \ addition modulo m
79     a m dmod to a            \ normalize a
80     b m dmod to b            \ normalize b
81     a UMAX b d- d<= if       \ no overflow..
82         a b d+ m dmod exit   \ ..built-in computation
83     then
84     \ ---- go with the algorithm    ;-)
85     a m a d- dumin { D: aA }
86     b m b d- dumin { D: bB }
87     b bB d=                                                  ( -- f ) \ leave a flag on stack
88     a aA d= if
89         bB aA du> if
90             if aA bB d+ m dmod else m bB aA d- d- then exit           ( f -- ) \ ..consume the flag
```

```
91          else
92              >r aA bB r> if  d+ else d- then m dmod exit                ( f -- ) \ ..consume the flag
93          then
94      else
95          if aA bB du> if m aA bB else m m bB aA d- then d- d- exit then ( f -- ) \ ..consume the flag
96      then
97      m aA bB d+ m dmod d-
98  ;
99
100 : d*mod { D: a D: b D: m -- d }     \ multiplication modulo m
101     a m dmod to a            \ normalize a
102     b m dmod to b            \ normalize b
103     a 1. d= if b exit then
104     b 1. d= if a exit then
105     a m a d- dumin { D: aA }
106     b m b d- dumin { D: bB }
107     aA d0= bB d0= or if 0. exit then
108     aA 1. d= if m b d- exit then
109     bB 1. d= if m a d- exit then
110     aA a d= bB b d= and aA a d<> bB b d<> and or  { pos }  \ pos is True if positive, False otherwise
111     aA UMAX bB d/ du<= if
112         aA bB d* m dmod pos 0= if m 2swap d- m dmod then exit
113     then
114     aA d2/          { D: a0 }
115     aA a0 d-        { D: a1 }
116     bB d2/          { D: b0 }
117     bB b0 d-        { D: b1 }
118     a1 b1 m recurse { D: p4 }
119     0. 0. 0.        { D: p1 D: p2 D: p3 }
120     a0 a1 d= b0 b1 d= and if
121         p4 to p1
122         p4 to p2
123         p4 to p3
124     else
125         a0 a1 d= if
126             p4 m a1 d- m dmod m d+mod 2dup   to p3
127                                             to p1
128             p4                              to p2
129         else
130             p4 m b1 d- m dmod m d+mod        to p2
131             b0 b1 d= if
132                 p2                          to p1
133                 p4                          to p3
134             else
135                 p4 m a1 d- m dmod m b1 d- m dmod 1. m d+mod m d+mod m d+mod    to p1
136                 p4 m a1 d- m dmod m d+mod                                      to p3
137             then
138         then
139     then
140     p1 p2 p3 p4 m d+mod m d+mod m d+mod pos 0= if m 2swap d- m dmod then
141 ;
142
143 : d**mod { D: base D: power D: m -- d }    \ exponentiation modulo m
144     1. { D: res }
145     begin power 0. du> while
146         1. power dand drop if              \ if power is odd
147             res base m d*mod    to res
148         then
149         base base m d*mod       to base
150         power 1 drshift         to power
151     repeat
152     res
153 ;
154
155 : miller_rabin { D: n W: rounds -- f }
156     n   1. du<=       if false exit then
```

```
157       n  19. du<= if
158           n drop                                \ to use single cell in the following tests
159           dup  2 = if drop true  exit then
160           dup  3 = if drop true  exit then
161           dup  5 = if drop true  exit then
162           dup  7 = if drop true  exit then
163           dup 11 = if drop true  exit then
164           dup 13 = if drop true  exit then
165           dup 17 = if drop true  exit then
166               19 = if      true  exit then   \ do not dup in the last test
167           false exit
168       else
169           n   2. dmod 0. d= if false exit then
170           n   3. dmod 0. d= if false exit then
171           n   5. dmod 0. d= if false exit then
172           n   7. dmod 0. d= if false exit then
173           n  11. dmod 0. d= if false exit then
174           n  13. dmod 0. d= if false exit then
175           n  17. dmod 0. d= if false exit then
176           n  19. dmod 0. d= if false exit then
177       then
178       n 1. d-                                  { D: d }
179       0                                        { W: s }
180       begin d 1. dand 0. d= while
181           d 1 drshift                       to d
182           s 1+                              to s
183       repeat
184       0. 0. 0. 0                               { D: a D: tst D: Rbig W: Rsmall }
185       begin rounds 1- dup to rounds 0> while
186           ran4 n 1. d- dmod 1. d+           to a
187           a d n d**mod 1. du<> if
188               1.                            to Rbig
189               0                             to Rsmall
190               begin Rsmall s u< while
191                   a Rbig d d* n d**mod      to tst    \ tst = powmod( a, R * d, n )
192                   n 1. d- tst du= if
193                       s                     to Rsmall \ cause a break from the loop
194                   else
195                       Rbig Rbig d+          to Rbig   \ double Rbig
196                       Rsmall 1+             to Rsmall
197                   then
198               repeat
199               n 1. d- tst du<> if
200                   false exit                          \ return false (composite number)
201               then
202           then
203       repeat
204       true                                            \ return true (maybe prime)
205   ;
206
207   \ ---- main and its internal words ----------------------------------------
208
209   : (main-inc-p)    ( u  --  u )    dup 2 + 3 mod 0= if 4 else 2 then +    ;
210
211   : (main-ba)  { W: n -- n' }
212       11               { W: p  }
213       0                { W: nd }
214       n s>d 2dup d*    { D: n2 }
215       begin
216           n2 p s>d dmod d>s            to nd
217           nd  1+  p mod 0= if 0 exit then
218           nd  3 + p mod 0= if 0 exit then
219           nd  7 + p mod 0= if 0 exit then
220           nd  9 + p mod 0= if 0 exit then
221           nd 13 + p mod 0= if 0 exit then
222           nd 27 + p mod 0= if 0 exit then
```

```
223          p (main-inc-p)              to p          \ inc. p by 2 or by 4
224          p n 1+ > if
225              n2 15. d+ 10 miller_rabin if 0 exit then
226              n2 19. d+ 10 miller_rabin if 0 exit then
227              n2 21. d+ 10 miller_rabin if 0 exit then
228              n2 25. d+ 10 miller_rabin if 0 exit then
229              ." Found n=" n . cr
230              n exit
231          then
232      again
233  ;
234
235  : main { W: limit -- d }
236      0. { D: sum }
237      limit 1+ 10 do                  \ i must be divisible by 10, loop only through multiples of 10
238          i 3 mod 0<> if              \ i can't be divisible by 3
239              i 4 + 7 mod 1 <= if     \ i % 7 must be either 3 or 4
240                  i (main-ba) s>d sum d+        to sum
241              then
242          then
243      10 +loop
244      ." sum=" sum ud. cr
245  ;
246
247  150000000 main
248  bye
```

Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 66, Februar 2008

Lezingen en projecten voor 2008 — Willem Ouwerkerk

Willem stellt einen Plan für beabsichtigte Vorträge (in den kommenden Zusammenkünften der HCC–Forthgebruikersgroep?) auf, von denen er hofft, dass sie auch im Vijgeblaadje ihren Niederschlag finden werden. Im Einzelnen soll besprochen werden: ATtiny45 und ByteForth (Jan van Kleef) – Bootloader in AVR–ByteForth (Ernst Kouwe) – FAT32 (Marc Hartjes) – altes DOS–Forth in virtueller Maschine (Dick Willemse) – ciForth–Fragen (Albert van der Horst) – des Autors Forth (Ben Koehorst) – Forth von der Pike auf (Ron Minke) – Strings in CHForth im DOS–Emulator (Coos Haak) – Micro Notebook mit Renesas CPU unter ciForth (Frans van der Markt) – F51–Projekt (Willem Ouwerkerk) – Zündmonitor für einen Viertaktmotor (Joergh Haas) – PIF, Programmieren In Forth (Albert Nijhof) – Forth–Ankopplung an grafische TCL/TK–Routinen (Joost van Ballegooijen).

Wanneer is het Pasen? — Frans van der Markt

Wann ist Ostern? Am ersten Sonntag nach dem ersten Vollmond im Frühjahr! Diesjahr (2008) ist es mit dem 23.März der frühestmögliche Termin – und der M16C berechnet diesen Tag unter ciForth in einem kleinen Programm des Autors tatsächlich genauestens (also kein Pentiumbug und kein Milleniumproblem!). Frans hat das Programm aus Kermits easter2.htm übertragen.

Renesas' M16C leert Forth (vervolg) — Albert van der Horst

Albert setzt seinen Bericht (wir haben ihn im vorigen VD–Heft ins Deutsche übertragen) über sein ciForth auf dem M16C–Prozessor von Renesas fort. Diesmal geht es um die Bibliothek und um das Tingel–Programm als Turnkey.

Forth und Skriptsprachen

Reinhold Straub

Wo läuft Forth? Traditionell gibt es zwei Antworten auf diese Frage: entweder läuft es „nativ auf der Hardware“ oder es läuft unter einem Betriebssystem. Hier soll eine dritte Antwort gegeben werden: Forth könnte auch auf eine Skriptsprache wie Python, Ruby oder Scheme portiert werden.

Theoretisch wäre ein Forth, das unter einer Skriptsprache statt unter einem Betriebssystem läuft, nichts Neues. So wie ein Forth sich z. B. auf die Ressourcen von Linux stützen kann, ebenso könnte es sich auf die Ressourcen von, sagen wir, Ruby stützen. Nur praktisch wurde es anscheinend noch nicht versucht.

Das ist erstaunlich, denn die Kombination von Forth mit einer Skriptsprache hat eine ganze Reihe von Vorteilen im Vergleich zur Kombination mit einem Betriebssystem.

Erstens sind Skriptsprachen nicht nur Funktionsbibliotheken, die den Zugang zu einem Dateisystem oder einem Grafiksystem verwalten, sondern komplette Programmiersprachen, die sich als solche von Forth aus nutzen ließen. Nun ist Forth bekanntlich selbst bereits ein Alleskönner. Dennoch enthalten Skriptsprachen regelmäßig hochabstrakte, leicht benutzbare Konstrukte, die schon vorhanden sind und nicht erst irgendwie gefunden, eingerichtet, geladen oder gar geschrieben werden müssen.

Ferner ist Skriptsprachen–Code, jedenfalls in den genannten Sprachen, plattformübergreifend. Die Mühsal, eine API für mehrere Betriebssysteme zu schreiben, und darunter insbesondere eine für das schwierige Windows, haben die tapferen Skriptsprachen–Entwickler auf sich genommen. Davon können auch die Forth–Leute profitieren.

Schließlich haben die Skriptsprachen große Programmbibliotheken, im Unterschied zu Forth — was in Forth–Kreisen häufig beklagt wird. Dieser Mangel wäre behoben. Außerdem würdem die Skriptsprachen eine GUI in die Beziehung einbringen; auch das fehlt den meisten Forths.

Die Stärken und Schwächen von Forth und die Skriptsprachen scheinen komplementär zu sein: in Forth lässt sich effizienter *Low–Level*–Code schreiben, man kommt an die Bits und Bytes direkt heran. Außerdem ist Forth flexibel; benötigt man eine neuartige Datenstruktur, dann ist man mit dem feinmechanischen Forth besser dran als mit den groben Bausteinen der Skriptsprachen. Die Forth–Kultur ist auch mehr daran orientiert, je eine einzelne Lösung für ein gegebenes Problem zu finden

```
int *dataspace, *systemvariable;
int *SP; int *RP;
int TOS;

static int
Forth_Cmd(ClientData cdata, Tcl_Interp *interp, int objc, Tcl_Obj * CONST objv[])
{
    int resultat;
    Tcl_Obj *eingabe = NULL;
    if (objc > 1) {
        int zahl;
        int i;
        for (i = 1; i < objc; i++) {
            resultat = Tcl_GetIntFromObj(interp, objv[i], &zahl);
/* (1) */           if (resultat == TCL_OK) {push(zahl);}
            else {
/* (2) */           eingabe = Tcl_ConcatObj(objc-i, objv+i);
                break;}
        }
        /* Falls nur eine Folge von Zahlen eingegeben wurde, sind wir schon fertig: */
        if (eingabe == NULL) {return TCL_OK;}
        /* Nun die Eingabe an eine Stelle im Wörterbuch schreiben, die von >in benutzt wird: */
        int len; systemvariable[1] = (int)Tcl_GetStringFromObj(eingabe,&len);
        systemvariable[2] = len;
    }
/* ... */
}
```

Listing 1: C–Implementierung des Befehls `Forth_Cmd`

```
int ende = 0;
while(1) {
        resultat = forthevaluate(TOS,(int)rSP,(int)rRP,(int)systemvariable,
                                 (int)dataspace,systemvariable[3]);
        switch(resultat) {
            case  0:    /* Eingabequelle erschöpft */
                    /* Der Speicher für eingabe kann freigegeben werden: */
                    Tcl_DecrRefCount(eingabe);
                    ende = 1; break;
            case  1:    /* Coroutine */
                    ende = 1; break;
            case  2:    /* ... */
        /* ... */
        }
        if (ende == 1) break;
    }
```

Listing 2: Die Endlosschleife von `Forth_Cmd`

anstelle einer allgemeinen Lösung für viele ähnliche Probleme. Somit würden sich die Abstraktionen der Skriptsprachen und die Konkretion von Forth sehr gut ergänzen.

Die API–Funktionen eines Betriebssystems sind allerdings sehr viel differenzierter als die Möglichkeiten der Skriptsprachen, so dass Forths wie Win32Forth für Windows oder Powermops für die Carbon–API des Mac OS durch ein Skriptsprachen–Forth nicht ganz ersetzbar sind. Forths hingegen, die nur in einer Konsole isoliert vom Rest des Betriebssystems laufen wie etwa gforth oder kforth würden auf jeden Fall sehr von der Einbindung in eine Skriptsprache profitieren, glaube ich.

Aber auch Powermops ist ein instruktives Beispiel: Es ist ein Forth mit einem optimierenden Compiler für PowerPC–Prozessoren, das die Carbon–API des Mac OS (Classic und X) unterstützt. Nun verkauft Apple keine PowerPC–Prozessoren mehr, und somit tun sich zwei Richtungen auf, in die Powermops weiterentwickelt werden könnte: man kann weiterhin auf Mac OS X setzen und müsste dann mittelfristig den Compiler auf Intel–Assembly umstellen. Oder man bleibt dem PowerPC treu, würde dann allerdings eine neue Betriebssystem–API benötigen. In dieser Situation ergibt es Sinn, Powermops für PPC auf eine Skriptsprache zu portieren und könnte damit *cross platform* auf dem PPC sowohl unter Mac OS X als auch unter Linux arbeiten.

Forth und Skriptsprachen haben zudem Gemeinsamkeiten, durch die es leicht fällt, sie ineinander zu integrieren und mit gemischtem Quellcode zu arbeiten. Vor allem sind beide interaktiv und eignen sich für *rapid prototyping*. Forth ist eine imperative Sprache und passt als solche zu Sprachen, die ebenfalls solche Konstrukte aufweisen.

Nun gibt es verschiedene Möglichkeiten, zwei Sprachen zu kombinieren. Die einfachste besteht wohl darin, beide in ihrer jeweiligen Umgebung laufen zu lassen und die Möglichkeit der Kommunikation zwischen Prozessen zu benutzen, wie etwa Applescript oder Unixröhren (*pipes*). Das geht, ist jedoch etwas teuer, so dass man sparsam damit umgehen würde und die Arbeit nur grobkörnig auf die beteiligten Sprachen aufteilen würde. Um die jeweiligen Vorteile optimal zu nutzen, wäre eine wesentlich engere und vielseitigere Verzahnung vermutlich sinnvoller.

Zweitens könnte man Skriptsprachen–Code von Forth aus aufrufen; das wäre so ähnlich wie die Benutzung einer Betriebssystem–API von Forth aus. Drittens könnte man die Skriptsprache um einen Forth–Evaluate–Befehl erweitern, so dass es möglich wird, Forth–Code von der Skriptsprache her auszuführen. Viertens könnte man das Forth und die Skriptsprache in Form von Coroutinen zusammenschalten. Diese letztgenannten drei Möglichkeiten sollen im Folgenden genauer betrachtet werden.

Ich arbeite an einem Projekt, das ein neues kleines spezielles Forth für PowerPC–Prozessoren auf Tcl/Tk zum Laufen bringen soll. Hier wird es jedoch nicht um das Forth gehen, sondern um die Methode, wie es in Tcl eingefügt wird. Die Wahl von Tcl/Tk als Skriptsprache ist einigermaßen willkürlich, da ich der Meinung bin, dass jedes Desktop–Forth es verdienen würde, auf *jede* Skriptsprache portiert zu werden; also insbesondere auch auf Tcl/Tk.

Allerdings kann es scheinen, dass Tcl eine besonders Forth–kompatible Sprache ist. Aus Forth–Sicht besteht ein Tcl–Skript aus einer Serie von Befehlswörtern; und zwar handelt es sich dabei jeweils um ein *parsing word*, das die folgenden Wörter bis zum Zeilenende liest, dann bestimmte Substitutionen vornimmt, wodurch die Argumente für das Befehlswort gebildet werden, die kommen dann — sozusagen — auf den Stapel und schließlich wird der Befehl ausgeführt.

Dieses kleine Beispiel zeigt etwa den Umgang von Tcl mit Variablen:

```
% set x 17
17
% puts $x
17
```

Das Kommando `set` parst die folgende Zeile, die Zeichenkette `x` wird der Name der Variablen, die Zeichenkette `17`

```
int forthevaluate(int lTOS, int lSP, int lRP, int systemvariable,
                  int dataspace, int addressofentry) {
    int linkvar, countvar, resultat;

    asm{
        mr      r3,lTOS  /* "move register", nämlich lTOS nach Register r3 */
        mr      r4,lSP   /* man kann hier keine globalen Variablen benutzen */
        mr      r5,lRP
        mr      r6,systemvariable
        mr      r7,dataspace
        mfctr   countvar /* "move from count register", nämlich "to countvar" */
        mflr    linkvar         /* "move from link register" - "to linkvar" */
        mtctr   addressofentry  /* "move to count register" */
        bctrl   /* "branch to count register and link": die Codeadresse der nächsten*/
                /*  Zeile ist jetzt auf dem link register. */
                /* Forth sichert sie dann als erstes für den Rücksprung. */
        mtlr    linkvar        /* alten Inhalt des link register wiederherstellen */
        mtctr   countvar       /* alten Inhalt des count register wiederherstellen */
        mr      dataspace,r7
        mr      systemvariable,r6
        mr      lRP,r5
        mr      lSP,r4
        mr      lTOS,r3
        mr      resultat,r8    /* Forth setzt die Flagge für Tcl auf Register r8 */
     }

    TOS = lTOS;
    SP = (int *)lSP;
    RP = (int *)lRP;
    return resultat;
}
```

Listing 3: Der Übergang von C nach Assembly

ihr Wert. Das Kommando puts parst den Rest der Zeile, da steht nur `$x`, `x` wird als Variablenname erkannt, das `$`–Zeichen holt den Wert aus der Variablen (substituiert die Zeichenkette `$x` durch die Zeichenkette `17`), damit ist der Parameter für `puts` gebildet, und `puts` schreibt ihn auf die Konsole.

Nun besteht bekanntlich auch Forthcode aus einer Folge von Befehlswörtern, wobei manche den Quellcode parsen, so dass sich Forth im Kontext von Tcl und Tcl im Kontext von Forth ganz gut sehen lassen könnten. (Aus Tcl–Sicht erscheint ein Forth–Wort wie ein parameterloser Befehl mit gewissen Seiteneffekten bzgl. Stapel, Wörterbuch, Datenraum.)

Man kann Tcl durch neue Befehlswörter erweitern. Zum Beispiel ließe sich ein Tcl–Befehl `forth` definieren, der die folgende Zeile liest und sie per `EVALUATE` bearbeitet. Umgekehrt kann Forth ein Wort `tcl-eval` enthalten, das Tcl–Code ausführen lässt, z. B.:

```
S" set x 17" tcl-eval
```

Neue Befehlswörter für Tcl lassen sich in C schreiben. Man erzeugt hierfür eine Datei mit Binärcode, die zur Laufzeit in Tcl geladen wird. Eine Möglichkeit hierfür liefert der Tcl–Befehl `package`.

```
% package require forth
```

Der Package–Befehl ruft nun eine Funktion namens `Forth_Init` auf. Die ist dafür zuständig, aus einer Datei die Konfigurationsdaten des Forth zu lesen, Speicher für Wörterbuch, Coderaum und Stapel zu besorgen, die Adressen an Orten zu notieren, wo Forth sie lesen kann — also alles, um Forth so weit zu starten, dass es Zeichenketten als Forth–Code evaluieren kann. Außerdem erzeugt `Forth_Init` einen Tcl–Befehl namens `forth`.

So ein Tcl–Befehl besteht wie gesagt aus einem Befehlswort am Zeilenanfang, gefolgt von einer Serie von Parametern, die am Zeilenumbruch (oder auch an einem Semikolon) enden. Theoretisch bräuchte man für `forth` nur einen Parameter, nämlich die Zeichenkette, die an `EVALUATE` übergeben wird. Dennoch kann man sich die Tatsache, dass mehrere Parameter möglich sind, zunutze machen: in Tcl gilt nämlich der Grundsatz, dass „Alles eine Zeichenkette ist“, aber Tcl–Variablen enthalten intern oft eine effizientere Darstellung, z. B. würde die Variable `x` nach

```
% set x 17
```

die Zahl `17` unter Umständen nicht nur als Zeichenkette `"17"`, sondern auch als Binärzahl enthalten. Die Umwandlung der Zeichenkettendarstellung in die Binärdarstellung erfolgt z. B., wenn eine mathematische Operation ausgeführt wird wie `incr $x`. `x` enthält jetzt die

```
:ppc_code EVALUATE
    r9              mflr,   \ Rücksprungadresse nach forthevaluate(C)
    r9  16  r6      stw,    \ an systemvariable[4] speichern

    \ Schleife: parsen, finden, usw.

    r9  16  r6      lwz,    \ toplevel Rücksprungadresse nach r9 geholt
    r9              mtlr,   \ toplevel Rücksprungadresse nun auf dem link register
                    blr,
;ppc_code
```

Listing 4: `EVALUATE` in PowerPC–Assembly

Binärzahl `18`, während die Zeichenkettendarstellung als ungültig gesetzt wird. Ein weiteres `incr $x` würde dann unmittelbar die Binärzahl verwenden und müsste nicht erst die Zeichenkette in eine solche umwandeln. Der Befehl `puts $x` würde dann aus der Binärzahl `18` wieder die Zeichenkette `"18"` machen — die Zeichenkettenversion wurde ja ungültig gesetzt — um sie auszugeben. Dieser Mechanismus hat Tcl schnell gemacht.

Wir können ihn auch für den `forth`–Befehl nutzbar machen, zum Beispiel so: Wir gehen die Parameter der Reihe nach durch und prüfen, ob es sich um Zahlen handelt. Falls ja, wird die Zahl sofort auf den Stapel gelegt. Treffen wir auf einen Parameter, der keine Zahl ist, dann verbinden wir diesen und alle weiteren Parameter zu einer einzigen Zeichenkette und evaluieren sie. Auf die Weise ist es möglich, Folgendes zu schreiben (in Tcl):

```
% set x 17; set y 4
4
% forth  $x $y + .
21
```

Das ist unter Umständen sehr viel schneller als:

```
% forth "$x $y + ."
21
```

Sofern `x` und `y` intern ihren Wert als Binärzahl halten, ist nämlich im ersten Fall keine Umwandlung Zahl → Zeichenkette → Wörterbuchsuche → Zahl nötig, wie sie im zweiten Fall geschieht, da die Zahlen `17` und `4` sofort auf den Stapel kommen und dann nur noch `+ .` in Forth evaluiert wird.

Der C–Code hierfür sieht etwa so aus, wie in Listing 1 auf Seite 22 zu sehen ist.

Der Parameter `Tcl_Obj * CONST objv[]` enthält die Serie der Tcl–Parameter mit denen der Befehl forth aufgerufen wurde:

```
% forth  par1 par2 par3
```

ergibt `objc = 4`, `objv[0] = "forth"`, `objc[1] = "par1"`, etc.

(1) Die C–Funktion `push(zahl)` legt `zahl` auf den Stapel. Sie kann z. B. so implementiert werden:

```
push(int zahl) {
  SP = SP-stackcellsdistance;
  *SP = zahl;
}
```

(2) `Tcl_ConcatObj` kopiert die Argumente in einen neuen Speicherbereich. Das ist etwas ineffizient; man könnte auch den Parameter `objv[objc]` direkt verwenden, falls es der erste Parameter ist, der keine Zahl enthält.

Jetzt kann das Forth aufgerufen werden. Da Forth selbst zwischendurch immer wieder Tcl–Funktionen benötigt (wie z. B. für `ALLOCATE`) — und ohnehin ein enges Zusammenspiel mit Tcl angestrebt wird — folgt nun eine Art von Coroutinen–Konstruktion: die Funktion `Forth_Cmd` ruft in einer Endlosschleife immer wieder Forth auf; braucht Forth eine Tcl–Funktion, unterbricht es sich, speichert die Adresse der nächsten Instruktion und gibt eine Flagge an `Forth_Cmd` zurück. Mit Hilfe der Flagge wird eine Tcl–Funktion aufgerufen und dann geht es entweder nach Forth zurück oder nach Tcl. Das sieht dann etwa so aus wie in Listing 2 auf Seite 23.

Hierbei enthält `systemvariable[3]` das Forth–xt, welches ausgeführt werden soll, also in der Regel

```
:ppc_code tcl-eval
    r8  3       li,     \ "load immediate", nämlich die Zahl 3 auf Register r8.
                        \ Dies ist der Rückgabewert für switch case Tcl_EvalObjEx
    r10         mflr,   \ Adresse der nächsten Instruktion
                        \ des tcl-eval aufrufenden Wortes nach r10 holen und
    r10 12  r6  stw,    \ an systemvariable[3] speichern (NB 3 4 * 12 =). (stw: "store word")
    r9  16  r6  lwz,    \ toplevel Rücksprungadresse aus systemvariable[4]
                        \ nach r9 holen ("load word")
    r9          mtlr,   \ und auf das link register schieben
                blr,    \ "branch to link register":
                        \ zurück geht's nach forthevaluate!
;ppc_code
```

Listing 5: `tcl-eval` in PowerPC–Assembly

`[']` `EVALUATE`. Es wird bereits in `Forth_Init` bestimmt und dort hineingeschrieben.

Da das Befehlswort `forth` in C geschrieben wird, liegt es sicherlich nahe, auch das Forth selbst, d. h. die Funktion `forthevaluate` in C zu schreiben; bzw. dürfte es für C–Programmierer nicht schwer sein, C–basierte Forths so nach Tcl zu portieren. Mein eigenes Forth ist allerdings in PowerPC–Assembly geschrieben, so dass also noch der Übergang von C nach Assembly gemacht werden muss, wie in Listing 3 auf Seite 24 zu sehen ist.

(So elegant geht das nur mit gcc leider nur unter Mac OS X. Bei anderen Systemen muss man gcc inline assembly verwenden.)

`addressofentry` ist hier wie gesagt in der Regel das xt von `EVALUATE`. Um Subroutinen aufzurufen hat der PowerPC zwei spezielle Register: das `count`–Register und das `link`–Register. Hier wird das `count`–Register verwendet. (Ich hoffe die Kommentare im Code machen einigermaßen klar, was hier passiert, auch wenn man PPC–Assembly vielleicht nicht versteht.)

Forth ist nun dran und tut seine Arbeit: Eingabe parsen, xt finden usw. (siehe Listing 4 auf der vorherigen Seite).

An `systemvariable[4]` steht jetzt die Adresse des Befehls `mtlr linkvar` aus `forthevaluate`. Damit kann sich Forth selbst jederzeit unterbrechen, um Tcl zu bemühen (Listing 5 auf der vorherigen Seite).

`Forthevaluate` gibt nun die Flagge `3` an `Forth_Cmd` zurück, also landet man in `case 3` des `switch`–Blocks:

```
case 3: /* addr-len-String als Tcl-Code evaluieren */
  resultat = Tcl_EvalObjEx(
               interp,
               Tcl_NewStringObj((char *)*NOS,TOS),0);
  *NOS = Tcl_GetStringFromObj(
               Tcl_GetObjResult(interp),&TOS);
  break;
```

`NOS` ist hierbei `SP+stackcellsdistance`. Das Kommando `Tcl_EvalObjEx` wertet die Zeichenkette, die auf dem Stapel liegt (welcher ja von `Forth_Cmd` aus zugänglich ist, da `forthevaluate` `TOS` und `SP` in C–Variablen sichert), als Tcl–Code aus und schreibt das Ergebnis in die Result–Variable des Tcl–Interpreters, der den `forth`–Befehl auswertet. Dort holen wir es — als Zeichenkette — heraus, wobei der zweite Parameter von `Tcl_GetStringFromObj` die Länge der Zeichenkette enthält: so liegt das Ergebnis als (c-addr len)–Zeichenkette auf dem Stapel. Das `break` verlässt den `switch` und die `while`–Schleife ruft wieder `forthevaluate`. Diesmal wird jedoch nicht das xt von `EVALUATE` übergeben, sondern die Adresse der nächsten Instruktion des `tcl-eval` aufrufenden Wortes, d. h. Forth setzt sein Werk an der Stelle fort, wo `tcl-eval` es unterbrochen hatte.

Angenommen, in Tcl–Hochsprache sei Folgendes kodiert gewesen:

```
% forth { S" puts $x" tcl-eval TYPE}
17
```

(In Tcl schließen geschweifte Klammern Zeichenketten ein.)

`tcl-eval` unterbrach den Textinterpreter, der nun weitermacht, das Wort `TYPE` liest und es ausführt. `TYPE` unterbricht Forth wieder und ruft eine Tcl–Funktion, die NOS und TOS als Zeichenkette erkennt und in einer *dynamischen Zeichenkettenvariablen* (DString) speichert:

```
case  6: /* unterstützt TYPE */
   Tcl_DStringAppend(&dsPtr,(char *)*NOS,TOS);
   twodrop(); break;
```

(`dsPtr` sammelt zwischenzeitlich alle Ausgaben per TYPE oder . (dot).)

Jetzt ist erstmal der Forthinterpreter wieder dran, aber da die Eingabe erschöpft ist, geht es mit der Flagge `0` nach Tcl zurück, infolgedessen wird die Variable `ende` auf `1` gesetzt, und die Endlosschleife `while(1)` verlassen.

Es folgt nur noch:

```
Tcl_DStringResult(interp, &dsPtr);
return TCL_OK;
```

Damit endet `Forth_Cmd`, die Result–Variable des Tcl–Interpreters enthält nun den DString `dsptr` mit allen Textausgaben, die der Forth–Aufruf erzeugt hat.

Sowohl das Forth als auch die Verbindung Forth–Tcl sind bislang erst rudimentär implementiert, eine ganze Reihe von Fragen ist noch offen; um nur eines zu nennen, man muss zum Beispiel damit rechnen, dass Tcl–Code, der von Forth aus evaluiert wird, seinerseits den `forth`–Befehl enthält und also Forthcode evaluiert werden soll, und dann stauen sich zwei unvollendete Forth–Evaluierungen an, es entsteht ein Stapel von Eingabe–Zeichenketten, der irgendwie verwaltet werden muss.

Dennoch glaube ich, dass man es ungefähr so machen kann, wie es hier skizziert wurde, und dass Forth von der Einbettung in eine Skriptsprache unmittelbar profitieren würde, indem die vielfältigen Möglichkeiten dieser Sprachen sofort zugänglich werden, während umgekehrt Forth bei den Skriptsprachen vielleicht eine neue Nische finden könnte, in der es von Nutzen wäre.

Dallas 1–Wire mit Forth ansprechen

Bernd Paysan

Das 1-Wire-Interface von Dallas/Maxim ist eine etwas neuere Alternative zu Bussen mit zwei oder drei Drähten (I^2C oder SPI). Es gibt inzwischen eine Reihe von kleinen Geräten, die man mit dem Interface ansprechen kann. Der Artikel zeigt, wie das von Forth aus geht.

Einleitung

Wie bereits im Bericht von der Münchner Forth–Gruppe im letzten Heft kurz angesprochen, beschäftige ich mich zur Zeit mit dem 1–Wire–Interface von Dallas. Am Beispiel eines iButton–Thermometers zeige ich hier, wie man mit so einem 1–Wire–Gerät spricht.

Ähnlich wie I^2S ist 1–Wire ein Open–Drain–Bus, d.h. ein Pullup zieht die Leitung nach oben (und versorgt einfache Geräte wie iButtons mit ein wenig Strom). Die Busteilnehmer haben alle einen NMOS–Transistor, der die Leitung kurz auf 0 ziehen kann — ein ganz kurzer Puls ist eine 1, ein etwas längerer Puls ist eine 0, und zwei wesentlich längere Pulse kennzeichnen Reset und die Present–Antwort eines Gerätes.

Serielle Schnittstelle

Als Interface nutze ich den Dallas DS2480B[1], der zunächst USB auf seriell umsetzt, und dann die serielle Schnittstelle auf 1–Wire. Vom Betriebssystem (Linux) wird die USB–Zwischenschicht im Treiber versteckt, d.h. man öffnet `/dev/ttyUSB0` wie eine serielle Schnittstelle. Lesen und Schreiben wird dann über `read-file` und `write-file` erledigt.

Die serielle Schnittstelle muss ich dann noch auf die richtige Geschwindigkeit setzen, und auch sonst noch einige Flags setzen — schließlich ist der DS2480B kein Terminal. Stattdessen nimmt er Kommandos an — solche zum Konfigurieren (die sind alle in `'stuff` untergebracht), oder solche, um den 1–Wire–Bus anzusprechen (die sind mit Konstanten benannt).

Ich lege mir noch ein paar Buffer an, mit denen ich Daten hin- und herschicken kann, ein `>cmd`, um Kommandos zu schicken, und eine Routine `>results`, um die Resultate ohne blockierendes Warten vom DS2480B zurückzubekommen. Damit kann ich das Protokoll jetzt implementieren.

Das 1–Wire–Protokoll

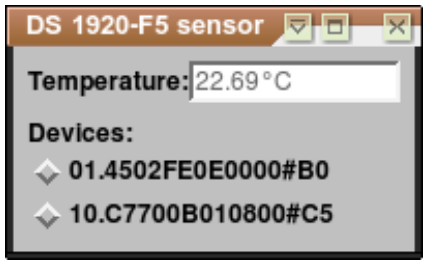
Die Sequenz eines 1–Wire–Packets fängt mit dem Reset– und Present–Puls an. Danach kommt ein Byte, das als "Master ROM Function Command" bezeichnet wird. Die möglichen Reaktionen:

Cmd	Name	Daten
33h	Read ROM	Device schickt 64–Bit–Adresse zurück
55h	Match ROM	Device erkennt 64–Bit–Adresse
F0h	Search ROM	Suche nach 64–Bit
ECh	Alarm search	Suche, wenn Alarm gesetzt ist
CCh	Skip ROM	Keine Adresse

Anschließend kommt dann die tatsächliche „Nutzlast", die Memory– und Kontrollfunktionen. Derer gibt es bei einem iButton–Thermometer fünf, nämlich

Cmd	Name	Daten
4Eh	Write Scratchpad	2 Bytes Scratchpad
BEh	Read Scratchpad	9 Bytes Scratchpad
48h	Copy Scratchpad	–/– (dauert 10ms)
44h	Convert Temp.	–/– (dauert 0.75s)
B8h	Recall	–/– (Trigger Value nach Scratchpad)

In den ersten beiden Stellen des Scratchpads steht die Temperatur auf ein halbes Grad genau, die letzten beiden (vor der Checksumme) enthalten einen Bruch des Restwerts. Damit haben wir nun genügend Informationen, um den Sensor am Bildschirm anzuzeigen:



Listing 1: Sensor

Die Beschreibung der genauen Prozedur beim Auflisten der Devices überlasse ich lieber dem Datenblatt von Dallas. Nur soviel: Auf dem 1–Wire–Bus werden für jedes Bit des Geräts drei Bits versendet, dabei die ersten zwei vom Gerät, das letzte vom Master: Eins normal, eins invertiert, und der Master dann, wie er sich entschieden hat. Auf der seriellen Schnittstelle erhält man für diesen Vorgang zwei Bits zurück, eines für die gewählte Adresse, das andere, ob es bei diesem Bit einen Konflikt gab.

Zusammenfassung

Mit etwas mehr als hundert Zeilen bekommt man einen iButton–Sensor mit Forth zum Laufen. Dallas hat auch ein User–Mode–Dateisystem für Linux geschrieben, das die ganzen Geräte ansprechen kann. Da sind viele tausend Zeilen Code vergraben, die zwar alle wunderschön offengelegt sind, und auch kommentiert, aber einfach so viel, dass ich lieber ein Reverse–Engineering–Tool (strace) angeworfen habe, um damit dann schneller zum Ziel zu kommen.

[1] Datasheet: `http://datasheets.maxim-ic.com/en/ds/DS2480B.pdf`

Listing

1wire.fs

```
\ 1-wire interface

[IFUNDEF] set-baud  include serial.fs [THEN]

0 Value 1-wire-fd

also dos
: set-tty ( fd -- )  filehandle @ >r
    t_old r@ tcgetattr drop
    t_old t_buf sizeof termios move
    $805 t_buf termios c_iflag !
    $000 t_buf termios c_oflag !
    $8BD t_buf termios c_cflag !
    0    t_buf termios c_lflag !
    0 t_buf termios c_cc VMIN + c!
    3 t_buf termios c_cc VTIME + c!
    $D dup t_buf termios c_ispeed ! t_buf termios c_ospeed !
    t_buf 1 r> tcsetattr drop ;
previous

: open-1-wire ( addr u -- ) r/w open-file throw to 1-wire-fd
    1-wire-fd set-tty ;

s" /dev/ttyUSB0" open-1-wire

: 1w-write ( addr u -- )
    1-wire-fd write-file throw ;
: 1w-read ( addr u -- n )
    1-wire-fd read-file throw ;
: 1w? ( -- n )  1-wire-fd check-read ;

$E3 Constant :cmd
$E1 Constant :data
$C1 Constant :reset
$C5 Constant :reset2
$BE Constant :read
$B1 Constant :accel
$A1 Constant :accel-off
Create cmdbuf  0 c,
Create 'cmd   :cmd c, 0 c, :data c,
Create 'address 8 0 [DO]   0 c, [LOOP]
Create 'dummies 16 0 [DO] $FF c, [LOOP]
Create 'search   16 0 [DO]   0 c, [LOOP]
Create 'stuff  $17 c, $45 c, $5b c, $0f c, $91 c,

Variable results $100 allot

Variable timeout

: waitx             &10 wait  timeout @ 0 after - 0< ;

: >results ( n -- )  &100 after timeout !
    BEGIN  dup 1-wire-fd check-read > WHILE  waitx  UNTIL
        results off  true abort" Timeout"  ELSE
        results cell+ swap 1w-read results !  THEN ;

: .results ( -- ) base push hex
```

```
58      results @+ swap bounds ?DO
59          I c@ 3 .r  LOOP ;
60
61  : >address ( -- )
62      'address 8 erase
63      $40 0 DO  results cell+ I 2* 1+ bit@ IF
64              'address I +bit
65          THEN  LOOP ;
66
67  Code bswap  AX bswap  Next end-code macro
68
69  : "address ( -- addr u ) base push hex
70      >address 'address 2@ bswap swap bswap swap
71      <#  # # '# hold 12 0 DO # LOOP '. hold # # #> ;
72  : .address "address type ;
73
74  : maxdisc ( -- n )  0
75      $40 0 DO  results cell+ I 2* bit@ IF  drop I  THEN  LOOP ;
76
77  : cmd ( n -- ) cmdbuf c! cmdbuf 1 1w-write ;
78  : cmdr ( n -- ) cmd 1 >results ;
79
80  : addr ( -- )  'address 8 1w-write 8 >results ;
81  : dummies8 ( -- )  'dummies 8 1w-write 8 >results ;
82  : dummies9 ( -- )  'dummies 9 1w-write 9 >results ;
83  : search16 ( -- )  'search 16 1w-write 16 >results ;
84
85  : >temp ( -- ) base push decimal  results cell+ w@ 2/
86      &100 * results cell+ 6 + c@ &100 results cell+ 7 + c@ */ - &75 +
87      extend under dabs <# 'C hold '° xhold # # '. hold #S rot sign #> ;
88
89  : .temp ( -- ) >temp type ;
90
91  : >cmd ( n -- ) 'cmd 1+ c!  'cmd 3 1w-write ;
92  : >cmdr ( n -- ) >cmd 1 >results ;
93
94  : reset2 ( -- )  :reset2 >cmdr ;
95  : init ( -- )    :reset >cmdr 'stuff 5 1w-write  5 >results reset2 ;
96  : accel ( -- )   :accel >cmd ;
97  : accel-off ( -- ) :accel-off >cmd ;
98
99  \ "skip ROM": no address
100 : noaddress reset2 $CC cmdr ;
101
102 \ "search ROM": apply an address
103 : address
104     reset2 $55 cmdr addr ;
105
106 : search-first ( -- ) 'search 16 erase
107     init $F0 cmdr accel search16 accel-off ;
108 : search-next ( -- )  results cell+ 'search 16 move
109     'search maxdisc 2* 1+ +bit
110     $40 maxdisc 1+ DO 'search I 2* 1+ -bit LOOP
111     init $F0 cmdr accel search16 accel-off ;
112
113 : search-all ( -- )  search-first .address cr maxdisc
114     BEGIN  search-next .address cr maxdisc tuck =  UNTIL  drop ;
115
116 : readout
117     noaddress
```

```
118      $44 cmdr &750 ms \ convert temperature
119      noaddress
120      $BE cmdr \ read scratchpad
121      dummies9 ;
122
123  : readrom \ would work if host didn't reply as well
124      reset2 $33 cmdr
125      dummies8 ;
126
127  init search-first >address
128
```

1wire.m

```
1   #! xbigforth
2   \ automatic generated code
3   \ do not edit
4
5   also editor also minos also forth
6
7   component class ds1920-f5
8   public:
9     early widget
10    early open
11    early dialog
12    early open-app
13    infotextfield ptr temp
14    varbox ptr devices
15    text-label ptr dev-first
16   ( [varstart] )  ( [varend] )
17  how:
18    : open     new DF[ 0 ]DF s" DS 1920-F5 sensor" open-component ;
19    : dialog   new DF[ 0 ]DF s" DS 1920-F5 sensor" open-dialog ;
20    : open-app new DF[ 0 ]DF s" DS 1920-F5 sensor" open-application ;
21  class;
22
23  include 1wire.fs
24  ds1920-f5 implements
25   ( [methodstart] ) : show super show
26    search-first ^^ 0 T[ ][ ]T "address rbutton new
27    dev-first widgets self devices add  maxdisc
28    BEGIN  search-next ^^ 0 T[ ][ ]T "address rbutton new
29           dev-first widgets self devices add  maxdisc tuck = UNTIL
30    drop
31    ^ 1 $1000 dup NewTask pass >o
32    BEGIN  readout >temp temp assign  AGAIN ; ( [methodend] )
33    : widget  ( [dumpstart] )
34          ^^ ST[  ]ST ( MINOS ) T" " X" Temperature:" infotextfield new  ^^bind temp
35            X" Devices:" text-label new  ^^bind dev-first
36          &1 varbox new ^^bind devices
37        &2 vabox new panel
38      ( [dumpend] ) ;
39    : init  ^>^^  assign  widget 1 :: init ;
40  class;
41
42  : main
43    ds1920-f5 open-app
44    $1 0 ?DO  stop  LOOP bye ;
45  script? [IF]  main  [THEN]
46  previous previous previous
```

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

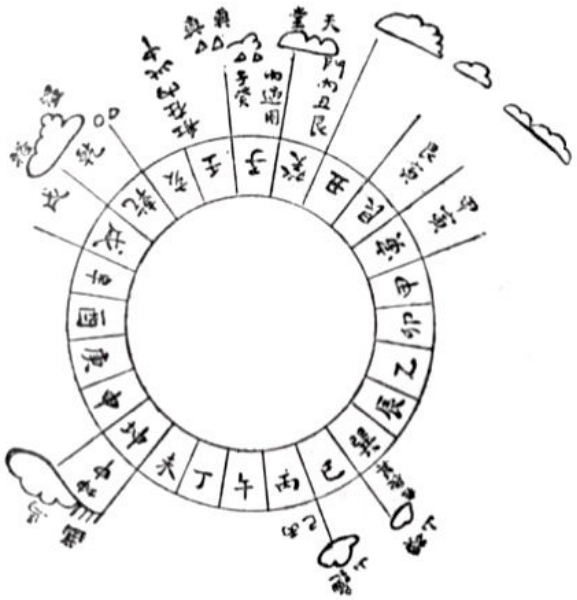
Carsten Strotmann
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

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`