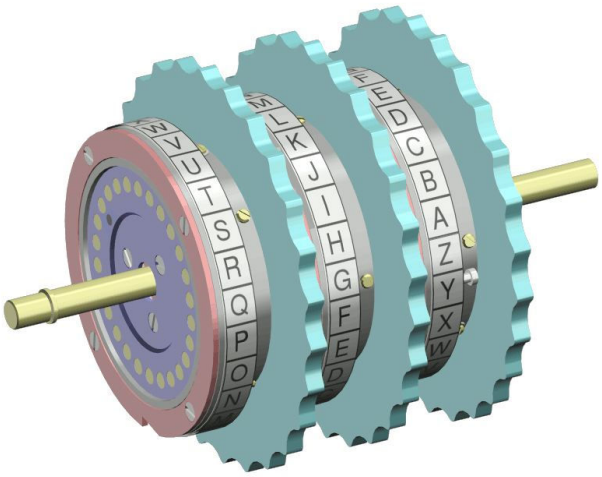
Vierte Dimension

2/2009

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.

Titelgrafik: ENIGMA–Walzensatz (Quelle: Wikipedia)

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 32 01 24
68273 Mannheim
Tel: ++49(0)6239 9201-85, Fax: -86
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

Manuskripte und Rechte

Berücksichtigt werden alle eingesandten Manuskripte. Leserbriefe können ohne Rücksprache wiedergegeben werden. Für die mit dem Namen des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung. Die in diesem Magazin veröffentlichten Beiträge sind urheberrechtlich geschützt. Übersetzung, Vervielfältigung, sowie Speicherung auf beliebigen Medien, ganz oder auszugsweise nur mit genauer Quellenangabe erlaubt. Die eingereichten Beiträge müssen frei von Ansprüchen Dritter sein. Veröffentlichte Programme gehen — soweit nichts anderes vermerkt ist — in die Public Domain über. Für Text, Schaltbilder oder Aufbauskizzen, die zum Nichtfunktionieren oder eventuellem Schadhaftwerden von Bauelementen führen, kann keine Haftung übernommen werden. Sämtliche Veröffentlichungen erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes. Warennamen werden ohne Gewährleistung einer freien Verwendung benutzt.

Liebe Leser,

willkommen zur zweiten Ausgabe unseres Forth–Magazins in diesem Jahr.

Ende März fand in Neuenkirchen bei Rheine die Jahrestagung der Forth–Gesellschaft statt. Da im Vorfeld ab Mittwoch bereits die Sitzung des Forth–200x–Standard–Komitees stattfand, wurde in Wilminks Parkhotel ganze 5 Tage über Forth gefachsimpelt und neueste Neuigkeiten ausgetauscht. Bernd Paysan und Klaus Schliesiek haben `Microcore` und `b16` auf das Altera–DE1–FPGA–Board portiert. Informationen über die Vorträge auf der Tagung finden sich auf unserer Web-Seite unter `http://www.forth-ev.de/article.php?story=20090330200148744`.

Auf dem Linuxtag (24.–27. Juni) in Berlin hat die Forth–Gesellschaft auch in diesem Jahr mit einem Stand über Forth informiert. Als Publikumsmagnet hat sich wieder einmal der Triceps–Roboter erwiesen, der aber diesmal nicht von einem PC gesteuert wurde, sondern von einem `b16`–Prozessor, der eben genau auf dem Altera-DE1–FPGA–Board lief. Außerdem konnten die Messebesucher AVR–Singleboard–Computer mit `amForth` und einer Sensor–Valance (Luftdruck, Temperatur, Feuchtigeit) bestaunen und sich so über die letzten Entwicklungen von Forth auf eingebetteten Systemen informieren.

In zahlreichen Gesprächen hat das Forth–Gesellschafts–Standpersonal (Bernd Paysan, Ewald Rieger, Carsten Strotmann, Erich Wälde, Ulrich Hoffmann) über die Vorzüge von Forth berichtet und spannende, interessante und teilweise auch kontroverse Diskussionen geführt.

Außerdem auf dem Linuxtag vertreten waren — neben den zahlreichen Ständen der Linux–Distributionen — das Core–Boot–Projekt `www.coreboot.org`, das ein freies BIOS zum Booten von PCs entwickelt; der Dante e.V. (`www.dante.de`), die deutsche TEX–Anwendervereinigung; das One–Laptop–Per–Child–Projekt (`www.laptop.org`) und die CAcert–Gemeinschaft (`www.cacert.org`), die sich für kostenfreie digitale Zertifikate einsetzt.

Aufgrund der Diskussionen mit den CAcert–Vertretern ist die Forth–Gesellschaft mittlerweile als Organisation durch CAcert zertifiziert und kann nun für ihre Mitglieder selbst Client–Zertifikate ausstellen, die sie dann für den sicheren Email–Transfer einsetzen können. Weitere Informationen kann ich gerne auf Anfrage geben (uho@forth-ev.de).

Ich freue mich, dass ich infolge des Linuxtages auch unsere neuen Mitglieder Gido Baumann, Michael Heycke, David Kühling und Erhadt Petter in der Forth–Gesellschaft begrüßen darf. Herzlich Willkommen!

So — und nun viel Vergnügen mit dem aktuellen Heft.

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

Zur Entstehung des Win32Forth

Win32Forth ist freie Software (public domain), ANS–kompatibel, und ein komplettes Entwicklungssystem für Applikationen, das die Sprache Forth benutzt. Begonnen haben es in den Jahren 1994 und 1995 Andrew McKewan, Tom Zimmer, Robert Smith und Jim Schneider. Es war als freie Software gemeint von Anfang an, mit Ausnahme von Jim Schneiders Assembler, der unter GPL gestellt wurde. Das Win32Forth ist ein fettes System geworden und folgt der Philosophie, die besagt, dass wenn ein Werkzeug gut ist, viele Werkzeuge besser sind. Es bestand zunächst aus der forth–üblichen interaktiven Konsole, und einem integrierten und erweiterbaren Quellcode–Debugger sowie dem WinEd Hypertext File Editor, mit dem man die Megabytes Quellcode erkunden konnte, aus denen Win32Forth inzwischen bestand. Nun sind noch dazugekommen eine komplette Entwicklungsumgebung (IDE, integrated development environment), mit der ForthForm, um Windows–Fenster zu erzeugen, einzurichten und in die eigene Forth–Applikation einzubinden.

Vielleicht möchte der eine oder andere auch etwas über Tom Zimmers Motive erfahren. Im Interview, das er in 2000 an Jim Lawless gab, ist einiges davon festgehalten: An Interview with Tom Zimmer: Forth System Developer In: `http://www.radiks.net/~jimbo/art/int6.htm`

Zur Projektgruppe

Die Win32Forth–Projekt–Gruppe hat sich 2002 auf Initiative von John A. Peters gebildet auf TopicA.com. Sie wuchs zu einem Verbund von Programmierern und Nutzern des Win32Forth Systems vom Tom Zimmer heran. Toms neue Aufgaben — hauptsächlich Java — zogen ihn schließlich seit 2001 vom Projekt ab. Tom versicherte jedoch, dass sein Produkt freie Software bleibe. Die Gruppe hat dem eigenen Bekunden nach, vor sein System weiter zu pflegen und zu einem System zu entwickeln, das ins 21. Jahrhundert passt. Kontakt: `http://tech.groups.yahoo.com/group/win32forth/`

Die Entwicklung des Systems erfolgt über das CSV bei: `http://sourceforge.net/projects/win32forth` Derzeit ist die beta-version 6.13 dort zu finden. Der letzte Release war 6.12.

Ein CVS (Concurrent Versions System) ist ein Werkzeug, das inzwischen von vielen Software–Entwicklern benutzt wird. Es dient dazu, die verzweigten Veränderungen im Baum von Quellcodes zu managen. Das CVS enthält den neusten Stand der Win32Forth–Ausgaben der Entwickler. Sie benutzen derzeit das Programm TortoiseCVS, um Entwicklungen in das CVS hochzuladen (commit) und die neuen Versionen zur Bearbeitung herunterzuholen (update on harddisk).

Wenn man selber etwas für das Win32Forth entwickelt hat, kann man Mitglied der Projektgruppe werden. Wie das im Einzelnen geht, besonders welche Software man dafür wie nutzen kann, ist näher beschrieben bei: Win32Forth, How to join us `http://win32forth.sourceforge.net/howtojoin.htm` (mka)

Projekt Rekonstrukt — interaktives Forth auf FPGAs

Passend zu unserer *Forth-im-FPGA*–Artikelserie ist das Projekt Rekonstrukt [1] von Hans Hübner, das ein interaktives Forth auf Xilinx-Spartan-3–FPGAs bietet.

Dazu kombiniert Hans Hübner das System–on–a–Chip *System09* [2] — eine VHDL–Implementierung eines 6809–Mikrocomputers — von John Kent und das *Maisforth* [3] von Albert Nijhof. Ein 6809–Simulator steht auch zur Verfügung, sollte im Moment kein Spartan–FPGA in Reichweite sein :-)

Nach passender Konfiguration des FPGAs meldet sich Maisforth an der seriellen Schnittstelle und kann in gewohnter Weise programmiert werden.

Ein Portieren auf das Altera–DE1–Board sollte nicht zu schwierig sein. Unser Mikrocontroller–Verleih hat ein Board im Angebot. Wer wagt's?

[1] `http://code.google.com/p/rekonstrukt/`
[2] `http://members.optushome.com.au/jekent/system09/index.html`
[3] `http://home.hccnet.nl/anij/index.html`

(uho)

Zu Carstens Ausführungen im vorliegenden Heft 2/2009

Als FG–Korrekturleser habe ich den Vorteil, mich mit den Artikeln schon vor Drucklegung intensiv befassen zu dürfen. Diesmal hat mich Carsten Strotmanns Beitrag beeindruckt und zum Nach– und Weiterdenken angeregt. Mehr oder weniger zum eigenen Brainstorming darf ich ein paar Stichpunkte hinzufügen. Punkte, bei denen ich mich (erinnerungsbedingt) irre, bitte ich, als Frage zu werten.

(1) Über DOS–Extender habe ich das schöne Buch *Extending DOS* von Duncan/Petzold/Baker/Schulman/Davis/Nelson/Moote seit 1990 im Regal: 10 in sich geschlossene Beiträge von 7 bekannten Autoren aus dem angelsächsischen Bereich.

(2) Wichtige Bestandteile zum Ansprechen des Protected–Modes vom Real–Mode aus sind EMS und DPMI (DOS–Protected–Mode–Interface).

(3) Eine zentrale Rolle beim Plazieren und Einholen von Inhalten aus dem hohen und ganz hohen Speicherbereich spielt der Interrupt 15h. RAM–Disks (ramdrive.sys) machen davon Gebrauch.

(4) Rick VanNorman hat sein schönes 32–Bit–Forth–System auf DPMI aufgebaut.

(5) Rick fügt seinem System für die Verwendung unter DOS einen Forth–DPMI–Extender an.

(6) Windows 3.11 enthält DPMI. (`himem.sys` in der `config.sys` ist dabei wichtig.) Wirft man Win–3.11 an und geht von dort aus wieder auf die DOS–Ebene zurück, dann kann man Ricks Forth–System sofort aufrufen und verwenden.

(7) Im VD–Heft 1/1995 lieferte Michael Schröder ein (experimentelles) Forth–System auf DPMI–Basis (*Extending Forth*).

(8) Beim *Protected*–Mode steht der *Schutz* im Vordergrund. Die 32–Bit–Adressierung (Flat–Model) ist dabei zweitrangig: Zur 32–Bit–Adressierung ist überhaupt kein Protected–Mode nötig. Sie lässt sich sehr schön schon im Real–Mode bewältigen (4 GByte an Adressraum!), vorausgesetzt, vom Protected–Mode aus wurden vorher ein paar *Schalter* umgelegt. Ein *Umschaltprogramm* in Forth (Turbo–Forth) findet man im VD–Heft 2/1998. (Das *Interrupt–Problem* wurde dabei noch nicht gelöst. Man könnte es mit den heutigen Hilfsmitteln und Kenntnissen vielleicht mal angehen). Die Flat–Adressierung läuft über das Präfix 67h, der Aufruf der erweiterten Register über 66h. Protected–Mode–Programme können also im Real–Mode nicht so ohne Weiteres ;-(ausgeführt werden. Die Präfixe 64h und 65h sind für die Einbeziehung der Segmente FS: und GS: von großem Wert.)

Windows–APIs sind gut, aber wozu den Protected–Mode, wenn es im Real–Mode auch geht? Es sind ja doch (weitestgehend) immer dieselben Maschinenbefehle, ob im Real– oder im Protected–Mode. Und Forth bleibt Forth — und kümmert sich weder um Schutz noch um (die unseligen) Schutzverletzungen (die mich seit geraumer Zeit beim Thunderbird (zwar unter *nur* ME, aber immerhin) zur Verzweiflung bringen).

Carsten hat mein Interesse aufs Neue geweckt. Danke, Carsten!

Fred Behringer

Cedrics Codierwettstreit in Heft 1/2009

Hallo VD–Redaktion, ich habe mich sehr über den Aufruf zum Wettstreit in der letzten VD gefreut. Samuel Falvo spricht im Interview auch über Haskell. Grund genug für mich, dass ich mir eine Lösung in Haskell überlegt habe, die ich Euch hiermit vorstellen möchte:

```
module Main where

import Data.List
removeDuplicates = Data.List.nub

unique l =  removeDuplicates l == l

gaps [x] = []
gaps (x0:(x1:xs)) = (x1-x0) : gaps (x1:xs)

largest = 10000
numbers = filter (unique . show) [1..largest]

main = do
  putStr "Here are the numbers: ";
  print numbers;
  putStr "The largets gap is ";
  print (maximum (gaps numbers));
  putStr "Total number of numbers displayed is ";
  print (length numbers)
```

Urteilt selbst über die Kürze und Klarheit dieser Lösung :-)

Es grüßt Euch, Johannes Brooks (uho)

Herausgefischt: Pro und Kontra zu Visual Forth

Neulich in der `Win32Forth@yahoogroups.com` ging es darum, ob man so etwas wie Visual Forth überhaupt braucht. Viel Meinung zum Für und Wieder, wenig Code, aber anregend fand ichs doch. Denn es werden recht unterschiedliche Programmierstile deutlich, wenn es darum geht größere Applikationen zu machen. Camille vertrat hier, soweit ich das erkennen konnte, noch allein, die Weiterentwicklung der `ForthForm` des Win32Forth hin zu einem Visual Forth:

Mittwoch, 22. Oktober, 2008 09:10 Uhr

Wie ich Forth benutze, hatte ich hier in der Gruppe schon mal erwähnt (und das kann man auch als Antwort zum Thema *Visual Forth* nehmen): Ich benutze die Forth–DLL als Kern, und das GUI (Graphical User Interface, Benutzerschnittstelle) kann dann von einem beliebigen Softwaresystem kommen, etwa von Visual Basic, Visual C++, LabView oder woher auch immer. Das ist sehr praktisch, auch weil einige unserer Auftraggeber ihre eigenen Oberflächen haben oder machen wollen (integrators). Software als DLL vorliegen zu haben ist günstig. Das GUI kommuniziert mit der Kern–DLL über Zeichenketten (strings) in einer Syntax die vom Forth interpretiert wird. So dient Forth sogar als Model für ein MVC–Architektur (Mode–View–Controller [1]). Von daher habe ich gar keinen Bedarf an Visual Forth. — Yves

und .. 20:51 Uhr:

Neulich musste ich LabView für einige Anwendungen benutzen. LabView ist eine Programmiersprache in der alles *visuell* erstellt wird. Man kann damit Programme machen ohne dafür auch nur eine Zeile Code tippen zu müssen — alles von den Kontrollstrukturen bis hin zu den Berechnungen wird in Flußdiagrammen dargestellt. Ganz nett gemacht, aber nach einer Weile merkte ich, dass das textbasierte Entwerfen doch schneller geht. Was ich sagen will ist, dass visuelle Werkzeuge für die visuelle Gestaltung (GUI) gut sind, aber das, was das Programm machen soll — die Logik — macht man besser in regulärem Code. Ich fürchte das ein Visual Forth für die meisten Anwendungsfälle keine Verbesserung der Situation bringt. — Yves

Da stimme ich zu. Die sichtbaren Teile einer Anwendung (oder eine Webseite usw.) mit gestalterischen Designwerkzeugen (visual tools) anzufertigen, macht Sinn, aber andere Teile werden besser in traditoneller Weise gemacht, also textbasiert. — Tom Dixon

.. 21:54 Uhr

LabView kenne ich gar nicht, aber der Beschreibung nach würde ich es auch nicht mögen. Das üble an Visual Basic oder so ist, dass man nicht an die Interna heran kommt. In Forth hingegen kann man das, auch in einem Visual Forth. Ich finde ein Visual Forth fügt nur eine

Schicht Forth hinzu, und es verbietet ja nicht den regulären Forth–Code. Die Idee dahinter ist es ein integriertes Werkzeug (IDE) zu haben, das auch das Windows–Interface erzeugen kann (was eine Pein ist, wenn man es über regulären Code händisch erstellen soll), wobei das Herzstück des Codes immer noch regulärer Forth–Code ist. Und das es dann in Forth geschrieben ist, sodass wir eben keine fremden Werkzeuge benötigen. — Camille

Genau das macht ja die `ForthForm` (des Win32Forth). Es geht um einen Zusatz für die Eigenschaften der Form–Elemente (properties), also ein Feld in dem Forth–Code–Schnipsel für die Aktionen stehen können. Diese legen schon in der Form fest, was die Schalter (pressing buttons) später machen sollen, anstatt dass diese Stellen dem generierten Code einer Form später hinzugefügt werden müssen. Dagegen bin ich gar nicht, solange der Editor dafür auf Scintilla [2] basiert und sich damit in die IDE einfügt — oder einfach die IDE aufruft, und den Code dann in die Datei der Form ablegt statt irgendwo separat.

Ich habe aber auch nichts dagegen, wenn externer Code aufgerufen wird (wie die DLLs), wenn das den Job erledigt der ansteht, solange man den Code durch eine API (oder bei Daten per SQL) von Forth aus konfigurieren kann und das nicht erst anders programmiert werden muss. — George

Donnerstag, 23. Oktober, 2008 00:22 Uhr

Camille, Visual Forth ist bestimmt ein sehr interessantes Konzept von dem auch ich über Jahren geträumt habe, und erreichte den Stand einer einheitlichen Software–Architektur von den untersten Ebenen (low level code words) bis zu den höchsten (Rapid Application Development). Und mit OOP, das war unzweifelhaft ein größerer Fortschritt in Forth, 1000x Dank an Tom Zimmer und andere. Und ich gebe zu, dass ich etwas enttäuscht von der `ForthForm` bin, die doch recht eingeschränkt und dabei auch noch umständlich zu benutzen ist. Was mich angeht, liebe ich LabView. Es ist die einzige Software die ich kenne, die es Studenten ermöglicht einen gutaussehenden Wellengenerator samt digitalem Oszilloskop in einem dreistündigem Praktikum zu erstellen. Man macht daraus EXEs, DLLs, es hat einen handlichen WEB–Server, um die eigenen grafischen Oberflächen mit einem WEB–Browser ansehen zu können, usw. Nur leider ist es keine freeware. ;-) — Yves

.. 12:49 Uhr

Ich hatte nicht viel Erfolg mit der `ForthForm`. Kann an mir liegen oder an der Integration ins Win32Forth. Wo der formspezifische Code am besten hin passt, weiss ich auch nicht, denn in Forth kann man machen was man will. Deides `*.f` oder `*.frm` kommen in Frage und sind von Hand anschließend aus der IDE heraus editierbar. — Georg

Verweise

[1] Model–View–Controller (MVC, „Modell/Präsentation/Steuerung“) bezeichnet ein Architekturmuster zur Strukturierung von Software-Entwicklung in die drei Einheiten: Datenmodell (model), Präsentation (view) und Programmsteuerung (controller). So soll ein flexibler Programmentwurf entstehen, der spätere Änderung oder Erweiterung erleichtert, und eine Wiederverwendbarkeit der einzelnen Komponenten ermöglicht.

Das MVC–Konzept wurde 1979 zunächst für Benutzungsoberflächen in Smalltalk durch Trygve Reenskaug beschrieben (Seeheim–Modell), der damals an Smalltalk im Xerox PARC arbeitete. Es gilt mittlerweile aber als De–facto–Standard für den Grobentwurf aller komplexen Softwaresysteme; teils mit Differenzierungen und oftmals mehreren jeweils nach dem MVC–Muster aufgeteilten Modulen. ...
Quelle: `http://de.wikipedia.org/wiki/MVC`

[2] Scintilla ist ein Werkzeug um Texte zu erfassen und zu edieren. Der Quellcode liegt vollständig offen vor, und die Lizenz erlaubt den Gebrauch in jedem freien oder kommerziellen Produkt.
Quelle: `http://www.scintilla.org/`

Viele Grüße, Michael

Traurig aber wahr...

Neulich in der mail. Dirk fragte:

> *Open Firmware is essentially a specification for a largely machine- independent BIOS based on ANS Forth...*
>
> Weisst Du, ob in den neuen Apple–Computern dieses BIOS immer noch existiert?

Ich wusste es nicht, aber Ulli:

> Nein - Intel-Macs haben einen Intel EFI-Boot-Loader und OpenBoot wird nicht mehr verwendet :-(

(`http://en.wikipedia.org/wiki/Open_Firmware`)

Wirklich jammerschade. (mka)

Der Mythos Forth — wo stehen wir heute?

> „Forth ist eine ungewöhnliche Computersprache, und wurde wohl häufiger auf ganz unterschiedliche Projekte angewendet als alle anderen. Wenn es um pünktlichen Abschluss geht, oder schnellen oder kompakten Code oder die Kombiantion davon, ist es die naheliegendste Wahl. Es wurde auch bezeichnet als *... eines der best gehüteten Geheimnisse in der Welt der Computer.* Und das ist keine Übertreibung: Große Firmen haben professionelle Forth Entwicklungssysteme von Lieferanten wie Laboratory Microsystems, Forth Inc. oder MicroProcessor Engineering bezogen, und sie auf Geheimhaltung eingeschworen. Einige spekulierten (unfreundlich) das solche Firmengiganten den Forthgebrauches dann schamhaft verschwiegen haben, aber ich glaube sie verbergen ihre geheime Waffe gegenüber Rivalen. Wann immer

> Forth direkt mit einer konventionelleren Sprache wie C im Wettstreit war, hat es ohne Anstrengung gewonnen, erzeugte kleineren, schnelleren und zuverlässigeren Code in erheblich kürzerer Zeit. Ich habe nach Beispielen gesucht die gegenteilig ausgegangen wären, konnte aber bisher keinen einzigen Fall finden."

Julian Noble starb am 11. März 2007. Sein *A Beginner's Guide to Forth* ist nach wie vor ein guter Einstieg ins Forth. Seine hier zitierte Meinung aus dem Vorwort hingegen hat in den 80er Jahren wohl gestimmt, aber entspricht heute nicht mehr der Wirklichkeit. Eure Meinungen interessieren mich und vielleicht ja doch auch andere.

Dankbar für Eure Leserbriefe, Euer Michael

Quelle:
`http://www.taygeta.com/fsl/docs/NobleArchive.html`

Zum Josephspfennig (VD 1/2009)

Michael führt mit seinem Beitrag aus der Unterhaltungs–Mathematik den Elektronen–Rechner auf seine ursprüngliche Bestimmung zurück, nämlich auf die Anfertigung numerischer Berechnungen. Und Forth hilft ihm dabei. Er führt (aus der von ihm aufgespürten und übersetzten Literaturquelle) an:

Aus 1 Pfennig im Jahre 0 zu 5% mit Zins und Zinseszins ergab sich im Jahr 2003 der rechnerische Betrag von:

27.679.996.896.051.261.677.068.884.476.135.650.875.110,12 DM.

Damit sollte (natürlich wohl) nicht behauptet werden, dass der wahre Ergebniswert x zwischen

27.679.996.896.051.261.677.068.884.476.135.650.875.110,11 DM
und
27.679.996.896.051.261.677.068.884.476.135.650.875.110,13 DM

liegt?

In Physik und Technik ist man, gerade bei solch langen Zifferreihen, an die Angabe eines (exakten) Fehlerintervalls gewöhnt.

Selbstverständlich wird man aus den obigen Ergebnissen heraus z. B. davon ausgehen dürfen (?), dass

27.000.000.000.000.000.000.000.000.000.000.000.000.000,00 DM
$< x <$
28.000.000.000.000.000.000.000.000.000.000.000.000.000,00 DM

Gibt es, so meine Frage, ein gangbares Verfahren, (mit Forth–Mitteln) eine genauere Aussage zu erreichen?

In Mathematik und Physik (und nicht nur dort) hat man sich daran gewöhnt, zu *idealisieren* und die Idealisierungen (unter Abstrich von mehr oder weniger gerechtfertigten *Vernachlässigungen*) als erste Näherungen zu betrachten. Nehmen wir in diesem Sinne also mal an, dass das Kapital nach Ablauf eines Jahres genau 1,05 mal so viel wie am Anfang des Jahres beträgt, dass also auch Pfennigbeträge bei der Verzinsung penibel berücksichtigt werden. Zur schnellen (ungefähren) Überprüfung liegt der (theoretische) Ansatz

$$x = delog(2003 * log(1,05))$$

nahe. Der Taschenrechner von Windows 95 liefert (wobei sich kein Unterschied zwischen log und ln ergab):

$$x = 2,767999689616e+42,$$

der von Windows XP liefert

$$x = 2,767999689615763453442122197746 4e+42.$$

Enorm, der von vielen hochgejubelte *Fortschritt* von 95 zu XP! Aber was soll's: Das (gleich folgende) Argument bleibt dasselbe. Vergleicht man egal welches dieser beiden Ergebnisse mit der Angabe bei Michael, könnte man zu der Annahme verleitet werden, dass auf jeden Fall zumindest

27.679.996.896.000.000.000.000.000.000.000.000.000.000,00 DM
$< x <$
27.679.996.897.000.000.000.000.000.000.000.000.000.000,00 DM

ist. Ist eine solche Annahme gerechtfertigt? Hebt sich eine solche Annahme belegbar aus einer Art subjektiver Wahrscheinlichkeit (einer plausiblen, aber unbegründbaren Vermutung) heraus?

Könnte man bei solchen Untersuchungen vielleicht die Befehle und die 80–Bit–Register der FPU der PC–Kompatiblen ab dem 486er sinnvoll einsetzen? Sollte man? Oder bringt das nichts? Mit der Steuerung der Gleitkomma–Befehle von Turbo–Forth aus habe ich mich (unter Berufung auf C.H. Ting) 1998 beschäftigt (VD–Heft 1/1998). In mein Transputer–Forth F–TP 1.00 (auf taygeta.com) hatte ich sie zu dieser Zeit schon eingebaut. Ein VD–Artikel, der aus den bei Conklin and Rather angegebenen 55 Gleitkomma–ANS–Forth–Befehlen (die alle mit F beginnen) schöpft, ist mir leider noch nicht begegnet. Vielleicht ist das aber auch meiner Aufmerksamkeit einfach nur entgangen (?)

Der Übergang beim Josephspfennig zur echten Zinseszinsrechnung, also zum kontinuierlichen Zinszuschlag, d.h. zur Exponentialfunktion, liegt nahe. (Damit hätte man auch einen Zugriff beispielsweise zur Berechnung des Spannungsabfalls beim Goldcap–Kondensator (ein Kondensator ist kein Akku, aber zur Überbrückung eines kurzen Netzaussetzers würde er reichen) oder zur Halbwertbestimmung der Reststrahlung beim radioaktivierten Kohlkopf aus der weiteren Umgebung von Tschernobyl.) Gleitkomma–Befehle sind eine gute Sache, aber zum Aufbau der e–Funktion wird eine ganze Folge von ihnen benötigt. Ich habe mich vor einiger Zeit unter Einbeziehung von FPU–Assembler–Befehlen mit der Aufbereitung der e–Funktion für Turbo–Forth und ZF beschäftigt. Vielleicht stelle ich das mal gelegentlich in die Vierte Dimension — und beziehe mich als Motivation auf den Josephspfennig. Dass die e–Funktion ein wichtiger Bestandteil eines Digital–Differential–Analyzers (zur Lösung von Differentialgleichungen) ist und dass ich mich von dem Gedanken nicht lösen kann, sowas mal mit den heutigen Maschinen–Kapazitäten unter Forth in Angriff zu nehmen, sei nur nebenbei bemerkt.

Fred Behringer

Das Beste beider Welten

Carsten Strotmann

Hin und wieder vermisse ich die simple Welt der DOS–Betriebssystem–Zeit. Keine Frage, grafische Benutzungsoberflächen wie Gnome, KDE, MacOS X Finder oder die Windows–GUI haben ihre Vorteile. Aber wenn ich nur kurz ein Forth–Prograämmchen starten möchte, dann ist ein kleines DOS–System auf einem USB–Stick genau das Richtige. Kein minutenlanges Warten, dass mehrere hundert Megabyte an Programmen geladen sind, kein Anmelden, kein scheinbar endloses Rattern der Festplatte. Sondern anschalten und loslegen.

Arbeite ich dann mit einem (Real–Mode–)DOS-basierten Forth wie VolksForth, so zeigt sich schnell, dass die DOS–Umgebung im Gegensatz zu neueren Betriebssystemen Einschränkungen hat: segmentiertes Speichermodell, nur 1MB direkt adressierbar, Grafikzugriff abhängig von der Grafikkarte, kein Netzwerk und vieles mehr. Wäre es nicht schön, die Vorteile von DOS (einfaches System, schnelle Ladezeit) mit dem Komfort (für Programmierer) von Windows zu verbinden?

Das ist nun zum Teil möglich, mit einem DOS/Win32API–Extender. Dieser Extender nennt sich HX DOS–Extender und kann als Freeware–Anwendung (inkl. Quellcode) von `http://www.japheth.de/HX.html` geladen werden.

Was ist nun dieser HX DOS–Extender? Klassische DOS–Extender sind schon seit dem Erscheinen des i386–Prozessors bekannt (1986 [1]), diese alten Extender (Phar Lap, DOS/4GW) erlaubten es DOS–Programmen die “Proteced Mode” des i386–Prozessors zu benutzen. Dies ermöglichte es DOS–Programmen, den erweiterten Speicherbereich der i386–Prozessoren zu nutzen. Der HX DOS–Extender implementiert zusätzlich einen größeren Teil der WIN32–API (Konsole IO, Memory Management, Threads, Netzwerk etc.). Weiterhin werden einige Windows–GUI–APIs unter DOS nachgebildet (DirectDraw, OpenGL). Mit dem HX DOS–Extender ist es möglich, Windows–32Bit–Konsolen–Programme und einige einfache GUI–Programme unter DOS auszuführen. Mit Forth–Systemen für Windows 32Bit ist es somit möglich, Programme zu erstellen, die sowohl unter DOS (mit dem HX–Extender), unter Windows 32Bit (Windows 95, 2000, XP, Vista, Windows 7) und auch unter Unix (mit WINE [2]) ausführbar sind. Neben Forth ermöglicht der HX–Extender es auch anderen Windows 32Bit Programmiersprachen, unter DOS ausgeführt zu werden (Rexx, Ruby, ja sogar Java und .NET/Mono).

Die Installation des HX DOS–Extenders ist denkbar einfach. Die ZIP–Dateien des HX DOS–Extenders werden auf der Festplatte entpackt (z. B. nach c:\HX), das Verzeichnis mit den ausführbaren HX–Programmen wird in den DOS–Suchpfad aufgenommen (z. B. C:\HX\BIN) und das HX Loader TSR (HXLDR32.EXE) wird in der AUTOEXEC.BAT in den Speicher geladen. Das HX Loader TSR klinkt sich in die Routine zum Laden und Ausführen von Programmen ein, erkennt Windows–32Bit–Programme und übernimmt dann das Laden anstelle des DOS–Systems.

Folgende (Windows–)Forth–Systeme habe ich (exemplarisch) mit dem HX DOS–Extender unter FreeDOS [3] getestet:

GNU Forth (GForth) Version 0.7.0 [4]: GForth benötigt Unterstützung für lange Dateinamen unter DOS, es muss “doslfn” geladen sein. Dann lässt sich gforth.exe unter DOS starten. Laden von Quellcode–Dateien funktioniert. Aus einem mir nicht erklärlichen Grund wird bei jeder Texteingabe/Ausgabe auf die Festplatte zugegriffen, was sich bei größerer Textausgabe bemerkbar macht (words).

BigForth Version 2.3.1 [5]: Die Datei “bigforth.exe” lädt, gibt das Begrüßungsbanner aus und hängt sich dann aber auf. Der Kernel “forthk~1.exe” lässt sich starten und dann auch benutzen. Allerdings lassen sich keine Quelldateien hinzuladen, es kommt die Fehlermeldung “libc.so.0 library not found”. Ein Artefakt der Linux/Unix–Abstammung dieser BigForth–Version.

4th [6] Version 4.5d: Das 4th–System für Windows–32Bit lässt sich über die Datei 4th.exe starten. Die Beispielprogramme lassen sich über 4th cx <quelldatei.4th> laden und ausführen. Ich habe einige der Beispielprogramme ausgeführt und konnte keine Fehler feststellen.

SP–Forth Version 4.19 [7]: SP–Forth ist ein Open–Source–Forth–System für Windows und Linux aus Russland. Auch SP–Forth für Windows läuft ohne Probleme. Dass man mit dem HX DOS–Extender die DOS–Beschränkungen hinter sich gelassen hat, merkt man am Beispiel “benchmark.f” im SP–Forth: Dieser Benchmark legt ein über 1MB großes Array im Speicher an und testet auch die Geschwindigkeit einiger Win32–API–Aufrufe. Der Benchmark läuft mit dem HX DOS–Extender ohne Probleme. Fertige Programme können per “SAVE” als Win32–EXE–Dateien gespeichert und unter DOS (mit HX), Windows und Linux (mit WINE) ausgeführt werden.

Fazit: Der HX DOS–Extender ermöglicht es, Forth–Programme und –Systeme unter DOS auszuführen, ohne auf Windows–Annehmlichkeiten wie 4GB Speicherzugriff, Netzwerkstack und natives Multithreading zu verzichten. Nur “Fenster” gibt es nicht, aber die vermisse jedenfalls ich nicht immer.

In der nächsten Ausgabe zeige ich Euch, wie man ein eigenes Linux so anpasst, das es (fast) so schnell in ein Forth–System bootet wie DOS. Bis dahin, viel Spaß mit Forth und dem HX DOS–Extender.

Die Links zum Artikel finden sich auf Seite 13

Forth von der Pike auf — Teil 12

Ron Minke

Die mit freundlicher Genehmigung der HCC–Forth–gebruikersgroep in der Vierten Dimension in einer Übersetzung von Fred Behringer wiedergegebene achtteilige Artikelserie erschien ursprünglich in den Jahren 2004 und 2005 in der Zeitschrift *Vijgeblaadje* unserer niederländischen Forth–Freunde. Der Autor arbeitet fleißig an seinem Projekt weiter: Im Vijgeblaadje waren Teil 9 und 10 der Serie (deutsche Übersetzung in den VD-Heften 3–4/2007 und 2/2008) und dann Teil 11 (Übersetzung in VD 1/2009) erschienen. Auch die heutige Übersetzung (Teil 12) aus dem Niederländischen (aus dem Vijgeblaadje 74) stammt von Fred Behringer.

Hier kommt nun also Teil 12 der Wiedergabe des Versuchs, ein AVR-Forth–System mit der Voraussetzung *from scratch* zu erstellen. Dieser Teil handelt von den Überlegungen zur Einteilung des RAM–Speichers.

Testen, Testen

Im letzten Teil meiner Serie *Forth von der Pike auf* habe ich über die Einteilung der Bytes bei der Stack–Reihenfolge gesprochen. Nach einigem Hin– und Herschieben der Festlegungen hat sich nun eine stabile Situation entwickelt, die auch dem ANSI–Standard genügt (siehe ANSI–Dokument, Punkt 3.1.4). Die Einteilung derselben Bytes im Dictionary im RAM–Speicher (Man erinnere sich: Diese Forth–Version kopiert sich selbst ins RAM) ist aber eine ganz andere Sache. Während des Testens des Forth–Kernels mit einer angepassten Version der Testsuite von John Hayes stellte es sich heraus, dass mit der Ausführung der Worte `2@` und `2!` etwas schieflief.

Die jetzige Situation

Die Einsichten aus Teil 11 der vorliegenden Serie wurden eingearbeitet. Im Forth–Kernel (im Flash–Speicher) wurde das Wort `LIT`, das das Einholen einer Zahl aus dem Dictionary besorgt, definiert als:

```
0630:
LIT:
    83          Laenge des Textes 'LIT' mit msb=1
    4C          'L'
    49          'I'
    D4          'T' mit msb=1
    0620        Link zum vorigen Wort
    0200        Pointer zu Code_Lit:
                        Maschinencode fuer LIT
```

Der zugehörige Maschinencode:

```
0200:
Code_Lit:
 Ld    R16,X+ ; get lo value from RAM dictionary
 Ld    R17,X+ ; get hi value,
              ; auto update IP to next word

 St    -Y,R16 ; store lo value on data stack
 St    -Y,R17 ; store hi value

 Next
```

An Platz 730 im Dictionary (im RAM) steht ein Stückchen Forth, das eine Zahl mit dem Wert `ABCD` auf den Datenstack legt:

```
0730:
    .dw     xxxx
    .dw     yyyy
    .dw     LIT
    .dw     0xABCD
    .dw     zzzz
```

Ein Hexdump dieses Stückchen RAMs liefert:

```
0730:  xx xx yy yy 30 06 CD AB zz zz .......
```

Der Maschinencode von `LIT` holt den Wert `ABCD` aus dem RAM und verfrachtet ihn auf den Stack. Das ist in Abbildung 1 auf der gegenüberliegenden Seite zu sehen.

Bis zu diesem Punkt ist nichts Besonderes zu bemerken. Forth arbeitet so, wie man es von ihm erwartet.

Inhalt des RAMs

Wenn wir uns den Inhalt des RAMs näher betrachten, fällt auf, dass der Zahlenwert `ABCD` *von hinten her* in den Speicher gelegt wurde. Das Gleiche gilt für den Speicherplatz des Wortes LIT (=0630). Für die Arbeit von Forth macht das natürlich nicht das Geringste aus; die zugehörigen Maschinencode–Abschnitte verarbeiten die Information korrekt. Doch hinterlässt das Ganze ein Gefühl der Unzufriedenheit. Ein Test mit einer *Double*–Zahl im Assembler–Quelltext zeigt:

```
    .org    0x410
    .dd     0x12345678
```

Dies wird vom Atmel–Assembler in den Flash–Speicher übersetzt zu:

```
0410:
    5678
    1234
```

Nach dem Kopieren ins RAM liefert ein Hexdump aus diesem Wert:

```
0640:  78 56 34 12 .......
```

Auch hier steht diese Zahl nach unserem Gefühl *verkehrt herum*. Außerdem fällt auf, dass das *high word* dieser Zahl hintendran steht. Und das entspricht nicht dem, was im ANSI–Standard empfohlen wird.

```
Code_Lit (  ---  n )                                  tmp.  Daten-
                                                      AVR-  Stack-
      Input            Output                         Reg.  Adresse

SP -> 0                0                              ---   0x100
      1                1   n unter. Byte = CD         R16   0x0FF
      2          SP -> 2   n oberes Byte = AB         R17   0x0FE
      3                3                                    0x0FD
```

Abbildung 1: Arbeitsweise von LIT

Die Testsuite von John Hayes

Die Vermutung, dass da im Forth–Kernel doch noch etwas nicht in Ordnung ist, wird von einem Abschnitt aus der Testsuite von John Hayes bestätigt. Beim Testen der Worte `2@` und `2!` tritt die Fehlermeldung auf, dass da mit den Resultaten auf dem Datenstack etwas nicht stimmt.

Der ursprüngliche Code im Forth–Kernel für das Wort `2@` ist (unter Hinzunahme der Zahlenwerte aus dem RAM):

```
;   2@              Replace the 16-bit address on top of
;                   the data stack with the 32-bit
;   nucleus         contents d of that memory address.
;
;                   addr   ---   d
;
;          Input                Output
;
;          0                    0
;          1 lo byte addr = 40  1   lo byte lo word d = 78
;  Dsp --> 2 hi byte addr = 06  2   hi byte lo word d = 56
;          3                    3   lo byte hi word d = 34
;          4             Dsp --> 4   hi byte hi word d = 12
;          5                    5

Code_TwoAt:

    Ld      ZH,Y+       ; get hi address          = 06
    Ld      ZL,Y+       ; get lo address          = 40

    Ld      R18,Z+      ; get lo value lo word    = 78
    Ld      R19,Z+      ; get hi value lo word    = 56

    Ld      R16,Z+      ; get lo value hi word    = 34
    Ld      R17,Z+      ; get hi value hi word    = 12

    St      -Y,R18      ; put on data stack
    St      -Y,R19

    St      -Y,R16
    St      -Y,R17

    Next
```

Wenn wir mit diesem Maschinencode den Zahlenwert von Adresse `640` holen, wird das in der richtigen Reihenfolge auf den Datenstack gelegt.

Jedoch ... im ANSI–Standard steht bei der Besprechung des Wortes 2@ (Punkt 6.1.0350) das Folgende: „... Das Wort 2@ entspricht der Hintereinander–Ausführung der folgenden Worte“:

```
DUP CELL+ @ SWAP @
```

Wenn wir das mit demselben Zahlenwert `12345678` auf Adresse `640` im RAM durchführen, kommt etwas ganz anderes dabei heraus. Machen wir das mal schnell:

```
Start           Stack

---             0640
DUP             0640  0640
CELL+           0640  0642
@               0640  1234
SWAP            1234  0640
@               1234  5678
```

Wenn wir uns in Erinnerung rufen, dass (nach ANSI) der Wert auf der oberen Stackposition das MSD einer Zahl sein muss, dann sehen wir, dass das hier nicht der Fall ist. Der Test von John Hayes hat mit seinem Ergebnis Recht: Die bisherige Implementation des Wortes `2@` liefert nicht das richtige Resultat.

Und wie nun weiter?

Um das alles dem ANSI–Standard genügen zu lassen, müssen offenbar Worte und müssen Bytes vertauscht werden. Für den Maschinencode von LIT ist das simpel: Wir holen den Wert aus dem RAM *andersherum* herein. Aber damit haben wir das Problem noch nicht bei der Wurzel gepackt. Das wirkliche Problem ist die Reihenfolge der Bytes im Flash–Speicher bei den `.dw`–Statements.

Ein Standard–Aufruf zum Setzen eines Zahlenwertes ins Flash

```
    .org    0x540
    .dw     0xABCD
```

liefert nach dem Assemblieren und dem Kopieren ins RAM

```
0540:  CD AB .......
```

Aber eigentlich wollen wir, dass dieser Wert hier erscheint als

```
0540:  AB CD .......
```

Die Lösung dieses Problems ist Gott sei Dank einfach: Es muss ein Macro–Befehl definiert und dann auf das Ganze angewandt werden, der beim Assemblieren die Reihenfolge der Bytes umdreht. Diesem Macro geben wir den Namen defwrd.

```
.macro  defwrd
    .db     high(@0), low(@0)
    .endmacro
```

Der Aufruf der Testzahl `ABCD` liefert dann

```
    .org     0x540
    defwrd   0xABCD
```

und nach dem Assemblieren und und dem Kopieren ins RAM erhalten wir

```
0540:  AB CD .......
```

Diese Abänderung hat auf den Assembler–Quelltext einen ziemlich großen Einfluss! Zum Glück hat der Editor eine *global replace*–Funktion, die das bewältigt: Alle `.dw` in `defwrd` umwandeln.

Was uns noch zu tun bleibt, ist das Anpassen des Maschinencodes für das Wort `2@`. Wie man sieht, werden die Register in einer anderen Reihenfolge angesprochen.

```
Code_TwoAt:

    Ld      ZH,Y+       ; get hi address
    Ld      ZL,Y+       ; get lo address

    Ld      R17,Z+      ; get hi value hi word
    Ld      R16,Z+      ; get lo value hi word

    Ld      R19,Z+      ; get hi value lo word
    Ld      R18,Z+      ; get lo value lo word

    St      -Y,R18      ; put on data stack
    St      -Y,R19
    St      -Y,R16
    St      -Y,R17

    Next
```

Nicht nur der Maschinencode für das Wort `2@` muss in Angriff genommen werden, alle Worte, die etwas mit dem Zugang zum RAM–Speicher zu tun haben, müssen verändert werden. Es geht um die Worte

```
LIT EXECUTE BRANCH ! @ +! 2@ 2! HERE (FIND)
RP! SP! DoConstant DoUser ((EMIT)) NEXT
```

Nach dieser doch recht aufwändigen Aktion habe ich den kompletten Quelltext durch den Assembler gejagt und ins Flash gesetzt, die Speisespannung an die Testplatine gelegt und ... es funktioniert!

Schnell noch den Test von John Hayes durchgeführt und wie zu erwarten: Keine Fehler!

Uff ... eine ganze Menge Arbeit diesmal wieder und das wegen eines so *einfachen Fehlers*. Damit haben wir jedoch nochmal einen weiteren Schritt in Richtung auf ein *Forth–von–der Pike–auf*–System getan!

Die Testsuite von John Hayes findet sich unter:
`ftp://ftp.taygeta.com/pub/Forth/Applications/ANS/core.fr`

Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 72, Februar 2009

Das Vijgeblaadje erscheint um den Ersten eines jeden geraden Monats herum. Beiträge bis 3 Wochen vor der jeweiligen Nummer an: f.l.van.der.markt@kader.hcc.nl

Und es geht weiter: Sehr zu meiner (des Rezensenten) Überraschung und Freude ist wieder eine Vijgeblaadje-Ausgabe erschienen. Ob und wie es weitergeht, kann man jederzeit unter `http://www.forth.hccnet.nl/nieuws` erfahren.

SD–kaart besturing (Teil 3a von 3) —

Willem Ouwerkerk

SD–Karten bis zu 2 GB, so der Autor, sind normalerweise im FAT16–Dateisystem formatiert. Es wird über Cluster gesprochen. Es folgen ein paar Worte über Vor– und Nachteile von FAT16 und FAT32. Dann ist die Rede vom Master–Boot–Record (MBR), von reservierten Sektoren, vom BIOS–Parameter–Block, von den FAT–Tabellen 1 und 2, vom Root–Directory und von den eigentlichen Daten (Dateien). Im MBR steht die Partitions–Tabelle, aus der die SD–Karte alle wichtigen Daten entnimmt. Dann wird die Arbeitsweise von FAT (FAT16) erklärt, der Umgang mit langen Dateinamen wird erwähnt, und das simple Verfahren, Dateien beim *Löschen* nur durch ein vorangesetztes E5h zu kennzeichnen. Dann kommt die Software zur Sprache, die benötigt wird, um auf der SD–Karte ein DOS–FAT–Dateisystem aufzubauen, gewissermaßen ein Forth–DOS.

Dann werden ein paar wichtige Hilfsroutinen erläutert:

```
MOUNT
@NEXT-CLUSTER
NEXT-SECTOR
VALID-DIRENTRY?
(EMPTY-DIRENTRY?
>DOS
(FIND-DOSNAME
LOAD-FILE
CD
```

Und schließlich folgt das Forth–Programm (übersichtlich angeordnet und gut kommentiert). Das Programm wird im Schluss–Teil 3b (in einem der nächsten Hefte) fortgesetzt. Es geht dann noch um die Schreib-Routinen und um ein paar Hilfsworte, wie `MAKE-DIR` und `DELETE-FILE`.

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 73, April 2009

Update F51 project, verslag werkgroep — Willem Ouwerkerk, Marc Hartjes

Ein Bericht der Arbeitsgruppe F51, ein Projekt, das offensichtlich mit Willems SD–Karten–Ansteuerung einhergeht und nun seiner Vollendung zustrebt. Berichtet wird über die Stromversorgung, den RS232–Treiber und den Prozessor AT89S8253. Bei der Stromversorgung wird ein FAN4855 eingesetzt, der aus einer Spannung von 1,0 bis 4,5 Volt problemlos stabile 3,3 oder 5,0 Volt bei 500 mA macht. Der RS232–Treiber wird in Form eines MAX3232 verwirklicht, da der Hauptprozessor mit 3,3 Volt gespeist werden soll, um eine einfache SD–Karten–Anbindung zu bekommen. Sodann wird über verschiedene Schwierigkeiten berichtet, die Platine zum Laufen zu bringen. Der Arbeitsgruppen–Abend dauerte von 20.30 bis 24.30 Uhr. *Demnächst* (der Rezensent: im nächsten Heft?) wird mehr über das Projekt zu hören sein.

Eerste ervaringen met een Arduino bord — Frans van der Markt

Frans nimmt Bezug auf einen Vortrag von Paul Wiegmans vom Oktober letzten Jahres. Die Arduino–Platine kann per USB angesteuert werden. Das bewog Frans, sich eine solche Platine anzuschaffen. Im Elektorheft vom März 2009 stand ein für absolute Änfänger bestimmter Artikel über den Arduino. Frans bestellte bei Turtle Creations, einem holländischen Lieferanten. Als Prozessor saß ein ATMega328 auf der Platine. Dokumentation war kaum nötig. Das Forum *www.arduino.cc* liefert alle Informationen. Frans erwähnt auch die Existenz von AMForth für AVR–Prozessoren (`amforth.sourceforge.net`) und verspricht, sich weiter mit der Frage einer Implementation von Forth auf dem Arduino zu beschäftigen.

Gevonden in de media — Ron Minke

Ron macht auf einen Bericht im Aprilheft der Zeitschrift Elektor aufmerksam. Es geht um ein 2,4–GHz–Transceiver–Development–Kit. Müsste doch für die Einbindung von Forth geeignet sein – sagt Ron. Genaueres unter `www.decibit.com`.

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 74, Juni 2009

Forth vanaf de grond 12 — Ron Minke

Der Hauptteil der diesmaligen Ausgabe des Vijgeblaadjes wird von Teil 12 von Rons Serie über *Forth von der Pike auf* eingenommen. Ron hat beim abschließenden Hayes–Test (siehe Teil 11) entdeckt, dass die Worte `2@` und `2!` immer noch Schwierigkeiten machen: Es geht um die ANS–konforme Reihenfolge von Lo–Word und Hi–Word. Alle 16 Worte, die den Verkehr mit dem RAM regeln, sind betroffen. Er löst das Problem über einen Makro–Befehl, der beim Assemblieren die richtige Reihenfolge herstellt. Ich (der Rezensent) sitze an der Übersetzung des gesamten Artikels für unsere deutschen VD–Leser.

Surplus: Centronics printer interface — Ron Minke

Von vor 15 Jahren: Etwa 10 Drucker–Interface–Platinen sind übrig: Drucker mit Centronics–Parallel seriell anschließen. Mit RAM, EPROM, einem 8052–Prozessor und — Forth. Näheres auf der Website unserer niederländischen Forth–Freunde (z. B. über `www.forth-ev.de`).

Links zum Artikel *Das Beste beider Welten* von Seite 9:

Links

[1] `http://en.wikipedia.org/wiki/DOS_extender`
[2] `http://en.wikipedia.org/wiki/Wine_(software)`
[3] `http://en.wikipedia.org/wiki/Freedos`
[4] `http://www.complang.tuwien.ac.at/forth/gforth/`
[5] `http://www.jwdt.com/~paysan/bigforth.html`
[6] `http://www.xs4all.nl/~thebeez/4tH/`
[7] `http://spf.sourceforge.net/`

Win32Forth — Wie ruft man damit Routinen von Windows auf?

Alex Mcdonald (Ins Deutsche übertragen von Michael Kalus)

Es macht Mühe, sich in all die Routinen einzufuchsen, die Windows bietet. Aber es ist auch kein Buch mit sieben Siegeln. Wer sich also da herantasten will, tut gut daran, ein Werkzeug zu nehmen, das erste Erfolgserlebnisse sofort ermöglicht. Und einen dann auf dem Weg zur vollen Kontrolle über die CALLs auch nicht im Stich lässt. Mein Eindruck ist, dass Win32Forth (W32F) gut vermittelt, wie das Betriebssystem Windows für eigene Programme genutzt werden kann.

Dieses Forth von Tom Zimmer, das er der Öffentlichkeit schenkte und bis 2002 noch selbst betreut hat, wird seither von einer aktiven Gruppe von Benutzern weitergepflegt[1]. So ist das W32F inzwischen zu einem erstaunlich umfangreichen System geworden. Neben der klassischen Forth–Konsole bietet es eine komfortable integrierte Entwicklungsumgebung (IDE) an, deren wichtigste Features sind: Syntax highlighting, Projektmanagement, Dialog Editor, Debugger. Und mit der ForthForm lassen sich inzwischen Ein– und Ausgabe–Fenster einfach zusammenklicken. Dabei ist ForthForm inzwischen ebenso Bestandteil der IDE geworden wie der Projekt–Manager. Die Standalone–Versionen der beiden Programme werden von der Gruppe nicht mehr weitergepflegt. Dies gilt auch für Toms Editor WinView (bzw. WinEd).

Um Berührungsängste abzubauen und einen Anfang zu finden, ist es vielleicht hilfreich, sich einmal das Grundlegende zu den DLLs aus der umfangreichen Dokumentation des W32F[2] heraus zu picken. Drum habe ich es ins Deutsche übersetzt. Die Entwicklergruppe war damit einverstanden, doch bat Dirk Busch um den Hinweis, dass einiges in der Dokumentation inzwischen von der aktuellen Entwicklung des W32F überholt worden ist. Fragen dazu werden gern aus der Gruppe beantwortet. Hingewiesen sei auch darauf, dass derzeit eine Betaversion der nächsten W32F Version 6.13 unter: `http://www.schneider-busch.de/dirk/forth/download/w32f61301.exe` zum Download bereitsteht. Ab dieser Version wird es ein nützliches Feature in Bezug auf DLLs geben. Im Menü *Tools* der Konsole findet sich der Eintrag *List DLL exports*, mit dem man sich die Namen der Funktionen, die von einer DLL exportiert werden, anzeigen lassen kann. (mka)

Grundlegendes

Es ist einfach, mit Win32Forth Windows–Aufrufe zu machen — wenn man weiß, wie. Dieses Tutorial erklärt, welche Schlüsselworte für die Verbindung zu den DLLs benutzt werden, und gibt einige einfache Beispiele, um das zu veranschaulichen.

Dabei wird angenommen, dass du eine Version des Win32Forth benutzt, die CALL– und PROC–Instruktionen bereits im Forthkern hat. Tippe .LIBS, um das zu prüfen. Die Bedeutung der Spalten wird weiter unten im Text noch erklärt werden; die Adressen und die Liste selbst mag auf deiner Maschine anders sein, das variiert je nach Betriebssystem und Win32Forth–Version.

```
.libs
Location Handle   Type Name
-------- -------- ---- -----------
000450D8 71710000 APP  COMCTL32.DLL
0002C91C 76B30000 APP  COMDLG32.DLL
0001794C 10000000 APP  WINCON.DLL
00017934 782F0000 APP  SHELL32.DLL
0001791C 7C2D0000 APP  ADVAPI32.DLL
00017908 77F40000 APP  GDI32.DLL
00013490 00D20000 APP  PRINT.DLL
0000E6D8 63180000 APP  SHLWAPI.DLL
0000BFA0 7C570000 KERN KERNEL32.DLL
0000B0FC 77E10000 APP  USER32.DLL
0000A970 00D40000 APP  CONSOLE.DLL
```

Grundlagen der DLLs

DLLs (Dynamic Link Libraries) sind auch nur Programme auf deinem PC wie alle anderen, haben jedoch einige wenige grundlegende Unterschiede.

DLLs haben gewöhnlich kein sichtbares Interface, also kein *Fenster*. Und zudem können sich ganz verschiedene Prozesse eine DLL teilen. So liegt gewöhnlich nur eine Kopie davon im Speicher, ganz egal, wie viele Programme die dann benutzen. DLLs stellen vorgefertigte Funktionen zur Verfügung, die dann von allen anderen Programmen benutzt werden können. Und um sie effektiv zu verwenden, braucht man deren Dokumentation.

Win32Forth–Worte dafür

Es braucht nur wenige Forthworte, um einen Aufruf externer DLLs zu machen:

- Debugging words: `.PROCS`, `.LIBS`
- Defining words: `PROC`, `WINLIBRARY`
- Calling words: `CALL`, `REL>ABS`, `ABS>REL`
- Naming words: `AS`

Schritt 1. Finde die Dokumentation der DLL

Ohne solch ein Dokument ist es weder möglich, herauszufinden welche Namen die Funktionen einer DLL haben, noch welche Parameter übermittelt werden müssen, noch was zurückkommt. Wertvolle Dokumentationen zu den grundlegenden DLLs findet man bei

msdn.microsoft.com, aber es gibt auch eine Menge weiterer Anbieter dafür. Für speziellere DLLs braucht man die von denen mitgelieferten Dokumente.

Schritt 2. Melde die DLL bei Win32Forth an

Wird dem Win32Forth der Name einer DLL mitgeteilt, lädt es diese. Die Anweisung WINLIBRARY KERNEL32.DLL ermöglicht es dem W32F, diese DLL zu laden, damit dann deren Routinen benutzt werden können. Der Suchpfad für die DLLs variiert je nach Betriebssystem (95/98/ME/NT/2000/XP). Wird einfach nur der DLL-Name angegeben, muss die DLL im Systemverzeichnis liegen, oder in dem Verzeichnis, von dem aus W32F selbst aufgerufen worden ist. Die meisten DLLs werden auf diese Weise dann auch gefunden, und es wäre ungewöhnlich, wenn Pfadangaben gemacht werden müssten. WINLIBRARY kann auch mehrfach aufgerufen werden, es wird trotzdem immer nur eine einzige Kopie davon im Speicher sein. Mit .LIBS erfährt man, was schon alles geladen wurde:

```
Location Handle   Type Name
-------- -------- ---- -----------
0003EA5C -noload- APP  TESTIT.DLL
00014C08 10000000 APP  WINCON.DLL
. . .
```

Dieses Wort wird gewöhnlich nur von der Kommandozeile einer Konsole aus benutzt. Im Beispiel sieht man, dass TESTIT.DLL zwar spezifiziert, aber noch nicht geladen wurde. Falls du Fehlermeldungen beim Aufruf von Routinen hast, die W32F eigentlich schon kennen sollte, schau bei der Adresse des *handle* nach. Wird dort noch `-noload-` angegeben, konnte W32F die Bibliothek für diese Routine nicht finden.

Schritt 3. Melde deren Prozeduren[3] bei Win32Forth an

Das Forthwort `PROC` wird verwendet, um die verschiedenen Prozeduren einer DLL (functions) vom W32F aus zu nutzen. In den Dokumenten der DLLs finden sich beispielsweise solche Angaben:

AllocConsole

Die Funktion `AllocConsole` startet eine neue Konsole für Programmaufrufe. (to allocate = Platz zuweisen)
`BOOL AllocConsole(void);`

Parameter

Die Funktion nimmt keine Parameter an.

Rückgabewerte

Wenn die Funktion erfolgreich beendet wurde, ist der zurückgegebene Wert verschieden von null (nonzero). Sollte die Funktion hingegen scheitern, ist der Wert null.

Diese Funktion legt man im W32F nun so an:

```
0 PROC AllocConsole
```

d. h., die Funktion nimmt 0 Parameter und heißt `AllocConsole`. Beachte dabei unbedingt die Groß– und Kleinschreibweise! Du MUSST die Namen so wie in deren Dokumention gezeigt eintippen. Obschon gerade in diesem Beispiel die Zahl nur symbolischen Wert hat und lediglich für die Dokumentation gedacht ist, denn ihr Wert kann 0 bis 127 sein, darf sie doch nicht negativ sein, weil damit spezielle Prozeduren des Systems gekennzeichnet werden. (Und wie schließt man die Konsole wieder? Lies aufmerksam weiter und du wirst es herausfinden.)

Dieser Schritt ist eigentlich optional (und aus didaktischen Gründen hier angeführt) — man muss die PROC–Anweisungen eigentlich nicht ausdrücklich angeben, weil W32F während der CALL–Anweisung alle Prozeduren dynamisch selbst erzeugt. Die Anzahl der Prozeduren ist so aber auf die maximal dynamisch erzeugbaren begrenzt, das sind derzeit so um die 100 bis 150 Stück. Mit .PROCS sieht man welche Prozeduren schon erzeugt worden sind, und man kann die Ausgabe auch eingrenzen auf einen Namensteil (hierbei ist die Groß– oder Kleinschreibung mal ohne Bedeutung):

```
.PROCS ALLOC
Location  ProcName          Prm  ProcEP    LibName
--------  --------          ---  --------  --------
0003EA5C  AllocConsole      0
00012814  GlobalLock             77E977E4  KERNEL32.DLL
000127F4  GlobalAlloc            77E96646  KERNEL32.DLL
0000B830  HeapReAlloc       4
0000B7F8  HeapAlloc         3    77FCB10F  KERNEL32.DLL
Allocated Space:   12,288 bytes
Free Space Left:    6,997 bytes
Displayed 5 of 184 procedures defined
```

Schritt 4. Funktion aufrufen

Da nun die DLL (library) angelegt ist, und optional auch deren Prozedur, können wir mit CALL AllocConsole ihre Funktionen ausführen — nochmal: Die Schreibweise ist von Bedeutung und muss der Doku genau entsprechen! Nach diesem CALL sollte man eine frisch geöffnete DOS–Konsole sehen und eine 0 auf dem Stack vorfinden.

Weitere Eigenschaften von DLLs

Die Abfolge der Parameter

Forth übermittelt alle Parameter an die Windows–Funktionen mit dem Datenstack. Dort liegen sie in umgekehrter Abfolge. Also, wenn die Dokumentation fordert, dass `BOOL Function ( A, B, C )` anzugeben ist, schreiben wir in Forth `C B A CALL Function`. Diese Umkehrung ist grundsätzlich nötig, um die Forth–Aufrufe mittels `CALL` laufen zu lassen.

Prozeduren benennen

Manche Funktionen im Windows haben bis zu drei Namen. Zum Beispiel gibt es da `CharUpperBuff`, `CharUpperBuffA` und `CharUpperBuffW` nebeneinander. Tatsächlich aber sind es nur zwei verschiedene Routinen. Schau dir die folgende Liste an, die mit `.PROCS` erstellt wurde[4]:

```
Location  ProcName             Prm  ProcEP    LibName
--------  --------             ---  --------  --------
0003EA98  CharUpperBuffW       2    77E1236F  USER32.DLL
0003EA74  CharUpperBuffA       2    77E1263D  USER32.DLL
00003350  CharUpperBuff        2    77E1263D  USER32.DLL
```

In der Spalte `ProcEP` steht die Startadresse (entrypoint) einer Prozedur. Wie man sieht, rufen `CharUpperBuff` und `CharUpperBuffA` dieselbe Funktion auf, nämlich die für ASCII–Zeichen genutzte, bei der jedes Zeichen durch 8 Bit dargestellt wird, also eine 1–Byte–Funktion. W hingegen ist die Funktion für den UNICODE, also die 2–Bytes–Funktion.

Gewöhnlich wirst du die Ascii–Funktion benutzen, da W32F ein 1–Byte–pro–Zeichen–System ist. Von jeder Funktion, die sowohl in der A– wie in der W–Variante vorkommt, lädt W32F automatisch nur die A–Version, ohne dass du das extra aussuchen musst. Falls du aber wirklich die W–Funktion haben willst, musst du sie ausdrücklich aufrufen. Also z.B. so:

```
2 PROC CharUpperBuffW
```

Die zurückgegebenen Werte (return values)

Generell lässt jede Funktion immer nur einen Wert auf dem Stack zurück. Weil die meisten Funktionen in C oder C++ geschrieben sind, und daher multiple Werte gar nicht zurückgeben können, muss man andere Techniken benutzen, soll eine Funktion mehrere Werte zurückgeben.

`BOOL` ist ein Beispiel für solch einen Rückgabewert. Dabei steht im Windows die 0 für *wahr* (oder OK) und die 1 für *falsch* (nicht–OK). Das ist anders herum als im Forth. Nimm `0=`, um `BOOL` auszuwerten und zum forthigen `TRUE` oder `FALSE` zu kommen:

```
CALL AllocConsole 0= IF ( alles ist OK ..)
```

Oder du wertest das so aus:

```
CALL FreeConsole ABORT" FreeConsole failed"
```

`INT`, `HANDLE`, usw. ergeben 32–Bit–Werte (unsigned or signed 32 bit) auf dem Stack. Verwende sie wie sonst im Forth auch. Auch `VOID` gibt einen 32–Bit–Wert auf den Stack, der hat aber keine Bedeutung (void = leer, Lücke) und kann einfach mittels `DROP` verworfen werden.

Parameter von Funktionen interpretieren

Leider kommen die Parameter von Funktionen in unterschiedlichen Geschmacksrichtungen daher. So muss neben deren Abfolge auch deren Datentyp stimmen, damit der `CALL` funktioniert. Ein Beispiel:

```
BOOL ReadFile( HANDLE hFile, // handle to file LPVOID
lpBuffer,                    // data buffer
DWORD nNumberOfBytesToRead,  // number of bytes to read
LPDWORD lpNumberOfBytesRead, // number of bytes read
LPOVERLAPPED lpOverlapped    // overlapped buffer
   );
```

wird im W32F zu:

```
0 VALUE HANDLE
CREATE BUFFER 255 ALLOT
20 VALUE BYTESTOREAD
VARIABLE BYTESREAD
: MYFUNC
  0               \ null address for overlapped buffer
  BYTESREAD       \ bytes read
  BYTESTOREAD     \ bytes to read
  BUFFER          \ address of buffer
  HANDLE          \ value of handle
  CALL ReadFile ;
```

Die Parameter–Typen `HANDLE`, `BYTE`, `WORD`, `DWORD` liefern einfach einen Stackwert. Ihre Namen werden also interpretiert. Falls der Wert in `HANDLE` 13 wäre, käme die 13 auf den Stack.

Die `LPxxx`–Typen (ohne `STR`) hingegen sind Zeiger auf Datenbereiche. Es klappt also für gewöhnlich nicht, wenn man `VALUE` <name> benutzt, um `LPxxx`–Parameter zu definieren. Besonders wenn die Funktion einen Wert zurückliefert geht das schief, wie im Fall von `BYTESREAD`. Benutze lieber `VARIABLE`, weil dann die Adresse auf den Stack kommt.

Und die `LPxSTR`–Typen behandeln Zeichenketten, die mit einer Null enden (Null (0) terminated strings). Man benutzt sie z. B. so:

```
: UPPERCASE ( caddr -- caddr )
    DUP COUNT SWAP
    call CharUpperBuff DROP ;
```

Es wird also die einfache Adresse einer Zeichenkette mit vorangehendem Zähler (counted string) in die Startadresse der Zeichen und die Länge der Kette umgewandelt, und deren Abfolge wird dann umgekehrt (len addr). Das ist nötig, weil der Windows–Funktionsaufruf zuerst den Zeiger auf die Zeichenkette und dann erst deren Länge erwartet. Zurück kommt dann wieder ein Wert, in diesem Fall ein n, an dem wir aber nicht weiter interessiert sind. Im Übrigen ist dieses Beispiel nicht einmal ein richtiger null–begrenzter Zeichenketten–Aufruf. Aber das hier ist einer:

```
int lstrlen(LPCTSTR lpString);
```

Die Funktion zählt die Anzahl der Zeichen in der Kette. So ergibt `Z" ABCDEFG" CALL lstrlen` eine 7 auf dem Stack, das ist die Länge der Zeichenkette. Folgende Zeichenketten–Typen des W32F sind null–begrenzt und sicher verwendbar auch in solchen Windows–Aufrufen:

`C"` — aber nur wenn der Zählwert berücksichtigt wird, geht es gut. Also immer so verwenden: `C" ABCDEFG" COUNT DROP` Nun steht die Adresse auf A, also auf dem Anfang der Zeichenkette, und nicht mehr auf dem Zähler.

`S" ABCDEFG"` legt (addr len) auf dem Stack ab. Hierbei zeigt die Adresse bereits richtig auf das A am Anfang der

null–begrenzten Kette. Ob len verwendet wird, hängt ab von der Funktion, die folgen soll.

`Z" ABCDEFG"` funktioniert wie `C"`, aber mit dem Unterschied, dass gar keine Längenangabe erfolgt. Daher können `Z"`–Ketten direkt benutzt werden für die Windows–Aufrufe.

Und hier noch eine Technik für den Fall, dass du deinen eigenen Puffer für eine null–begrenzte Zeichenkette erzeugen willst.

```
CREATE MYSTR 256 ALLOT  \ my string
S" ABCDEGF" MYSTR PLACE \ move string to MYSTR
MYSTR +NULL             \ add null on the end
MYSTR CALL lstrlen \ use in call.
```

AS verwenden

Gelegentlich ist es hilfreich, ein Forth–Wort zu haben, mit dem das XT einer Prozedur verwendet werden kann. Auch sind manche DLL–Namen recht lang und obskur ausgefallen, und eine bedeutsamere kürzere Benennung wäre wünschenswert. So fügt

```
1 PROC ExitThread AS EXIT-TASK
```

das Wort `EXIT-TASK` an die aktuelle Wortliste an (current wordlist). AS muss nach der Deklaration einer Prozedur stehen, Kommentare dürfen dazwischen sein.

Gefährliche Tricks

Nun folgt noch ein gefährlicher Trick. Nutze so etwas nur, wenn du DLL–Funktionen schon sicher aufrufen kannst. Nimm an, dass die Funktion `BOOL function(LPWORD param);` eine 10 auf die Adresse schreibt, die ihr übergeben worden ist. Zusätzlich gibt sie eine 1 zurück. Dann kann auch so etwas gemacht werden:

```
0 SP@ ... CALL function
```

Der Teil vor dem CALL manipuliert den Stack:
`0 | addr of previous cell |`

Nach dem Funktionsaufruf hat der Stack dann die Werte
`10 | 1`

Macht man es so, ist keine `VARIABLE` mehr nötig, um die 10 zu übergeben! Der ReadFile–Aufruf, den wir weiter oben schon mal ausprobiert hatten, kann dann auch so geschrieben werden:

```
: MYFUNC 0          \ null address for overlapped buffer
  0 SP@             \ bytes read = zukünftige 'Variable'
  BYTESTOREAD       \ bytes to read
  BUFFER            \ address of buffer
  HANDLE            \ value of handle
  CALL ReadFile ;
```

Die Funktion liefert auf diese Weise dann 2 Werte direkt auf dem Stack ab, einmal die Anzahl Bytes, die gelesen wurden, und dann oben drauf noch den Rückgabewert. Sie hat also die von `sp@` ermittelte Adresse wie eine Variable benutzt, um darin die 10 zu übergeben.

Anmerkung

Die Dokumentation so eines umfangreichen Systems wie das Win32Forth ist keine leichte Aufgabe, und die neuesten Änderungen am System werden dort nicht immer sofort eingearbeitet sein. Weil es eine Gruppenarbeit ist, war nicht immer so ganz klar, welchem Autor jeweils der Dank gebührt für einen bestimmten Beitrag zu der Dokumentation. Manches war auch von vornherein eine Gemeinschaftsarbeit. Namentlich genannte Mitarbeiter, denen ich an dieser Stelle meinen Dank und meine Anerkennung für ihre mühevolle Arbeit aussprechen möchte, sind (in alphabetischer Reihenfolge): Robert Dudley Ackerman, Ezra Boyce, Dirk Busch, Thomas Dixon, Brad Eckert, Bruno Gauthier, George Hubert, Alex McDonald, Rod Oakford, Andrew Stephenson, Jos van de Ven und noch andere.

Dirk Busch danke ich sowohl für die freundliche Hilfe bei der Durchsicht der Übersetzung als auch für die Hinweise auf die neuen Features des W32F 6.13.

Fußnoten

(1) Win32Forth *We*–Benutzergruppe:
`http://tech.groups.yahoo.com/group/win32forth/`

(2) Grundlagen der DLLs:
http://win32forth.sourceforge.net/doc/p-windlls.htm

(3) Im englischen Original werden die Begriffe procedure, function und routine synonym benutzt. In der Übersetzung werden daher ebenfalls Prozedur, Funktion oder Routine als Begriffe für dieselbe Sache benutzt.

(4) Die Liste `CharUpperBuff` usw. hat erst dann Einträge in den Spalten `ProcEP` und `LibName`, wenn die Funktionen auch einmal gelaufen sind. Bis dahin bleiben diese Spalten leer. Um das Beispiel nachvollziehen zu können, überlege, wie du einen Aufruf der Funktionen machen könntest.

Quellen

1. *Win32Forth, Calling Windows Procedures, The Basics.*,
Document `$ Id: p-windlls.htm,v 1.1 2004/12/21 00:18:57 alex_mcdonald Exp $`
auf: `http://win32forth.sourceforge.net/doc/p-index.htm`

Wurstkessel — Kryptographie in Forth

Bernd Paysan

Dieser Artikel behandelt das Thema „symmetrische Verschlüsselung“ anhand eines Beispiels. Zur symmetrischen Verschlüsselung gehört auch ein kryptographisch starker Hash und ein ebensolcher Zufallsgenerator. Und da man Programme, die man schreibt, auch testen muss, können Überlegungen zum Knacken der Verschlüsselung auch nicht fehlen.

Motivation

Auf die Idee, einen eigenen Algorithmus zur symmetrischen Verschlüsselung zu entwickeln, kam ich, als ich die Implementierung des „Whirlpool“-Algorithmus [1] angesehen habe. Whirlpool ist kein Haushaltsgerät, sondern ein relativ neuer Hash–Algorithmus. MD5 ist ja geknackt, und damit wertlos. Auch SHA–1 hat bekannte Schwächen, und wenn einmal eine Schwäche bekannt ist, dann taucht bald die nächste auf [2]. Und SHA–2 basiert auf demselben Algorithmus, ist also wohl auch nicht besser. Also müssen zumindest neue Hash–Algorithmen her.

Als Ziel habe ich mir dann gesetzt, dass der Algorithmus einfach implementiert werden kann, sowohl in Hardware als auch in Software — und da auch auf sehr einfach gestrickten CPUs — und dass er ein inhärent hohes Maß an Parallelität hat, und damit zumindest theoretisch auch ein hoher Durchsatz möglich ist. Und natürlich sollte der Algorithmus auch skalieren, d.h. es soll möglich sein, die Schlüssel- bzw. Hashlänge je nach Bedarf zu variieren.

Die aktuelle Runde der NIST, einen Kandidaten für SHA–3 zu finden, habe ich zwar verpasst. Das ist aber nicht wirklich schlimm, denn Kryptographie gedeiht nicht mit Schnellschüssen.

Kryptologische Grundlagen

In der VD 1/2005 habe ich ja schon mal über Verschlüsselung geschrieben, damals über den RSA–Algorithmus, der zum Austausch von Schlüsseln und zur digitalen Signatur verwendet wird. Damals ging es um asymmetrische Verschlüsselung, deren Hauptzweck der Austausch von Schlüsseln (Public–Key–Verfahren) und digitale Signaturen sind. Das damals war nur eine schematische Modellimplementierung, der zur Anwendung noch einige Details fehlten, neben der Bignum–Bibliothek insbesondere

- ein guter Pseudo–Zufall
- ein symmetrisches Verschlüsselungsverfahren
- und ein kryptographischer Hash

Wir werden sehen, dass die drei Verfahren auf derselben Basis aufbauen können. Gerade der Zufall ist ein wichtiges Element, denn wenn man eine wirklich gute Zufallsfolge hat, kann man mit ihr auch perfekt verschlüsseln. Der sogenannte „One–Time–Pad“ ist die einzige kryptographische Methode, die wirklich nicht brechbar ist — wenn man sie bestimmungsgemäß anwendet, d.h. den zu verschlüsselnden Text mit einer Folge von Zufallszahlen verknüpft, die mindestens so lang wie der Text ist, und genau einmal gebraucht wird. In der Praxis ist diese Methode nur in Spezialfällen brauchbar, da der Aufwand zum sicheren Austausch des Schlüssels genauso hoch ist wie der Aufwand zum geheimen Austausch einer unverschlüsselten Nachricht — und ein vorab ausgetauschter Schlüssel dann auch noch entsprechend sicher verwahrt werden muss.

Trotzdem gibt uns diese Methode ein Ziel vor: Unsere Verschlüsselung soll sich ohne Kenntnis des Schlüssels nicht von Zufall unterscheiden, und der Schlüssel selbst darf nur durch Ausprobieren gefunden werden. Etwas Vergleichbares gilt auch für einen Hash: Der Hash–Wert einer Nachricht ist zwar deterministisch durch die Nachricht bestimmt, soll aber ebenfalls wie ein völlig zufälliger Wert aussehen, der sich zwar mit jeder Änderung der Nachricht ändert, aber in einer unvorhersehbaren Weise — ändert man ein Bit in der Nachricht, so sollten sich ca. die Hälfte aller Bits des Hashes andern (und natürlich nicht vorhersehbar sein, welche Hälfte der Bits sich ändert). Aus dem Hash–Wert soll nicht auf die Nachricht zurückgeschlossen werden können; genauso wenig soll es möglich sein, durch gezielte Modifikation der Nachricht einen bestimmten Hashwert zu erzeugen.

Whirlpool verwendet zwar einen anderen theoretischen Ansatz, die Implementierung für 64–Bit–Prozessoren ist aber tabellengetrieben. Aus genau diesem tabellengetriebenen Ansatz habe ich dann meinen Algorithmus hergeleitet. Basis ist eine Funktion, die zwei 512 Bit lange Zahlen so durch den Fleischwolf dreht, dass nur noch Wurst herauskommen kann:

$$\phi(a, s, e) \rightarrow a', s'$$

wobei

a ein Akkumulator ist,

- der am Ende der Transformation als Hashwert ausgegeben wird,
- bei der symmetrischen Verschlüsselung mit dem Text verknüpft wird,
- oder als Zufallszahl dient,

s ein interner Zustand, der nicht preisgegeben werden soll, und

e eine Entropiequelle, also bei Hash und Verschlüsselung die Botschaft selbst, beim Generieren von Zufallszahlen externe, echt zufällige Ereignisse, die mit dem Pseudozufallsgenerator gemischt werden können.

Die Anforderung an ϕ ist schlicht: Sie soll sich leicht und schnell berechnen lassen, aber die Umkehrung ϕ^{-1} soll praktisch unmöglich berechenbar sein.

Algorithmus

Ich will also aus Eingangsvektoren möglichst schnell möglichst „zufällige“ Zahlen bekommen. Was liegt näher, als die Eingangsvektoren über eine Tabelle einfach auf wirklich zufällige Werte abzubilden? Ich habe mir auf `www.random.org` also 256 wirklich zufällige 64–Bit–Werte besorgt. Die Indices in diese Tabelle sind also Bytes aus meinem Akkumulator und dem internen Zustand. Diese 64–Bit–Werte verknüpfe ich mit einem XOR–Operator; damit Vertauschungen auch unterschiedliche Werte geben, rotiere ich den Wert vor der nächsten XOR–Verknüpfung um eins nach links.

Die Idee dabei ist: Indizierung von 256 zufälligen 64–Bit–Werten ergibt eben diese 256 möglichen zufälligen Werte. Verknüpfe ich zwei davon, einer um ein Bit rotiert, bekomme ich $256^2 = 64\text{k}$ Werte, zwischen denen kein erkennbarer Zusammenhang besteht. Acht Werte derart verknüpft, ergibt ca. $256^8 = 2^{64}$ verschiedene Werte, zwischen denen ebenfalls kein erkennbarer Zusammenhang besteht. Wenn die Theorie stimmt, dann sollten dabei 2^{32} Kollisionen entstanden sein, d.h., bestimmte Ausgangswerte liefern das gleiche Ergebnis. Das ist so gewünscht, das würde nämlich auch passieren, wenn man statt der Abbildung einfach 2^{64} echte Zufallszahlen ziehen würde.

Damit das Ganze noch weniger deterministisch wird, starte ich diese Verknüpfung mit einem 64–Bit–Wert aus dem Status; dabei permutiere ich die Status–Wörter in einem möglichst langen Zyklus.

Den Index in die Tabelle bekomme ich, indem ich je ein Byte aus dem Status mit einem Byte aus dem Message–Akkumulator verXODERe. Dabei transponiere ich den Status, d.h., ich schreibe die Wörter byteweise in Zeilen und laufe dann spaltenweise durch. Den Message–Akkumulator durchlaufe ich mit Primzahlschritten, jede Runde eine andere Schrittweite. Am Ende einer Runde wird der alte Status mit dem Message–Akkumulator ver-XODERt. Damit stelle ich sicher, dass selbst für den Fall einer Kollision entweder der Message–Akkumulator oder der Status einen anderen Wert enthält, also bei der nächsten Runde wieder ein anderes Ergebnis gefunden wird.

Oder in mathematische Formelsprache gekleidet:

rng_x 256 64–bittige Zufallszahlen von `www.random.org` in einer Tabelle

$rol(rng_x, 1) \oplus rng_y$ gibt 64k zufällig aussehende Zahlen

$r^j = \bigoplus_{i=0}^{7} rol(rng_{x_i^j}, i)$ gibt ca. $2^{64} - 2^{32}$ zufällig aussehende Zahlen

$x_i^j = a_{p(8j+i)} \oplus s_j^i$ Index in die Tabelle: Bytes aus dem Akkumulator (in verschiedenen Schrittweiten durchlaufen) mit einem Byte des transponierten Status verknüpfen

$s^j = s^{p(j)} \oplus r^j$ Die 64–Bit–Worte des Status werden permutiert und mit den generierten Zahlen verXODERt.

$e' = a \oplus e, a' = s \oplus e'$ Akkumulator, Message („Entropie“) und Status miteinander verXODERn.

Die „naive“ Implementierung in Forth sieht dann so aus (nur die wesentlichen Teile):

```
8table round#
   13 , 29 , 19 , 23 , 31 , 47 , 17 , 37 ,
8table permut#
   2 , 6 , 1 , 4 , 7 , 0 , 5 , 3 ,
   \ permut length 15
: mix2bytes
  ( index n k -- b1 .. b8 index' n )
  wurst-state + 8 0 DO
    >r over wurst-source + c@
    r@ c@ xor -rot dup >r + $3F and
    r> r> 8 + LOOP
  drop ;
: bytes2sum ( ud b1 .. b8 -- ud' )
     >r >r >r >r  >r >r >r >r
     r> rngs wurst  r> rngs wurst
     r> rngs wurst  r> rngs wurst
     r> rngs wurst  r> rngs wurst
     r> rngs wurst  r> rngs wurst ;
: round ( n -- ) dup 1- swap  8 0 DO
  wurst-state I permut#
     64s + 64@ -64swap
  I mix2bytes 2>r bytes2sum 2r>
     64swap nextstate I 64s + 64!
     LOOP 2drop update-state ;
: rounds ( n -- )  message swap
    dup $F and 8 umin 0 ?DO
I rounds# round
dup 'round-flags Ith and IF
    swap +entropy swap
THEN
    LOOP 2drop ;
```

Performance

Die naive Implementierung war zunächst sehr langsam (bigForth: ca. 1MB/s auf einem 2,5GHz Phenom). Wahrscheinlich war ein Code–Daten–Konflikt das Problem — nach Beseitigung dessen ging die Performance auf 23MB/s hoch (Gforth: 20MB/s, dank 64 Bit nur unwesentlich langsamer). Um die Performance zu verbessern, habe ich die Schleife aufgerollt, was die Performance auf 46MB/s verdoppelte (bigForth).

Das war immer noch nicht sehr befriedigend, also habe ich das libcc–basierte C–Interface in Gforth genutzt, um den aufgerollten Schleifen–Code mit dem GCC zu compilieren. Mit knapp über 500MB/s kam dabei dann ein Ergebnis heraus, das auch den Erwartungen entspricht.

Als Benchmark nutze ich das Wort `time-rng`, als Berechnungsgrundlage dient eine Runde pro 512 Bits. Der Zufallsgenerator, der hier getimed wird, macht zwei Runden pro 512 Bits, liefert also ca. 250MB/s Zufallszahlen.

Analyse

Um festzustellen, wie sicher der Algorithmus ist, muss man natürlich versuchen, ihn zu brechen. Die Funktion ϕ soll ja unumkehrbar sein, d.h., wir sollten uns ansehen, wie sich z.B. das Ändern eines einzelnen Bits auswirkt. Nach einer Runde unterscheidet sich ein 64–Bit–Wort, und dieses 64–Bit–Wort hat auch nur genau einen unserer 256 Zufallswerte getauscht. Das heißt, wir müssen uns „nur“ die 64k möglichen Kombinationen zwischen zwei Zufallswerten und 8 mögliche Rotationen ansehen, um auf die Änderung dieses einzelnen Bits zurückschließen zu können.

Bei der nächsten Runde breitet sich die Änderung des Status entsprehend aus. Alle 8 Status–Worte sind nun anders und haben mindestens eine Zufallszahl getauscht; da wir in den meisten Fällen auch Auswirkungen auf die Änderung des Message–Akkumulators haben, im Durchschnitt sogar zwei Zufallszwahlen. Dazu müssten wir nun 2^{32} mögliche Kombinationen durchsuchen, und zwar pro Wort. Eine weitere Runde sind dann alle Zufallszahlen im Status ausgetauscht, ein Zurückschließen ist weitgehend unmöglich. Der Message–Akkumulator hinkt prinzipbedingt immer eine Runde hinterher, weil hier nur der alte Status ver-XODERt wird. Wir hängen also noch eine vierte Runde an, um sicher zu gehen, dass sowohl Status als auch Message–Akkumulator ausreichend durcheinandergewürfelt sind.

Wie viele Runden müssen wir zwischen dem Hinzufügen von Information beim Hashen ausführen? Vermutlich nur eine. Wir müssen aber am Ende der Hash–Berechnung mindestens 4 Leerrunden durchführen, damit kein Rückschluss auf die Message möglich ist.

Wieviele müssen wir beim Verschlüsseln ausführen? Mindestens 2. Um das zu beweisen, reicht eine einfache Klartextattacke: Angenommen, wir kennen die ersten 128 Bytes einer Message (inklusive Länge). Wir können dann aus den ersten beiden Blöcken des verschlüsselten Texts sowohl den Akkumulator als auch den Status berechnen — einfach, indem wir die beiden aufeinanderfolgenden Blöcke ver-XODERn, und die darin enthaltenen Message–Blöcke herausrechnen. Mit diesem vollständig rekonstruierten internen Status können wir den Rest des Texts entschlüsseln.

Das klappt nicht, wenn wir eine zweite Runde hinzufügen — denn dann befindet sich in jedem Block der verschlüsselten Message die XODER–Verknüpfung von zwei internen Zuständen. Daraus einen einzelnen herauszurechnen ist schwer möglich.

Anweisungen zum sicheren Gebrauch

Kryptographie ist nur sicher, wenn sie sorgfältig betrieben wird. Die folgenden Anweisungen beschreiben, wie man bekannte Probleme umgeht.

Signatur Signiert wird der Hash–Wert der Datei, indem er mit dem privaten Schlüssel verschüsselt wird. Es wird empfohlen, einen zufälligen Startwert im Akkumulator zu verwenden. Dieser zufällige Startwert stellt sicher, dass ein Angreifer nicht vorab eine Kollision berechnen kann, und damit später ein anderes Dokument unterschieben kann. Nachträglich kann ein Angreifer immer noch eine Kollision finden, aber das ist erheblich schwerer (verdoppelt die Anzahl Bits in der Stärke des Hashs). Bruce Schneier empfiehlt, als Startwert den öffentlichen Schlüssel zu verwenden — das ist aber wenig sinnvoll, da der dem Angreifer ja bekannt ist.

Verschlüsselung Verschlüsselt man mit einem Schlüssel mehrere Nachrichtenteile (z.B. Anhänge in einer E–Mail), so ist es ratsam, den Akkumulator für jeden Teil mit einem anderen Zufallswert vorzubelegen. Will man eine Replay–Attacke verhindern, kann man auch einen Timestamp in die Vorbelegung einbeziehen.

Literatur

[1] Vincent Rijmen und Paulo S. L. M. Barreto *Whirlpool Page* `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`

[2] Meldung auf Heise Security vom 11.6.2009 *Attacke auf SHA-1 weiter vereinfacht*

[3] Der komplette Quellcode, der zu lang zum Abdrucken ist, ist Teil des bigForth–Repositorys `http://www.forth-ev.de/repos/bigforth/wurstkessel.fs`

Die ForthForm des Win32Forth erweitern — ein visualForth machen

Dirk Brühl

Dirk Brühl lebt heute im Pennsylvania, USA. 1993 hat er in Nürnberg die Forthtagung ausgerichtet, an der auch ein gewisser Herr Tom Zimmer teilgenommen hatte. Dessen Forths [1] haben etliche von uns inspiriert und seither begleitet, auch beruflich. Neulich im Internet sind wir uns wieder begegnet, Dirk Brühl und ich. Dabei ging es auch um Beiträge für die *Vierte Dimension* — und ich war erfreut zu erfahren, was sich so in dem Blockhaus in Pennsylvanien alles tut.

Dirk schrieb: „... so will ich sehen, sobald wie möglich eine Vorabkopie zusammenzustellen und Dir zuzuschicken. Ist nicht sehr umfangreich, ich hoffe, dass Du Win32Forth Version 6.12 (oder 6.13) zur Verfügung hast. Mein visualForth ist eine Ergänzung zu ForthForm von Win32Forth Version 6.12 (Ezra Boyce hat unvorsichtigerweise irgendwo geschrieben, dass jeder damit machen kann, was er will), so werden nur einige Files benötigt, die in ForthForm einkopiert werden, und dann wird das System neu aufgebaut (geht praktisch automatisch). Also, falls Du Interesse hast, Dir mein visualForth mal kritisch anzuschauen, lass es mich wissen! Mit den besten Grüßen aus dem noch teilweise weißen Springs“

Und ob ich da interessiert war! Gleich schrieb ich zurück, dass ich das gerne ausprobieren und vorstellen wolle, und erhielt noch am selben Tage die Antwort: „Habe ich womöglich irgendeinen Nerv getroffen?“Na klar, hatte er — den Forth–Nerv der *Älteren*: Wissen wollen, ob und wie was geht. M.Kalus

Warum die ForthForm von Win32Forth erweitert gehört.

Wenn Du Win32Forth Version 6.12 installieren konntest, hast Du bereits ForthForm (ist unter Apps zu finden). Wenn Du versuchst, damit ein GUI zu machen, dann geht das perfekt, aber wenn Du versuchst, ein Forth–Programm damit zu verbinden, wird es Dir wahrscheinlich so wie mir gehen, dass Du da wie ein Ochs vorm Berg stehst und nicht weißt, wie das anzustellen ist.

Ich habe dann versucht herauszufinden, wie das geht, erst für mich alleine, dann mit eMail–Fragen in der Forth–Yahoo–Gruppe, dann hatte ich einige Ideen, wie man das Ganze für mich und Leute wie mich verwendbar machen kann, und bat Ezra, das doch mal anzuschauen und einzubauen. Aber erst als ich mich dann selbst an die Arbeit machte und alle meine Widerstände gegen den Umgang mit Windows–Programmierung überwunden hatte und eine erste arbeitsfähige Lösung hatte, war Ezra auch begeistert, und das erste, was er machte, war, einen anderen Weg einzuschlagen und das Ganze in die IDE einzubinden. Da mir zum Weiterarbeiten dann deren Source–Code fehlte, bin ich bei meiner Lösung geblieben.

Beruflich hatte ich bei Lucent mit visualBasic zu tun.[1] Ein passendes Forth gab es damals leider nicht. Aber wenigstens den BASIC–Code schrieb ich Forth–ähnlich, und die ganze Zeit dachte ich mir, wie schön es wäre, ein visualForth zu haben. Es ging anderen wohl ähnlich, Überlegungen dazu gab es in Foren und Konferenzen, z. B. Knaggs [2] oder Prinz [3]. Immer wurde der Gedanke verworfen, weil man dachte, es sei für Einzelne zu umfangreich und daher nicht zu verwirklichen. Bernd

[1] EBID–tool; Windows–Programm, um EEPROMs automatisch mit Fertigungsdaten zu beschreiben.

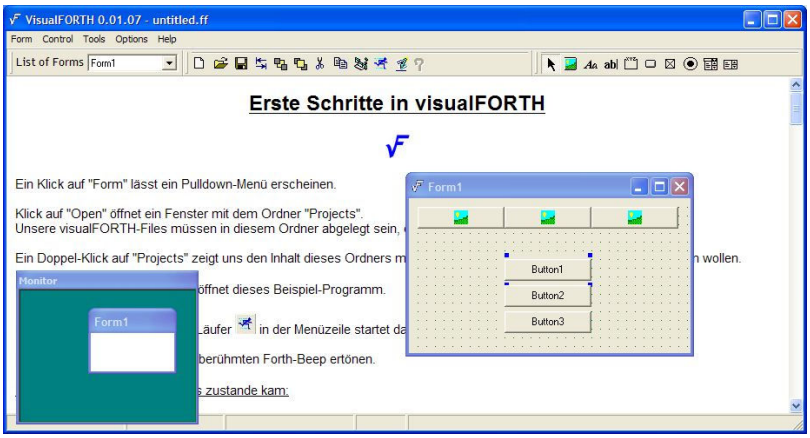


Bild 1: Ein neues Fenster (Form1) wird angelegt.

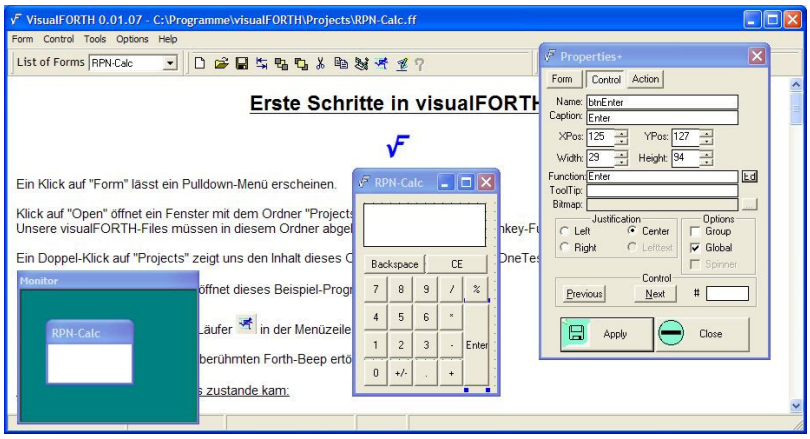


Bild 2: Beispiel für ein Projekt - erzeugen eines RPN Rechners.

Paysan machte dann mit MINOS vor, dass diese Befürchtungen eigentlich unbegründet sind [4]. Minos ist ein erstaunliches Tool, um GUIs zu erstellen. Da Bernd aber vornehmlich in einer Linux Umgebung entwickelt, ist die Portierung in die Windowsumgebung nicht so glücklich verlaufen, es verweigert sich mir dort zu oft völlig, auch aus nicht ersichtlichen Gründen.

Inzwischen ist ja einige Zeit verstrichen, es gibt mehr Leute, die Windows XP programmieren können, selbst unter denen, die Forth mögen. Und nun, da es die ForthForm des W32F gibt, ist es möglich geworden, den Gedanken an ein visualForth noch mal aufzugreifen, und eigene Wünsche umzusetzen.

So hatte ich kürzlich während der Arbeit an meinem Webpage–Maker einiges in einer Wunschliste notiert, die ich bei Gelegenheit abzuarbeiten gedachte. Jetzt war die Herausforderung da, und da nun alle wichtigen Leute geweckt wurden[2], bleibt mir nichts anderes mehr übrig, als die erste brauchbare Version verfügbar zu machen.

Was ich der ForthForm hinzugefügt habe — neue Features.

Man nehme ein Win32Forth zur Hand und patche die alpha–Version meines Ansatzes (visualForth0107) darüber, der Vorgang wird von einem Batch–Programm unterstützt. Dann stehen beide Ausführungen zur Verfügung, die bisherige ForthForm und das, was ich mir ausgehend davon derzeit unter visualForth vorstelle. Die simple Idee, die der ForthForm zugrunde liegt, wurde beibehalten: Eine neue Form öffnen, dann aus einem Menü ein Kontrollelement (control) auswählen — also ein Knopf (button) oder Bildchen (bitmap) oder Dialogfeld — das per Mausklick einfügen, fertig. Wirklich sehr simpel und sofort einsichtig.

Dann der nächste Schritt. Ein Klick mit der rechten Maustaste auf so einen Knopf, und es erscheint ein Fenster, in dem allerlei Eigenschaften dieses Elementes eingestellt werden können (properties). In der ForthForm beschränkte sich das bisher auf wenige Eigenschaften, wie die Position des Elements, blättern von Element zu Element, Namen und Titel. Zusammen mit der Möglichkeit, das Ganze dann als Forthcode in ein File abzulegen, ermöglichte schon sehr viel. Von da an konnte man auf Forth–Quellcode–Ebene weiter machen, hatte den mühseligen Teil der Windowsprogrammierung schon geschafft. Eigentlich ein mächtiges Werkzeug in den Händen von Experten! Nur wurde Menschen wie mir dabei nicht klar, was denn damit gemeint ist, Code lokal oder global anzulegen, was es mit den Objekten auf sich hat, oder wo denn nun der ausführbare Code hingehört, der laufen soll, wenn man seinen soeben frisch erzeugten Knopf anklickt — frustrierend. Also eigentlich genau das nicht, was Forth sonst immer bietet: Einen leichten Einstieg über einfache Funktionen, von wo aus man sich dann nach und nach im Selbststudium in kompliziertere Zusammenhänge der Maschine vorarbeiten kann.

Also habe ich ein Feld zur Eingabe eines Forthwortes hinzugefügt und dem einen Editor hinzugefügt. So kann schon *in* der Form dem Kontrollelement seine Forthfunktion hinzugefügt werden, auch als Dummy, um dann später im Quellcode die Stelle wiederzufinden, an der man weitermachen kann, falls man das dann überhaupt noch möchte (und dabei nicht die Hände über dem Kopf zusammenschlägt und gleich wieder zumacht angesichts all der Windows–Funktionen). Denn es geht auch so aus der Form heraus, seine Applikation zu erstellen. Das macht es ja gerade so einfach: Man konzentriert sich auf Forth. Dieses war der wichtigste Schritt.

Dann kamen zu den Hilfen, die man über die rechte Maustaste für das Propertiesfenster erhält, folgende hinzu: Die derzeit bearbeitete Form in ihrer Quelle sehen können (view source), komplette Kontrollelemente kopieren, auch multipel zu Elementfeldern (copy control), Elementengruppen markieren und löschen, Listen der schon generierten Elemente und controls erzeugen, und eine automatische Nummerierung dieser Elemente, damit man sie gezielt aufrufen und in der Quelle leicht finden kann.

Und schließlich habe ich noch hinzugefügt, testweise eine so erzeugte Form mit nur einem einzigen Mausklick

[2] Die versammelte Expertenrunde auf der Forthtagung 2009 in Neuenkirchen bei Rheine.

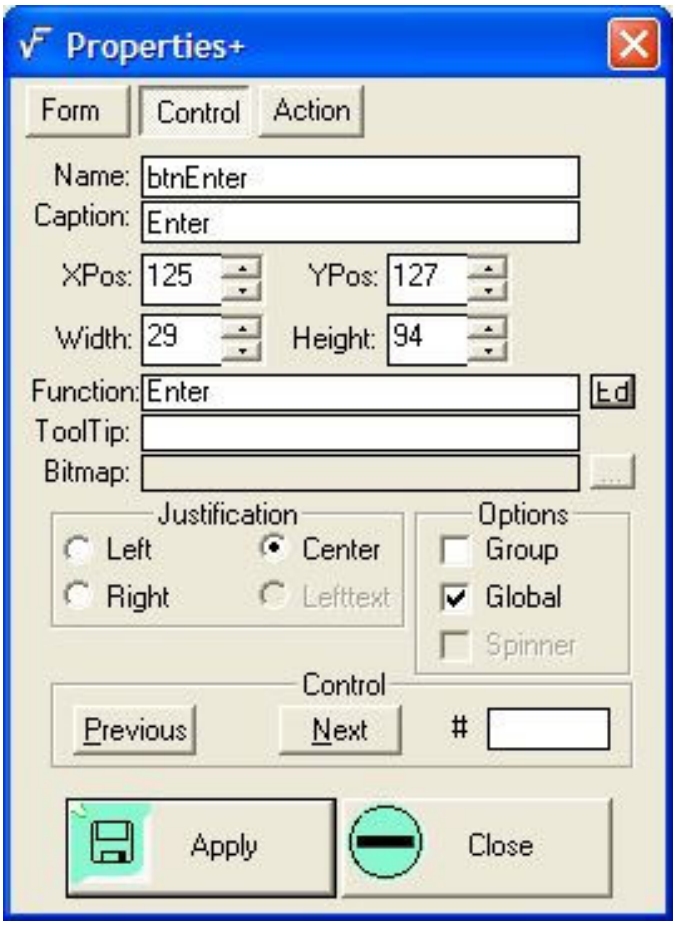


Bild 3: Eigenschaften der Form festlege, eine Forth Funktion zuweisen.

in eine `EXE`–Datei umzusetzen. Damit man sofort sehen kann, ob und wie das läuft, was man da gerade gemacht hat. Also nicht nur die *run*–Funktion der ForthForm, mit der man sehen kann, wie die Form im Forth compiliert und läuft, sondern Anlage der selbständigen ausführbaren Stand–Alone–Datei (`*.EXE`) um das Endergebnis sofort zu haben, wobei die gesamte Quelle dafür ebenfalls abgelegt wird (`*.frm`), was den Prozess der Fehlerbehebung unterstützt. Denn das war über die bisherige ForthForm alles doch sehr holperig: Quelle der Form generieren, händisch die turnkey–Prozedur anfügen, Funktionen den einzelnen Controls zuordnen, compilieren, `EXE` generieren, Fehler finden, alles von vorn; Form öffnen usw. Ich hoffe, dass dieser mühselige Zyklus mit dem vF nun erheblich abgekürzt werden konnte.

Arbeitsweise in visualForth

Bei visualFORTH ist es so, dass mit einer leeren Form begonnen wird, und der graphische Teil zuerst gemacht wird. Meine Arbeitsweise bei der Erstellung des Webseitengenerators war so, dass ich erst einmal den graphischen Teil, also die Bedienoberfläche, erstellt habe, denn nur damit machte es Sinn, die anderen Programm–Module zu erstellen: Erst muss die I/O da sein.

Das war auch stets bei der Entwicklung von Mikroprozessorsystemen bei mir so. Ohne Hardware gab es keine Möglichkeit, die Software zu testen — manche tun es trotzdem, aber mit immensem Aufwand, den ich stets vermieden habe. Mit der bestehenden Testmöglichkeit in Form der GUI macht es Sinn, mit der Programmerstellung weiterzumachen.

Das ist Forth–gemäßes Programmieren, auch inkrementelles Programmieren genannt. Dadurch, dass alle Verknüpfungen bereits in der Form untergebracht werden können, ist die Form an sich ihr eigener Projektmanager. Mit der einfachen ForthForm war das so nicht möglich.

Was zu tun war.

Bis zum heutigen Stand (Mai2009) von visualFORTH 0.01.07, der Alpha–Version, habe ich 249 Änderungen und Ergänzungen in den 49 ForthForm–Files vorgenommen. Um eine Umleitung der Key– und Emit–Funktionen in eine Form zu ermöglichen, waren zwei neue Files, und zwar FormKey.f und TextboxEmit.f erforderlich, und eine Änderung des Forthkernels in `fkernel.f`, also des *Allerheiligsten*, weil im Gegensatz zu früheren Forth–Versionen in Win32Forth für `KEY` und `KEY?` inzwischen Windows–Funktionen verwendet werden, die diese I/O–Umleitung unmöglich machten. Also musste ich `KEY` neu definieren. Für mich als Mikroprozessor–Entwickler (Embedded Systems Engineer) ist diese Umleitungs–Möglichkeit außerordentlich wichtig, denn dadurch kann man einen Text auf ein

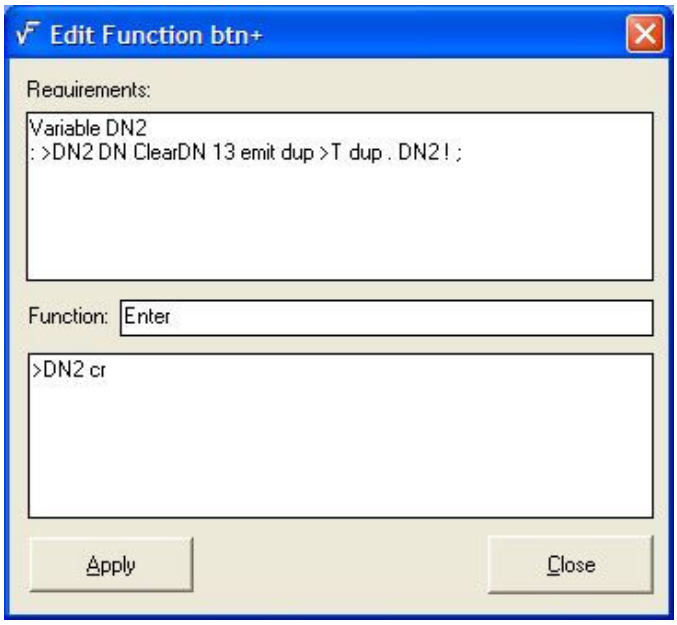


Bild 4: Der PLUS Taste ihre Forth Funktion zuweisen.

LCD–Display mit dem gleichen Programm–Modul ausgeben, das den Text auf den PC–Bildschirm bringt. (Bei meinen Mikroprozessor–Programmen wurden Tasten der Hardware gescannt und durch I/O–Umleitung direkt ins Forth eingespeist, sodass das gleiche Programm sowohl mit Hardware als auch ohne Hardware laufen konnte; im RSC–Forth war das eingebaut.) So habe ich mir diese Funktionen jetzt mit visualFORTH und Win32Forth Version 6.12.07 auch gemacht, zur Verwendung von simulierter *Hardware*, simulierten LEDs und Tasten, *Form* genannt.

Das Zusammenspiel zwischen der erstellten und kompilierten Form und dem W32F–Kern ist außerordentlich interessant. Es ist eine Art von Multitasking. Von der W32F–Console aus (und damit auch von einem kompilierten Forth–Programm) kann mit dieser Form kommuniziert werden.

Das eigentliche Problem war, all die Stellen zu finden, an denen Eingriffe möglich und erforderlich waren. Dabei war die Win32Forth–IDE mit *Find Text in Files* (<Strg>+<Shift>+F) eine großartige Hilfe. Ohne dieses speziell für Forth entwickelte Werkzeug wäre es nur mit erheblich höherem Zeitaufwand möglich gewesen, visualFORTH zum Leben zu erwecken. Aber bevor ich in den Tiefen irgendetwas geändert habe, musste zunächst einmal das Herzstück für visualFORTH gemacht werden: Funktionen den Objekten zuordnen. Angefangen habe ich mit einer einfachen Taste (button) im Fenster. Das war am einfachsten zu bewerkstelligen, es sollte überhaupt irgendetwas geschehen, wenn man auf diese Taste klickte. Dazu wurde das ForthForm–Fenster *Properties+* um ein Schreibfeld erweitert. Das wiederum war mit Hilfe der ForthForm nach Starten von `FORMPROPERTY.ff` möglich. So gab es bald eine erweiterte `FORMPROPERTY.ff`, aber noch fehlte die Anbindung an Forth.

Dazu habe ich nach den Texten in *Properties+*, z. B. nach *Bitmap:* gesucht, und mit Hilfe der IDE alle damit zusammenhängenden Funktionen nacheinander gefunden. Diese Module habe ich kopiert und mit einem neuen Namen versehen, d.h. aus *Bitmap:* habe ich letztendlich die *Function:* gemacht. Das war im Grunde genommen alles für den ersten Schritt. Ein Problem dabei gab es allerdings dabei zu lösen: Wie werden die Eigenschaften (properties) einer Form eigentlich gespeichert und wo liegen diese Daten? Die Datenstruktur habe ich dann gefunden, und einfach meine Ergänzungen angehängt. Beim ersten Test dann funktionierte die ForthForm danach nicht mehr, stürzte ab mit einer Fehlermeldung. Bis ich herausgefunden hatte, dass der Speicherplatz für die Properties — aus welchen Gründen auch immer — begrenzt ist. Da in der Form selbst nur wenige Definitionen erforderlich sind, reicht aber der zur Verfügung stehende Speicherplatz gut aus. Hier liegt noch einiges im Dunkeln und ich hoffe auf Mitstreiter bei der Aufklärung.

Kleine visuelle Vorschau

In den im folgenden angesprochenen Bildern (screenshots) ist der Prozess dargestellt, mit dem im visualForth gearbeitet wird. Das Bild 1 auf Seite 21 zeigt eine neu angelegte Form. Die nächsten Bilder stammen aus dem Taschenrechner–Beispiel (RPN–Calc). Im Bild 2 auf Seite 22 sieht man, wie der Taschenrechner entsteht. In der Mitte davon siehst du den Entwurf der Taschenrechner–Form. Rechts davon ein Fenster, in dem gerade die Eigenschaften der ENTER–Taste festgelegt werden. Und links ein kleines Fenster mit dem die Position auf dem Bildschirm ausprobiert wird. In Bild 3 auf der vorherigen Seite sieht man dann deutlicher, wie die ENTER–Taste angelegt wurde. Schließlich zeigt Bild 4 als Beispiel, wie die PLUS–Taste programmiert worden ist.

Was ist zu tun?

Noch viel. Und das hat weniger mit Forth selbst zu tun, daher ist es für mich schwierig. Und die Wunschliste erhebt nicht mal Anspruch auf Vollständigkeit ;-)

- Eine selbstentpackende Distributions–`EXE` automatisch erstellen.
- Bilder (BMP) innerhalb einer Turnkey–`EXE` ablegen können.
- Bimaps an verschiedene Fenstergrößen anpassen können, um ein automatisches Setzen von Bildern vom Programm aus reibungslos zu ermöglichen (ist irgendwo in den W32F–Sourcen oder in Windows selbst gegeben, habe es bisher nicht gefunden).
- DLLs innerhalb einer Turnkey–`EXE` speichern, die beim Ausführen automatisch ausgeladen werden, falls erforderlich. Damit ist es dann möglich, alles in einem einzigen `EXE`–File zu haben. Denn es ist sehr ärgerlich, wenn ein Programm mit einer Fehlermeldung startet *File nicht gefunden* und dann nicht läuft. Mit dem Einpacken aller erforderlichen Extras in die EXE ist diesem Frust ein Ende gesetzt. Alle anderen Bestandteile von Forth sind stets in der `EXE` enthalten, so sollte es auch mit Bildern und DLLs sein.

Wie geht es weiter?

Da die Quellen in der Win32Forth–Distribution (meine derzeit: Version 6.12.07) vorhanden sind, ist es gut möglich, dass, wer möchte und kann, einen Beitrag zur Vervollständigung von visualFORTH leistet, und dieser Beitrag muss nicht unbedingt ein Sourcecode sein — Wünsche und Bemerkungen, was in visualFORTH fehlt, um es noch vollständiger zu machen und den Schritt zur Beta–Version zu gehen, sind herzlich willkommen und dienen allen, die mit visualFORTH arbeiten oder arbeiten wollen.

Mit dem Konzept eines visualFORTH sind nun alle Fenster geöffnet, um frischen Wind ins Forth–Geschehen zu bringen, und alle Türen sind geöffnet, um die verborgenen Schätze von Forth zu heben und verfügbar zu machen. Mit einer vereinheitlichten und stabilen Version von visualFORTH ist es möglich, interessante, im Internet frei verfügbare Forth–Applikationen nutzbar zu machen. Jeder, der die Power von Forth erkannt hat, wird erkennen, dass diese Power mit visualFORTH potenziert wird! Alles, was für die Nutzbarmachung von Windows durch Forth benötigt wurde, ist bereits vorhanden. Alles ist fertig, Win32Forth ist ein mächtiges Forth–Werkzeug, das alle Wünsche erfüllt, wenn es richtig verwendet wird. Und das letzte Glied der Kette, *the missing Link*, visualFORTH, ist jetzt auch da, und visualFORTH ist problemlos erweiterbar, kompatibel zum W32F von Anfang an.

Jetzt geht es darum, die Power von Forth sichtbar zu machen, zu visualisieren, und in visualFORTH zum Ausdruck zu bringen! Ich hoffe, dass Du die Möglichkeit dieser Möglichkeiten auch siehst!

Nachtrag

Inzwischen (Juli 2009) ist Win32Forth Version 6.13 da und wurde gegenüber 6.12 so überarbeitet, dass visualForth damit nicht mehr funktioniert. Das visualForth Konzept muss daher als eigenständig angesehen werden. Zur Drucklegung dieses Heftes wird visualForth0112 da sein. Der gesamte Code für ein damit erzeugtes `EXE`-File von einer eigenen Applikation ist in einem Quellfile *.ftk niedergelegt. Bitmaps werden darin jetzt automatisch eingebunden und auch das zoomen der Fenster funktioniert inzwischen. db

Referenzen

[1] `http://tomzimmer.blogspot.com/`
... Author or Co–Author of several Forth computer language development systems including VIC–Forth, ColorForth (for the Radio Shack Color COmputer), 64Forth (for the Commodore 64), F–PC (for DOS), Win32Forth (for Windows) and TCOM (a native DOS compiler), Thomas Zimmer, win32forth@me.com

[2] Peter K. Knaggs, Department of Computing and Information Systems, University of Paisley, Scotland; 1996; *Visual Forth — The Forth Integrated Development Environment.*

[3] Friedrich Prinz, Jahrestagung FG 2/1996. Bericht: Erfahrungen mit dem Win32Forth.
Dabei Wunsch nach einem VisualForth, das den Entwickler von der Kenntnis der Windowsprogrammierung befreien sollte und seine Konzentration auf das Inhaltliche ermöglichen kann. In der lebhaften Diskussion wurde befürchtet, dass zum einen Windowsprogrammierung ohne Windowskenntnisse eine Utopie bleiben wird und dass zum anderen ein VisualForth mit den vorhandenen Entwicklungskapazitäten kaum zu machen sein wird.

[4] Bernd Paysan. Minos und bigForth `http://www.jwdt.com/~paysan/bigforth.html`
„MINOΣ ist die Antwort zu der Frage: »Gibt es da etwas wie Visual BASIC in Forth?«... *Visual Forth?* Etwas gibt mir zu denken: In diese Pakete, die echte Programmierung visueller Forms ermöglichen, sind viele Jahre Programmierung investiert. Microsoft, Borland, und IBM mögen Hunderte von Programmierern für so ein Projekt einstellen. Diese Manpower gibt es für kein Forth–Projekt. Aber halt: Forth behauptet, dass gute Programmierer mit Forth sehr viel effizienter programmieren können...“

*F*orth *I*nspired *C*ommand *L*anguage (*FICL*[1])

Gerd Franzkowiak

Auf der Suche nach einem kleinen FORTH–System für meine Embedded–Linux–Entwicklung stieß ich vor einiger Zeit auf FICL. Erfreulicherweise konnte ich auf einem ARM9–System mit `Clibc-Linux`[2] ein ca. 100 KB großes statisches *ficl* erzeugen. Dies schien mir ansprechend genug, um das System mal etwas näher zu beleuchten.

Zitat:
Wenn die meisten sich schon armseliger Kleider und Möbel schämen, wie viel mehr sollten wir uns da erst armseliger Ideen und Weltanschauungen schämen.
(Albert Einstein)

Einführung

Eigentlich ist *FICL* nicht als 100%–iges FORTH entstanden. Es lehnt sich aber stark an den ANS von 1994 und vermittelt mit der ersten Anwendung sofort FORTH–feeling.

Entwickelt wurde *FICL* von *John Sadler* und auf der 20^{th} FORML Conference 1989 vorgestellt. John Sadler wollte ein Tool, welches für schnelle und effektive Entwicklungen in andere Anwendungen, vorrangig C/C++, einzubetten ist und auf verschiedenster Firmware nutzbar sein kann. Aufgrund dessen hat John den Kern komplett in ANSI–C geschrieben.

Die Lizenz von *FICL* ist im Stil der BSD–Lizenz und stellt somit kaum ein Hindernis für eine große Zahl von Projekten dar.

Dementsprechend kann man *FICL* auch als Bootloader für den Kernel von FreeBSD finden.

Erste Schritte

Mir ging es bei meinem ersten Versuch darum, einfach mal meinen Treiber, den ich für ein FPGA auf einer ARM9–Anwendung geschrieben hatte, anzusprechen.

Gesagt, getan, nahm ich mir als Toolchain die Scratch-Box von Nokia (GPL) zur Hand, schob die Quellen von ficl4.0.31 hinein, compilierte erfolgreich und kopierte es auf das genannte Board.

Ich startete <./ficl> und — siehe da; *loading, Date, version* — es lief.

Mit der Eingabe von *WORDS* fühlte ich mich auch gleich in der richtigen Umgebung. Es ist fast alles vorhanden, was man in FORTH so benötigt.

Um nun meine Treiber mal anzusprechen, versuchte ich,

```
s" /dev/myfpga" r/w open-file throw Value fd
```

einzugeben, und konnte mit

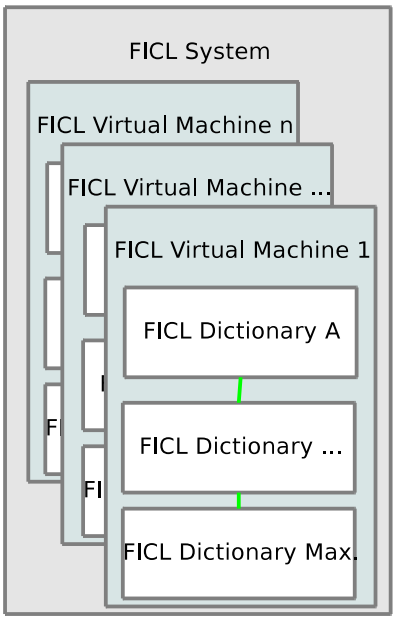
```
s'' xxx'' fd write-file fd close-file
```

erste Reaktionen erleben.

Das nahezu ideale Tool !

Welche Reaktionen das auf dem Board waren, ist hier nicht so wichtig, es wurden jedenfalls Tabellen–Werte im FPGA eingetragen.

Der Aufbau



In FICL wird mit dem Start ein System angelegt, welches aus mindestens einer virtuellen Maschine, mit zugehörigen Dictionarys, besteht, aber auch aus mehreren VMs bestehen kann.

Mit dem Anlegen eines FICL–Systems existiert automatisch ein ANS Forth CORE word set, welches dann, nach der entsprechenden Intitialisierung, die Dictionarys der virtuellen Maschine(n) anlegt. Das Schöne an den virtuellen Maschinen ist, dass jede einen eigenen Status hält und eigene I/O–Kanäle möglich sind.

Darüber hinaus ist, zu des Programmierers Glück, FICL–Code auch noch *thread safe* und *re–entrant*.

Für denjenigen, welcher sich in FICL einarbeiten möchte, liegt eine umfangreiche Dokumentation vor und die Sourcen bieten natürlich beste Informationen.

[1] `http://ficl.sourceforge.net`

[2] Linux wurde compiliert mit der für eingebettete Systeme optimierten C library `Clibc`

Das Prinzip

FICL ist, entsprechend Anton Ertls Beschreibung[3] über Threaded Code als Switch Threading Forth entwickelt worden.

Das Application Interface gestaltet sich recht einfach, gut und überschaubar. Ein Beispiel für den Aufruf aus anderen C–Applikationen kann so aussehen, wie es in *main.c* vorliegt:

```
#include <stdio.h>
#include <stdlib.h>
#include "ficl.h"
int main(int argc, char **argv) {
    int returnValue = 0;
    char buffer[256];
    ficlVm *vm;
    ficlSystem *system;
    system = ficlSystemCreate(NULL); /* System anlegen */
    ficlSystemCompileExtras(system);
    vm = ficlSystemCreateVm(system); /* Virtual Machine erzeugen */
    returnValue = ficlVmEvaluate(vm,
                        ".ver .( " __DATE__ " ) cr quit");
    while (returnValue != FICL_VM_STATUS_USER_EXIT) {
        fputs(FICL_PROMPT, stdout);
        fgets(buffer, sizeof(buffer), stdin);
        returnValue = ficlVmEvaluate(vm, buffer);
    }
    ficlSystemDestroy(system);
    return 0;
}
```

Genau auf diese Art und Weise ist man in einem C–Projekt auch in der Lage, mal einen kleinen interaktiven Part einzubinden und sich ein wenig im System umzusehen.

Mit der Funktion ficlSystemCreate und einer angepassten Struktur, statt dem NULL–Pointer in der Parameterübergabe, bieten sich dann weitere Möglichkeiten eines eigenen Systems an.

```
struct ficlSystemInformation {
 int size;           /* structure size tag for versioning */
 void *context;      /* Initializes VM's context pointer -
                        for application use */
 int dictionarySize; /* Size of system's Dictionary, in cells */
 int stackSize;      /* Size of all stacks created, in cells  */
 ficlOutputFunction textOut;  /* default textOut function */
 ficlOutputFunction errorOut; /* textOut function used for errors */
 int environmentSize; /* Size of Environment dictionary, in cells */
};
```

Zu erkennen ist, dass mit dieser Struktur schon eigene Ausgabe–Funktionen festgelegt werden können. Soll dies für jede VM erfolgen, so wird es über Callback–Funktionen der VMs realisiert.

Netzwerk–Anbindung

Schaut man sich z.B. FORTH–83–Systeme an, so war es schon eine Kleinigkeit, wenn man den Ein–/Ausgabestrom auf andere Hardware umlenken wollte. Es genügte schon, die Worte KEY?, KEY und EMIT anzupassen, und ein FORTH–System funktionierte beispielsweise über eine serielle Schnittstelle.
Bedenkt man, welche Klimmzüge C–Programme da vollziehen mussten und welche Software–Monster (PC–Irgendwas) da notwendig wurden, war FORTH immer schon entwicklerfreundlich.

Mit Netzwerkanbindungen, z.B. über einen TCP/IP–Stack, ist es natürlich nicht so einfach. Nicht einfach ASCII–Zeichen werden übertragen, sondern ein kompletter Protokollstapel muss durchlaufen werden.
FICL bietet mit seiner Methode jedoch ein recht elegantes Verfahren, was ein FORTH, sofern in dessen Umgebung ein TCP/IP–Stack vorhanden ist, schnell über ein Netzwerk bedienbar macht.
Meine Versuche waren das Anbinden des TCP/IP–Stacks an das FICL–System auf folgende Art:

```
/*
 * my.c
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include "ficl.h"
/*
 * Example of server using TCP protocol.
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#define SERV_TCP_PORT 65535
#define SERV_HOST_ADDR ""
void catcher();
typedef struct {
  FILE   *fptr;
  int    newsockfd;
} myVmSockPar;
myVmSockPar vmSockPar;
int         sockfd;
/*
 * Callback function
 */
void myCbTextOut(ficlCallback *pCB, char *text) {
  char buffer[256];
  FICL_IGNORE(pCB);
  if (text != NULL) {
    fputs(text, stdout);
  }
  else {
    fflush(stdout);
  }
  return;
}
/*
 *Callback function for TCP connection
 */
void mySocketTextOut(ficlCallback *pCB, char *text) {
  FICL_IGNORE(pCB);
  if (text != NULL) {
    fputs(text, vmSockPar.fptr);
  }
  return;
}
int myVmSocketConnection(ficlVm *vm, int vmsockfd) {
  char   buffer[256];
  int    returnValue = 0;
  vmSockPar.fptr = fdopen(vmsockfd, "r+");
  /* The old one was vmSetTextOut */
  ficlVmSetTextOut(vm, &mySocketTextOut);
  ficlVmSetErrorOut(vm, &mySocketTextOut);
  while(returnValue != FICL_VM_STATUS_USER_EXIT) {
    fputs(FICL_PROMPT, vmSockPar.fptr);
    fgets(buffer, sizeof(buffer), vmSockPar.fptr);
    returnValue = ficlVmEvaluate(vm, buffer);
  }
  fclose(vmSockPar.fptr);
  close(vmsockfd);
  return FICL_VM_STATUS_USER_EXIT;
}
void catcher() {
  close(vmSockPar.newsockfd);
  close(sockfd);
  exit(1);
}
#define NUMBEROFSYSTEMS 2
#define NUMBEROFVMS 2
#define STARTUP 0
```

[3] http://www.complang.tuwien.ac.at/forth/threaded-code.html

```
int main(int argc, char **argv) {
 int        netuse;
 int        returnValue = 0;
 int        pid;
 int        state;
 char       buffer[256];
 char       *selstr = buffer;
 int        clilen;
 ficlVm     *vm[NUMBEROFVMS];
 ficlSystem *system[NUMBEROFSYSTEMS];
 ficlSystem *showtest;
 struct sockaddr_in server_addr;
 struct sockaddr_in client_addr;
 ficlSystemInformation myFiclInfo;
 system[STARTUP] = ficlSystemCreate(NULL);
 ficlSystemCompileExtras(system[STARTUP]);
 buildMyInterface(system[STARTUP]);
 vm[STARTUP] = ficlSystemCreateVm(system[STARTUP]);
 signal(SIGINT,catcher);
 returnValue = ficlVmEvaluate(vm[STARTUP],
                              ".ver .( " __DATE__ " ) cr quit");
 /* cmd line argument for socket connection ? */
 if (argc > 1) {
  selstr = argv[1];
  netuse = strcmp(selstr, "tcp");
 }
 else {
  netuse = -1;
 }
 /* Without TCP/IP ? */
 if(netuse != 0) {
   /* stdin/stout loop */
   while (returnValue != FICL_VM_STATUS_USER_EXIT) {
     fputs(FICL_PROMPT, stdout);
     fgets(buffer, sizeof(buffer), stdin);
     returnValue = ficlVmEvaluate(vm[STARTUP], buffer);
   }
 }
 /* Remote via TCP/IP */
 else {
   /*
    * Open a TCP socket (an Internet stream socket ).
    */
   if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
     fprintf( stderr, "server: can't open stream socket !");
   /*
    * Bind our local address so that the client can send to us.
    */
   bzero( (char *) &server_addr, sizeof(server_addr));
   server_addr.sin_family      = AF_INET;
   server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
   server_addr.sin_port        = htons(SERV_TCP_PORT);
   if(bind(sockfd, (struct sockaddr *) &server_addr,
           sizeof(server_addr)) < 0)
     fprintf( stderr, "SERVER: can't bind local address !");
   listen(sockfd, 5);
   while (returnValue != FICL_VM_STATUS_USER_EXIT) {
     /*
      * Wait for a connection from client process.
      * This is an example of a concurrent server.
      */
     clilen = sizeof(client_addr);
     vmSockPar.newsockfd = accept(sockfd, \
                   (struct sockaddr *)&client_addr, \
                   &clilen);
     if (vmSockPar.newsockfd < 0)
       fprintf( stderr, "SERVER: accept error !");
     /* Child process successfully created ? */
     if( (pid = fork()) < 0)
       fprintf( stderr, "server: fork error !");
     /* child process ? */
     else if (pid == 0) {
       /* close listening socket */
       close(sockfd);
       /* process the requests */
       returnValue = myVmSocketConnection(vm[STARTUP],
                                          vmSockPar.newsockfd);
       exit(returnValue);
     }
     else {
       pid = wait(&state);
       /* child process terminated with error ? */
       if(WIFEXITED(state) == 0) {
         close(vmSockPar.newsockfd);
       }
       else {
         /* 8-bit equality ? */
         if(WEXITSTATUS(state)==(FICL_VM_STATUS_USER_EXIT&0xFF))
           returnValue = FICL_VM_STATUS_USER_EXIT;
       }
     }
   }
 }
 ficlSystemDestroy(system[STARTUP]);
 return 0;
}
```

Sicherlich kann man das eleganter ausführen, aber prinzpiell ist zu erkennen, dass nur die Parameter der der C–Funktionen FGETS und FPUTS unterschiedlich sind.
Ausgewählt wird mit dem Kommandozeilenparameter "*tcp*", welcher Ein–/Ausgabepfad genutzt wird. Existiert "*tcp*", dann wird die Socket–Verbindung aufgebaut und eine entsprechende Ein–/Ausgabe–Schleife über die C–Funktion "MYVMSOCKETCONNECTION" aufgerufen. In dieser Funktion ist auch ein Beispiel, wie die Ausgabe einer virtuellen Maschine zugeordnet wird.
Fehlerbetrachtungen und weitere Dinge sind hier nicht einbezogen.

Der Bootloader und FICL

FreeBSD hat als Bestandteil seines Boot–Loaders FICL integriert. Nicht anders als bei anderen Betriebssystemen in der x86–Welt besteht das Problem, dass im Master–Boot–Record nur 446 Bytes zur Verfügung stehen, und so wurde in BSD folgender Ablauf während des "Bootstrap–Vorgangs" festgelegte:

1. **boot0** 446 Byte Boot–Manager im MBR
2. **boot1** 512 Byte Programm im Boot–Record eines Slices, einer Partition, welches Informationen enthält und boot2 sucht
   boot2 Schnittstelle, welche den Loader oder einen Kernel laden kann.
3. **Loader** wie es das BSD–Handbuch beschreibt — *eine schöne und einfach zu bedienende Boot–Konfigurations–Schnittstelle* — oder man kann auch FICL sagen.

Loader–Ablauf

Der Loader sucht während seiner Initialisierung nach Konsolen und Laufwerken, findet heraus, von welchem Laufwerk er gerade bootet und setzt dementsprechend bestimmte Variablen. Dann wird ein Interpreter gestartet, der Befehle interaktiv oder von einem Skript empfangen kann.

Danach liest der Loader die Datei /boot/loader.rc aus, welche ihn standardmäßig anweist, /boot/defaults/loader.conf zu lesen, wo sinnvolle Standardeinstellungen für diverse Variablen festgelegt werden und wiederum /boot/loader.conf für lokale Änderungen an diesen Variablen ausgelesen wird. Anschließend arbeitet dann loader.rc entsprechend dieser Variablen und lädt die ausgewählten Module und den gewünschten Kernel.

In der Voreinstellung wartet der Loader 10 Sekunden lang auf eine Tastatureingabe und bootet den Kernel, falls keine Taste betätigt wurde. Falls doch eine Taste betätigt wurde, wird dem Benutzer eine Eingabeaufforderung angezeigt. Sie nimmt einen einfach zu bedienenden Befehlssatz entgegen, der es dem Benutzer erlaubt, Änderungen an Variablen vorzunehmen, Module zu laden, alle Module zu entladen oder schließlich zu booten bzw. neu zu booten.

Ich habe hier mal den Auszug aus dem BSD–Handbuch übernommen, da dieser eigentlich schon alles erklärt.

Wissenswert in diesem Zusammenhang ist noch, dass der Loader in einem ***B**oo**T** e**X**tender (BTX) ausgeführt*

wird. BTX ist ein Code–Teil, der den Prozessor in den Protected Mode umschaltet, Descriptor–Tabellen anlegt und eine SysCall–Möglichkeit schafft. BTX ist somit eine Schutzumgebung für das Programm. [4]

Was geht und was geht nicht

Wie in meinen ersten Ausführungen schon dargelegt, ist fast alles, was ANS–FORTH so bietet, auch in FICL vorhanden. Ein paar kleine Einschränkungen gibt es jedoch.

Die Worte KEY? und KEY existieren normal nicht, sind aber z.B. im BSD–Loader durch den Loader, der dann FICL einbindet, realisiert.

Ein FORTH–Wort in C

Auf meinem System habe ich z.B. KEY? mal eingebunden und wer die unterschiedlichen Möglichkeiten in .nixen und Linux kennt, der weiß, welche Herausforderung das werden kann. Ich habe es mal — nicht portabel — realisiert und möchte damit auch ein Beipiel aufzeigen, wie z.B. eine C–Funktion in FICL als FORTH–Wort eingebunden werden kann.

1. Code des Wortes/Primitives

```
/*****************************************************************
** functions used by primitives
*****************************************************************/
static int arrived_char = -1;
int availchar(void) {
   char buffer[1] = "";   /* Buffer bereitstellen      */
   struct termio old,new; /* struct termio in termio.h */
   int flags, counted, rval; /* deklariert */
   if (arrived_char != -1)
      return FICL_TRUE; /* char from previos keypressed() */
   /* waitchar OFF */
   ioctl(0,TCGETA,&old); /* Alte Parameter holen */
   new = old; /* und merken */
   new.c_lflag = ~(ECHO|ICANON); /* Echo ausschalten */
   new.c_cc[VMIN] = 0; /* Pufferung aus */
   new.c_cc[VTIME] = 0; /* Warte 0/10 s auf Zeichen */
   ioctl(0,TCSETA,&new); /* Neue Werte setzen */
   counted = read(0, buffer, 1);
   if (counted==1) {
      arrived_char = buffer[0];
      rval = FICL_TRUE;
   }
   else
      rval = FICL_FALSE;
   /* waitchar ON */
   ioctl(0,TCSETA,&old); /* Alte Werte setzen, haengt bash */
   return rval;
}
```

2. FORTH primitive

```
/*****************************************************
**          key? - check for a character from stdin (FACILITY)
**
** key? ( – flag )
**
*****************************************************/
static void ficlPrimitiveKeyQuestion(ficlVm *vm) {
#if FICL_ROBUST > 1
   FICL_STACK_CHECK(vm->dataStack, 0, 1);
#endif
   /* But here do the right thing. */
   ficlStackPushInteger( vm->dataStack, availchar() );
   return;
}
```

3. FORTH–Wort erstellen

```
/*
 * Corresponding ficlBuild  calls...
 */
ficlBuild(pSys, "KEY?", ficlPrimitiveKeyQuestion, FICL_WORD_DEFAULT);
```

Nicht ganz ausgereift

Nach einigen Experimenten bin ich auf ein Problem mit doppeltgenauen Stackwerten gestoßen. Meines Wissens erzeugt die Eingabe einer Zahl mit einem Punkt eine doppeltgenaue Zahl auf dem Stack, welche ihren high–Anteil auf dem TOS liegen hat und den low–Anteil auf dem NOS.

Leider war die Reihenfolge vertauscht und ich habe mich bemüht, eine Lösung über ?NUMBER zu realisieren. Die Lösung funktioniert erst einmal, aber auch die Tatsache, dass der Punkt an beliebiger Stelle in der Zahl stehen kann, ist nicht realisiert und bedarf noch mehr Aufwand. Darüber hinaus überschaue ich noch nicht alle Varianten, welche durch solch eine Änderung eintreten können.
Ich nahm daraufhin Kontakt mit John Sadler auf, bekam aber bedauerlicherweise die Antwort, dass er gegenwärtig nicht mehr aktiv entwickelt — okay — kommt Zeit, kommt Lösung. Es ist ja *open source*.

Fazit

FICL ist, auch wenn noch nicht alles ausgereift ist und angewendet wurde, ein sehr elegantes gut beschriebenes Software–Tool, welches gerade im Embedded–Bereich mit vorhandenen 32-Bit–Betriebssystemen sehr nützlich ist. Es überzeugt besonders durch seine Kompaktheit und die vorhandene Geschwindigkeit.

[4] Sollte sich jemand an meinen Leserbrief aus der VD 4/2006 "*Die Anwendung – Automatisierung*" erinnern, so war dort der Gedanke aufgekommen, ein FORTH für Roboter zu verwenden. Ich denke, mit diesem Ansatz des BSD–Loaders sind schon ein paar kleine Voraussetzungen gegeben, welche aufgegriffen werden können.

Bootmanager und FAT-Reparatur: Vierter Fort(h)schritt (patchbares BIOS im RAM)

Fred Behringer

Im vorliegenden Artikel werden einige Forth–Worte entwickelt, mit deren Hilfe man eine *Hilfsdiskette* so präperieren kann, dass sie ganz früh in den Bootprozess eingreift, so früh, dass es z.B. möglich wird, das ROM–BIOS des PCs ins RAM zu kopieren und es ab dann (bis zum *Abschalten*) vom RAM aus wirken zu lassen. Das so erzeugte RAM–BIOS kann auch gepatcht werden. Die Forth–Worte aus den bisherigen Teilen 1 bis 3 dieser Artikelserie werden mit der Ausnahme des Puffers `sectbuf` hier nicht benötigt: `sectbuf` wird der Vollständigkeit halber hier noch einmal angegeben, und zwar genau so, wie er in Teil 1 bereits eingeführt wurde.

Mein Ziel

Alle bisherigen Vorhaben meiner Serie waren systemnahe. *Normalerweise* würde man dabei an ein Programmieren in Assembler denken (ich betrachte PC–Kompatible und gehe von mindestens dem 80486–Befehlssatz aus). In dem Assembler–Programm, auf das ich mich hier stark beziehe (siehe [WL]), steht „Ein dreifach Hoch auf die Syntax des MASM!“, und der betreffende Autor freut sich darüber, dass der genannte Assembler den speziellen Trick, den er sich vorstellt, tatsächlich mitmacht.

Sklave des Assemblers

In Forth hat der Anwender überraschend viele Möglichkeiten. In Forth braucht man sich nicht zum Sklaven eines unveränderbaren kommerziellen *Assemblers* machen zu lassen. In Forth ist der Assembler Teil des Systems. Eine der Stärken von Forth–Systemen liegt darin, dass sie einen fließenden Übergang vom Assembler zum Assemblat (und von da zu Forth und zurück, auch mitten aus dem Betrieb heraus) gestatten. Solcher Programmier–Freiheiten wegen beschäftige ich mich (als Amateur) mit Forth.

In den Bootprozess eingreifen

Ich muss zugeben, dass ich eigentlich über das gesetzte Ziel hinausschieße: Ich wollte mir schnell eine Forth–Möglichkeit verschaffen, mit der ich zu einem sehr frühen Zeitpunkt in den Bootprozess eingreifen kann. Dazu hätte weitaus weniger Aufwand genügt als der, den ich im unten stehenden Listing treibe. Aber die (hier weniger betonte) eigentliche Idee, das BIOS *on the fly* zu patchen, faszinierte mich seit eh und je. In der CHIP–Ausgabe 3/2009 wird auf dem Titelblatt vom *Update–Wahnsinn* gesprochen, und davon, was *gegen unsinnige Updates* hilft. Dem stimme ich zu. Ich darf den Gedanken aber gleich auf BIOS–Updates ausdehnen: Mir gefällt es nicht, wenn ich auf BIOS–Sonderwünsche die stereotype Antwort bekomme: „Ja, da müssen Sie im Internet nach einem BIOS–Update suchen.“Für wohlvertraute Computersysteme, die mir unschätzbare Dienste leisten, die aber von keiner Firma mehr herstellt werden? Ich möchte mir meine *Updates* schnell selbst zurechtfeilen.

Frage an mich selbst

In meiner Rezension des Vijgeblaadjes 65 (VD–Heft 3+4/2007) habe ich folgende Frage gestellt: Gibt es eine Möglichkeit, dieses BIOS (es war von *One Laptop per Child* die Rede) zeitweilig (so wie bei Zeichensätzen), einem Schatten–BIOS ähnlich, ins RAM meines Rechners zu schalten? Am liebsten über einen Bootmanager (bei mir XFDISK [PM]). Carsten Strotmann nahm meine Frage zum Anlass, einen bereits in seiner Schublade schlummernden sehr interessanten Bericht über seine Arbeiten (VD 2/2008 [CS]) zu veröffentlichen. Friederich Prinz [FP] hat gezeigt (VD 3/2008), wie aufmerksam die VD–Leser das Berichtete verfolgen. Er wies darauf hin, dass bei der gemeinsamen Beschäftigung mit Forth in den VD–Artikeln auch der Lern– und Lehreffekt, die gegenseitige Anregung, eine nicht zu unterschätzende Rolle spielt. Meine Frage jedenfalls (siehe oben) war ernst gemeint. Sie war aber in erster Linie auch an mich selbst gerichtet: Schaffe ich es, auch als Nichtfachmann? Antwort: In Forth, ja.

Der NEAT–Chipsatz

Zu Zeiten des 80286/80386 mit dem NEAT–Chipsatz gab es die Möglichkeit, das ROM–BIOS ins darüberliegende RAM zu kopieren. Es war aber schwierig, den dann zumeist einsetzenden Schreibschutz auszuhebeln, wenn man die Gelegenheit nutzen wollte, ins BIOS ändernd einzugreifen. In der c't 8/1989 steht ein Artikel von Martin Ernst [ME] über gerade diese Dinge. Sein Programm `RAMADAP.ASM` findet sich auf der c't–PC–Sammeldiskette 20 [SD]. Zwar war sein Ziel nicht unbedingt die Verlegung oder/und Änderung des BIOSystems, sondern eigentlich *nur* die Ausnutzung brachliegenden Hintergrund–RAMs zur Erweiterung des verfügbaren Arbeitsspeichers. Aber das geht ja von den Programmiertechniken her alles Hand in Hand: Auch bei Martin Ernst spielte die Neufassung des Bootsektors eine beträchtliche Rolle. Ich hatte zwar beim 80286 auch mit `RAMADAP` gearbeitet, hatte aber damals (als Amateur) nicht den geringsten Mut, am Programm selbst herumzubasteln. Heute (immer noch als Amateur), da ich in Forth einen starken Rückhalt habe, traue ich mir mehr zu.

BIOS ins RAM

Und wie ist es jetzt? Ich kann bei meinem AMD–K6–2 (mit dem zugeordneten Chipsatz) die Bereiche `C80000` bis `DFFFFF` *cachen*, aber komme ich so schon ans BIOS und kann dort Veränderungen vornehmen? Beim Stöbern bin ich auf einen uralten Gegenstand meines Interesses gestoßen: Auf das (Assembler–)Programm `RAMBIOS.ASM` von Wolfgang Lorenz aus dem Jahre 1991 (Zeitschrift TOOL [WL]): Bei diesem wird vom DOS–RAM–Grundbereich ein gewisser Teil abgezwackt, in welchen dann das ROM–BIOS kopiert wird. Dann werden alle Interrupt–Vektoren, die ins BIOS (ab Segment `F000`) zeigen, auf den abgezwackten Bereich verbogen (Stack und andere Dinge werden angepasst) und schließlich wird ein Soft–Reset eingeleitet, der insofern keinen Schaden anrichtet, als er das BIOS da belässt, wo es ab dann liegt, nämlich im abgesonderten RAM–Bereich. Damit das funktioniert, wird ein Trick angewandt: Das BIOS–Kopieren und das Umbiegen der PC–Interrupt–Vektoren geschieht aus dem Bootsektor einer speziell dafür präparierten Diskette heraus. Das ist dann im Bootvorgang des PCs früh genug, um bei Aufruf des Reset–Interrupts `19h` das ins RAM kopierte BIOS und die verbogenen Interrupt–Vektoren nicht mehr anzutasten. Das *Kaltstart–Flag* `0000:0472` wird dabei vorher auf `1234h` gesetzt. Warum das dann später, bei schon aufgerufenem DOS, mit einem erneut getätigten Reset nicht mehr geht, wird mich als Nächstes interessieren. Etwas eigenartig ist das schon, da ja andererseits der wiederholte Einsatz der Hilfsdiskette beim Zusammenspiel mit dem Bootmanager XFDISK (siehe unten im Abschnitt Kaskadierung) bestens funktioniert.

Programm von Wolfgang Lorenz

Im ASM–Quelltext des Programms steht:
Copyright: 1991 TOOL. Autor: Wolfgang Lorenz

Gemeint ist mit großer Wahrscheinlichkeit die Zeitschrift TOOL aus dem Verlag Vogel & Partner, die 1991 im vierten Jahr ihres Erscheinens stand. Und ich vermute, dass das Programm von der Begleitdiskette eines der Hefte dieser Zeitschrift stammt. Computerhefte, Programme und Disketten sind sehr kurzlebig. Ich konnte jedenfalls in meinen Archiven das gesuchte TOOL–Heft nicht mehr finden. Andererseits haben meine IOMEGA–ZIP–Disketten, allen Unkenrufen zum Trotz, ihre Inhalte noch nicht verloren. Und genau da konnte ich das besagte Programm von Wolfgang Lorenz wiederaufspüren.

Das Programm liegt mir als Assembler–Quelltext und als Exe–Datei vor. Die Exe–Datei funktioniert prima und ich hätte damit arbeiten können. Was mich stört, ist nur der Umstand, dass ich gern alles in Forth–Assembler vor mir haben würde. Ich möchte ja mit den üblichen (interaktiv leicht zu handhabenden) Forth–Mitteln nach Belieben selbst am Programm herummodifizieren dürfen.

Programm als Hex–Listing

Ursprünglich wollte ich also das Programm von Wolfgang Lorenz 1:1 nach Forth umsetzen. Dann wurde mir aber bewusst, dass ich in Forth ja *gemischt* arbeiten kann, hier ein bisschen Forth–Assembler, da ein bisschen High–Level–Forth. Das wäre dann aber im Programm von Wolfgang Lorenz auf Abkürzungen einerseits und Anpassungen andererseits hinausgelaufen. Also habe ich mich dann doch dazu entschlossen, den (an sich ja nicht sehr großen) Teil des Programms von Wolfgang Lorenz, der in den Bootsektor der anzufertigenden Hilfsdiskette geschrieben werden soll, in uralter Heimcomputer–Manier als Hex–Listing zu fassen. Abgetippt braucht ja heutzutage sowieso nichts mehr zu werden: Die Zeitschrift *Vierte Dimension* (VD) steht im Internet als PDF–Datei zur Verfügung und von dort lässt sich jeder Textteil einer PDF, also auch die besagte Hex–Tabelle, (mit dem Textwerkzeug des Acrobat–Readers) leicht extrahieren und über die Windows–Zwischenablage mit Hilfe von NotePad in eine Textdatei schreiben. Andererseits war man sich ja auch zu Zeiten der damals überall zu findenden Turbo–Pascal–Programmierung nicht zu fein, Inline–Hex–Code einzuschieben (vergleiche die semiprofessionellen Listings in den diversen CHIP–Special–Turbo–Pascal–Sonderheften so um 1990/91 herum); Inline–Hex–Code, den man beim Lesen kaum auf Anhieb verstehen kann und den man beim Programmieren sicher nicht immer auswendig weiß.

Abgesehen vom *vertrauensvollen* Übernehmen des Programms von Wolfgang Lorenz in Hexform blieb für Forth, zumindest bei der Anfertigung der Hilfsdiskette genügend zu tun übrig.

Interpretativ

So, wie ich das Programm zur Erzeugung der Hilfsdiskette eingerichtet habe (siehe Listing), nämlich interpretativ schon beim Einlesen (per `INCLUDE` oder `FLOAD`), werden auf jeden Fall etwas mehr als `512d` Datenstack–Plätze (zur Zwischenspeicherung der Hex–Werte zeitweilig) benötigt. Das lässt sich natürlich sofort auch anders einrichten.

Welches Forth

Ich gehe von Turbo–Forth in der 16–Bit–Version [TF] aus und überprüfe alles auch noch mit ZF [ZF]. Beide Systeme fußen auf F83 und im *Grünen Buch* von Ronald Zech [RZ] ist auch der Assemblerteil von F83 gut beschrieben. Mit der besagten Vorgehensweise habe ich gleichzeitig einen Grund gefunden, die in den bisherigen Teilen 1 bis 3 meiner Serie entwickelten Forth–Worte um einige diskettensektorbezogene Worte zu ergänzen.

Kaskadierung der BIOS–Vorverlegung

Interessant war für mich eine Zufallsentdeckung beim Experimentieren. Ich arbeite, wie schon gesagt, mit dem Bootmanager XFDISK von Florian Painke und Ulrich Müller [PM]. XFDISK liegt im normalerweise nicht benutzten Bereich der Festplatte zwischen dem MBR und

dem Bootsektor der ersten primären Partition und bearbeitet nach dem Booten von der Festplatte die Partitionstabelle im MBR. So weit, so gut. Die hier anzufertigende Hilfsdiskette beeinflusst diesen Vorgang nicht: Ich kann im Bootmanager–Menü auch nach der Vorverlegung des BIOS–Bereichs über die Hilfsdiskette noch andere Betriebssysteme als DOS aufrufen. Es besteht aber am Anfang des Bootvorgangs im XFDISK grundsätzlich auch die Möglichkeit, durch Drücken von F1 oder F2 von einer im Laufwerk a: liegenden Diskette zu booten — vorausgesetzt, dort liegt überhaupt eine bootbare Diskette. Lasse ich nun nach dem ersten Wirken der Hilfsdiskette zur Vorverlegung des BIOS ins RAM die Hilfsdiskette im Schacht `a:` und drücke F1 (oder F2), dann werden weitere 64 KB vom System zur freien Verfügung gestellt und das BIOS wird ein zweites Mal vorverlegt — und das könnte dann *beliebig* oft wiederholt werden. Schon allein diese Erscheinung würde mich reizen, demnächst doch mal das Programm von Wolfgang Lorenz genauer unter die Lupe zu nehmen, es in Forth–Assembler zu fassen und dann die Möglichkeit eines schnellen Veränderns der entsprechenden Parameter (vielleicht sogar *on the fly*) auszuschöpfen — ohne mich mit der Frage, ob `TASM` oder `MASM` oder sonstwas für ein `ASM`, und wer was macht oder nicht macht, herumplagen zu müssen. In Forth bin ich auch in Bezug auf das Assemblieren mein eigener Herr.

Nachteil

Man wird immer wieder darauf aufmerksam gemacht, dass man sich ein Soft–Reset (ein Neubooten) des PCs auch mitten aus dem Programm heraus durch einen intersegmentellen Sprung an eine bestimmte Stelle im BIOS erzwingen kann. Da der Computer sowieso zurückgesetzt werden soll, ist es dabei natürlich egal, ob Sprung (über `jmp-far`) oder Unterprogramm–Aufruf (über `call-far`). In Forth (auf hex geschaltet) ließe sich das mit folgendem Wort bewirken:

```
code reset far 0 0ffff #) call next end-code
```

In meinem (ROM–)BIOS steht an der angesprungenen Stelle `EA5BE000F0`. In Turbo–Forth überzeugt man sich davon per `0ffff 0 80 ldump`. `EA` steht für *direkten Weit–Sprung* und die darauffolgenden vier Bytes stehen (Achtung: Little–Endian) für `F000:E05B`. Wendet man nun das BIOS–Verlagerungs–Programm von Wolfgang Lorenz an, so steht an der entsprechenden Stelle immer noch `F000:E05B`. Mit anderen Worten, kommen im ROM–BIOS absolute Bezüge auf sich selbst, also mit Segment `F000`, vor, dann werden diese im RAM–verlagerten BIOS nicht wieder auf sich selbst, also bei meinem System auf den Anfang des RAM–Segments `8FC0`, bezogen (das könnte man ja bei Verfeinerung des Verlagerungs–Programms von Wolfgang Lorenz auch noch ergänzen), sondern bleiben auf das ROM–BIOS–Segment `F000` eingestellt.

Vorteil

Im eben gebrachten Beispiel ist der genannte Umstand kein direkter Nachteil, da ja das ursprüngliche BIOS (das ROM–BIOS) sein Leben lang da bleibt, wo es ist. Ein Sprung (oder Call) nach `F000:E05B` löst also auch nach BIOS–Verlagerung immer noch einen Neustart des PCs aus. Andererseits kann ich aber gerade dieses Beispiel zum schnellen Aufzeigen der eventuellen Vorteile der BIOS–Verlagerung verwenden: Ein *ganz schnell* eingegebenes

```
cb 0ffff 0 lc!
```

verwandelt das `EA` (`jmp-far`) in ein `CB` (`retf`). Die Wirkung des eben eingeführten Forth–Wortes reset ist nach einer solchen Eingabe dann also aufgehoben — solange aus dem `CB` nicht wieder ein `EA` gemacht wird. Und das alles bei Bedarf mitten aus den (eigenen) Programmen heraus. Ob das nun wirklich ein stichhaltiges Argument–pro ist, bleibe dahingestellt (das eben eingeführte Forth–Wort reset könnte man ja mit viel einfacheren Mitteln, auch nachträglich noch, unwirksam machen). Mit diesem Beispiel ist aber jedenfalls gezeigt, dass das umkopierte BIOS (das RAM–BIOS) tatsächlich gepatcht, verändert oder auch neugestaltet werden kann — ohne auf ein BIOS–Update vom Hersteller warten zu müssen. Ein solches Update würde ich ja für ein BIOS aus dem Jahre 1999 sowieso nicht mehr bekommen. Eine neue Hauptplatine kaufen? Hm — ich hätte da ja ein paar solcher *neueren* Systeme. Aber wie versehe ich die mit EISA–Steckplätzen für meine Transputer–Karten.

Für größere BIOS–Patch–Vorhaben

stünde mir die oben erwähnte Möglichkeit zur Verfügung, die (schon vorliegende) Hilfsdiskette zweimal wirken zu lassen, um mir einen RAM–Freiraum von weiteren 64 KB (oder sogar mehr) zu verschaffen, der von der RAM–Verwaltung (bis zum *Abschalten*, will sagen, Neubooten) nicht mehr angetastet wird.

Ich fasse zusammen

(1) Bei Disketten scheint das Lesen und Schreiben von Sektoren nicht ohne Fehlerabfrage und eventuell mehrfachen Versuchen möglich zu sein (siehe Kommentare bei `getdiskbootsect` und `putdiskbootsect` im Listing).

(2) Das PC–BIOS kann über eine Hilfsdiskette in einen abgetrennten RAM–Bereich gelegt und von dort aus wirksam werden.

(3) Das aus dem RAM heraus wirkende BIOS kann auch gepatcht werden.

(4) Die besagte Hilfsdiskette wird mit Forth–Mitteln hergestellt und steht danach *für immer* zur Verfügung.

(5) Der Computer wird per Hilfsdiskette gebootet, der Bootvorgang leitet dann auf die Festplatte über.

(6) In Zusammenarbeit mit dem Bootmanager XFDISK kann der BIOS–Vorverlegungsvorgang auch wiederholt werden (Hilfsdiskette im Schacht lassen). Damit könnte dann mehr als nur ein einziges Stück RAM bis zum Abschalten freigemacht werden. Das überschüssige RAM könnte für größere BIOS–Patch–Vorhaben verwendet werden.

(7) Ein einfaches Rücksetzen des Computers über den Reset–Knopf, über einen intersegmentellen Sprung nach `ffff:0000` oder über den Interrupt `19h` (Affengriff) macht die BIOS–Verlegung ins RAM wieder rückgängig. Nimmt man die Hilfsdiskette dabei vorher aus dem Laufwerksschacht, dann sind die Verhältnisse wieder so, wie sie vom PC–(BIOS–)Hersteller eigentlich vorgesehen waren.

Literatur und Links

[CS] Strotmann, Carsten: Forth ohne/als Betriebssystem. VD–Heft 2/2008.
[FP] Prinz, Friederich: Leserbrief im VD–Heft 3/2008.
[ME] Ernst, Martin: Treiber verschiebt Treiber. c't 8/1989.
[PM] Painke, Florian, und Ulrich Müller: XFDISK.EXE. Public Domain.
[RZ] Zech, Ronald: Forth–83. Franzis–Verlag München (1987).
[SD] c't–PC–Sammeldiskette 20
[TF] Petremann, Marc: Turbo–Forth, Guide de référence. Editions REM CORP 1990.
Das System liegt u.a. auf dem amerikanischen FTP–Server taygeta.com.
[WL] Lorenz, Wolfgang: RAM–BIOS. Höchtwahrscheinlich Zeitschrift TOOL, 1991.
[ZF] Zimmer, Tom: ZF–System (16 Bit). Public Domain.

Listing

```
1   \ ******************************************************
2   \ *                                                    *
3   \ *  BOOTMA-4.FTH                                      *
4   \ *                                                    *
5   \ *  Zutaten fuer FAT-Reparatur und Bootmaster unter   *
6   \ *  Turbo-FORTH-83 und ZF                             *
7   \ *                                                    *
8   \ *  Fred Behringer - Forth-Gesellschaft - 10.5.2009   *
9   \ *                                                    *
10  \ ******************************************************
11
12  \ ======================================================
13  \ Bei Arbeiten mit ZF:
14  \ zf fload bootma-4.fth     -     .fth nicht vergessen!
15  \ attributs off wegnehmen!  attributs in ZF unbekannt.
16  \ Ansonsten scheint auch unter ZF alles zu gehen.
17  \ ======================================================
18
19  attributs off         \ Fuer den Fall, dass kein ANSI.SYS in der
20                        \ CONFIG.SYS ist. Oder dann aber ANSI.COM aus dem
21                        \ Internet hereingooglen und ins System (in die
22                        \ AUTOEXEC.BAT) legen! Bei Arbeiten mit ZF
23                        \ auf jeden Fall wegnehmen.
24  hex
25
26  210 allot here        \ Platz fuer mind. 1 Sektor = 512d Bytes
27  here 0f and -         \ sectbuf an Paragraphenanfang
28  200 -                 \ Anfang des Sektorpuffers
29  constant sectbuf      \ Liefert Adresse des Sektorpuffers
30
31  \ Sektor-Puffer am Bildschirm anzeigen
32
33  : showsectbuf         ( -- ) sectbuf 200 dump ;
34
35  \ Nur die ersten 100 Bytes davon anzeigen
36
37  : showsectbuf100      ( -- ) sectbuf 100 dump ;
38
39  \ ====================================================================
40  \ Vorweg: Alle Forth- und DOS-Befehle sind im Vorliegenden von Gross- und
```

```
\ Kleinschreibung unabhaengig. Das vorliegende Programm BOOTMA-4.FTH ist von
\ den bisherigen Programmen BOOTMA-1.FTH bis BOOTMA-3.FTH aus den VD-Heften
\ 3/2008 bis 1/2009 unabhaengig. Zur Herstellung einer Hilfsdiskette boote
\ man den PC von der Festplatte (beispielsweise unter DOS 6.2) mit dem
\ gewuenschten Forth und dem Programm aus dem vorliegenden Artikel. Man lege
\ die zu praeparierende Diskette ein und lade das vorliegende Programm (in
\ Turbo-Forth per include, in ZF per fload) ein. Danach ist die eingelegte
\ Diskette praepariert. Fuer eine unmittelbare Analyse stehen die Worte
\ showsectbuf und showsectbuf100 aus Teil 1 dieser Artikelserie und
\ natuerlich auch das in Forth eingebaute dump zur Verfuegung. Will man
\ daneben auch die bisher entwickelten Forth-Worte aus den Teilen 1 bis 3 der
\ vorliegenden Artikelserie zur Verfuegung haben, so wird man den oben
\ stehenden Programmteil (bis zur Definition von showsectbuf100), da in den
\ Teilen 1 bis 3 schon vorhanden, weglassen. Sowohl das ROM-BIOS im Segment
\ F000 wie auch das ins RAM kopierte BIOS lassen sich im System Turbo-Forth
\ bequem ueber das segmentuebergreifende ldump analysieren. Wie ZF-Fans das
\ machen, muss ich erst noch erkunden. Die zu erstellende Hilfsdiskette
\ braucht nicht formatiert, also auch nicht mit irgendeinem Betriebssystem
\ versehen zu sein (vergleiche aber unbedingt das weiter unten im Zusammenhang
\ mit DEBUG Gesagte). Zum Beweis habe ich eine 3.5"-Diskette (von 720 KB)
\ genommen und deren Bootsektor mit den Forth-Hilfsmitteln aus der
\ vorliegenden Artikelserie durchgaengig auf 0 gesetzt. Anschliessend habe
\ ich diese Diskette unter Turbo-Forth durch INCLUDEn des vorliegenden
\ Listings zu einer Hilfsdiskette (zum BIOS-Kopieren ins RAM) umfunktioniert:
\ Geht sehr gut. Allerdings nimmt es nicht wunder, dass man bei einer
\ solcherart fuer DOS unbrauchbar gemachten Diskette nicht mehr nach Laufwerk
\ a: schalten kann ('Allgemeiner Fehler beim Lesen von Laufwerk A'):
\ Die diskettenbezogenen (formatspezifischen) Parameter (fuer das
\ BIOS-Umkopieren werden sie nicht benoetigt) sind ja jetzt (nach dem 0-Setzen)
\ nicht mehr vorhanden. Nimmt man dagegen zur Anfertigung einer
\ Umkopier-Hilfsdiskette eine ganz normal von DOS anerkannte Diskette
\ (bootbar oder nicht), dann geht es. Ich habe das u.a. mit einer FreeDos-,
\ einer DOS-6.2- und einer OS/2-Warp-Diskette ueberprueft.
\ ========================================================================

code (getdiskbootsect) ( -- fl ) \ fl <> 0 --> Bootsektor-Lesefehler
           si push    \ si (= ip) momentan freigeben
       1 # di mov     \ Bei Aussprung: OK, wenn di = 0
       3 # si mov     \ Zaehler fuer drei Leseversuche ansetzen
           ds push    \ ds --> es
           es pop
    begin
           si dec     \ Schleifenzaehler um 1 herabsetzen
         ah ah xor
         dl dl xor
           13 int     \ Diskettensystem zuruecksetzen
        dx dx xor     \ dl = Diskette a:, dh = Seite 0
       1 # cx mov     \ Spur 0, Sektor 1
sectbuf # bx mov     \ bx auf den Anfang des Puffers setzen.
    201 # ax mov     \ Zum Lesen des Bootsektors
           13 int     \ Platten-und-Disketten-Interrupt aufrufen.
       u>= if         \ Falls cf = 0,
         si si xor    \ dann Aussprung vorbereiten
           di dec     \ und OK-Indikator auf 0 setzen.
           then
       0 # si cmp 0=  \ Sonst di = 1 belassen und Test auf Schleifenende
    until
           si pop     \ si (= ip) wiederherstellen
           di push    \ Stackwert <> 0 --> Leseversuche vergeblich
         next end-code
```

```
101
102 : getdiskbootsect ( -- )
103       (getdiskbootsect) if cr abort" Bootsektor-Lesefehler" then ;
104
105 \ getdiskbootsect ist eine Spezialisierung von (getsect) aus VD-Heft 3/2008.
106 \ Statt der Kennnummer der ersten Festplatte, also 80, wird die Kennnummer der
107 \ Diskettenstation a:, also 0, eingesetzt. (Fuer ein eventuell noch vorhandenes
108 \ Disketten-Laufwerk b: waere hier 1 einzusetzen.) Einen MBR gibt es bei
109 \ Disketten nicht. Der erste Sektor einer Diskette ist also der Bootsektor. Der
110 \ Bootsektor wird von getdiskbootsect in den Sektorpuffer sectbuf geschrieben
111 \ und kann dort weiterverarbeitet werden.
112
113 \ Fehlerfrage: Urspruenglich wollte ich in meiner Artikelserie der Einfachheit
114 \ halber eigentlich auf jedwede Fehlerabfrage verzichten. Schliesslich nimmt
115 \ man ja in Forth-Programmen auch bei arithmetischen oder logischen Operationen
116 \ fuer gewoehnlich keine Fehlerueberpruefung vor. Nun habe ich aber beobachtet,
117 \ dass bei einem vereinfachten getdiskbootsect (ohne Mehrfachversuchsschleife
118 \ und Diskettensystemruecksetzung) das Einlesen des Bootsektors nicht klappte -
119 \ und das nicht nur manchmal nicht, sondern nach Einschalten des Computers
120 \ immer nicht. Wurde dagegen getdiskbootsect anschliessend erneut aufgerufen,
121 \ also nach dem Einschalten ein zweites Mal, dann ging es - bei meinen
122 \ Versuchen 'immer'. Frage: Woran liegt das? Habe ich da vielleicht etwas
123 \ Grundsaetzliches nicht beachtet?
124
125 \ Das gleich folgende Forth-Wort putdiskbootsect mit dem dazugehoerigen
126 \ (putdiskbootsect) ist das Pendant zu getdiskbootsect. Alle Bemerkungen
127 \ uebertragen sich sinngemaess. Der Inhalt des Sektorpuffers sectbuf wird in
128 \ den Bootsektor der im Laufwerk a: liegenden Diskette geschrieben. Eine nicht
129 \ eingelegte Diskette oder eventuelle Schreibfehler werden durch die Meldung
130 \ 'Bootsektor-Schreibfehler' angezeigt.
131
132 code (putdiskbootsect) ( -- fl ) \ fl <> 0 --> Bootsektor-Schreibfehler
133           si push    \ si (= ip) momentan freigeben
134       1 # di mov     \ Bei Aussprung: OK, wenn di = 0
135       3 # si mov     \ Zaehler fuer drei Leseversuche ansetzen
136           ds push    \ ds --> es
137           es pop
138     begin
139           si dec     \ Schleifenzaehler um 1 herabsetzen
140        ah ah xor
141        dl dl xor
142           13 int     \ Diskettensystem zuruecksetzen
143        dx dx xor     \ dl = Diskette a:, dh = Seite 0
144       1 # cx mov     \ Spur 0, Sektor 1
145 sectbuf # bx mov     \ bx auf den Anfang des Puffers setzen.
146     301 # ax mov     \ Zum Beschreiben des Bootsektors
147           13 int     \ Platten-und-Disketten-Interrupt aufrufen.
148       u>= if         \ Falls cf = 0,
149        si si xor     \ dann Aussprung vorbereiten
150           di dec     \ und OK-Indikator auf 0 setzen.
151           then
152       0 # si cmp 0=  \ Sonst di = 1 belassen und Test auf Schleifenende
153     until
154           si pop     \ si (= ip) wiederherstellen
155           di push    \ Stackwert <> 0 --> Schreibversuche vergeblich
156         next end-code
157
158 : putdiskbootsect ( -- )
159       (putdiskbootsect) if cr abort" Bootsektor-Schreibfehler" then ;
160
```

```
161  \ Zunaechst einmal haette ich mir gern eine Colon-Definition gewuenscht, die
162  \ (beispielsweise beim Einlesen per include, bzw. fload) sofort ausgefuehrt
163  \ wird und dann wieder verschwindet. In Turbo-Forth hat das Namenlos-Wort ::
164  \ diese Wirkung. Das Wort :noname aus ANS-Forth waere zwar auch 'namenlos',
165  \ wuerde aber wohl nach seiner Ausfuehrung nicht mehr aus dem Dictionary
166  \ verschwinden. In ZF gibt es keines dieser beiden Worte. In ZF gibt es aber,
167  \ genau wie in Turbo-Forth, das Wort forget. Mit diesem baue ich mir eine
168  \ Hilfskonstruktion, die sowohl in Turbo-Forth wie auch in ZF verwendbar ist
169  \ und die die Wirkung von :: hat. (Statt xxx kann natuerlich jeder andere
170  \ noch nicht verbrauchte Name verwendet werden.)
171
172  \ Bootsektor der DOS-formatierten Hilfsdiskette nach sectbuf einlesen
173
174  getdiskbootsect
175
176  \ Code zum Ueberspringen von 1Eh Bytes an den Anfang von sectbuf schreiben
177
178  0eb sectbuf c!  01c sectbuf 1 + c!  090 sectbuf 2 + c!
179
180  \ Formatspezifische Parameterdaten der eingelegten Diskette ueberspringen und
181  \ dann den Rest des nach den Angaben von Wolfgang Lorenz aus der Zeitschrift
182  \ TOOL (1991) angefertigten Bootsektors nach sectbuf kopieren.
183
184                                                          033 0FF
185  0FA 08E 0D7 0BC 000 07C 0FB 0B8 000 010 08E 0C0 0BE 03D 07C 090
186  08E 0DF 0B9 058 001 0FC 0F3 0A4 0EA 000 000 000 010 033 0C0 08E
187  0D8 08B 01E 013 004 083 0EB 040 089 01E 013 004 0B1 006 0D3 0E3
188  08E 0C3 0BA 000 0F0 08E 0DA 033 0F6 033 0FF 0B9 000 080 0F3 0A5
189  08E 0DE 0AD 0AD 03B 0C2 075 003 089 05C 0FE 081 0FE 000 004 072
190  0F1 039 016 0AA 004 075 004 089 01E 0AA 004 0BE 0E0 000 08A 0C7
191  0E8 044 000 08A 0C3 0E8 03F 000 0BE 0AC 000 0E8 04F 000 033 0C0
192  08E 0D8 08E 0C0 0BF 003 000 0B8 001 002 0BB 000 07C 0B9 001 000
193  0BA 080 000 0CD 013 073 01B 032 0E4 0B2 080 0CD 013 04F 075 0E7
194  0BE 0F3 000 0E8 027 000 032 0E4 0CD 016 0BE 0F0 000 0E8 01D 000
195  0CD 019 0EA 000 07C 000 000 050 0B1 004 0D2 0E8 0E8 003 000 058
196  024 00F 004 090 027 014 040 027 02E 088 004 046 0C3 0B8 00D 00E
197  0CD 010 02E 0AC 022 0C0 075 0F8 0C3 044 061 073 020 052 04F 04D
198  02D 042 049 04F 053 020 062 065 069 020 024 046 030 030 030 020
199  077 075 072 064 065 020 061 06E 020 064 069 065 020 053 065 067
200  06D 065 06E 074 061 064 072 065 073 073 065 020 024 03F 03F 03F
201  03F 020 075 06D 06B 06F 070 069 065 072 074 02E 00D 00A 00A 000
202  04C 065 073 065 06E 020 064 065 073 020 042 04F 04F 054 02D 053
203  065 06B 074 06F 072 073 020 076 06F 06E 020 064 065 072 020 046
204  065 073 074 070 06C 061 074 074 065 020 06E 069 063 068 074 020
205  06D 094 067 06C 069 063 068 02E 00D 00A 053 079 073 074 065 06D
206  02D 044 069 073 06B 065 074 074 065 020 065 069 06E 06C 065 067
207  065 06E 020 075 06E 064 020 054 061 073 074 065 020 064 072 081
208  063 06B 065 06E 000 000 000 000 000 000 000 000 000 000 000 000
209  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
210  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
211  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
212  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
213  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
214  000 000 000 000 000 000 000 000 000 000 000 000 000 055 0AA
215
216  \ Hex-Listings wie diese waren zu Zeiten der Home-Computer en vogue. Inzwischen
217  \ sind sie verpoent. Die Tabelle der oben stehenden Hexzahlen schien mir aber
218  \ der schnellste Weg zu sein, an mein Ziel zu gelangen, naemlich mit
219  \ Forth-Mitteln eine Diskette herzustellen, die beim Hochfahren des PCs das
220  \ ROM-BIOS ins RAM kopiert und von dort aus zur Verfuegung stellt. Anliegen
```

```
\ im vorliegenden Artikel 4 meiner Serie war mir eigentlich die Bearbeitung
\ von Disketten-Bootsektoren mit Forth-Mitteln. Es haette nur unnoetig viel
\ Platz im VD-Heft gekostet, wenn ich, wie ich es urspruenglich vorhatte, den
\ Assembler-Quelltext des Konstruktions-Programms von Wolfgang Lorenz aus der
\ Zeitschrift TOOL (1991) hier zuvor erst nach Forth-Assembler uebertragen und
\ von dort dann in mein eigenes Hilfsdisketten-Erzeug-Programm eingebaut haette.
\ Bei Interesse kann man sich den Ueberblick ueber das hinter der Hex-Tabelle
\ stehende Assembler-Programm (nach erfolgreicher Erstellung einer
\ Hilfsdiskette) rueckwirkend mit dem DOS-Debugger DEBUG wie folgt verschaffen
\ (natuerlich ohne Kommentar-Schraegstriche!):

\ DEBUG
\ -L 0100 0 0 1
\ -D 0100 02FF
\ -U 0100 02FF
\ -Q

\ Man erhaelt damit die Bildschirmanzeige des Ganzen, den Hex-Dump und das
\ disassemblierte Programm. Schoen fuer mich (ich wollte mir das Eintippen
\ ersparen) fuer die halbautomatische Uebernahme des Hex-Dumps war die
\ Moeglichkeit, unter DOS das von -D erzeugte Debug-Elaborat in eine Textdatei
\ zu bannen (von wo aus ich die Hexwerte ueber ein x-beliebiges
\ Textverarbeitungssystem uebernehmen konnte), und zwar durch Eingabe von:

\ ECHO L 0100 0 0 1 >XXX.TXT
\ ECHO D 0100 02FF >>XXX.TXT
\ ECHO Q >>XXX.TXT
\ DEBUG <XXX.TXT >HEXDUMP.TXT
\ DEL XXX.TXT

\ Dabei gehe ich davon aus, dass die von Wolfgang Lorenz stammende Datei
\ RAMBIOS.EXE schon die Hilfsdiskette erzeugt hat und die Hilfsdiskette im
\ Laufwerk a: liegt. Die eben genannten fuenf Zeilen (natuerlich ohne die
\ Kommentar-Striche) sind in einer Batch-Datei zusammenzufassen und die
\ Batch-Datei muss dann von der DOS-Kommandozeile aus aufgerufen werden. Die
\ Hilfsdiskette muss fuer das Arbeiten mit DEBUG aus einer Diskette entstanden
\ sein, die (vorher schon) von DOS anerkannt wurde (vergleiche das am Anfang
\ dieses Listings Gesagte ueber mein Experiment mit dem 0-Setzen der
\ diskettenbezogenen Parameter im Bootsektor).

\ Die eben besprochenen Experimente mit DEBUG habe ich natuerlich nicht vom
\ Diskettenlaufwerk aus gemacht, sondern von Laufwerk c: (Festplatte). Zum
\ Abschluss des Themas DEBUG noch ein weiteres Experiment, mit dem dann alle
\ Eventualitaeten erlaeutert sein duerften: Ich habe mir per DISKCOPY eine
\ 1:1-Kopie einer DOS-6.2-Boot-Diskette hergestellt. Von der Kopie habe ich
\ einige DOS-Programme, die ich fuer mein Vorhaben bestimmt nicht mehr
\ benoetige, entfernt. Dann habe ich FORTH.COM (also Turbo-Forth),
\ BOOTMA-4.FTH (also das vorliegende Listing), DEBUG.COM (von DOS 6.2) und
\ die oben besprochene Batch-Datei (unter dem Namen DUMPBOOT.BAT) mit auf
\ die kopierte Diskette gepackt. Diese habe ich in den Diskettenschacht
\ gelegt und dann den Computer von Diskette gebootet. Von der kopierten
\ Diskette (also immer noch von Laufwerk a: aus) habe ich dann FORTH [ret]
\ und daraufhin INCLUDE BOOTMA-4 [ret] eingegeben - und siehe da: Im
\ Diskettenschacht lag dann eine praeparierte Diskette (eine 'Hilfsdiskette'
\ im oben erwaehnten Sinne), von der aus aber immer noch DOS-Programme
\ aufgerufen werden konnten. So konnte ich denn als Abschluss dieses
\ Experimentes durch Eingeben von DUMPBOOT den Bootsektor der kopierten
\ Diskette (eben der frisch gefertigten Hilfsdiskette) in die Datei
\ HEXDUMP.TXT schreiben und auf sich selbst (auf die Hilfsdiskette) speichern.

```

```
\ Nicht unschoen waere es fuer den vorliegenden Bericht gewesen, haette ich
\ mit den Stichwoertern 'TOOL', 'Wolfgang Lorenz' und 'RAM-BIOS' den
\ geneigten Leser einfach auf Google und das Internet verweisen koennen. Ich
\ hatte aber leider keinen Sucherfolg.

\ Das folgende Wort xxx legt die oben stehende Tabelle, die sich ja inzwischen
\ auf dem Forth-Datenstack befindet, ab dem Offset 1e in den Puffer sectbuf. In
\ Turbo-Forth haette man auch :: 1e2 0 do sectbuf 1ff + i - c! loop ; schreiben
\ koennen und waere dann ohne xxx und forget xxx ausgekommen. In ZF ist aber
\ kein :: zu finden.

: xxx 1e2 0 do sectbuf 1ff + i - c! loop ;  xxx forget xxx

\ Jetzt wird der so angepasste Puffer sectbuf als neuer Bootsektor in die
\ Hilfsdiskette geschrieben.

putdiskbootsect

\ Ab jetzt (in diesem Include-Vorgang) ist (und bleibt) das Praeparier-Programm
\ von Wolfgang Lorenz aus der Zeitschrift TOOL (1991) in den Puffer sectbuf
\ geladen und kann von dort aus (beliebig oft, zum Herstellen weiterer
\ Hilfsdisketten ;-) als neuer Bootsektor in eine anzufertigende Hilfsdiskette
\ geschrieben werden.

\ Der ganze Vorgang des Herstellens der Hilfsdiskette laeuft ab dem Einladen
\ des vorliegenden Forth-Programms (per include bei Turbo-Forth oder per fload
\ in ZF) ohne weiteres Zutun interpretativ ab. Will man vorsichtig (oder
\ neugierig) vorgehen, dann kann man putdiskbootsect auch zunaechst einmal per
\ '\' auskommentieren und es erst spaeter durch manuelle Eingabe von der
\ Forth-Kommandozeile aus wirken lassen. Ein INCLUDEn (FLOADen) des vorliegenden
\ Programms bei offengelassenem Disketten-Schacht hat dieselbe Wirkung.

\ Die Hilfsdiskette wird natuerlich normalerweise nur ein einziges Mal
\ hergestellt und leistet ab dann beliebig oft ihre oben erwaehnten Dienste:
\ BIOS ins RAM kopieren, Interrupts auf das RAM-BIOS ausrichten und PC von
\ der im BIOS-Setup eingestellten Festplatte booten. Dass man schon in diesen
\ Herstellungs-Prozess nach eigenem Gutduenken eingreifen kann, ist klar: Man
\ braucht ja 'nur' vor dem Uebertragen des Hilfsdisketten-Bootsektors (mit
\ Forth-Mitteln) den Inhalt des Puffers sectbuf geeignet zu veraendern. Dass
\ man das RAM-BIOS patchen kann, ist klar. Zum eventuellen Vergleich liegt das
\ ROM-BIOS ausserdem nach wie vor bei F000:0000 und ueber die Lage des RAM-BIOS
\ gibt die Bildschirm-Meldung, unmittelbar nachdem der PC ueber die Hilfsdiskette
\ gebootet wurde, Auskunft.

\ Vorsicht beim Patchen! Die 'verbogenen' Interrupt-Vektoren liegen (bis zum
\ 'Sitzungsende') festgezurrt und wuerden es uebelnehmen, wenn das RAM-BIOS an
\ irgendeiner Stelle auseinandergezogen werden wuerde.
```

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

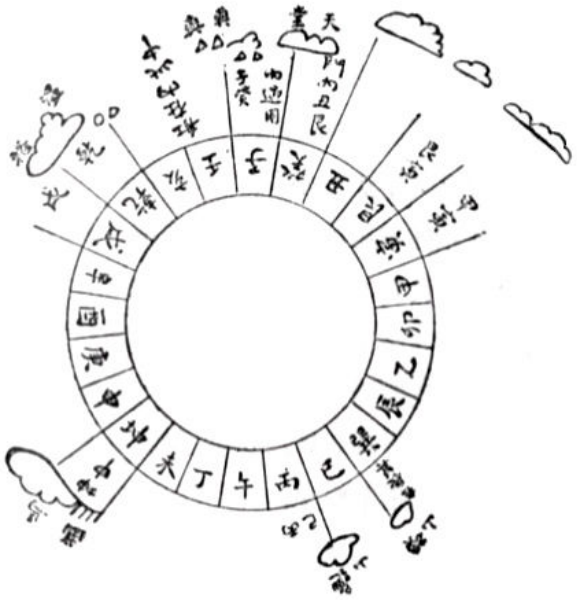
Carsten Strotmann
microcontrollerverleih@forth-ev.de
mcv@forth-ev.de

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.

EuroForth 2009
25th EuroForth Conference
University of Exeter, England
4th to 6th September, 2009

www.exeter.ac.uk

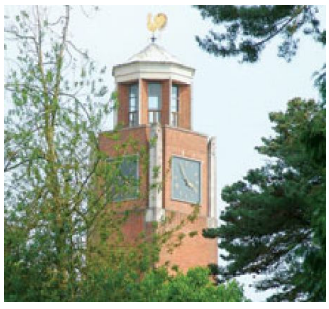
EuroForth is an annual conference on the Forth programming language, stack machines, and related topics, and has been held since 1985. The 25th EuroForth will be hosted by the School of Engineering, Computing and Mathematics at the University of Exeter. The conference will be preceeded by a Forth 200x standards meeting.

May 29:	Deadline for draft papers (academic stream)
July 1:	Notification of acceptance of academic stream papers
August 24:	Deadline for camera-ready paper submission (academic and industrial stream)
September 2-3:	non-voting Forth200x meeting
September 3-4:	voting Forth200x meeting
September 4-6:	EuroForth 2009 conference starts on 4th at 2pm, buffet provided before the start

Information on earlier conferences can be found at the EuroForth home page.

Links

- Call for Papers (`http://www.complang.tuwien.ac.at/anton/euroforth/ef09/cfp.html`)
- EuroForth 2009 Home page (`http://www.complang.tuwien.ac.at/anton/euroforth/ef09/`)
- EuroForth Home (`http://www.complang.tuwien.ac.at/anton/euroforth/index.html`)



University of Exeter (Photos: `www.exeter.ac.uk`)

This information can also be found at `http://www.complang.tuwien.ac.at/anton/euroforth/ef09/`.