Vierte Dimension

4/2008

Das Forth–Magazin

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

In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

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

Gegründet 1985 als Partnerinstitut der FH–Wedel beschäftigten wir uns in den letzten Jahren vorwiegend mit Industrieelektronik und Präzisionsmeßtechnik und bauen z. Z. eine eigene Produktpalette auf.

Know–How Schwerpunkte liegen in den Bereichen Industriewaagen SWA & SWW, Differential–Dosierwaagen, DMS–Messverstärker, 68000 und 68HC11 Prozessoren, Sigma–Delta A/D. Wir programmieren in Pascal, C und Forth auf SwiftX86k und seit kurzem mit Holon11 und MPE IRTC für Amtel AVR.

LEGO RCX–Verleih

Seit unserem Gewinn (VD 1/2001 S.30) verfügt unsere Schule über so ausreichend viele RCX–Komponenten, dass ich meine privat eingebrachten Dinge nun Anderen, vorzugsweise Mitgliedern der Forth–Gesellschaft e. V., zur Verfügung stellen kann.

Angeboten wird: Ein komplettes LEGO–RCX–Set, so wie es für ca. 230,– € im Handel zu erwerben ist.

Inhalt:

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

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

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

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl–Schöpe

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

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

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 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

Liebe Leser,

vor Euch liegt die vierte Ausgabe dieses Jahres. Ich würde mich freuen, wenn Ihr sie ebenso interessant findet wie ich.

Diesen Herbst hat die Firma IntellaSys `http://www.intellasys.net/` den SEAforth–40C18–Chip vorgestellt: eine größere Variante des SEAForth24–Chips mit 40 Kernen des von Chuck Moore entwickelten C18–Prozessors. Ich denke, es ist sehr interessant, sich genauer mit der Programmierung dieser Chips auseinanderzusetzen, die — ersten Eindrücken nach — ganz und gar anders ist, als herkömmliche Ansätze: ganz anders als übliche Programmierung von eingebetteten Einprozessor–Systemen in Assember oder C, aber eben auch ganz anders als die Programmierung in klassischem Forth. Der begrenzte Speicher und die begrenzten Stacks, die jedem Kern zur Verfügung stehen, zwingen einen, die Programme ganz unkonventionell zu strukturieren. Das ist alles andere als einfach und muss erlernt und geübt werden. Hier sehe ich Chancen und Herausforderungen für die Forth–Gemeinde. Wer hat Interesse und Lust, an einer diesbezüglichen Arbeitsgemeinschaft mitzuwirken? Wäre doch prima, könnten wir in den kommenden Ausgaben genauer über SEAforth berichten. Meldet Euch doch einfach bei mir.

Doch zurück zum vorliegenden Heft. Fred Behringer berichtet in seinem zweiten Artikel über Bootsektoren und Partitionstabellen, wie man von Forth aus auf diese lebenswichtigen Daten eines PCs zugreift. In einem weiteren Artikel hat er sich auch der Problematik angenommen, dass die bisher vorgestellten Lösungen für das Euler9–Problem ohne Weiteres nur auf 32–Bit–Forth–Systemen liefen: er präsentiert eine Lösung für die 16–Bit–Systeme Turbo–Forth und ZF. Im eingebetteten Bereich stellt Matthias Trute einen AVR–Assembler für sein amForth vor, der von Lobos Pekny entwickelt wurde. Erich Wälde schildert uns im fünften Teil seiner Abenteuerserie, warum und wie er vom Renesas–R8C–Controller auf Atmel–AtMega–Prozessoren umgestiegen ist, und dass dank Forth der Portierungsaufwand seiner Programme übersichtlich bleibt. Michael Kalus, auch aktiv in der amForth–Entwicklung, stellt seine Implementierung des Hex–Dumps vor und erläutert, wie er mit den verschiedenen Speicherbereichen im AVR umgeht. Ich selbst steuere zu dieser Ausgabe einen Bericht von der EuroForth–Tagung 2008 bei, die Ende September in Wien stattfand.

Apropos Tagung: Unsere Erfahrungen mit der Forth–Tagung 2007 in Wien waren so aufmunternd, dass erste Überlegungen laufen, eine weitere Tagung außerhalb Deutschlands zu veranstalten. Die engen Kontakte, die wir zu den niederländischen Forth–Freunden haben, legen nahe, nach einem geeigneten Ort dort zu suchen. Konkrete Pläne gibt es noch nicht, aber vielleicht hat der ein oder andere ja noch eine Idee, wie und wo so etwas stattfinden kann.

Auch die Forth–Tagung 2009 wirft ihre Schatten voraus. Die Einladung zur Tagung findet Ihr auf der hinteren Umschlagseite. Sie findet vom 27. bis 29. März 2009 in Neuenkirchen bei Rheine statt — einen Monat früher als gewöhnlich. Meldet Euch also bitte rechtzeitig an.

So — nun wünsche ich uns allen eine schöne Winterzeit und viel Energie für das kommende Jahr.

Ulrich Hoffmann <uho@forth-ev.de>

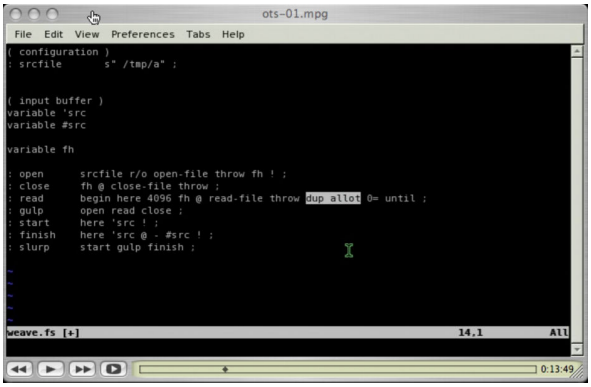
Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://www.forth-ev.de/filemgmt/index.php`

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

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

Beim Fortheln über die Schulter geschaut 1

Hättest du Lust dazu, jemandem der Forth kennt, bei der Arbeit über die Schulter zu schauen? Mit zu erleben, wie er denkt beim Entwurf des Programms? Neugierig, ob er dabei herumgeht und Kaugummi kaut oder nur kodiert und gleichzeitig erklärt, was er da tut? Wie oft sagt er „huch“oder „hmm“?



Nun, hier hast du die Gelegenheit dazu. Samuel A. Falvo II hat sein Video bereitgestellt, in dem er festhielt, wie er von einem blanken Gforth zu einem kleinen Programm gekommen ist, das eine breite Palette an Anforderungen für einen Text–Präprozessor erfüllt.

Link: `http://www.falvotech.com/blog/index.php?/archives/372-Over-the-Shoulder-1-Text-Preprocessing-in-Forth.html`

Quelle: `http://www.forth.com/`

Viel Vergnügen, Michael

Vijgeblaadjes und Forth von der Pike auf

Tag Fred,

mit etwas Verzögerung wegen eines kaputten PCs kommt hier die elektronische Version des kompletten Vijgeblaadjes 70.

Frans (van der Markt) und ich versuchen, das Vijgeblaadje im üblichen Rahmen von je zwei Monaten an jeden, der sich anmeldet, per E–Mail zu verschicken. Unsere (übergeordnete) Organisation HCC findet es zu teuer, jedesmal eine papierene Version per Post zu versenden. Es wird eingespart, wo es nur geht. Eine elektronische Version kostet halt nichts. Wir wissen noch nicht genau, in welcher Form und in welchem Format das neue Vijgeblaadje erscheinen wird. Die Forth–Gruppe (innerhalb der HCC) hat zwar noch genügend Geld in der Kasse, um (höchstwahrscheinlich) auch im Dezember noch ein Vijgeblaadje per Post zu verschicken. Das ist dann aber wohl die allerletzte (papierene) Ausgabe.

Ich finde es schön, dass du dir wieder so viel Mühe machst, die Forth–von–der–Pike–auf–Geschichte (diesmal Teil 11) zu übersetzen.

Ich bin momentan stark damit beschäftigt, die dazu vorgesehene Software (FIG–Forth++) mit den Hayes–Tests zu überprüfen. Es stellen sich immer noch (kleine) Fehler heraus. Wenn das Ganze durchgetestet ist, gebe ich dir Bescheid.

Bis zum nächsten Mal herzliche Grüße von Ron

Übersetzt von Fred Behringer

Neues vom R8C/13

Burkhard Kainka: Basiskurs R8C/13 Elektor–Verlag, Süsterfeldstr. 25, 52072 Aachen. 39,80 Euro.

Gutes Reservoir für den Bastler. Beschreibungen, Abbildungen und Schaltpläne zu 48 Projekten aus dem R8C–Wettbewerb vom Mai 2006. Weitere Details auf der CD, die dem Buch beiliegt.

Aus der Sicht des Forth–Begeisterten fehlt etwas: Die Bemühungen von Bernd Paysan und anderen um Forth für den R8C. Schade, nachdem Forth in den bisher erschienenen zwei Microcontroller–Sonderheften des Elektor–Verlags so gut angelaufen ist! Aber was nicht ist, kann ja noch werden. Wahrscheinlich hätte man ein weiteres Buch gebraucht, um dem Leser die Programmierung des R8Cs in Forth näherzubringen. Es werden 5 Bücher des Autors zu verwandten Themen aus der Elektronik zitiert. Außerdem werden 6 relevante Internet–Adressen angegeben.

Eine davon ist die der Firma Reichelt (Elektronikring 1, 26452 Sande). Im Reichelt–Katalog 5/2008 wird auf den Seiten 312 und 352 unter der Bestellnummer EVB R8C13 das Elektor–Controller–Board auf Basis des RENESAS R8C/13 für 11,95 Euro angeboten.

Zu den Wettbewerbsprojekten finden sich auf der Begleit–CD je eine PDF–Datei mit dem Beschreibungstext und eine (in den meisten Fällen sehr reichhaltige) ZIP–Datei mit den sonstigen Unterlagen und Programmen. Zu einigen Projekt–Beschreibungen braucht man holländische, zu anderen französische Sprachkenntnisse. Und selbstverständlich geht es nicht ohne Englisch ab (schon die extrem ausführlichen Data–Sheets von RENESAS sind ja englisch gehalten).

Und was, wenn sich FG–Mitglieder fänden, die die Programme (oder einige davon) aus den Projekten nach Forth übersetzten? Übersetzen macht doch auch Spaß, und man lernt für den eigenen Gebrauch viel dabei.

Die einzelnen Kapitelüberschriften des Buches lauten:

Die ersten Schritte
Das Application–Board
Erste Software–Experimente
R8C/13–Schnittstellen
Der R8C/13 als Oszilloskop
Das Grafik–LCD
Der I2C–Bus
Ein universelles PC–Interface
HF–Messlabor mit den R8C/13
HF–Generatoren und Empfängersteuerung
Projekte aus dem R8C–Wettbewerb

Fred Behringer

Forth–Tag im Silicon Valley

Dear Fred,

du wirst dich sicher für den Forth–Day–Report von Jason Damisch interessieren? Ich habe ihn gerade unter meinen SVFIG–E–Mails gefunden. `http://www.jasondamisch.net/forth/forth-day.html` . Jason ist ein neuer Tagungsbesucher. Sein Bericht ist sehr viel technischer, als ich ihn je verfassen könnte. Die Bilder sind von ihm selbst

aufgenommen. Die Übertragung ins Deutsche wird wohl ohne Zusatzerklärungen nicht ganz einfach sein, aber die VD–Leser werden sich zumindest darüber freuen zu wissen, dass und wo dieser Bericht im Web steht.

Grüße, Henry

Heft 2008/3

Der Fotograf und Künstler Theo Windges hat uns freundlicherweise erlaubt, Bilder aus seinem Zyklus „Die Zeit“ in der Vierten Dimension zu drucken, über sein Belegexemplar schreibt er uns:

Hallo Herr M. Kalus,

Ja. Das Heft ist angekommen und ich bin begeistert über diese Art und Weise mit der Zeit umzugehen. Als visueller Mensch mit einem Gespür für Bild und Wortspiele bin [ich] beeindruckt über soviel intellektuelles Wissen über die Zeit.

Theo Windges der Zeitbildermacher.

Kurzvorstellung — CHForth

Was ist CHForth?

CHForth (van Coos Haak) ist eine 16–bit–ANSI–Standard–Implementierung für Intel 80x86–Processoren unter MSDOS oder DRDOS. Es lauft im real mode (16–bit–8086) und verwendet mehrere Segmente, wodurch es viel mehr Speicher bereitstellt, als F83 oder die MS-DOS Version des ciforth. Mit CHForth können turnkey-(selbststartende)-Programme geschrieben werden. Der Gebrauch von CHForth ist frei, auch für kommerzielle Zwecke. Die Dokumentation liegt als PDF–Datei vor. Das Forth selbst kann exclusive Dokumentation herunter geladen werde. Für eventuelle Fragen und Hilfen, kann mit Coos Haak Kontakt aufgenommen werden . Für dieses gratis bereitgestellte System gibt der Autor allerdings keine Garantien.

Was beinhaltet CHForth?

Online–*help* für alle Forth–Worte (auch für viele andere Begriffe). Ein 80x86–Assembler, ein simples DOS–Interface, ein Zeileneditor in dem zurückgeblättert werden kann. Lokale Variablen, umfangreiches Abfangen von Fehlern, `TO` für alle Datatypen. Interrupt–Unterstützung, einen schicken Decompiler und Disassembler, einen umfangreichen Satz an Bibliotheks–Dateien, turnkey–Vorbilder, usw.

Ferner gibt es Hilfe im Internet: `http://home.hccnet.nl/j.j.haak/CHForthHelp/index.htm`

Webseite des Autors: `http://home.hccnet.nl/j.j.haak/`

Quelle: `http://www.forth.hccnet.nl/`

Im Laufe der Zeit
Bild aus dem Zyklus „Die Zeit“ von Theo Windges

Assembler in amforth

Matthias Trute

Zusammenfassung

Amforth verfügt über einen in Forth geschriebenen Assembler, der von Lubos Pekny programmiert und der Community bereitgestellt wurde. Um seine Arbeit zu unterstützen, erhielt das Kernsystem die Worte `code` und `end-code`. Dieser Artikel soll Lust auf mehr machen, ohne alle Details zu beleuchten.

Jedes Forth braucht einen Assembler. Heißt es. Bei amforth ist der Assembler ein optionales und nachladbares Modul und ist selbst in Forth geschrieben.

Das amforth–Kernsystem ist um zwei Worte erweitert worden, die für die Erzeugung der Codeworte und deren Einbindung in den inneren Interpreter zuständig sind: `code` und `end-code`. Jedes dieser Worte benutzt Informationen, die nur zur Übersetzungszeit des Quellcodes portabel verfügbar sind: die Einsprungadresse des inneren Interpreters `NEXT` und das in verschiedenen Definitionsworten genutzte `(DOCREATE)`, mit dem sich der Rumpf eines Dictionaryeintrags erstellen lässt.

Alle weiteren Assemblerbefehle sind als Forth–Code umgesetzt. Die Syntax lehnt sich dabei sowohl an die Konventionen des AVR–Assemblers als auch an Forth an. Dieser Artikel beruht auf der Version 1.1 des Assemblers.

Hier auch gleich eine Warnung: Der Einsatz des Assemblers erfordert etwas mehr als nur grundlegende Kenntnisse der Mikrocontroller und der Interna von amforth. Der Programmtext ist nicht portabel, etwa zu PC–Forths, und, wenn man nicht aufpasst, noch nicht einmal zwischen verschiedenen AVR–Controllertypen.

AVR Atmega

Die AVR–Atmega–Mikrocontroller der Firma Atmel sind kleine System–On–A–Chip–Bausteine die aus einem Kernsystem und einer Anzahl von Hardwaremodulen bestehen. Das Kernsystem umfasst den Registersatz, die Instruktionslogik, RAM, EEPROM und Flashspeicher. Je nach Typ sind unterschiedliche Speichervolumina vorgesehen, wobei RAM und EEPROM eher knapp bemessen sind (wenige KB), Flash dagegen vergleichsweise viel vorhanden sein kann (derzeit bis zu 256KB).

Die Architektur hat zudem die Besonderheit, dass direkt ausführbarer Maschinencode ausschließlich im Flash sein kann. Andere Speichertypen können dafür nicht genutzt werden.

Der Registersatz besteht aus 32 8–bit–Registern, von denen einige als 16–bit–Registerpaare genutzt werden können. In amforth werden 8 der 32 Register ausdrücklich für den Einsatz in Code–Wörtern bereitgehalten (Im Quellcode als temp1 bis temp8 bezeichnet). Diese können problemlos innerhalb eines Wortes benutzt werden. Alle anderen Register (und dazu gehören auch die 4 Registerpaare) müssen am Ende des Assemblercodes wiederhergestellt werden, sollten sie genutzt werden. Wirklich ungenutzt sind derzeit nur 4 Register, was sich aber im Zuge der bei der Euroforth2008 diskutierten Erweiterungen der Forth–VM ändern kann[1].

Der Befehlssatz orientiert auf Register/Register–Operationen. Dabei sind die meisten Befehle für alle oder viele Register einsetzbar. Einige wenige Befehle benutzen feststehende Register, etwa die Hardwaremultiplikation oder der Zugriff auf den Flash.

An dieses Kernsystem hat Atmel je nach Typ weitere Module angebaut. Dabei sind einige Module bei allen Controllern vorhanden (z. B. serielle Schnittstelle oder A/D–Wandler), andere Module sind nur bei einigen Typen vorhanden (CAN– oder USB–Schnittstellen, LCD–Ansteuerung).

Die Module werden über definierte (leider bei jedem Typ andere) Adressen für die Konfiguration und die Daten angesprochen. Einige Module bieten die Möglichkeit, Interrupts auszulösen, die beim Eintreffen der Interruptbedingung aktiviert werden. Genau diese Interrupts sind neben Geschwindigkeitsoptimierungen der Grund, den Assembler einzusetzen: Nicht alle Interrupts lassen sich mit der in amforth enthaltenen generischen Interrupt–Service–Routine behandeln, da sie spezielle Aktionen ausführen müssen, die der Hardware signalisieren, dass der Interrupt bearbeitet wurde. Unterlässt man diese Signalisierung, löst die Hardware umgehend einen neuen Interrupt aus, was das System faktisch lahmlegt.

Syntax

Der Assembler nutzt die Postfix–Notation, die für Forth so typisch ist. Zuerst werden die Operanden angegeben und danach die Kommandos. Die Namen selbst entstammen den AVR–Definitionen. Die Kommandoworte haben ein Komma als Suffix, um zu signalisieren: Hier wird in das Dictionary geschrieben.

AVR–Standard	amforth

ldi R16, 10
subi R8, 1
ld R17, Y+
	R16 10 ldi,
	R8 1 subi,
	R17 Y+ ld,

Der Assembler schreibt das Ergebnis seiner Arbeit direkt in das Dictionary und setzt den `HERE`–Pointer entsprechend weiter. Bei Sprüngen wird das Sprungziel ggf. nachträglich eingearbeitet, die erforderlichen Angaben werden auf dem Datenstack verwaltet.

[1] Derzeit sind die A/B–Register implementiert, die X/Y harren noch der weiteren Entwicklung.

Codeworte

Codewörter hießen früher Primitive und stellen in Maschinencode geschriebene und für den Forthinterpreter nutzbare Worte dar. Ein einfaches Codewort ist das folgende:

```
code dup
  -Y R23 st,
  -Y R22 st,
end-code
```

Im Beispiel wird ein Dictionaryeintrag `dup` erzeugt, dessen Execution Token auf den eigenen (noch leeren) Datenbereich zeigt. Im Unterschied zu Definitionswort : (COLON) wird nicht in den Compile-Modus umgeschaltet, die Assemblerworte werden ganz normal interpretiert. Dabei werden auf dem Datenstack die Operanden zusammengestellt, die die Assemblerworte dann passend in den Flash compilieren.

Die Register R22/R23 bilden das Top–Of–Stack–Element des Datenstacks, das Registerpaar Y ist der Datenstackpointer. Die Mnemonic `st` bezeichnet die Storeoperation (Speichern von Registerinhalten in RAM). `-Y` signalisiert einen Auto-Decrement des Y–Registerpaars.

Abgeschlossen wird eine code–Definition mit dem Wort `end-code`, das den obligatorischen Sprung in den inneren Interpreter compiliert.

Interrupts

Etwas interessanter wird es, wenn man Interruptroutinen im Assembler umsetzen will oder muss. Das kann, wie eingangs erwähnt, erforderlich sein, um innerhalb der Interruptbearbeitung gewisse Hardwareflags zurückzusetzen, was bei in Forth geschriebenen Interruptroutinen nicht möglich ist.

Da Interruptroutinen nicht direkt vom Forth-Interpreter aus aufrufbar sein sollen, legt man am einfachsten einen headerlosen Eintrag im Dictionary an, wobei die Startadresse gespeichert werden sollte.

```
here
R18 push,
R18 3F in,

...

3F R18 out,
R18 pop,
reti,

940c 6 i!
     7 i!
```

Zuerst wird die Startadresse ermittelt (`here`). Anschließend wird der Assemblercode in das Dictionary compiliert und am Ende die eingangs gespeicherte Startadresse als Ziel eines Sprungelements in die Interrupttabelle des Controllers geschrieben.

940c ist die AVR–Codierung für den absoluten Sprungbefehl, die Adresse 6 ist für den Interrupt 3 zuständig. Bei den meisten Atmega–Typen werden zwei Flashzellen für jeden Interruptvektor benötigt: Eine für den Befehlscode (940c) und eine für die Adresse. Bei kleineren Typen (ATmega8) ist nur eine Flashzelle pro Vektor vorgesehen, dann muss man einen relativen Sprung codieren, der in einer Flashzelle sowohl den Befehlscode als auch die Sprungdistanz enthält.

Wer nutzt es?

Lubos Pekny selbst hat mit diesem Assembler einen kleinen Stand–Alone–Rechner mit einem 4–zeiligem Text–LCD und einer PS/2–Standard–Tastatur gebaut. Details und ein Video sind im Internet über `www.forth.cz/download.html` zu finden.

Michael Kalus hat sich die Mühe gemacht, den Assembler systematisch auf Herz und Nieren zu prüfen; Ergebnis: alles funktioniert korrekt.

Euler 9 für Turbo–Forth und ZF

Fred Behringer

Zusammenfassung

Henry Vinerts (private Mitteilung) wollte Euler 9 gern mit Turbo–Forth in 16–Bit–Breite berechnen. Eine schnelle Abschätzung zeigt, dass man mit 16 Bit nicht duchkommt. Zwar gibt es (auch) in Turbo–Forth doppeltgenaue Arithmetik, aber ein paar Überlegungen zur Anpassung sind schon noch nötig. Insbesondere muss die Dreifachmultiplikation $a * b * c$ geeignet bewältigt werden.

Für 32–Bit–Systeme kennen wir den Lösungsvorschlag von Michael Kalus aus Seite 25 des VD–Heftes 2/2008 und das Verfahren von Ulrich Hoffmann aus Heft 3/2008, Seite 38, welches letztere das Ziehen der Quadratwurzel vermeidet. Ich habe mir für das 16–Bit–System Turbo–Forth Ullis Vorgehen als Ausgangspunkt gewählt. Als Nebenprodukt springt bei diesen Bemühungen heraus, dass das unten stehende Listing ohne jede Änderung auch für ZF verwendbar ist (ich habe es ausprobiert). Ich weiß, dass insbesondere Friederich Prinz und Martin Bitter im Zusammenspiel mit der Lokalen Forth–Gruppe Moers in früheren Zeiten intensiv mit ZF (von Tom Zimmer) gearbeitet und es modifiziert haben, und ich bin auf weitere Reaktionen zum Thema *Euler 9* gespannt.

Natürlich muss man bei meinem abgekürzten Verfahren zur Dreifachmultiplikation (siehe Listing) sichergestellt haben, dass das Dreifachprodukt in ein 32–Bit–Paket passt. Aber dieses Problem, dass nämlich Forth bei arithmetischen Operationen nicht standardmäßig gegen Überlauf des Gesamtergebnisses abgesichert ist, hat man ja generell. Man muss eben Vorüberlegungen einsetzen lassen oder durch Zusatzprogrammierung selbst Warn–Abhilfen schaffen. Was passiert aber mit den Zwischenergebnissen? Kann es da Überlauf geben? Michael Kalus geht im VD–Heft 2/2008 auf Seite 6 ausführlich auf diese Frage ein. Ich will hier keinen strengen Beweis für das Funktionieren des *abgekürzten* Verfahrens der Dreifachmultiplikation aus dem unten stehenden Listing geben. Ich will aber anhand eines ganz kleinen Beispiels versuchen, zur Verdeutlichung beizutragen.

Ich betrachte für dieses Beispiel ein 4—Bit–Forth (das ist nicht verboten :-) und lege als Dezimalwerte 5 6 7 auf den Stack. Im weiteren Verlauf gebe ich hier die Stackwerte (in einer Art von DEBUG–Vorgang) binär wieder. Das Verfahren geht davon aus, dass man weiß, dass das Dreierprodukt in einem 8–Bit–Paket Platz hat. (Diejenigen Zahlen, die in den folgenden Erläuterungen offensichtlich keine Binärzahlen sind, seien als Dezimalzahlen zu lesen. Die *frühesten* Stackwerte stehen links. Bei der Anzeige doppeltgenauer Werte ist in Little–Endian–Manier die Reihenfolge zu vertauschen. Man mache sich klar, dass man das Paar einfachgenauer Werte a b auf dem Stack auch als doppeltgenauen Wert $a + 16 * b$ deuten kann, usw.) Wie das genau aussieht, kann man in Abbildung 1 sehen.

Es konnte bei den Zwischenergebnissen kein Überlauf auftreten! Warum nicht? Kritisch sind die Stellen mit `*` und `+` . Sie können als Tricks betrachtet werden. Insbesondere vermeidet die Stelle mit `+` den *Umweg* über eine doppeltgenaue Addition `d+` und den damit verbundenen zusätzlichen Aufwand. (Die Stackbelegung der mit `rot` eingeleiteten Zeile hätte auch zu einem Paar doppeltgenauer Werte `0010 0110 0000 0111` erweitert werden können, das dann per `d+` hätte zusammengeführt werden müssen.)

```
     0101 0110 0111   (= 5 6 7  )
>r   0101 0110        (= 5 6    )
um*  1110 0001        (=  14 1  ) (= 14 + 16*1 = 00011110 = 30)
r@   1110 0001 0111   (=   30 7 ) (Ziel: 14*7 + 16*7)
*    1110 0111        (= 14 1*7 )
swap 0111 1110        (=  7 14  )
r>   0111 1110 0111   (=  7 14 7)
um*  0111 0010 0110   (=  7 14*7) (= 0111 01100010 = 7 98)
rot  0010 0110 0111   (=  98 7  ) (= 14*7 + 16*7)
+    0010 1101        (=  2 13  ) (= 2 + 16*(7+6) = 2+208)
d.   11010010 ok      (= 128+64+16+2 = 210 = 5*6*7)
```

Abbildung 1: Der Stackverlauf der Dreifachmultiplikation in einem 16–Bit–System

Listing

```
1  \ Euler 9 uho (VD-Heft 3/2008) 2008-08-24
2  \ Turbo-Forth-Adaption: beh 2008-10-10
3
4  \ Geht auch fuer ZF
5
6  \ Turbo-Forth: forth [ret] include euler.fth
7  \ ZF:          zf    [ret] fload   euler.fth
8
9  decimal
10
11 \ In Turbo-Forth gibt es kein UNLOOP
12 \ Hier eine High-Level-Definition:
13 : (UNLOOP) ( -- ) R> R> R> R> 2DROP DROP >R ;
14 : UNLOOP  ( -- ) COMPILE (UNLOOP)           ; IMMEDIATE
15
16 \ COMPILE aus Turbo-Forth gibt es in ANS-Forth nicht.
17 \ In ANS-Forth waere dafuer POSTPONE zu verwenden.
18
19 \ a + b + c = 1000
20 : a_b_c ( a b -- a b c ) 2dup + 1000 swap - ;
21
22 \ a^2 + b^2 = c^2
23 : pytriple? ( a b c -- flag )
24   >r dup um* rot dup um* d+ r> dup um* d=  ;
25
26 : euler9? ( a b -- flag ) a_b_c pytriple?   ;
27
28 : euler9 ( -- a b c ) 500 dup 1
29   DO dup I
30      DO J I euler9?
31         IF drop J I a_b_c UNLOOP UNLOOP EXIT THEN
32      LOOP
33   LOOP drop 0 0 0 ;
34
35 : .solution ( a b c -- )
36   dup  IF
37   >r cr ." a=" over . ." b=" dup . ." c=" r@ .
38   cr ." a+b+c=" 2dup + r@ + .
39   cr ." a*b*c=" um* r@ * swap r> um* rot + d.
40   EXIT THEN
41   drop drop drop cr ." No solution"        ;
42
43 \ Schwierigkeiten macht in Turbo-Forth die Zeile mit a*b*c.
44 \ Turbo-Forth ist nur 16-bittig und schafft die dreifache
45 \ Multiplikation in der geforderten Breite nicht. Zwar kann
46 \ man mit um* zwei 16-bittige Werte zu einem 32-Bit-Wert
47 \ multiplizieren, es gibt aber in Turbo-Forth kein Wort,
48 \ das die Multiplikation eines 32-Bit-Wertes mit einem
49 \ 16-Bit-Wert gestattet. In ANS-Forth, also auch in jedem
50 \ normgerechten 16-Bit-ANS-System, gibt es das Wort m*/
51 \ ( d1 n1 +n2 -- d2 ), das nach Setzen von +n2 = 1 genau
52 \ das Ebenerwaehnte leistet. Ich habe hier auf die explizite
53 \ Einfuehrung von m*/ verzichtet und mich im Forth-Wort
54 \ .solution mit der Nachbildung der Funktion von m*/
55 \ speziell fuer +n2 = 1 mit den Mitteln von Turbo-Forth
56 \ begnuegt.
57
58 euler9 .solution
59
```

```
60  \ Weitere Aufgaben:
61  \ Man ersetze 1000 durch 40 . Was kommt dabei heraus?
62  \ Genau! Es kommt genau das heraus, was man beim Durchlesen
63  \ der Zeilen von Henry Vinerts auf Seite 7 des VD-Heftes
64  \ 3/2008 erwartet. Merke: Berechnung ist gut, Ueberlegung
65  \ ist besser! Man erweitere das Programm so, dass saemtliche
66  \ Moeglichkeiten mit a+b+c = d bei d < 1000 ausgeschoepft
67  \ werden. Wie sieht es fuer d > 1000 aus? (Man beachte auch
68  \ die DO-Grenzen.) Wie weit kann man mit meiner
69  \ Turbo-Forth-Programm-Version (in 16-Bit-Breite) gehen?
```

c18 colorForth Compiler

Charles Moore

Im folgenden Artikel aus dem Jahr 2001 beschreibt Chuck Moore seinen colorForth–Compiler für den c18–Prozessor, der heute Grundlage der SEAforth–Prozessoren ist.

Zusammenfassung

c18 ist ein kleiner Computer, der auf einem Siliziumdie oft vervielfacht werden kann. Seine Architektur und seine Instruktionen sind für Forth optimiert. colorForth ist ein Forth–Dialekt, der voranalysierten Quellcode und Farbe für die Zeichensetzung verwendet.

Es ist ein Compiler in colorForth enstanden, der colorForth–Quellcode in c18–Maschinencode übersetzt. Der Code kann im c18–ROM oder –RAM abgelegt und von einem Simulator ausgeführt werden. Oder in serielles EPROM/Flash, um von einem c18 ausgeführt zu werden, wenn er existiert.

Die Betonung liegt hier darauf, wie einfach ein colorForth–Cross–Compiler sein kann. Der c18 ist interessant, weil er extrem schnell bei wenig Leistung arbeitet: 2400 Mips @ 20 mW.

c18–Instruktionen

Hier sind die momentanen c18–Instruktionen, die übersetzt werden müssen. Während sich das Design weiterentwickelt, sind Änderungen offensichtlich leicht vorzunehmen.

	Register
T	oberstes Stack–Element
S	2. Stack–Element
R	oberstes Returnstack–Element
A	Adressregister
B	Adressregister

Beachtet, dass das Laden ein Element auf den Stack legt und das Speichern und die zweistelligen Operatoren ein Element vom Stack entfernen.

Code	Op	Aktion
0	;	aus Unterprogramm zurückspringen
1		
2	Wort	Unterprogramm aufrufen
3	word ;	zum Unterprogramm springen (Endrekursion)
4		
5		
6	if	springe zu 'then' wenn T0-T17 alle null sind
7	-if	springe zu 'then' wenn T17 null ist
8	n	lade Literal
9	@+	lade von Adresse in A; erhöhe A
a	@b	lade von Adresse in B
b	@	lade von Adresse in A
c	!r	speichere in die Adresse in R; erhöhe R
d	!+	speichere in die Adresse in A; erhöhe A
e	!b	speichere in die Adresse in B
f	!	speichere in die Adresse in A
10	+*	addiere S zu T, wenn T0=1; schiebe rechts
11	2*	schiebe T 1 Bit links
12	2/	schiebe T 1 Bit rechts; erhalte T17
13	-	1er–Komplement von T
14	+	addiere S zu T
15	or	exklusiv-oder von S und T nach T
16	and	und von S und T nach T
17	drop	bewege S nach T
18	dup	dupliziere T
19	over	lade S
1a	pop	lade R
1b	a	lade A
1c	.	tue nichts
1d	b!	speichere in B
1e	push	speichere in R
1f	a!	speichere in A

colorForth

Eine vollständige Beschreibung ist unter `www.colorforth.com` verfügbar. Die relevante Besonderheit hier ist, dass die Farbe eines Wortes seine Funktion bestimmt:

rot	-	definiere ein neues Wort
grün	-	übersetze
gelb	-	führe aus
weiß	-	kommentiere

Um Farbe in diesem Artikel zu vermeiden:

GESPERRT	-	definiere
normal	-	übersetze
fett	-	führe aus
kursiv	-	kommentiere

Fett wird auch benutzt, wenn Wörter im Text referenziert werden.

Fortsetzung auf Seite 18

Programmieraufgabe: Texte in Wörter zerlegen?

Ulrich Hoffmann

Auf Alexander von Leitners (Fefe) Blog `blog.fefe.de` habe ich neulich Folgendes gefunden [1]:

Sun Dec 7 2008

> Ich mache gerade einen kleinen Test, bei dem ihr mir helfen könnt, wenn ihr Lust habt. Es geht darum, eine vergleichsweise triviale Aufgabenstellung in Skriptsprachen zu implementieren. Die Aufgabenstellung ist: stdin lesen, in Wörter splitten, für die Wörter dann jeweils die Häufigkeit zählen, am Ende die Wörter mit Häufigkeiten nach letzteren sortiert ausgeben. Einfach genug, in perl ist das in unter 10 Zeilen ordentlich zu machen.
>
> Die Idee ist, dass man das in diversen Sprachen mal implementiert. Und zwar schon so, dass das nicht unnötig langsam abläuft, aber auch nicht groß performance-rumhacken. D. h.: in perl implementiert man das in perl und nicht als C–Extension. In C benutzt man malloc und stdio und keinen custom allocator bzw eigene I/O-Pufferverwaltung. Also schon versuchen, schnell ablaufenden Code zu schreiben, aber so, wie man das in der Sprache tun würde. Für C++ heisst das: STL und iostreams benutzen.
>
> Am Ende kann man das natürlich nutzen, um da mal zu gucken, welche Implementation am schnellsten läuft. Hier geht es nicht darum, die Überlegenheit von C zu zeigen, sondern die relative Performance von Interpretern zu gucken. Es geht aber eben nicht nur um Performance, sondern der Code soll auch demonstrieren, wie man sowas in der jeweiligen Sprache löst. Damit man sich die Sourcen anschauen kann, und sich entscheiden kann, ob man diese Sprache für hinreichend elegant hält, dass man sich die mal näher anschauen will.
>
> Wieso ihr mithelfen könnt: ich kann nicht alle Sprachen und habe auch keine Lust, mich da jetzt jeweils reinzugoogeln. Denn ich will den Sprachen ja auch gerecht werden und nicht bloss eine laufende Variante nehmen, die dann am Ende unfair schlecht aussieht oder unnötig langsam ist. Daher wäre es mir lieb, wenn mir jemand für die fehlenden Sprachen Sourcen zuschickt. Ich erwarte da jetzt nicht AppleScript oder so, aber C# wäre z. B. nett, und Tcl. Und vielleicht noch sowas wie LISP oder Scheme.

Von Forth spricht Fefe erst gleich gar nicht :-(Aber wie sähe denn eine Lösung in (Standard-)Forth aus? Im Aufspalten von Texten ist Forth ja dank `WORD` und `PARSE` gut bestückt und einzelne Wörter kann man prima im Dictionary verwalten und dort auch ihre Häufigkeiten zählen.

Also lieber Leser, wie sieht Deine Forth-Lösung aus?

Einsendungen nimmt die Redaktion der Vierten Dimension gerne entgegen. Wir drucken Sie dann in den kommenden Ausgaben und leiten Sie auch gerne an Fefe weiter. Zeigt mal, was in Euch und was in Forth steckt.

Links

[1] Fefes Aufruf `http://blog.fefe.de/?ts=b7c295e7`
[2] Fefe's call in english `http://ptrace.fefe.de/wp/README.txt`

> Faust Habe nun, ach! Philosophie, Juristerei und Medizin, Und leider auch Theologie Durchaus studiert, mit heißem Bemühn. Da steh ich nun, ich armer Tor! Und bin so klug als wie zuvor; Heiße Magister, heiße Doktor gar Und ziehe schon an die zehen Jahr Herauf, herab und quer und krumm Meine Schüler an der Nase herum- Und sehe, daß wir nichts wissen können! Das will mir schier das Herz verbrennen. Zwar bin ich gescheiter als all die Laffen, Doktoren, Magister, Schreiber und Pfaffen; Mich plagen keine Skrupel noch Zweifel, Fürchte mich weder vor Hölle noch Teufel- Dafür ist mir auch alle Freud entrissen, Bilde mir nicht ein, was Rechts zu wissen, Bilde mir nicht ein, ich könnte was lehren, Die Menschen zu bessern und zu bekehren. Auch hab ich weder Gut noch Geld, Noch Ehr und Herrlichkeit der Welt; Es möchte kein Hund so länger leben! Drum hab ich mich der Magie ergeben, Ob mir durch Geistes Kraft und Mund Nicht manch Geheimnis würde kund; Daß ich nicht mehr mit saurem Schweiß Zu sagen brauche, was ich nicht weiß; Daß ich erkenne, was die Welt Im Innersten zusammenhält,

Text zum Testen des gesuchten Splitt–Programms, das folgendes Ergebnis liefern sollte:

ich 9	und 7	mir 5	nicht 4	Und 4	was 4
die 3	zu 3	noch 3	der 3	mich 2	Daß 2
als 2	ein, 2	hab 2	bin 2	nun, 2	Bilde 2
auch 2	weder 2	an 2	Magister, 2	mit 2	so 2
all 1	Ehr 1	Innersten 1	alle 1	bekehren. 1	Schüler 1
Herz 1	quer 1	Das 1	Juristerei 1	Habe 1	Kraft 1
wissen 1	Magie 1	Drum 1	lehren, 1	Geistes 1	Die 1
wissen, 1	Ob 1	Pfaffen; 1	das 1	Geld, 1	Doktor 1
Rechts 1	sagen 1	nichts 1	sehe, 1	Jahr 1	bessern 1
⋮	⋮	⋮	⋮	⋮	

Bootmanager und FAT-Reparatur: Ein zweiter Fort(h)schritt

Fred Behringer

Wenn Sie (zumindest) unter DOS 6.2 den Kommandozeilen-Befehl DIR eingeben, bekommen Sie am Anfang drei Zeilen auf den Bildschirm. Die zweite davon lautet `Datenträgernummer: 3E5A-1F06`, wobei diese Zahl auf meiner Experimentier-Anlage dem eingeschalteten Laufwerk `E:`, dem zweiten logischen Laufwerk der erweiterten Partition, zugeordnet ist. Haben Sie sich schon einmal gefragt, wo diese Zahl herkommt, oder zumindest, wie man sie sich in einem (Forth–)Programm beschaffen und dann nutzbar machen könnte? Geben Sie (aus Forth heraus) in meinem unten stehenden Listing `2 showboot16` ein. Im Teil 2 der Bildschirmanzeige (nach Drücken einer Taste) steht dann, dass ab dem Bootsektor–Offset 027h über 4 Bytes hinweg die `Dateisystems-ID (Seriennnummer)` verzeichnet ist, und zwar in meinem hier berichteten Fall in der Form `06 1F 5A 3E` (little endian!). Alles, was Sie zu diesem kleinen Vorab–Experiment brauchen, ist das Wissen darum, dass es sich bei meinem `E:` um ein FAT16–formatiertes logisches Laufwerk handelt — `2 showboot32` würde falsche Anzeigen liefern.

Bei keinem der im vorliegenden Artikel eingeführten Forth–Worte wird an der Festplatte etwas verändert. Es werden Informationen von der Festplatte lediglich hereingeholt und dann für die diversen Anzeigen weiterverarbeitet. Mit anderen Worten: Mit den hier neu eingeführten Forth–Worten kann bedenkenlos experimentiert werden, ohne befürchten zu müssen, dass wesentliche Daten an der Festplatte verstellt werden. (Das gilt natürlich nicht für alle Forth–Worte aus meinem Artikel im VD–Heft 3/2008.)

Es bleibt noch viel für mich zu untersuchen: Unter `n showboot16` liefert der zweite Anzeigeteil (nach Drücken einer Taste) die Erkenntnis, dass die eben erwähnte Seriennummer zur Unterscheidung von auswechselbaren Datenträgern verwendet werden kann. In der Tat liefern ZIP–Disketten oder übliche Disketten verschiedene Seriennummern, wenn sie auf verschiedenen Anlagen oder unter verschiedenen Betriebssystemen (ich habe es mit reinem DOS 6.2, mit FreeDOS, mit OS/2 und in der DOS–Vollbox von Windows 95 ausprobiert) formatiert wurden. Die Seriennummer scheint also dem Datenträger anzuhaften, und nicht dem Laufwerk. Werden allerdings verschiedene solcher Datenträger in ein und demselben Laufwerk formatiert, dann scheinen sie alle dieselbe Seriennummer zu haben. Es wäre vielleicht gar nicht so schlecht, wenn ich mein bisheriges Augenmerk nicht nur auf Festplatten, sondern auch auf Disketten und ZIP–Disketten (und CDs?) richten würde?

Im VD–Heft 3/2008 habe ich dargelegt, was mich dazu bewogen hat, mich mit dem FAT16–Dateisystem zu beschäftigen: Ein zerschossenes Windows 98 und der Wunsch, meine verschwundenen E–Mail–Dateien wiederzubeleben. Dass ich dabei in die Fußstapfen von Bootmanager–Entwicklern trat, ist meinem Interesse für Forth und meiner ständigen Neugierde zuzuschreiben. Wäre es nur um einen neuen, noch besser zu meinen Vorhaben passenden Bootmanager gegangen (der von mir bevorzugte `XFDISK` ist ja an sich schon recht prima), dann hätte ich das Rad bestimmt nicht neu erfinden wollen. Dann hätte ich vielleicht auf den allerneuesten Stand der Technischen Evolution (wie langsam die doch geht, gemessen am kurzen Leben des Menschen) gewartet und mir das PC–Praxis–Sonderheft 03/08 mit der menügeführten Super–GRUB–CD als ISO-Datei auf der Heft–DVD gekauft. (Auf der Ultimate Boot–CD 3.4 aus PC–Professionell 9/06 war die Super–GRUB–Disk in der Version 0.925 ja auch schon enthalten und arbeitet (in der Live–Version) auch auf meiner schon etwas ältlichen K6/2–Anlage bestens.)

Im vorliegenden Artikel geht es mir um die Bootsektoren. Sie sind für FAT16 etwas anders aufgebaut als für FAT32, ähneln sich aber doch sehr. Ich darf daher hier schonmal damit beginnen, mich neben FAT16 auch für FAT32 zu interessieren. Nicht nur die primären Partitionen, auch die logischen Laufwerke der erweiterten Partition (bei mir momentan 16 an der Zahl) haben, jedes für sich, einen Bootsektor — der nicht mit der jeweiligen Partitionstabelle der Laufwerke (in ihrer kettenartigen Verschachtelung) zu verwechseln ist. Neben dem generellen Master-Boot-Record (CHS = 001) enthalten auch die einzelnen Bootsektoren (z. B. CHS = 011) wesentliche Informationen über die Organisation der bootenden Festplatte. Im vorliegenden Artikel stelle ich die zur Anzeige benötigten Werkzeuge im Rahmen von Forth dar. Bei meinen Reparaturarbeiten am zerschossenen Windows–98–System haben sie mir sehr geholfen. Es ist immer wieder faszinierend zu sehen, wie leicht man ein einmal erstelltes Forth–Wort (sogar im laufenden Betrieb) den sich ändernden Ansprüchen anpassen kann.

Ich setze das gesamte Listing aus dem Artikel *Bootmanager und FAT-Reparatur: Erste Fort(h)schritte* aus dem VD–Heft 3/2008 und die zugehörigen Erklärungen voraus: Einfach das diesmalige Listing an das vorausgegangene anheften — oder mit einem geschachtelten include (bei Turbo–Forth) oder fload (bei ZF) arbeiten.

Der (programmiertechnischen) Einfachheit halber unterscheide ich bei den anzuzeigenden Bootsektoren zwischen FAT16, mit `showboot16 ( n - )`, und FAT32, mit `showboot32 ( n - )`. Man muss also vorher wissen, ob 16 oder 32, wenn man die Bildschirmanzeige richtig interpretieren will. Ein Irrtum wäre aber nur halb so wild, da die ersten Anzeigehälften von `showboot16`

und `showboot32` gleich sind. Außerdem unterscheide ich zwischen der jeweils aktivierten Primärpartition und den logischen Laufwerken der erweiterten Partition: `showboot16` und `showboot32` beziehen sich auf die logischen Laufwerke der erweiterten Partition (und benötigen die Laufwerksnummer n), `showbootactive16` und `showbootactive32` kommen ohne einen solchen Parameter n aus und beziehen sich auf die jeweils aktivierte (als *bootbar* geschaltete) Primärpartition. Sollte es jemand für sinnvoll halten, mehr als eine Partition aktivzuschalten, dann wird aus der generellen Partitionstabelle (im MBR) die Zeile mit der ersten der aktivierten Partitionen herausgepickt.

Der Schwerpunkt des vorliegenden Artikels liegt auf der Bildschirm–Anzeige des gewünschten Bootsektors, nicht auf dessen Berichtigung und Reparatur. Worte wie `putbootactive` und n `putboot` (die bei Falscheingabe sowieso Schaden an den Daten anrichten könnten) habe ich mir für später vorgemerkt. Ich will sie hier noch nicht behandeln.

Listing

```
1   \ ******************************************************
2   \ *                                                    *
3   \ *  BOOTMA-2.FTH                                      *
4   \ *                                                    *
5   \ *  Zutaten fuer FAT-Reparatur und Bootmanager unter  *
6   \ *  Turbo-FORTH-83 und ZF                             *
7   \ *                                                    *
8   \ *  Fred Behringer - Forth-Gesellschaft - 15.11.2008  *
9   \ *                                                    *
10  \ ******************************************************
11
12  \ Fuer Turbo-Forth: FORTH INCLUDE BOOTMAST.FTH INCLUDE BOOTMA-2.FTH [ret].
13  \ Hierbei ist die erste Include-Datei die aus Heft 3/2008 und die zweite
14  \ die hier zu besprechende. Man beachte, dass man hier bis zum letzten
15  \ INCLUDE (man darf natuerlich auch alles kleinschreiben) alle Eingaben
16  \ 'schachteln' kann, ohne per [ret] 'absetzen' zu muessen.
17
18  \ Fuer ZF: ZF [ret] [ret] FLOAD BOOTMAST.FTH [ret] FLOAD BOOTMA-2.FTH [ret]. In
19  \ BOOTMAST.FTH muss vorher attributs off per \ auskommentiert werden. Bei ZF,
20  \ zumindest in der erweiterten Fassung der Lokalen Forth-Gruppe Moers, geht die
21  \ Schachtelung der Kommandozeilen-Eingaben nicht, ohne dass man zwischendurch
22  \ per [ret] absetzt. Die Eingabe von ZF muss sogar zweimal quittiert werden.
23
24
25  \ Glossar
26
27  \ getbootactive    ( -- ) Bootsektor der aktivierten Partition nach sectbuf
28  \ (showboot16-1)   ( -- ) Ersten Textteil des Bootsektors anzeigen
29  \ (showboot16-2)   ( -- ) Zweiten Textteil des Bootsektors bei FAT16 anzeigen
30  \ (showboot32-1)   ( -- ) Ersten Textteil des Bootsektors anzeigen
31  \ (showboot32-2)   ( -- ) Zweiten Textteil des Bootsektors bei FAT32 anzeigen
32  \ (showboot)       ( n -- ) Bootsektor dumpen. n=10h: 4 Zeilen, 20h: 6 Zeilen
33  \ (showboot16)     ( -- ) Inhalt von sectbuf als FAT16-Bootsektor anzeigen
34  \ (showboot32)     ( -- ) Inhalt von sectbuf als FAT32-Bootsektor anzeigen
35  \ showboot16       ( n -- ) Wie (showboot16), aber erst Bootsektor von HD holen
36  \ showboot32       ( n -- ) Wie (showboot32), aber erst Bootsektor von HD holen
37  \ showbootactive16 ( -- ) Wie showboot16, aber auf aktivierte Partition bezogen
38  \ showbootactive32 ( -- ) Wie showboot32, aber auf aktivierte Partition bezogen
39
40
41  hex
42
43  \ getbootactive laedt den Bootsektor der (ersten) aktiven Partition in den
44  \ Puffer sectbuf . 'Normalerweise' ist das eine primaere Partition. Ist die
45  \ erste aktivgeschaltete Partition (beabsichtigt oder nicht) die erweiterte
46  \ Partition, dann wird deren erstes logisches Laufwerk angesprochen und der
47  \ zugehoerige Bootsektor nach sectbuf geholt. Ist keine (der 4 moeglichen!)
```

```
48  \ Partitionen aktiv, dann wird getbootactive mit einer Fehlermeldung verlassen.
49
50  : getbootactive ( -- fl )
51     getmbr                              \ MBR nach sectbuf holen
52     sectbuf 1ae +                       \ Ausgangsposition im Puffer
53     4 0                                 \ 4 relevante Zeilen im MBR
54     do
55       10 + dup c@ 0 <>                  \ Partition aktiv?
56       if 2 + @ 1 swap (getsect) 0 leave \ Ja, dann Bootsektor nach sectbuf
57       then                              \ und raus.
58     loop
59     0 <>
60     if cr ." Keine Partition aktiv!" 0  \ Sonst Fehlermeldung.
61     else 1
62     then ;
63
64
65  \ (showboot16) interpretiert den Inhalt des Puffers sectbuf als Bootsektor eines
66  \ logischen Laufwerks der erweiterten Partition und zeigt den darin enthaltenen
67  \ Festplatten-Informationsteil mit bildschirmgerechten Erlaeuterungen am
68  \ Bildschirm an. Achtung: (showboot16) kuemmert sich nicht darum, ob im Puffer
69  \ sectbuf wirklich ein brauchbares Abbild eines Bootsektors liegt. Vor Aufruf
70  \ von (showboot16) muss man daher gegebenenfalls den Bootsektor des
71  \ interessierenden Laufwerks erst per n getboot von der Festplatte in den Puffer
72  \ sectbuf holen. n=1 ist dabei das erste Laufwerk der erweiterten Partition, mit
73  \ entsprechender Durchnummerierung fuer die weiteren Laufwerke.
74
75  \ Im Teil 1 dieser Artikelserie (VD-Heft 3/2008) interessierten bei getboot nur
76  \ die logischen Laufwerke der erweiterten Partition. Ich moechte aber hier auf
77  \ die primaeren Partitionen nicht verzichten und habe soeben mit dem Forth-Wort
78  \ getbootactive ( -- ) ein Mittel zum Schreiben des Bootsektors der momentan
79  \ gerade aktivierten primaeren Partition in den Puffer sectbuf eingefuehrt.
80
81  \ (showboot16) geht nur auf den Inhalt des Puffers sectbuf ein und braucht also
82  \ keinen Unterschied zwischen getboot und getbootactive zu machen. In showboot16
83  \ dagegen (siehe unten) wird je nach Gegebenheit entweder auf getboot oder
84  \ getbootactive umgeschaltet. (Es lebe die Unfaehigkeit des Entwicklers, schon
85  \ beim ersten Zusammenstellen eines VD-Artikels eventuelle Vorhaben fuer weitere
86  \ Artikel vorauszusehen!)
87
88  \ Wie in Teil 1 dieser Artikelserie soll auch hier mein Hauptaugenmerk auf FAT16
89  \ gerichtet bleiben. Da sich aber die Bootsektoren bei FAT16 von denen bei FAT32
90  \ nicht besonders stark unterscheiden, bietet es sich an, schon hier auch auf
91  \ die Verhaeltnisse bei FAT32 einzugehen. Bei den im Listing eingefuehrten
92  \ Forth-Worten sind das die Stellen, an welchen in den Wort-Namen 16 durch 32
93  \ ersetzt ist.
94
95  \ (showboot16-1) ist der erste Teil der Bildschirmanzeige von (showboot16) .
96  \ Dieser erste Teil ist bei (showboot32) derselbe wie bei (showboot16) . Bei
97  \ (showboot16) setzt hier nach Druecken einer Taste (mit Ausnahme von [ret]) der
98  \ zweite Teil, naemlich (showboot16-2), ein. Bei (showboot32) schiebt sich hier
99  \ als zweiter Teil (showboot32-2) ein.
100
101 : (showboot16-1) ( -- )
102 ." Ofs Lng Inhalt"                                                    cr
103 ." 000  3  Sprung zum Boot-Code (EB 3C 90 = JMP 003E NOP)."           cr
104 ." 003  8  OEM-Name, z.B. MSDOS6.2 oder MSWIN4.1 ."                   cr
105 ." 00B  2  Bytes pro Sektor, i.A. 200 (512d)."                        cr
106 ." 00D  1  Sektoren pro Cluster."                                     cr
107 ." 00E  2  Anzahl reservierter Sektoren (mit Bootsektor, also stets >= 1)."  cr
```

```
108  ." 010  1  Anzahl der FATs (i.A. 2)."                                    cr
109  ." 011  2  Max. Eintraegezahl im Stammverz. Bei FAT32 ungenutzt und = 0000." cr
110  ." 013  2  Gesamtsektorzahl (max. 65535d) fuer Part. < 32MB. Bei > 32MB ist" cr
111  ."         es 0 und der Wert steht bei 020 (4 Bytes). Bei FAT32 ungenutzt."  cr
112  ." 015  1  Media-Descriptor-Byte. Seit Windows Server 2003 veraltet."      cr
113  ." 016  2  Sektorzahl/FAT. Bei FAT32 stets gleich 0000 (siehe Offset 24)." cr
114  ." 018  2  Sektoren pro Spur."                                            cr
115  ." 01A  2  Anzahl der Seiten bzw. Schreib-Lese-Koepfe."                   cr
116  ." 01C  4  Bei HDs Zahl der Sektoren zwischen MBR und Bootsektor der ersten" cr
117  ."         prim. Partition. Bei nicht partitionierten Medien stets 0000."  cr
118  ." 020  4  Gesamtsektoranzahl (lo/hi) fuer Partitionen > 32MB."           cr
119    ;
120
121  : (showboot16-2) ( -- )
122  ." Ofs Lng Inhalt"                                                        cr
123  ." 024   1 Phys. Laufwerksnummer (00 bei Disketten, 80, 81,... bei HDs)."  cr
124  ."         Ist nur fuer Bootlaufwerke relevant, da diese Nummer beim Booten" cr
125  ."         fuer BIOS-Aufrufe zum Zugriff auf das Medium benutzt wird."     cr
126  ." 025   1 Reserviert (stets 00 bei FAT 12/16)."                          cr
127  ." 026   1 Erweiterte Bootsignatur."                                      cr
128  ." 027   4 Dateisystems-ID (Seriennummer). Dient der Unterscheidung"      cr
129  ."         verschiedener Medien (z.B. bei Wechselmedien)."                 cr
130  ." 02B  0B Name des Dateisystems (max. 11d Zeichen, durch Leerzeichen"     cr
131  ."         aufgefuellt). Veraltet. Wird durch einen speziellen"            cr
132  ."         Verzeichniseintrag im Stammverzeichnis abgeloest."              cr
133  ." 036   8 FAT-Art, mit Leerzeichen aufgefuellt, z.B.: 'FAT12   ',"       cr
134  ."         'FAT16   '."                                                    cr
135  ." 03E 1C0 Beginn des Bootprogramms."                                      cr
136  ." 1FE   2 BIOS-Bootsektorsignatur. Enthaelt die Hex-Bytes 55 AA, anhand"  cr
137  ."         derer das BIOS beim Booten einen gueltigen Bootsektor erkennt." cr
138    ;
139
140  : (showboot32-1) ( -- ) (showboot16-1) ;
141
142  : (showboot32-2) ( -- )
143  ." Ofs Lng Inhalt"                                                        cr
144  ." 024   4 Anzahl der Sektoren pro FAT"                                   cr
145  ." 028   2 FAT-Flags"                                                     cr
146  ." 02A   2 FAT-32-Version. Derzeit stets 0000 bei Microsoft."             cr
147  ." 02C   4 Clusternummer, an der das Stammverzeichnis beginnt. Meistens 2." cr
148  ." 030   2 Sektornummer des 'FS-Information-Sectors' (in der Regel: 1)"    cr
149  ." 032   2 Sektornummer der Bootsektorkopie (in der Regel: 6)"            cr
150  ." 034  0C Reserviert fuer spaetere Erweiterungen. Derzeit stets 0."       cr
151  ." 040   1 Phys. BIOS-Laufwerksnummer"                                    cr
152  ." 041   1 Reserviert"                                                     cr
153  ." 042   1 Signatur"                                                       cr
154  ." 043   4 Dateisystems-ID (Seriennummer)"                               cr
155  ." 047  0B Name des Dateisystems (ungenutzt)"                             cr
156  ." 052   8 FAT-Version. Stets: 'FAT32   '"                                cr
157  ." 05A 1A4 x86-Maschinencode des Bootloaders"                             cr
158  ." 1FE   2 BIOS-Bootsektorsignatur. Enthaelt die Hex-Bytes 55 AA ."       cr
159    ;
160
161
162  \ Bei Eingabe von 10 (showboot) wird ein Dump der ersten 4 Zeilen ausgegeben.
163  \ Das wird bei Interpretation als FAT16-Bootsektor (10h = 16d) eingesetzt. Bei
164  \ Eingabe von 20 (showboot) werden 6 Zeilen gedumpt, was bei Interpretation als
165  \ FAT32-Bootsektor (20h = 32d) zum Einsatz kommt. Der Unterschied kommt bei den
166  \ gleich folgenden Forth-Worten (showboot16) und (showboot32) zum Tragen.
167
```

```
168  : (showboot) ( n -- )  \ n=10h: 4 Zeilen des Bootsektors, n=20h: 6 Zeilen
169      sectbuf over 10 =
170      if 40 else 60 then dump
171  \   (cursor)                    \ Gibt es nur in Turbo-Forth
172      #out @ #line @              \ Geht sowohl in Turbo-Forth wie auch in ZF
173      nip over 10 =
174      if 5 else 7 then
175      - 0 swap at              cr \ at gibt es  in Turbo-Forth und auch in ZF
176      ."   Offset    0"
177  \   (cursor)                    \ Gibt es nur in Turbo-Forth
178      #out @ #line @              \ Geht sowohl in Turbo-Forth wie auch in ZF
179      swap 30 + swap at ." 0"  cr \ at gibt es  in Turbo-Forth und auch in ZF
180      ."    000:  "            cr
181      ."    010:  "            cr
182      ."    020:  "            cr
183      ."    030:  "            cr
184      20 = if
185      ."    040:  "            cr
186      ."    050:  "            cr
187           then                cr ;
188
189
190  \ Menuegesteuerte Bildschirmanzeige der Festplatten-Informationen des im Puffer
191  \ sectbuf gespeicherten Bootsektors. Dieser muss vorher per n getboot oder
192  \ getbootactive in den Puffer geholt worden sein. Die Anzeige geht vom
193  \ DOS-ueblichen Bildschirm-Modus von 80 Zeichen mal 25 Zeilen aus und
194  \ interpretiert den Bootsektor als unter FAT16 geschrieben.
195
196  : (showboot16) ( -- )
197      begin
198        dark 10 (showboot) (showboot16-1)
199        18 spaces ." [Taste]: Teil2    [Return]: Raus "
200        key 0d = if exit then
201        dark 10 (showboot) (showboot16-2)
202        18 spaces ." [Taste]: Teil1    [Return]: Raus "
203        key 0d = if exit then
204      again ;
205
206
207  \ Menuegesteuerte Bildschirmanzeige der Festplatten-Informationen des im Puffer
208  \ sectbuf gespeicherten Bootsektors. Dieser muss vorher per n getboot oder
209  \ getbootactive in den Puffer geholt worden sein. Die Anzeige geht vom
210  \ DOS-ueblichen Bildschirm-Modus von 80 Zeichen mal 25 Zeilen aus und
211  \ interpretiert den Bootsektor als unter FAT32 geschrieben.
212
213  : (showboot32) ( -- )
214      begin
215        dark 10 (showboot) (showboot32-1)
216        18 spaces ." [Taste]: Teil2    [Return]: Raus "
217        key 0d = if exit then
218        dark 20 (showboot) (showboot32-2)
219        18 spaces ." [Taste]: Teil1    [Return]: Raus "
220        key 0d = if exit then
221      again ;
222
223
224  \ showboot16 leitet die Bildschirmanzeige des Bootsektors n der erweiterten
225  \ Partition ein. Wird der Laufwerk-Parameter n vergessen, dann wird (soweit
226  \ auf dem Datenstack nicht noch ein weiterer Eintrag uebriggeblieben ist) an
227  \ Stelle des vergessenen Parameters der spezielle Wert n = 1 genommen.
```

```
228
229   : showboot16 ( n -- )
230       depth 0 = if 1 then getboot (showboot16) ;
231
232
233   \ showboot32 wirkt wie showboot16, nur dass die Anzeige als unter FAT32 erfolgt
234   \ zu interpretieren ist.
235
236   : showboot32 ( n -- )
237       depth 0 = if 1 then getboot (showboot32) ;
238
239   : showbootactive16 ( -- ) getbootactive if (showboot16) then ;
240
241   : showbootactive32 ( -- ) getbootactive if (showboot32) then ;
```

Fortsetzung des Artikels von Chuck Moore von Seite 11

Compiler

Der c18 hat 192 Worte Speicher mit je 18 Bit. RAM-Adressen sind 0–7f, ROM sind 80–bf. Instruktionen sind 5 Bit breit. Es gibt 4 Plätze (Slots) für Instruktionen in jedem Wort. Von links nach rechts: S0, S1 und S2 sind 5 Bit breit, S3 ist 3 Bit breit und wird um zwei niederwertige Nullen ergänzt.

Der Pentium–Compiler interpretiert colorForth–Quellcode und packt Instruktionen in die niederwertigen 18 Bits seines 32–Bit–Speichers.

Es gibt einige Einschränkungen:

- Nur die ...00-Instruktionen passen in Slot 3.
- Sprung–Instruktionen sind auf Slot 0 und 1 beschränkt. Die Sprungadresse ist in Slot 2 und Slot 3.
- Sprünge gehen zu Slot 0 des Ziels.

c18–Quellcode ist grün, genau wie Pentium–colorForth. Aber grüne Zahlen werden redefiniert, so dass sie ausgeführt werden. Jedes Wort kompiliert seinen Instruktions–Code in den nächsten verfügbaren Slot. Und grüne Zahlen werden redefiniert, so dass sie ein c18–Literal kompilieren. Gelbe Wörter oder Zahlen sind in c18–Quellcode nicht angebracht, weil c18–Wörter selber nicht ausgeführt werden können.

Um c18–Wörter zu definieren, die einen Unterprogrammaufruf kompilieren, wird das colorForth–Konstrukt **class** verwendet. Der Code, den die Variable **class** bestimmt, wird ausgeführt, wann immer ein Wort definiert wird. Für c18 ist das ein Makro, das Pentium–Code kompiliert, der die c18–Adresse auf den Stack legen und dann ein Wort aufrufen wird, das einen c18–Unterprogrammaufruf kompiliert.

Natürlich ist **empty** neu definiert, so dass es das normale Verhalten grüner Wörter, grüner Zahlen und von **class** wieder herstellt.

Beachte, dass Sprung–Instruktionen oft ein neues Instruktionswort erzwingen. Manchmal kann ein Umsortieren von Code ansonsten leer gelassene Slots füllen. Das Beste ist, Definitionen klein und einfach zu halten und darauf zu achten, wie gut Instruktionen sich packen.

Quellcode des Compilers

Hier ist ein Ausschnitt des Compilers. Für die, die colorForth laufen lassen können, gibt es die 3 Blöcke voranalysierten Quellcode unter `www.colorforth.com`.

```
S4 n   h @  ip ! 8192 * , 1  slot ! ;
S0 n   8192 *
SN n   dup  call? !  ip @ +! 1  slot +! ;
S1 n   256 * sn ;
S2 n   8 * sn ;
S3 n   dup 3 and drop if 7 sn s4 ; then 4 / sn ;
I, n    slot @ jump s0 s1 s2 s3 s4
```

I, springt zum Code für den aktuellen Slot. **Slot**, **h**, **ip** und **call?** sind Variablen. **Ip** ist die nächste Instruktions–Adresse. **H** ist die nächste verfügbare Adresse. **H** ist **ip**+1, außer wenn ein Literal kompiliert wurde. Slot 4 markiert das Ende der aktuellen Instruktion. Die meisten Instruktionen werden durch **i,** definiert. Zum Beispiel:

```
@    b i, ;
2*  11 i, ;
+   14 i, ;
.   1c i, ;
```

Literale brauchen besondere Aufmerksamkeit.

```
N   defer   8 f@ execute  8 i, 3ffff and , ;
```

Defer liefert die Adresse des folgenden Codes, der grünen Zahlen zugeordnet wird (alle 18–Bit–Zahlen sind auf einem 32–Bit–Host verkürzt). Danach treffen diese auf die Phrase **8 f@ execute**, die die Zahl aus dem voranalysierten Wort wie eine gelbe kurze Zahl extrahiert. Das liefert das Literal auf dem Stack. Nun wird die Instruktion 8 kompiliert und das Literal abgeschnitten und in das nächste Wort kompiliert. Bis zu 4 Literale können von der gleichen Instruktion referenziert werden, die von den tatsächlichen Literalen gefolgt wird.

Sprung–Instruktionen benutzen **adr**, das den Instruktions–Code kompiliert und die Adresse der Instruktion liefert. **Break** zwingt die folgende Instruktion in ein neues Instruktions–Wort:

Fortsetzung auf Seite 30

Adventures in Forth 5

Erich Wälde

Umstieg auf `amforth`

Am Ende meiner kleinen Artikelserie mit Code für den Renesas–`r8c`–Kontroller war ich ein wenig traurig: der Datenspeicher, in dem mein Programm aufbewahrt wird, war leider fast voll. Und dabei wollte ich jetzt erst richtig loslegen. Zunächst habe ich ein wenig mit `ByteForth` von Willem Ouwerkerk [3] herumexperimentiert. Allerdings benötigt man zum Arbeiten ein DOS. In meiner GNU/Linux–Welt lässt sich das mit `dosemu` zwar halbwegs komfortabel handhaben, aber eben nur halbwegs.

Dann bekam ich Wind von einem gewissen Matthias Trute und `amforth` [4]. Ein nagelneues Forth für die `atmega`–Familie von Atmel, angelehnt an die Artikelserie von Ron Minke [5]. Einen arbeitslosen atmega32–Kontroller hatte ich noch herumliegen. Also flugs an die Arbeit.

Es stellte sich heraus, dass ich an meinem Code für den `r8c` nur wenig Änderungen vornehmen musste. Der Umstieg war leicht.

- `amforth` ist in kleinen Buchstaben programmiert: `IF` ist schlecht, `if` ist gut.
- Etliche Worte sind erst durch Eintragen in `dict_appl.inc` verfügbar.
- statt bit–bang i2c Code ein simples `include lib/twi.frt`
- `,` (comma) schreibt ins Flash, das muss man mit `i@` lesen.
- Nur `allot` reserviert RAM, das kann man dann mit `@` lesen und mit `!` beschreiben.
- `bset`, `bclr`, `btst` fehlen.
- Das Thema mit den `include`–Dateien habe ich anders gelöst. Ich expandiere meine Forth–Dateien zuerst mit einem `perl`–Filter und übertrage dann den kompletten (um Kommentare erleichterten) Quelltext auf den Kontroller. Es gibt ein nettes python–Programm `amforth-upload.py`, welches das genauso gut erledigt.
- Ich habe zuerst die `timeup`–Uhr mit den Überläufen von `timer2` gefüttert und damit meine Zeitverwaltung realisiert. Das ist in [2] beschrieben und mit sehr geringen Änderungen auch auf `amforth` lauffähig.

Bit Flags

Inspiriert durch den Artikel *Named Bits* von M. Kalus, M. Trute [6] habe ich Bit–Variablen realisiert.

Die Definition von `flag:` ist verblüffend einfach

```
: flag: create ( addr bit -- )
  1 swap lshift , ,
does>          ( -- bitmask addr )
  dup i@ swap 1+ i@
;
```

Die Bitnummer wird in eine Maske umgewandelt (`1 swap lshift`) und gespeichert. Die Adresse der Variablen, die die Bits speichert, wird ebenfalls in Flash geschrieben. Zur Laufzeit wird die Adresse der Variablen und die Bitmaske auf den Stapel gelegt. Premiere: das war meine erste eigene Verwendung von `create ... does>`.

Zur Verwendung erzeugt man zunächst eine Variable, in der 16 Bits gespeichert werden können, dann definiert man eine Bitvariable, etwa so:

```
variable FLAGS
FLAGS 4 flag: Fdebug
```

Außerdem definiert man Worte, die die auf den Stapel gelegten Werte `bitmask` und `addr` benutzen, z.B. `fset`:

```
: fset ( bitmask addr -- )
  dup @          ( -- mask addr value )
  rot            ( -- addr value mask )
  or             ( -- addr new-value )
  swap !         ( -- )
;
```

Die Variable `FLAGS` wird in einer Lesen–Ändern–Schreiben–Sequenz bearbeitet, so dass die anderen Bits unverändert bleiben.

Die folgenden Worte habe ich definiert:

- `fset` — Bit setzen
- `fclr` — Bit löschen
- `fset?` — wahr, wenn Bit gesetzt ist
- `fclr?` — wahr, wenn Bit gelöscht ist
- `ftgl` — das Bit setzen, wenn es gelöscht ist, und umgekehrt

Das lässt sich dann auch gleich ausprobieren:

```
> variable FLAGS
 ok
> 0 FLAGS !
 ok
> FLAGS @ .
 0  ok
> FLAGS 2 flag: F_debug
 ok
> F_debug fset? .
 0  ok
> F_debug fset
 ok
> FLAGS @ .
 4  ok
> F_debug fset? .
 -1  ok
> F_debug ftgl
 ok
> F_debug fset? .
 0  ok
> FLAGS @ .
 0  ok
```

So 'ne Art Objekt: `sensor:`

Mein derzeitiges Kontrollerprogramm liest alle 10 Sekunden zweierlei Temperatursensoren aus, LM75– und LM92–Sensoren. Der Datentransfer ist bei beiden exakt identisch, nur die Anzahl der signifikanten Bits in der Antwort ist verschieden. Es gibt ein Wort `lm.get`, welches 2 Byte von der angegebenen Adresse liest:

```
: lm.get ( i2c_addr -- xh xl ok | err )
  dup twi.ping? if
      >r
  0 1 r@ >i2c
    2 r> <i2c
    0          \ ok
  else
    drop \ addr
    -1         \ err
  then
;
```

Wenn der Sensor erreichbar war, wird zusätzlich eine `0` auf den Stapel gelegt, sonst nur eine `-1`. Die beiden Funktionen `>i2c` und `<i2c` schreiben bzw. lesen N Bytes von einer i2c-Adresse (s. Listings und [1]):

```
: >i2c ( x1 .. xN.msB N addr -- ) ... ;
: <i2c ( N addr -- xN.msB .. x1 ) ... ;
```

Die beiden von `lm.get` gelesenen Bytes werden dann dekodiert und in Centigrad Celsius umgerechnet:

```
: lm75.decode ( xh xl -- T*100 [C] )
  [ hex ]
  80 and swap 8 lshift +
  [ decimal ]
  100 256 */
;
```

Die gleiche Funktion für den LM92 unterscheidet sich nur an zwei Stellen:

```
: lm92.decode ( xh xl -- T*100 )
  [ hex ]
  f8 and swap 8 lshift +
  [ decimal ]
  100 128 */
;
```

`f8` statt `80` und `128` statt `256`. Die könnte man zwar glatt als Argumente übergeben, aber das ist sehr hässlich.

Wenn man die Sensoren ausliest, muss man also immer wissen, welche Sorte man gerade ausliest:

```
...
addr_lm75 lm.get 0= if lm75.decode . then
addr_lm92 lm.get 0= if lm92.decode . then
...
```

Das fand ich doof. Also habe ich ein *Objekt* definiert, welches Zeiger auf die richtigen Funktionen speichert. Wird das Objekt zur Laufzeit aufgerufen, dann holt es die Werte vom Sensor, dekodiert und korrigiert diese und legt das Ergebnis auf den Stapel. Dann braucht's nur noch eine passende Anzeigefunktion.

```
: sensor:
  create ( id i2c-addr 'get 'decode a_0 -- )
    \ keep order of arguments in (flash) memory
    >r >r >r >r , r> , r> , r> , r> ,
  does>  ( -- id skalierterWert ok | id err )
    >r             ( -- )
      r@    i@     ( -- id )
    1 r@ +  i@     ( -- id i2c-addr )

    2 r@ +  i@     ( -- id i2c-addr 'get )
    execute        ( -- id xN..x1 ok | id err )
    0= if          ( -- id xN..x1 )
      3 r@ +  i@   ( -- id xN..x1 'decode )
      execute      ( -- id T*100 )
      4 r@ +  i@ + ( -- id T*100+a_0 )
      0            ( -- id T*100+a_0 ok )
    else           ( -- id )
      -1           ( -- id error )
    then
    r> drop
;
```

Beim Definieren eines `sensor:`–Objekts werden eine ID, die Adresse des Sensors, je ein Zeiger auf eine Funktion zum Lesen des Sensors, sowie zum Dekodieren der gelesenen Daten, und schließlich eine Korrektur (Offset) abgespeichert. Die Verwendung wird jetzt einfacher:

```
: .T ( id value ok | id err -- )
  0= if
    swap .id 2 +.f
  else
    .id ." *"
  then
;
...
01 a_th1 ' lm.get ' lm75.decode -150 sensor: T1
02 a_th2 ' lm.get ' lm92.decode  122 sensor: T2
...
T1 .T   T2 .T
...
```

Das ist jetzt so eine Art Objekt mit privaten Konstanten, ohne private Variablen und mit nur einer Methode (`does>`). Nachteilig ist, dass man hier nicht verschiedene Methoden basteln kann. Immerhin kann man mehrere Instanzen anlegen, die sich in ihrem Innenleben verschieden verhalten. Diese Verschiedenheit wird über Funktionszeiger transportiert. Die Instanzen lassen sich daher auch für andere Sensoren verwenden, die komplett anders anzusprechen sind und/oder eine andere Art und/oder Anzahl von Ergebnissen produzieren. Diese müssen nur korrekt weiterverarbeitet werden (z.B. Anzeigefunktion)

So 'ne Art Objekt: `filter_mean:`

Ich wollte die Messwerte über 10 Minuten mitteln und am Ende der 10 Minuten nur als Liste von Werten N, x_{min}, x_{mean}, x_{max} ausgeben. Beim Mittelwert ist es einfach, ich muss nicht alle Werte aufheben, sondern nur die Summe und die Anzahl der Messwerte. $x_{mean} = x_{sum}/N$, x_{min} und x_{max} können fortlaufend bestimmt werden.

Also brauche ich 4 Variablen im RAM: N, x_{min}, x_{sum}, x_{max}. Davon x_{sum} besser mit doppelter Länge.

`filter_mean:` bekommt eine ID und einen Zeiger auf den reservierten RAM–Bereich.

```
: filter_mean:
  create ( id -- )
    ,      \ store id in dictionary entry
    heap , \ store next RAM addr
    5 cells allot \ create RAM space
  does>  ( -- id ram_addr )
    dup i@
    swap 1+ i@
;
```

`filter_mean:` speichert beim Anlegen eines Filters zuerst eine ID. Dann wird die Adresse der nächsten freien RAM–Zelle gespeichert, und anschließend im RAM die nötige Anzahl von Zellen reserviert. Zur Laufzeit werden lediglich die ID und die RAM–Adresse auf den Stapel gelegt.

Die Belegung im RAM–Speicher soll mit Konstanten lesbarer gemacht werden. Dazu kommen Worte, die das Offset zu einer gegebenen Adresse addieren.

```
\ offsets into ram_addr
0 constant mean_sum
2 constant mean_N
3 constant mean_min
4 constant mean_max
: mean_sum+ ( mean_sum cells + ) noop ;
: mean_N+     mean_N   cells + ;
: mean_min+   mean_min cells + ;
: mean_max+   mean_max cells + ;
```

Drei Methoden habe ich definiert, um mit dem RAM–Bereich zu arbeiten:

- `reset ( id ram_addr - )`
  die Werte im Speicher auf Null setzen
- `addup ( x id ram_addr - )`
  einen neuen Messwert in die Daten einfügen, Dabei x_{min} und x_{max} überprüfen und ggf. aktualisieren.
- `eval ( id ram_addr - id max d.sum min N>0 | id 0 )`
  die vorhandenen Daten auswerten

Neu war für mich an dieser Stelle nur die korrekte Bearbeitung von x_{sum} als doppelt lange Variable. Dafür gibt es die Helferlein `2@`, `2!`, `s>d`, `d+` und `m*/`, die sich in der Datei `lib/ans94/2x.frt` befinden.

```
: mean_eval
( id ram_addr -- id max d.sum min N>0 | id 0 )
  >r
  r@ mean_N+ @  0=  if
    0 \ N=0, no data
  else
    r@ mean_max+  @ \ mean_max
    r@           2@ \ mean_sum
    1
    r@ mean_N+    @ \ mean_N
    m*/             \ mean_sum*1/mean_N
    r@ mean_min+  @ \ mean_min
    r@ mean_N+    @ \ mean_N
  then
  r> drop \ ram_addr
;

> decimal ok
> 10 filter_mean: F10 ok
> F10 mean_reset ok
> : .F10 cr . . d. . . ; ok
> 1000 F10 mean_addup ok
> 2000 F10 mean_addup ok
> 3000 F10 mean_addup ok
> 4000 F10 mean_addup ok
> F10 mean_eval .F10
 4  1000  2500  4000  10  ok
```

Im Unterschied zu `sensor:` enthält `filter_mean:` private Variablen und drei Methoden, die auf den Variablen operieren. Wahrscheinlich sind das keine *Objekte nach der reinen Lehre*, aber als Beispiel erhellend. Die Methoden sind hier für alle Instanzen gleich.

Referenzen

1. E. Wälde, Adventures in Forth, Die 4. Dimension 3/2006, Jahrgang 22
2. E. Wälde, Adventures in Forth 2, Die 4. Dimension 4/2006, Jahrgang 22
3. ByteForth von Willem Ouwerkerk, `http://www.forth.hccnet.nl/byteforth.htm`
4. `http://amforth.sourceforge.net/`
5. Ron Minke, Forth von der Pike auf Die 4. Dimension 3+4/2005 . . . 4/2006, sowie Sonderheft AVR und 3+4/2007
6. M. Kalus, M. Trute, Named Bits, Die 4. Dimension 2/2007, Jahrgang 23

Listings

```
\ 2008-01-01 EW   flags.fs

\ use a variable as 16 bit variables

\ usage:
\ variable mainFlags
\ mainFlags 0 flag: Fdebug
\ Fdebug fset  \ set bit
\ Fdebug fclr  \ clear bit
\ Fdebug fset? \ true if bit is set
\ Fdebug fclr? \ true if bit is NOT set

\ create: store address and bitmask
\ does>:  fetch bitmask and address
: flag: create ( addr bit -- )
  1 swap lshift , ,
does>          ( -- bitmask addr )
  dup i@ swap 1+ i@
;

\ bitvalue, convert number of bit to mask
: bv ( bitnumber -- bitmask )
  1 swap lshift
;

: fset ( bitmask addr -- )
  dup @          ( mask addr value )
  rot            ( addr value mask )
  or             ( addr new-value )
  swap !
;

: fclr ( bitmask addr -- )
  dup @          ( mask addr value )
  rot            ( addr value mask )
  invert and     ( addr new-value )
  swap !
;


: fset? ( bitmask addr -- t/f )
  @ and 0<>
;

: fclr? ( bitmask addr -- t/f )
  @ and 0=
;

: ftgl ( bitmask addr -- )
  over over      ( mask addr mask addr )
  fset?
  if fclr else fset then
;
```

```
\ 2008-10-11 EW  i2c_lmXX.fs
\ read lm75,92 temperature sensors

\ words:
\     lm.get ( i2c_addr -- xh xl ok | err )
\     lm75.decode ( xh xl -- T*100 )
\     lm92.decode ( xh xl -- T*100 )

\ read 2 bytes from sensor, if available
: lm.get ( i2c_addr -- xh xl ok | err )
  dup twi.ping? if
      >r
  0 1 r@ >i2c
    2 r> <i2c
    0 \ ok
  else
    drop   \ addr
    -1 \ err
  then
;
\ lm75: decode 2 bytes into T*100, 9bit signed
: lm75.decode ( xh xl -- T*100 )
  [ hex ]
  80 and swap 8 lshift +
  [ decimal ]
  100 256 */
;
\ lm92: decode 2 bytes into T*100, 11bit signed
: lm92.decode ( xh xl -- T*100 )
  [ hex ]
  f8 and swap 8 lshift +
  [ decimal ]
  100 128 */
;
```

```
\ 2008-01-10 EW   i2c.fs

\ send n bytes to addr
: >i2c ( x1 .. xN.msB N addr -- )
  twi.start
  twi.tx    \ uses addr
  0 do      \ uses N
    twi.tx  \ uses xN .. x1
  loop
  twi.stop
;

: <i2c ( N addr -- xN.msB .. x1 )
  twi.start
  1+ twi.tx \ uses addr
  1- dup 0 > if
    0 do twi.rx loop
  else
    drop
  then
  twi.rxn
  twi.stop
;
```

```
\ 2008-04-15 EW   sensor.fs

\ words:
\ sensor:
\ data layout
\      id id of "sensor:"
\    addr address of hw sensor on bus
\    'get pointer to read function, result:
\         N Bytes 0 (ok) | -1 (err)
\ 'decode pointer to decode function, result:
\         scaled physical value(s) of sensor
\     a_0 zero order scaled correction (offset)

: sensor:
  create ( id i2c-addr 'get 'decode a_0 -- )
```

```
    \ keep order of arguments in (flash) memory
    >r >r >r >r , r> , r> , r> , r> ,
  does>  ( -- id skalierterWert ok/err )
    >r              ( -- )
      r@    i@      ( -- id )
    1 r@ +  i@      ( -- id i2c-addr )

    2 r@ +  i@      ( -- id i2c-addr 'get )
    execute         ( -- id xN..x1 ok | id err )
    0= if           ( -- id xN..x1 )
      3 r@ +  i@    ( -- id xN..x1 'decode )
      execute       ( -- id T*100 )
      4 r@ +  i@ +  ( -- id T*100+a_0 )
      0             ( -- id T*100+a_0 ok )
    else            ( -- id )
      -1            ( -- id error )
    then
    r> drop
;
\ 2008-10-18 EW filter_mean.fs
\
\ needs ans94/2x.frt
\
\ usage:
\ 10 filter_mean: F10
\ F10 mean_reset
\ ...
\ <value> F10 mean_addup
\ ...
\ F10 mean_eval . . d. . .
\
\ RAM layout (cells)
\   0  mean_sum
\   2  mean_N
\   3  mean_min
\   4  mean_max

: filter_mean:
  create ( id -- )
    ,       \ store id in dictionary entry
    heap , \ store next RAM addr
    5 cells allot \ create RAM space
  does>  ( -- id ram_addr )
    dup i@
    swap 1+ i@
;

\ offsets into ram_addr
0 constant mean_sum
2 constant mean_N
3 constant mean_min
4 constant mean_max

: mean_sum+ ( mean_sum cells + ) noop ;
: mean_N+     mean_N   cells + ;
: mean_min+   mean_min cells + ;
: mean_max+   mean_max cells + ;

\ reset filter
: mean_reset ( id ram_addr -- )
  >r
  0 s>d r@         2! \ mean_sum=0
  0     r@ mean_N+  ! \ mean_N=0
  r> drop \ ram_addr
  drop    \ id
;

: mean_eval
( id ram_addr -- id max d.sum min N>0 | id 0 )
  >r
  r@ mean_N+ @  0=  if
    0 \ N=0, no data
  else
    r@ mean_max+  @ \ mean_max
    r@           2@ \ mean_sum
    1
    r@ mean_N+    @ \ mean_N
    m*/             \ mean_sum*1/mean_N
    r@ mean_min+  @ \ mean_min
    r@ mean_N+    @ \ mean_N
  then
  r> drop \ ram_addr
;

: mean_addup ( x id ram_addr -- )
  >r
  drop \ id
  r@ mean_N+ @  0=  if
    dup  r@ mean_min+  !
    dup  r@ mean_max+  !
  else
    dup  r@ mean_min+ @  <  if
      dup  r@ mean_min+  !
    then
    dup  r@ mean_max+ @  >  if
      dup  r@ mean_max+  !
    then
  then
  s>d  r@ mean_sum+ 2@  d+
  r@ mean_sum+ 2! \
  1  r@ mean_N+  +!
  r> drop \ ram_addr
;
```

Die EuroForth–Tagung 2008 in Wien

Ulrich Hoffmann

Zusammenfassung

Vom 26. bis zum 28. September fand in Wien die EuroForth–Konferenz mit umfangreicher internationaler Beteiligung statt. Hier ein paar Impressionen der Konferenz.

Der Wecker klingelt. Es ist Donnerstag, drei Uhr morgens — Zeit für die EuroForth 2008. Jetzt gilt es nur noch, von Hamburg nach Wien zu kommen. Die Reise läuft wie am Schnürchen, S–Bahn, Bus. Der Abflug verzögert sich ein bisschen. Um kurz vor sieben hebt die A319 ab und landet 75 Minuten später auf dem Wiener Flughafen. S–Bahn, U–Bahn, und so stehe ich gegen neun Uhr vor dem Hotel Erzherzog Rainer in der Wiedener Straße — dem Konferenzhotel der diesjährigen EuroForth-Tagung. Bevor die Konferenz am morgigen Freitag beginnt, findet heute bereits das Forth 200x–Standardisierungstreffen statt [1]. Beim Frühstücken im Hotel treffe ich Peter Knaggs, der schon am Vortag angereist war. Das Schicksal des frühen Aufstehens hat auch er erlitten. Am Nachmittag beginnt das Forth200x–Treffen mit ca. 15 Teilnehmern und es werden alte und neue Standardisierungsvorschläge durchgesprochen. Hierbei geht es um externe Floating–Point–Stacks gegenüber mit dem Daten–Stack kombinierten, es geht um Worte zum Definieren von Strukturen (Records), um lokale Variablen und weitere Themen [2].

Das Standardisierungstreffen und die EuroForth–Konferenz, die dann am Freitagnachmittag beginnt, finden in der kleinen aber feinen Bibliothek des Instituts für Computersprachen der TU Wien statt, das in einem um 1915 errichteten Gebäude in unmittelbarer Nähe der Karlskirche untergebracht ist. 17 Teilnehmer und 5 Gäste aus 9 Ländern (Südafrika, Estland, Deutschland, England, Irland, Schweiz, Österreich, Spanien, USA) haben den Weg nach Wien gefunden — die Konferenzsprache ist Englisch. Wir hören eine Reihe sehr interessanter Vorträge (siehe Tabelle 1) mit fruchtbaren und auch kontroversen Diskussionen. Wichtig, wie immer, sind die Gespräche in den Kaffeepausen.

Für das Abendessen sind für uns Tische im historischen Zwölf–Apostel–Keller reserviert, der sich in der Tat mehrere Stockwerke unter der Erde erstreckt. Den Abschluss des Abends bildet ein ausgedehnter Spaziergang mit Konferenzleiter Anton Ertl als Reiseführer durch Wiens

Teilnehmer der EuroForth–Konferenz 2008 auf dem Mozartplatz in Wien

Autor	Thema
Angel Robert Lynas, Bill Stoddart	Using Forth in a Concept-Oriented Computer Language Course
Jaanus Pöial	Java Framework for Static Analysis of Forth Programs
Andrew Haley	An Infix Syntax for Forth (and its compiler)
Bernd Paysan	A Debugger for the b16 CPU
M. Anton Ertl	Cleaning up After Yourself
Ulrich Hoffmann	Forth System Hooks—Metaobject Protocol in Forth Style
Stephen Pelc	Updating the Forth Virtual Machine
Yunhe Shi, Kevin Casey, M. Anton Ertl, David Gregg	Virtual Machine Showdown: Stack versus Registers
Bernd Paysan	Porting MINOS to VFX
Bill Stoddart, Angel Robert Lynas	Filters for Unicode Markup, a Work in Progress Report
Gerald Wodni	GLforth – an Ego Shooter
Federico de Ceballos	A high level interface to SQLite

Tabelle 1: Themen der EuroForth 2008

Weltkulturerbe–Innenstadt mit seinen prächtigen Bauten.

Am Samstagmorgen hören wie weitere Vorträge (siehe wiederum Tabelle 1). Der Ausflug am Samstag führt uns in den Nordwesten von Wien und auf eine Wanderung auf den Leopoldsberg und von dort über den Cobenzl an das Weingut am Reisenberg, wo wir den Abend ausklingen lassen.

Am Sonntagmorgen finden die Workshops statt. Wir diskutieren neue Ansätze für lokale Definitionen, dann Worte zur Manipulation von System–Hooks und auch die Einbettung von SQL in Forth. Eine weitere Diskussion dreht sich um die Frage, wie man portable Bibliotheken definiert. Im letzten Workshop diskutieren wir *Root Forth*: Die Idee, ein kleines, modernes, standard–konformes Forth–Assembler–Listing im Stile der FIG–Forth–Listings zu haben. Es soll einen einfachen Einstieg in Forth bieten, vergleichbar mit der Art, wie viele von uns Forth gelernt haben. Diese Idee trifft auf allgemeines Interesse. Wer bringt den Stein ins Rollen?

Der formale Teil der Konferenz endet am Sonntagmittag.

Es war wieder eine phantastische Forth–Konferenz mit vielen interessanten Gesprächen und Diskussionen. Wien selbst bot das dazu passende Ambiente. Mein herzlicher Dank gilt Anton Ertl, der die EuroForth in hervorragenster Weise vorbereitet und geleitet hat, und auch Ewa Vesely, die im Hintergrund für das gute Gelingen gesorgt hat. Die Messlatte liegt hoch.

Die EuroForth 2009 soll in Großbritannien stattfinden.

Ulrich Hoffmann

Links

[1] `http://www.forth200x.org/`
[2] `http://www.forth200x.org/meetings/agenda2008.pdf`
[3] `http://www.complang.tuwien.ac.at/`

Lebenszeichen

Bericht aus der FIG Silicon Valley: *Henry Vinerts*

Dear Fred,

ich hatte dir einen Bericht über die alljährliche SVFIG–Forth–Tagung versprochen, sofern ich Zeit fände, sie zu besuchen. Nun, ich war heute da und ich war hocherfreut darüber, auf diesem Treffen mehr Leute zu sehen als auf den Forth-Tagungen etwa der letzten 10 Jahre. Im Verlauf des Tages trafen mehr als 45 Teilnehmer ein, und das nicht nur, weil Dr. Ting alle und jeden mit seinem traditionellen Barbecue-Lunch verköstigte. Den Vogel schoss Chuck Moore mit seinen acht Mitstreitern von IntellaSys ab, die mit ihm gemeinsam über den neuesten SEAforth–40C18–Chip sprachen. Der Chip ist das Ergebnis ihrer engen Zusammenarbeit. Begleitet wurden die Vorträge von Vorführungen. Ich möchte nicht ins Detail gehen, da du darüber ja unter `http://www.intellasys.net/` mehr erfahren kannst. Wir erlebten an diesem Tag zwölf Darbietungen, die von Chuck mit seinem traditionellen *fire-side chat* (Kamingeplauder) beendet wurden. Dabei nahm uns Chuck an die Hand auf dem Weg durch seine 40 Jahre Erfahrung mit Forth, wobei er schließlich mit einer Beschreibung der Funktionen des 40C18 endete.

Zu erwähnen wäre noch, dass uns der Präsident der Firma Forth–Inc., Leon Wagner, während der Vormittagssitzung über seine Eindrücke von der kürzlich abgehaltenen EuroForth-Konferenz berichtete. Er verteilte ein paar Unterlagen von `www.forth200x.org`, die aus den Vorschlägen von Dr. Anton Ertl zur Aktualisierung des alten Forth-Standards aus dem Jahre 1994 stammen. Leon sagte, dass er sich über die Gastfreundschaft und den herzlichen Empfang, die ihm auf der Konferenz entgegengebracht wurden, sehr gefreut habe, dass er uns aber auch nicht verschweigen könne, wie oft er bei Diskussionen um die neuen Standards die Frage *Wo sind denn die Amerikaner?* zu hören bekam.

Soweit ich mich erinnere, habe ich über dieses Problem schon ein anderes Mal geschrieben. Ich hoffe, dass der Hauptgrund nur in dem unseligen Handicap besteht, den ich mit einem meiner Lieblingswitze so umschreiben möchte:

- *Wie nennt man eine Person, die mehrere Sprachen spricht?* Antwort: *Mehrsprachig.*
- *Wie nennt man eine Person, die zwei Sprachen spricht?* Antwort: *Zweisprachig.*
- *Wie nennt man eine Person, die nur eine Sprache spricht?* Antwort: *Einen Amerikaner.*

Chuck schlug vor, dass jeder neue Standard eigentlich in zwei Teile aufgespalten werden sollte, einen Grundbestandteil für kleinere Forth–Anwendungen und einen *Mega*–Standard, der die Aufgabe habe, alles zu erfassen, was jemals im Namen von Forth produziert wurde. Leon fügte hinzu, solche Ideen seien schon dabei, umgesetzt zu werden. Ich kann mich noch gut an einen Tag vor einigen Jahren erinnern, als ich Chuck sagen hörte, er sei für keine Standards zu haben und wenn es doch zu solchen käme, dann sei es an der Zeit, sich anderen Dingen zuzuwenden.

Hm, wenn auch ich meine bescheidene Meinung zu diesem Thema sagen darf: In den Jahren meiner Tätigkeit als Konstrukteur habe ich viele Standards kennengelernt und dabei beobachten können, dass sie größtenteils nicht befolgt wurden. Und zwar aus verschiedenen Gründen: Hauptsächlich, weil sie zu undurchsichtig, zu willkürlich und in ihrer peniblen Anwendung zu umständlich waren. In den allermeisten Fällen behält der festgetrampelte Pfad über die Wiese über jegliche Art von architektonisch begründetem Standard die Oberhand. Aber es ist nicht meine Aufgabe, über die Standardisierung von Forth nachzudenken.

Meinen Bericht möchte ich jedenfalls mit der Bemerkung abschließen, dass ich nach dem heutigen Tag aus tiefer Überzeugung sagen kann, dass Forth im Silicon Valley voll weiterlebt.

Mit meinen besten Wünschen,

Henry Übersetzt von Fred Behringer

Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 68, Juni 2008

Forth vanaf de grond 10 — Ron Minke

Rons Artikel nimmt den übergroßen Teil des diesmaligen Vijgeblaadjes ein. Seine Artikelserie ist uns aus der Vierten Dimension unter dem Titel *Forth von der Pike auf* bekannt geworden. Der Teil 10 ist inzwischen im VD–Heft 2/2008 in deutscher Übersetzung erschienen.

Communicatiepoorten op nieuwere computers — Frans van der Markt

Bei den neueren Computern gibt es kaum noch serielle oder parallele Anschlüsse. Außerdem macht Windows, und insbesondere Vista, Schwierigkeiten mit beispielsweise HyperTerminal oder ähnlichen Programmen, die beim Kommunizieren mit Forth–Boards ständig gebraucht werden. Frans berichtet über Experimente mit einem Anpassungskabel, das per USB eine serielle Schnittstelle emuliert.

Poortbesturing op het internet — Paul Wiegmans

In Win32Forth ist der direkte Zugang zur parallelen Schnittstelle nicht möglich. Paul hat einen Weg gefunden, das Unmögliche möglich zu machen:
`http://sikkepitje.ath.cx/%7Epaul/w/index.php/Win32Forth#Win32Forth_en_inpout32.dll`

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 69, August 2008

SD–kaart besturing (deel 1 van 3) — Willem Ouwerkerk

Dieser Artikel über die Ansteuerung von SD–Karten erstreckt sich diesmal über den gesamten verfügbaren Platz des Vijgeblaadjes. Willem sagt in seiner Einleitung: Hier kommt eine Beschreibung der SD–Karten–Routinen, so wie sie auf 8052–ANS–Forth aufsetzend entwickelt wurden und für das in Entwicklung befindliche F51–Board bestimmt sind. Der Code soll in drei Teilen beschrieben werden. Der dritte Teil ist etwas umfangreicher und wird wohl über mehr als ein Vijgeblaadje verteilt werden müssen.
Teil 1: Die Schnittstellen zur SD–Karte
Teil 2: Ein einfaches Blocksystem
Teil 3: Ein Dateisystem (FAT16).

Soweit die einleitenden Worte von Willem. Aus dem Inhalt: Es gibt zwei Möglichkeiten, das SD–Protokoll und ein SPI–Interface. Eine brauchbare Beschreibung existiert nur für das SPI–Interface. Das Forth für die F51–Karte wird innerhalb einer halben Sekunde von der SD–Karte eingeladen und gestartet. Gelesen und geschrieben wird sektorweise, wobei ein Sektor 512 Bytes umfasst. Die Sektoradressen sind 32 Bit breit. Für den Rezensenten interessant ist die Routine DUM*, die eine 32–Bit–Zahl mit einer 16–Bit–Zahl multipliziert und das Ergebnis wieder in ein 32–Bit–Paket steckt:

```
: DUM* ( ud1 u -- ud2 )
    >R R@ UM* SWAP ROT R> UM* D+ ;
```

Man vergleiche den Euler–9–Artikel des Rezensenten im vorliegenden VD–Heft — und man experimentiere.

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 70, Oktober 2008

Forth vanaf de grond 11 — Ron Minke

Ron präsentiert einen elften Teil seiner bei uns als *Forth von der Pike auf* in der Vierten Dimension erschienenen Artikelserie.

Mededelingen van het bestuur — Redactie

Der Rezensent übersetzt die gesamte Mitteilung des Vorstandes: Unlängst ist der Plan entstanden, 2010 zusammen mit unseren deutschen Mitstreitern in Sachen Forth eine gemeinsame Tagung zu veranstalten. Die Deutschen haben dann ihre jährliche Forth–Tagung, die drei Tage dauert und ungefähr 25 Teilnehmer erwarten lässt. Am zweiten Tag (einem Samstag) wäre für Niederländer und Deutsche zusammen ein Programm von forthbezogenen Vorträgen zu organisieren.
Was denken unsere Mitglieder darüber? Wir könnten dieses Vorhaben auf unserem nächsten Treffen besprechen. Solltet ihr da nicht kommen können, so besteht immer noch die Möglichkeit, per E–Mail zu reagieren:
forthmail3@hccnet.nl
Die vorliegende Ausgabe ist die letzte papierene Version des Vijgeblaadjes. Die Finanzierung der Gebruikersgroeps innerhalb der HCC hat sich geändert und es besteht keine Möglichkeit mehr, das Vijgeblaadje in der vertrauten Form per Post zu verschicken. Wenn ihr trotzdem

auf der Höhe des Forth–Geschehens bleiben wollt, könnt ihr eine E–Mail an die oben genannte Adresse schicken. Wir können euch dann einen Newsletter per E–Mail zukommen lassen.

Das Treffen im Dezember entfällt. Es wird auf den zweiten Samstag im Januar 2009 verschoben. Ansonsten bleibt das Schema für 2009 unverändert: Jeden zweiten Samstag eines geraden Monats.

I/O–poorten en DLL's in Win32Forth — Paul Wiegmans

Übersetzung der ersten paar Zeilen: Früher war alles besser. In DOS und Windows 95/98 konnte man in den Programmen die Ports des PCs direkt ansprechen und Signale zum Auslesen und Einlesen unmittelbar an die serielle oder parallele Schnittstelle schicken. Später änderte sich das. Windows NT und das daraus abgeleitete Windows 2000, XP und Vista geben den Anwendungen keine Möglichkeit mehr, auf die Ports zuzugreifen. Der Zugriff bleibt den Geräte–Treibern im Kernel–Modus vorbehalten. – Paul bespricht die Datei INPOUT32.DLL für Win32Forth aus http://sikkepitje.nl/wiki/index.php/Win32Forth .

Redactie — Frans van der Markt

Das Vijgeblaadje erscheint per E–Mail um den Ersten eines jeden geraden Monats herum.

Dump — Kleine Helfer fürs amforth

Michael Kalus

In Microcontrollern wie dem AVR–Atmega gibt es drei verschiedene Speicherbereiche — Ram, Flash und Eeprom — die unterschiedlich organisiert sind. Auf das Ram kann byteweise zugegriffen werden. Das Flash wird zellenweise angesprochen, ebenso wie das Eeprom. So braucht jeder Speicherbereich sein eigenes fetch, um daraus einen Wert zu lesen und auf den Forth–Daten–Stack zu holen. `i@` `e@` und `@` erledigen das im amforth[1].

Für allzu kompliziert formulierte *Dumper* ist da z. B. im Butterfly kein Platz. Üppigere Versionen von `dump`, wie für Ram–basierte Systeme, kann man nicht importieren. Deshalb macht man sich seine eigenen kleinen Helfer. Für jeden Bereich ist ein eigenes `dump` schnell gemacht; je ein `idump` für das Flash, `edump` für das Eeprom und `dump` für das Ram. Gut ausfaktorisiert hat der Code dann zwar zunächst mehr Header, liefert dafür aber gleich die Minitools wie das `?` (Fragezeichen) mit. Mit dem Fragezeichen wird bekanntlich der Inhalt einer Adresse angezeigt. Hier gibt es dann naturgemäß ein `i?` `e?` und `?` sozusagen gratis dazu. Da `dump` traditionell auch die ASCII–Werte der Daten mit anzeigt, aus Platzgründen aber auf komplizierte Aufbauten hier im amforth verzichtet wird, gibt es zumindest ein `i??` `e??` und `??`, welche neben dem Wert in einer Adresse auch seine ASCII–Werte wiedergibt. Dargestellt wird immer der Wert einer Zelle (cell), also 2 Bytes. Dies geschieht, weil im Flash jede Adresse einer Zelle entspricht und der Einfachheit halber im Forth mit `@` auch aus dem Ram und Eeprom gleich eine Zelle geholt wird. Das folgende Beispiel illustriert die Verwendung. Die Dumper liegen auch als Assemblerquelle vor.

```
> : test  ." hallo" ;  test
hallo ok
> ' test 10 - 20 idump
 C58   4B2 1E98 1DC3 1E63 1E63  797 1C8D   72
 C60    98 1E98 1EAD 1DC3 1E5C 1E5C  797 1C41
 C68  7404 7365   74 1FEB 1C0A  235 6805 6C61
 C70  6F6C  261 1C41 FFFF FFFF FFFF FFFF FFFF
 C78  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
 ok

> c68 i??
7404 t.  ok
> c70 i??
6F6C ol  ok
```

Wörter

`i?` `e?` `?` `i??` `e??` `??` `idump` `edump` `dump`
Die übrigen Wörter dieser Quelle können auch headerless gehalten werden, wenn man mag.

Quellen

`http://amforth.sourceforge.net/`
`http://www.forth-ev.de/wiki/doku.php/words:dump`

Listing

```
1   hex
2
3   ( item -- )
4   : .item     4 u.r space ;  \ asm ok
5
6   ( addr -- )
7   : i?     i@ .item ;
8   : e?     e@ .item ;
9   : ?       @ .item ;
10
11  ( addr n -- addr+n )
12  : .icells   0 do  dup i?    1+ loop ; \ flash
13  : .ecells   0 do  dup e? cell+ loop ; \ eeprom
14  : .rcells   0 do  dup  ? cell+ loop ; \ ram
15
16  ( addr -- )
17  : .addr     cr .item space ;
18
19  ( addr1 len1 -- addr2 len2 )
```

[1] Das `i` im `i@` stammt aus der Bezeichnung des Flash als *instruction memory*.

```
20  : trim     swap fff8 and swap 7 or ;
21
22  ( addr len -- ) \ numbers are in hex.
23  : idump   trim 0 ?do   dup .addr   8 .icells   8 +loop ;
24  : edump   trim 0 ?do   dup .addr   8 .ecells  10 +loop ;
25  :  dump   trim 0 ?do   dup .addr   8 . cells  10 +loop ;
26
27  \ tested ok on amforth-2.9 06.10.2008 mk
28
29  \ type a character
30  : .ascii ( c -- )  7F and dup 20 < if drop 2E then emit ; \ ignore bit8
31
32  \ type content of cell and its ascii values.
33  : .aa  ( adr -- )  dup ff00 and ff / .ascii  00ff and .ascii  space ;
34
35  \ inspect an address
36  ( adr -- )
37  : i??     i@ dup .item .aa ;
38  : e??     e@ dup .item .aa ;
39  :  ??      @ dup .item .aa ;
40
41  \ tested ok on amforth-2.9 08.10.2008 mk
42  \ ca. 222 cells of flash
43  \ finis
```

Fortsetzung des Artikels von Chuck Moore von Seite 18

```
BREAK      4  slot ! ;
ADR n-a    slot @ -2 and drop if . adr ; then
           s3? if . adr ; then  i,  ip @ break ;
CALL       defer a   ff and 2 adr +! ;
IF -a      6 adr ;
-IF -a     7 adr ;
THEN a     h @ ff and swap +! 0  call? ! break ;
UNTIL a    ff and 7 adr +! ;
```

Adr kompiliert **.** bis Slot 0 oder 1 verfügbar ist. **Until** springt zurück bis der Stack negativ wird.

```
;   call? @ dup 4000 or drop if dup 200 or drop if
   0 and i, ; then then 2/  ip @ +! ;
```

; überprüft, ob die letzte Instruktion ein Unterprogrammaufruf war. Wenn ja, so wird der Aufruf in einen Sprung geändert. Das ist *Endrekursion* und spart eine Return–Instruktion, ohne Syntax für einen Sprung zu benötigen. **call?** ist eine Variable, die die letzte Instruktion enthält. Ihr Wert ist geshiftet und drückt so aus, welcher Slot belegt ist.

Target code

Hier ist ein Beispiel für Zielcode. Der c18 wird gebeten seinen Flash–Speicher zu lesen und in internes RAM zu legen.

```
90 org
BIT --1  -1 64 begin  over   + until drop ;
@F  --1n    36 begin  bit b! + until b@ ;
WORDS na    a! begin  @f !+  + until drop ;
80 org  102 b!  63 0 words
```

Dieser Code nutzt den Vorteil aus, dass **until** bei einer -1 auf dem Stack endet. **Bit** zählt eine halbe Clock–Periode. **@f** erzeugt 18 Clock–Impulse und liest das Eingabe–Shift–Register. **B!** schaltet in diesem Fall die Clock–Leitung um. **Words** liest und speichert die Eingabe–Worte.

Bestimmter Code wird standardmäßig im ROM stehen. Zum Beispiel Schiebe– und Multiplizier–Routinen:

```
*7 nn-nn   *+ *+ *+ *+
           *+ *+ *+ ;
2*15 n-n   2* 2* 2*
2*12       2* 2* 2*
2*9        2* 2* 2*
2*6        2* 2* 2*
           2* 2* 2* ;
2/15 n-n   2/ 2/ 2/
2/12       2/ 2/ 2/
2/9        2/ 2/ 2/
2/6        2/ 2/ 2/
           2/ 2/ 2/ ;
```

Diese vordefinierten Routinen aufzurufen, tauscht Zeit gegen Platz ein. Ein Unterprogrammaufruf braucht 3–4 Slots und 3 Zyklen. Ein in-line 2*15 würde die Zyklen sparen, aber braucht so 19 Slots, weil 2+ nicht in Slot 3 sein kann. Natürlich, um 8 Stellen nach links zu Schieben:

```
2* 2* 2*6
2* 2*6 2*
2*6 2* 2*
```

was immer auch am besten packt.

Übersetzt von Ulrich Hoffmann

[1] Englisches Original unter `http://www.complang.tuwien.ac.at/anton/euroforth/ef01/moore01a.pdf`

Forth–Gruppen regional

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

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

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

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

Gruppengründungen, Kontakte

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

µP–Controller Verleih

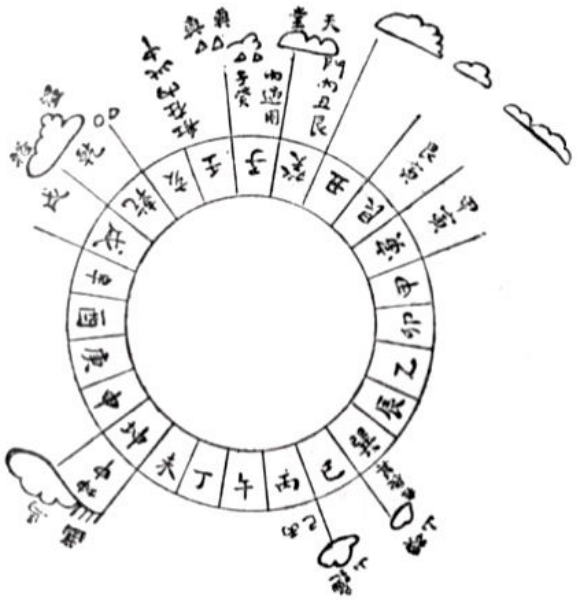
Carsten Strotmann
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix)
Klaus Schleisiek–Kern
Tel.: **(0 40) – 37 50 08 03** (g)

KI, Object Oriented Forth, Sicherheitskritische Systeme
Ulrich Hoffmann
Tel.: **(0 43 51) – 71 22 17** (p)
Fax: **– 71 22 16**

Forth–Vertrieb
volksFORTH
ultraFORTH
RTX / FG / Super8
KK–FORTH
Ingenieurbüro
Klaus Kohl–Schöpe
Tel.: **(0 70 44)–90 87 89** (p)



Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen? Schreiben Sie einfach der VD — oder rufen Sie an — oder schicken Sie uns eine E–Mail!

Hinweise zu den Angaben nach den Telefonnummern:
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.

Einladung zur

Forth–Tagung 2009

vom 27. bis 29. März 2009

in Wilminks Parkhotel (`www.wilminks-parkhotel.de`)

Wettringer Straße 46, 48485 Neuenkirchen (bei Rheine)

Programm

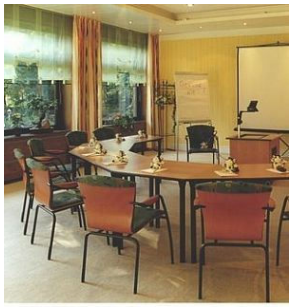
Donnerstag, 26.03.2009
14:00 Uhr Forth–Schulung

Freitag, 27.03.2009
9:00 Uhr (Rad-)Wanderung
Forth–200x–Standard–Treffen
Forth–Schulung
14:00 Uhr Beginn der Forth–Tagung

Samstag, 28.03.2009
14:00 Uhr Ausflug (z. B. Bagno in Steinfurt)

Sonntag, 29.03.2009
09:00 Uhr Mitgliederversammlung
14:00 Uhr Ende der Tagung

Anreise siehe: `http://www.wilminks-parkhotel.de/de/anfahrt_wegbeschreibung/`



Über das Parkhotel Wilminks:

„Willkommen in Neuenkirchen! Mit diesen Worten werden Sie in rund 18 gleichnamigen Orten in ganz Deutschland begrüßt. Verwechslungen sind da nicht ausgeschlossen. Doch ein Neuenkirchen, gelegen in der ländlichen Idylle Westfalens, ist unverwechselbar. Es hat nämlich etwas ganz Besonderes: ein familiengeführtes Hotel mit so etwas wie »Eigenleben«. Keine Sorge, damit meinen wir nicht etwa Holzwürmer – sondern eine bewegende Atmosphäre, die es schafft, jeden zu überraschen, zu begeistern und das Gastsein neu kennen zu lernen...

Unser Hotel ist so vielseitig wie unsere Gäste. So hat zum Beispiel jedes Zimmer seinen ganz eigenen Charakter. Mal klassisch, mal modern, mal romantisch, mal anders. “

Ideal also auch für unsere Forth–Tagung mit ihren Vorträgen, Workshops und Nachtsitzungen.

Quellen: `http://www.wilminks-parkhotel.de/`