Vierte Dimension

1/2010

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

Manuskripte und Rechte

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

Liebe Leser,

herzlich willkommen zur ersten Ausgabe des 26. Jahrgangs unseres Forth-Magazins.

Dieser Winter 2009/10 hat uns eine Menge Schnee gebracht und wir hoffen alle, dass sich bis zu unserer Jahrestagung Ende März der Winter aus diesen Breiten verzogen hat und wir in Warnemünde Frühlingsluft schnuppern dürfen. Ich freue mich schon darauf und hoffe, dass Ihr das vorliegende Heft noch vor der Tagung in den Händen halten werdet.

Was bietet uns also dieses Heft. Nun, Michael Kalus und Adolf Krüger berichten uns, wie man auf den ATMega–Prozessoren Interrupts so behandelt, dass die Service–Routine nicht in Assembler, sondern in Forth programmiert werden kann. Der Artikel von Karsten Roederer stellt das FlashForth für PIC–Microcontroller vor und das Glossar gibt einen guten Überblick über dieses bisher von uns wenig beachtete Forth–System. In Fred Behringers sechstem Teil seiner Bootsektor–Serie lernen wir, wie man Bootsektoren zuverlässig kopiert. Das ist insbesondere für Sicherheitskopien wichtig, die man vor Experimenten mit dem Bootsektor und der Partitionstabelle unbedingt anfertigen sollte! — Was geschieht, wenn man die Programmiersprache APL mit Forth kreuzt? Bernd Ulmann gibt in seinem Artikel die Antwort darauf: Er nennt seine Hybrid–Programmiersprache `5` und stellt sie ab Seite 24 vor. Mit der Implementierung einer weiteren — nicht ganz ernst gemeinten — Programmiersprache, `fHQ9+`, geht es weiter. Michael Kalus zeigt, wie man einen Compiler für diese esoterische Sprache ganz einfach in ein paar Zeilen Forth realisieren kann.

Viele Microcontroller werden über eine RS232–Schnittstelle angebunden und die Erfahrung zeigt, dass das immer wieder auch kleine und große Probleme mit sich bringt. Insbesondere der richtige Umgang mit den Handshake–Leitungen ist nicht einfach. Ganz auf sie zu verzichten, ist auch nur eine halbe Lösung. Unsere Host–Forth–Systeme laufen heute im Wesentlichen unter Linux, Mac OS X und Windows und für alle sieht die Anbindung der RS232 anders aus. Häufig kommen auch USB/serielle–Adapter mit ihren Spezifika ins Spiel. Nach meiner Beobachtung gibt es beinahe ebenso viele RS232–Routinen wie Forth–Systeme selbst :-) Die unterscheiden sich erheblich in ihrem Funktionsumfang und in der Art, wie man sie aus Anwendungsprogrammen heraus anspricht.

ANS–Forth kennt Standard–Worte für die Positionierung des Cursors auf dem Terminal und ist damit eine der wenigen Programmiersprachen, mit denen man portable Full–Screen–Anwendungen realisieren kann.

Wie könnte also ein einheitlicher Wortschatz zum Ansprechen der seriellen Schnittstelle aussehen, auf dessen Basis man plattformunabhängig portable RS232–Kommunikationsprogramme erstellen könnte?

Und wieder möchte ich Euch aufrufen, Artikel für die *Vierte Dimension* zu schreiben. Ihr macht alle spannende Dinge, über die wir gerne berichten wollen. Schickt mir ganz unkompliziert eine Mail (uho@xlerb.de).

So, nun viel Spaß beim Lesen des Hefts,

Ulrich Hoffmann

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

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

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

Warum gibt es die Kopf– und Code–Abschnitte in Forth?

In comp.lang.forth löste Helmut Giese neulich einen interessanten Austausch über Forthsysteme aus, die abschnittsweise angelegt sind. Er fragte nach dem Sinn von Kopf– und Codeabschnitten [1]. Es wurden dort etliche Möglichkeiten der Aufteilung je nach Problemstellung aufgezeigt, also nicht nur diese beiden Abschnitte.

Darüber hinaus hat mich die Bemerkung von John Passaniti angesprochen, dass es sicherlich Gelegenheiten gäbe, in denen auch der Compiler aus der Application entfernt werden muss, er aber selbst dann gerne einen minimalen Lademechanismus darin belässt, mit dem Code in das ansonsten abgeschlossene System nachgeschoben und ausgeführt werden kann. Bis hin zu einem vollständigen Forth. Er erwähnte dabei das *Drei–Instruktionen–Forth* (The *three instruction Forth* idea). Damit meinen viele wohl die Funktionen: Lade den Speicher, lese den Speicher, rufe eine Routine auf.

Bernd Paysan wies darüber hinaus auf eine andere Möglichkeit hin, die *forthiger* ist und mal von Egmont Woitzel veröffentlich wurde. Dabei werden die drei Instruktionen

```
2KEYS KOMMA EXECUTE
```

als Lader vorgesehen. `2KEYS` liest ein 16bit Wort von der seriellen Schnittstelle (weil die Embedded–Controller gewöhnlich 16–Bitter sind), `KOMMA` kompiliert sie ins Wörterbuch, und `EXECUTE` führt ein Wort aus. `EXECUTE` ist dabei implizit angelegt, weil der Monitor (serial handler) nur aus der Schleife

```
BEGIN 2KEYS EXECUTE AGAIN
```

besteht. Diese Worte benutzen alle die Stacks. Und dann ist `@` (fetch) gewöhnlich das erste Wort, welches mit dem Lader erzeugt wird. Um einen 16–Bit– Wert in das Verzeichnis zu kompilieren, sendet man dann

```
<2keys> <literal> <,>
```

über den Draht. mk

[1] comp.lang.forth vom 09.01.2010: Why are there 2 sections: code and header?

Whois?

Liebe Leser,
es gibt auf unserer Webseite `www-forth-ev.de` auch die Kategorie *Whois?* und die Frage ist, muss ich den Titel eindeutschen oder nicht? Und wenn dann wie?

Ein komplettes *who's who* der Forthgesellschaft ist es natürlich nicht. Da werkelt kein Redakteur, der bezahlt seinen Forschungsschwerpunkt im abendländischen (Forth)Kulturkreis hat, und eine Personenenzyklopädie herausgeben könnte. Aber vielleicht will ja doch der eine oder andere sich noch eintragen.

Eintragen kann man sich selbst dort, es ist einfach nur ein blog. Mich würds freuen, Euch dort zu sehen.

Kleines und verständliches Forth

Am 25. Dezember fand ich folgende Frage von deech auf comp.lang.forth:

Hi Alle,
ich bin ein Langzeitleser, aber poste erstmalig. Forth fasziniert mich, seit ich das JonesForth [1] gelesen habe. Gibt es eigentlich noch andere kleine verständliche Forth–Implementierungen, vielleicht auch welche die für echte Arbeiten benutzt werden? Ich trage mich mit dem Gedanken, mir ein eigenes kleines Forth zu machen, und würde gern verschiedene Implementierungen vergleichen bzw. gegenüberstellen. So etwas wie gforth ist zu groß!
Danke!
–deech

[1] `http://www.annexia.org/forth`
[2] `http://www.jwdt.com/~paysan/gforth.html`

Und mat antwortete:
Hallo, ein minimalistisches und simples Forth–System ist:
`www.retroforth.com`
Die Versionen <10 sind als Unterprogrammaufrufe (subroutine threaded) gemacht, das aktuelle System wurde vollständig neu geschrieben und basiert auf einer kleinen virtuellen Stack Maschine (VM). Du kannst da auch mit einer erweiterten und schnelleren VM–Entwicklung herumspielen, die einen (bisher noch rudimentären) AOT–Compiler bietet, um nativen Code zu bekommen (native code generation):

`http://github.com/Mat2/extended-ngaro`

4p ist ein schnelles Forthsystem mit JIT–Compiler:
`http://maschenwerk.de/` Dieses Forth wird seit Jahren für produktive Arbeit eingesetzt, soweit ich weiß. Das gilt auch für Reva:

`http://ronware.org/reva/`

Die genannten sind nicht ANS–kompatibel, aber verwandt.

ANS–kompatibel:
Wenn du lieber in einer betriebssicheren Umgebung lernen möchtest, versuche:

`http://home.arcor.de/a.s.kochenburger/minforth.html`

Weitere Forths, die dich interessieren könnten, sind:
`http://www.gnu.org/software/gforth/`
`http://home.hccnet.nl/a.w.m.van.der.horst/ciforth.html`
`http://code.google.com/p/fina-forth/`

Und falls du mit Nonkonformistischem, aber Innovativem experimentieren möchtest:
`http://rainbowforth.sourceforge.net/`
`http://www.colorforth.com/`
`http://muforth.nimblemachines.com/`
`http://christophe.lavarenne.free.fr/ff/`
–Mat.

Vergnügliches Erkunden. mk

Interrupt–Service–Routine für den atmega168: Uhr mit MAIN–Wort

Michael Kalus, A. Krüger

Klar ist wohl, dass zeitkritische Interrupts sofortigen Service erfordern. Sie müssen also zwingend auf der Maschinencode–Ebene angestoßen werden. Aber lange nicht alle Prozeduren sind dann in der folgenden Ausführung wirklich so zeitkritisch, dass sie komplett in Maschinencode formuliert sein müssten. Also möchte man Forth benutzen, weil es einfacher zu handhaben ist[1]

Doch sollte man dann alles in Forth machen? Gibt es bei Steuerungsaufgaben überhaupt Prozeduren, die weder pünktlich noch schnell bearbeitet werden müssten, also tatsächlich einem irgendwie gearteten Forth–Tasker überlasen werden könnten? Letzteres möchten wir praktisch verneinen. So haben wir das klassische Konzept benutzt, eine ISR immer durch den dem Prozessor eigenen Mechanismus zu beginnen. Interrupts können sich dann unterbrechen. Register werden einfach auf den Returnstack gesichert, und von dort zurückgeholt. Es ist einfach und übersichtlich. Setzt allerdings voraus, dass man seinen Prozessor kennt — bzw. zieht nach sich, dass man ihn kennen lernt. Auf Forth High Level muss man deswegen jedoch keineswegs verzichten. Das kommentierte Listing zeigt, wie ein Schrittmacher für eine Uhr auf dem ATmega168 gemacht werden kann. Aber erstmal schildern wir das Prinzip.

So gehts

Über den Interrupt–Vektor des Timer2 wird zunächst die zugehörige ISR angesprungen. Diese ruft schließlich das Forthwort auf, welches den zeitunkritischen Service macht. Forth kehrt dann zu seiner aufrufenden ISR zurück, um von dort aus den Interrupt zu benden. Auf diese Weise rettet jede ISR ihre wichtigen Systemzustände, sie sind gekapselt. Und damit gegenseitig unterbrechbar.

Das folgende Beispiel für den atmega168 und amforth3.6 zeigt die Details — den Assembler von Lubos Pekny hatten wir zuvor in den atmega168 auf das amforth geladen. Zunächst probieren wir das Gesagte als Code–Routine aus, also noch ohne den Interrupt zu benutzen.

```
code test
 XH push,           ( sichere alte IP)
 XL push,
 here 4 + dup
  ff and  XL swap ldi, ( setze neue IP)
  100 /   XH swap ldi,
end-code ( next = neue IP )
    ." hallo world" ( Ist *nicht* statesmart, )
             ( compiliert sofort ins flash. )
  [ here 1+ ,       ( Zeiger auf xt )
    here 1+ ,       ( Zeiger zum folgenden Code )
 XL pop,            ( hole alte IP )
 XH pop,
end-code            ( next = alte IP )
>
 ok
> test
hallo world ok
```

Uhr für den atmega168

Nun wenden wir diese Technik für eine ISR an, um einen Forth–Schrittmacher zu erzeugen. Im atmega168 arbeitet der Timer2 und kann freilaufend einen Überlauf–Interrupt auslösen. Damit ist er gut geeignet, um so einen Schrittmacher zu treiben. Das Listing zeigt im Detail, wie es gemacht wird.

Um für die Uhr eine Sekunde zu erzeugen, lassen wir den Timer einfach im gegebenen Prozessortakt laufen[2], geteilt durch den prescaler=1024. Bei jedem Nulldurchgang des Timer2 wird dessen Interrupt tim2ov ausgelöst, und der Prozessor geht in die tim2ISR. Zunächst wird darin nur das Nötigste auf den Returnstack gerettet, um sofort einen Phasen–Akkumulator weiter zu zählen. Nur bei dessen Nulldurchgang wird die Sekunde eins höher gesetzt, sonst verlassen wir die ISR sofort wieder. So erhalten wir einen recht genauen Sekundentakt aus einer maximal schnellen ISR und ohne den Inner–Interpreter (next) zu bremsen.

Erst wenn die Sekunde inkrementiert wird, geht die ISR durch eine Prozedur, bei der die benötigten Zeiger[3]des Forthsystems auf den Returnstack gerettet werden. Und springt dann nach dem oben vorgestellten Prinzip das

[1] Damit es nicht in Vergessenheit gerät, sei hier betont, dass auch Forth Maschinensprache ist. Denn letztendlich laufen auch die Forthworte auf der untersten Ebene als Instruktionen der zugrunde liegenden Maschine ab. Der innere Interpreter ist low level codiert. Jedenfalls gilt dies immer, wenn die Maschine nicht selbst generisch Forth ausführt. Kennt man den Aufbau des Forth–Systems, kann man es auch für eigene Zwecke manipulieren. Am Beispiel des amforth für den atmega168 ist das schön zu sehen.

[2] Der Software–Phasen–Akkumulator ermöglicht es, bei beliebigem Prozessortakt — Quarze sind auch nicht ganz genau, und außerdem temperaturabhängig — im Mittel eine recht genaue Zeit zu erhalten. Es braucht lediglich ein passendes Vielfaches. Reichen zwei Bytes dafür nicht, kann ohne weiteres eine feinere Teilung gemacht werden, indem man drei oder noch mehr Bytes benutzt; 32–Bit–Akkus sind üblich. In diesem Verfahren wird statt genauer Auflösung — die ja letztendlich Illusion ist — ein Phasenrauschen in Kauf genommen, besser: bewusst genutzt. Dabei sind die aufeinander folgenden Sekunden zwar durchaus verschieden lang, diese Abweichungen schwanken aber um einen Mittelwert, so dass sich letztendlich eine recht genaue Zeit ergibt. Diese direkte digitale Synthese (kurz DDS) ist ein Verfahren in der digitalen Signalverarbeitung zur Erzeugung periodischer, bandbegrenzter Signale mit praktisch beliebig feiner Frequenzauflösung.

```
sp@ depth 1- cells dump ↩
```

Interrupt request ⇒ interrupt flag set
pc → interrupt vector → ISR . . . →forth-word → ISR → `rti`

Abbildung 1: Interrupt–Schema

`MAIN`–Wort an, welches High–Level–Forth ausführt. Von da kommt der Prozess zurück in die ISR. So kann man mit dem Forthsystem drum herum fast ungestört weiter arbeiten, um z.B. Temperaturwerte auszulesen, oder ein Log auszulesen. Es kann auch Neues compiliert werden, während der Schrittmacher im Hintergrund abläuft. So ist es sehr komfortabel möglich, direkt im embedded controller zu entwickeln, derweil die Uhr bereits läuft und Anwendungen ausführt. Und um diese wunderbare interaktive Funktionalität zu haben, braucht man gar nichts zu tun! Denn sie ist in Forth ja schon eingebaut.

Anmerkung

Die Quelle `tim2ISR.asm` ist in AVR–Assembler verfasst, damit die Uhr gleich mit dem amforth geladen wird. Im Kern des amforth ist dazu eine Ergänzung nötig — das deferred word `MAIN` muss im EEprom angelegt werden. Dafür fügen wir in der Datei `amforth.asm` Folgendes ein vor dem abschließenden `edp`:

```
...
; ederferd main word of application:
;  ' <name> is mainword
EE_MAINWORD:
    .dw XT_XLED1
.edp:
```

Was dann vom `MAIN`–Wort ausgeführt wird, kann später bestimmt werden, hier ist bereits `xled1` zugewiesen worden, was eine LED toggelt. Außerdem wurden sämtliche Interrupt–Routinen des amforth herausgenommen, und für `KEY` und `EMIT` die gepollten Fassungen verwendet. So gibt es keine undurchsichtigen Störungen und das Ganze lief bisher stabil. Mit drei solcher atmeag168 im LAN wird eine Solarkollektor–Anlage betrieben.

Links

http://embedded.com/IntroductiontoReentrancy
http://www.atmel.com/products/AVR/
http://de.wikipedia.org/wiki/Direct_Digital_Synthesis
http://en.wikipedia.org/wiki/Direct_digital_synthesis(phaseaccumulator)
http://en.wikipedia.org/wiki/Numerically-controlled_oscillator

Listing: tim2iSR.asm

```
1   ; Timer Interrupt Service Routine (tim2ISR)
2   ; Die tim2ISR arbeitet periodisch. Dessen Frequenz wird geteilt im
3   ; Phasen-Akkumulator, der taktet die Uhr und ein Forthwort.
4
5
6   ; Aufbau der globalen Variablen im Ram:
7      .set heap = heap + CELLSIZE
8
9   ; -- Platz fuer die Uhr bereit stellen,
10  ;    zugreifen per Basisadresse "time" :
11  ; ( -- adr )
12  VE_TIME:
13      .dw $ff04
14   .db "time"
15      .dw VE_HEAD
16      .set VE_HEAD = VE_TIME
17  XT_TIME:
18      .dw PFA_DOVARIABLE
19  PFA_TIME:
```

[3] Hier werden nur die Zeiger gerettet und dann wiederhergestellt, die auch in der ISR benutzt werden. Da die virtuelle Maschine des Forth genau bekannt ist, lassen sich so sehr sparsame und daher schnelle Routinen erstellen. Das ist bei Compilern, deren Bibliotheksaufrufe unbekannte Register und Zeiger benutzen, so nicht möglich.

```
20      .dw heap
21  .set stunde=heap
22      .set heap = heap + 1
23  .set minute=heap
24      .set heap = heap + 1
25  .set sekunde=heap
26      .set heap = heap + 1
27  ; um eine genaue Sekunde zu bekommen, wird ein Phasen-Akkumulator benutzt:
28  .set phaseaccu=heap
29      .set heap = heap + CELLSIZE
30
31  ; -- Platz fuer das Datum bereit stellen,
32  ;    zugreifen per Basisadresse "date" :
33  ; ( -- adr )
34  VE_DATE:
35      .dw $ff04
36   .db "date"
37      .dw VE_HEAD
38      .set VE_HEAD = VE_DATE
39  XT_DATE:
40      .dw PFA_DOVARIABLE
41  PFA_DATE:
42      .dw heap
43  .set jahr=heap
44      .set heap = heap + CELLSIZE
45  .set monat=heap
46      .set heap = heap + 1
47  .set tag=heap
48      .set heap = heap + 1
49
50
51
52  ; timer2_ofv Sprungvektor in die Interupt-Tabelle eintragen:
53  .equ bevortim2jmp = pc
54  .org $0012
55      jmp tim2_ovf ; Timer/Counter2 Overflow ISR
56  .org bevortim2jmp
57
58
59
60  ; Timer2 initialisieren, starten und Interrupt einschalten:
61  ; ( -- )
62  VE_TIM2INIT:
63      .dw $ff08
64   .db "tim2init"
65      .dw VE_HEAD
66      .set VE_HEAD = VE_TIM2INIT
67  XT_TIM2INIT:
68      .dw DO_COLON
69  PFA_TIM2INIT:
70      .dw XT_INTOFF, XT_DROP ; drop SREG left by -int.
71  ; Timer-Register setzen:
72      .dw XT_DOLITERAL,$0
73      .dw XT_DOLITERAL,TCCR2A
74      .dw XT_CSTORE
75      .dw XT_DOLITERAL,7 ; prescaler 1024 - langsamer Takt ist angestrebt.
76      .dw XT_DOLITERAL,TCCR2B
77      .dw XT_CSTORE
78      .dw XT_DOLITERAL,$1
79      .dw XT_DOLITERAL,TIMSK2
```

```
80        .dw XT_CSTORE
81        .dw XT_INTON
82        .dw XT_EXIT
83
84
85
86    ; Sprungvektor aufbauen fuer das main word der application:
87    ; ( -- n*y ) System Value
88    ; R( -- )
89    ; deferred action
90    VE_MAINWORD:
91        .dw $ff08
92        .db "mainword"
93        .dw VE_HEAD
94        .set VE_HEAD = VE_MAINWORD
95    XT_MAINWORD:
96        .dw PFA_DODEFER
97    PFA_MAINWORD:
98        .dw EE_MAINWORD
99        .dw XT_EDEFERFETCH
100       .dw XT_EDEFERSTORE
101
102
103
104   ; tim2ISR:
105   ; -- Minimum an Status retten:
106   tim2_ovf:
107      nop
108      push XH
109      push XL
110      lds  XL,0x5F   ; Statusregister
111      push XL
112   ; -- Uhr:
113   ;    phase accumulator liefert periodisch ein carry-bit-set = uhrtakt.
114   ;    Mit diesem Uhrentakt wird eine recht genaue Sekunde gemacht.
115   ; phaseaccu test1:
116   ; Mo. 07.12.09 07:28 - Di 08.12.09 20:24 => Laufzeit: 132960 sec
117   ; mit: accuteiler = (-1431*3/2) = -2.146,5
118   ; ergab: .time 20 22 27  ok
119   ; ging nach um 93 sec in der Zeit.
120   ; accuteiler = phaseaccu * prescaler * tim2ov / f_cpu = -2.147,483648
121   .set accuteiler = (-2147)
122
123   tim2uhr:
124        lds  XL,phaseaccu
125        subi XL,low(accuteiler)
126        sts  phaseaccu,XL
127        lds  XL,phaseaccu+1
128        sbci XL,high(accuteiler)
129        sts  phaseaccu+1,XL
130   ; tim2ISR beenden, falls die Phase noch nicht um ist:
131      brcs tim2_llret ; <--- ganz rausspringen
132        lds  XL,sekunde
133        inc  XL
134        cpi  XL,60
135      brlo tim2sekunde ; branch if lower
136        lds  XL,minute
137        inc  XL
138        cpi  XL,60
139      brlo tim2minute
```

```
140       lds  XL,stunde
141       inc  XL
142       cpi  XL,24
143     brlo tim2stunde
144       ldi  XL,0
145   tim2stunde:
146       sts  stunde,XL
147       ldi  XL,0
148   tim2minute:
149       sts  minute,XL
150       ldi  XL,0
151   tim2sekunde:
152       sts  sekunde,XL
153   ;   jmp  tim2_llret ; fuer Testzwecke
154
155
156   ; Im Sekundentakt wird nun im Hintergrund Forth ausgefuehrt.
157   ; Darauf achten, dass nur reentry-feste Forthworte benutzt werden.
158
159   ; Hier wird der Programmfluss an high level forth uebergeben:
160   ; -- Status retten:
161     push ZL
162     push ZH
163     push R24        ; wl
164     push R25        ; wh
165     push R22        ; tosl = r22
166     push R23        ; tosh = r23
167     push YL
168     push YH
169     push temp0
170     push temp1
171   ; -- IP setzen:
172     ldi  XL,low(pfa_forthISR)
173     ldi  XH,high(pfa_forthISR)
174     jmp  DO_NEXT
175   ; -- Zeiger auf das colon word, das als nächstes ausgefuehrt wird:
176   pfa_forthISR:
177     .dw  XT_MAINWORD
178   ; -- Programmfluss in die tim2ISR zurückleiten:
179     .dw  PC+1  ; next IP
180     .dw  PC+1  ; next cfa
181   ; -- Benutzte Register wiederherstellen:
182     pop temp1
183     pop temp0
184     pop YH
185     pop YL
186     pop R23
187     pop R22
188     pop R25
189     pop R24
190     pop ZH
191     pop ZL
192   tim2_llret: ; <--- Aussprung, X und Statusregister wiederherstellen.
193     pop XL
194     sts 0x5F,XL
195     pop XL
196     pop XH
197     reti
198
199   ; finis
```

Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 78, Februar 2010

SPI spy : de software II — Ron Minke

Im vorigen Artikel wurde die Grundlage für den SPI–spy besprochen. Im vorliegenden Artikel kommt das Monitor–Programm (zur Mitverfolgung des Programm–Ablaufs) an die Reihe. Besprochen wird: Was muss es tun? (Memory–Dump vom Slave) — Das Protokoll (auf Interrupt–Basis hereinholen) — Kontrolle über Kontrolle (Verzögerungen) — Der Rest vom Protokoll ... — Ohne Weiteres auch gleich mehr (mindestens 16 Bytes hintereinander) — Andere Möglichkeiten (auch Bytes versenden) — Auch vom Flash holen? (Ja, auch das. AVR–Flash ist jedoch in Words organisiert, das Hereinholen in Bytes) — Aber Achtung: Schreiben ins Flash geht mit diesem Protokoll nicht. Hierzu ist ein AVR–Prozessor mit SPM–Befehl nötig. Slave–EEPROMs sind zu träge. Werden hier nicht berücksichtigt.

Ober hulpje — Frans van der Markt

Ein paar Zeilen für den Forth–Einsteiger: Eine elektronische Hosentaschen–Addiermaschine für den Ober. Das heißt, *nur* das Forth–Programm dazu. Das forthfähige iPod oder iPad wird stillschweigend vorausgesetzt. (Der Ober braucht allerdings kein Forth zu beherrschen!)

Das Vijgeblaadje und die HCC–Forth–gg

Das Vijgeblaadje erscheint um den Ersten eines jeden geraden Monats herum. Neues von der Forth–gebruikersgroep erfährt man per `http://www.forth.hcc.nl/nieuws`. Die Mitgliederversammlung (Algemene Leden Vergadering) 2010 findet am 10. April ab 13.00 Uhr statt.

Das FlashForth von Mikael Nordman

Karsten Roederer

Nach einigen Projekten mit den PICs von Microchip, bevorzugt damals die 16F(C)XXX Serie, keimte schon bald der Wunsch nach einem interaktiven Forth auf. Dieses macht aber nur Sinn in der 18FXXXX Serie, da bei diesen Typen das *Pagen* der Speicherbereiche ein Fremdwort ist. Die PICs sind die zur Zeit verbreitetsten Prozessoren im Automotive–Bereich. Auf der letzten Forth-Tagung in Neuenkirchen entblößte ich mich mit dem Wunsch nach einem solchem Forth, was von Adolf brummig sogleich mit einem: „gibt's doch schon alles" kommentiert wurde. Bei den jetzt anstehenden Recherchen stieß ich zunächst auf das Pic18Forth (leider nicht interaktiv) und dann auf das FlashForth von Mikael Nordman aus Finnland. Das ist meines Erachtens sehr kompetent entwickelt worden.

Feature

Das FlashForth schien all das eingebaut zu haben, was mein Herz begehrt. Die Worte sind im Großen und Ganzen ANS'94–Standard. Der PICeigene Stack wird als Returnstack verwendet. Der Compiler schreibt entweder ins flash, eeprom oder ram, je nachdem, was man im Source geschaltet hat. Das betrifft die Worte: create allot variable , c, value und defer. Casesensitive. Die Interruptroutinen können irgendwo stehen und unterscheiden sich vom restlichen Code durch ein schlichtes ;i am Ende der Definition. Interpretiert und kompiliert wird über die serielle Schnittstelle. Leider, und jetzt kommt der Wermutstropfen, gesteuert über XON und XOFF. Nun weiß aber ein jeder etwas über den FIFO in seinem Rechner zu berichten. So ein Handshake verträgt sich nicht mit einem solchen doch wünschenswerten Puffer in der „Timesharingwelt". Wenn man diesen disabled, würde die Kommunikation zwar klappen mit einem kleinen Restrisiko. Ich habe mir erlaubt, XON/XOFF durch einen RTS–CTS–Handshake zu ersetzen. Was im Assembler auf der PIC–Seite noch gut zu machen war, gestaltete sich mit Win32Forth leider erst einmal mangelhaft. Es funktionierte nur, wenn ich vorher ein anderes Programm gestartet hatte, welches ein solches Feature richtig umgesetzt hatte. Dank Ullis Hilfe klappt das jetzt auch mit den Win32Forth–eigenen Windowcalls. Ein weiteres Problem machte dann meine UART-Hardware auf meinem Rechner. Wenn der PIC mit CTS wackelte, stoppte der PC zwar die Übertragung beim nächsten Byte; ließ aber, nachdem CTS wieder clear war, das folgende Byte fallen – nach Recherchen ein wohlbekannter National bug im FIFO. Die mir zur Verfügung stehenden USB-RS232-Adapter haben dieses *Feature* zum Glück nicht eingebaut und so arbeite ich jetzt, auch wenn der Rechner eine serielle Schnittstelle besitzt, mit solchen Adaptern.

FlashForth Backmischung

Das Übersetzen des Assemblerprogramms funktionierte bei mir mit dem MPASM–Standardassembler nicht. Mit gputils eingebunden in Microchips MPASM funktionierte dann build mit success.

Die Programmierung des PIC übernimmt der upgegradete PICStart+. Er kann auch über einen Adapter das ICSP (in circiut serial programming) ausführen. Auf der Microchipseite ist die Programmierung spezifiziert. Es gibt da aber auch andere Low–Cost–Lösungen. TAIT über die parallele Schnittstelle oder warum nicht auch mal eine Lösung in Forth über die serielle Schnittstelle unter der Verwendung der Handshakeleitungen. Für die verschiedenen Baudraten bei den verschiedenen Clocks, mit denen der Prozessor betrieben werden soll, haben mir die Macros für das Einstellen der zugehörigen Register nicht sehr geholfen. Es führt nichts drum herum, das Datenblatt zu diesem Thema zu wälzen und dem Macro den besten krummen Baudwert zum Fraß vorzuwerfen, der zu einem Standard–Baudwert mit dem geringsten Fehler führt. Das ist gerade bei z.B. 115200 Baud mühselig auszuprobieren. Ein gutes Beispiel dafür, dass Macros kontraproduktiv sind in neuen Designs. Gilt es doch einzukreisen, ob etwas mit der Hard- oder Software des Prozessors oder zwischen den Ohren nicht stimmt, wenn das gerade Programmierte nicht nach Wunsch ausgeführt wird.

Arbeiten mit dem PIC

Nachdem das Produkt mit dem Emulator seinen Feinschliff bekommen hatte, konnte ich denselbigen in die Ecke stellen und die Programmentwicklung direkt auf dem PIC weiter betreiben. Schmerzlich vermissen tue ich jetzt nur noch einen Singlestepper. Sehr bewährt hat sich, vor der Programmausführung schon mal ein paar Werte auf den Stack zu legen und nachher nachzuschauen, ob noch alle Elemente da sind. Stackunderflow wird prompt durch einen reset des Watchdogs quittiert. Mit cold holt man sich den letzten *gültigen* Zustand der Maschine zurück und alles frisch Programmierte ist vergessen.

Die I/O-Pins RB6, RB7 und MCLR werden für die ICSP verwendet. Deshalb habe ich nach einigem Hin und Her RB6 als RTS und RB7 als CTS ausgeguckt. In meinen Schaltungsdesigns berücksichtige ich durch eine spezielle Beschaltung ein einigermaßen entkoppeltes Dasein beider Features. Also gleichzeitig die ICSP und die Nutzung als Handshakeleitungen. Verschiedene Beispielprogramme können auch hier downgeloaded werden. Sie sind eine sehr gute Grundlage, um sich mit der speziellen PIC–Programmierung (inlining von Code etc.) anzufreunden.

Als Terminalprogramm stelle ich gern eine Win32Forth–Version zur Verfügung. Hier wird mit F12 ein Fenster zur Verfügung gestellt, wo eine Source zum uploaden ausgesucht werden kann. Das, was der PIC bekommt,

wird nach einiger Zeit geechoed im Terminalfenster dargestellt, wenn Windows den FIFO des UART von seinem Inhalt befreit. Hier sollte man dann noch mal schauen, ob der Compiler nicht gemeckert hat, was er nur durch ein schlichtes Fragezeichen im Code zu tun pflegt.

Literatur

[1] TP016 (91016b.pdf) auf `http://www.Mircochip.com`

[2] Elektor (welche Ausgabe ist leider auf der Kopie nicht mehr zu erkennen) PIC18Flash Entwicklungssystem von Peter Moreton

[3] FF can be downloaded from `http://www.sourceforge.net/projects/flashforth`

Wortschatz des FlashForth

```
Dictionary of FlashForth 3.3 on PIC18F

!           ( x addr -- )
            Store x to addr

!p          ( addr -- )
            Store addr to p(ointer) register

!p>r        ( addr -- )   Compile Only
            Push contents of p to return stack and
            store addr to p

#           ( u1 -- u2 )  Compile Only
            Convert 1 digit to formatted numeric string

#>          ( u1 -- c-addr u )    Compile Only
            Leave address and count of formatted
            numeric string

#s          ( u1 -- u2 )  Compile Only
            Convert remaining digits to formatted
            numeric output

'           ( -- xt )
            Parse word and find it in dictionary

'source     ( -- a-addr )     User Variable
            Current input source

(           ( -- )
            Skip input on the same line until ) is
            encountered

*           (u1/n1 u2/n2 -- u3/n3
            Signed and unsigned 16*16->16 bit
            multiplikation

+           ( n1 n2 -- n3 )
            Add n1 to n2

+!          ( n addr -- )
            Add n to cell at addr.

,           ( x -- )
            Append x to the current data section

-           ( n1 n2 -- n3 )
            Subtract n2 from n1

.           ( n -- )
            Display n signed according to base

.s          ( -- )
            Display the stack contents

.st         ( -- )
            Emit status string for base, current data
            section, and display the stack contents.

/           ( n1 n2 -- n3 )
            16/16->16 bit division

/string     ( addr u n -- addr+n u-n )
            Trim string

0<          ( n -- flag )
            Leave true flag if n is less than zero

0=          ( x -- flag )
            Leave true flag if x is zero

1           ( -- 1 )
            Leave one

1+          ( n -- n1 )
            Add one to n

1-          ( n -- n1 )
            Subtract 1 from n

2*          ( u1 -- u2 )
            Shift u1 left one bit

2+          ( n -- n1 )
            Add two to n

2-          ( n -- n1 )
            Subtract 2 from n

2/          (n1 -- n2 )
            Shift n1 right one bit.

2@          ( a-addr -- x1 x2 )
            Fetch two cells

2!          ( x1 x2 a-addr -- )
            Store two cells

2drop       ( x1 x2 -- )
            Drop two cells

2dup        ( x1 x2  -- x1 x2 x1 x2 )
            Duplicate two top cells

:           ( "name" -- )
            Begin a colon definition

:noname     ( -- addr )
            Define headerless forth code
```

```
;          ( -- )        Compile Only
           End a colon definition

;i         ( -- )        Compile Only
           End a interrupt word

<          ( n1 n2 -- flag )
           Leave true flag if n1 is less than n2

<#         ( -- )        Compile Only
           Begin numeric conversion

<>         ( x1 x2 -- flag )
           Leave true flag if x1 and x2 are not equal

=          ( x1 x2 -- flag )
           Leave true flag if x1 and x2 are equal

>          ( n1 n2 -- flag )
           Leave true flag if n1 is grater than n2

>body      ( xt -- a-addr )
           Leave the parameter field address of
           a created word

>digit     ( n -- c )
           Convert n to ascii character value

>in        ( -- a-addr )      User Variable
           Holds offset into tib

>number    ( n1 addr1 u1 -- n2 addr2 u2 )
           Convert string to number

>r         ( x -- ) ( R: -- x )  Compile Only
           Push x from the parameter stack to the
           return stack

?abort     ( flag c-addr u -- )
           Print message and abort if flag is true

?abort?    ( flag -- )
           If flag is true output ? and abort

?negate    ( n1 n2 -- n3 )
           Negate n1 if n2 is negative

@          ( addr -- x )
           Fetch x from addr

@+         ( addr1 -- addr2 x )
           Fetch cell from addr1 and increment addr1
           by a cell

@p         ( -- addr )
           Fetch the p register to the stack

@ex        ( addr -- )
           Fetch vector from addr and execute.

[          ( -- )
           Enter interpreter state

[']        ( "name" -- )         Compile Only
           Compile xt of name as a literal

[char]     ( "char" -- )         Compile Only
           Compile inline ascii character

\          ( -- )
           Skip rest of line

]          ( -- )
           Enter compilation state

abort      ( -- )
           Reset stack pointer and execute quit

abort"     ( "string" -- )     Compile Only
           Compile inline string and postpone abort?

abs        ( n -- n1 )
           Leave absolute value of n

accept     ( c-addr +n -- +n' )
           Get line from terminal

again      ( addr -- )                  Compile Only
           begin ... again

align      ( -- )
           Align the current data section dictionary
           pointer to cell boundary

aligned    ( addr -- a-addr )
           Align addr to a cell boundary.

allot      ( n -- )
           Adjust the current data section dictionary
           pointer

and        ( x1 x2 -- x3 )
           Bitwise and of x1 and x2

base       ( a-addr )     User Variable
           Numeric conversion base

begin      ( -- a-addr )                Compile Only
           begin ... again
           begin ... until
           begin ... while ... repeat

bin        ( -- )
           Set base to binary

bl         ( -- c )
           Ascii space

c!         ( c addr -- )
           Store c to addr

c@         ( addr -- c )
           Fetch c from addr

c@+        ( addr1 -- addr2 c )
           Fetch char from addr1 and increment addr1

c,         ( c -- )
           Append c to the current data section

cell       ( -- n )
           Leave the size of one cell in characters.

cell+      ( addr1 -- addr2 )
           Add cell size to addr1

cells      ( x1 -- x2 )
           Convert cells to address units.

char       ( "char" -- n )
           Parse a char and leave ascii value on stack

char+      ( c-addr1 -- c-addr2 )
           Add one to c-addr1
```

```
chars      ( x1 -- x2 )
           Convert characters to address units

cf,        ( xt -- )
           Compile xt into the flash dictionary.

cfa>nfa    ( addr1 -- addr2 )
           Convert cfa to nfa

cmove      ( addr1 addr2 u -- )
           Move u chars from addr1 to addr2

cold       ( -- )
           Make a cold start. Reset all dictionary
           pointers.

con        ( x "name" -- )
           Create a constant in rom as inline code

constant   ( x "name" -- )
           Create a constant in rom with docreate
           as runtime

cr         ( -- )
           Emit CR LF

create     ( "name" -- )
           Create a word definition and store the
           current data section pointer.

cse        ( -- addr )          ram variable
           Variable holding the current data section
           value

cwd        ( -- )
           Clear the WatchDog counter.

decimal    ( -- )
           Set numeric base to decimal 10.

defer      ( "name -- )
           Define a deferred execution vector

di         ( -- )
           Disable interrupts

digit?     ( c -- n flag )
           Convert char to a digit according to base

does>      ( -- )                    Compile Only
           Define the runtime action of a created word.

dp         ( -- addr ) Eeprom variable mirrored in ram
           Leave the address of the current data
           section dictionary pointer

drop       ( x1 -- )
           Drop top of stack

dump       ( addr u -- )
           Display a memory dump

dup        ( x -- x x )
           Duplicate top of stack

ei         ( -- )
           Enable interrupts

end        ( task-addr -- )
           Remove a task from the task list.

eeprom     ( -- )
           Set data section context to eeprom

else       ( addr1 -- addr2 )       Compile Only
           if ... else ... then

emit       ( c -- )
           Emit c to the serial port FIFO. FIFO is
           46 chars. Executes pause.

evaluate   ( c-addr n -- )
           Evaluate buffer

execute    ( addr -- )
           Execute word at addr

exit       ( -- )
           Exit from a word.

false      ( -- 0 )

flash      ( -- )
           Set data section context to flash

fill       ( c-addr u c -- )
           Fill u bytes with c staring at c-addr

find       ( c-addr -- c-addr 0/1/-1 )
           Find a word in dictionary
           Leave 1 if immediate, -1 if normal,
           0 if not found

for        ( u -- )              Compile Only
           Loop u times. for ... next

forget     ( "name -- )
           Forget name

here       ( -- addr )
           Leave the current data section dictionary
           pointer

hex        ( -- )
           Set numeric base to hexadecimal

hold       ( c -- )      Compile Only
           Append char to formatted numeric string

hp         ( -- a-addr )            User Variable
           Hold pointer for formatted numeric output

i,         ( x -- )
           Append x to the flash data section.

ic,        ( c -- )
           Append c to the flash data section.

ihere      ( -- addr )
           Start of free flash

if         ( -- a-addr )        Compile Only
           if ... else ... then

iflush     ( -- )
           Flush the flash write buffer

immed?     ( addr -- n )
           Leave a nonzero value if addr contains a
           immediate flag

immediate  ( -- )
           Mark latest definition as immediate
```

```
in?         ( nfa -- flag )
            Leave true flag if nfa has inline bit set

inline      ( "name" -- )
            Inline the following word.

inlined     ( -- )
            Mark the latest compiled word as inlined.

interpret   ( c-addr u - )
            Interpret the buffer

invert      ( x1 -- x2 )
            Ones complement of x1

irq         ( -- a-addr)          Ram value
            Interrupt vector. Cleared at warm start.

is          ( x "name" -- )
            Set the value a deferred word

key         ( -- c )
            Get a character from the serial port FIFO.
            Execute pause until a character is
            available

key?        ( -- flag )
            Leave true if character is waiting in the
            serial port FIFO

khz         ( -- u )
            Leave the cpu clock in KHz.

latest      ( -- a-addr )
            Variable holding the address of the latest
            defined word

leave       ( -- )              COMPILE ONLY
            Leave a for/next loop when next is
            encountered.
            Sets top of return stack to zero

literal     ( x -- )
            Compile a literal into the dictionary

lshift      ( x1 u -- x2
            Shift x1 u bits to the left

m+          ( d1 n -- d2 )
            Add double number d1 to n

marker      ( "name" -- )
            Mark a dictionary state

max         ( n1 n2 -- n3 )
            Leave max of n1 and n2

mclr        ( mask ram-addr -- )
            AND the contents of caddr with the
            complement of mask

min         ( n1 n2 -- n3 )
            Leave min of n1 and n2

ms          ( +n -- )
            Pause for +n milliseconds

mset        ( mask ram-caddr -- )
            OR the contents of caddr with mask.

mtst        ( mask addr -- x )
            AND the contents of caddr with mask

n=          ( c-addr nfa u -- flag )
            Compare strings in ram(c-addr) and
            flash(nfa)
            flag is true if strings match. u<32.

negate      ( n -- -n )
            negate n

next        ( bra-addr bc-addr -- )      Compile Only
            for ... next

nfa>lfa     ( addr1 -- addr2 )
            Convert nfa to lfa

nip         ( x1 x2 -- x2 )
            Remove x1 from the stack

number?     ( c-addr -- n/c-addr flag )
            Convert string to number
            # is decimal prefix
            $ is hexadecimal prefix
            % is binary prefix

operator    ( -- addr )
            Leave the address of the operator task

or          ( x1 x2 -- x3 )
            Or bitwise x1 with x2

over        ( x1 x2 -- x1 x2 x1 )
            Copy x1 to top of stack

p+          ( -- )
            Increment p register by one

p++         ( n -- )
            Add n to the p register

p!          ( x -- )
            Store x to the location pointed by the p
            register

pc!         ( c -- )
            Store c to the location pointed by the p
            register

p@          ( -- x )
            Fetch the cell pointed by the p register

pc@         ( -- c )
            Fetch the char pointed by the p register

pad         ( -- a-addr )
            Pad is ram here $20 +

pause       ( -- )
            Switch task

place       ( addr1 u addr2 -- )
            Place string from addr1 to addr2 as a
            counted string

postpone    ( "name" -- )          Compile Only
            Postpone action of immediate word

prompt      ( -- a-addr )          Eeprom defer
            Deferred execution vector for the info
            displayed by quit. Default is .ST .
```

```
quit        ( -- )
            Interpret from keyboard

r>          ( -- x ) ( R: x -- )   Compile Only
            Pop x from the return stack to the
            parameter stack

r>p         ( -- )                 Compile Only
            Pop from return stack to p register

r@          ( -- x ) ( R: x -- x ) Compile Only
            Copy x from the return stack to the
            parameter stack

ram         ( -- )
            Set data section context to ram

rcnt        ( -- a-addr )    User Variable
            Number of saved return stack cells

rdrop       ( -- )             Compile Only
            Remove top elemnt from return stack

repeat      ( addr2 addr1 -- )    Compile Only
            begin ... while ... repeat

rhere       ( -- addr )
            Start of free ram

rot         ( x1 x2 x3 -- x2 x3 x1 )
            Rotate three top stack items

rsave       ( -- a-addr )  User variable
            Return stack save area

rshift      ( x1 u -- x2 )
            Shift x1 u bits to the right

run         ( task-addr -- )
            Link the task to the task list. The task
            starts running immediately.

s0          ( -- a-addr )
            Variable for start of parameter stack


scan        ( c-addr u c -- c-addr' u'
            Scan string until c is found.
            c-addr must point to ram. u<255

sign        ( n -- )
            Append minus sign to formatted numeric
            output

sign?       ( addr1 n1 -- addr2 n2 flag )
            Get optional minus sign

single      ( -- )
            End all tasks except the operator task.
            Removes all tasks from the task list
            except the operator task.

skip        ( c-addr u c -- c-addr' u' )
            Skip string until c not encountered.
            c-addr must point to ram. u<255

sp@         ( -- addr )
            Leave parameter stack pointer

sp!         ( addr -- )
            Set the parameter stack pointer to addr

s"          ( "text" -- )          Compile Only
            Compile string into flash

."          ( "text" -- )          Compile Only
            Compile string to print into flash

source      ( -- c-addr n )
            Current input buffer

space       ( -- )
            Emit one space character

spaces      ( n -- )
            Emit n space characters

ssave       ( -- a-addr )        User Variable
            Saved return stack pointer

state       ( -- a-addr)         User Variable
            Comilation state

swap        ( x1 x2 -- x2 x1 )
            Swap two top stack items

task        ( tibsize stacksize rstacksize addsize -- )
            Define a task

tinit       ( taskloop-addr task-addr -- )
            Initialise the user area and link it to a
            task loop

then        ( addr -- )          Compile Only
            if ... else ... then

tib         ( -- addr )     User variable
            Leave address of cell containing the
            address of the terminal input buffer

tibsize     ( -- n )
            Size of terminal input buffer

ticks       ( -- u )
            System ticks

to          ( x "name" -- )
            Store x into value "name".

true        ( -- -1 )

tuck        ( x1 x2 -- x2 x1 x2 )
            Insert x2 below x1 in the stack

turnkey     ( -- a-addr ) Eeprom value mirrored in ram
            Vector for user startup word

type        ( c-addr u -- )
            Type line to terminal. u < $100

u*/mod      ( u1 u2 u3 -- u4(remainder) u5(quotient) )
            Unsigned u1*u2/u3 with 32 bit intermediate
            result

u.          ( u -- )
            Display u unsigned according to numeric
            base

u.r         ( u +n -- )
            Display u in field of width n. 0<n<256

u/          ( u1 u2 -- u3 )
            Unsigned 16/16->16 bit division
```

```
u/mod      ( u1 u2 -- u3(remainder) u4(quotient)
          Unsigned 16/16->16 bit division

u<        ( u1 u2 -- flag )
          Leave true flag if u1 is less than u2

u>        ( u1 u2 -- flag )
          Leave true flag if u1 is greater than u2

ulink     ( -- a-addr)
          Link to next task

um*       ( u1 u2 -- ud )
          Unsigned 16x16 -> 32 bit multiply

um/mod    ( ud u1 -- u2(remainder) u3(quotient)
          unsigned 32/16 -> 16 bit division

umax      ( u1 u2 -- u )
          Leave the unsigned larger of u1 and u2.

umin      ( u1 u2 -- u )
          Leave the unsigned smaller of u1 and u2.

until     ( flag -- )    Compile only
          begin..until

up        ( -- a-addr )
          Variable holding the user pointer

user      ( n "name" -- )
          Define a user variable at offset n

value     ( x "name" -- )
          Define a value

variable  ( "name" -- )
          Create a variable in the current data
          section

warm      ( -- )
          Make a warm start

while     (addr1 -- addr2 addr1 )       Compile Only
          begin ... while ... repeat

within    ( x xl xh -- flag )
          Leave true if  xl <= x < xh

word      ( c -- c-addr )
          Copy a word delimited by c to c-addr

words     ( -- )
          List words

xor       ( x1 x2 -- x3 )
          Xor bitwise x1 with x2.
```

ASSEMBLER words

The following definitions are in the core dictionary. The rest is in asm.fth

```
as0       ( opcode "name" -- ) ( -- )
          Create a zero parameter assembler word

as1       ( opcode "name" -- ) ( k -- )
          Create a one parameter assembler word

as2       ( opcode "name" -- ) ( f a -- )
          Create a two parameter assembler word

as3       ( opcode "name" -- ) ( f d/b a -- )
          Create a three parameter assembler word

br?       ( rel-addr limit -- clipped-rel-addr )
          Clip a relative address and check for
          overflow

br1       ( opcode "name" -- ) (rel-addr -- )
          Create conditional branch assembler word

br2       ( opcode "name" -- ) ( rel-addr -- )
          Create a relative branch assembler word

br3       ( opcode "name" -- ) ( addr -- )
          Create a absolute address branch assembler
          word

call,     ( addr -- )
goto,     ( addr -- )
rcall,    ( rel-addr -- )
bra,      ( rel-addr -- )
z,        ( -- cc )
nz,       ( -- cc )
not,      ( cc -- not-cc )
if,       ( cc -- here )
else,     ( back-addr -- here )
then,     ( back-addr -- )
begin,    ( -- here )
again,    ( back-addr -- )
until,    ( back-addr cc -- )
```

Bootmanager und FAT–Reparatur: Sechster Fort(h)schritt (Diskbootcopy)

Fred Behringer

Im vorliegenden Artikel wird mit den Hilfsmitteln aus Teil 4 dieser Serie ein Forth–Wort, diskbootcopy, entwickelt, das es gestattet, den Bootsektor einer Diskette auf eine andere Diskette per *Knopfdruck* (als Aufruf eben dieses Forth–Wortes oder von der DOS–Ebene aus über eine Batchdatei) zu übertragen. diskbootcopy kann mit Disketten beliebiger (beliebig abgeänderter!) Struktur umgehen und arbeitet auch da, wo RawWrite und Diskcopy versagen.

Was wurde schon gebracht?

Der Artikel stützt sich hauptsächlich auf Teil 4 der Serie (VD–2/2009). Die drei benötigten Forth–Worte aus früheren Serien–Teilen werden im Listing wiederholt. Das Programm läuft unter allen gängigen DOS–Versionen und in jedem Windows–System, welches nichts gegen simple 16–Bit–DOS–Programme und direkte Festplatten– oder/und Disketten–Zugriffe einzuwenden hat. Dazu gehören DOS 6.2, DOS 7.0 (aus Windows 95), DOS 7.1 (aus Windows 98), DOS 8.0 (aus Windows ME) und FreeDOS.

Ausgangspunkt

In Teil 4 der Serie [FB] habe ich über die Anfertigung und die Verwendung einer *Hilfsdiskette* gesprochen, mit der man den PC booten und dazu veranlassen kann, vor dem endgültigen Booten per Festplatte die Speicher–Endadresse im BIOS–Datenbereich zurückzusetzen und das ROM–BIOS ins freigewordene RAM zu platzieren.

Bootsektor kopieren

Von dieser *Hilfsdiskette* wird einzig und allein der Bootsektor benötigt. Ist dieser nach dem anfänglichen Booten von der Hilfsdiskette abgearbeitet, dann hat die Hilfsdiskette ihre Schuldigkeit schon getan. Der Bootcode erledigt alles Weitere und schaltet den PC insbesondere hinüber zum Booten von der Festplatte. Will man die Hilfsdiskette (z. B. zur Weitergabe an einen Forth–Kollegen) kopieren, dann sollte es nur allzu plausibel erscheinen, die Kopierarbeit dem DOS–Programm diskcopy oder, wenn das nicht geht, dem

Alleskönner RawWrite

zu überlassen. Aber ach! Von wegen RAW–Write–n! Man kann nicht davon ausgehen, dass bei beliebiger Quelldiskette alles roh (Sektor für Sektor) übertragen wird. Ich habe es mit dem RAWWRITE 0.7 von John Newbigin [JN] versucht und war bass erstaunt: Unmittelbar nach dem Einlegen der *Hilfsdiskette* lieferte rawwritewin.exe die Meldung *Error reading disk*. Beim anschließenden Versuch mit DOS–, FreeDOS– und Windows–DOS–Disketten kam diese Meldung zwar erst nach 97% des Lesevorgangs, aber sie kam. Und das bei einem so einfachen Vorgang wie dem Kopieren einer Diskette (bei der es sogar auf nichts weiter als auf den Bootsektor ankommt)! Die bisher entwickelten Forth–Hilfsmittel meiner Serie schaffen es ja auch — ohne RawWrite!

RawWrite geht

allerdings bei einer Hilfsdiskette, die aus einer DOS–formatierten Diskette hervorgegangen ist. (Es geht auch bei anders formatierten Disketten, z.B. bei Tom Oehsers Ein–Disketten–Linux tomsrtbt-1.7 [TO]). Ich hatte aber im Bootsektor der Hilfsdiskette aus Teil 4 alle Bytes außerhalb des Bootcodes ausgenullt (da überflüssig) - und da ging es eben nicht. Im Listing des vorliegenden Artikels biete ich das Forth-Wort

Diskbootcopy

an. Das arbeitet (mit Bedien-Anweisungen auf dem Bildschirm) so wie der externe DOS–Befehl diskcopy, nur eben nur auf das Kopieren des Bootsektors ausgerichtet. Natürlich können damit auch andere Aufgaben erledigt werden, bei welchen es nur auf das Kopieren des Bootsektors ankommt (der Rest der Quelldiskette wird nicht angerührt). Und um das Einknopf–Feeling perfekt zu machen, soll hier noch schnell eine Batch–Datei angegeben werden, die (ähnlich diskcopy) von der DOS–Kommandozeile aus aufgerufen werden kann:

```
@echo off
forth echo off hex include dbc.fth diskbootcopy bye
```

Diese Zeile als Batch–Datei verpacken! Das DOS–Programm forth ist dabei das Turbo–Forth–System (bei mir forth.com). Man konstruiere dasselbe für ZF!

Wirkung der Batch–Datei

Aus der oben stehenden Zeile mit einem Editor eine Text–Datei mit Namen dbc.txt machen und statt der Endung .txt die Endung .bat anfügen. (dbc soll für diskbootcopy stehen.) Nach Aufruf von dbc.bat (kurz dbc) in der DOS–Kommandozeile passiert Folgendes: @echo off ist der übliche DOS–Batch–Befehl. Mit forth wird das Turbo–Forth–System aufgerufen. Alles, was in der obigen Zeile hinter forth kommt, sind Forth–Worte. Sie werden als Forth–Übergabestring Wort für Wort abgearbeitet, bevor sich der Forth–Prompt meldet. Das Forth–Wort bye leitet schließlich vom Forth–System wieder in die DOS–Ebene zurück.

Der Mechanismus dieser Batch–Datei

kann übrigens überall da angewendet werden, wo man nach stereotypem Muster mit einer kurzen Forth–Sequenz einen DOS–Befehl aufbauen möchte, ohne für jeden solchen *konstruierten* DOS–Befehl immer gleich den ganzen Forth–Überbau mitschleppen zu müssen. (Ein einmaliges Zurverfügungstellen des Forth–Systems (bei mir forth.com) im gleichen DOS–Verzeichnis reicht.) Aus BASIC kennt man beispielsweise PEEK und POKE. Nichts ist einfacher, als das in einer solchen Forth–DOS–Batch–Datei nachzuahmen! Aber das ist ein anderes Thema, ein Thema für weitere VD–Vorhaben ...

Der Einfachheit halber beschränke ich mich hier auf das Wesentliche und arbeite in meiner Vorstellung nur mit

3,5"–Disketten

in HD– oder DD–Format. Man beachte aber, dass von *Format* (und Formatierung) gar keine Rede ist und dass es mit dem hier gezeigten Mechanismus auch ohne Weiteres möglich ist, den Bootsektor beispielsweise von einer HD–Diskette auf eine DD–Diskette zu übertragen, oder umgekehrt. — Der Bootsektor wird also im wortwörtlichen Sinne raw–ge–write–t!

Schnellüberprüfung

Ich habe eine DOS–formatierte 3,5"–DD–Diskette genommen, die einige DOS–Programme enthielt (gar nicht so leicht, noch eine in der Schublade zu finden), und den Bootsektor der *Hilfsdiskette* auf eben diese DD–Diskette übertragen (per (getdiskbootsect) und (putdiskbootsect) — oder eben per diskbootcopy). Die Programme der DD–Diskette waren dann, wie nicht anders zu erwarten, nicht mehr ansprechbar. Aber als *Hilfsdiskette* tat die so präparierte DD–Diskette ihren Dienst. Dann habe ich eine weitere DOS–beschriebene DD–Diskette genommen und deren Bootsektor auf die erstere DD–Diskette übertragen. Die dortigen Programme konnten dann wieder so angesprochen werden, als ob nichts gewesen wäre.

Literatur und Internet–Adressen

[FB] Behringer, Fred: Bootmanager und FAT–Reparatur, Teil 1 bis 5. Vierte Dimension, Hefte 3–4/2008 und 1–3/2009.

[JN] Newbigin, John: `http://www.chrysocome.net/rawwrite`

[TO] Oehser, Tom: `http://www.toms.net/rb/`

Listing

```
\ ******************************************************
\ *                                                    *
\ * DBC.FTH ( in der Serie eigentlich BOOTMA-6.FTH ) *
\ *                                                    *
\ * Zutaten fuer Bootmanager und FAT-Reparatur unter *
\ * Turbo-FORTH-83 und ZF                              *
\ *                                                    *
\ * Fred Behringer - Forth-Gesellschaft - 01.03.2010 *
\ *                                                    *
\ ******************************************************

\ ======================================================
\ Bei Arbeiten mit ZF:
\ zf [ret] fload dbc.fth - Endung fth nicht vergessen!
\ attributs off wegnehmen!  attributs in ZF unbekannt.
\ Ansonsten scheint auch unter ZF alles gut zu laufen.
\ ======================================================

attributs off        \ 'off' fuer den Fall,  dass kein ANSI.SYS in der
                     \ CONFIG.SYS ist. Oder dann aber ANSI.COM aus dem
                     \ Internet hereingooglen  und  in das  DOS-System
                     \ (in die AUTOEXEC.BAT) legen!  Beim Arbeiten mit
                     \ ZF auf jeden Fall wegnehmen!
hex

\ Der folgende Sektorpuffer wurde schon in frueheren Teilen dieser
\ Artikelfolge verwendet und wird hier der Vollstaendigkeit halber
\ wiederholt.

```

```
30  210 allot here       \ Platz fuer mindestens 1 Sektor = 512d Bytes
31  here 0f and -        \ sectbuf an Paragraphenanfang
32  200 -                \ Anfang des Sektorpuffers
33  constant sectbuf     \ Liefert Adresse des Sektorpuffers
34
35
36  \ ================================================================
37  \ Vorweg gesagt: Alle Forth- und DOS-Befehle koennen im Vorliegenden
38  \ in beliebiger Mischung  gross oder auch  klein  eingegeben werden.
39  \ ================================================================
40
41  \ Glossar
42  \ -------
43
44  \ sectbuf            : Wiederholung aus VD-Heft 3/2008
45  \ (getdiskbootsect)  : Wiederholung aus VD-Heft 2/2009
46  \ (putdiskbootsect)  : Wiederholung aus VD-Heft 2/2009
47  \ diskbootcopy       : Kopieren  des Bootsektors  einer  beliebigen
48  \                      Quelldiskette in eine beliebige Zieldiskette
49
50  \ Die folgenden Worte (getdiskbootsect) und (putdiskbootsect) werden gleich
51  \ anschliessend, in diskbootcopy, benoetigt. Es sind Wiederholungen aus
52  \ meinem Artikel 'Bootmanager und FAT-Reparatur: Vierte Fort(h)schritte
53  \ (patchbares BIOS im RAM)' aus dem VD-Heft 2/2009. Auf die ausfuehrlichen
54  \ Kommentare, die dort gegeben wurden, darf hier verzichtet werden.
55
56  \ Das folgende Wort (getdiskbootsect) erwartet im Laufwerk a: eine Diskette,
57  \ von welcher es den Bootsektor (einen MBR gibt es bei Disketten nicht!) holt
58  \ und in den Sektorpuffer sectbuf schreibt. Die Struktur der Diskette (DOS
59  \ oder nicht, HD oder DD) kann beliebig sein. Das System kennt alle fuer den
60  \ Vorgang benoetigten Daten aus den BIOS-Setup-Einstellungen. Das hier der
61  \ Einfachheit halber verwendete Laufwerk a: braucht fuer (getdiskbootsect)
62  \ nicht bootbar zu sein. Man kann auf diesem Weg den Bootsektor einer
63  \ 'Hilfsdiskette' der Art von Teil 4 klonen. Die so hergestellte Kopie der
64  \ Hilfsdiskette braucht fuer die BIOS-Verlegung ins RAM nach Teil 4 der
65  \ Serie nicht wieder von Laufwerk a: aus betrieben zu werden. Man muss aber
66  \ dann sicherstellen, dass das Disketten-Laufwerk, von dem aus der PC spaeter
67  \ gebootet werden soll, dafuer auch wirklich geeignet ist und dass man das
68  \ BIOS entsprechend eingestellt hat.
69
70  code (getdiskbootsect) ( -- fl )
71     si push  1 # di mov  3 # si mov  ds push  es pop
72     begin
73       si dec  ah ah xor  dl dl xor  13 int  dx dx xor  1 # cx mov
74       sectbuf # bx mov  201 # ax mov  13 int
75       u>= if  si si xor  di dec  then
76       0 # si cmp 0=
77     until
78     si pop  di push  next end-code
79
80  \ Das folgende Wort (putdiskbootsect) erwartet im Laufwerk a: eine Diskette,
81  \ in deren Bootsektor es den Inhalt des Sektorpuffers sectbuf legt. Vorsicht!
82  \ Das System kuemmert sich nicht darum, ob die Daten in sectbuf z.B. fuer das
83  \ Uebertragen des Bootsektors von Diskette zu Diskette geeignet sind.
84  \ Andererseits kann nicht viel passieren: Die Quell-Diskette nimmt durch das
85  \ Lesen keinen Schaden und die Ziel-Diskette (mit dem geklonten Bootsektor)
86  \ kann sich nachtraeglich hoechstens als unbrauchbar erweisen - und dann muss
87  \ man eben den Vorgang, diesmal richtig angepackt, wiederholen.
88
89  code (putdiskbootsect) ( -- fl )
```

```
90      si push  1 # di mov  3 # si mov  ds push  es pop
91      begin
92        si dec  ah ah xor  dl dl xor  13 int  dx dx xor  1 # cx mov
93        sectbuf # bx mov  301 # ax mov  13 int
94        u>= if  si si xor  di dec  then
95        0 # si cmp 0=
96      until
97      si pop  di push  next end-code
98
99    \ Das folgende Wort diskbootcopy fasst die beiden Worte (getdiskbootsect)
100   \ und (putdiskbootsect) zusammen und fuegt Fehlerabsicherungen und
101   \ Bedienungsanweisungen als Bildschirmausgaben hinzu.
102
103   : diskbootcopy ( -- )
104       cr ." Quelldiskette (beliebiger Struktur) in Laufwerk a: legen"
105       cr ." und Eingabetaste druecken!"
106       begin key d = until
107       cr (getdiskbootsect)
108       if
109         cr ." Lesefehler! Gesamtvorgang wiederholen!  Das heisst also:"
110         cr ." Eingabetaste druecken und wieder dbc.bat [ret] eingeben!
111         cr bye
112       then
113       cr ." Quelldiskette aus dem Laufwerk nehmen,"
114       cr ." Zieldiskette"
115       cr ." (beliebiger Struktur, beliebiger Kapazitaet, "
116          ." formatiert oder nicht)"
117       cr ." in Laufwerk a: legen und Eingabetaste druecken!"
118       begin key d = until
119       cr (putdiskbootsect)
120       if
121         cr ." Schreibfehler!  Gesamtvorgang  wiederholen!  Das heisst:"
122         cr ." Eingabetaste druecken und wieder dbc.bat [ret] eingeben!"
123         cr bye
124       then
125       cr ." Kopiervorgang  erfolgreich  beendet.  Zieldiskette entnehmen!"
126       cr
127       cr ." Etwa vergessen, von der Quell- zur Ziel-Diskette zu wechseln?"
128       cr ." Macht nichts! Die (im Laufwerk vergessene) Quell-Diskette hat "
129       cr ." sich  durch das Wiederbeschreiben  mit dem eigenen Bootsektor"
130       cr ." nicht veraendert. Den Gesamtvorgang dann einfach wiederholen!"
131       cr
132       cr ." Ob Vorgang erledigt oder Wiederholung: Eingabetaste druecken!"
133       begin key d = until
134       cr bye ;
```

Lebenszeichen

Berichte aus der FIG Silicon Valley: *Henry Vinerts*

Dear Fred,

ich danke dir für deine Mitteilung und ich gratuliere euch dafür, dass ihr die neue VD–Ausgabe unter deiner Mithilfe doch noch verhältnismäßig schnell herausgebracht habt.

Da auch du mit Gleitkomma–Zahlen gearbeitet hast, wird es dich sicher interessieren, dass Robert Smith, der Autor des Gleitkomma–Teils in Tom Zimmers F–PC, entdeckt hat, dass ihm ein Fehler/ein Versäumnis unterlaufen ist. Die SVFIG–Mitglieder haben von Bob (Robert) die folgende Nachricht erhalten:

> Ich arbeite mit Tom Zimmers letztem Win32Forth–System. Ich war der ursprüngliche Autor des Gleitkomma–Pakets und als solcher ist es mir peinlich, — nach all den Jahren — gestehen zu müssen, in der Exponentialfunktion einen Fehler entdeckt zu haben. Der Fehler besteht darin, dass die Funktion fexp eine –NAN (–NotaNumber) ausgibt, wenn als Argument f–inf (–infinity) genommen wird. Als Ergebnis sollte dabei aber eine f0.0 (floating point 0) herauskommen. Entsprechend sollte finf fexp das Ergebnis finf liefern, was aber ebenfalls zu einem Fehler führt, nämlich wiederum zu –NAN. Ich bin mit den neuesten Win32forth–Systemen nicht vertraut und weiß also nicht, ob dieser Fehler inzwischen korrigiert wurde. Meine Assembler–Kenntnisse in Bezug auf dieses Thema sind etwas eingerostet. Ich hätte also nichts dagegen, wenn sich jemand an die Korrektur wagen würde. Sollte keine Reaktion kommen, würde ich selbst versuchen, den Fehler zu beseitigen. In der Zwischenzeit werde ich das Problem auf High–Level–Ebene lösen.

Bob fügte später hinzu, dass derselbe Fehler auch die trigonometrischen Funktionen in Win32Forth beeinflusst und dass er um Hilfe bei der Fehlerbeseitigung Ausschau halten werde, da er selbst mit der Assembler–Programmierung nicht mehr so fix bei der Hand ist.

Kevin Appert hat den Vorschlag gemacht, diese Nachricht in `comp.lang.forth` zu *posten*. Über kurz oder lang werden also die regelmäßigen Besucher von `comp.lang.forth` das Gesagte sicher zu Gesicht bekommen.

Soweit ich erfahren habe, wird das SVFIG–Treffen diesen Monat wieder in Stanford abgehalten werden, ich habe mich aber noch nicht dafür entschieden hinzufahren. Ting hat angekündigt, dass er den ganzen Vormittag mit seinem Vortrag ausfüllen wird, und um ehrlich zu sein, ist das meiste von dem, über was er spricht, für mich zu hoch.

With best regards,
Henry

6. Februar 2010:

Dear Fred,

inzwischen (am 3.2.2010) erschien am Schwarzen E–Mail–Brett der SVFIG die folgende Nachricht von Kevin Appert an Bob Smith:

> Gibt es was Neues über das Gleitkomma–Problem in Win32Forth? Ich habe in Erinnerung, dass es da einen Patch für das eine oder andere dieser Probleme gab, ich kann mich aber nicht mehr erinnern, wo ich das gelesen habe.

Ich glaube, dass auch ich über diese Sache eine Nachricht gelesen habe, die betreffende E–Mail habe ich aber leider schon gelöscht. Wenn ich irgendeine Antwort von Bob Smith lese, werde ich euch informieren. Bis dahin wird es wohl das Beste sein, zu warten und zu sehen, ob in `comp.lang.forth` Neues zum Thema Gleitkomma in Win32Forth erscheint.

Gleitkomma–Arithmetik in Forth ist mir zu hoch, als dass ich mich an der Diskussion beteiligen könnte. Andererseits sehe ich, dass Michael (Kalus) im VD–Heft 4/2009 sechs Seiten dazu geschrieben hat, und ich wollte sichergehen, dass er Gelegenheit findet, sich zu Bob Smiths Gedanken über mögliche *unerfreuliche Überraschungen mit Gleitkommazahlen* in Win32Forth (oder eben auch in F–PC) zu äußern.

With best regards,
Henry

Übersetzt von Fred Behringer

Forth + APL? 5!

Bernd Ulmann

Einleitung

Gefragt danach, welche Programmiersprachen die interessantesten Konzepte beinhalten und am ehesten dazu geeignet sind, nicht nur kurze Implementationszeiten, hohe Flexibilität und andere direkt messbare Eigenschaften zu gewährleisten, sondern darüber hinaus auch das Denken und Problemlösen an sich nachhaltig zu verändern, antworte ich stets (in alphabetischer Reihenfolge): APL, Forth und Perl.

Jede dieser Sprachen verfügt über Eigenschaften, die sie einzigartig machen und aus der Masse der herkömmlichen Programmiersprachen herausheben. Über Forth kann ich an dieser Stelle sicherlich nichts sagen, was nicht allen Lesern der „Vierten Dimension“ längst bekannt wäre – an erster Stelle der bemerkenswerten Eigenschaften stehen hier für mich die Klarheit und Prägnanz von Forth-Programmen, die Möglichkeit, die Sprache mit den ihr eigenen Mitteln nahezu unbegrenzt erweitern zu können, und die wunderbare Interaktivität, die wie in kaum einem anderen Umfeld zum Experimentieren einlädt.

So sehr ich diese Eigenschaften schätze und liebe, so sehr vermisse ich jedoch oft die Möglichkeit, auf abstrakterer Ebene mit Forth zu arbeiten – HPs „Reverse Polish LISP“, kurz „RPL“, kommt mir in dieser Hinsicht schon weiter entgegen, indem nahezu beliebige Objekte (nicht im Sinne einer Objektorientierung) auf dem Stack abgelegt und verarbeitet werden können, hat aber den Nachteil, nur auf HP-Taschenrechnern verfügbar zu sein.

An APL schätze ich vor allem die wunderbare Klarheit und Prägnanz seiner Konzepte sowie die Eleganz, mit der Konzepte aus der linearen Algebra und anderen Bereichen der Mathematik direkt auf die Lösung klassischer IT-Fragestellungen angewandt werden können. Als – zumindest in meiner Erfahrung – enormes Problem erweisen sich jedoch fast stets der für APL notwendige arkane Zeichensatz sowie unterschiedliche, mehr oder weniger subtil voneinander abweichende Keymappings, die ein Arbeiten an anderen Rechnern als dem eigenen oft unmöglich machen.

Das zu Beginn an dritter Stelle genannte Perl ist die Sprache, in der ich die Mehrzahl meiner Programme schreibe – nicht zuletzt dank des Perl-typischen „There Is More Than One Way To Do It“-Ansatzes, der meiner Arbeitsweise stark entgegenkommt. Im Laufe der vergangenen Jahre habe ich Perl quasi als Allzweckvehikel schätzen und lieben gelernt, so dass die Idee nahelag, auf der Basis von Perl einen Interpreter für eine kleine Programmiersprache zu entwickeln, welche die Grundkonzepte von Forth und APL in sich vereinigt.

Diese Sprache, kurz `5` genannt, soll im Folgenden kurz dargestellt werden. Wichtig ist an dieser Stelle die Anmerkung, dass der gegenwärtig existierende `5`-Interpreter zwar voll funktionsfähig ist, aber noch an etlichen Stellen weiterentwickelt werden muss (unter anderem fehlt noch eine Vielzahl typischer APL-Operatoren und Funktionen). So gesehen versteht sich der vorliegende Artikel sowohl als ein „Request for Comments“ seitens der Leserschaft als auch als Bitte um Mithilfe bei der weiteren Entwicklung von `5`.

`5` im Überblick

Zentrales Element von `5` ist, wie nicht anders zu erwarten, ein Stack, der in der Lage ist, beliebige Objekte aufzunehmen, solange diese als Skalare oder als beliebig tief verschachtelte Arrays darstellbar sind. Hierunter fallen auch Operatoren, d.h., es ist nicht nur möglich, sondern für einige Funktionen auch notwendig, Operatoren wie `+`, `-`, `*` oder `/` auf dem Stack abzulegen[1].

Im Hinblick auf elementare Operatoren verhält sich `5` (im Großen und Ganzen) wie ein traditionelles Forth:

```
5> : square dup * ;
5> 5 square .
25
5>
```

Die eigentlich interessanten Eigenschaften von `5` treten jedoch erst zu Tage, wenn mit verschachtelten Strukturen (`5` wendet skalare Operatoren wie beispielsweise `+` elementweise auf alle Elemente zweier strukturgleicher Vektoren an, so dass sich einfache Vektoroperationen syntaktisch nicht von herkömmlichen skalaren Operationen unterscheiden) und APL-typischen Operatoren gearbeitet wird. Sollen zum Beispiel zwei dreielementige Vektoren addiert werden, kann dies wie folgt geschehen: `[1 2 3] [4 5 6] +` liefert als Resultat den Vektor `[5 7 9]`. Entsprechend liefert `[1 2 3 4] 1 +` als Resultat den Vektor `[2 3 4 5]`.

Beispiele

Im Folgenden werden die wesentlichen Grundideen von `5` anhand einiger kleiner Beispiele demonstriert.

[1] Einen Returnstack kennt `5` nicht.

```
1   # Simuliere n Wuerfe mit einem 6-seitigen Wuerfel und bestimme das
2   # arithmetische Mittel hieraus. n wird auf dem TOS erwartet.
3   : wuerfel_mittel
4     dup      # Erzeuge eine Kopie von n fuer die Division am Ende.
5     iota     # Erzeuge einen Vektor (0 .. n - 1).
6     defined  # Erzeuge einen Vektor, der eine 1 an jeder Stelle enthaelt,
7              # an der obiger Vektor ein definiertes Element besitzt. Dies
8              # gilt fuer alle Vektorelemente, so dass das Ergebnis ein
9              # Vektor der Form (1, 1, 1, ... 1) ist.
10    6 *      # Multipliziere den Vektor elementweise mit 6 -> (6, 6, ..., 6).
11    ?        # Erzeuge hieraus einen Vektor mit n Pseudozufallszahlen
12             # zwischen 0 und 6 (ausschliesslich).
13    int 1 +  # Runde ab und addiere 1, so dass nun alle Elemente des
14             # Vektors zwischen 1 und 6 liegen.
15    '+       # Lege den Operator + auf den Stack und
16    reduce   # wende die reduce-Funktion an, um den Vektor elementweise
17             # aufzusummieren.
18    swap /   # Dividiere das Resultat durch n.
19  ;
20
21  100 wuerfel_mittel .
```

Listing 1: Ein Zufallszahlengenerator in 5

Arithmetisches Mittel von Pseudozufallszahlen

Es ist das arithmetische Mittel von 100 simulierten Würfen mit einem sechsseitigen Würfel zu bestimmen. Die Grundidee zur Lösung dieses Problemes in 5 ist Folgende:

1. Erzeuge einen Vektor mit 100 Elementen, die jeweils den Wert 6 besitzen.
2. Wende auf diesen Vektor den Operator ? an. Dieser entspricht dem APL-Operator dice, der eine Pseudozufallszahl zwischen 0 und einem (auf dem TOS) vorgegebenen Wert (ausschließlich) generiert.
3. Da 5 skalare Operatoren bei Vorliegen eines Arrays elementweise auf alle Elemente eines Arrays anwendet, erzeugt dies einen ebenfalls 100-elementigen Vektor, der nun allerdings 100 Pseudozufallszahlen im Bereich zwischen 0 und 6 (ausschließlich) enthält.
4. Zur Bestimmung der Summe dieser 100 pseudozufälligen Werte kommt in der Folge die APL-typische Operation reduce zum Einsatz, die auf dem Stack einen skalaren Operator, in diesem Falle +, sowie einen Vektor erwartet. Der skalare Operator wird zwischen je zwei benachbarten Elementen eines Vektors zur Anwendung gebracht, was im vorliegenden Fall der Summe über alle Vektorelemente entspricht.
5. Als abschließenden Schritt muss die solchermaßen erzeugte Summe noch durch die ursprüngliche Anzahl von Elementen des Ausgangsvektors dividiert werden, um das gewünschte arithmetische Mittel zu bestimmen.

In 5 sieht dieses Beispielprogramm nun so aus, wie in Listing 1 zu sehen ist.

Stackdump

Mit Hilfe von Arrayoperationen kann in 5 auch einfach ein Wort implementiert werden, das den gesamten Stackinhalt nicht-destruktiv ausgibt. Von zentraler Bedeutung sind hier die beiden Funktionen compress und expand, die im Folgenden kurz erläutert werden:

compress: Fasse n Elemente auf dem Stack in einen einzigen n-elementigen Vektor zusammen. Beispiel: `1 2 3 3 compress` liefert `[1 2 3]` zurück.

expand: Zerlege einen Vektor in seine Elemente und pushe diese auf den Stack. Am Ende der Operation wird die Anzahl der Elemente auf dem Stack abgelegt. `[1 2 3] expand` liefert also `1 2 3 3` zurück, was wieder als Eingabe für compress genutzt werden könnte.

Die Grundideen zur zerstörungsfreien Ausgabe des Stackinhaltes sind nun die folgenden:

1. Erzeuge einen Vektor aus allen Elementen, die sich auf dem Stack befinden und dupliziere diesen, um einen für die Ausgabe der Elemente zu verwenden und mit Hilfe des anderen den Stack nach Abschluss der Operation zu rekonstruieren.
2. Bilde eine Schleife über den entstandenen Vektor und entferne jeweils ein Element, das ausgegeben wird. Die Schleife wird abgebrochen, sobald der Vektor keine Elemente mehr enthält.
3. Rekonstruiere den ursprünglichen Stackinhalt, indem die Kopie des alle Stackelemente enthaltenden Vektors mit expand ausgepackt wird.

Die praktische Umsetzung hat dann die Gestalt in Listing 2 auf der nächsten Seite.

```
1   : .s
2     depth          # Bestimme die Tiefe des Stacks.
3     compress       # Packe alle Stackelemente in einen n-elementigen Vektor.
4     dup            # Dupliziere diesen Vektor.
5     do             # Schleifenbeginn
6       dup length   # Dupliziere den Vektor und bestimme die Anzahl
7                    # seiner Elemente.
8       0 ==         # Ist die Laenge gleich Null?
9       if           # Falls ja,
10        break      # verlasse die Schleife.
11      then
12      0 extract . # Extrahiere das erste Element und gib es aus.
13    loop
14    drop           # Enferne den nun leeren Vektor.
15    expand         # Packe den zu Beginn kopierten Vektor aus und
16    drop           # loesche die Anzahl ausgepackter Elemente.
17  ;
```

Listing 2: Das Wort `.S` in `5`

Primzahlen

Das folgende Beispiel ist ein typisches Beispiel für die Mächtigkeit von APL – nun allerdings in `5` umgesetzt. Die Aufgabe besteht darin, alle Primzahlen zwischen `2` und einer vorgegebenen oberen Grenze zu bestimmen (Effizienz steht hierbei, wie sich zeigen wird, nicht im Vordergrund).

Die zugrunde liegenden Ideen zur Berechnung aller Primzahlen zwischen `2` und einem auf dem TOS gegebenen Wert `n` sind die folgenden:

1. Erzeuge einen Vektor der Form `[2 3 4 ... n]` (von diesem Vektor werden im Folgenden einige Kopien benötigt) mit Hilfe des `iota`-Operators.

2. Erzeuge auf Basis dieses Vektors eine Matrix als äusseres Produkt zweier dieser Vektoren mit Hilfe der Funktion `outer`. Diese Matrix enthält alle Produktkombinationen der in den beiden Vektoren enthaltenen Elemente. Da die Vektoren kein 1-Element enthielten, besteht diese Matrix ausschließlich aus zusammengesetzten Zahlen – in ihr findet sich keine Primzahl[2].

3. Unter Verwendung des ursprünglichen Vektors wird nun elementweise bestimmt, welche Elemente des Vektors in der Matrix auftreten und welche nicht. Die nicht auftretenden Elemente sind die gesuchten Primzahlen. Mit Hilfe der Funktion `in`, welche die Mengenfunktion „Element von“ abbildet, wird aus der Matrix und diesem Vektor ein Auswahlvektor erzeugt, der aus einer Folge von `1`- und `0`-Elementen besteht, die angeben, ob das korrespondierende Vektorelement in der Matrix auftrat oder nicht.

4. Nach elementweisem Invertieren dieses Vektors liegt ein Auswahlvektor vor, der an jeder Stelle, deren Wert im ursprünglichen Vektor einer Primzahl entspricht, eine `1` aufweist. Dieser Steuervektor wird dann verwendet, um aus dem ursprünglichen Vektor nur die primen Elemente zu selektieren und hieraus einen neuen Vektor zu generieren.

Die Implementation dieses Verfahrens hat in `5` folgende Gestalt:

```
: prime_list
  1 - iota 2 +
  dup dup dup
  '* outer
  swap
  in not
  select
;

100 prime_list .
```

Sonstiges

Obwohl sich `5` noch im Prototypenstadium befindet, hat es sich bereits bei der Auswertung von Messdaten und als Demonstrationssprache in einer Vorlesung bewährt. Typische `5`-Programme sind meist ähnlich kurz wie klassische APL-Implementationen, ohne jedoch die Nachteile des speziellen Zeichensatzes sowie der doch recht ungewohnten Auswertung von Ausdrücken von rechts nach links zu übernehmen. Gerade die von Forth entlehnte große Interaktivität vereint mit der Möglichkeit, eigene Wörter zu definieren, macht `5` zu einem mächtigen und interessanten Instrument.

[2] Die naheliegende Vermutung, dass es ausreicht, eine Matrix mit Kantenlänge $\sqrt{n}$ zu betrachten, trifft hier nicht zu – die Kantenlänge muss $\frac{n}{2}$ betragen.

Wie bereits erwähnt, ist der Interpreter in Perl implementiert und damit weitgehend maschinen- und betriebssystemunabhängig (gegenwärtig wird 5 bereits unter Mac OS X, LINUX, OpenVMS und Windows verwendet). Zu seinem Betrieb ist lediglich ein Perl-Interpreter notwendig (es werden keine besonderen Zusatzpakete benötigt).

Das komplette 5-Paket, das neben dem eigentlichen Interpreter auch eine kleine Standard-Library (`stdlib.5`), einige einfache Beispiele sowie eine recht ausführliche Dokumentation (in Englisch) enthält, kann unter `http://lang5.sourceforge.net` in Form eines ZIP-Files (ca. 250 kB) heruntergeladen werden.

Dr. Bernd Ulmann kann unter `ulmann@vaxman.de` erreicht werden und freut sich über jede Rückmeldung sowie (und vor allem) über mögliche Mitstreiter, deren Neugierde auf 5 der vorliegende kurze Artikel geweckt haben mag.

```
[0]    Z←FN DSMIN A;H;L;N;R;S;T;U;Y
[1]    ⍝N-dimensional minimization using Nelder & Mead downhill simplex method
[2]    ⍝ Arguments:
[3]    ⍝     FN   - character vector containing the name of the function to
[4]    ⍝            minimize.  This function must be monadic, accept a matrix
[5]    ⍝            having an x-vector in each row, and return a vector of y
[6]    ⍝            values.
[7]    ⍝   1⊃A    - n+1 by n matrix of initial simplex vertices.  Each row
[8]    ⍝            holds an x-vector.
[9]    ⍝   2⊃A    - stopping criterion; maximum allowable fractional
[10]   ⍝            difference in the y values of the vertices.
[11]   ⍝ Result:
[12]   ⍝      X   - matrix of final simplex vertices, with lowest point first.
[13]   ⍝
[14]   ⍝ Adapted from the algorithm in "Numerical Recipes in C", 2nd ed.
[15]   ⍝ Not clear whether this is a copyright violation...
[16]   ⍝ Original reference is Nelder, J.A., and Mead, R., _Computer Journal_,
[17]   ⍝   vol. 7, pp. 308-313 [1965].
[18]   ⍝
[19]   ⍝ Note: NRC advises calling this function _twice_, using the low point
[20]   ⍝       from the first call as one of the vertices in the second call.
[21]   ⍝
[22]   (X S)←A
[23]   Y←⍎FN,' X'
[24]  L1:                          ⍝ Loop
[25]   (L N H)←(⍋Y)[1,¯1 0+⍴Y]     ⍝   indices of lowest and two highest points
[26]   R←(|-/R)÷.5×+/|R←Y[H,L]     ⍝   fractional range of y values
[27]   →(R>S)/L2                   ⍝   If range is small enough,
[28]   X[L,1;]←X[1,L;]             ⍝     put the low point first
[29]   →0                          ⍝     and quit
[30]  L2:T←DSMIN2 ¯1               ⍝   try a reflection
[31]   ⍝ Note: DSMIN2 alters X[H;] and Y[H] if a lower point is found
[32]   →(~T≤Y[L])/L3               ⍝   If it's below the lowest point,
[33]   T←DSMIN2 2                  ⍝     try expanding the reflection
[34]   →L1                         ⍝   Else,
[35]  L3:→(~T≥Y[N])/L1             ⍝     If worse than next highest,
[36]   U←Y[H]
[37]   T←DSMIN2 .5                 ⍝       do 1-D contraction
[38]   →(~T≥U)/L1                  ⍝       If still not better,
[39]   X←.5×X+(⍴X)⍴X[L;]           ⍝         contract around the lowest point
[40]   Y←⍎FN,' X'                  ⍝         reevaluate all pts
[41]   →L1                         ⍝ Endloop
```

```
[0]    Z←DSMIN2 F;G;V
[1]    ⍝Macro used by DSMIN
[2]    ⍝ Caution--globals:  X←  Y←  H  FN
[3]    G←(1-F)÷1↓⍴X
[4]    V←(G×+⌿X)-X[H;]×G-F ⍝ new point
[5]    V←(1,⍴V)⍴V          ⍝ 1-row matrix
[6]    Z←⍎FN,' V'          ⍝ try it
[7]    →(~Z<Y[H])/0        ⍝ If it's below the highest,
[8]    Y[H]←Z              ⍝   use it
[9]    X[H;]←,V
```

Das Downhill–Simplex–Verfahren nach Nelder und Mead von 1965 in APL
(Quelle `http://www.chilton.com/~jimw/minimiz.html`)

Das Selbstbau-Magazin MAKE

Seit 2005 erscheint in Amerika viermal im Jahr in Zusammenarbeit mit dem O'Reilly–Verlag das Magazin MAKE. Diese Zeitschrift sollte nicht nur, aber auch für Forth–Programmierer interessant sein.

Auf jeweils rund 200 Seiten berichtet das englischsprachige Magazin über die Projekte professioneller Bastler (Mythbusters, u. a.) und stellt einfache und anspruchsvolle Selbstbauprojekte (*Do It Yourself, DIY*) vor. Die Bandbreite geht dabei von einfachen mechanischen (Metall, Holz) bis hin zu elektronischen und Mikroprozessor–gesteuerten Projekten, etwa den Bau kleiner Roboter. Es stellt auch Hacks vor, also die kreative Andersbenutzung von Dingen abseits ihres ursprünglichen Verwendungszwecks. Zudem gibt es regelmäßige Kolumnen, in denen die Autoren Tipps, Tricks, Einführungen, Erklärungen und Beobachtungen mit den Lesern teilen.

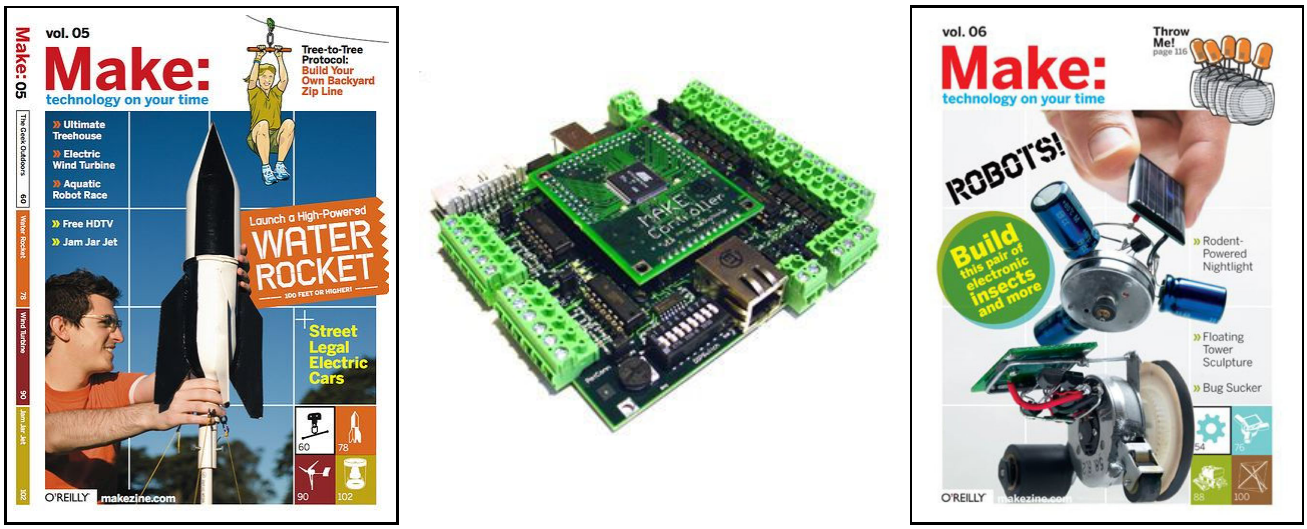
MAKE sieht sich in guter Tradition — es ist sowas wie Steve Ciarcias Circuit Cellar aus dem Byte–Magazin aber mit einem breiteren Spektrum, das nicht nur Elektronik-Projekte behandelt.

Im Februar 2010 ist MAKE 21 erschienen. Wie viele MAKE–Hefte, so widmet sich auch diese Ausgabe einem bestimmten Thema. Hier geht es um *Desktop Manufactoring*, also um das semi–professionelle Fertigen von Produkten im Heimbereich. Normalerweise ist das ja der Industrie vorbehalten. MAKE stellt einen Selbstbau 3D-Scanner vor und verrät, wie man für 800 Dollar eine CNC–Maschine bauen kann. Das vollständige Inhaltsverzeichnis dieser sowie aller bisherigen Ausgaben findet sich auf der MAKE–Homepage unter `http://makezine.com/volumes/`.

MAKE hat auch ein Mikroprozessor–Kit im Angebot. Es ist mit einem Atmel ARM-Prozessor (Atmel AT91SAM7X256), analog und digital IO ausgestattet und wird breit durch Projekte und Entwicklungstools unterstützt.

Das MAKE–Magazin kann für rund 50 Dollar im Jahr (vier Ausgaben) direkt in Amerika (gedruckte und digitale Version) abonniert werden. Einzelexemplare kauft der Interessierte z. B. direkt bei Amazon.de zum Preis von knapp 15 Euro pro Ausgabe.

Insgesamt ist MAKE eine Bereicherung, auch für die, die nur staunen wollen. Das Magazin macht einen liebevoll und professionell hergestellten Eindruck. Es sei jedem, dem Experimentieren und Bastel–Projekte Freude machen, wärmstens empfohlen. (uho)



Zwei Titelseiten des MAKE–Magazins,
in der Mitte das MAKE–Controller–Kit, huckepack auf Anschlussträger mit Klemmanschlüssen

Links

`http://www.MAKEzine.com`	Homepage des MAKE–Magazins
`http://www.circuitcellar.com`	Steve Ciarcias Circuit Cellar
`http://en.wikipedia.org/wiki/MAKE_(magazine)/`	Eintrag über MAKE in der englischsprachigen Wikipedia
`http://www.makezine.com/controller/`	MAKE–Controller–Kit

fHQ9+

Michael Kalus

Die Beschäftigung mit esoterischen Programmiersprachen kann zu tieferem Verständnis seriöser Programmiersprachen sowie zur Verbesserung strukturellen Denkens führen. Praktische Bedeutung haben die esoterischen Programmiersprachen somit in Forschung und Lehre. Klar, dass wir da als Forth–Verein nicht tatenlos zusehen können.

Das Konzept

Auch HQ9+ ist so eine esoterische Programmiersprache, entwickelt von Cliff L. Biffle. HQ9+ ist nicht turingvollständig, da weder bedingte Anweisungen noch Schleifen möglich sind. Er entwickelte mit HQ9+ eine Sprache, mit der in Programmierkursen häufig gestellte Aufgaben trivial gelöst werden können. Das Prinzip der Trivialisierung setzt sich in der objekt–orientierten Variante HQ9++ von David Morgan–Mar fort, bei der Objekte erstellt werden können, die, wie auch der Zähler, keine Funktion erfüllen können.

Hier möchte ich mich demonstrativ an die Implementation des klassischen HQ9+ halten, und zeigen, dass es in Forth simpel ist, seine eigene Programmiersprache zu schaffen. In Abweichung vom Original wird allerdings ein Delimiter (space) benutzt, um die Befehle zu trennen. Da sowieso jede Implementation irgend einer Programmiersprache ihre spezifischen umgebungsbedingten kleinen Abweichungen (environmental dependency) hat, ist dies durchaus legitim. Implementiert worden ist der komplette Befehlssatz: `H Q 9 +`

Beispielcode

Um die Leistungsfähigkeit von HQ9+ zu erkunden, gebe z. B. folgendes ein:

```
H ↩
Q Q Q Q ↩
H Q H Q ↩
9 ↩
```

Viel Vergnügen.

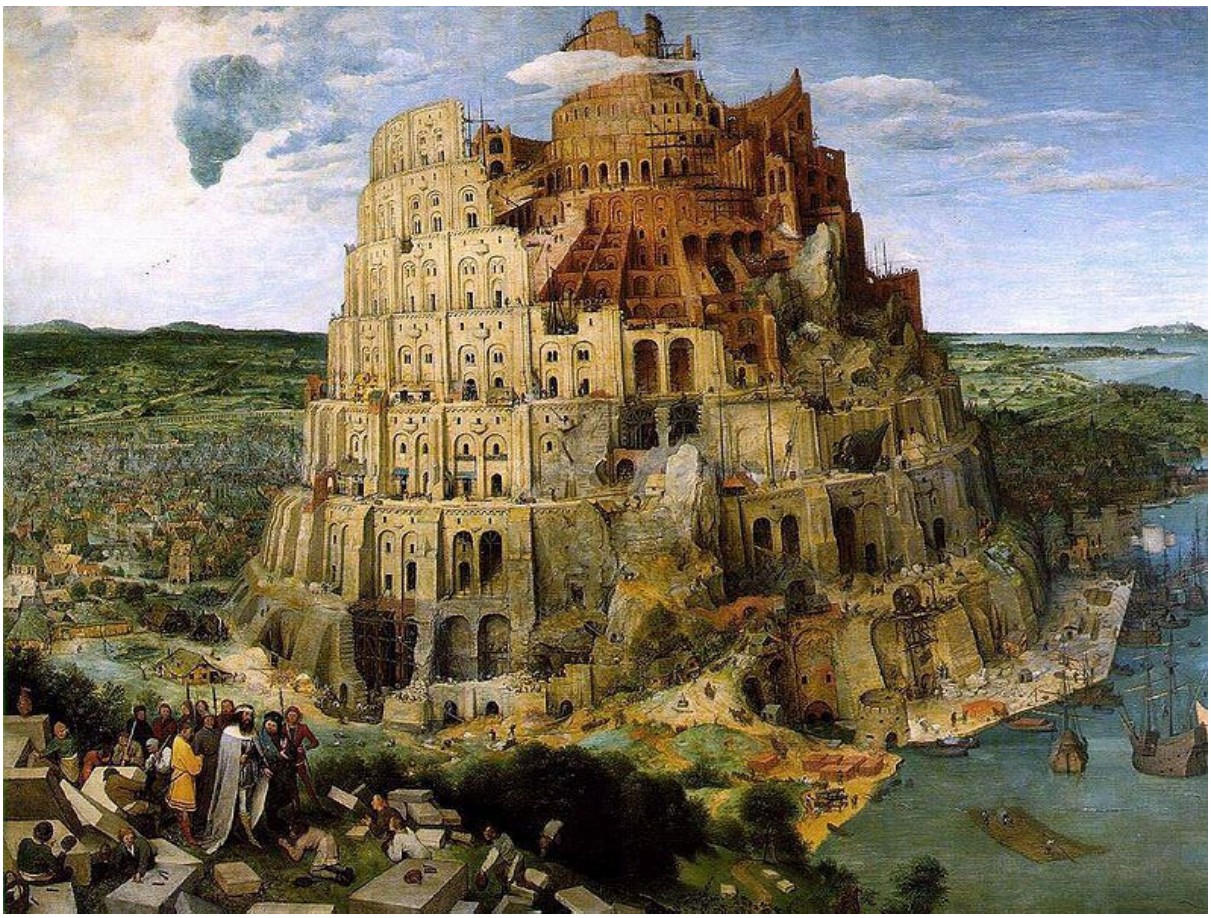
Quellen:

`http://www.cliff.biffle.org/esoterica/hq9plus.html`
`http://de.wikipedia.org/wiki/Esoterische_Programmiersprache`
`http://de.wikipedia.org/wiki/HQ9%2B`
`http://99-bottles-of-beer.net/lyrics.html`
`http://de.wikipedia.org/wiki/The_Complexity_of_Songs` — Über die Komplexität von Liedern, Donald E. Knuth, 1977

Code

```
1   ( HQ9+ in gforth )
2
3   vocabulary hq9+  hq9+ definitions   : .. bye ;
4
5   : H    ." hello World" ;
6
7   : Q    source type ;
8
9     : .xbottles { x -- }  \ x{ 99..3}
10      cr  x . ." bottles of beer on the wall, " x . ." bottles of beer."
11      cr ." Take one down and pass it around, " x 1- . ." bottles of beer on the wall."
12      cr ;
13
14    : .2bottles
15      cr 2 . ." bottle of beer on the wall, " 2 . ." bottle of beer."
16      cr ." Take one down and pass it around, " 1 . ." bottle of beer on the wall."
17      cr ;
18
19    : .1bottle
20      cr 1 . ." bottle of beer on the wall, " 1 . ." bottle of beer."
21      cr ." Take one down and pass it around, no more bottles of beer on the wall."
22      cr ;
23
```

```
24      : .0bottles
25        cr ." No more bottles of beer on the wall, no more bottles of beer."
26        cr ." Go to the store and buy some more, 99 bottles of beer on the wall."
27        cr ;
28
29      : .99bottles
30        99 3 - for i 3 + .xbottles next .2bottles .1bottle .0bottles ;
31
32    : 9    cr cr .99bottles cr cr ;
33
34    0 value accumulator
35    : +     accumulator 1+ to accumulator ;
36
37    words cr cr
38    \ finis
```



Der Turmbau zu Babel von Pieter Bruegel dem Älteren von 1563 (Quelle: Wikipedia)

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

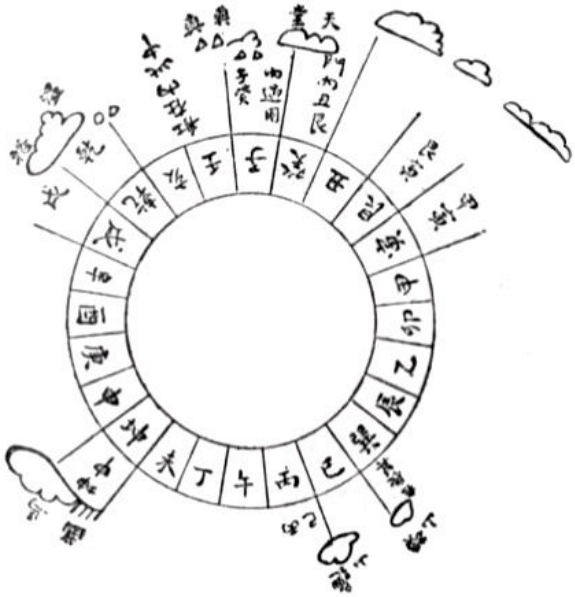
Carsten Strotmann
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

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 2010

vom 26. bis 28. März 2010

im Strand–Hotel Hübner

Seestraße 12, 18119 Rostock-Warnemünde

http://www.hotel-huebner.de

Geplantes Programm

Donnerstag, 25.03.2010
nachmittags Treffen der Frühankommer
Forth–200x–Standard–Treffen

Freitag, 26.03.2010
nachmittags Beginn der Forth–Tagung
Vorträge und Workshops

Samstag, 27.03.2010
vormittags Vorträge und Workshops
nachmittags Exkursion

Sonntag, 28.03.2010
09:00 Uhr Mitgliederversammlung
nachmittags Ende der Tagung

Anreise siehe: http://www.hotel-huebner.de/kontakt/anfahrt.html

Quellen: www.hotel-huebner.de